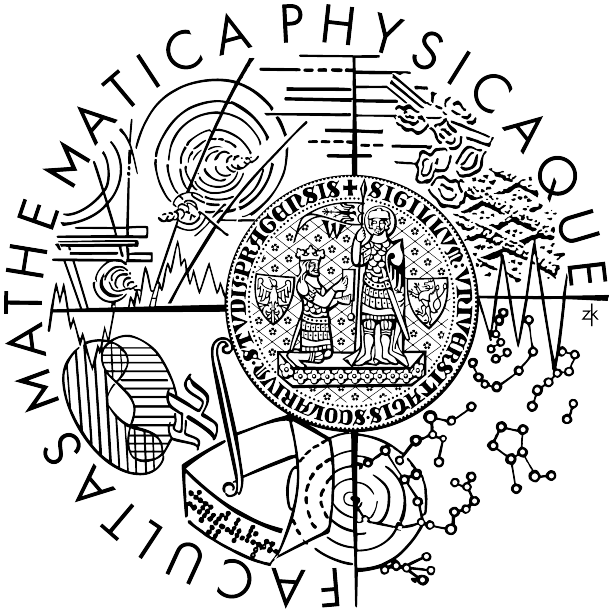
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

# DIPLOMOVÁ PRÁCE




Matúš Ondreička

**Rozšíření Faginova algoritmu pro více uživatelů**

Katedra softwarového inženýrství
Vedoucí diplomové práce: Prof. RNDr. Jaroslav Pokorný, CSc.
Studijní program: Informatika, softwarové systémy

Na tomto mieste by som rád poďakoval vedúcemu diplomovej práce Prof. RNDr. Jaroslavovi Pokornému, CSc. za jeho odborné rady a pripomienky, ktoré zásadne pomohli pri vytváraní tohto textu.

Prohlašuji, že jsem svou diplomovou práci napsal samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 10. dubna 2008 Matúš Ondreička

# Obsah

*Název práce*: Rozšíření Faginova algoritmu pro více uživatelů
*Autor*: Matúš Ondreička
*Katedra (ústav)*: Katedra softwarového inženýrství
*Vedoucí diplomové práce*: Prof. RNDr. Jaroslav Pokorný, CSc.
*e-mail vedoucího*: jaroslav.pokorny@mff.cuni.cz

*Abstrakt:*

Táto práca sa zaoberá problematikou vyhľadávania K najlepších objektov pre viacerých užívateľov. Každý užívateľ pritom preferuje objekty inak. Užívateľské preferencie sú modelované lokálne pomocou fuzzy funkcií a globálne pomocou agregačnej funkcie. Práca sa zaoberá Faginovými algoritmami, ktoré však vyhľadávajú K najlepších objektov iba vzhľadom na agregačnú funkciu. Kvôli použitiu lokálnych preferencií pri výpočte Faginovho algoritmu vznikla potreba vytvoriť nový model zoznamov, nad ktorými prebieha výpočet Faginovho algoritmu. Nový model zoznamov je založený na $B^+$-stromoch, v ktorých sú indexované objekty nezávisle na preferenciách užívateľov. Práca sa ďalej zaoberá použitím viacrozmerého B-stromu na vyhľadávanie K najlepších objektov podľa užívateľských preferencií vzhľadom ku Faginovým algoritmom. Práca prináša nové riešenie založené na viacrozmernom B-strome, VB-algoritmus.V závere práce sú uvedené výsledky testov všetkých popisovaných algoritmov.

*Kľúčové slová*:

užívateľské preferencie, top-k problém, Faginov algoritmus, $B^+$-strom, VB-strom.

*Title*: Extending Fagin's algorithm for more users
*Author*: Matúš Ondreička
*Department*: Department of software engineering
*Supervisor*: Prof. RNDr. Jaroslav Pokorný, CSc.
*Supervisor's e-mail address*: jaroslav.pokorny@mff.cuni.cz

*Abstract*:

We discuss the issue of searching the best K objects in more attributes for more users. Every user prefers objects in different ways. User preferences are modelled locally with a fuzzy function and globally with an aggregation function. Also, we discuss the issue of searching the best K objects without accessing all objects. We deal with the use of local preferences when computing Fagin's algorithms. We created a new model of lists for Fagin's algorithms based on $B^+$-trees. Furthermore, we deal with the use of a multidimensional B-tree for searching the best K objects. We developed an MD-algorithm, which can effectively find the best K objects in a multidimensional B-tree in accordance with user's preferences and without accessing all the objects. Last but not least, we show results of all the tests of described algorithms. MD-algorithm achieves better results in the number of accessed objects than Fagin's algorithms.

*Keywords*:

user preferences, top-k problem, Fagin's algorithm, $B^+$-tree, multidimensional B-tree.

# 1 Úvod

V dnešnej dobe existuje v rôznych databázach veľké množstvo dát o rôznych objektoch s rôznymi vlastnosťami. Užívatelia sa podľa týchto dát snažia vyhľadávať objekty, ktoré potrebujú. Každý z objektov má určité vlastnosti, množinu atribútov. Užívatelia sa potom podľa hodnôt týchto atribútov rozhodujú, ktorý z objektov je pre nich viac alebo menej vhodný a vyhľadávajú tak najvhodnejšie objekty [10].

Dáta o objektoch rovnakého druhu sú najčastejšie uložené v relačných databázach, v ktorých sa v súčasnosti bežne uchovávajú státisíce objektov. Aby užívateľ našiel niekoľko požadovaných objektov, musí zadať určité požiadavky, ktoré zo všetkých objektov vyberú určitú podmnožinu objektov. Táto podmnožina môže byť prázdna alebo naopak, užívateľovi sa zobrazí veľké množstvo vyhovujúcich objektov. V lepšom prípade sa pre vhodne zadané požiadavky užívateľovi zobrazí rozumný počet objektov, z ktorých už musí sám vybrať najvhodnejšie objekty.

Táto práca sa zaoberá problematikou priameho vyhľadania $K$ najlepších objektov pre užívateľa. Ktorý objekt je pre užívateľa lepší a ktorý zase horší určujú jeho užívateľské preferencie (viz kapitola 2). Vo všeobecnosti každý užívateľ preferuje objekty inak. Pred vyhľadaním $K$ najlepších objektov užívateľ nastaví svoje užívateľské preferencie, ktoré určujú vhodnosť objektu v závislosti na jeho hodnotách atribútov. Následne sa podľa nastavených preferencií užívateľa vyhľadá $K$ najlepších objektov. V prípade $K = 1$ sa vyhľadá najlepší objekt pre užívateľa.

Problém vyhľadania $K$ najlepších objektov súčasne podľa hodnôt viacerých atribútov sa označuje ako *top-k* problém (viz kapitola 3). Pri vyhľadávaní $K = 10$ najlepších objektov je neefektívne prechádzať státisíce objektov, preto sa práca zaoberá vyhľadávaním $K$ najlepších objektov bez prejdenia všetkých objektov. Práca sa zmeriava na použitie tzv. Faginovho algoritmu [1] a jeho rôznych modifikácií (viz kapitola 4), pretože čiastočne podporuje použitie užívateľských preferencií a je jedným z najefektívnejších algoritmov, ktoré riešia top-k problém.

V tejto práci sa predpokladá, že objekty sú uložené v jednej spoločnej dátovej štruktúre pre všetkých užívateľov, v ktorej je možné podľa nastavených preferencií konkrétneho užívateľa nájsť $K$ najlepších objektov. Uloženie objektov v klasickej relačnej databáze už nepostačuje, pretože objekty musia byť špeciálne indexované tak, aby bolo možné efektívne využiť výhody Faginovho algoritmu a zároveň používať zavedený model užívateľských preferencií (viz kapitola 2).

Preto sa v tejto práci na uloženie dát o objektoch používajú $B^+$-stromy (viz kapitola 5). Použitím $B^+$-stromov sa efektívne odstránia všetky problémy spojené s použitím užívateľských preferencií, vzhľadom na Faginov algoritmus. Pôvodná verzia Faginovho algoritmu používa ako vstup zostupne zotriedené zoznamy (viz kapitola 4), namiesto ktorých sa použijú $B^+$-stromy (viz kapitola 6).

V klasických $B^+$-stromoch je možné indexovať objekty iba podľa jedného atribútu. Perspektívne riešenie ponúka VB-strom. Vo VB-strome sa dajú objekty indexovať aj podľa viacerých atribútov súčasne. Táto práca sa ďalej zaoberá použiteľnosťou VB-stromu na riešenie top-k problému vo vzťahu k Faginovmu algoritmu. Vo VB-strome nie je možné priamo použiť Faginov algoritmus. Práca preto prináša nové top-k riešenie založené na VB-stromoch, VB algoritmus (viz kapitola 7).

Cieľom práce je implementovať popísané riešenia problému top-k a skúmať ich výhody, nevýhody, a porovnať vzájomný vzťah týchto riešení. Implementácia je vytvorená v programovacom jazyku Java. V práci sú implementované dve verzie Faginovho algoritmu a VB algoritmus. Ďalej práca obsahuje implementáciu stromových dátových štruktúr, ktoré tieto algoritmy používajú počas výpočtu, a ďalších potrebných metód pre testovanie popisovaných top-k algoritmov (viz kapitola 8).

Ďalej sa práca zaoberá testovaním top-k algoritmov (viz kapitola 9). Algoritmy sú testované napríklad nad rôznymi množinami objektov, pre rôzny počet vyhľadávaných objektov atď. Predmetom testovania je čas výpočtu top-k algoritmov a hlavne počet prehľadaných objektov top-k algoritmami počas hľadania *K* najlepších objektov.

V kapitole 10 sú uvedené jej prínosy, zhrnutie dosiahnutých výsledkov a motivácia pre ďalšiu prácu. Kapitola 11 obsahuje použitú literatúru.

# 2 Modelovanie užívateľských preferencií

Táto práca sa zaoberá vyhľadávaním $K$ najlepších objektov z danej množiny objektov $X$, ktoré majú $m$ atribútov $A_1, \ldots, A_m$. Na to, aby bolo možné nájsť $K$ najlepších objektov z množiny objektov $X$, musí byť možné pre dva objekty posúdiť, ktorý z nich je lepší a ktorý je horší. Je potrebné mať k dispozícii vhodnú *reláciu usporiadania*, pomocou ktorej je možné rozhodnúť, ktorý objekt je najlepší, ktorý je druhý najlepší až $K$-ty najlepší. Takto je možné nájsť množinu $K$ najlepších objektov z množiny $X$.

## 2.1 Ohodnotenie objektu

Jednou z možností, ako dosiahnuť usporiadanie objektov od najlepšieho po najhorší, je zaviesť pre objekty množiny $X$ mieru vhodnosti $F : X \rightarrow R$, kde $R$ je množina, na ktorej je definovaná relácia usporiadania. Miera vhodnosti objektu $x \in X$ (ohodnotenie objektu $x$) bude reprezentovaná ako zobrazenie $F(x) : x \rightarrow R$ a bude sa označovať ako *ohodnocovacia funkcia*. Pre potreby modelovania užívateľských preferencií sa ako množina $R$ najčastejšie používa interval reálnych čísel $R = [0, 1]$. Pre najhorší objekt $x \in X$ platí $F(x) = 0$ a pre najlepší objekt platí $F(x) = 1$. Pre dva rôzne objekty $x_1, x_2 \in X$ platí $F(x_1) > F(x_2)$, keď $x_1$ je lepší ako $x_2$, a platí $F(x_1) = F(x_2)$, keď $x_1, x_2$ sú rovnako vhodné.

Pre účely tejto práce je navyše potrebné, aby bola miera vhodnosti $F$ objektov užívateľsky závislá, aby zobrazenie $F(x)$ vyjadrovalo preferencie konkrétneho užívateľa. Na jednej množine objektov $X$ preferujú rôzni užívatelia rôzne objekty. Pre dvoch rôznych užívateľov môže množina $K$ najlepších objektov obsahovať iné objekty z množiny $X$.

Pri modelovaní užívateľských preferencií sa rozlišujú dva typy preferencií:

- *Lokálne preferencie*
  - vyjadrujú ako užívateľ preferuje objekty v rámci jednotlivých atribútov.
- *Globálne preferencie*
  - vyjadrujú ako užívateľ preferuje objekty vzhľadom na všetky atribúty.

## 2.2 Lokálne preferencie

Každý objekt $x \in X$ má definovaných $m$ hodnôt týchto atribútov. Lokálna preferencia vyjadruje ako sa preferujú objekty len podľa $i$-tého atribútu $A_i$, $i = 1, \ldots, m$. V tejto podkapitole sa preto uvažuje len ohonocovacia funkcia $f_i(x) : x \rightarrow [0, 1]$, ktorá priraďuje objektom $x$ ich lokálne ohodnotenia iba v závislosti na $i$-tom atribúte $A_i$.

### 2.2.1 Fuzzy funkcie

V klasickej dvojhodnotovej logike by bolo možné reprezentovať, len či je objekt $x \in X$ pre užívateľa vhodný (ohodnotenie 1), alebo či je preňho nevhodný (ohodnotenie 0). Pre ohonocovaciu funkciu by platilo $f_i(x) : x \rightarrow \{0, 1\}$, kde $\{0, 1\}$ je dvojprvková množina pravdivostných hodnôt .

Pre potreby modelovania užívateľských preferencií je oveľa vhodnejšie zaviesť viachodnotovú logiku, *fuzzy logiku*. Vo fuzzy logike môže byť množinou pravdivostných hodnôt ľubovoľná usporiadaná množina. Najčastejšie sa používa interval reálnych čísel [0, 1].

Ako $a_i^x$ sa bude označovať hodnota *i*-tého atribútu $A_i$ objektu $x \in X$. Ako $adom(A_i)$ sa bude označovať aktuálna doména *i*-tého atribútu $A_i$, čiže množina všetkých hodnôt *i*-tého atribútu $a_i^x$ všetkých objektov $x \in X$. Keď $adom(A_i)$ obsahuje číselné hodnoty, potom pre *i*-tý atribút existuje interval $R_i$ reálnych čísel, ktorý obsahuje všetky hodnoty z $adom(A_i)$. V prípade, že sa za $R_i$ vyberie najmenší možný interval, platí $R_i = [a_i^{min}, a_i^{max}]$, kde $a_i^{min}$ je najmenší prvok a $a_i^{max}$ je najväčší prvok z $adom(A_i)$. Takýto interval $R_i$ sa bude v tejto práci označovať ako doména $dom(A_i)$ *i*-tého atribútu $A_i$.

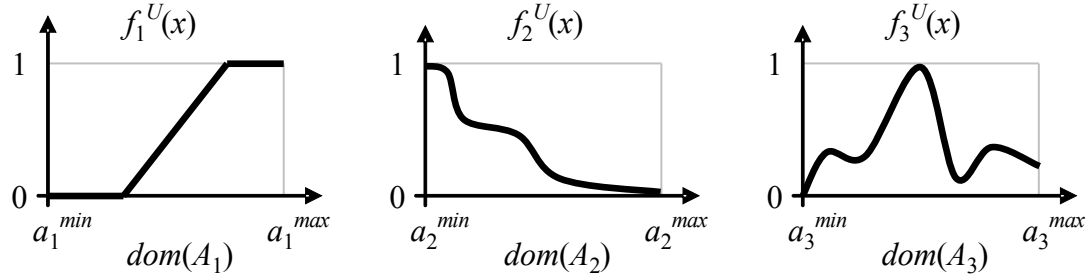
Ak je atribút $A_i$ číselného typu, potom je možné pre tento atribút zaviesť *fuzzy funkciu*. V tejto práci sa *fuzzy funkciou* bude rozumieť zobrazenie

$$f_i : R_i \rightarrow [0, 1],$$

ktoré je definované na intervale $R_i$ a doménu *i*-tého atribútu $dom(A_i)$ zobrazuje na interval [0, 1]. Lokálne preferencie užívateľa *U* pre atribúty $A_1, \ldots, A_m$ vyjadrujú *užívateľské fuzzy funkcie*, ktoré sa budú označovať ako $f_1^U(x), \ldots, f_m^U(x)$. Užívateľskou fuzzy funkciou $f_i^U(x)$ pre *i*-tý atribút sa bude rozumieť zobrazenie

$$f_i^U(x) : a_i^x \rightarrow [0, 1], \quad \text{kde } i = 1, \ldots, m,$$

ktoré objekty $x \in X$ zobrazuje podľa hodnôt ich *i*-tého atribútu $a_i^x$ na interval [0, 1]. Užívateľské fuzzy funkcie užívateľa *U* takto ohodnocujú objekty podľa *i*-tého atribútu. Porovnaním $f_i^U(x_1)$ a $f_i^U(x_2)$ dvoch objektov $x_1$ a $x_2$ je možné rozhodnúť, ktorý z nich je podľa hodnoty *i*-tého atribútu pre užívateľa *U* vhodnejší.




*Obrázok 2.2.1, Užívateľské fuzzy funkcie*

Na obrázku 2.2.1 sú znázornené fuzzy funkcie užívateľa *U* pre množinu objektov s tromi atribútmi $A_1$, $A_2$, $A_3$. Na každom z grafov x-ová os reprezentuje doménu atribútu $dom(A_i)$, y-ová os reprezentuje hodnotu $f_i^U(x)$ pre všetky hodnoty $a_i^x$ *i*-tého atribútu všetkých objektov $x \in X$. Užívateľské fuzzy funkcie môžu mať rôzny tvar. Podľa prvého atribútu $A_1$ užívateľ *U* preferuje objekty *x* s vyššou hodnotou $a_1^x$, pre druhý atribút preferuje objekty s vyššou hodnotou. Užívateľ môže preferovať aj objekty s nejakou optimálnou hodnotou atribútu. Napr. pre tretí atribút $A_3$ preferuje užívateľ *U* objekty $x \in X$, ktoré majú hodnoty tretieho atribútu $a_3^x$ približne v strede domény $dom(A_3)$, tam kde sa nachádza „najvyššie položený úsek" fuzzy funkcie $f_3^U(x)$.

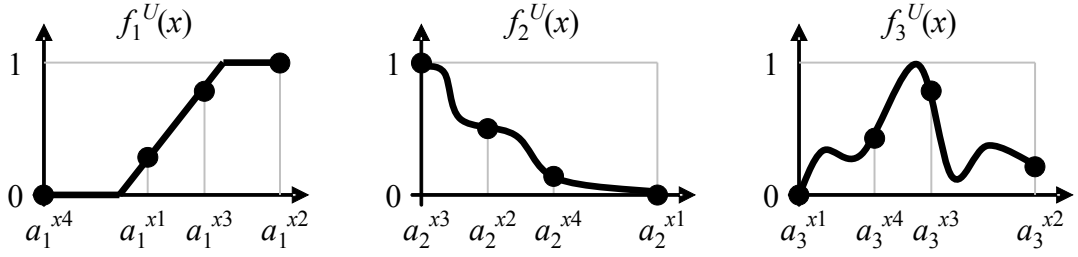
### 2.2.2 Interpretácia

Užívateľskú fuzzy funkciu je možné interpretovať dvoma spôsobmi:

- *hodnotová interpretácia*
- *usporiadavacia interpretácia*

Ak záleží len na ohodnotení objektu $x \in X$ podľa užívateľskej fuzzy funkcie $f_i^U(x)$, použije sa *hodnotová interpretácia* fuzzy funkcie. Podľa hodnoty $i$-tého atribútu $a_i^x$ objektu $x$ je možné priamo pomocou užívateľskej fuzzy funkcie vyčísliť lokálne hodnotenie objektu. Ak sa pre $i$-tý atribút $A_i$ vyčíslia hodnoty $f_i^U(x)$ všetkých objektov $x \in X$, následne je možné zotriediť množinu objektov $X$ podľa $i$-tého atribútu $A_i$ od najlepšieho objektu po najhorší objekt pre užívateľa $U$.

Pri *usporiadavacej interpretácii* fuzzy funkcie je možné užívateľskú fuzzy funkciu chápať ako nové usporiadanie domény $i$-tého atribútu $A_i$. Na konkrétnych hodnotách atribútov $a_i^x$ už nezáleží, na zotriedenie objektov $X$ podľa $i$-tého atribútu $A_i$ sa použije tvar fuzzy funkcie $f_i^U(x)$.




*Obrázok 2.2.2, Usporiadavacia interpretácia fuzzy funkcie*

Na obrázku 2.2.2 je znázornená situácia, ktorá znázorňuje usporiadavaciu interpretáciu fuzzy funkcie. Nech množina $X$ obsahuje štyri objekty $X = \{x_1, x_2, x_3, x_4\}$, ktoré majú tri atribúty $A_1$, $A_2$, $A_3$. Na x-ovej osi grafov $f_1^U(x)$, $f_2^U(x)$, $f_3^U(x)$ sa nachádzajú hodnoty atribútov objektov množiny $X$. Pre všetky objekty množiny X sa dokreslia na graf príslušnej fuzzy funkcie $f_i^U(x)$ body, ktoré znázorňujú objekty. Tieto body budú ležať na grafe $i$-tej fuzzy funkcie presne nad hodnotu atribútu objektu $x$, na súradniciach $\{a_i^x, f_i^U(x)\}$, ako je znázornené na obrázku.

Nech $p$ je priamka rovnobežná s x-ovou osou na grafe atribútu $A_i$. Priamka $p$ sa na začiatku položí tak, aby prechádzala bodom 1 na y-ovej osi grafu. Priamka $p$ sa potom pohybuje smerom k osi x, až nakoniec na osi x skončí. Pri tomto pohybe priamka pretne všetky body, ktoré znázorňujú na grafe objekty. Navyše priamka $p$ pretína tieto body presne v takom poradí, ako keď sa zotriedia body $x$ zostupne podľa $i$-tého atribútu $A_i$ podľa hodnôt $f_i^U(x)$.

Pre užívateľskú fuzzy funkciu $f_1^U(x)$ z obrázku vzniká poradie objektov $x_2$, $x_3$, $x_1$, $x_4$, pre $f_2^U(x)$ poradie $x_3$, $x_2$, $x_4$, $x_1$, pre $f_3^U(x)$ poradie $x_3$, $x_4$, $x_2$, $x_1$.

Podľa tvaru užívateľskej fuzzy funkcie $f_i^U(x)$ je teda možné usporiadať objekty podľa užívateľskej preferencie, bez vyčíslenia jej hodnoty.

### 2.2.3 Normalizácia

Z hľadiska všestrannosti použitia fuzzy funkcií je vhodné zaviesť jednotný model užívateľskej fuzzy funkcie. Pre každý atribút $A_i$ je vhodné normalizovať jeho doménu $dom(A_i)$ na reálny interval [0, 1]. Potom sa problém rôznorodosti jednotlivých atribútov stáva problémom normalizácie a v tejto práci sa ďalej môže používať jednotný model užívateľskej fuzzy funkcie

$$f_i^U : [0, 1] \rightarrow [0, 1].$$

Preto sa v tejto práci sa pre každý atribút $A_i$ zavedie *normalizácia* ako zobrazenie

$$N_i : dom(A_i) \rightarrow [0, 1], \quad \text{kde } i = 1, \ldots, m,$$

ktoré doménu $i$-tého atribútu zobrazuje na interval [0, 1]. Podľa rozloženia hodnôt v doméne atribútu a dátového typu atribútu existujú dva prípady normalizácie.

#### Spojité atribúty

Prvým prípadom je normalizácia spojitých atribútov, ktoré sa dajú charakterizovať ako interval reálnych čísel. Pre množinu objektov $X$ s atribútom $A_i$ s jeho doménou $dom(A_i)$ existuje interval $R_i$ reálnych čísel, ktorý obsahuje všetky hodnoty z $dom(A_i)$ a jeho hranice tvoria najmenší a najväčší prvok z $dom(A_i)$, $R_i = [a_i^{min}, a_i^{max}]$.

Ako normalizácia $N$ môže byť v tomto prípade použitá lineárna fukcia ( f(x) = ax + b ). Potom platí pre okrajové body intervalu $N(a_i^{min}) = 0$, $N(a_i^{max}) = 1$ a vnútorné body $a_i^x$ intervalu $R_i$ sa zobrazia ako vzdialenosť od počiatočného bodu intervalu vydelená dĺžkou intervalu. Pre normalizáciu $N$ realizovanú pomocou lineárnej funkcie platí

$$N(a_i^x) = \frac{a_i^x - a_i^{min}}{a_i^{max} - a_i^{min}} \quad .$$

*Príklad*: Nech existuje množina bytov s atribútom *Rozloha*. Doménu tohto atribútu je možné vyjadriť ako reálny interval od rozlohy bytu s najmenšou rozlohou 17 m$^2$ po rozlohu najväčšieho bytu 305 m$^2$, teda [17, 305]. Potom rozloha 17 m$^2$ bude po normalizácii zobrazená ako $N(17) = 0$, rozloha 305 m$^2$ ako $N(305) = 1$. Vnútorné hodnoty rozlohy $r$ z intervalu [17, 305] sa potom zobrazia $N(r) = (r - 17) / (305 - 17)$.

#### Diskrétne atribúty

Druhým prípadom je normalizácia diskrétnych atribútov, ktoré sa nedajú tak ako v predchádzajúcom prípade charakterizovať reálnym intervalom. Doména atribútu sa dá vnímať len ako $p$ prvková množina, $S = \{s_1, \ldots, s_p\}$. Keďže fuzzy funkciu je možné zaviesť len pre atribút číselného typu, je nutné aj takúto doménu atribútu normalizovať na interval [0, 1]. To sa dá realizovať tak, že sa množina S vhodným spôsobom zotriedi (napr. lexikograficky). Vznikne zotriedená množina $S' = \{s_1', \ldots, s_p'\}$. Normalizácia $N$ bude v tomto prípade vyzerať tak, že pre prvý prvok $s_1'$ a posledný prvok $s_p'$ množiny $S'$ platí $N(s_1') = 0$, $N(s_q') = 1$. Pre prvky množiny $S'$ je výsledok normalizácie $N$ :

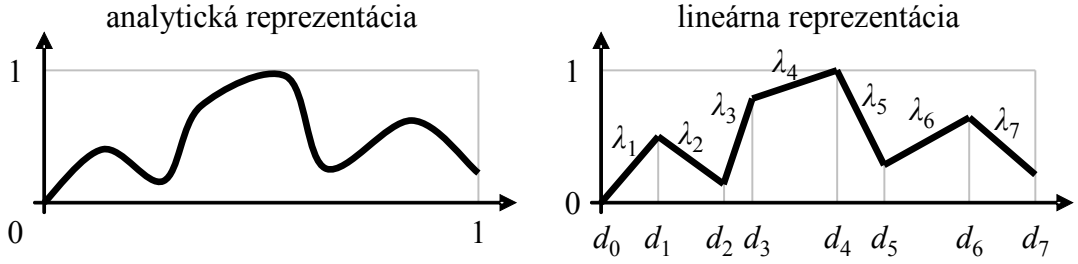
$$N(s_i') = \frac{i - 1}{p - 1} \quad .$$

*Príklad*: Nech existuje množina bytov s atribútom *Mesto*. Vznikne zotriedená množina $S'$ = {Brno, Kladno, Kolín, Plzeň, Praha}. Potom platí $N$(Brno) = 0, $N$(Kladno) = 0.25, $N$(Kolín) = 0.5, $N$(Plzeň) = 0.75, $N$(Praha) = 1.

### 2.2.4 Lineárna reprezentácia

Fuzzy funkciou môže byť ľubovoľná spojitá funkcia $f : [0, 1] \rightarrow [0, 1]$ definovaná na intervale [0, 1]. Môže to byť funkcia $f(x) = x^2$ alebo $f(x) = ½ \cos((2x + 1)\pi) + ½$. Takáto analytická reprezentácia fuzzy funkcií je však implementačne zbytočne náročná a pre bežného užívateľa by bolo nepraktické popisovať svoje preferencie pomocou analytickej reprezentácie funkcií.

Vhodnejším a implementačne menej náročným riešením je fuzzy funkciu reprezentovať ako *zjednotenie lineárnych funkcií* $\lambda_1, ..., \lambda_n$. Nech existuje pre číslo $n$ postupnosť reálnych čísel $d_0, ..., d_n$ taká, že $d_0 = 0$, $d_n = 1$, a platí $d_i < d_{i+1}$, potom $i$-tá funkcia $\lambda_i$, z $n$ lineárnych funkcií $\lambda_1, ..., \lambda_n$ je definovaná na intervale $[d_{i-1}, d_i]$.




*Obrázok 2.2.4, Lineárna reprezentácia fuzzy funkcie*

Na obrázku 2.2.4 je znázornená analytická a lineárna reprezentácia rovnakej fuzzy funkcie. Lineárna reprezentácia vznikla ako zjednotenie $n = 7$ lineárnych funkcií. Pre väčšie $n$ by sa lineárna reprezentácia ešte viac podobala analytickej a pre dostatočne veľké $n$ by bol rozdiel týchto dvoch reprezentácií zanedbateľný.

## 2.3 Globálne preferencie

Množina $X$ obsahuje konečnú množinu objektov ktoré majú m atribútov $A_1, \ldots, A_m$. Každý objekt $x \in X$ má definovaných $m$ hodnôt týchto atribútov. Globálne preferencie vyjadrujú ako užívateľ preferuje objekty vzhľadom na všetky atribúty. V tejto podkapitole sa uvažuje zobrazenie $F(x) : x \rightarrow [0, 1]$ a označovaná ako ohodnocovacia funkcia, ktorá priraďuje objektom globálne ohodnotenie.

### 2.3.1 Agregačná funkcia

V predošlom texte bol zavedený model, ktorý umožňuje podľa užívateľských preferencií, pomocou fuzzy funkcií, lokálne ohodnotiť objekty podľa jedného atribútu. Pri hodnotení objektu podľa viacerých atribútov, sa bude v tejto práci vychádzať z tohto modelu a pomocou užívateľskej agregačnej funkcie $@^U$ sa budú hodnotenia jednotlivých atribútov objektu agregovať do jediného globálne ohodnotenia $F(x)$ objektu $x$.

Na množine $X$ objektov s $m$ atribútmi sa označuje ako agregačná funkcia $@$ funkcia s $m$ premenným $p_1, \ldots, p_m \in [0, 1]$, ktorej obor hodnôt je reálny interval [0, 1].

$$@ : [0, 1]^m \rightarrow [0, 1].$$

Pre užívateľa $U$ s jeho užívateľskými fuzzy funkciami $f_1^U$, …, $f_m^U$ sa označí ako užívateľská ohodnocovacia funkcia $@^U$ agregačná funkcia, ktorá vznikne substitúciou $p_i = f_i^U(x)$, kde $i = 1, \ldots, m$. Potom pre každý $x \in X$ platí

$$@^U(x) = @(p_1, \ldots, p_m) = @(f_1^U(x), \ldots, f_m^U(x)).$$

Pomocou užívateľskej ohodnocovacej funkcie $@^U(x)$ je možné pre všetky objekty $x \in X$ vyjadriť „mieru vhodnosti'' objektu pre užívateľa $U$. Podobne ako pri fuzzy funkciách platí $@^U(x) = 1$ pre najlepší objekt a $@^U(x) = 0$ pre najhorší objekt. Porovnaním $@^U(x_1)$ a $@^U(x_2)$ dvoch objektov $x_1$ a $x_2$ je možné rozhodnúť, ktorý z objektov je pre užívateľa $U$ vhodnejší.

## 2.3.2 Typy agregačnej funkcie

Výhodou agregačnej funkcie je, že užívateľ môže určiť *vzájomný vzťah atribútov*, ktorý atribút je pre neho viac alebo menej dôležitý. Výberom vhodného typu reprezentácie agregačnej funkcie je možné definovať, aj na ktorých atribútoch užívateľovi záleží, na ktorých mu nezáleží.

Globálne ohodnotenie objektu $x$ vzniká ako výsledok agregačnej funkcie $@(p_1, \ldots, p_m)$, kde $p_1, \ldots, p_m$ sú už vypočítané hodnotenia objektu podľa hodnôt jeho $m$ jednotlivých pomocou fuzzy funkcií, $p_i = f_i^U(x)$.

### Aritmetický priemer

Najobecnejší typ agregačnej funkcie je aritmetický priemer

$$@(x) = \frac{(p_1 + \ldots + p_m)}{m} .$$

### Vážený priemer

Pre zavedenie užívateľského vplyvu do agregačnej funkcie je možné použiť vážený priemer, kde váhy jednotlivých $u_1, \ldots, u_m$ atribútov $A_1, \ldots, A_m$ určujú na ktorých atribútoch ako užívateľovi záleží

$$@(x) = \frac{(u_1 p_1 + \ldots + u_m p_m)}{(u_1 + \ldots + u_m)} .$$

*Poznámka:* Ak užívateľovi nezáleží pri ohodnocovaní objektov na $i$-tom atribúte, tak pre tento atribút sa do agregačnej funkcie dosadí 0, teda $u_i = 0$. Týmto spôsobom si užívateľ volí *hodnotiace kritériá,* podľa ktorých hodnotí objekty. Výber hodnotiacich kritérií je výber podmnožiny atribútov $A_1, \ldots, A_m$, na ktorej je závislá agregačná funkcia, na ktorej je závislé hodnotenie objektov.

### Euklidovská vzdialenosť

Ďalším typom agregačnej funkcie môže byť euklidovská vzdialenosť

$$@(x) = \sqrt{\frac{(p_1^2 + \ldots + p_m^2)}{m}} .$$

Taktiež v prípade euklidovskej vzdialenosti je možné zaviesť do agregačnej funkcie užívateľský vplyv nasledujúcou úpravou

$$@(x) = \sqrt{\frac{(u_1 p_1^2 + \ldots + u_m p_m^2)}{u_1 + \ldots + u_m}} \quad .$$

**Minimum, Maximum**

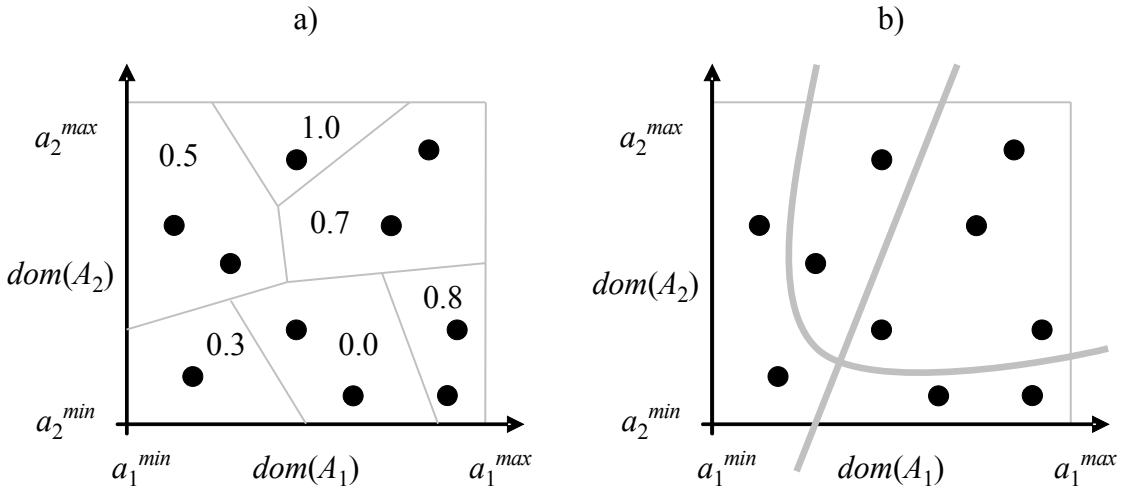
Ako agregačná funkcia môžu byť použité aj funkcie minimum alebo maximum.

$$@(x) = min(p_1, \ldots, p_m),$$

$$@(x) = max(p_1, \ldots, p_m).$$

### 2.3.3 Ďalšie modely

Ďalším často používaným modelom, pomocou ktorého je možné získavať globálne ohodnotenia objektov, sú *bodovacie funkcie*. Nech $X$ je množina objektov, ktoré majú $m$ atribútov $A_1, \ldots, A_m$ s doménami $dom(A_i)$. Každý objekt $x \in X$ má definovaných $m$ hodnôt týchto atribútov $a_1^x, \ldots, a_m^x$. Bodovacia funkcia sa definuje na $m$-rozmernom priestore $D = dom(A_1) \times \ldots \times dom(A_m)$. Objekty $x \in X$ sa chápu ako body $m$-rozmerného priestoru $D$. Bodovacia funkcia môže byť ľubovoľné zobrazene, ktoré bude objekty, čiže body priestoru $D$, zobrazovať na interval [0, 1].




*Obrázok 2.3.3, Bodovacie funkcie*

Na obrázku 2.3.3 sú znázornené dva grafy. Nech množina objektov $X$ obsahuje objekty, ktoré majú dva atribúty $A_1$, $A_2$. Na každom z grafov x-ová os reprezentuje doménu atribútu $dom(A_1)$ a y-ová os reprezentuje doménu atribútu $dom(A_2)$. Podľa hodnôt atribútov sú v grafoch znázornené objekty $x \in X$ ako body, ktoré ležia na súradniciach $\{a_1^x, a_2^x\}$. Dvojrozmerný interval $[a_i^{min}, a_i^{max}] \times [a_i^{min}, a_i^{max}]$ sa označí ako $I$.

Prípad a) na obrázku znázorňuje bodovaciu funkciu, ktorej interval $I$ je rozdelený na triedy ekvivalencie, ktoré majú priradené ohodnotenia z intervalu [0, 1]. Globálne ohodnotenie objektu sa získa tak, že sa podľa jeho hodnôt atribútov zistí, v ktorej triede ekvivalencie sa nachádza, a objektu sa priradí ohodnotenie tejto triedy. Objekty, ktoré sa nachádzajú v rovnakej triede ekvivalencie, majú rovnaké hodnotenie.

Prípad b) znázorňuje bodovaciu funkciu, pomocou ktorej je možné získať globálne ohodnotenie objektu v závislosti na vzdialenosti jeho obrazu od priamky alebo krivky.

## 2.4 Použitý model

Pre potreby vyhľadávania $K$ najlepších objektov podľa užívateľských preferencií by mal byť zvolený model, ktorý je pre užívateľa príjemný a užívateľ dokáže pomocou tohto modelu najvýstižnejšie popísať svoje preferencie. Treba zohľadniť aj fakt, že bežný užívateľ nebude chcieť modelovať svoje preferencie napr. pomocou definovania analytických funkcií alebo delením viacrozmerného priestoru na triedy ekvivalencie.

Algoritmy popisované v ďalších kapitolách vyhľadávajú $K$ najlepších objektov vzhľadom na agregačnú funkciu. Preto požiadavka kladená na použitý model je podpora agregačnej funkcie.

Táto práca teda vychádza z modelu, kde pre každý atribút $A_i$ existuje užívateľská fuzzy funkcia $f_i^U$, ktorá lokálne ohodnocuje objekty podľa $i$-tého atribútu. Následne sa výsledky jednotlivých lokálnych ohodnotení objektu $x$ agregujú pomocou užívateľskej agregačnej funkcie $@^U$ do globálneho ohodnotenia objektu $x$ [7]. Každý objekt $x$ z množiny objektov $X$ má určené svoje globálne ohodnotenie, mieru vhodnosti pre užívateľa $U$.

Podľa globálneho ohodnotenia jednotlivých objektov je možné porovnávať mieru vhodnosti objektov pre užívateľa a usporiadavať objekty podľa tejto miery vhodnosti. V množine objektov $X$ sa tak dá jednoznačne určiť $K$ najlepších objektov pre užívateľa.

# 3 Top-k problém

V kapitole 2 bol zavedený model, podľa ktorého je možné ohodnocovať objekty $x \in X$, z danej množiny objektov $X$ s $m$ atribútmi $A_1$, …, $A_m$, pomocou ohodnocovacej funkcie $F(x)$ súčasne podľa viacerých atribútov. Algoritmy, ktoré dokážu nájsť $K \geq 1$ objektov z množiny objektov $X$ s najvyšším ohodnotením $F(x)$, sa označujú *top-k algoritmy*.

Vstupom top-k algoritmu je dátová štruktúra, v ktorej sú uložené dáta o objektoch (množina objektov $X$ s hodnotami atribútov), ohodnocovacia funkcia $F(x)$ a číslo $K$. Výstupom top-k algoritmu je $K$ objektov $x \in X$, ktoré majú najvyššie ohodnotenie $F(x)$.

## 3.1 Základný algoritmus

Algoritmicky najjednoduchšie a zároveň korektné riešenie top-k problému, poskytuje tzv. základný algoritmus.

Popis základného algoritmu:

I. Pre každý objekt $x \in X$ sa načítajú hodnoty všetkých $m$ atribútov, podľa ktorých je objekt ohodnocovaný.

II. Pre všetky objekty $x \in X$ sa podľa ich $m$ hodnôt ohodnocovacích atribútov vypočíta globálne ohodnotenie objektu $F(x)$.

III. Ohodnotené objekty $x \in X$ sa zotriedia zostupne podľa $F(x)$, vzniká zoznam $Z$.

IV. Vyberie sa prvých $K$ objektov zoznamu $Z$.

### 3.1.1 Viacužívateľský prístup

Základný algoritmus dokáže nájsť $K$ najlepších objektov z množiny objektov $X$ vzhľadom na ohodnocovaciu funkciu $F(x)$. Každý užívateľ však môže preferovať objekty $x \in X$ odlišným spôsobom. Každý vyjadruje svoje preferencie pomocou svojej ohodnocovacej funkcie. Určitý objekt $x$ môže byť pre jedného užívateľa $U_1$ najvhodnejší $F_1(x) = 1$ a pre iného užívateľa $U_2$ už objekt $x$ už nemusí byť najvhodnejší $F_2(x) < 1$. Pri hľadaní $K$ najlepších objektov $x \in X$ pre dvoch užívateľov $U_1$ a $U_2$ s rôznymi preferenciami $F_1$ a $F_2$ nájde top-k algoritmus vo všeobecnosti $K$ rôznych objektov.

Pred spustením top-k algoritmu musí užívateľ $U$ zadať svoje preferencie, zadať ohodnocovaciu funkciu $F(x)$. Podľa použitého modelu z kapitoly 2.4 užívateľ zadá užívateľ svoje lokálne preferencie (fuzzy funkciu pre každý z ohodnocovaných atribútov) globálne preferencie (agregačnú funkciu). Následne sa pre takto zadanú funkciu $F(x)$ spustí top-k algoritmus, ktorý nájde $K$ najlepších objektov vzhľadom na $F(x)$, vzhľadom na zadané preferencie užívateľa $U$. Pre každú zadanú ohodnocovaciu funkciu (pre každého užívateľa) sa vykoná vždy nový výpočet top-k algoritmu.

Ako zavedenie viacužívateľského prístupu pri riešení top-k problému je v tejto práci uvažovaný prípad, kde dáta o objektoch sú spoločné pre všetkých užívateľov. V závislosti na konkrétnom užívateľovi sa mení len ohodnocovacia funkcia $F(x)$.

### 3.1.2 Počet prístupov

Pri veľkej mohutnosti $N$ množiny $X$ s $m$ atribútmi je výpočet základného algoritmu časovo a pamäťovo veľmi náročný. Hlavným problémom je potreba načítania všetkých $m$ hodnôt atribútov všetkých objektov množiny $X$. Nech sa načítanie jednej hodnoty atribútu pre jeden objekt označí ako *jeden prístup* do dátovej štruktúry, v ktorej sú uložené všetky objekty $x$ množiny $X$. Základný algoritmus musí načítať celú množinu $X$, musí teda vykonať $N \cdot m$ prístupov.

Výstupom top-k algoritmu je $K$ najlepších objektov (s hodnotami $m$ atribútov). Top-k algoritmy sú zamerané na vyhľadávanie malého počtu objektov z danej množiny $X$ (napr. $K = 10$). Pritom množina objektov $X$ môže obsahovať rádovo oveľa viac objektov ako $K$ požadovaných (napr. $N = 500\,000$). Z tohto hľadiska je základný algoritmus neefektívny.

Táto práca sa v ďalších kapitolách (viz kapitoly 4 a 7) venuje top-k algoritmom, ktoré dokážu nájsť $K$ najlepších objektov bez prejdenia všetkých objektov (bez vykonania $N \cdot m$ prístupov).

## 3.2 Použitý zdroj dát

Práca predpokladá, že existuje množina objektov rovnakého druhu $X$ s $m$ atribútmi $A_1, \ldots, A_m$, kde každý objekt má $m$ hodnôt atribútov. Objekty $x \in X$ sú pritom uložené a indexované v relačnej databáze. Pre prehľadnosť sa predpokladá, že množina objektov $X$ je uložená v relačnej databáze ako jedna tabuľka. Táto tabuľka má vo všeobecnosti $n \geq m$ atribútov, kde podľa $m$ atribútov $A_1, \ldots, A_m$ sa určuje ohodnotenie objektov a ostatné atribúty obsahujú doplňujúce informácie o objektoch.

Jeden z týchto doplňujúcich atribútov môže byť napríklad kľúčový atribút $A_{\mathrm{ID}}$, ktorého hodnotami sú unikátne identifikátory objektov. Vo všeobecnosti je potom úlohou top-k algoritmu je nájsť v relačnej databáze identifikátory $K$ najlepších objektov vzhľadom na danú ohodnocovaciu funkciu $F(x)$.

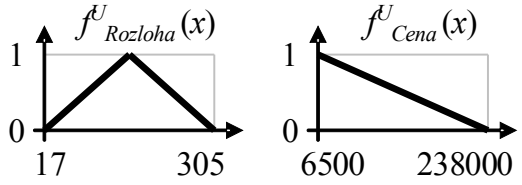
### 3.2.1 Generovanie dotazov

Najjednoduchším riešením problému top-k je nájsť $K$ najlepších objektov pre užívateľa podľa jeho ohodnocovacej funkcie $F(x)$ pomocou dotazovacieho jazyka (napr. SQL). Podľa zadaných užívateľských preferencií $F(x)$ sa vytvorí dotaz typu SELECT. Tento dotaz sa spustí priamo v databázovom stroji, ktorý vráti množinu $K$ najlepších objektov.

**Príklad 1, Generovanie dotazu**

Nech sú v jednej tabuľke v relačnej databáze uložené byty s atribútmi Id, Velkost, Mesto, Lokalita, Rozloha a Cena. Užívateľ $U$ tieto byty preferuje podľa atribútov Rozloha a Cena. Ostatné atribúty nemajú vplyv na ohodnotenie bytov. Svoje lokálne preferencie vyjadruje užívateľ pomocou dvoch užívateľských fuzzy funkcií $f^{U}_{Cena}(x)$ a $f^{U}_{Rozloha}(x)$.

Užívateľ môže napríklad preferovať byty s hodnotou atribútu Rozloha v strede domény atribútu, nech je analytické vyjadrenie príslušnej fuzzy funkcie $y = (1 - 2 \cdot | x - 0.5 | )$. Okrem toho môže užívateľ preferovať byty s nižšou hodnotou atribútu Cena, nech je analytické vyjadrenie príslušnej fuzzy funkcie $y = (1 - x)$.




Pre normalizáciu domén jednotlivých atribútov na interval [0, 1] platí:

$N_{Rozloha}(x) = (a_{Rozloha}{}^x - 17) / (305 - 17),$

$N_{Cena}(x) = (a_{Cena}{}^x - 6500) / (238000 - 6500).$

Potom normalizované fuzzy funkcie pre atribúty Rozloha a Cena vyzerajú nasledovne:

$f^U_{Rozloha}(x) = (1 - 2 \cdot |N_{Rozloha}(x) - 0.5| ),$

$f^U_{Cena}(x) = (1 - N_{Cena}(x) ).$

Globálne preferencie vyjadrí užívateľ pomocou užívateľskej agregačnej funkcie $@(x)$. Napríklad užívateľ môže použiť vážený priemer, kde váhy atribútov Rozloha a Cena sú v pomere 2 : 3. Potom pre ohodnocovaciu funkciu $F(x)$ platí:

$$F(x) = \frac{2 \cdot f^U_{Rozloha}(x) + 3 \cdot f^U_{Cena}(x)}{2+3}$$

Podľa ohodnocovacej funkcie $F(x)$ sa vygeneruje SQL dotaz, pomocou ktorého sa v relačnej databáze vyhľadá $K$ najlepších bytov vzhľadom na $F(x)$. Pre $K = 10$ môže dotaz vyzerať nasledovne:

```
SELECT Id, Velkost, Mesto, Lokalita, Rozloha, Cena,
(  2*  (1 - 2 * abs((( Rozloha -17)/288) - 0.5) )  +
   3*  (1 - (( Cena -6500)/231500) )
)  / 5 as ohodnotenie
from byty order by ohodnotenie desc limit 5
```

Výsledok:

| *Id* | *Velkost* | *Mesto* | *Lokalita* | *Rozloha* | *Cena* | *ohodnotenie* |
|---|---|---|---|---|---|---|
| 43581 | 4 + 1 | Praha 9 | Kyje | 156 | 16000 | 0,96148908 |
| 43332 | 4 + 1 | Praha 2 | Vinohrady | 157 | 25000 | 0,94094073 |
| 43426 | 4 + 1 | Praha 9 | Letňany | 150 | 20000 | 0,93445524 |
| 42820 | 5 + kk | Praha 5 | Slivenec | 140 | 14000 | 0,92222822 |
| 41760 | 3 + 1 | Praha 4 | Krč | 170 | 27000 | 0,92186825 |

Pri každom novom zadaní užívateľských preferencií sa musí vygenerovať nový SQL dotaz, ktorý sa vykoná priamo v relačnej databáze. Nevýhodou takéhoto riešenia je potreba pri každom vykonaní dotazu prepočítať ohodnotenie všetkých objektov, čiže prejsť všetky riadky tabuľky. Nakoniec je ešte potrebné objekty podľa ohodnotenia zotriediť. Takéto riešenie je neefektívne, pretože je potrebné vykonať $N \cdot m$ prístupov.

# 4 Faginov algoritmus

V článku [1] R. Fagin a kolektív popisujú algoritmy *TA* (threshold algorithm) a *NRA* (no random access), ktoré riešia top-k problém bez prejdenia všetkých objektov. V ďalšom texte sa budú tieto algoritmy jednotne nazývať ako *Faginov algoritmus*.

## 4.1 Predpoklady

Faginov algoritmus predpokladá, že existuje konečná množina objektov $X$ rovnakého druhu s mohutnosťou $N$, s $m$ atribútmi $A_1, \ldots, A_m$. Každý objekt $x$ z množiny $X$ má $m$ hodnôt atribútov $a_1^x, \ldots, a_m^x$, podľa ktorých sa pomocou agregačnej funkcie @ určujú ohodnotenia jednotlivých objektov.

Vstupom Faginovho algoritmu je množina objektov $X$, agregačná funkcia @ a číslo $K$. Výstupom top-k algoritmu je $K$ objektov $x \in X$, ktoré majú podľa agregačnej funkcie najvyššie ohodnotenie @($x$).

### 4.1.1 Zostupne zotriedené zoznamy

Faginov algoritmus vyžaduje, aby boli objekty množiny $X$ spolu s hodnotami ich atribútov uložené v $m$ usporiadaných zoznamoch $L_1, \ldots, L_m$, kde $i$-tý zoznam $L_i$ obsahuje *dvojice* objekt a hodnota $i$-tého atribútu, $\{x, a_i^x\}$. Zoznamy $L_1, \ldots, L_m$ sú pritom zostupne zotriedené podľa hodnôt atribútov objektov $x \in X$.

| $L_1$ | $L_2$ | $L_3$ | $L_4$ | $L_5$ |
|---|---|---|---|---|
| $\{x_1, 1.0\}$ | $\{x_3, 1.0\}$ | $\{x_1, 1.0\}$ | **$\{x_2, 1.0\}$** | $\{x_6, 1.0\}$ |
| **$\{x_2, 0.8\}$** | $\{x_4, 0.7\}$ | $\{x_4, 0.9\}$ | $\{x_4, 0.8\}$ | $\{x_4, 0.8\}$ |
| $\{x_3, 0.5\}$ | $\{x_6, 0.6\}$ | **$\{x_2, 0.8\}$** | $\{x_6, 0.5\}$ | $\{x_5, 0.7\}$ |
| $\{x_4, 0.4\}$ | $\{x_1, 0.3\}$ | $\{x_3, 0.4\}$ | $\{x_1, 0.3\}$ | $\{x_3, 0.5\}$ |
| $\{x_5, 0.2\}$ | $\{x_5, 0.1\}$ | $\{x_5, 0.3\}$ | $\{x_3, 0.2\}$ | **$\{x_2, 0.3\}$** |
| $\{x_6, 0.0\}$ | **$\{x_2, 0.0\}$** | $\{x_6, 0.0\}$ | $\{x_5, 0.0\}$ | $\{x_1, 0.0\}$ |

*Obrázok 4.1.1, Pozície objektu $x_2$ v zoznamoch $L_1, \ldots, L_5$*

Na obrázku 4.1.1 je znázornené, akým spôsobom sa môžu objekty vyskytovať v piatich zoznamoch $L_1, \ldots, L_5$. Napríklad objekt $x_2$ je v zozname $L_1$ ako prvý, má najvyššiu hodnotu atribútu. V zozname $L_5$ je zase objekt $x_2$ ako posledný. Hodnota prvého atribútu $a_1^{x1}$ objektu $x_1$ je v zozname $L_1$ väčšia ako hodnota prvého atribútu $a_1^{x2}$ objektu $x_2$, preto je objekt $x_1$ v zozname vyššie. V zozname $L_4$ je poradie týchto dvoch objektov opačné.

Faginov algoritmus pri výpočte postupuje zoznamami $L_1$ , …, $L_m$ zhora nadol. Pretože všetky zoznamy $L_1$ , …, $L_m$ sú zotriedené zostupne podľa hodnôt atribútov objektov, priechodom $i$-tého zoznamu $L_i$ zhora nadol sa získavajú dvojice $\{x, a_i^x\}$ v zostupnom poradí podľa hodnôt $a_i^x$, počnúc dvojicou s najväčšou hodnotou $a_i^x$ až po dvojicu s najmenšou hodnotou $a_i^x$.

### 4.1.2 Monotónnosť agregačnej funkcie

Faginov algoritmus vyhľadáva $K$ najlepších objektov vzhľadom na agregačnú funkciu. Pomocou agregačnej funkcie @ sa ohodnocujú objekty, na ktoré algoritmus narazil počas zostupovania v zoznamoch. Tvorí sa tak množina $K$ najlepších objektov, až do okamihu, kedy je zrejmé, že ďalším zostupovaním v zoznamoch sa už nedá nájsť lepší objekt ako dovtedy nájdených $K$ najlepších objektov. Potom už algoritmus nemusí v zoznamoch zostupovať ďalej a algoritmus končí.

Aby Faginov algoritmus takýmto spôsobom korektne našiel $K$ najlepších objektov skôr ako príde až na koniec všetkých zoznamov, musí byť agregačná funkcia @ podľa článku [1] *monotónna* vzhľadom ku zoradeniam v zoznamoch $L_1$, …, $L_m$ .

Funkcia $f$ s $m$ premennými je monotónna vzhľadom na všetky premenné práve vtedy keď platí:

$$\forall i \in \{1, \ldots, m\} : p_i \leq q_i \Rightarrow f(p_1, \ldots, p_m) \leq f(q_1, \ldots, q_m).$$

Takáto monotónna funkcia $f$ môže byť vzhľadom na všetky svoje premenné netrasúca alebo neklesajúca. Keďže zoznamy $L_1$, …, $L_m$ sú zotriedené zostupne podľa hodnôt atribútov objektov, pre správnu funkčnosť Faginovho algoritmu je potrebné použiť práve neklesajúcu agregačnú funkciu @.

*Dôsledok*: Nech má objekt $x_1 \in X$ hodnoty atribútov $a_1^{x1}$, …, $a_m^{x1}$ a objekt $x_2 \in X$ hodnoty atribútov $a_1^{x2}$, …, $a_m^{x2}$. Ohodnotenie objektu $x_1$ podľa agregačnej funkcie je potom $@(x_1) = @(a_1^{x1}, \ldots, a_m^{x1})$. Nech objekt $x_2$ nemá hodnotu ani jedného z atribútov väčšiu ako príslušná hodnota atribútu objektu $x_1$, $\forall i \in \{1, \ldots, m\} : \; a_i^{x2} \leq a_i^{x1}$. Ak je agregačná funkcia @ neklesajúca vzhľadom na všetky jej premenné, potom pre ohodnotenie objektu $x_2$ platí $@(x_2) \leq @(x_1)$.

Agregačné funkcie ako aritmetický priemer, vážený priemer, euklidovská vzdialenosť, minimum a maximum sú neklesajúce vzhľadom na všetky svoje premenné. Môžu byť preto použité ako agregačná funkcia @ vo Faginovom algoritme.

Pri zostupovaní v zostupne zotriedených zoznamoch využíva Faginov algoritmus vlastnosti monotónnej agregačnej funkcie pre odhad ohodnotenia zatiaľ neznámych objektov.

*Príklad:* Nech sa v zoznamoch $L_1$, …, $L_m$ na obrázku 4.1.1 vyhľadajú všetky hodnoty atribútov objektu $x_4$ priechodom každého zoznamu zhora nadol. Z každého $i$-tého zoznamu sa získavajú $\{x, a_i^x\}$ dvojice až dovtedy, kým sa nenarazí na dvojicu $\{x_4, a_i^{x4}\}$. Počas tohto získavania dvojíc sa v žiadnom zo zoznamov nezíska dvojica $\{x_5, a_i^{x5}\}$. Pretože zoznamy $L_1$, …, $L_m$ sú zotriedené zostupne podľa hodnôt atribútov objektov, pre hodnoty atribútov objektu $x_5$ bude platiť $\forall i \in \{1, \ldots, m\} : \; a_i^{x5} \leq a_i^{x4}$. Potom z monotónnosti agregačnej funkcie vyplýva, že $@(x_5) \leq @(x_4)$. Takto je možné odhadnúť ohodnotenie objektu $x_5$ aj bez znalosti hodnôt jeho atribútov.

### 4.1.3 Pristupovanie do zoznamov

K objektom, ktoré sú v zoznamoch uložené ako dvojice $\{x, a_i^x\}$, je možné pristupovať dvoma spôsobmi.

- *Priamy prístup*
  - umožňuje v *i*-tom zozname $L_i$ priamo pristúpiť k požadovanému objektu $x$ bez ohľadu na pozíciu dvojice $\{x, a_i^x\}$ v zozname $L_i$.
- *Sekvenčný prístup*
  - umožňuje pristupovať k objektom v zozname postupne zhora nadol po jednotlivých dvojiciach $\{x, a_i^x\}$.

Podľa spôsobu pristupovania do zoznamov $L_1$ , …, $L_m$ sú v článku [1] uvedené dva základné algoritmy TA a NRA. TA algoritmus využíva okrem sekvenčného prístupu aj priamy prístup. NRA algoritmus využíva výhradne iba sekvenčný prístup. Oba tieto algoritmy teda využívajú sekvenčný prístup k objektom v zoznamoch.

Spôsob, akým sa sekvenčne pristupuje do zoznamov $L_1$ , …, $L_m$ sa označuje ako *heuristika*. Heuristika určuje do ktorých zo zoznamov $L_1$ , …, $L_m$ má algoritmus v ďalšom kroku sekvenčne pristúpiť, z ktorých zoznamov má algoritmus získať nové objekty aj s príslušnými hodnotami atribútov.

V článku [1] sa predpokladá, že do zoznamov je možné pristupovať *paralelne*. V tomto článku je popisovaná pre sekvenčný prístup do zoznamov pôvodná *heuristika TA*, podľa ktorej sa v každom kroku Faginovho algoritmu získajú objekty zo všetkých zoznamov $L_1$, …, $L_m$. Algoritmus tak v každom kroku získa $m$ objektov, vykoná $m$ prístupov.

V tejto práci sa pre názornosť uvažujú *jednoprístupové heuristiky*, ktoré v každom kroku algoritmu určia iba jeden *i*-tý zoznam $L_i$ v ktorom sa vykoná jeden sekvenčný prístup a získa sa tak jeden objekt $x$ hodnotou jeho *i*-tého atribútu $a_i^x$, čiže sa získa dvojica $\{x, a_i^x\}$. Jednoprístupová heuristika $H$ sa dá vnímať ako zobrazenie $H : p \rightarrow \{1, ..., m\}$, kde $p$ vyjadruje $p$-ty krok algoritmu pri ktorom sa získava sekvenčným prístupom nový objekt zo zoznamu $L_i$, kde $i \in \{1, ..., m\}$.

Každá heuristika, ktorá do zoznamov pristupuje paralelne sa dá jednoducho previesť na jednoprístupovú heuristiku. Napríklad pôvodnú heuristiku TA je možné previesť na jednoprístupovú heuristiku takú, že v prvom kroku Faginovho algoritmu sa pristúpi do prvého zoznamu $L_1$, v druhom kroku do zoznamu $L_2$, v $m$-tom kroku do zoznamu $L_m$, v ($m$+1)-tom kroku znovu do zoznamu $L_1$ a podobne ďalej. Táto heuristika sa označí ako $H_{TA}$ a je možné ju vyjadriť ako $H_{TA}(p) = p \ modulo \ m$.

## 4.2 TA algoritmus

Algoritmus TA potrebuje ako vstup zostupne zotriedené zoznamy $L_1$, …, $L_m$ dvojíc $\{x, a_i^x\}$ a monotónnu agregačnú funkciu @.

TA algoritmus potrebuje počas svojho výpočtu do týchto zoznamov pristupovať sekvenčne a aj priamo. Pri sekvenčnom prístupe TA algoritmus prechádza zoznamy $L_1$, …, $L_m$ zhora nadol a podľa heuristiky $H$ (napr. heuristika $H_{TA}$) z nich postupne získava dvojice $\{x, a_i^x\}$. Pri priamom prístupe potrebuje TA algoritmus z *i*-tého zoznamu $L_i$ pre určitý objekt $x$ okamžite získať dvojicu $\{x, a_i^x\}$. Algoritmus TA teda predpokladá,

že zoznamy $L_1$, …, $L_m$ takéto priame získanie dvojice umožňujú. Priamy prístup a sekvenčný prístup do zoznamov $L_1$, …, $L_m$ sú navzájom nezávislé operácie. Vykonanie priameho prístupu do zoznamu teda neovplyvní pozíciu v zozname, kam sa algoritmus TA dostal počas sekvenčného pristupovania v tomto zozname. Spôsob realizácie priameho prístupu do zoznamov závisí na konkrétnej implementácii a na použitých dátových štruktúrach.

Algoritmus TA používa zoznam $T_K$, v ktorom si uchováva aktuálnych $K$ najlepších objektov zoradených podľa ich ohodnotenia. Počas svojho výpočtu algoritmus TA prepočítava *prahovú hodnotu* (threshold) označovanú ako $t_h$, ktorá sa vypočíta dosadením naposledy videných hodnôt atribútov $a_1^{pos}$, …, $a_m^{pos}$ v zoznamoch $L_1$, …, $L_m$ pri sekvenčnom prístupe do agregačnej funkcie @. Potom pre prahovú hodnotu platí $t_h = @(a_1^{pos}, \ldots, a_m^{pos})$. Hodnoty naposledy videných atribútov sú vo všeobecnosti hodnotami atribútov rôznych objektov naposledy získaných v zoznamoch $L_1$, …, $L_m$ pomocou sekvenčného prístupu.

Popis algoritmu TA:

I. Heuristika $H$ vyberie $i$-ty zoznam $L_i$ v ktorom sa vykoná sekvenčný prístup a v zozname $L_i$ sa získa ďalší najlepší objekt $x$.

   Keď ešte objekt $x$ nebol videný v žiadnom z predchádzajúcich prístupov:

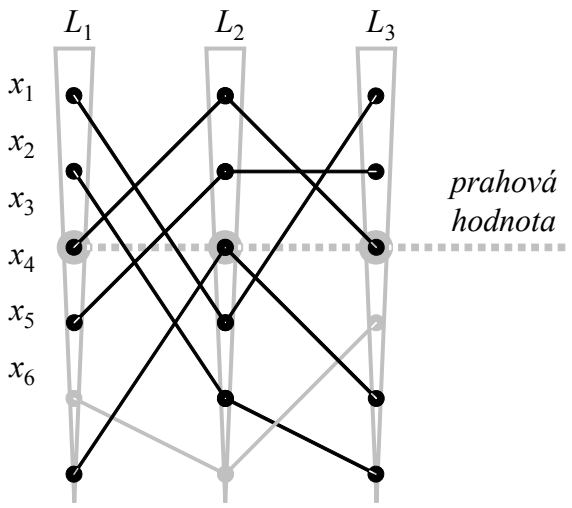
   - Priamym prístupom do ostatných zoznamov sa získajú všetky ďalšie hodnoty atribútov objektu $x$.
   - Pomocou @ sa vypočíta nové ohodnotenie $@(x)$ objektu $x$.
   - Ak v zozname $T_K$ ešte nie je $K$ objektov, objekt $x$ sa vloží do zoznamu $T_K$ na správne miesto podľa $@(x)$.
   - V opačnom prípade sa $@(x)$ porovná s ohodnotením $K$-teho najlepšieho objektu v $T_K$. Keď je $@(x)$ väčšie, $K$-ty objekt sa zo zoznamu $T_K$ vyhodí a objekt $x$ sa vloží do $T_K$ na správne miesto.

II. Vypočíta sa nová prahová hodnota $t_h$. Ak $K$-ty objekt zoznamu $T_K$ má väčšie ohodnotenie ako je $t_h$, algoritmus končí a vráti zoznam $T_K$ ako $K$ najlepších objektov. Inak algoritmus prejde na fázu I.

Podľa článku [1] algoritmus TA korektne nájde $K$ najlepších objektov vzhľadom na agregačnú funkciu @.

Po skončení algoritmu TA zoznam $T_K$ obsahuje $K$ najlepších objektov a ohodnotenie objektov je kompletné, pretože pre každý z objektov sa priamym prístupom do ostatných zoznamov zistili všetky hodnoty ich atribútov. Algoritmus končí, keď prahová hodnota $t_h$ klesne natoľko, že je menšia ako ohodnotenie všetkých $K$ najlepších doposiaľ videných objektov, čiže je menšia ako ohodnotenie $K$-teho objektu v zozname $T_K$. Ďalšie objekty $x$, ktoré by sa získali v zoznamoch $L_1$, …, $L_m$, by mali všetky hodnoty atribútov $a_1^x$, …, $a_m^x$ menšie alebo rovné ako naposledy videné hodnoty atribútov $a_1^{pos}$, …, $a_m^{pos}$, pretože zoznamy $L_1$, …, $L_m$ sú zostupne zotriedené podľa hodnôt atribútov objektov. Z monotónnosti agregačnej funkcie vyplýva, že ohodnotenie takýchto objektov $x$ by teda bolo menšie ako $@(x) \leq t_h$ . Preto už nie je potrebné zo zoznamov získavať ďalšie objekty a algoritmus TA končí.

| $L_1$ | $L_2$ | $L_3$ |
|---|---|---|
| $\{x_1, 1.0\}$ | $\{x_3, 1.0\}$ | $\{x_1, 1.0\}$ |
| $\{x_2, 0.8\}$ | $\{x_4, 0.8\}$ | $\{x_4, 0.8\}$ |
| $\{x_3, 0.6\}$ | $\{x_6, 0.6\}$ | $\{x_3, 0.6\}$ |
| $\{x_4, 0.4\}$ | $\{x_1, 0.4\}$ | $\{x_5, 0.4\}$ |
| $\{x_5, 0.2\}$ | $\{x_2, 0.2\}$ | $\{x_2, 0.2\}$ |
| $\{x_6, 0.0\}$ | $\{x_5, 0.0\}$ | $\{x_6, 0.0\}$ |




*Obrázok 4.2, Algoritmus TA na hľadá 3 najlepšie objekty*

**Príklad 2, Použitie prahovej hodnoty**

Na obrázku 4.2 je v tabuľke znázornený príklad troch zoznamov $L_1$, $L_2$, $L_3$, v ktorých je podľa hodnôt atribútov zostupne zoradených šesť objektov $x_1$, …, $x_6$. Na pravom diagrame je znázornená tá istá situácia tak, že body reprezentujú objekty v zozname a pre jeden objekt vo všetkých zoznamoch sú tieto body prepojené úsečkou.

Pre nájdenie troch najlepších objektov stačí aby algoritmus TA podľa heuristiky $H_{TA}$ vykonal deväť sekvenčných prístupov, do každého zoznamu tri. Pri použití agregácie $@(x) = a_1^x + a_2^x + a_3^x$ po deviatom cykle algoritmu pre objekty z $T_K$ platí $@(x_1) = 2.4$, $@(x_3) = 2.2$, $@(x_4) = 2.0$. Pre prahovú hodnotu platí $t_h = @(0.6, 0.6, 0.6) = 1.8$. Algoritmus končí a vracia zoznam troch najlepších objektov $T_K$ s objektmi $x_1$, $x_3$, $x_4$, pretože $@(x_4) > t_h$. V štvrtom cykle algoritmu sa do zoznamu $T_K$ už dostali tri objekty $x_1$, $x_3$, $x_2$. V piatom cykle sa do $T_K$ dostal objekt $x_4$ a objekt $x_2$ sa vyhodil. Potom v ôsmom cykle sa objekt $x_6$ nedostal do zoznamu $T_K$. Algoritmus TA potom v deviatom cykle skončil aj bez toho, aby sa niečo dozvedel o objekte $x_6$.

## 4.3 NRA algoritmus

Algoritmus NRA, podobne ako TA potrebuje ako vstup zostupne zotriedené zoznamy $L_1$, …, $L_m$ dvojíc $\{x, a_i^x\}$ a monotónnu agregačnú funkciu @. Algoritmus NRA nepoužíva priamy prístup do zoznamov. Pre objekt $x$ nie je možné okamžite získať chýbajúce hodnoty atribútov objektu $x$ a vypočítať jeho presné ohodnotenie $@(x)$. Preto sa pre objekt $x$ zavádza *najhoršie možné ohodnotenie* $W(x)$ a *najlepšie možné ohodnotenie* $B(x)$.

$W$-ohodnotenie objektov sa vypočíta pomocou agregačnej funkcie @, do ktorej sa namiesto chýbajúcich hodnôt atribútov objektu dosadí 0. Napríklad ak sú pre objekt $x$ známe iba hodnoty atribútov $a_1^x$, $a_3^x$, potom pre jeho najhoršie možné ohodnotenie platí $W(x) = @(a_1^x, 0, a_3^x, 0, ..., 0)$.

Podobným spôsobom sa pomocou agregačnej funkcie @ vypočíta aj $B$-ohodnotenie objektov. Zo zostupného zoradenia $L_1$, …, $L_m$ zoznamov je zrejmé, že chýbajúce hodnoty atribútov objektu $x$ budú menšie alebo rovné ako naposledy videné hodnoty

atribútov $a_1^{pos}$, …, $a_m^{pos}$ v zoznamoch $L_1$, …, $L_m$. Preto sa do agregačnej funkcie namiesto chýbajúcich hodnôt atribútov objektu $x$ dosadia hodnoty $a_1^{pos}$, …, $a_m^{pos}$. Napríklad ak sú pre objekt $x$ známe len hodnoty jeho atribútov $a_2^x$, $a_m^x$, potom $B(x) = @(a_1^{pos}, a_2^x, a_3^{pos}, ..., a_{m-1}^{pos}, a_m^x)$.

Z monotónnosti agregačnej funkcie vyplýva, že pre presné ohodnotenie, $W$-ohodnotenie a $B$-ohodnotenie objektu $x$ platí $W(x) \leq @(x) \leq B(x)$. V prípade že algoritmus doposiaľ nevidel žiadnu hodnotu atribútu objektu $x$, platí $B(x) = t_h = @(a_1^{pos}, …, a_m^{pos})$. Ak sú všetky hodnoty atribútov objektu o známe, platí $W(x) = @(x) = B(x)$.

Algoritmus NRA používa zoznam $T_K$ ako *aktuálny zoznam K najlepších objektov*, zotriedený zostupne podľa $W$-ohodnotení, ktorý obsahuje $K$ objektov s najväčšími aktuálnymi $W$-ohodnoteniami. Ak má viacero objektov rovnaké $W$-ohodnotenie, tieto objekty sú zotriedené podľa $B$-ohodnotenia. Ďalej algoritmus používa *množinu kandidátov* $T_C$. V množine $T_C$ si algoritmus udržuje všetky videné objekty, ktoré majú $B$-ohodnotenie väčšie ako $W$-ohodnotenie $K$-teho objektu a nie sú v aktuálnom zozname $K$ najlepších objektov $T_K$. Ako $M_K$ sa bude označovať $W$-ohodnotenie aktuálneho $K$-teho najlepšieho objektu v $T_K$.

Popis algoritmu NRA:

I. Heuristika $H$ vyberie $i$-ty zoznam $L_i$ v ktorom sa vykoná sekvenčný prístup a v zozname $L_i$ sa získa ďalší najlepší objekt $x$.

   Vypočítajú sa nové $W(x)$ a $B(x)$ objektu $x$.

   Keď $W(x) > M_K$ :

   - Keď $x \in T_K$, $x$ sa presunie v $T_K$ na správne miesto, fáza I končí.
   - Keď $x \in T_C$, tak $x$ z $T_C$ nej vyberie.
   - Objekt $x$ sa vloží do $T_K$ na správne miesto.
   - Zo zoznamu $T_K$ sa vyhodí posledný objekt.
   - Keď $B$-ohodnotenie vyhodeného objektu je vyššie nové $M_K$, vyhodený objekt sa vloží do $T_C$.

   Keď $B(x) > M_K$ a $x$ ešte nie je v množine $T_C$, tak sa do nej pridá.

   Keď $B(x) \leq M_K$ a $x$ je v množine $T_C$, tak sa z nej vyhodí.

II. Pre každý objekt $x$ množiny $T_C$ sa vypočíta $B(x)$ a keď $B(x) \leq M_K$, $x$ sa vyhodí z $T_C$. Keď $|T_K| \geq K$ a množina $T_C$ je prázdna, algoritmus končí a vráti zoznam $T_K$ ako $K$ najlepších objektov. Inak sa prejde na fázu I.

Algoritmus NRA podľa článku [1] korektne nájde $K$ najlepších objektov vzhľadom na agregačnú funkciu $@$.

Nevýhodou tohto algoritmu je, že vráti v zozname $T_K$ len množinu $K$ najlepších objektov. Neposkytuje presné hodnoty ohodnotenia objektov a ani presné poradie $K$ najlepších objektov. Taktiež počas výpočtu potrebuje algoritmus v každom kroku preratúvať $B$-ohodnotenia všetkých relevantných objektov a porovnať ich s hodnotou $M_K$ . Tento fakt môže prispieť k celkovému spomaleniu algoritmu, pretože v množine $T_C$ môže byť v krajnom prípade až $N$ objektov, kde $N$ je mohutnosť množiny objektov $X$.

### 4.3.1 3P-NRA algoritmus

Tento algoritmus prevzatý z práce [4] je modifikáciou algoritmu NRA. Algoritmus 3P-NRA rieši hlavnú nevýhodu algoritmu NRA. Snaží sa obmedziť časté prepočítavanie *B*-ohodnotení objektov z množiny kandidátov, ktoré je vyčlenené ako samostatná fáza algoritmu 3P-NRA a počas svojho výpočtu sa algoritmus snaží obmedziť časté prechádzanie touto fázou. Algoritmus 3P-NRA používa zoznam *najlepších objektov T*, ktorý sa udržuje zotriedený zostupne podľa *W*-ohodnotení objektov. Ako $M_K$ sa bude označovať *W*-ohodnotenie aktuálneho *K*-teho najlepšieho objektu v zozname *T*.

Popis algoritmu 3P-NRA:

I. Zostupovanie s heuristikou $H_1$.
   1. Heuristika $H_1$ vyberie *i*-ty zoznam $L_i$ v ktorom sa vykoná sekvenčný prístup a v zozname $L_i$ sa získa ďalší najlepší objekt *x*.
   2. Vypočítajú sa nové $W(x)$ a $B(x)$ objektu *x*.
   3. Keď $x \in T$ a $B(x) < M_K$, objekt *x* sa vyhodí z *T*, inak sa x zaradí v *T* na správne miesto podľa $W(x)$.
   4. Vypočíta sa nové $t_h$. Keď $|T| \geq K$ a $t_h < M_K$ algoritmus prejde na fázu II, inak opakuje fázu I.

II. Odstraňovanie objektov.
   1. Algoritmus prechádza objekty *x* zoznamu *T* od (*K*+1)-tého po posledný a prepočítava $B(x)$. Keď $B(x) < M_K$, *x* sa vyhodí z *T*.
   2. Keď $|T| > K$, algoritmus prejde na fázu III, inak vráti zoznam *T* a končí.

III. Zostupovanie s heuristikou $H_2$.
   1. Heuristika $H_2$ vyberie zoznam $L_i$ v ktorom sa vykoná sekvenčný prístup a v zozname $L_i$ sa získa ďalší najlepší objekt *x*.
   2. Keď $x \in T$ :
      - Vypočítajú sa nové $W(x)$ a $B(x)$ objektu *x*.
      - Keď $B(x) \leq M_K$, objekt *x* sa vyhodí z *T*, inak sa objekt *x* zaradí v *T* na správne miesto podľa $W(x)$.
      - Keď $|T| = K$, algoritmus vráti zoznam *T* a končí.
   3. Ak kleslo $t_h$ alebo sa zmenilo $M_K$, algoritmus prejde na fázu II, inak opakuje fázu III.

Keď agregačná funkcia @ je neklesajúca kladná agregačná funkcia, podľa článku [4] platí, že algoritmus 3P-NRA korektne nájde najlepších *K* objektov a algoritmus 3P-NRA pritom vykoná rovnaký počet sekvenčných prístupov ako algoritmus NRA.

Z popisu algoritmu je zrejmé, že algoritmus 3P-NRA potrebuje oveľa menej prepočtov *B*-ohodnotení ako NRA. Kým v NRA sa prepočítava *B*-ohodnotení pre všetky objekty množiny $T_C$ (analogicky v 3P-NRA sú to objekty zoznamu *T* od $(K + 1)$-tého až po posledný) po každom sekvenčnom prístupe, 3P-NRA v prvej a tretej fáze prepočítava iba jedno *B*-ohodnotenie pre každý sekvenčný prístup. Jedinou výnimkou je druhá fáza,

ktorá sa vykonáva až po skončení prvej fázy a v tretej fáze v prípade zmeny $B$-ohodnotenia $K$-teho prvku v zozname $T$ alebo pri klesnutí hodnoty prahu. Keby sa fáza II vynechala, zbytočne by sa niektoré objekty uchovávali v zozname, až pokiaľ by sa na ne nenarazilo pri niektorom sekvenčnom prístupe vo fáze III.

### 4.3.2 Doplnenie chýbajúcich hodnôt atribútov

Algoritmus 3P-NRA má oproti TA algoritmu jednu nevýhodu. Základný algoritmus dokáže vrátiť zoznam top-k objektov v správnom poradí aj s globálnymi ohodnoteniami $@(x)$ objektov $x$, pretože pre každý objekt $x$ priamym prístupom získal všetky jeho hodnoty atribútov.

V prípade algoritmu 3P-NRA je možné ako top-k objektov vrátiť zoznam $T$, ktorý obsahuje objekty iba s $B$-ohodnotením a $W$-ohodnotením. Z konštrukcie algoritmu 3P-NRA je zrejmé, že na to aby algoritmus našiel top-k objektov, nemusí nutne poznať všetky hodnoty atribútov všetkých objektov. Teda medzi top-k objektmi môžu existovať aj objekty, na ktoré algoritmus počas svojho výpočtu nenarazil v niektorých zo zoznamov $L_1, \ldots, L_m$. Ak sú známe všetky hodnoty atribútov objektu $x$, platí $W(x) = @(x) = B(x)$. Ale ak niektorá z hodnôt atribútov nie je známa, platí len slabšia podmienka $W(x) \leq @(x) \leq B(x)$. Preto algoritmus 3P-NRA vo všeobecnosti nedokáže vrátiť presné hodnoty $@(x)$.

Aby bolo možné dosiahnuť presné ohodnotenia $@(x)$ top-k objektov, je potrebné získať chýbajúce hodnoty atribútov týchto objektov v zozname $T$. To sa dá v prípade algoritmu 3P-NRA dosiahnuť pridaním fázy IV, na ktorú algoritmus prejde v prípade, že už našiel top-k objektov a vracia ich v zozname $T$. Namiesto toho aby algoritmus 3P-NRA skončil, vykoná ešte fázu IV.

Popis fázy IV algoritmu 3P-NRA:

IV. Doplnenie chýbajúcich hodnôt atribútov.

1. Ak sú známe všetky hodnoty všetkých objektov $x \in T$, algoritmus vráti zoznam $T$ a končí.
2. Heuristika vyberie také $i$, že niektorému z objektov $y \in T$ chýba $i$-tá hodnota atribútu. V zozname $L_i$ sa potom získa ďalší najlepší objekt $x$.
3. Keď $x \in T$, vypočíta sa nové $W(x)$ objektu $x$.
4. Ak v sa zozname $L_i$ získal objekt $x$ s hodnotou $i$-tého atribútu $a_i^x = 0$, algoritmus nastaví každému objektu $y \in T$ s chýbajúcou $i$-tou hodnotou atribútu $a_i^y = 0$.
5. Algoritmus opakuje fázu IV.

V zozname $T$ algoritmus po vykonaní fáze IV vráti $K$ najlepších objektov s presnými hodnotami ohodnotenia $@(x)$, ktoré sú rovné $W$-ohodnoteniu týchto objektov, $W(x) = @(x)$. V 4. kroku fáze IV sa využíva fakt, že ak v niektorom zozname $L_i$ naraził na objekt s hodnotou atribútu rovnou 0, v zozname $L_i$ nemusí pokračovať. Keďže zoznamy sú zostupne zotriedené, všetky ďalšie objekty zoznamu $L_i$ majú hodnotu atribútu rovnú 0.

Z konštrukcie algoritmu 3P-NRA vyplýva fakt, že nemôže nastať pre niektorý z $K$ najlepších objektov nasledujúca situácia:

1. Objekt $x$ je videný v niektorom zozname a dostane sa do $T$.
2. Objekt $x$ sa z $T$ odstráni vo fáze II.
3. Objekt $x$ je znovu videný v niektorom zozname a dostane sa znovu do $T$.

Nemôže sa stať, že algoritmus 3P-NRA vo fáze IV neuvidí chýbajúce hodnoty atribútov $K$ najlepších objektov. Algoritmus preto korektne vráti v zozname $T$ presné hodnoty globálneho ohodnotenia $@(x)$ objektov $x \in T$.

## 4.4 Viacužívateľský prístup

Samotný Faginov algoritmus nedokáže vyhľadať $K$ najlepších objektov vzhľadom na model užívateľských preferencií zavedený v kapitole 2. Predpokladá, že v zoznamoch $L_1, \ldots, L_m$ sú uložené konkrétne hodnoty atribútov $a_1^x, \ldots, a_m^x$ objektov $x \in X$, ako dvojice $\{x, a_i^x\}$, a zoznamy sú podľa týchto hodnôt atribútov $a_i^x$ zostupne zotriedené. Algoritmus potom nájde $K$ najlepších objektov vzhľadom na monotónnu agregačnú funkciu. Podľa zvedeného modelu užívateľských preferencií (viz kapitola 2.4), by mali byť hodnoty atribútov objektov ohodnotené lokálne podľa užívateľských fuzzy funkcií $f_1^U, \ldots, f_m^U$ a globálne ohodnotenie objektu $x$ by sa následne vypočítalo agregáciou lokálnych ohodnotení pomocou užívateľskej agregačnej funkcie $@$.

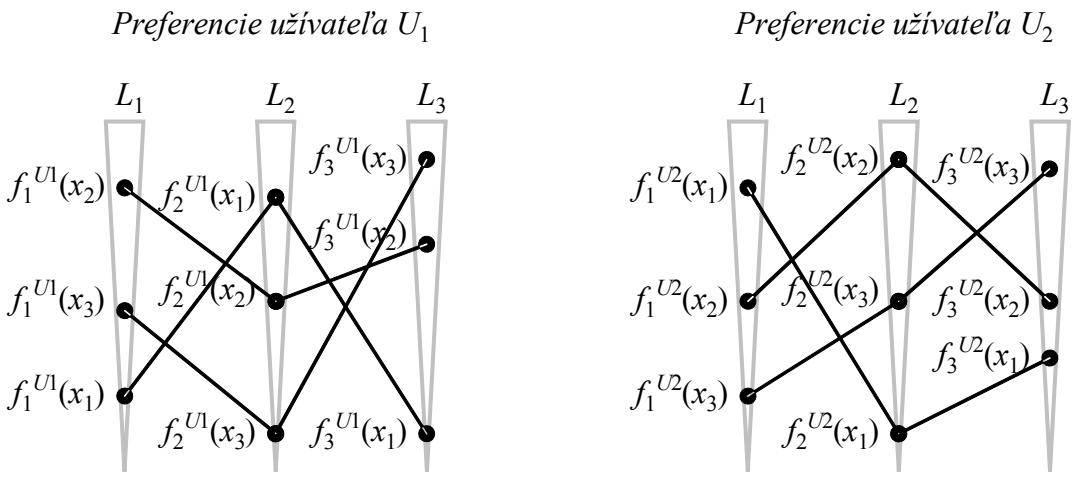
Faginov algoritmus poskytuje možnosť ohodnocovať objekty iba globálne pomocou agregačnej funkcie $@$. Pre zavedenie viacužívateľského prístupu vo Faginovom algoritme je preto potrebné, aby pred výpočtom algoritmu každý $i$-tý zoznam $L_i$ obsahoval pre užívateľa $U$ dvojice: objekt $x$ a hodnota $i$-tého atribútu lokálne ohodnotená pomocou užívateľskej fuzzy funkcie, $\{x, f_i^U(x)\}$. Podľa predpokladov Faginovho algoritmu musí byť každý $i$-ty zoznam $L_i$ zotriedený zostupne, v tomto prípade podľa $f_i^U(x)$.

Po takejto úprave zoznamov sa pri výpočte Faginovho algoritmu do agregačnej funkcie $@$ už dosadzujú pre objekt $x \in X$ hodnoty $f_1^U(x), \ldots, f_m^U(x)$, čím vzniká užívateľská ohodnocovacia funkcia $@^U$ (viz kapitola 2.3.1). Potom Faginov algoritmus nájde $K$ najlepších objektov aj pre užívateľa $U$ podľa jeho užívateľských preferencií.

V jednej množine objektov s viacerými atribútmi majú rôzni užívatelia rôzne lokálne preferencie. Vo všeobecnosti preferujú dvaja užívatelia $U_1$ a $U_2$ objekty $x \in X$ podľa hodnôt $i$-tého atribútu pomocou rôznych fuzzy funkcií. Napríklad jeden užívateľ môže pre jeden atribút preferovať objekty s vyššou hodnotou tohto atribútu. Iný užívateľ zase môže preferovať objekty z nižšou hodnotou atribútu alebo objekty s nejakou optimálnou hodnotou. Pre každého užívateľa sú potom objekty v zoznamoch v špecifickom poradí v závislosti na lokálnych preferenciách užívateľa.

**Príklad 3, Užívatelia s rôznymi lokálnymi preferenciami**

Na obrázku 4.4 sú znázornené tri zoznamy $L_1$, $L_2$, $L_3$ a tri objekty tak, že body reprezentujú hodnoty atribútov objektov $x_1$, $x_2$, $x_3$ lokálne ohodnotené pomocou troch užívateľských fuzzy funkcií. Pre jeden objekt vo všetkých zoznamoch sú tieto body prepojené úsečkou. Obrázok znázorňuje situáciu pre dvoch užívateľov $U_1$ a $U_2$ s rôznymi lokálnymi preferenciami, ktoré pre užívateľa $U_1$ vyjadrujú fuzzy funkcie $f_1^{U1}(x), f_2^{U1}(x), f_3^{U1}(x)$ a pre užívateľa $U_2$ fuzzy funkcie $f_1^{U2}(x), f_2^{U2}(x), f_3^{U2}(x)$.




*Obrázok 4.4, Zoznamy $L_1$, ..., $L_3$ dvoch užívateľov s rôznymi preferenciami*

Na vyhľadanie $K$ najlepších objektov pre užívateľa $U$ by sa museli pred spustením Faginovho algoritmu vytvoriť zoznamy $L_1$, …, $L_m$ podľa jeho užívateľských preferencií $f_1^U$, …, $f_m^U$. To by znamenalo pre každý zoznam $L_i$ vypočítať $f_i^U(x)$ pre každý objekt $x \in X$ a následne objekty zotriediť zostupne podľa $f_i^U(x)$. Pri mohutnosti $N$ množiny $X$ s $m$ atribútmi by to znamenalo $N \cdot m$ prepočtov a $m$ triedení nad $N$ objektmi. Hlavným problémom takéhoto riešenia je fakt, že pred spustením Faginovho algoritmu treba načítať všetky hodnoty atribútov všetkých objektov množiny $X$, čo znamená vykonať $N \cdot m$ prístupov. V tomto prípade už nemá zmysel použiť Faginov algoritmus, ktorý sa snaží vyhľadať $K$ najlepších objektov bez vykonania $N \cdot m$ prístupov.

Problém vytvárania zoznamov podľa užívateľských preferencií pred spustením Faginovho algoritmu je možné efektívne vyriešiť použitím nového modelu zoznamov $L_1$, …, $L_m$. Nový model zoznamov je založený na $B^+$-stromoch, v ktorých sú indexované objekty podľa hodnôt atribútov. $B^+$-stromom sa venuje kapitola 5.

# 5 Indexačné metódy

Kvôli zoznamom Faginovho algoritmu je potrebné vhodným spôsobom uchovávať množinu objektov $X$ s $m$ atribútmi $A_1$, …, $A_m$, kde každý objekt $x \in X$ má $m$ hodnôt atribútov $a_1^x$, …, $a_m^x$. V tejto práci sa preto používajú indexačné metódy založené na stromových štruktúrach. Tieto štruktúry patria k najobvyklejším metódam indexácie na vonkajších pamätiach. Uzly stromov sú tvorené stránkami vonkajšej pamäte. Dáta môžu byť uložené priamo v uzloch stromu (alebo len v listových uzloch), alebo sú dáta uložené mimo strom a zo stromu je na ne iba odkazované.

Dáta sú v strome indexované podľa *kľúčov*. Sú vyhľadávané priechodom cez vnútorné uzly stromu pomocou konkrétneho kľúča. Nech sú v strome uložené dáta o objektoch množiny $X$ a nech sú tieto dáta indexované podľa hodnôt $i$-tého atribútu $A_i$, teda hodnoty z domény $dom(A_i)$ sú v tomto strome kľúčmi. Potom pre $i$-tý atribút $A_i$ sa *vyhľadávacím kľúčom* rozumie jedna konkrétna hodnota $k$ z domény $dom(A_i)$, ku ktorej sa v strome vyhľadávajú objekty množiny $X$ také, že tieto objekty majú hodnotu $i$-tého atribútu práve $k$.

Existuje viacero stromových štruktúr, ktoré sa odlišujú svojou štruktúrou, pravidlami pre prechod vnútornými uzlami a sémantikou dátových položiek.

## 5.1 B-strom

*B-stromy* (*rádu m*) sú rozvetvené výškovo vyvážené stromy [2], [8], ktoré spĺňajú nasledujúce podmienky:

1) koreň má najmenej dvoch potomkov, ak nie je list,
2) každý uzol okrem koreňa a listu má najmenej $\lceil m / 2 \rceil$ a najviac $m$ potomkov,
3) každý uzol má najmenej $\lceil m / 2 \rceil - 1$ a najviac $m - 1$ dátových záznamov,
4) všetky vetvy sú rovnako dlhé,
5) dáta v uzloch sú organizované takto:

   $$p_0, (k_1, p_1, d_1), (k_2, p_2, d_2), ..., (k_n, p_n, d_n), u$$

   kde $p_0, p_1, ..., p_n$ sú ukazovatele na potomkov, $k_1, k_2, ..., k_n$ sú kľúče, $d_1, d_2, ..., d_n$, sú asociované dáta alebo ukazovatele na dáta ležiace mimo strom, $u$ je nevyužitý priestor a záznamy $(k_i, p_i, d_i)$ sú usporiadané vzostupne podľa kľúčov, pričom

   $$\lceil m / 2 \rceil - 1 \leq n \leq m - 1,$$

6) keď zodpovedá ukazovateľu $p_i$ podstrom $U(p_i)$, potom platí:

   (i) pre každé k v $U(p_{i-1})$ je $k < k_i$,

   (ii) pre každé k v $U(p_{i-1})$ je $k > k_i$, kde $i \in \{1, ..., n\}$.

B-stromy podľa tejto definície sa nazývajú *neredundantné*, pretože každý kľúč sa nachádza v strome iba raz. V praktických implementáciách sa však kvôli minimalizácii výšky stromov častejšie používajú *reundantné* B-stromy. Sú to stromy, v ktorých sa asociované dáta (alebo ukazovatele na dáta ležiace mimo strom)

nachádzajú iba v listových uzloch. Oproti pôvodnej definícii neredundantných B-stromov sú v definícii neredundantného B-stromu pozmenené podmienky 5) a 6):

5) dáta v nelistových uzloch sú organizované takto:

$$p_0, (k_1, p_1), (k_2, p_2), ..., (k_n, p_n), u$$

a v listových uzloch takto:

$$(k_1, d_1), (k_2, d_2), ..., (k_n, d_n), u$$

kde $p_0, p_1, ..., p_n$ sú ukazovatele na potomkov, $k_1, k_2, ..., k_n$ sú kľúče, $d_1, d_2, ..., d_n$, sú asociované dáta alebo ukazovatele na dáta ležiace mimo strom, $u$ je nevyužitý priestor, záznamy v nelistových uzloch $(k_i, p_i)$ a záznamy v listových uzloch $(k_i, d_i)$ sú usporiadané vzostupne podľa kľúčov, pričom

$$\lceil m / 2 \rceil - 1 \leq n \leq m - 1,$$

6) keď zodpovedá ukazovateľu $p_i$ podstrom $U(p_i)$, potom platí:

(i) pre každé k v $U(p_{i-1})$ je $k \leq k_i$,

(ii) pre každé k v $U(p_{i-1})$ je $k > k_i$, kde $i \in \{1, ..., n\}$.

Podľa modifikácie podmienky 5) vznikajú dva druhy uzlov, *listové* (*indexové*) a *nelistové* (*riadiace*). Medzi nimi existujú štrukturálne rozdiely, ktoré sa môžu prejaviť pri odlišnej implementácii uzlov. Podľa modifikácie podmienky 6) sa jeden kľúč môže v strome vyskytovať aj viackrát (vzniká redundantný B-strom).

## 5.2 $B^+$-strom

*Rozsahový dotaz* pre dve hodnoty $a$, $b$ vyhľadá v strome dáta pre všetky kľúče $k$ pre ktoré platí, $a \leq k \leq b$. Nevýhodou B-stromov je pomerne komplikovaný algoritmus pre rozsahové dotazy, ktorý musí používať zásobník pre prechádzanie stromu, čo je neefektívne vzhľadom k počtu načítaných stránok a prináša komplikácie pri viacnásobnom prístupe. Tento problém je možné efektívne riešiť vhodnou modifikáciou B-stromu.

$B^+$-strom je B-strom pre ktorý platí:

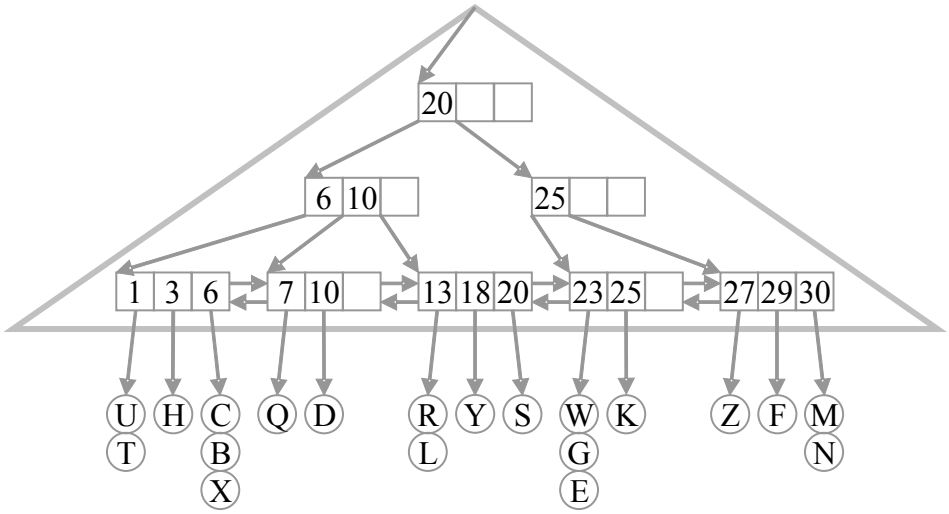
- uzly každej úrovne sú jednosmerne zreťazené,
- každý uzol obsahuje ukazovateľ na svojho pravého suseda.

V tejto práci sa používa varianta $B^+$-stromu pre ktorú platí:

- vzniká z redundantného B-stromu,
- uzly listovej úrovne sú obojsmerne zreťazené, každý listový uzol obsahuje ukazovatele na svojho ľavého a pravého suseda.

Nech sú v strome uložené objekty množiny $X$ a nech sú tieto objekty indexované podľa hodnôt $i$-tého atribútu $A_i$. Každý objekt $x \in X$ je indexovaný podľa kľúča $k \in dom(A_i)$, ktorého hodnota je rovná hodnote $i$-tého atribútu $a_i^x$ objektu $x$, $k = a_i^x$. Pre jeden takýto kľúč $k$ môže v množine $X$ existovať viacero objektov, s rovnakými hodnotami $i$-tého atribútu.

Preto sa pre účely tejto práce zavádza *pole objektov* ako nový typ uzlu v strome. V poli objektov sa uchovávajú objekty s rovnakou hodnotou kľúča $k$. V každom zázname ($k$, $d$) listového uzlu B$^+$-stromu je potom $d$ ukazovateľ na toto pole objektov. Pre každý kľúč $k \in dom(A_i)$ teda existuje v strome práve jedno pole objektov. Napr. na obrázku 5.2 pre kľúč 23 sú v príslušnom poli objektov objekty $x_W$, $x_G$ a $x_E$. V ďalšom texte práce bude B$^+$-stromom vždy myslená práve popísaná varianta B$^+$-stromu.




*Obrázok 5.2, B$^+$-strom s poliami objektov*

B$^+$-stromy je vhodné použiť pri častom použití rozsahových dotazov od $a$ po $b$. Stačí jediným priechodom stromu nájsť prvú hodnotu kľúča väčšiu alebo rovnú ako $a$. Potom sa priechodom cez listy a pomocou ukazovateľa na pravého suseda získavajú, pomocou ukazovateľov na polia objektov, objekty až pokiaľ je hodnota kľúča menšia alebo rovná ako $b$. Týmto spôsobom sa získajú objekty s hodnotami príslušného atribútu v rozmedzí od $a$ po $b$, ktoré sú navyše podľa týchto hodnôt atribútov zotriedené vzostupne. Pomocou ukazovateľa na ľavého suseda v listovej úrovni je možné podobným spôsobom získať aj dáta zotriedené zostupne.

*Príklad*: V strome na obrázku 3.2 sa vyhľadávajú objekty (na obrázku krúžky) s hodnotou od 18 do 28. Stačí jedným priechodom prechodom vnútornými uzlami stromu nájsť kľúč s hodnotou väčšou alebo rovnou 18. Potom sa postupuje „smerom doprava" až po hodnotu 27. Takto sa získa vzostupne zoradená postupnosť objektov $x_Y$, $x_S$, $x_W$, $x_G$, $x_E$, $x_K$, $x_Z$, prejdením minimálneho možného počtu uzlov.

## 5.3 VB-strom

B-stromy, tak ako boli zavedené vyššie neumožňujú indexovať viacrozmerné dáta. Nie je pomocou nich možné indexovať objekty podľa viacerých atribútov súčasne. Túto možnosť poskytujú *viacrozmerné B-stromy* [2] označované ako *VB-stromy*. Pomocou VB-stromov je možné indexovať objekty aj podľa viacerých atribútov súčasne v jednej dátovej štruktúre. Nech sú vo VB-strome uložené dáta o objektoch z množiny objektov $X$ s $m$ atribútmi $A_1$, …, $A_m$ a nech sú tieto dáta indexované podľa hodnôt všetkých atribútov $A_1$, …, $A_m$. Kľúčom sa v tomto prípade rozumie usporiadaná $m$-tica ($a_1$, …, $a_m$), kde $a_i \in dom(A_i)$.

VB-strom je stromová štruktúra, ktorej základ tvorí strom hĺbky $m$ (strom má $m$ *úrovní*), kde $m$ je počet indexovaných atribútov objektu. VB-strom spĺňuje nasledujúce podmienky.

1) Uzly VB-stromu sú neredundantné B-stromy také, že na $i$-tej úrovni zodpovedajú $i$-tému atribútu $A_i$ a majú rád $n_i$.
2) Uzly B-stromov majú štruktúru:

   $$p_0, (k_1, f_1, d_1), (k_2, f_2, d_2), ..., (k_n, f_n, d_n),$$

   kde ukazovateľ $f_j$ stromu v $i$-tej úrovni ukazuje na koreň stromu ($i$+1)-tej úrovne, ktorý obsahuje hodnoty z $dom(A_{i+1})$ také, že každá z nich sa vyskytuje s $k_j$ v jednom objekte. Pre danú hodnotu $k_j$ je určená pomocou ukazovateľa $f_j$ množina hodnôt z $dom(A_{i+1})$, ktorá sa nazýva *synovskou množinou*.
3) Na $i$-tej úrovni každý $f_j$ ukazuje na príslušný záznam, pre $i = m$, alebo ukazuje na koreň B-stromu, ktorý ďalej vedie k objektu, pre $i < m$.
4) Synovské množiny úrovne $i + 1$ sú zreťazené ukazovateľmi *NEXT* v koreňoch B-stromov tak, že ich poradie zodpovedá usporiadaniu kľúčov na úrovni $i$.
5) Koreň každého B-stromu na úrovni $i$ obsahuje ešte ukazovatele *LEFT* a *RIGHT*, ktoré ukazujú na najľavejšiu resp. na najpravejšiu synovskú množinu, ktoré zodpovedajú danému B-stromu na úrovni $i + 1$.
6) Pole $LEVEL(i)$, $i = 1, \ldots, n$, obsahuje ukazovatele na počiatočné B-stromy zreťazenia danej úrovne.

VB-stromy boli prvý krát publikované v práci [5] ako dátová štruktúra vhodná pre *viacrozmerné rozsahové dotazy*. Viacrozmerný rozsahový dotaz vo VB-strome efektívne vyhľadá pre interval $D = [u_1, v_1] \times \ldots \times [u_m, v_m]$ všetky objekty, ktoré majú kľúč $k = \{a_1,\ldots, a_m\}$ taký, že $k \in D$.

V prvej úrovni VB-stromu sa vyhľadajú kľúče z intervalu $[u_1, v_1]$ a ich synovské množiny z druhej úrovne. Vo všetkých týchto synovský množinách sa vyhľadajú kľúče z intervalu $[u_2, v_2]$ a ich synovské množiny. Postup sa analogicky opakuje až po poslednú úroveň VB-stromu.

V prípade viacrozmerného rozsahového dotazu pri ktorom nezáleží na hodnotách $i$-tého atribútu vyhľadávaných objektov nie je potrebné prehľadávať stromy v $i$-tej úrovni. Pre takýto strom $S$ sa rovno prechádza na strom v $(i + 1)$-tej úrovni pomocou ukazovateľa *LEFT* stromu $S$ a pomocou ukazovateľa NEXT sa prechádzajú stromy $(i + 1)$-tej úrovne až po strom na ktorý ukazuje ukazovateľ *RIGHT* stromu $S$. Keď nezáleží na hodnotách prvých $p$ atribútov, použije sa ukazovateľ $LEVEL(p + 1)$ a prejde sa na prvý strom $(p + 1)$-tej úrovne a pomocou ukazovateľov NEXT sa prejde celá táto úroveň až po jej posledný strom.

Pre rovnakú množinu objektov s $m$ atribútmi existuje viacero možností vytvorenia VB-stromu, ktoré sa odlišujú poradím atribútov v jednotlivých úrovniach. Voľba poradia atribútov záleží na konkrétnom použití VB-stromu.

*Príklad*: Nech $X$ je množina bytov, s atribútmi Cena, Mesto a Rozloha. Keď sa bude predpokladať, že najčastejším dotazom bude vyhľadanie bytov v jednom konkrétnom meste $M$ s vhodnou cenou a rozlohou, je v tomto prípade vhodné VB-strom vytvoriť

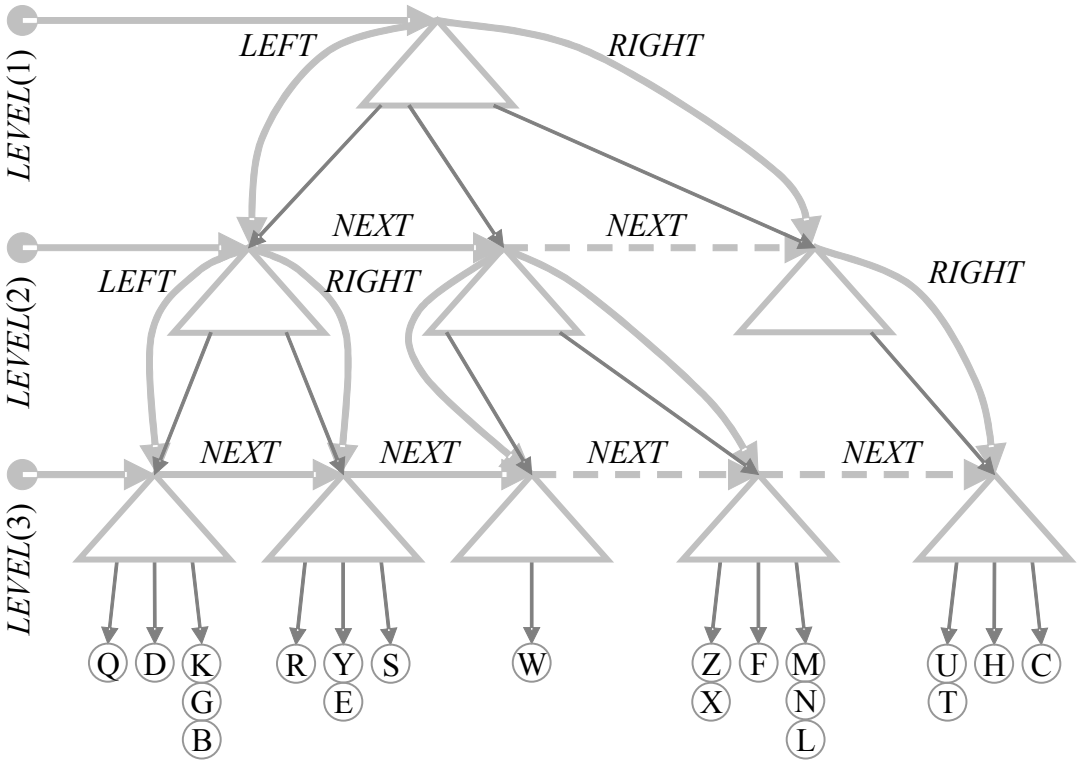
tak, aby bol atribút Mesto v jeho prvej úrovni. Potom sa pri predpokladanom dotaze v prvej úrovni vždy vyhľadá synovská množina mesta *M*, ktorá sa potom prehľadáva ďalej. Synovské množiny ostatných miest sa už neprehľadávajú. Pri dotaze sa tak prehľadá len malá časť stromu.

V tejto práci sa používa varianta VB-stromu (obrázok 5.3), ktorej základom sú redundantné $B^+$-stromy. Analogicky ako pri $B^+$-stromoch sa zavádza *pole objektov* ako nový typ uzlu vo VB-strome. V poli objektov sa uchovávajú objekty s rovnakou hodnotou kľúča $\{a_1, \ldots, a_m\}$.

Listové uzly týchto stromov *i*-tej úrovne majú tvar

$$(k_1, f_1), (k_2, f_2), \ldots, (k_n, f_n), u$$

kde $k_1, \ldots, k_n$ sú kľúče (hodnoty z domény *i*-tého atribútu) a $f_1, \ldots, f_n$ sú ukazovatele, ktoré ukazujú na príslušné $B^+$-stromy v nasledujúcej úrovni pre $i < m$, alebo v prípade poslednej úrovne ($i = m$) ukazujú na priamo na príslušné pole objektov. V ďalšom texte bude VB-stromom vždy myslená práve popísaná varianta.




*Obrázok 5.3, VB-strom*

# 6 Model zoznamu založený na B$^+$-strome

Faginov algoritmus potrebuje počas svojho výpočtu sekvenčne pristupovať do zoznamov (viz kapitola 4.1.3). Preto postačí model zoznamov $L_1, \ldots, L_m$, pri ktorom je možné z $i$-tého zoznamu $L_i$ získavať sekvenčne objekty $x$, dvojice $\{x, f_i^U(x)\}$. Nie je preto potrebné získať celý zoznam pred spustením algoritmu. Stačí aby bolo možné pri prvom prístupe do zoznamu $L_i$ vrátiť objekt $x_1$ s najvyššou hodnotou $f_i^U(x_1)$, pri druhom prístupe objekt $x_2$ s druhou najvyššou hodnotou $f_i^U(x_2)$ a podobne ďalšie objekty $x \in X$, až po koniec zoznamu.

Prvá požadovaná vlastnosť nového modelu zoznamov je uloženie objektov spolu s hodnotami $i$-tého atribútu v takej dátovej štruktúre, ktorá by umožňovala dobrý sekvenčný prístup. Druhá požadovaná vlastnosť modelu je podpora lokálnych užívateľských preferencií. V tejto práci sa preto bude vychádzať z modelu zoznamu založenom na B$^+$-strome [3].

Kvôli prehľadnosti sa bude v tejto kapitole predpokladá, že $X$ je množina objektov s jedným atribútom $A$, každý objekt $x \in X$ má hodnotu tohto atribútu $a^x$. Všetky objekty $x \in X$ sú ako dvojice $\{x, a^x\}$ uložené v B$^+$-strome (viz kapitola 5.2) a indexované podľa hodnôt $a^x$. Ďalej sa bude predpokladať, že v B$^+$-strome sú uložené hodnoty atribútov normalizované na interval [0, 1] (viz kapitola 2.2.3) a lokálnu preferenciu užívateľa $U$ vyjadruje lineárne reprezentovaná fuzzy funkcia $f^U$ (viz kapitola 2.2.4), ktorá má potom tvar $f^U : [0, 1] \rightarrow [0, 1]$. Keďže $X$ je množina objektov s jedným atribútom, Faginov algoritmus v takomto prípade potrebuje pre svoj výpočet sekvenčne pristupovať iba do jedného zoznamu $L$.

*Poznámka:* V takomto prípade nie je potrebné použiť Faginov algoritmus, pretože lokálne ohodnotenie objektu je zároveň globálnym ohodnotením objektu. Viacero lokálnych ohodnotení sa nemusí agregovať do jedného globálneho ohodnotena. Ak sú v zozname $L$ objekty zostupne zotriedené podľa $f^U$, potom prvých $K$ objektov zoznamu $L$ je zároveň $K$ najlepších objektov pre užívateľa $U$.

## 6.1 Získavanie dvojíc z B$^+$-stromu

V B$^+$-strome sú ako kľúče použité hodnoty atribútu $a^x$ objektov $x \in X$. Z listovej úrovne stromu, ktorá obsahuje všetky kľúče, sa odkazuje na polia objektov, kde každé z nich obsahuje objekty s rovnakými hodnotami atribútu. Keďže listové uzly B$^+$-stromu sú zreťazené, je možné strom prejsť po listovej úrovni cez všetky kľúče. Kľúče sú v listovej úrovni vzostupne usporiadané. Pomocou odkazov na polia objektov je teda možné získať postupnosť objektov zotriedenú podľa hodnoty atribútu. Ak sa listovou úrovňou prechádza zľava doprava, získa sa vzostupne zotriedená postupnosť objektov, ak sprava doľava, tak zostupne zotriedená postupnosť.

V zmysle získavania dvojíc $\{x, a^x\}$ počas výpočtu Faginovho algoritmu je možné priechodom listovej úrovne B$^+$-stromu sprava doľava (zľava doprava) sekvenčne získavať dvojice $\{x, a^x\}$ zoznamu $L$ v zostupnom (vzostupnom) poradí podľa hodnoty atribútu $a^x$ objektov $x \in X$. Takto získané dvojice zoznamu $L$ ale zatiaľ nie sú ohodnotené podľa lokálnej užívateľskej preferencie $f^U$, obsahujú iba hodnotu atribútu objektu. Kvôli aplikácií užívateľskej preferencie $f^U$ vo Faginovom algoritme je však potrebné počas jeho výpočtu získavať dvojice $\{x, f^U(x)\}$.

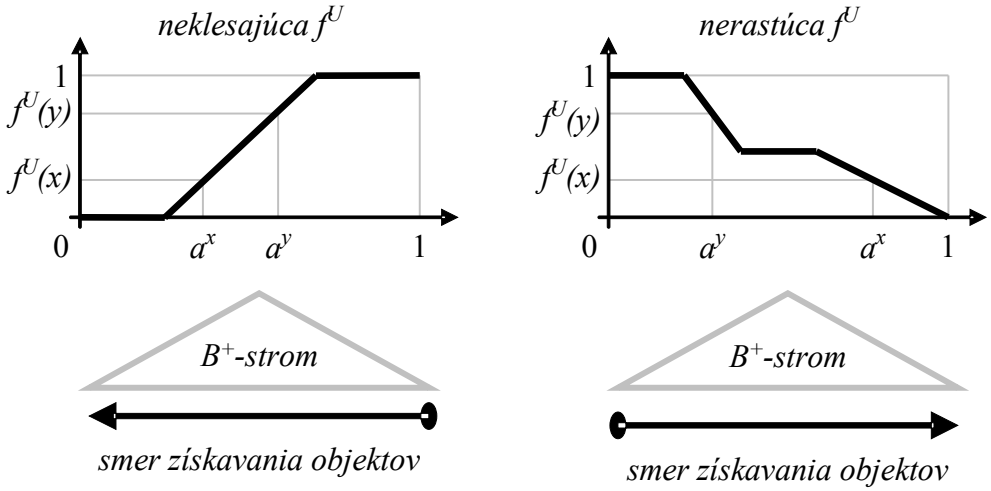
### 6.1.1 Monotónna fuzzy funkcia

Podľa usporiadavacej interpretácie fuzzy funkcie (viz kapitola 2.2.2), je možné užívateľskú fuzzy funkciu chápať ako nové usporiadanie objektov podľa hodnôt atribútov. Na konkrétnych hodnotách atribútu $a^x$ objektov $x \in X$ nezáleží, na zotriedenie objektov sa použije tvar (priebeh) fuzzy funkcie $f^U$.

Ak je fuzzy funkcia $f^U$ na celom intervale [0, 1] (na celom jej definičnom obore) monotónna, potom pre získavanie dvojíc z $B^+$-stromu s použitím užívateľskej preferencie platí nasledujúce:

1) Nech je užívateľská fuzzy funkcia $f^U$ neklesajúca, pre dva objekty $x, y \in X$ platí $a^x \leq a^y \Rightarrow f^U(x) \leq f^U(y)$. Potom priechodom listovej úrovne $B^+$-stromu sprava doľava je možné získavať dvojice $\{x, f^U(x)\}$ zoznamu $L$ v zostupnom poradí podľa užívateľskej preferencie $f^U$.
2) Nech je užívateľská fuzzy funkcia $f^U$ nerastúca, pre dva objekty $x, y \in X$ platí $a^x \geq a^y \Rightarrow f^U(x) \leq f^U(y)$. Potom priechodom listovej úrovne $B^+$-stromu zľava doprava je možné získavať dvojice $\{x, f^U(x)\}$ zoznamu $L$ v zostupnom poradí podľa užívateľskej preferencie $f^U$.

Pri získavaní objektu $x$ zo stromu sa získa dvojica $\{x, a^x\}$, následne sa vypočíta $f^U(x)$ a vzniká dvojica $\{x, f^U(x)\}$. Takto získavané dvojice zoznamu $L$ sú už ohodnotené podľa užívateľskej fuzzy funkcie $f^U$, obsahujú lokálnu užívateľskú preferenciu.




*Obrázok 6.1.1, Získavanie objektov v $B^+$-strome*

### 6.1.2 Cesty

Pre získavanie objektov z $B^+$-stromu sa zavádza pojem *cesta*. Cesta je orientovaná úsečka, ktorá určuje postupovanie listovou úrovňou $B^+$-stromu pri získavaní objektov. Každá cesta $w$ má svoj začiatok $w^{beg}$, koniec $w^{end}$ a smer, ktorý je doľava alebo doprava.

Na obrázku 6.1.1 sú znázornené fuzzy funkcie. Pod grafmi funkcií sú znázornené $B^+$-stromy pod ktorými sú znázornené smery získavania objektov ako orientované úsečky. Tieto úsečky sú cesty.

Pre prípad cesty smerujúcej doprava (doľava) sa získa prvý objekt tak, že sa prechodom vnútornými uzlami $B^+$-stromu nájde v listovej úrovni stromu $B^+$-stromu prvý kľúč $k \geq w^{beg}$ ($k \leq w^{beg}$) a z príslušného poľa objektov sa vyberie prvý objekt. Hodnota atribútu $a^x$ získaného objektu $x$ je rovná hodnote kľúča $k$.

Ďalší objekt sa podľa cesty $w$ získa ako ďalší objekt toho istého poľa objektov (objekt s rovnakou hodnotou atribútu). Pokiaľ už v tomto poli objektov nie je žiaden ďalší objekt, tak sa v listovej úrovni prejde na pole objektov pravého susedného kľúča a objekt sa vyberie odtiaľ.

Ďalší objekt sa podľa cesty $w$ už nedá získať ak už v poli objektov pre aktuálny kľúč $k$ nie je ďalší objekt a nasledujúci pravý (ľavý) sused aktuálneho kľúča buď neexistuje (koniec listovej úrovne stromu) alebo nasledujúci kľúč je mimo cesty $w$ (hodnota kľúča je väčšia ako $w^{end}$).

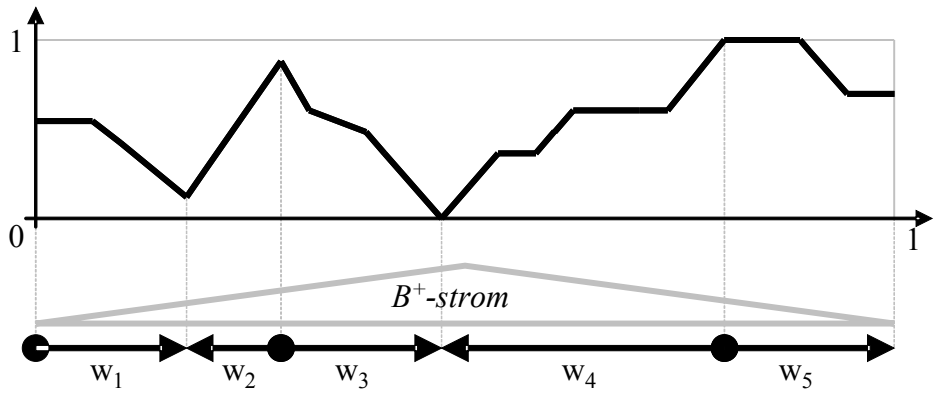
Pomocou cesty $w$ je takýmto postupom možné z $B^+$-stromu získavať objekty $x \in X$ aj s ich hodnotami atribútu $a^x$. Ak sa pre takto získaný objekt $x$ vypočíta lokálne ohodnotenie $f^U(x)$, vzniká dvojica $\{x, f^U(x)\}$.

Ak je užívateľská fuzzy funkcia $f^U$ na celom intervale [0, 1] neklesajúca, pre získavanie objektov sa použije cesta $w$, ktorá má začiatok $w^{beg} = 1$, koniec $w^{end} = 0$ a smer doľava. Ak je $f^U$ na celom intervale [0, 1] nerastúca, použije sa cesta $w$, ktorá má začiatok $w^{beg} = 0$, koniec $w^{end} = 1$ a smer doprava. V obidvoch prípadoch sa tak získajú dvojice $\{x, f^U(x)\}$ zoznamu $L$ v zostupnom poradí podľa $f^U$ (viz obrázok 6.1.1).

### 6.1.3 Všeobecná fuzzy funkcia

Vo všeobecnosti nie je užívateľská fuzzy funkcia $f^U$ monotónna na celom intervale [0, 1]. Pre získavanie dvojíc zoznamu $L$ podľa užívateľskej preferencie $f^U$ sa nedá použiť jedna cesta cez celý interval [0, 1]. Interval [0, 1], ako definičný obor fuzzy funkcie $f^U$ je však možné rozdeliť na postupnosť $n$ intervalov $I_1, \ldots, I_n$ tak, že na každom z týchto intervalov je fuzzy funkcia $f^U$ monotónna, teda neklesajúca alebo nerastúca.

Podľa intervalov $I_1, \ldots, I_n$ je možné vytvoriť cesty $w_1, \ldots, w_n$ tak, že každá cesta $w_i$ má ako začiatok a koniec hraničné body intervalu $I_i$. Smer cesty $w_i$ je doľava, keď je fuzzy funkcia $f^U$ na intervale $I_i$ neklesajúca, alebo doprava, keď je $f^U$ na intervale $I_i$ nerastúca. Cesty $w_1, \ldots, w_n$ potom určujú ako sa bude prechádzať $B^+$-stromom pri získavaní objektov podľa užívateľskej preferencie $f^U$ (viz obrázok 6.1.3).




*Obrázok 6.1.3, Vytvorenie ciest podľa tvaru všeobecnej fuzzy funkcie*

Keď existuje viacero ciest $w_1, \ldots, w_n$, na každej z nich môžu existovať objekty $x \in X$ s rôznymi hodnotami $f^U(x) \in [0, 1]$. Objekty z jednotlivých ciest je potrebné získavať v jednom celkovom zostupnom poradí podľa $f^U(x)$, v tvare dvojíc $\{x, f^U(x)\}$ zoznamu $L$. Pri získavaní dvojíc $\{x, f^U(x)\}$ zo B$^+$-stromu podľa viacerých ciest je preto nutné všetkými cestami postupovať paralelne.

## 6.2 Formálny popis modelu

Popis získavania dvojíc z B$^+$-stromu:

I. Pred získavaním dvojíc zoznamu $L$.

   1. Pre B$^+$-strom sa vytvoria cesty $w_1, \ldots, w_n$ podľa užívateľskej fuzzy funkcie $f^U$ a označia sa ako relevantné.
   2. Na všetkých cestách $w_1, \ldots, w_n$ sa skúsia získať objekty $x_1, \ldots, x_n$. Cesty na ktorých sa nepodarilo získať objekty sa označia ako irelevantné. Získané objekty sa označia ako množina aktuálnych objektov. Prepočíta sa $f^U(x)$ všetkých aktuálnych objektov.

II. Získavanie dvojíc zoznamu $L$.

   1. Z aktuálnych objektov sa vyberie objekt $x_j$ s najväčšou hodnotou $f^U(x_j)$. Vytvorí sa dvojica $\{x_j, f^U(x_j)\}$ zoznamu $L$. Objekt $x_j$ sa vyhodí z množiny aktuálnych objektov.
   2. Ak je to možné, na ceste $w_j$ sa získa ďalší objekt $x_j'$ a ten sa vloží do aktuálnych objektov a prepočíta sa jeho $f^U(x_j')$. Ak objekt nie je možné získať, cesta sa $w_j$ sa označí ako irelevantná a pri ďalšom vyhľadávaní objektov sa už nepoužije.
   3. Ak už neexistuje žiadna relevantná cesta, nie je možné vrátiť dvojicu zoznamu L. V opačnom prípade sa vráti dvojica $\{x_j, f^U(x_j)\}$ z 1. kroku.

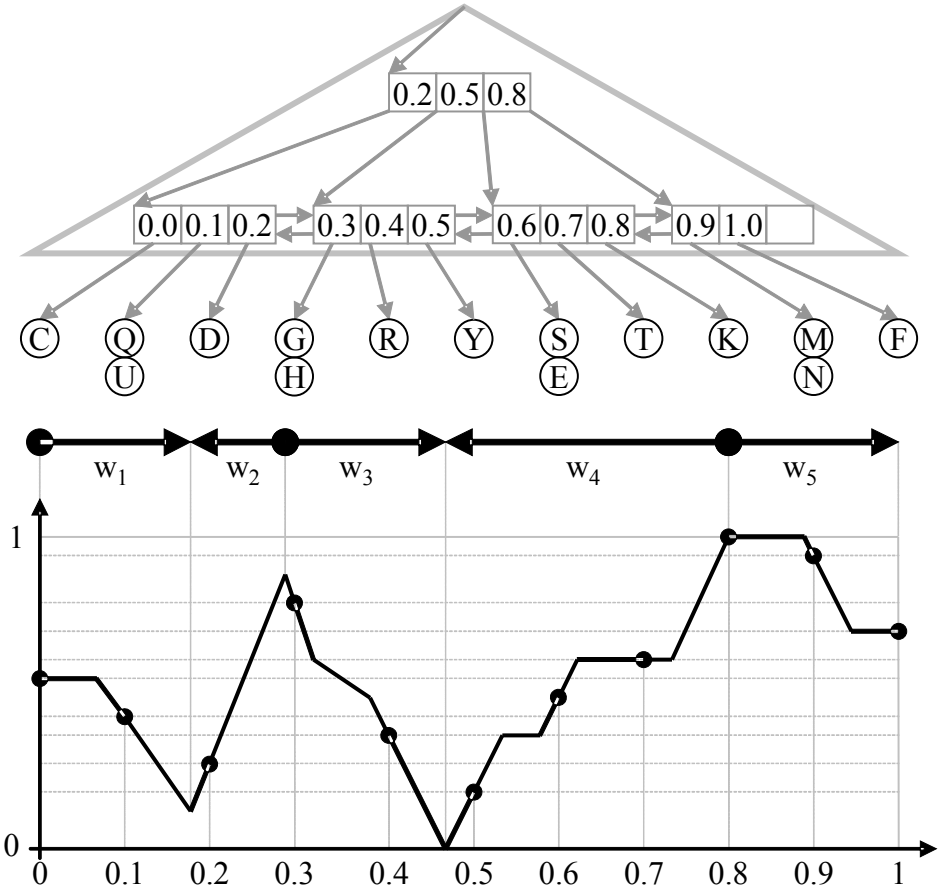
Keď sa z B$^+$-stromu už vrátil celý zoznam $L$, získali sa dvojice pre všetky objekty v B$^+$-strome, zostanú všetky cesty irelevantné a množina aktuálnych objektov prázdna.

Pri použití zoznamu $L$ bez aplikácie užívateľskej preferencie je potrebné B$^+$-stromom z jeho koreňa prejsť na prvú hodnotu v strome a sekvenčným priechodom cez listovú úroveň sa získajú všetky objekty stromu. Pri použití tohto modelu sa na začiatku získavania dvojíc v B$^+$-strome musí z koreňa stromu postupne prejsť na začiatky všetkých ciest a ďalšími operáciami sú už len sekvenčné prechádzanie listovej úrovne podľa ciest. Teda pri hľadaní prvého objektu môže byť fáza I vzhľadom na počet prístupov do B$^+$-stromu o niečo náročnejšia (v závislosti na počte ciest) ako prípad bez požitia užívateľskej preferencie.

**Príklad 4, Ukážka funkčnosti modelu**

Na obrázku 6.2 je znázornený B$^+$-strom, v ktorom sú uložené objekty podľa hodnoty atribútu. Hodnoty atribútov objektov $x \in X$ sú normalizované na interval [0, 1]. Pre názornosť sú v strome viaceré objekty s rovnakou hodnotu atribútu, ktoré tvoria viacprvkové polia objektov. Pod B$^+$-stromom je znázornená užívateľská fuzzy funkcia $f^U$, pomocou ktorej je vyjadrené, akým spôsobom užívateľ $U$ preferuje objekty podľa hodnoty atribútu. Obrázok je zámerne vytvorený tak, aby objekty v strome boli

vertikálne zarovnané podľa hodnôt atribútov vzhľadom ku bodom na x-ovej osi fuzzy funkcie. Na grafe fuzzy funkcie sú taktiež zvýraznené body, ktoré reprezentujú všetky objekty $x \in X$ zo stromu a vznikli ako funkčné hodnoty $f^U(x)$ objektov.




*Obrázok 6.2, Získavanie dvojíc zoznamu L v $B^+$-strome podľa $f_i^U$*

Podľa kroku 1 vo fáze I formálneho popisu získavania dvojíc zoznamu z $B^+$-stromu sa na začiatku vytvoria cesty $w_1$, …, $w_5$ poľa monotónnych častí fuzzy funkcie $f^U$. Keď je na hranici dvoch ciest časť fuzzy funkcie tvorená konštantnou lineárnou funkciou, treba rozhodnúť do ktorej cesty sa táto časť začlení. Taktiež v hraničných bodoch ciest musí byť jasné, ktorej ceste tento bod patrí. Riešenie týchto otázok záleží na konkrétnej implementácii. Na obrázku je znázornená varianta, podľa ktorej hraničný bod a aj prípadná lineárna časť sú pričlenené k pravej ceste. Podľa kroku 2 vo fáze I sa na cestách $w_1$, …, $w_5$ získajú porade objekty $x_C$, $x_D$, $x_G$, $x_T$, $x_K$ a vypočíta sa ich $f^U$.

Pri prvom získavaní dvojice zoznamu $L$ podľa obrázku, je najvyššou hodnotu $f^U(x_K)$ objektu $x_K$. Vytvorí sa dvojica zoznamu $\{x_K, f^U(x_K)\}$. Objekt $x_K$ sa vyhodí z množiny aktuálnych objektov. Objekt $x_K$ bol nájdený na ceste $w_5$, preto sa na ceste $w_5$ získa nový objekt $x_M$, ktorý sa vloží do množiny aktuálnych objektov. Nakoniec sa vráti $\{x_K, f^U(x_K)\}$ ako prvá dvojica zoznamu $L$.

Podobným postupom sa vráti $\{x_M, f^U(x_M)\}$ ako druhá dvojica zoznamu $L$ a do množiny aktuálnych objektov sa vloží objekt $x_N$, ktorý sa nachádza v tom istom poli objektov

ako $x_M$. Z hľadiska zostupnej zotriedenosti zoznamu $L$ podľa $f^U$ nezáleží na poradí objektov s rovnakými ohodnotením $f^U(x)$, teda nezáleží či sa vráti skôr $x_M$ alebo $x_N$.

Ďalšie tri vrátené objekty budú $x_N$, $x_G$ a $x_N$. V množine aktuálnych objektov potom zostanú objekty $x_C$, $x_D$, $x_R$, $x_T$, $x_F$. Pri nasledujúcom získavaní dvojice sa vráti $\{x_F, f^U(x_F)\}$. Pritom na ceste $w_5$ už nie je možné získať ďalší objekt, cesta $w_5$ sa preto označí ako irelevantná a pri ďalšom vyhľadávaní objektov sa už nepoužije. Dôsledkom toho sa množina aktuálnych objektov zmenší o jeden prvok. Analogicky sa postupuje pre získavanie ďalších dvojíc zoznamu $L$, až pokiaľ sa nezískajú dvojice pre všetky objekty v B$^+$-strome.

*Pozorovanie:* Nech $p$ je priamka rovnobežná s x-ovou osou grafu funkcie $f^U$. Na začiatku sa priamka položí tak aby prechádzala bodom 1 na y-ovej osi. Priamkou $p$ sa bude pohybovať smerom ku x-ovej ose až s ňou nakoniec splynie. Pri tomto pohybe bude priamka $p$ postupne pretínať zvýraznené body na grafe fuzzy funkcie, ktoré reprezentujú objekty zo stromu. Poradie v ktorom sú body objektov preťaté priamkou $p$ je potom totožné z poradím objektov získaných v tvare dvojíc zoznamu $L$ podľa modelu získavania dvojíc z B$^+$-stromu.

## 6.3 Použitie vo Faginovom algoritme

V kapitole 4 sa problém top-k rieši pomocou Faginovho algoritmu, ktorý nájde $K$ najlepších objektov $x \in X$ pomocou menšieho počtu prístupov ako je $N \cdot m$, kde $N$ je mohutnosť množiny objektov $X$ a $m$ je počet ohodnocovaných atribútov. Pôvodný Faginov algoritmus dokáže nájsť $K$ najlepších objektov len vzhľadom na agregačnú funkciu. Úlohou tejto práce je však implementovať Faginov algoritmus tak, aby podporoval viacužívateľský prístup založený na modele užívateľských preferencií zavedenom v kapitole 2.4.

Podľa práve zavedeného modelu zoznamov je možné implementovať Faginov algoritmus tak, že bude používať ako zoznamy $L_1, \ldots, L_m$ pole $m$ B$^+$-stromov. V každom $i$-tom B$^+$-stromov sú potom indexované objekty $x \in X$ podľa (normalizovaných) hodnôt jedného z $m$ atribútov. Pri výpočte Faginovho algoritmu sa potom z $i$-tého B$^+$-stromu získavajú dvojice $\{x, f_i^U(x)\}$ v zostupnom poradí podľa $i$-tej užívateľskej fuzzy funkcie $f_i^U$.

Pri použití poľa $m$ B$^+$-stromov dokáže Faginov algoritmus nájsť $K$ najlepších objektov pre užívateľa vzhľadom na jeho preferencie. Tieto B$^+$-stromy sú pritom spoločné pre všetkých užívateľov. Uloženie objektov s hodnotami ich $m$ atribútov v poli $m$ B$^+$-stromov je nezávislé na užívateľských preferenciách.

Konkrétny užívateľ $U$ už len zadá svoje lokálne preferencie pomocou užívateľských fuzzy funkcií $f_1^U, \ldots, f_m^U$, globálne preferencie pomocou užívateľskej agregačnej funkcie $@^U$. Faginov algoritmus potom vyhľadá $K$ najlepších objektov vzhľadom na preferencie užívateľa $U$. Ďalší užívateľ zadá iné lokálne preferencie a Faginov algoritmus znovu nájde $K$ najlepších objektov vzhľadom na preferencie tohto užívateľa.

Použitím B$^+$-stromov je možné vyriešiť doposiaľ hlavný problém aplikácie lokálnych preferencií vo Faginovom algoritme. Už nie je potrebné pred jeho výpočtom vytvárať celé zoznamy $L_1, \ldots, L_m$ podľa lokálnych preferencií každého užívateľa (znamenalo by to vykonať $N \cdot m$ prístupov a nemalo by v tom prípade zmysel použiť Faginov algoritmus, viz kapitola 4.4).

### 6.3.1 NRA algoritmus

Algoritmus NRA potrebuje počas svojho výpočtu sekvenčne pristupovať do zoznamov $L_1, \ldots, L_m$, ktoré obsahujú zostupne zotriedené dvojice objekt s jeho ohodnotením podľa príslušnej lokálnej užívateľskej preferencie. Pre sekvenčné získavanie dvojíc z každého zoznamu použije model zoznamu založený na $B^+$-strome.

Pre potreby NRA algoritmu je potrebné pred jeho spustením vytvoriť pole $m$ $B^+$-stromov, kde v každom $i$-tom $B^+$-strome sú indexované objekty (identifikátory objektov) $x \in X$ podľa (normalizovanej) hodnoty $i$-tého atribútu.

### 6.3.2 TA algoritmus

Situácia pre algoritmus TA je obdobná ako v prípade algoritmu NRA. Algoritmus TA potrebuje počas svojho výpočtu sekvenčne pristupovať do zoznamov $L_1, \ldots, L_m$. Preto sa pred jeho spustením taktiež vytvorí $m$ $B^+$-stromov, v ktorých budú indexované objekty $x \in X$ podľa príslušných hodnôt atribútov.

Algoritmus TA navyše využíva aj priamy prístup do zoznamov $L_1, \ldots, L_m$, pri ktorom je potrebné podľa objektu (identifikátoru objektu) získať z $i$-tého zoznamu priamo hodnotu $i$-tého atribútu objektu. Ako zoznamy sa však používajú $B^+$-stromy, ktoré túto operáciu neumožňujú. V $B^+$-strome nie je možné podľa objektu priamo získať jeho hodnotu atribútu.

Pre potreby TA algoritmu sa vytvorí okrem poľa $m$ $B^+$-stromov ďalšia dátová štruktúra, ktorá umožní podľa objektu (identifikátoru objektu) získať požadované hodnoty jeho atribútov. Počas výpočtu algoritmu sa teda realizuje priamy prístup do zoznamov $L_1, \ldots, L_m$ pomocou tejto ďalšej dátovej štruktúry (napr. asociatívne pole) a sekvenčný prístup sa realizuje v poli $m$ $B^+$-stromov. Tieto dve dátové štruktúry sú pritom nezávislé (viz kapitola 4.2).

# 7 VB algoritmus

V každej dátovej štruktúre, ktorá obsahuje potrebné dáta o $N$ objektoch (hodnoty $m$ atribútov), je možné vyhľadať $K$ najlepších objektov podľa užívateľských preferencií. Najjednoduchším riešením je načítať z dátovej štruktúry všetky hodnoty $m$ atribútov všetkých $N$ objektov, vypočítať ohodnotenia všetkých objektov a vrátiť $K$ objektov s najvyšším ohodnotením. Pritom sa však musia prejsť všetky objekty a je potrebné vykonať $N \cdot m$ prístupov.

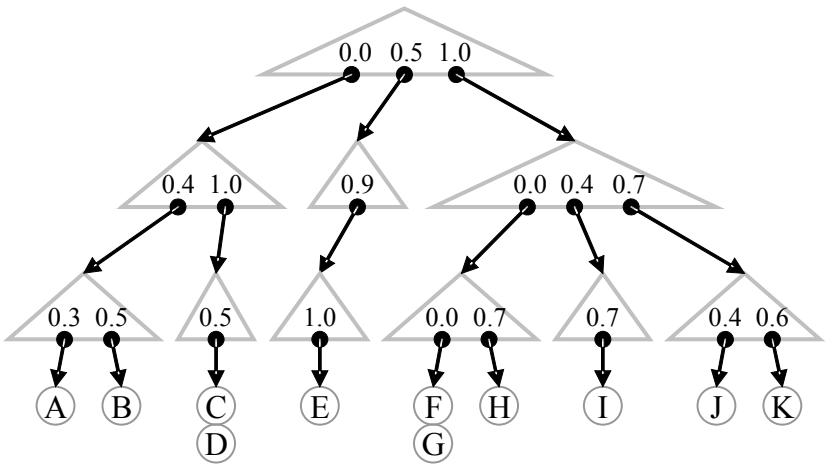
Táto kapitola sa zaoberá použiteľnosťou VB-stromu na riešenie problému top-k pomocou menšieho počtu prístupov ako je $N \cdot m$. Práca prináša VB algoritmus, ktorý dokáže vyhľadať $K$ najlepších objektov bez prejdenia všetkých objektov vo VB-strome, v ktorom sú indexované objekty podľa $m$ (normalizovaných) hodnôt atribútov.

## 7.1 Vlastnosti VB-stromu

VB-stromy (viz kapitola 5.3) umožňujú indexovať viacrozmerné dáta. Pomocou nich je možné indexovať množinu objektov $X$ aj podľa $m > 1$ atribútov $A_1, \ldots, A_m$ súčasne v jednej dátovej štruktúre. VB-strom má potom $m$ úrovní. V každej $i$-tej úrovni sa nachádzajú B$^+$-stromy, v ktorých sú kľúčmi hodnoty z $dom(A_i)$. Pre potreby tejto práce sú hodnoty atribútov objektov normalizované na interval [0, 1].

Na obrázku 7.1 je v tabuľke jedenásť objektov $x_A, \ldots, x_K$ s hodnotami troch atribútov $A_1, A_2, A_3$. Vedľa tabuľky je znázornený VB-strom v ktorom sú uložené tie isté objekty.

| | $A_1$ | $A_2$ | $A_3$ |
|---|---|---|---|
| A | 0.0 | 0.4 | 0.3 |
| B | 0.0 | 0.4 | 0.5 |
| C | 0.0 | 1.0 | 0.5 |
| D | 0.0 | 1.0 | 0.5 |
| E | 0.5 | 0.9 | 1.0 |
| F | 1.0 | 0.0 | 0.0 |
| G | 1.0 | 0.0 | 0.7 |
| H | 1.0 | 0.0 | 0.7 |
| I | 1.0 | 0.4 | 0.7 |
| J | 1.0 | 0.7 | 0.4 |
| K | 1.0 | 0.7 | 0.6 |




*Obrázok 7.1, Uloženie objektov vo VB-strome*

Je zrejmé, že v prvej úrovni VB-sromu sa vždy nachádza iba jeden B$^+$-strom. Jeho kľúčmi sú všetky hodnoty z $dom(A_1)$. V ďalších úrovniach VB-stromu sa môže vyskytovať viacero B$^+$-stromov. Jednotlivé B$^+$-stromy v úrovniach $i > 1$ vo všeobecnosti neobsahujú ako kľúče všetky hodnoty z $dom(A_i)$ a v krajnom prípade môžu obsahovať iba jeden kľúč. Podľa definície VB-stromu obsahuje B$^+$-strom úrovne $i > 1$ ako kľúče také hodnoty z $dom(A_i)$, pre ktoré existujú objekty s rozdielnymi hodnotami atribútu $A_i$ a s rovnakými hodnotami atribútov $A_1, \ldots, A_{i-1}$. Konkrétny B$^+$-strom $i$-tej úrovne je preto jednoznačne určený kľúčmi $k_1, \ldots, k_{i-1}$, kde $k_j \in dom(A_j)$ pre $j = 1, \ldots, i - 1$.

V tejto práci sa pre jednoznačné určenie $B^+$-stromu vo VB-strome používa postupnosť kľúčov, ktorá sa bude označovať ako *identifikátor stromu*. Identifikátor $B^+$-stromu v *i*-tej úrovni sa označí $(k_1, \ldots, k_{i-1})$. Identifikátor stromu v prvej úrovne sa označí ako (Ø).

*Príklad*: Na obrázku 7.1 je postupnosť kľúčov $(k_1, k_2)$= (1.0, 0.0) identifikátorom stromu v tretej úrovni, ktorý obsahuje kľúče 0.0, 0.7 a ďalej ukazuje na objekty $x_F$, $x_G$, $x_H$. Identifikátor (0.0) zase určuje strom v druhej úrovni, ktorý obsahuje kľúče 0.4, 1.0.

## 7.2 Hľadanie K najlepších objektov

Najjednoduchší postup ako nájsť vo VB-strome *K* objektov $x \in X$ s najvyšším ohodnotením podľa agregačnej funkcie @(*x*) je získať z VB-stromu všetky objekty aj s hodnotami ich atribútov. Potom stačí pre všetky objekty $x \in X$ vypočítať ohodnotenia @(*x*) a vybrať *K* objektov s najvyššími ohodnoteniami (analogicky ako v základnom algoritme viz kapitola 3.1).

### 7.2.1 Rekurzívna procedúra

Získanie všetkých objektov zo stromu je možné realizovať pomocou *rekurzívnej procedúry*. Táto procedúra rekurzívne prehľadáva VB-strom do hĺbky. Pritom prejde všetky $B^+$-stromy a všetky polia objektov vo VB-strome, čiže prejde všetky objekty.

Rekurzívna procedúra `searchVB` sa spustí vždy na jednom konkrétnom $B^+$-strome. Ak tento $B^+$-strom pomocou kľúčov odkazuje na $B^+$-stromy v ďalšej úrovni, procedúra `searchVB` sa spustí aj na tieto $B^+$-stromy. Ak tento $B^+$-strom pomocou kľúčov odkazuje na polia objektov, získajú sa z nich objekty. Pre jednoznačné určenie $B^+$-stromu, v ktorom sa práve vykonáva `searchVB` procedúra, sa bude používať identifikátor stromu. Získané objekty sa postupne pridávajú do zoznamu objektov *T*.

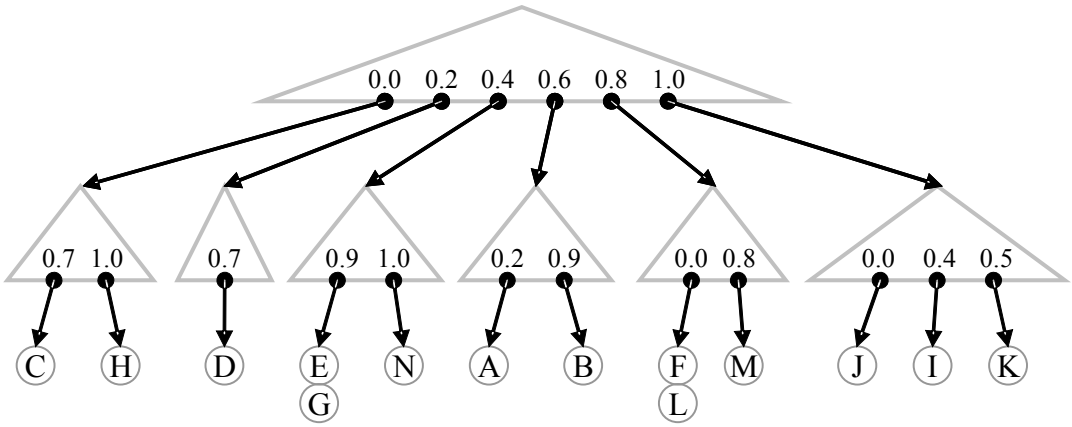
Popis rekurzívneho načítania všetkých objektov s hodnotami ich atribútov:

```
Input: VBstrom VB-stom;
Output: Zoznam T;
var Zoznam T;
begin
   searchMDBtree( VB-stom, (Ø) );
   return T;
end.

procedure searchVB(VBstrom VB-stom, Identikator (k1, …, ki-1));
   while( existuje ďalší kľúč v B+-strome s identifikátorom (k1, …, ki-1) ) do
      V zadanom B+-strome i-tej úrovne s identifikátorom (k1, …, ki-1) sa vyberie
      (ďalší) kľúč ki. s (ďalšou) najväčšou hodnotou i-tého atribútu v B+-strome.

      if( ukazovateľ kľúča ki odkazuje na B+-strom )          {úroveň i < m}
         searchMDtree( VB-stom, (k1, …, ki-1, ki) ) ;

      if( ukazovateľ kľúča ki odkazuje na pole objektov )     {úroveň i = m}
         objekty tohto poľa objektov sa pridajú do zoznamu T
         aj s hodnotami ich atribútov k1, …, ki-1, ki ;
   endwhile;
end.
```




*Obrázok 7.2.1, VB-strom s dvoma úrovňami*

**Príklad 5, Získanie všetkých objektov**

Vo VB-strome na obrázku 7.2.1 sú indexované objekty $x \in X$ podľa dvoch hodnôt atribútov $A_1$, $A_2$. V prvej úrovni VB-stromu sa nachádza jeden B$^+$-strom, v ktorom sú kľúčmi hodnoty z $dom(A_1)$ a pomocou nich je odkazované na B$^+$-stromy v druhej úrovni. V B$^+$-stromoch druhej úrovne sú kľúčmi hodnoty z $dom(A_2)$, pomocou ktorých je odkazované na polia objektov. Použitím procedúry `searchVB` vo VB-strome s dvoma úrovňami na obrázku 7.2.1 vznikne v zozname $T$ čiastočne zotriedená postupnosť objektov $x_K$, $x_I$, $x_J$, $x_M$, $x_F$, $x_L$, $x_B$, $x_A$, $x_N$, $x_E$, $x_G$, $x_D$, $x_H$, $x_C$.

Po samotnom vykonaní procedúry `searchVB` však nemusia byť objekty v zozname $T$ zostupne zotriedené vzhľadom na ohodnotenie pomocou agregačnej funkcie @. Objekty sú v zozname $T$ sú zostupne zoradené iba podľa hodnôt prvého atribútu. V prípade rovnakej hodnoty prvého atribútu sú objekty v zozname $T$ zostupne radené podľa hodnôt ďalších atribútov. Toto poradie zaručuje fakt, že sa B$^+$-strome vyberá vždy kľúč s ďalšou najväčšou hodnotou (viz procedúra `searchVB`).

Vo všeobecnosti prvý objekt v zozname $T$ nie je objekt $x$ s najvyšším ohodnotením $@(x)$. Napríklad pri hľadaní najlepšieho objektu vo VB-strome na obrázku 7.2.1 pri použití agregačnej funkcie $@(a_1, a_2) = a_1 + a_2$ je ohodnotenie prvého objektu $x_K$ v zozname $T$ rovné $@(x_K) = 1.0 + 0.5 = 1.5$. Pritom ohodnotenie štvrtého objektu $x_M$ je rovné $@(x_M) = 0.8 + 0.8 = 1.6$, takže platí $@(x_K) < @(x_M)$. Objekt $x_M$ je najlepší objekt vzhľadom na zadanú agregačnú funkciu, aj keď nie je po vykonaní procedúry `searchVB` na prvom mieste v zozname $T$.

Aby bolo možné nájsť vo VB-strome $K$ najlepších objektov, je potrebné získať z VB-stromu všetky objekty $x \in X$ (ako zoznam $T$), vypočítať pre všetky objekty $x \in T$ ohodnotenie $@(x)$ a vybrať $K$ objektov s najvyšším ohodnotením.

Takýto postupom sa musia získať všetky objekty VB-stromu analogicky ako v základnom algoritme (viz kapitola 3.1). Faginov algoritmus (viz kapitola 4) dokáže nájsť $K$ najlepších objektov bez prejdenia všetkých objektov $x \in X$. Poskytuje tak novú motiváciu pre vytvorenie optimálnejšieho riešenia top-k problému vo VB-strome.

### 7.2.2 Použitie Faginovho algoritmu vo VB-strome

Pôvodný Faginov algoritmus dokáže v zostupne zoradených zoznamoch $L_1, \ldots, L_m$ nájsť $K$ najlepších objektov vzhľadom na agregačnú funkciu @, ktorá je monotónna

vzhľadom ku zoradeniam v zoznamoch (viz kapitola 4.1.2). Každý zoznam $L_i$ pritom obsahuje pre všetky objekty $x \in X$ dvojice objekt $x$ s hodnotu $i$-tého atribútu $a_i^x$, $\{x, a_i^x\}$. Pred výpočtom Faginovho algoritmu je každý zoznam zostupne zotriedený podľa $a_i^x$ .

Obidve verzie Faginovho algoritmu (TA a NRA) potrebujú sekvenčne pristupovať do zoznamov $L_1$ ,…, $L_m$ a získavať tak postupne dvojice $\{x, a_i^x\}$. Pre získanie takejto dvojice by sa musel získať z VB-stromu objekt . Na to je potrebné prejsť VB-strom z koreňa cez všetky jeho úrovne až po pole objektov, v ktorom sa nachádza objekt. Počas tohto priechodu VB-stromom až k nejakému objektu sa musí prejsť v každej úrovni cez jeden $B^+$-strom. Z každého $B^+$-stromu sa pomocou kľúča (hodnoty atribútu) v jeho listovej úrovni prejde na $B^+$-strom v ďalšej úrovni, alebo v prípade $m$-tej (poslednej) úrovne sa prejde na pole objektov, ktoré obsahuje objekt.

Pri získaní ľubovolného objektu $x$ sa tak pri priechode VB-stromom sa zistia hodnoty všetkých jeho atribútov. VB-strom teda neposkytuje možnosť sekvenčne získavať dvojice $\{x, a_i^x\}$ (objekt s $i$-tou hodnotou jedného atribútu). V zmysle Faginovho algoritmu sa nedá sekvenčne pristupovať do zoznamov $L_1$ ,…, $L_m$. Nie je preto možné použiť Faginov algoritmus vo VB-strome.

Algoritmy TA a NRA však majú užitočné vlastnosti, ktoré priaznivo vplývajú na počet prístupov. Aj keď nie je možné vo VB-strome na vyhľadanie $K$ najlepších objektov vzhľadom na agregačnú funkciu @ priamo použiť algoritmy TA a NRA, je možné použiť niektoré ich vlastnosti, ktoré znižujú počet získaných objektov z VB-stromu, čiže znižujú počet prístupov do VB-stromu.

### 7.2.3 Výhody algoritmu TA

$K$ najlepších objektov je možné vo VB-strome nájsť rekurzívnym prehľadaním stromu. Pomocou procedúry `searchVB` sa z VB-stromu postupne získavajú objekty. Keďže pri získaní objektu $x$ z VB-stromu sú všetky jeho hodnoty atribútov známe, je možné okamžite vypočítať ohodnotenie objektu $@(x)$. Analogicky sa v algoritme TA pre novozískaný objekt priamym prístupom získavajú neznáme hodnoty atribútov objektu $x$. Celkovo sa tak získa nový objekt aj so všetkými jeho hodnotami atribútov.

Z tohto dôvodu je vhodné na vyhľadávanie $K$ najlepších objektov VB-strome použiť analogickú konštrukciu ako v prípade algoritmu TA. Podobne ako v algoritme TA, použije sa pomocný zoznam $T_K$, v ktorom bude aktuálne $K$ najlepších objektov, ktoré budú vždy zostupne zoradené podľa ich ohodnotenia $@(x)$. Ohodnotenie $K$-teho najlepšieho objektu v $T_K$ sa bude označovať $M_K$. Pre každý novozískaný objekt $x$, ktorý sa získal počas prechádzania VB-stromu procedúrou `searchVB`, sa bude postupovať rovnako ako v algoritme TA:

```
var Zoznam T_K;
procedure pridajObjekt(Object x, Aggregacia @, int K);
    if( |T_K| < K )then
        objekt x sa vloží do zoznamu T_K na správne miesto podľa @(x) ;
    else
        if( @(x) > M_K )then begin
            zo zoznamu T_K vyhodí K-ty objekt ;
            objekt x sa vloží do T_K na správne miesto ;
        end;
end.
```

V zozname $T_K$ nezáleží na presnom poradí objektov, ktoré majú rovnaké ohodnotenie podľa agregačnej funkcie @. Tak ako v prípade TA, zoznam $T_K$ nikdy neobsahuje viac ako $K$ objektov. Ak už zoznam $T_K$ obsahuje $K$ objektov a novozískaný objekt $x$ má rovnaké ohodnotenie @($x$) ako ohodnotenie $K$-teho najlepšieho objektu v zozname $T_K$, tak už sa objekt $x$ do zoznamu $T_K$ nedostane. Ak sa pri hľadaní $K$ najlepších objektov v množine X podľa agregačnej funkcie @ vyskytne viacero objektov s ohodnotením rovnakým ako $K$-ty najlepší objekt, vo výslednom zozname $T_K$ sa na $K$-tom mieste objaví iba ten objekt, ktorý sa ako prvý získal z príslušnej dátovej štruktúry.

Rekurzívnym prehľadaním celého VB-stromu sa získajú všetky objekty $x \in X$. Pri použití zoznamu $T_K$ v ňom zostane $K$ najlepších objektov $x \in X$ s najvyšším ohodnotením @($x$). Týmto postupom sa však stále musí prehľadať celý VB-strom.

### 7.2.4 Výhody algoritmu NRA

Užitočnou vlastnosťou algoritmu NRA je použitie $W$-ohodnotenia $W(x)$ a $B$-ohodnotenia $B(x)$ objektov. V algoritme NRA sa prepočítavajú $B(x)$ a $W(x)$ v prípade, ak objekt $x$ nemá známe všetky hodnoty atribútov. Vo VB-strome takáto situácia ale nemôže nastať, pretože v momente, keď sa objekt získa z VB-stromu, sú už všetky jeho hodnoty atribútov známe.

Výhody $B(x)$ a $W(x)$ sa ale dajú vo VB-strome použiť iným spôsobom. Vo VB-strome sa preto zavádza *B-ohodnotenie* $B^+$-stromu. Analogicky ako pri zavedení $W$-ohodnotenia a $B$-ohodnotenia pri algoritme NRA sa predpokladá, že agregačná funkcia @($p_1$, …, $p_m$) je neklesajúca vzhľadom na všetky svoje premenné $p_1$, …, $p_m$. (viz kapitola 4.1.2).

Ďalej sa predpokladá, že hodnoty atribútov objektov sú normalizované na interval [0, 1], $dom(A_j) \subseteq [0, 1]$, $j = 1, ..., m$. V takomto prípade je zrejmé, hodnota ľubovolného atribútu objektu $x \in X$ bude maximálne 1.

**Definícia 1:** *B-ohodnotenie* $B(S)$ $B^+$-stromu $S$ *i-t*ej úrovne s identifikátorom ($k_1, ..., k_{i-1}$) sa rozumie ako ohodnotenie zatiaľ neznámeho objektu $x$ vypočítané pomocou neklesajúcej agregačnej funkcie @($a_1^x$, …, $a_m^x$), kde hodnoty prvých $i$ – 1 atribútov objektu $x$ sú $k_1$, …, $k_{i-1}$ a hodnoty ostatných atribútov sú 1. Potom platí:

$$B(S) = @(k_1, \ldots, k_{i-1}, 1, \ldots, 1).$$

### 7.2.5 Množina dosiahnuteľných objektov

Pre každý $B^+$-strom $S$ vo VB-strome existuje jednoznačne určená podmnožina všetkých objektov množiny $X$, ktorá sa bude označovať *množina dosiahnuteľných objektov* $X^S$ z $B^+$-stromu $S$. Množina $X^S$ teda obsahuje iba objekty, ktoré sú potomkami $B^+$-stromu $S$.

*Poznámka*: Množina $X^S$ sa dá predstaviť ako množina všetkých objektov vo VB-strome, ku ktorým sa dá dostať z $B^+$-stromu $S$ pomocou ukazovateľov jeho listovej úrovne, ktoré ukazujú na $B^+$-stromy v ďalších úrovniach VB-stromu a tie ďalej na polia objektov v ktorých sa nachádzajú práve objekty množiny $X^S$. Napríklad pre $B^+$-strom prvej úrovne je jeho množina dosiahnuteľných objektov rovná celej množine objektov $X$.

**Definícia 2:** Nech $S$ je $B^+$-strom $i$-tej úrovne s identifikátorom ($k_1$, …, $k_{i-1}$). Potom množina dosiahnuteľných objektov $X^S$ z $B^+$-stromu $S$ obsahuje všetky objekty $x \in X$ pre ktoré platí, že hodnoty prvých $i$ – 1 atribútov $a_1^x$, …, $a_{i-1}^x$ týchto objektov sú rovné hodnotám $k_1$, …, $k_{i-1}$ z identifikátora tohto $B^+$-stromu $S$.

**Tvrdenie 1:** Nech $S$ je B$^+$-strom $i$-tej úrovne vo VB-strome s identifikátorom $(k_1, \ldots, k_{i-1})$, nech $B(S)$ je jeho $B$-ohodnotenie, $X^S$ je množina dosiahnuteľných objektov z B$^+$-stromu $S$ a agregačná funkcia @ je neklesajúca vzhľadom na všetky svoje premenné. Potom platí:

$$\forall\, x \in X^S : @(x) \leq B(S).$$

Dôkaz tohoto tvrdenia priamo vyplýva z predchádzajúcich definícií 1 a 2.

Takýmto spôsobom je možné odhadnúť ohodnotenie objektov $x \in X^S$ z množiny dosiahnuteľných objektov z B$^+$-stromu $S$. Tento fakt je možné použiť pri hľadaní $K$ najlepších objektov vo VB-strome, objektov $x \in X$ s najvyšším ohodnotením $@(x)$ podľa neklesajúcej agregačnej funkcie @.

### 7.2.6 Využitie B-ohodnotenia B$^+$-stromu

Pri rekurzívnom získavaní objektov z VB-stromu sa nová procedúra `searchVB` spustí v B$^+$-strome $S$ $i$-tej úrovne s identifikátorom $(k_1, \ldots, k_{i-1})$. V tomto B$^+$-strome sa postupne získavajú kľúče $k_i \in dom(A_i)$. Pomocou získaných kľúčov sa pristupuje do ďalších častí VB-stromu, odkiaľ sa získavajú ďalšie nové objekty.

Ak ukazovateľ kľúča $k_i$ odkazuje na B$^+$-strom, v tomto B$^+$-strome $S$ s identifikátorom $(k_1, \ldots, k_{i-1}, k_i)$ by sa mala spustiť nová procedúra `searchVB`. Spustením procedúry `searchVB` v B$^+$-strome $S$ by sa získali objekty $x \in X^S$ . Množina dosiahnuteľných objektov $X^S$ zo B$^+$-stromu $S$ nie je v tej chvíli známa. Aj napriek tomu je zrejmé, že ohodnotenie každého objektu $x \in X^S$ bude menšie alebo rovné ako $B$-ohodnotenie B$^+$-stromu $S$, pretože pre všetky objekty $x \in X^S$ platí $@(x) \leq B(S)$.

Keď pri rekurzívnom hľadaní $K$ najlepších objektov vo VB-stromu pomocný zoznam $T_K$ už obsahuje $K$ objektov je možné posúdiť, či má význam pre novozískaný kľúč $k_i$ pristupovať do ďalších častí VB-stromu.

Ak ukazovateľ kľúča $k_i$ odkazuje na B$^+$-strom $S$ a $B(S)$ nie je väčšie ako ohodnotenie $K$-teho najlepšieho objektu $M_K$ v zozname $T_K$ , potom pre všetky objekty $x \in X^S$ platí $@(x) \leq B(S) \leq M_K$. V takomto prípade nemá zmysel spustiť procedúru `searchVB` v B$^+$-strome $S$, pretože žiaden novozískaný objekt $x \in X^S$ sa už nemôže dostať do zoznamu $T_K$. Spustením procedúry `searchVB` v B$^+$-strome $S$ by sa len prehľadala zbytočná časť VB-stromu a vykonali by sa zbytočné prístupy do VB-stromu.

Táto situácia nastáva aj v B$^+$-stromoch v $m$-tej úrovni VB-stromu. V takomto B$^+$-strome $S$ s identifikátorom $(k_1, \ldots, k_{m-1})$ sa postupne získavajú kľúče $k_m \in dom(A_m)$. Každý ukazovateľ kľúča $k_m$ odkazuje na pole objektov $P_o$ . Už pri získaní kľúča $k_i$ je už možné vypočítať ohodnotenie $@(x)$ všetkých objektov $x \in P_o$ , ktoré sa bude označovať $@(P_o)$. Pretože sú už všetky hodnoty atribútov známe, platí $@(P_o) = @(k_1, \ldots, k_{m-1}, k_m)$.

Ak toto ohodnotenie $@(P_o)$ nie je väčšie ako ohodnotenie $K$-teho najlepšieho objektu $M_K$ v zozname $T_K$, pre všetky objekty $x \in P_o$ potom platí $@(x) \leq M_K$. V takomto prípade tiež nemá zmysel získavať objekty z tohto poľa objektov, pretože žiaden z nich sa už nemôže dostať do zoznamu $T_K$.

Na základe B-ohodnotenia je teda možné rozhodnúť, kedy má zmysel pre novozískaný kľúč pristupovať do ďalších častí VB-stromu. Tento fakt popisuje nasledujúce tvrdenie.

**Tvrdenie 2:** Nech sa v procedúre `searchVB` v niektorom jej (while) cykle vyberie kľúč $p$ v B$^+$-strome s identifikátorom $(k_1, \ldots, k_{i-1})$. Ukazovateľ kľúča $p$ pritom odkazuje na B$^+$-strom $P$ ďalšej úrovne s identifikátorom $(k_1, \ldots, k_{i-1}, p)$. Nech je agregačná funkcia @ je neklesajúca vzhľadom na všetky svoje premenné a $K$-ty objekt zoznamu $T_K$ má ohodnotenie $M_K$. Potom platí:

$$B(P) \leq M_K \Rightarrow \forall x \in X^P : @(x) \leq M_K .$$

**Dôkaz:** Ak ukazovateľ kľúča $p$ odkazuje na $P$, potom podľa tvrdenia 1 pre všetky objekty $x \in X^P$ platí $@(x) \leq B(P)$. Nech $T_K$ už obsahuje $K$ objektov je známa hodnota $M_K$. Ak $B(P) \leq M_K$ potom platí $\forall x \in X^P : @(x) \leq B(P) \leq M_K$ . □

**Dôsledok:** Keď ukazovateľ kľúča $p$ odkazuje na B$^+$-strom $P$ s identifikátorom $(k_1, \ldots, k_{i-1}, p)$ a zároveň platí $B(P) \leq M_K$, žiaden z objektov $x \in X^P$ nebude mať lepšie ohodnotenie ako $K$-ty objekt zoznamu $T_K$. Preto nemá zmysel získavať objekty z $P$, nie je potrebné spustiť procedúru `searchVB(` *VB-strom*`,` $(k_1, \ldots, k_{i-1}, p)$ `)` .

### 7.2.7 Využitie monotónnosti agregačnej funkcie

Algoritmus TA používa prahovú hodnotu, ktorá sa vypočíta dosadením naposledy videných hodnôt atribútov v zostupne zotriedených zoznamoch do monotónnej agregačnej funkcie @. Algoritmus prepočítava túto prahovú hodnotu počas získavania nových objektov. V momente keď $K$-ty objekt zoznamu $T_K$ má väčšie ohodnotenie ako je prahová hodnota, algoritmus končí, pretože žiaden ďalší novozískaný objekt sa už nemôže dostať do zoznamu $T_K$. Vyplýva to z vlastností neklesajúcej agregačnej funkcie a zároveň zo zostupného zoradenia zoznamov podľa hodnôt atribútov.

V prípade rekurzívneho získavania objektov z VB-stromu je možné vlastnosti neklesajúcej agregačnej funkcie využiť v procedúre `searchVB`. Nech sa procedúra `searchVB` spustí v B$^+$-strome $S$ $i$-tej úrovne s identifikátorom $(k_1, \ldots, k_{i-1})$. V tomto B$^+$-strome $S$ sa postupne vyberá vždy ďalší kľúč $k_i$ s najväčšou hodnotou $i$-tého atribútu v B$^+$-strome $S$. Kľúče sa teda v B$^+$-strome $S$ vyberajú v zostupnom poradí.

**Pozorovanie 1:** Nech sa vo procedúre `searchVB` v niektorom jej (while) cykle vyberie kľúč $p$ v B$^+$-strome $S$ a v nasledujúcom while cykle kľúč $q$. Keďže kľúče sa v B$^+$-strome $S$ vyberajú v zostupnom poradí, platí

$$p \geq q.$$

**Pozorovanie 2:** Ak B$^+$-strom $S$ nie je v poslednej úrovni, ukazovatele kľúčov $p$ a $q$ odkazujú na B$^+$-stromy $P$ a $Q$ v ďalšej úrovni VB-stromu. Z vlastností neklesajúcej agregačnej funkcie @ vyplýva nasledovné:

$$p \geq q \Rightarrow @(k_1, \ldots, k_{i-1}, p, 1, \ldots, 1) \geq @(k_1, \ldots, k_{i-1}, q, 1, \ldots, 1) \Rightarrow B(P) \geq B(Q).$$

**Pozorovanie 3:** Ak je B$^+$-strom $S$ v poslednej úrovni, ukazovatele kľúčov $p$ a $q$ odkazujú na polia objektov $P$ a $Q$. Z vlastností neklesajúcej agregačnej funkcie @ vyplýva nasledovné:

$$p \geq q \Rightarrow @(k_1, \ldots, k_{i-1}, p) \geq @(k_1, \ldots, k_{i-1}, q) \Rightarrow @(P) \geq @(Q).$$

Z týchto pozorovaní a z predchádzajúceho tvrdenia a plynie nasledujúce tvrdenie.

**Tvrdenie 3:** Nech sa v procedúre `searchVB` postupne získavajú kľúče z B$^+$-stromu $S$ s identifikátorom ($k_1$, …, $k_{i-1}$). Ukazovateľ kľúča $p$ pritom odkazuje na B$^+$-strom ďalšej úrovne $P$ resp. na pole objektov $P$. Nech je agregačná funkcia @ je neklesajúca vzhľadom na všetky svoje premenné a $K$-ty objekt zoznamu $T_K$ má ohodnotenie $M_K$. Ak platí $B(P) \leq M_K$, potom procedúra `searchVB(` *VB-stom*`,` ($k_1$, …, $k_{i-1}$) `)` môže v B$^+$-strome $S$ skončiť.

**Dôkaz:** Keďže kľúče sa z B$^+$-stromu $S$ získavajú v zostupnom poradí, pre každý ďalší kľúč $q$ získaný neskôr ako kľúč $p$ platí $p \geq q$ (pozorovanie 1). Ukazovatele kľúčov $p$ a $q$ odkazujú na B$^+$-stromy $P$ a $Q$ v ďalšej úrovni MD-tree resp. na polia objektov $P$ a $Q$. Podľa pozorovania 2 a pozorovania 3 platí $B(P) \geq B(Q)$. Z tvrdenia 2 potom pre všetky objekty $x \in X^Q$ platí $@(x) \leq B(Q) \leq B(P) \leq M_K$, čiže $@(x) \leq M_K$. Pre všetky ďalšie kľúče $q \leq p$ získané z B$^+$-stromu $S$ nie je už potrebné prehľadať B$^+$-strom $Q$ (resp. pole objektov $Q$) pomocou procedúry `searchVB(` *VB-stom*`,` ($k_1$, …, $k_{i-1}$, $q$) `)`, pretože žiaden objekt $x \in X^Q$ sa už nemôže dostať do zoznamu $T_K$ . Z B$^+$-stromu $S$ nie je už potrebné získavať ďalšie kľúče a procedúra `searchVB(` *VB-stom*`,` ($k_1$, …, $k_{i-1}$) `)` môže v B$^+$-strome $S$ skončiť. □

## 7.3 VB algoritmus

Použitím predchádzajúcich tvrdení vzniká *VB algoritmus*. VB algoritmus používa na prehľadávanie VB-stromu do hĺbky novú rekurzívnu procedúru `searchTopK`, ktorá vychádza z procedúry `searchVB`. Nová procedúra `searchTopK` má navyše v sebe zakomponované tvrdenie 3. Zabráni sa tak zbytočnému prehľadávaniu niektorých častí (vetiev) VB-stromu, v ktorých už nie je možné nájsť niektorý z požadovaných $K$ najlepších objektov.

VB algoritmus teda dokáže vo VB-strome vyhľadať $K$ najlepších objektov vzhľadom na monotónnu agregačnú funkciu @ bez prejdenia všetkých objektov.

Popis VB algoritmu:

**Input:** `VBstrom` *VB-stom*`, Aggregacia @, int` $K$`;`

**Output:** `Zoznam` $T_K$`;`

**var** `Zoznam` $T_K$`;`
**begin**
  `searchTopK(` *VB-stom*`, (Ø) , @, int` $K$ `);`
  **return** $T_K$`;`
**end.**

**procedure** `dalsiKluc(VBstrom` *VB-stom*`, Identikator` ($k_1$, …, $k_{i-1}$) `);`

V zadanom B$^+$-strome $i$-tej úrovne s identifikátorom ($k_1$, …, $k_{i-1}$) sa vyberie (ďalší) kľúč $k_i$. s (ďalšou) najväčšou hodnotou $i$-tého atribútu v B$^+$-strome.

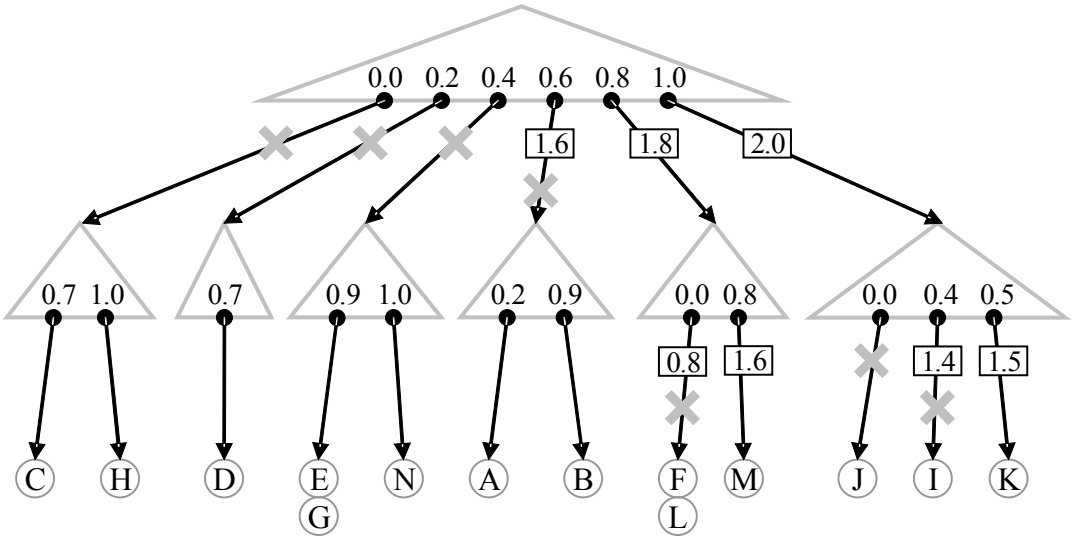
**return** $k_i$ `;`

**end.**

```
procedure searchTopK( VBstrom VB-stom, Identikator (k1, …, ki-1),
                      Aggregacia @, int K );
   while( existuje ďalší kľúč v B+-strome s identifikátorom (k1, …, ki-1) ) do
      ki = dalsiKluc( VB-stom, (k1, …, ki-1) ) ; { ukazovateľ kľúča ki
           odkazuje na B+-strom P v ďalšej úrovni resp. na pole objektov P. }
      if( |TK| = K and B(P) ≤ MK ) then return; { Tvrdenie 3 }
      if( P je B+-strom ) then
         searchTopK( VB-stom, (k1, …, ki-1, ki) , @, int K );
      if( P je pole objektov ) then;
         while( existuje ďalší objekt x v P ) do
            if( |TK| < K ) then
               objekt x sa vloží do TK na správne miesto podľa @(x) ;
            else
               if( @(x) > MK ) then begin
                  zo zoznamu TK vyhodí K-ty objekt ;
                  objekt x sa vloží do TK na správne miesto ;
               end
               else return; { Tvrdenie 3 }
         endwhile;
   endwhile;
end.
```

**Príklad 6, Hľadania najlepšieho objektu**

Pre lepšiu predstavu o VB-algoritme sa uvádza príklad vyhľadania najlepšieho objektu vo VB-strome. Na obrázku 7.3.1 je znázornený VB-strom s dvoma úrovňami, v ktorom je znázornený postup pri vyhľadávaní najlepšieho objektu vzhľadom k agregačnej funkcii $@(a_1, a_2) = a_1 + a_2$.




*Obrázok 7.3.1, Hľadanie najlepšieho objektu vo VB-strome*

Procedúra `searchTopK` sa spustí na začiatku v B$^+$-strome prvej úrovne s identifikátorom (Ø) a vyberie sa kľúč 1.0. Pre tento kľúč sa spustí procedúra `searchTopK` v B$^+$-strome druhej úrovne s identifikátorom (1.0) a vyberie sa z neho kľúč 0.5. Objekt $x_K$ sa vloží do zoznamu $T_K$ ako zatiaľ najlepší videný objekt s ohodnotením $@(x_K) = 1.5$. V B$^+$-strome (1.0) sa vyberie ďalší kľúč 0.4. Objekt sa $x_I$ sa už z poľa objektov nezíska, pretože ohodnotenie poľa objektov je menšie ako ohodnotenie najlepšieho objektu v zozname $T_K$. Procedúra následne `searchTopK` v B$^+$-strome (1.0) končí, pretože už v tomto B$^+$-strome neexistuje lepší objekt ako $x_K$.

V B$^+$-strome prvej úrovne s identifikátorom (Ø) sa vyberie ďalší kľúč 0.8. Pre tento kľúč sa spustí procedúra `searchTopK` v B$^+$-strome druhej úrovne s identifikátorom (0.8) a vyberie sa z neho kľúč 0.8. Objekt $x_M$ sa vloží do zoznamu $T_K$ ako najlepší objekt s ohodnotením $@(x_M) = 1.6$. Objekt $x_K$ sa z $T_K$ vyhodí, lebo má menšie ohodnotenie ako $x_M$. V B$^+$-strome (0.8) sa vyberie ďalší kľúč 0.0. Do príslušného poľa objektov sa nepristúpi, pretože jeho ohodnotenie je menšie ako $@(x_M)$ a procedúra `searchTopK` v B$^+$-strome (0.8) končí.

V B$^+$-strome prvej úrovne s identifikátorom (Ø) sa vyberie ďalší kľúč 0.6. Tento kľúč by sa mala spustiť procedúra `searchTopK` v B$^+$-strome druhej úrovne s identifikátorom (0.6). To sa však nestane, pretože B-ohodnotenie tohto stromu nie je väčšie ako ohodnotenie najlepšieho objektu v $T_K$. Z monotónnosti agregačnej funkcie @ vyplýva, že už vo VB-strome nie je možné získať objekt s lepším ohodnotením ako $@(x_M)$, preto procedúra `searchTopK` v B$^+$-strome (Ø) končí, a teda končí aj VB algoritmus.

V zozname $T_K$ nakoniec zostane objekt $x_M$, s najlepším ohodnotením $@(x_M)$.

## 7.4 Viacužívateľský prístup

VB algoritmus nerieši problematiku lokálnych užívateľských preferencií. Dokáže nájsť $K$ najlepších objektov vzhľadom na monotónnu agregačnú funkciu. Podľa zvoleného modelu užívateľských preferencií (viz kapitola 2.4), by mali byť hodnoty atribútov objektov ohodnotené lokálne podľa užívateľských fuzzy funkcií $f_1^U, \ldots, f_m^U$ a potom by sa už globálne ohodnotenie objektu $x$ sa vypočítalo agregáciou lokálnych ohodnotení pomocou užívateľskej agregačnej funkcie @.

### 7.4.1 Lokálne preferencie vo VB-strome

Podľa modelu zoznamu založenom na B$^+$-strome (viz kapitola 6 ) je možné z B$^+$-stromu získavať dvojice $\{x, f^U(x)\}$ v zostupnom poradí podľa lokálnej užívateľskej preferencie vyjadrenej pomocou užívateľskej fuzzy funkcie $f^U$. Keďže v tejto práci sa používa varianta VB-stromu založená na B$^+$-stromoch, je možné tento model získavania dvojíc aplikovať v jednotlivých B$^+$-stromoch VB-stromu.

Ak sa na získavanie kľúčov z B$^+$-stromu použije model zoznamu založený na B$^+$-strome, pomocou príslušnej fuzzy funkcie je možné získavať z B$^+$-stromu lokálne ohodnotené kľúče v zostupnom poradí vzhľadom ku fuzzy funkcii $f^U$. Keďže v B$^+$-stromoch VB-stromu nie sú podľa kľúčov (podľa hodnôt atribútov) priamo indexované objekty ale len ukazovatele na ďalšie uzly VB-stromu, nie je možné získavať dvojice zložené z objektu $x$ a jeho ohodnoteného atribútu $\{x, f^U(x)\}$. V tomto prípade sa z B$^+$-stromu získavajú dvojice zložené z kľúča $k$ a jeho ohodnotenia $f^U(k)$, čiže dvojice $\{k, f^U(k)\}$.

Normalizované hodnoty *i*-tého atribútu objektov $x \in X$ sú vo VB-strome uložené v B$^+$-stromoch *i*-tej úrovne (ako kľúče). Preto sa v každom B$^+$-strome *i*-tej úrovne použije na získavanie dvojíc fuzzy funkcia *i*-ta užívateľská fuzzy funkcia $f_i^U$. Z každého B$^+$-stromu *i*-tej úrovne sa teda budú získavať dvojice $\{k, f_i^U(k)\}$ v zostupnom poradí podľa $f_i^U(k)$.

### 7.4.2 Aplikácia lokálnych preferencií vo VB algoritme

V prípade VB algoritmu je možné aplikovať lokálne užívateľské preferencie priamo pri rekurzívnom hľadaní *K* najlepších objektov vo VB-strome. V pôvodnej procedúre `searchTopK` algoritmu VB sa získavajú kľúče z príslušného B$^+$-stromu pomocou procedúry `dalsiKluc`. Kľúče sa získavajú v zostupnom poradí podľa hodnôt kľúčov (atribútov). V B$^+$-strome *i*-tej úrovne sa vyberá vždy ďalší kľúč s ďalšou najväčšou hodnotou *i*-tého atribútu v B$^+$-strome.

Pre zavedenie lokálnych preferencií vo VB algoritme postačí, ak sa upraví procedúra `dalsiKluc` tak, že z príslušného B$^+$-stromu *i*-tej úrovne sa získavajú kľúče *k* v zostupnom poradí podľa $f_i^U(k)$, teda podľa ohodnotenia kľúčov podľa *i*-tej užívateľskej fuzzy funkcie $f_i^U$. Procedúra `dalsiKluc` bude potom vyzerať nasledovne:

**procedure** dalsiKluc(VBstrom *VB-stom*, Identikator ($k_1$, …, $k_{i-1}$), FuzzyFunkcia $f_i^U$ );

V zadanom B$^+$-strome *i*-tej úrovne s identifikátorom ($k_1$, …, $k_{i-1}$) sa vyberie (ďalší) kľúč $k_i$. s (ďalšou) najväčšou hodnotou $f_i^U(k_i)$ podľa *i*-tej užívateľskej fuzzy funkcie $f_i^U$.

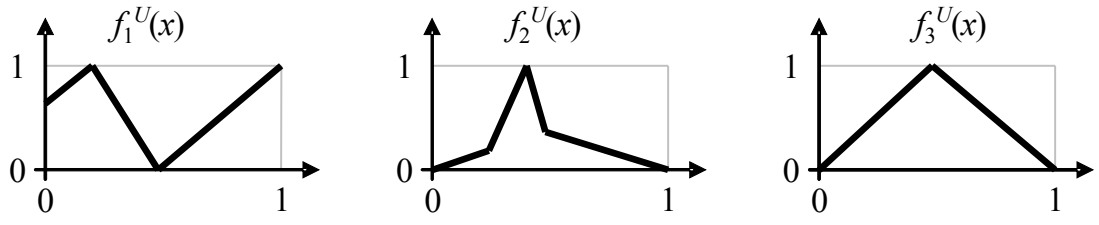
**return** $k_i$ ;

**end.**

V procedúre `dalsiKluc` pribudne ďalší vstupný parameter, *i*-tej užívateľskej fuzzy funkcie $f_i^U$, preto treba mierne upraviť aj procedúru `searchTopK`, tak aby bola procedúre `dalsiKluc` správne volaná.

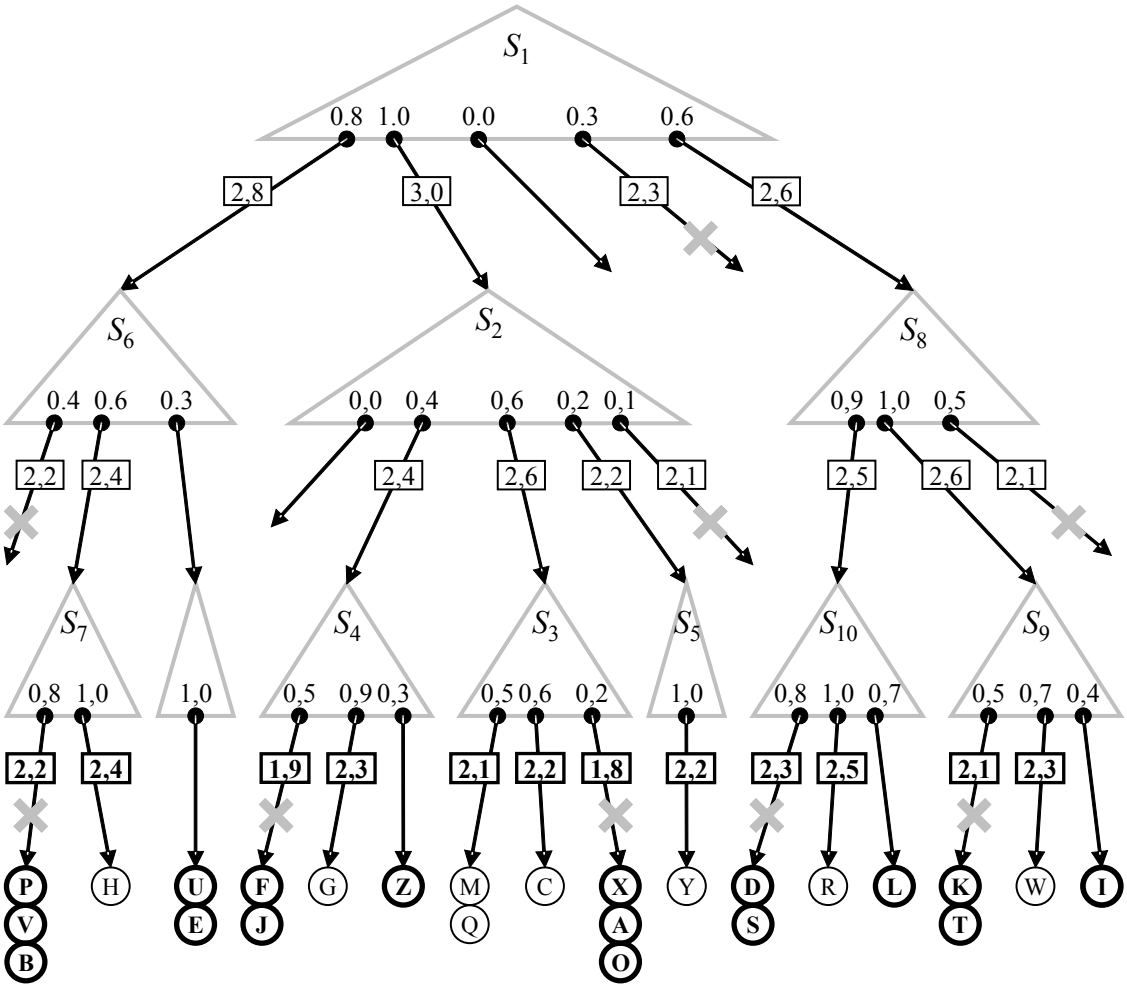
**Príklad 7, Hľadanie *K* najlepších objektov s aplikáciou lokálnych preferencií**

Ako VB algoritmus hľadá *K* najlepších objektov vo VB-strome, vzhľadom na preferencie konkrétneho užívateľa *U*, ukazuje nasledujúci príklad. Lokálne preferencie užívateľ *U* vyjadruje pomocou fuzzy funkcií $f_1^U$, $f_2^U$, $f_3^U$. Nech užívateľ preferuje objekty podľa troch atribútov tak, ako je znázornené na obrázku 7.4.3a.




*Obrázok 7.4.3a, Užívateľské fuzzy funkcie*

Pre názornosť sa predpokladá, že užívateľ $U$ pre určenie vzájomného vzťahu atribútov použil agregačnú funkciu $@(a_1, a_2, a_3)= a_1 + a_2 + a_3$. ohodnotenie ideálneho objektu 3. Užívateľ potrebuje vo VB-strome na obrázku 7.4.3b vyhľadať tri najlepšie objekty.




*Obrázok 7.4.3b, Hľadanie najlepšieho objektu vo VB-strome*

Na začiatku VB algoritmu je pomocný zoznam $T_K$ prázdny a procedúra `searchTopK` sa spustí v B$^+$-strome $S_1$ prvej úrovne. Z tohto stromu sa ako prvý vyberie kľúč s najväčšou hodnotou $f_1^U(k)$ podľa prvej užívateľskej fuzzy funkcie $f_1^U$.

Pre názornosť je na obrázku 7.4.3b nad každým kľúčom v B$^+$-strome $i$-tej úrovne znázornené jeho ohodnotenie $f_i^U(k)$ podľa $i$-tej užívateľskej fuzzy funkcie $f_i^U$. Ako je vidno už v B$^+$-strome prvej úrovne, tieto ohodnotenia kľúčov nie sú zoradené zostupne sprava doľava. Spôsobuje to fakt, že užívateľ preferuje objekty podľa prvého atribútu pomocou fuzzy funkcie $f_1^U$, ktorá je znázornená na obrázku 7.4.3a.

Zoznam $T_K$ je prázdny zatiaľ prázdny, preto sa spustí nová procedúra `searchTopK` v B$^+$-strome $S_2$ druhej úrovne a vyberie sa v ňom kľúč $k$ s najväčšou hodnotou $f_2^U(k)$. Následne sa spustí ďalšia procedúra `searchTopK` v B$^+$-strome $S_3$ tretej úrovne a vyberie sa v ňom kľúč $k$ s najväčšou hodnotou $f_3^U(k)$. Ukazovateľ tohto kľúča

odkazuje na pole objektov z ktorého sa získa objekt $x_C$. V B$^+$-strome $S_3$ sa potom vyberie kľúč $k$ s ďalšou najväčšou hodnotou $f_3^U(k)$, ktorý odkazuje na pole objektov z ktorého sa získajú objekty $x_M$ a $x_Q$. Zoznam $T_K$ už v tej chvíli obsahuje tri zostupne zoradené objekty $@(x_C) = 2.2$, $@(x_M) = 2.1$ a $@(x_Q) = 2.1$.

V B$^+$-strome $S_3$ sa vyberie kľúč $k$ s ďalšou najväčšou hodnotou $f_3^U(k)$, ktorý odkazuje na pole objektov $P$. Vypočíta sa $@(P) = 1.8$ a to je menšie ako ohodnotenie tretieho objektu v zozname $T_K$, $@(x_Q) = 2.1$. Preto aktuálna procedúra `searchTopK` v B$^+$-strome $S_3$ končí a pokračuje sa v procedúre `searchTopK` v B$^+$-strome $S_2$.

V B$^+$-strome $S_2$ sa vyberie kľúč $k$ s ďalšou najväčšou hodnotou $f_2^U(k)$. Ukazovateľ tohto kľúča odkazuje na B$^+$-strom $S_4$ ktorého $B$-ohodnotenie je $B(S_4) = 2.4$. Keďže $B(S_4)$ je väčšie ako ohodnotenie tretieho objektu v zozname $T_K$, $@(x_Q) = 2.1$. v B$^+$-strome $S_4$ sa spustí nová procedúra `searchTopK` a získa sa v ňom kľúč $k$ s najväčšou hodnotou $f_3^U(k)$. Tento kľúč odkazuje na pole objektov, z ktorého sa získa objekt $x_G$ s ohodnotením $@(x_G) = 2.3$. Objekt $x_G$ sa vloží do zoznamu $T_K$ na správne miesto. Zoznam $T_K$ potom obsahuje tri zostupne zoradené objekty $@(x_G) = 2.3$, $@(x_C) = 2.2$ a $@(x_M) = 2.1$.

V B$^+$-strome $S_4$ sa potom vyberie kľúč $k$ s ďalšou najväčšou hodnotou $f_3^U(k)$, ktorý odkazuje na pole objektov $P$ s ohodnotením $@(P) = 1.9$ ktoré je menšie ako tretí objekt v zozname $T_K$. Preto aktuálna procedúra `searchTopK` v B$^+$-strome $S_4$ končí a pokračuje sa v procedúre `searchTopK` v B$^+$-strome $S_2$.

V B$^+$-strome $S_2$ sa vyberie kľúč $k$ s ďalšou najväčšou hodnotou $f_2^U(k)$. Ukazovateľ tohto kľúča odkazuje na B$^+$-strom $S_5$ pomocou ktorého sa podarí získať objekt $x_Y$ s ohodnotením $@(x_Y) = 2.2$. Zoznam $T_K$ potom obsahuje tri zostupne zoradené objekty $@(x_G) = 2.3$, $@(x_C) = 2.2$ a $@(x_Y) = 2.2$.

V B$^+$-strome $S_2$ sa vyberie kľúč $k$ s ďalšou najväčšou hodnotou $f_2^U(k)$. Ukazovateľ tohto kľúča odkazuje na B$^+$-strom $S$, ktorého $B$-ohodnotenie je $B(S) = 2.1$. Teda nemá zmysel pokračovať a procedúra `searchTopK` v B$^+$-strome $S_2$ končí a pokračuje sa v procedúre `searchTopK` v B$^+$-strome $S_1$.

VB algoritmus ďalej pokračuje až pokiaľ neskončí procedúra `searchTopK` v B$^+$-strome $S_1$. Nakoniec zoznam $T_K$ obsahuje tri najlepšie objekty pre užívateľa $U$, ktoré sú v zozname $T_K$ zostupne zoradené: $@(x_R) = 2.5$, $@(x_H) = 2.4$ a $@(x_G) = 2.3$.

Kvôli znázorneniu, že VB algoritmus dokáže vo VB-strome nájsť $K$ najlepších objektov bez prejdenia všetkých objektov vo VB-strome, sú na obrázku 7.4.3b zvýraznené objekty, ktoré nemusel VB algoritmus získať počas svojho výpočtu.

# 8 Implementácia

Cieľom tejto práce je implementovať popisované top-k algoritmy, ktoré vyhľadávajú $K$ najlepších objektov, pričom dáta o objektoch sú uložené v relačnej databáze.

## 8.1 Analýza možných implementácií

Ideálnym riešením by bolo priamo v databázovom systéme implementovať popisované top-k algoritmy tak, že by bolo možné vyhľadať $K$ najlepších objektov priamo pomocou SQL dotazu.

V prípade VB algoritmu by sa musel v DB systéme implementovať VB-strom. Kvôli podpore lokálnych užívateľských preferencií by implementovať aj indexáciu pomocou $B^+$-stromov tak, aby bolo možné získavať dvojice $\{x, f_i^U(x)\}$ počas výpočtu algoritmov. Okrem toho by sa musela v databázovom systéme implementovať vhodná reprezentácia fuzzy funkcií, ktorá by takéto získavanie dvojíc podporovala.

Takéto riešenie by vyžadovalo veľký objem implementačnej práce. Pravdepodobne by sa vyskytlo aj veľa technických problémov pri úpravách zdrojových kódov databázového systému, ktoré priamo nesúvisia s riešením problému top-k. Zmeny databázového systému preto nebudú predmetom tejto diplomovej práce.

Aby bolo možné efektívne využiť výhody top-k algoritmov, spolu s modelom zoznamov založených na $B^+$-stromoch, musia byť objekty spolu s ich hodnotami atribútov indexované v dátovej štruktúre, ktorej základom sú $B^+$-stromy. Táto štukatúra musí umožňovať získavanie dvojíc $\{x, f_i^U(x)\}$ počas výpočtu algoritmov podľa fuzzy funkcií v zostupnom poradí vzhľadom ku $f_i^U(x)$. Žiadna z doposiaľ existujúcich relačných databáz takúto funkčnosť neposkytuje, preto pre účely tejto práce uloženie objektov $x \in X$ v klasickej relačnej databáze už nepostačuje.

### 8.1.1 Indexácia mimo databázový systém

Pre výpočet top-k algoritmov sa mimo relačnú databázu vytvoria dátové štruktúry založené na stromových štruktúrach. Relačná databáza sa v tejto práci použije len ako pomocné úložisko dát. Samotný výpočet implementovaných algoritmov bude potom prebiehať nad týmito novými dátovými štruktúrami. V prípade ak relačná databáza obsahuje ďalšie dáta o objektoch a užívateľ tieto dáta požaduje, po nájdení $K$ najlepších objektov je možné požadované dáta získať z relačnej databázy vykonaním ďalších dotazov.

Táto práca sa zameriava na vyhľadávanie $K$ najlepších objektov bez prejdenia všetkých objektov, respektívne pomocou čo najmenšieho počtu prístupov. Predmetom skúmania je preto hlavne počet vykonaných prístupov do príslušných dátových štruktúr počas výpočtu použitých algoritmov. Preto pre účely tejto práce postačí, keď budú požadované dátové štruktúry uložené v operačnej pamäti.

V operačnej pamäti je možné rýchlo a jednoducho tento počet prístupov spočítať. (napr. jednoduchou úpravou algoritmu). Navyše ako statické úložisko dát o objektoch už slúži relačná databáza. Pre potreby tejto práce preto nie je potrebné udržiavať potrebné stromové štruktúry na pevnom disku alebo na iných pamäťových médiách.

## 8.2 Spôsob implementácie

Implementácia je vytvorená v programovacom jazyku Java a ako úložisko dát je použitá relačná databáza MySQL.

Keďže implementované top-k algoritmy používajú rovnaký zdroj dát (relačnú databázu) všetky tieto algoritmy sú implementované v jednom spoločnom nástroji *TreeTopK*, ktorý sprostredkúva komunikáciu s relačnou databázou.

Pretože top-k algoritmy vyhľadávajú najlepšie objekty podľa opakovane zadávaných užívateľských preferencií, nástroj TreeTopK sprostredkúva aj interaktívny kontakt s užívateľom alebo s viacerými užívateľmi. Preto bolo v rámci vytvorené aj grafické rozhranie, v ktorom je možné pohodlne zadávať užívateľské preferencie a zadávať všetky potrebné nastavenia.

Pre výpočet top-k algoritmov je potrebné vytvoriť nové dátové štruktúry založené na stromových štruktúrach v operačnej pamäti. Tieto nové stromové štruktúry nie sú závislé na zvolených užívateľských preferenciách. V nástroji TreeTopK sa preto pred výpočtami top-k algoritmov vytvorí jedna spoločná dátová štruktúra pre všetkých užívateľov (spoločná pre rôzne zadané užívateľské preferencie). Pre účely tejto práce postačí uchovávať tieto stromové štruktúry v operačnej pamäti.

## 8.3 Zdroj dát

Nástroj TreeTopK predpokladá, že každá množina objektov rovnakého druhu je v databáze MySQL uložená ako jedna tabuľka. Nech $N$ je počet objektov a počet ich atribútov je $n$, potom má tabuľka $n$ stĺpcov a $N$ riadkov.

Popisované top-k algoritmy potrebujú pre svoj výpočet stromové dátové štruktúry. Uchovávanie objektov v klasickej relačnej databáze už nepostačuje. Kvôli aplikácii užívateľských preferencií musia byť objekty indexované v $B^+$-stromoch (vo VB-strome), tak aby bolo možné získavať objekty v zostupnom poradí vzhľadom na užívateľské fuzzy funkcie.

Pre výpočet top-k algoritmov je potrebné vytvoriť nové dátové štruktúry založené na $B^+$-stromoch (vo VB-strome). Tieto nové stromové štruktúry nie sú závislé na zvolených užívateľských preferenciách. V nástroji TreeTopK sa preto pred výpočtami top-k algoritmov vytvorí jedna spoločná dátová štruktúra pre všetkých užívateľov (spoločná pre rôzne zadané užívateľské preferencie).

### 8.3.1 Nastavenie zdroja dát

Nástoj TreeTopK sa musí na začiatku pripojiť k požadovanej databáze, ktorá je štandartne určená servrom (host), portom, menom, heslom a menom tejto databázy. Potom musí nástoj TreeTopK načítať informácie o požadovanej tabuľke. Načítajú nasledujúce informácie o stĺpcoch (o atribútoch ) požadovanej tabuľky:

- meno
- dátový typ
- minimálna hodnota
- maximálna hodnota
- veľkosť aktuálnej domény atribútu (počet rôznych hodnôt)

Podľa týchto informácií o jednotlivých atribútoch tabuľky je potrebné pred vytvorením požadovaných stromových štruktúr určiť nasledujúce:

- hodnoty ktorého atribútu budú použité ako identifikátory objektov,
- podľa ktorých $m \leq n - 1$ atribútov sa bude určovať ohodnotenie objektov (určí sa $m$ ohodnocovacích atribútov).

Ako identifikátor objektov môže byť vybraný len atribút, ktorého veľkosť domény je rovná počtu objektov $N$ v tabuľke. Pre potreby tejto práce je možné v nástroji TreeTopK použiť na určovanie ohodnotenia objektov dva druhy atribútov, ktoré spĺňajú nasledujúce podmienky.

- *spojitý atribút*
  - Hodnoty atribútu sú celočíselného typu a veľkosť aktuálnej domény atribútu je dostatočná (napr. aspoň 15 rozdielnych hodnôt).
- *diskrétny atribút*
  - Hodnoty atribútu sú iného typu (napr. textový reťazec) a veľkosť aktuálnej domény atribútu je dostatočne malá (napr. maximálne 15 rozdielnych hodnôt).

### 8.3.2 Lokálne preferencie

Podľa použitého modelu užívateľských preferencií musí užívateľ pred výpočtom top-k algoritmu zadať svoje lokálne preferencie pre jednotlivé atribúty. Na zadávanie lokálnej preferencie podľa jedného atribútu sa používa trieda *Preference*. Táto trieda obsahuje určité hodnoty z domény príslušného atribútu, ktoré užívateľ ohodnocuje. Užívateľ takto vyjadruje svoje lokálne preferencie.

Z tabuľky sa získa pre každý ohodnocovací atribút reprezentatívna vzorka hodnôt tohto atribútu, ktoré bude užívateľ neskôr ohodnocovať.

#### Spojitý atribút

Pre spojitý atribút sa získa vzostupne zotriedená postupnosť určitých hodnôt atribútu. Prvou hodnotou postupnosti bude minimálna hodnota atribútu a poslednou maximálna hodnota atribútu. Ohodnotením týchto hodnôt je možné vyjadriť preferencie pre celú doménu atribútu. Pre pre hodnotu atribútu postupnosti je jej ohodnotenie priamo určené užívateľom a ohodnotenie ostatných hodnôt atribútu je možné dopočítať zavedením lineárne reprezentovanej funkcie (viz kapitola 2.2.4).

**Príklad 8:** Nech existuje tabuľka bytov s atribútom *Rozloha* s celočíselnými hodnotami. Nech je veľkosť domény tohto atribútu dostatočná (napr. 100 rozdielnych hodnôt), minimálna hodnota atribútu je 17 $m^2$ a maximálna hodnota atribútu je 305 $m^2$. Z tabuľky sa získa 5 hodnôt atribútu *Rozloha*, ktoré bude užívateľ ohodnocovať, napríklad hodnoty 17, 57, 91, 135, 305.

#### Diskrétny atribút

Pre diskrétny atribút sa získajú všetky hodnoty z aktuálnej domény atribútu. Veľkosť domény diskrétneho atribútu by nemala byť príliš veľká (napr. maximálne 15), pretože užívateľ by bol nútený ohodnotiť veľké množstvo konkrétnych hodnôt atribútu.

**Príklad 9:** Nech existuje tabuľka bytov v Prahe s atribútmi *Okres* a *Ulica*. Veľkosť domény atribútu *Okres* bude 10 (hodnoty atribútu sú Praha 1 až Praha 10) a pre atribút *Okres* užívatelia budú ohodnocovať 10 hodnôt atribútu. Atribút *Ulica* nože mať aj viac ako 300 rozdielnych hodnôt, preto nemá zmysel použiť tento atribút. Pre užívateľov by bolo nepraktické ohodnocovať veľký počet hodnôt atribútu.

### 8.3.3 Normalizácia

Pred vytvorením stromových štruktúr v pamäti je potrebné pre každý ohodnocovací atribút vytvoriť normalizér, ktorý bude hodnoty atribútu zobrazovať na interval [0, 1]. Potom je možné pre modelovanie užívateľských preferencií použiť jednotný model fuzzy funkcií (viz kapitola 2.2.3). V TreeTopK nástroji sa pre tento účel používa abstraktná trieda *Normalizer*, ktorá obsahuje metódu zobrazujúcu hodnoty atribútu na [0, 1].

Pre spojitý atribút sa používa rozšírená trieda *NormalizerNumber*, ktorá je implementovaná pomocou lineárnej funkcie (viz kapitola 2.2.3). Na vytvorenie tohto normalizéru postačí získať z tabuľky minimálnu a maximálnu hodnotu atribútu.

Pre spojitý atribút sa používa rozšírená trieda *NormalizerDiscreet*, ktorá normalizuje hodnoty atribútu vzhľadom na poradie hodnoty v zotriedenom poli hodnôt (viz kapitola 2.2.3). Pre vytvorenie tohto normalizéru je potrebné získať z tabuľky všetky hodnoty atribútu.

Tieto hodnoty sa už získali pri vytváraní lokálnych preferencií, nie je potrebné získavať nové dáta z tabuľky. Preto je výhodné pre všetky ohodnocovacie atribúty vytvoriť triedy *Preference* a *Normalizer* spoločne. Trieda *DatabaseInterface* pre tento účel obsahuje metódu *prepearPreferencesAndNormalizers*.

## 8.4 Dátové štruktúry

TreeTopK nástroj obsahuje tri dátové štruktúry, ktoré používajú implementované top-k algoritmy. V každej z nich sú uložené všetky objekty so svojim identifikátorom a hodnotami *m* atribútov, ktoré sú normalizované na interval [0, 1].

Prvou štruktúrou je pole *m* $B^+$-stromov. V každom $B^+$-strome sú indexované identifikátory objektov podľa hodnôt jedného atribútu. $B^+$-stromy sa využívajú ako zostupne zotriedené zoznamy Faginovho algoritmu a realizuje sa pomocou nich sekvenčný prístup.

Druhou štruktúrou je mapa objektov, v ktorej sú mapované všetky hodnoty atribútov objektov podľa identifikátorov objektov. Pre ľubovoľný identifikátor objektu je tak možné rýchlo získať hodnoty všetkých atribútov objektu. Táto štruktúra simuluje priamy prístup do zoznamov Faginovho algoritmu.

Treťou štruktúrou je VB-strom, v ktorom sú indexované identifikátory objektov podľa všetkých hodnôt atribútov.

### 8.4.1 Získanie objektov z databázy

Podľa požiadaviek top-k algoritmov sa na začiatku inicializujú potrebné dátové štruktúry a určí sa atribút obsahujúci identifikátory objektov a *m* ohodnocovacích atribútov. Pre každý ohodnocovací atribút sa vytvorí normalizér.

Potom sa z tabuľky v relačnej databáze načíta požadovaných $m + 1$ hodnôt atribútov všetkých objektov. Pre každý objekt sa

- hodnoty jeho $m$ atribútov normalizujú podľa príslušných normalizérov,
- identifikátor objektu spolu s normalizovanými hodnotami atribútov sa pridá do dátových štruktúr.

Získanie všetkých potrebných dát z tabuľky v relačnej databáze a následné vybudovanie dátových štruktúr je implementované v triede *DatabaseInterface* ako metóda *prepearTrees*.

### 8.4.2 B$^+$-strom

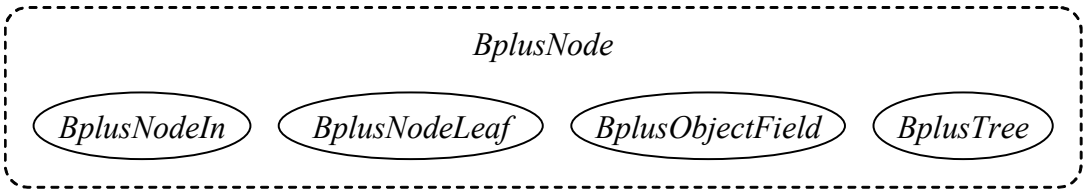
Základnou stavebnou jednotkou použitých dátových štruktúr je B$^+$-strom (viz kapitola 4.2), ktorý je implementovaný v triede *BplusTree*. Každý strom je tvorený vnútornými uzlami (trieda *BplusNodeIn*) a listovými uzlami (trieda *BplusNodeLeaf* ). Z listových uzlov je pomocu ukazovateľov kľúčov odkazované na polia objektov (trieda *BplusObjectField*), ktoré obsahujú identifikátory objektov. Kľúče B$^+$-stromu sú hodnoty z intervalu [0, 1], zotriedené vzostupne zľava doprava. Polia objektov obsahujú objekty s rovnakou hodnotou atribútu. Keď sa do B$^+$-stromu vkladá nový objekt s identifikátorom *id* a s hodnotou atribútu *a*, postupuje sa nasledovne.

- Ak už v B$^+$-strome existuje kľúč *a*, *id* sa pridá do príslušného poľa objektov.
- Ak v B$^+$-strome ešte kľúč *a* neexistuje, kľúč *a* sa vloží do B$^+$-stromu a pre tento kľúč sa vytvorí nové pole objektov, do ktorého sa vloží *id* objektu.

### 8.4.3 VB-strom

VB-stom (viz kapitola 5.3) je implementovaný ako samostatná trieda *VBTree*, ktorá obsahuje ukazovateľ na B$^+$-strom v jeho prvej úrovni. Pôvodný B$^+$-strom je upravený tak, že ak nie je v poslednej úrovni VB-stromu, tak je z listových uzlov pomocou ukazovateľov kľúčov odkazované na B$^+$-stromy v ďalšej úrovni.

Kvôli tejto úprave B$^+$-stromu vznikla abstraktná trieda *BplusNode*, ktorá reprezentuje uzol v stromovej štruktúre. Konkrétne typy uzlov sú implementované v triedach, ktoré sú rozšírením triedy *BplusNode* (viz obrázok 8.2.3). Z listového uzlu B$^+$-stromu sa všeobecne odkazuje na uzol. Týmto uzlom je buď B$^+$-strom alebo pole objektov, v prípade poslednej úrovne VB-stromu.




*Obrázok 8.2.3, Rozšírenia triedy BplusNode*

Postup vloženia nového objektu s identifikátorom *id* a s hodnotami atribútov $a_1, \ldots, a_m$ do $m$ úrovňového VB-stromu naznačujú nasledujúce dva body.

- Ak už v $B^+$-strome *i*-tej úrovne existuje kľúč $a_i$, prejde sa na príslušný strom v ďalšej úrovni $(i + 1)$ VB-stromu.
- Ak v $B^+$-strome *i*-tej úrovne ešte kľúč $a_i$ neexistuje, sa vloží sa do $B^+$-stromu a pre tento kľúč sa vytvorí nový $B^+$-strom v ďalšej úrovni $(i + 1)$ VB-stromu. Do novovzniknutého stromu sa vloží kľúč $a_{i+1}$. Postup sa analogicky opakuje až dovtedy, kým sa nevytvorí strom v poslednej úrovni s kľúčom $a_m$. Pre tento kľúč sa nakoniec vytvorí nové pole objektov, do ktorého sa vloží *id* objektu.

## 8.5 Užívateľské preferencie

Po vytvorení požadovaných stromových štruktúr v pamäti už môžu užívatelia vyhľadávať objekty podľa svojich preferencií. Po každom zadaní nových preferencií top-k algoritmus podľa nich vyhľadá K najlepších objektov. Pre zadávanie užívateľských preferencií je použitý model založený na fuzzy funkciách a následnej agregácii (viz kapitola 2.4).

Pred výpočtom top-k algoritmu musí užívateľ nastaviť svoje preferencie. Globálne preferencie užívateľ nastaví tak, že nastaví váhy jednotlivých atribútov. Pre jednotlivé atribúty užívateľ nastaví svoje lokálne preferencie tak, že pre každý ohodnocovací atribút ohodnotí reprezentatívnu vzorku hodnôt tohto atribútu.

### 8.5.1 Agregačná funkcia

Agregačnú funkciu reprezentuje abstraktná trieda *Aggregation*, ktorá sa vždy vytvorí pred výpočtom top-k algoritmu podľa nastavených váh atribútov. Trieda *Aggregation* obsahuje dve metódy ohodnocovania objektov. Prvá metóda dostáva ako vstup všetky hodnoty atribútov objektu a vypočíta jeho ohodnotenie. Druhá metóda dokáže ohodnotiť aj objekt, ktorý nemá známe všetky hodnoty atribútov. Ako vstup dostáva iba známe hodnoty atribútov objektu a náhradné hodnoty atribútov. Chýbajúce hodnoty atribútov objektu sa doplnia príslušnými náhradnými hodnotami. Táto metóda umožňuje jednoducho vypočítať *W*-ohodnotenie a *B*-ohodnotenie objektov použitím vhodných náhradných hodnôt.

Najnázornejším typom monotónnej agregačnej funkcie je pre užívateľov vážený priemer, ktorý je implementovaný v rozšírenej triede *WeightedAverage*. Ako príklady iných agregačných funkcií sú implementované euklidovská vzdialenosť so zohľadnením váh jednotlivých atribútov v triede *EuclidDistance*, minimum zo všetkých hodnôt atribútov v triede *Minimum* a maximum v triede *Maximum*.

### 8.5.2 Fuzzy funkcia

Pre každý ohodnocovací atribút sa podľa ohodnotenia reprezentatívnu vzorku hodnôt tohto atribútu vytvorí fuzzy funkcia, ktorá je implementovaná v triede *FuzzyFunction*. Táto trieda a jej metódy sú implementované tak, že sú nezávislé na tom, či je atribút spojitý alebo diskrétny.

Trieda *FuzzyFunction* obsahuje zoznam bodov roviny so súradnicami hodnota atribútu a jej ohodnotenie. Tieto body reprezentujú začiatky a konce úsečiek, ktoré tvoria lineárnu reprezentáciu fuzzy funkcie (viz kapitola 2.2.4). Fuzzy funkcia sa vytvára pridávaním týchto bodov (metóda *addPoint*). Pre každú z úsečiek fuzzy funkcie sa uchováva jej sklon (derivácia), aby bolo možné rýchlo získať pre určitú hodnotu atribútu ohodnotenie podľa tejto fuzzy funkcie (metóda *addPoint*).

**Získavanie dvojíc z B$^+$-stromu**

Fuzzy funkcia sa používa aj na získavanie dvojíc z B$^+$-stromu v zostupnom poradí (viz kapitola 6). Dvojica je implementovaná ako trieda *RatedObject*, ktorá obsahuje identifikátor objektu s ohodnotenou hodnotou atribútu objektu podľa fuzzy funkcie. Podľa monotónnosti fuzzy funkcie sa vytvoria cesty (metóda *createWays*), pomocou ktorých sa budú získavať dvojice z B$^+$-stromu (viz kapitola 5.1.2).

Podľa kapitoly 5.2 je potrebné pred získavaním dvojíc z požadovaného B$^+$-stromu podľa ciest vyhľadať množinu aktuálnych objektov (dvojíc). K tomu je určené metóda *activateWays*. Potom už je možné z B$^+$-stromu získavať dvojice opakovaným volaním metódy *getRatedObject*, ktorá vráti *RatedObject*.

Podobne sa sa fuzzy funkcia používa aj vo VB-strome. V rekurzívnej procedúre VB algoritmu sa z určitého B$^+$-stromu získavajú kľúče *k*, pomocou ktorých je pomocou ukazovateľa *p* odkazované na B$^+$-strom v ďalšej úrovni VB-stromu alebo na pole objektov. Z tohto B$^+$-stromu sa taktiež získavajú dvojice, ale iného tvaru ako pre samostatné B$^+$-stromy. Dvojica vo VB-strome obsahuje ukazovateľ *p* s ohodnotenou hodnotou kľúča *k* podľa fuzzy funkcie. Táto dvojica je implementovaná ako trieda *RatedSonSet*. Miesto ukazovateľa *p* by bolo možné použiť identifikátor B$^+$-stromu vo VB-strome. Takéto riešenie by však vyžadovalo pri každom prístupe do tohto B$^+$-stromu prejsť VB-stromom od jeho koreňa až po požadovaný B$^+$-strom.

Kvôli rozdielnemu tvaru dvojíc získavaných z B$^+$-stromov vo VB-strome sú v triede *FuzzyFunction* implementované rozdielne metódy pre vyhľadanie aktuálnych dvojíc (metóda *activateWaysInVB*) na začiatku získavania dvojíc a pre získavanie dvojíc (metóda *getRatedSonSet*).

**Pohyb v listovej úrovni B$^+$-stromu**

Pri získavaní kľúčov (dvojíc) z B$^+$-stromu v zostupnom poradí podľa ohodnotenia fuzzy funkciou sa využíva pomocná trieda *ValueInTree*. Pri postupovaní listovou úrovňou B$^+$-stromu podľa ciest sa v tejto triede uchováva pozícia v listovej úrovni B$^+$-stromu, ktorá je jednoznačne určená konkrétnym kľúčom.

V tiriede *BplusTree* sú kvôli získavaniu kľúčov implementované metódy, ktoré dokážu nájsť požadovanú pozíciu v listovej úrovni B$^+$-stromu. Metóda *firstLeft* vyhľadá pre zadanú hodnotu prvú pozíciu v listovej úrovni, ktorej kľúč je menší alebo rovný ako zadaná hodnota. Pre postupovanie v listovej úrovni smerom doľava sa používa metóda *nextLeft*, ktorá pre zadanú pozíciu vráti ľavej susednú pozíciu v listovej úrovni, teda triedu *ValueInTree*. Podobne pre pohyb v listovej úrovni smerom doprava sa používajú metódy *firstRight nextRight*.

## 8.6 Top-k algoritmy

Po zadaní užívateľských preferencií je možné v grafickom rozhraní TreeTopK nástroja nastaviť počet *K* požadovaných objektov, požadovaný algoritmus, typ agregačnej funkcie a tvar výsledkov. Pre výpočet top-k algoritmov trieda *DatabaseInterface* obsahuje metódu *findTopK*, v ktorej sa podľa užívateľských požiadaviek pripravia vstupy pre top-k algoritmus. Následne sa spustí top-k algoritmus a nájde identifikátory *K* najlepších objektov spolu s ich globálnym ohodnotením. V prípade ak užívateľ požaduje zobraziť kompletné informácie o nájdených objektoch, podľa identifikátorov nájdených objektov sa dotiahnu tieto dáta z relačnej databázy.

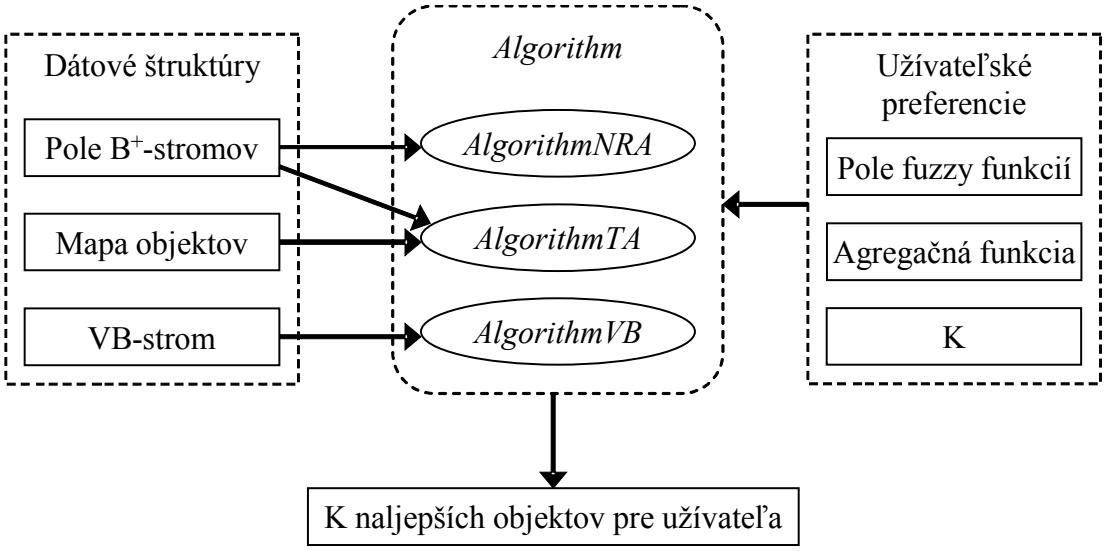
### 8.6.1 Trieda Algorithm

Abstraktná trieda *Algorithm* vykonáva vlastný top-k algoritmus. Pri inicializácii tejto triedy sa načítajú potrebné dátové štruktúry pre výpočet top-k algoritmu. Trieda *Algorithm* obsahuje metódu *findTopK* na vyhľadanie *K* najlepších objektov podľa lokálnych preferencií a podľa agregačnej funkcie. Táto metóda vracia zoznam identifikátorov *K* najlepších objektov spolu s ich globálnym ohodnotením.

Top-k algoritmy počas svojho výpočtu potrebujú uchovávať množinu aktuálnych *K* najlepších objektov v zozname *T*. Navyše zoznam *T* sa musí udržiavať počas výpočtu zostupne zotriedený. V prvotnej verzii implementácie práca so zoznamom *T* podstatne zvyšovala čas výpočtu. Preto vznikla potreba nového modelu zoznamu *T*, ktorý by umožňoval vyhľadať požadovaný objekt, vloženie objektu na správne miesto v zozname a prístup ku *K*-temu najlepšiemu objektu s menšími časovými nárokmi.

Zoznam *T* je preto implementovaný tak, že objekty sú uložené v triede *ArrayList* a zároveň aj v triede *HashMap*. Uloženie v triede *ArrayList* umožňuje udržiavať objekty zostupne zotriedené podľa ich ohodnotenia. Uloženie v triede *HashMap* umožňuje rýchlo pristúpiť k požadovanému objektu. Pre vloženie objektu na správne miesto v zozname *T* je sa používa logaritmické vyhľadávanie pozície v zozname *T*. Na túto pozíciu sa objekt následne vloží. Zoznam *T* sa takto udržuje zotriedený.

Konkrétne implementácie top-k algoritmov popisovaných v tejto práci obsahujú triedy *AlgorithmTA*, *AlgorithmNRA* a *AlgorithmVB*.




*Obrázok 8.4.1, Vstupy top-k algoritmov*

Obrázok 8.4.1 znázorňuje, aké dátové štruktúry používajú implementované top-k algoritmy pri hľadaní *K* najlepších objektov podľa užívateľských preferencií.

Pre potreby testovania trieda *Algorithm* obsahuje metódy *getRunTime* a *getAccessCount*, ktoré po skončení výpočtu top-k algoritmu vrátia čas strávený výpočtom algoritmu a počet vykonaných prístupov do dátových štruktúr. Jednotlivé zložky počtu prístupov je možné po skončení algoritmu získať pomocou ďalších metód *getAccessInTreeArray*, *getAccessInObjectMap*, *getAccessInVBTreeKey*, *getAccessObject*.

### 8.6.2 TA algoritmus

V triede *AlgorithmTA* je implementovaný TA algoritmus podľa kapitoly 3.2.1. Algoritmus TA používa dve dátové štruktúry. Pre sekvenčný prístup do zostupne zotriedených zoznamov používa algoritmus pole $B^+$-stromov. Keďže tieto $B^+$-stromy neumožňujú priamy prístup ku hodnote atribútu objekty podľa identifikátoru objektu, na priamy prístup do zoznamov algoritmus využíva mapu objektov, v ktorej sú mapované všetky hodnoty atribútov objektov podľa identifikátorov objektov. Táto štruktúra simuluje priamy prístup do zoznamov Faginovho algoritmu.

V algoritme TA sú zavedené lokálne užívateľské preferencie, preto algoritmus používa model zoznamov založený na $B^+$-stromoch. Z každého $B^+$-stromu sa tak získavajú objekty aj s ohodnotením podľa príslušnej užívateľskej fuzzy funkcie v zostupnom poradí vzhľadom na túto fuzzy funkciu. Získavanie objektov z jednotlivých $B^+$-stromov určuje heuristika $H_{TA}$ (viz kapitola 3.2), podľa ktorej sa pravidelne získavajú objekty zo všetkých $B^+$-stromov.

### 8.6.3 NRA algoritmus

Algoritmus 3P-NRA (viz kapitola 3.2.3) vykoná rovnaký počet prístupov ako algoritmus NRA s menšími časovými nárokmi. Pre kvalitnejšiu implementáciu bol preto použitý 3P-NRA algoritmus. Používa iba sekvenčný prístup do zoznamov, preto počas svojho výpočtu používa ako dátové štruktúry iba pole $B^+$-stromov, podobne ako algoritmus TA.

Na to aby pôvodný algoritmus 3P-NRA vyhľadal $K$ najlepších objektov, nemusí poznať všetky hodnoty atribútov nájdených objektov. Môže sa stať, že nájdené objekty nie sú presne ohodnotené. Tento problém je možné vyriešiť doplnením fázy IV (viz kapitola 3.2.3.1). Algoritmus 3P-NRA doplnený o fázu IV je implementovaný v triede *AlgorithmNRA*, ktorý nájde $K$ najlepších objektov aj s ich presným globálnym ohodnotením.

NRA algoritmus na získavanie objektov používa heuristiku $H_{TA}$ iba vo fáze I, pokiaľ prahová hodnota neklesne natoľko, že je menšia ako ohodnotenie $K$-teho najlepšieho objektu v zozname aktuálne najlepších objektov $T$. Vo fáze III algoritmus 3P-NRA používa heuristiku označovanú ako *holes*, podľa ktorej sa získavajú objekty takého *i-t*ého $B^+$-stromu, že niektorému z prvých $K$ objektov zoznamu $T$ chýba *i-t*á hodnota atribútu. Heuristika *holes* je podľa práce [4] pre fázu III algoritmu 3P-NRA najefektívnejšia vzhľadom na počet prístupov. Vo fáze IV sa používa na získavanie objektov taktiež heuristika *holes*, pretože vo fáze IV sa z $B^+$-stromov získavajú už len chýbajúce hodnoty atribútov $K$ najlepších objektov.

### 8.6.4 VB algoritmus

V triede *AlgorithmVB* je implementovaný VB algoritmus podľa kapitoly 7.3. Ako dátovú štruktúru používa iba VB-strom. V hlavnej procedúre sa spustí rekurzívna procedúra na $B^+$-strom v prvej úrovni. Potom sa pomocou rekurzívnej procedúry prehľadáva VB strom do hĺbky. Rekurzívna procedúra má ako vstupný parameter identifikátor $B^+$-stromu. Pre rýchlejší prístup do tohto stromu a pre zrýchlenie výpočtu má rekurzívna procedúra ako ďalšie dva vstupné parametre úroveň vo VB-strome a ukazovateľ na tento $B^+$-strom.

Pre zavedenie lokálnych užívateľských preferencií VB algoritmus používa model zoznamu založený na $B^+$-strome. V každej rekurzívnej procedúre sa nové kľúče získavajú podľa príslušnej fuzzy funkcie, ktorá sa vyberie podľa úrovne v ktorej sa $B^+$-strom nachádza. Takto sa z $B^+$-stromu získavajú kľúče v zostupnom poradí vzhľadom ku fuzzy funkcii. Pri získaní identifikátora objektu sú už známe všetky hodnoty atribútov objektu. Získané hodnoty atribútov sú už ohodnotené podľa lokálnych preferencií užívateľa, preto sa môže pre novozískaný identifikátor objektu vypočítať globálne ohodnotenie objektu podľa užívateľských preferencií.

## 8.7 Výsledok implementácie

V tejto práci sa podarilo vytvoriť TreeTopK nástroj, v ktorom sú implementované top-k algoritmy TA, NRA, a navrhnutý VB algoritmus, ktoré vyhľadávajú *K* najlepších objektov podľa zadaných užívateľských preferencií. TreeTopK nástroj umožňuje testovať implementované algoritmy.

Ako vedľajší produkt vzniklo grafické rozhranie TreeTopK nástroja, ktoré obsahuje taktiež plnú funkčnosť TreeTopK nástroja a pre komunikáciu s užívateľmi používa štandardné grafické rozhranie (windows aplikácia). Táto aplikácia sa dokáže pripojiť na relačnú databázu MySQL a použiť tabuľky tejto databázy ako dáta o objektoch. Pomocou grafického rozhrania môže užívateľ (aj viacero užívateľov) zadávať užívateľské preferencie a vyhľadávať *K* najlepších objektov. Bližšie informácie sú uvedené v prílohe B tejto práce, ktorá obsahuje užívateľskú dokumentáciu aplikácie TreeTopK.

# 9 Testovanie top-k algoritmov

V tejto práci boli implementované top-k algoritmy, ktoré dokážu v stromových dátových štruktúrach v pamäti vyhľadať $K$ najlepších objektov podľa užívateľských preferencií. V týchto dátových štruktúrach je pritom uložených $N$ objektov s hodnotami ich $m$ ohodnocovaných atribútov. Získanie jednej hodnoty atribútu jedného objektu sa v tejto práci bude chápať ako *jeden prístup*. Implementované top-k algoritmy dokážu nájsť $K$ najlepších objektov bez vykonania $N \cdot m$ prístupov.

Úlohou tejto kapitoly je skúmať práve tento počet prístupov. Okrem počtu prístupov sa bude skúmať čas, ktorý top-k algoritmy potrebujú pre vyhľadanie $K$ najlepších objektov.

Čas strávený výpočtom top-k algoritmu a počet vykonaných prístupov do dátových štruktúr v pamäti sa budú skúmať:

- v závislosti na konkrétnych dátach o objektoch,
- v závislosti na použitom top-k algoritme,
- pre rôzny počet $K$ najlepších hľadaných objektov,
- pre rôzne nastavenia užívateľských preferencií.

## 9.1 Spôsob merania

Pre potreby testovania obsahuje trieda *Algorithm* metódy (viz kapitola 8.4.1), ktoré po skončení výpočtu top-k algoritmu vrátia počet vykonaných prístupov a čas strávený výpočtom algoritmu.

### 9.1.1 Čas výpočtu

Čas, ktorý potrebuje pre svoj výpočet konkrétny top-k algoritmus, je závislý na prostredí, v ktorom je algoritmus spustený. Testovanie top-k nástroja prebiehalo na počítači s 1GB RAM, s procesorom AMD Athlon 64 X2 2.00GHz a operačným systémom MS Windows XP. V prípade počítač Pri použití počítača s inými parametrami by samozrejme boli výsledky odlišné, preto namerané časy výpočtov algoritmov budú slúžiť len na ich vzájomné porovnanie.

*Poznámka*: Počas testovania top-k nástroj potreboval na vytvorenie dátových štruktúr v pamäti približne od 20MB až po 200MB operačnej pamäte podľa toho, aká tabuľka s dátami o objektoch bola v top-k nástroji použitá ako zdroj dát počas testovania. Táto práca sa nezoberá veľkosťou vytvorených dátových štruktúr a s tým súvisiacimi pamäťovými nárokmi top-k algoritmov.

### 9.1.2 Počet prístupov

Algoritmy implementované v top-k nástroji využívajú tri dátové štruktúry v pamäti, z ktorých sa získavajú dáta o objektoch. Pri použití užívateľských preferencií sa z týchto dátových štruktúr získavajú identifikátory objektov a lokálne ohodnotené hodnoty atribútov objektov.

### Pole m $B^+$-stromov

Prvou štruktúrou je pole *m* $B^+$-stromov, ktorú využívajú pri výpočte top-k algoritmy TA a NRA na získavanie dvojíc identifikátor objektu s lokálne ohodnotenou hodnotou určitého atribútu. Pred získavaním dvojíc z požadovaného $B^+$-stromu podľa fuzzy funkcie je na začiatku potrebné podľa ciest vyhľadať množinu aktuálnych objektov (dvojíc). Potom sa už získavajú zo stromu dvojice tak, že na niektorej z ciest sa vyberie najlepšia dvojica a namiesto nej sa z $B^+$-stromu získa nová dvojica. Získanie dvojice z $B^+$-stromu bude v tomto prípade chápané ako vykonanie jedného prístupu.

Top-k algoritmy TA a NRA používajú *m* $B^+$-stromov ako *m* zostupne zotriedených zoznamov, do ktorých sekvenčne pristupujú podľa ciest príslušných *m* fuzzy funkcií. Na konci týchto algoritmov sa dá zistiť, koľko dvojíc sa získalo počas výpočtu z ktorého zoznamu. Je možné spočítať počet vykonaných prístupov do zoznamov. Pre celkový počet prístupov do $B^+$-stromov sa ale musia ešte pripočítať dvojice, ktoré sa získali v $B^+$-stromoch počas postupovania ich listovou úrovňou podľa ciest príslušných fuzzy funkcií a neboli získané pri výpočte. Taktiež je možné spočítať tieto dvojice. Dvojíc bude maximálne toľko, koľko je ciest vo všetkých fuzzy funkciách.

Takýmto spôsobom je možné spočítať počet prístupov do poľa *m* $B^+$-stromov, ktoré vykonajú pri svojom výpočte top-k algoritmy TA a NRA. Keďže algoritmus NRA používa pre svoj výpočet len pole *m* $B^+$-stromov takto sa spočíta jeho celkový počet vykonaných prístupov.

### Mapa objektov

Algoritmus TA potrebuje počas výpočtu pristupovať podľa identifikátorov objektu ku hodnotám ich atribútov. Podľa pôvodného algoritmu by mali zostupne zotriedené zoznamy umožňovať aj priamy prístup. V použitom modele $B^+$-stromov však priamy prístup nieje možný. Preto pre novozískaný objekt získaný z poľa *m* $B^+$-stromov sa musia chýbajúce hodnoty atribútov získať podľa identifikátoru objektu z mapy objektov. Keďže novozískaný objekt sa získal z niektorého $B^+$-stromu aj s hodnotou jedného atribútu, z mapy objektov je potrebné získať ďalších *m* + 1 hodnôt atribútov, preto sa prístup do mapy objektov bude počítať ako vykonanie *m* + 1 prístupov.

Celkový počet prístupov, ktoré vykoná algoritmus TA počas svojho výpočtu sa dostane sčítaním počtu prístupov do p*oľa m* $B^+$-stromov a počtu prístupov do mapy objektov.

### VB-strom

VB Algoritmus potrebuje počas svojho výpočtu pristupovať iba do VB-stromu. Pri získaní identifikátora objektu z VB-stromu sú už všetky hodnoty atribútov známe, preto sa nedá v tomto prípade chápať jeden prístup ako získanie dvojice (identifikátor objektu a hodnota jedného atribútu). Vo VB-strome sa jeden prístup bude chápať ako získanie jedného kľúča z $B^+$-stromu v rekurzívnej procedúre alebo získanie identifikátora objektu z poľa objektov.

V rekurzívnej procedúre sa z určitého $B^+$-stromu podľa ciest príslušnej fuzzy funkcie získavajú kľúče v zostupnom poradí podľa ich ohodnotenia fuzzy funkciou. Počet takto získaných kľúčov počas výpočtu rekurzívnej procedúry sa dá ľahko spočítať. Pre celkový počet získaných kľúčov z $B^+$-stromu sa musia ešte pripočítať dvojice, ktoré sa získali v $B^+$-strome počas postupovania ich listovou úrovňou podľa ciest fuzzy funkcie a neboli použité pri výpočte.

Po skončení rekurzívnej procedúry je možné zistiť počet kľúčov získaných z príslušného $B^+$-stromu. Pre celkový počet prístupov do VB-stromu je potrebné sčítať počet získaných kľúčov zo všetkých $B^+$-stromov a ku výsledku ešte pripočítať počet všetkých identifikátorov objektov, ktoré boli získané z polí objektov. Takto sa získa celkový počet prístupov, ktoré počas svojho výpočtu vykoná VB algoritmus.

## 9.2 Štruktúra dát

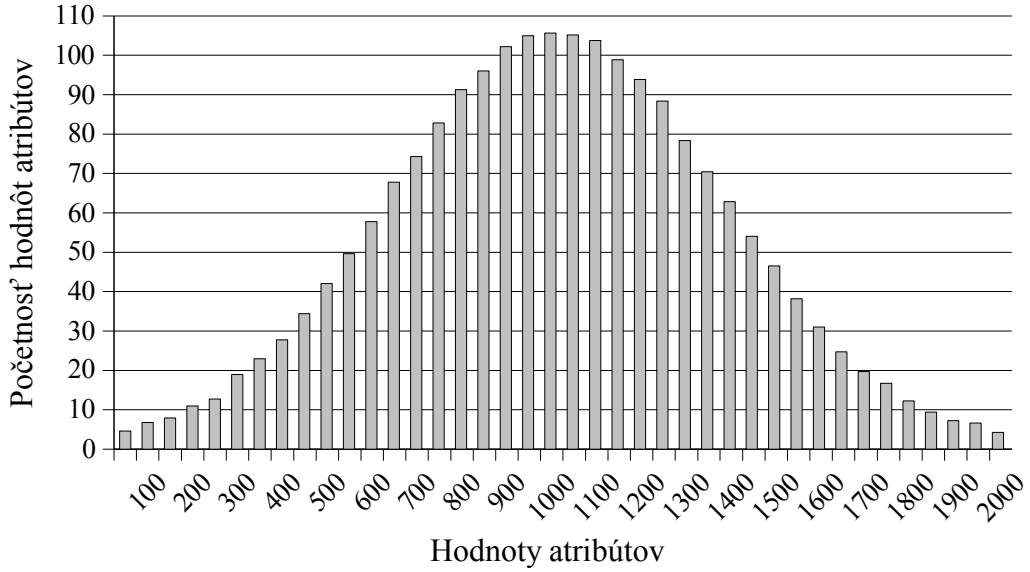
Dáta o objektoch sú uložené v relačnej databáze MySQL ako jedna tabuľka, ktorá má $n$ atribútov. Z týchto $n$ atribútov sa však použije na vytvorenie dátových štruktúr v pamäti len $m + 1$ atribútov ($m \leq n - 1$). Pre všetkých $N$ objektov sa bude podľa hodnôt $m$ atribútov určovať ohodnotenie objektu a hodnoty jedného atribútu budú použité ako identifikátory objektov. Preto pre účely testovania postačí tabuľka, ktorá bude mať $m + 1$ atribútov.

Tak ako v prácach ktoré sa zaoberajú podobnou problematikou, na účely testovania sa použije množina 100 000 objektov, ktoré majú 5 hodnôt atribútov. Testovacie dáta pre top-k algoritmus budú tvorené rôznymi tabuľkami, ktoré budú obsahovať 100 000 objektov (riadky) s 5 + 1 hodnotami atribútov (stĺpce).

Pre potreby testovania sa vytvorili tri tabuľky s rôznym rozložením hodnôt atribútov. V každej z týchto tabuliek je uložených 100 000 objektov s identifikátorom a piatimi celočíselnými hodnotami atribútov z intervalu [1, 2 000]. Hodnoty atribútov objektov sú v každej tabuľke náhodne generované tak, aby sa dosiahlo potrebné rozloženie hodnôt atribútov.

### 9.2.1 Normálne rozloženie

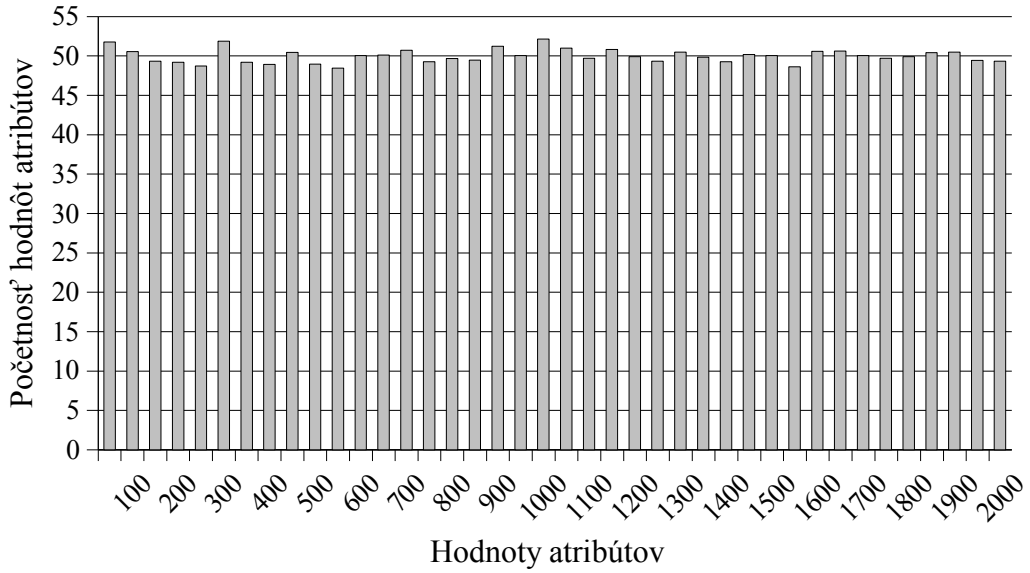
Prvá tabuľka obsahuje dáta o objektoch, kde hodnoty všetkých piatich atribútov objektov sú rozložené normálne. Rozloženie jednotlivých hodnôt jedného z atribútov znázorňuje graf 9.2.1.1.




*Graf 9.2.1*

### 9.2.2 Rovnomerné rozloženie

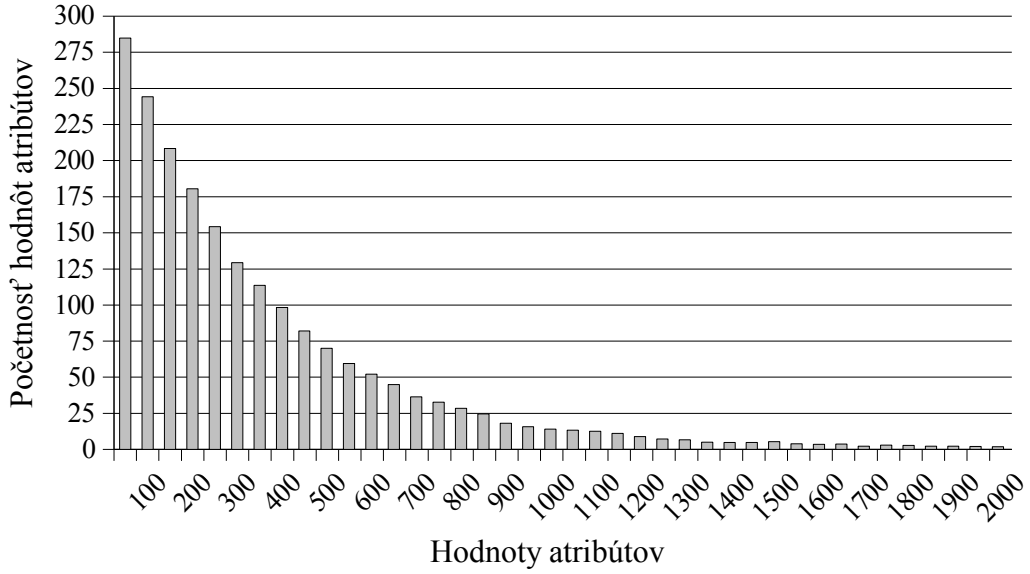
Druhá tabuľka obsahuje dáta o objektoch, kde hodnoty všetkých piatich atribútov objektov sú rozložené rovnomerne. Rozloženie jednotlivých hodnôt jedného z atribútov znázorňuje graf 9.2.2.




*Graf 9.2.2*

### 9.2.3 Exponenciálne rozloženie

Tretia tabuľka obsahuje dáta o objektoch, kde hodnoty všetkých piatich atribútov objektov sú rozložené exponenciálne. Rozloženie jednotlivých hodnôt jedného z atribútov znázorňuje graf 9.2.3.




*Graf 9.2.3*

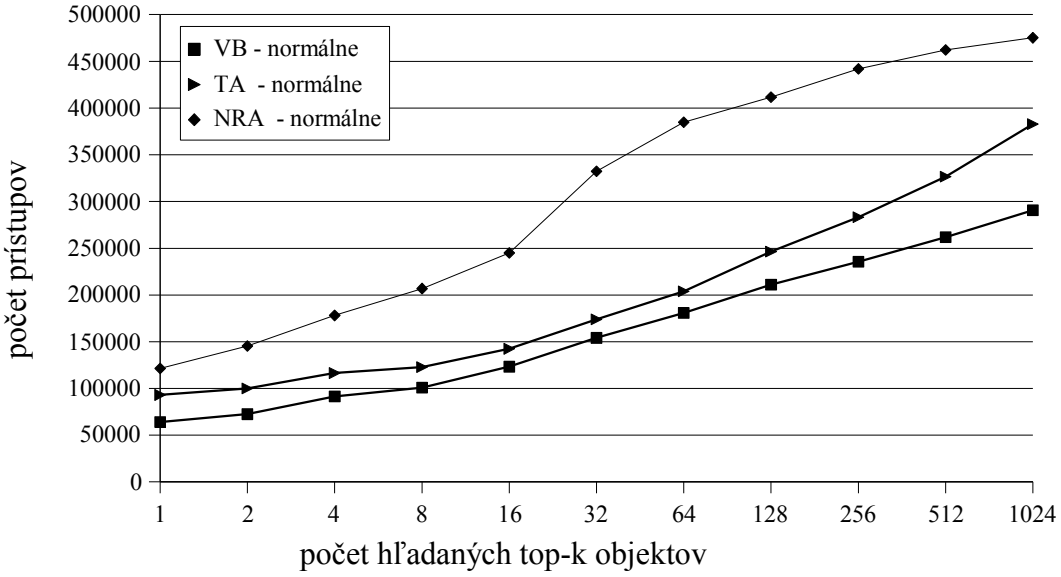
## 9.3 Testovanie rôznych rozložení dát

Pre testovanie algoritmov NRA, TA a VB pre rôzny počet hľadaných top-k objektov sa ako zdroj dát o objektoch použili tri tabuľky (množiny objektov), s normálnym, rovnomerným a exponenciálnym rozložením hodnôt atribútov.

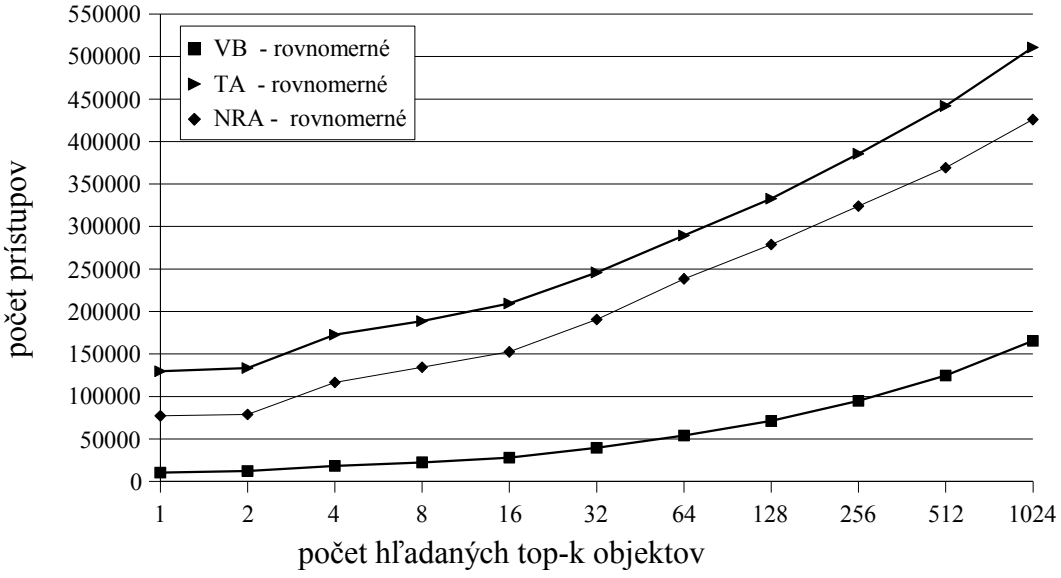
Ako lokálne užívateľské preferencie boli pre všetky atribúty použité fuzzy funkcie tvaru f(x) = x (čím vyššia hodnota atribútu tým lepšie). Ako agregačná funkcia bol použitý aritmetický priemer (vážený priemer s rovnakými váhami všetkých atribútov).

### 9.3.1 Počet prístupov

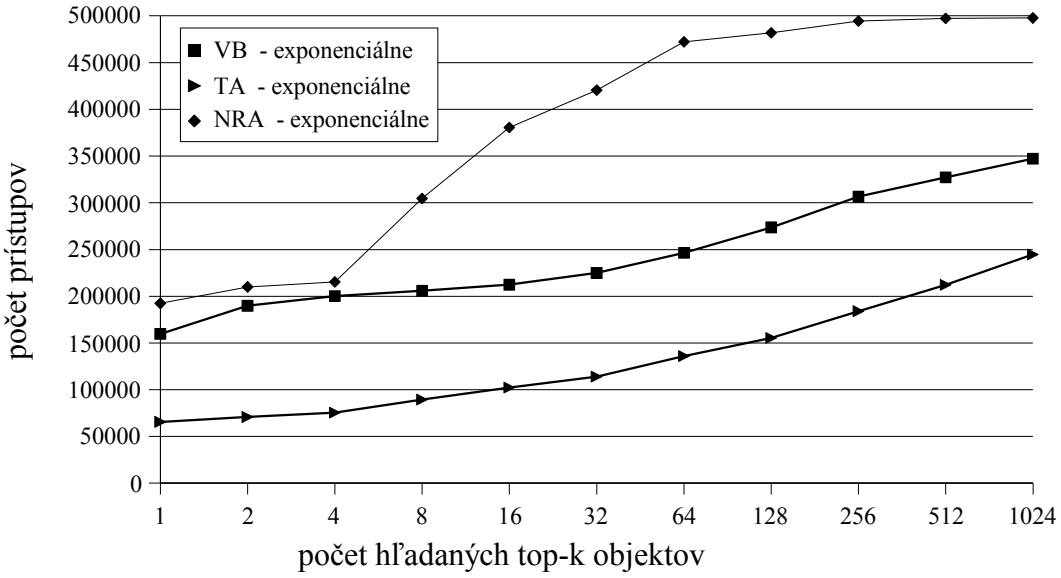
Pre každú z tabuliek sa skúmal počet vykonaných prístupov testovaných algoritmov.




*Graf 9.3.1.a, Normálne rozloženie hodnôt atribútov*




*Graf 9.3.1.b, Rovnomerné rozloženie hodnôt atribútov*




*Graf 9.3.1.c, Exponenciálne rozloženie hodnôt atribútov*

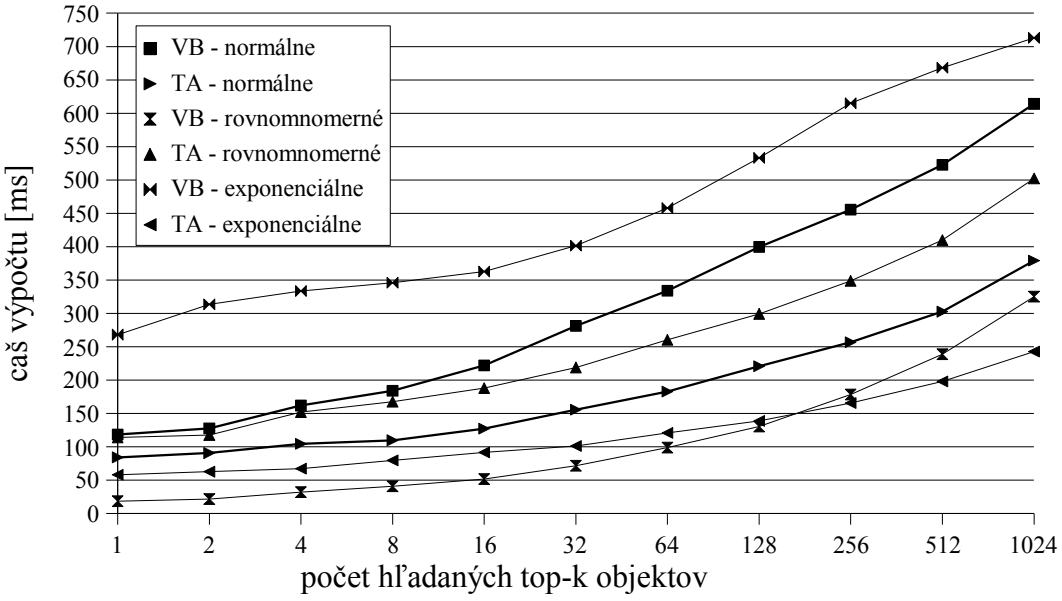
Na grafoch 9.3.1.a, 9.3.1.b, 9.3.1.c sú znázornené výsledky jednotlivých algoritmov. Najlepšie výsledky dosiahol VB algoritmus na dátach s rovnomerným rozložením hodnôt atribútov. Pri hľadaní jedného najlepšieho objektu dokonca vykonal približne 10 krát menej prístupov ako algoritmy TA a NRA. Taktiež pri hľadaní 1024 najlepších objektov potreboval vykonať približne 3 krát menej prístupov ako algoritmy TA a NRA.

Pri exponenciálnom rozložení hodnôt atribútov síce dosiahol algoritmus TA lepšie výsledky ako VB algoritmus, ale tento fakt spôsobuje špecifická konštrukcia algoritmu TA, ktorá je najvhodnejšia práve pre exponenciálne rozloženie hodnôt atribútov. Pri rovnomernom rozložení hodnôt atribútov dokonca algoritmus TA vykonal viac ako 500 000 prístupov (počet objektov krát počet atribútov). Tento fakt zase vyplýva z konštrukcie algoritmu TA, pretože TA algoritmus získava niektoré dvojice objekt s hodnotou atribútu aj viac krát počas jeho výpočtu (najskôr priamym prístupom a potom na ne narazí pri sekvenčnom pristupovaní do zoznamov, viz kapitola 4.2).

### 9.3.2 Čas výpočtu

V tomto teste sa pre každú z tabuliek skúmal čas výpočtu prístupov testovaných algoritmov. Pri testovaní bol nejednoznačne najpomalší algoritmus NRA. Kým TA algoritmus a VB algoritmus pri hľadaní 1024 najlepších objektov dosahovali časy lepšie ako 1s, výpočet algoritmu NRA trval rádovo minúty. Tento fakt je spôsobený častým prepočítavaním W-ohodnotení a B-ohodnotení objektov. Nemá preto zmysel uvádzať časy výpočtu NRA algoritmu. Časy výpočtov algoritmov TA a VB pre rôzny počet hľadaných top-k objektov znázorňuje graf 9.3.2.

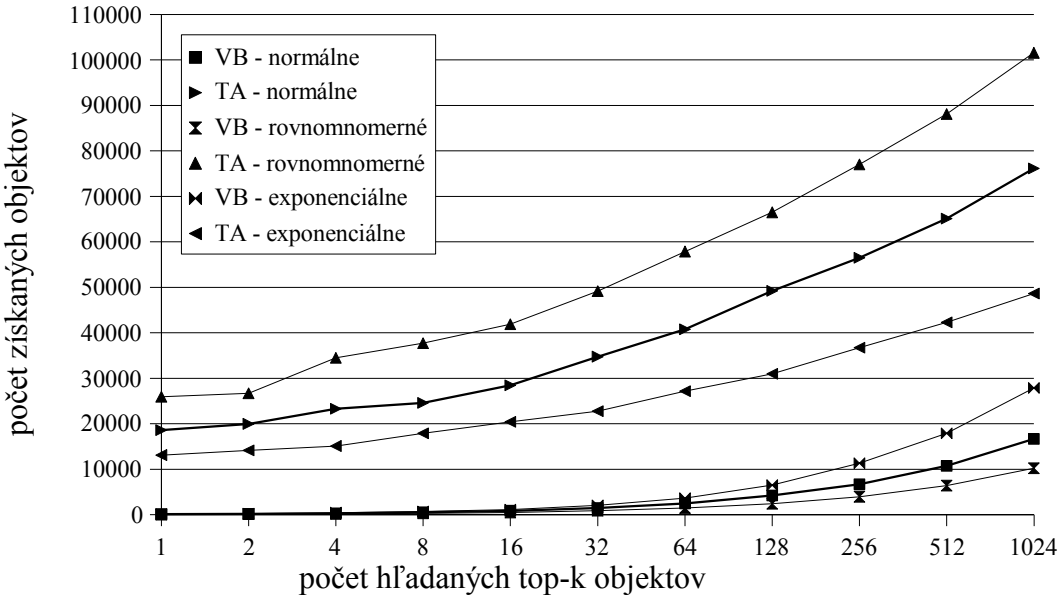
Z výsledkov testu je zrejmé, že najlepšie výsledky dosahuje VB algoritmus pre malý počet hľadaných top-k objektov. Keďže čas výpočtu je výrazne závislý na konkrétnej implementácii a testovacom prostredí, nie porovnávanie časov výpočtov algoritmov až tak podstatné. Napríklad TA algoritmus počas výpočtu získava chýbajúce hodnoty atribútov objektov z asociatívneho poľa. Takéto získanie hodnôt zaberie minimum času výpočtu narozdiel od získavania hodnôt atribútov z VB stromu vo VB algoritme.




*Graf 9.3.2, Čas výpočtu algoritmov VB a TA*

### 9.3.3 Počet získaných objektov

Pri algoritmoch TA a VB je možné spočítať počet získaných objektov (identifikátorov objektov) z príslušných dátových štruktúr. V prípade VB algoritmu sa získanie chápe ako získanie objektu z niektorého poľa objektov vo VB-strome. V algoritme TA je získanie objektu chápané získanie chýbajúcich hodnôt atribútov pre nejaký objekt. Výsledky tohto testu znázorňuje graf 9.3.3.




*Graf 9.3.3, Počet získaných objektov z dátových štruktúr*

Z výsledkov testu je zrejmé, že na vyhľadanie *K* najlepších objektov stačí pri výpočte VB algoritmu získať jednoznačne menej objektov ako v prípade algoritmu TA. Najvýraznejší rozdiel v počte získaných objektov algoritmov TA a VB sa prejavil pri rovnomernom rozložení hodnôt atribútov. Pre vyhľadanie jedného najlepšieho objektu potreboval VB algoritmus dokonca získať vyše 400 krát menej objektov ako algoritmus TA. Z výsledkov je ďalej zrejmé, že VB algoritmus dosahuje mnohonásobne lepšie výsledky ako ostatné testované algoritmy hľadaní *K* najlepších objektov práve vtedy, keď *K* je veľmi malé.

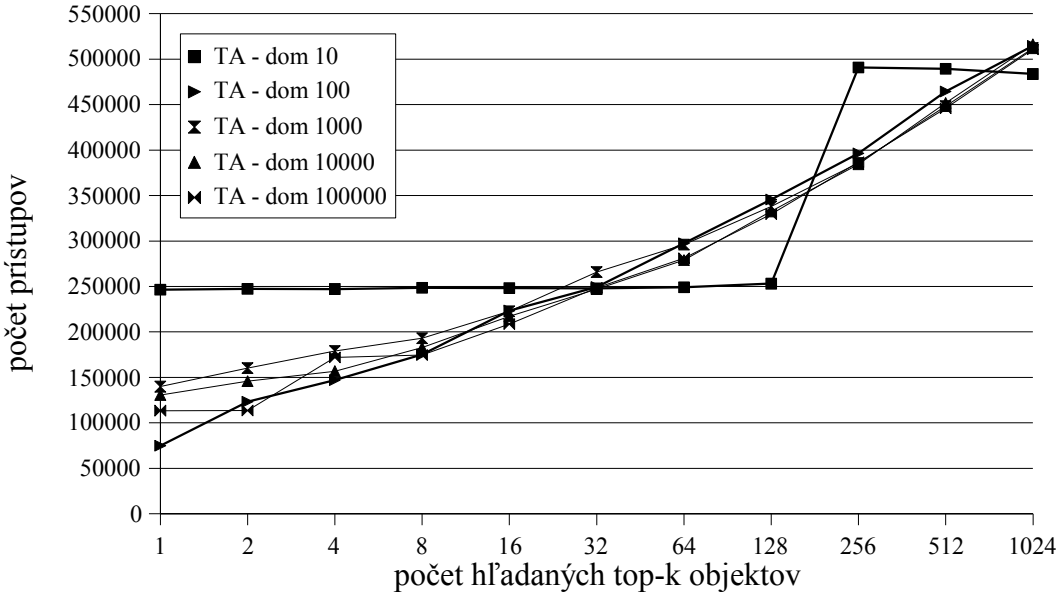
## 9.4 Testovanie veľkostí domén atribútov

V tomto teste boli skúmané výsledky top-k algoritmov vzhľadom na veľkosti domén atribútov. Ako zdroj dát o objektoch použilo viacero tabuliek (množiny objektov). Každá z nich obsahovala 100 000 objektov s identifikátorom a piatimi celočíselnými hodnotami atribútov. Hodnoty atribútov týchto objektov boli náhodne generované tak, aby sa dosiahli požadované veľkosti domén týchto piatich atribútov. Vznikli tak tabuľky s rovnomerným rozložením hodnôt atribútov.

Ako lokálne užívateľské preferencie boli pre všetky atribúty použité fuzzy funkcie tvaru f(x) = x (čím vyššia hodnota atribútu tým lepšie). Ako agregačná funkcia bol použitý aritmetický priemer (vážený priemer s rovnakými váhami všetkých atribútov).

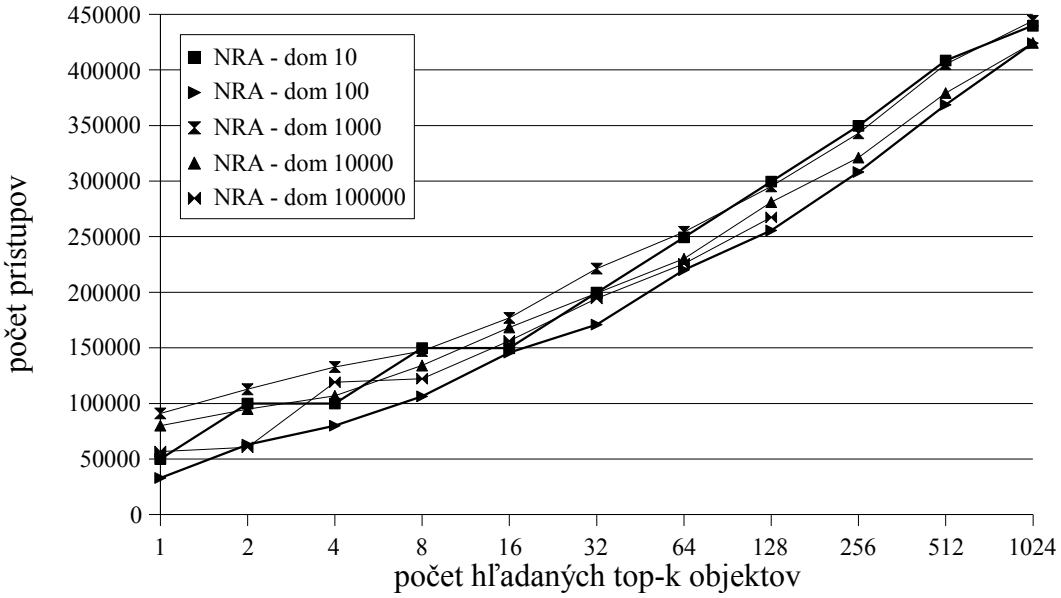
### 9.4.1 Rovnaké veľkosti domén atribútov

Pre tento test sa vytvorilo päť rôznych tabuliek. Každá tabuľka má hodnoty všetkých svojich atribútov náhodne vygenerované v určitom rozmedzí 1 až *w*. Pre test sa vytvorili tabuľky pre *w* rovné 10, 100, 1 000, 10 000 a 100 000. Pre každú tabuľku je veľkosť domén všetkých atribútov rovná *w*. Iba tabuľka pre *w* rovné 100 000 má veľkosti domén atribútov približne 63 150, pretože náhodný generátor čísel nedokáže pre 100 000 objektov vygenerovať vždy rôznu hodnotu atribútu od 1 do 100 000.

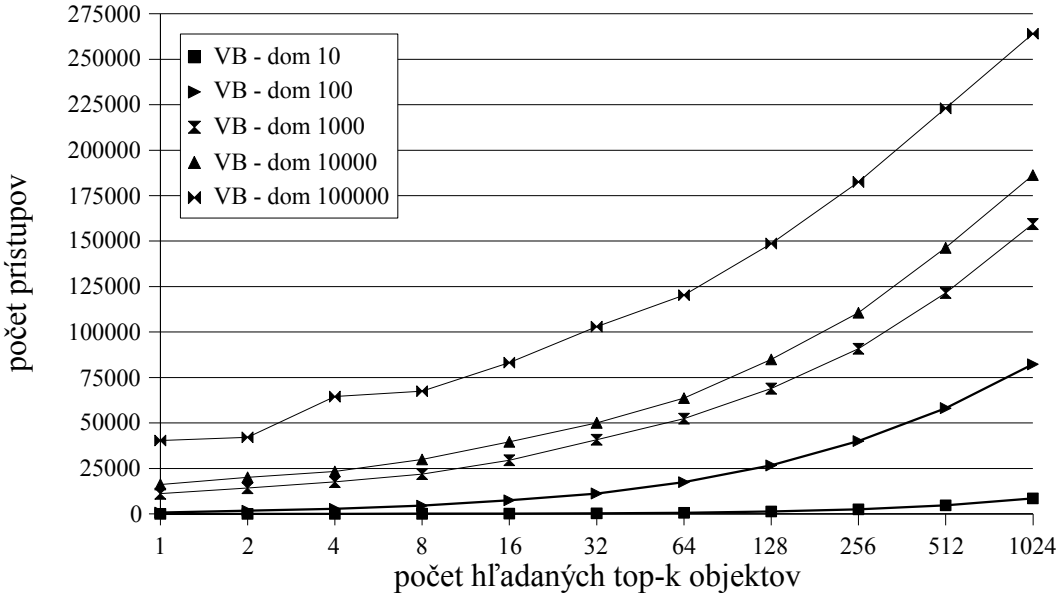



*Graf 9.4.1.a, Počet prístupov vykonaný algoritmom TA*

Na grafoch 9.4.1.a, 9.4.1.b, 9.4.1.c sú znázornené výsledky testov pre jednotlivé algoritmy. Graf 9.4.1.a znázorňuje počet prístupov vykonaných algoritmom TA pre rôzny počet top-k objektov v piatich testovaných množinách objektov. Algoritmus TA dosahoval približne rovnaký počet prístupov pre všetky veľkosti domén atribútov pri hľadaní rovnakého počtu top-k objektov. Odlišnosť výsledku pri množine objektov, kde veľkosti domén atribútov boli 10, je spôsobený konštrukciou TA algoritmu. Je možné vyvodiť záver, že pre hľadanie top-k objektov nie je vhodné použiť algoritmus TA, v prípade ak veľkosti domén atribútov sú veľmi malé.




*Graf 9.4.1.b, Počet prístupov vykonaný algoritmom NRA*




*Graf 9.4.1.c, Počet prístupov vykonaný algoritmom VB*

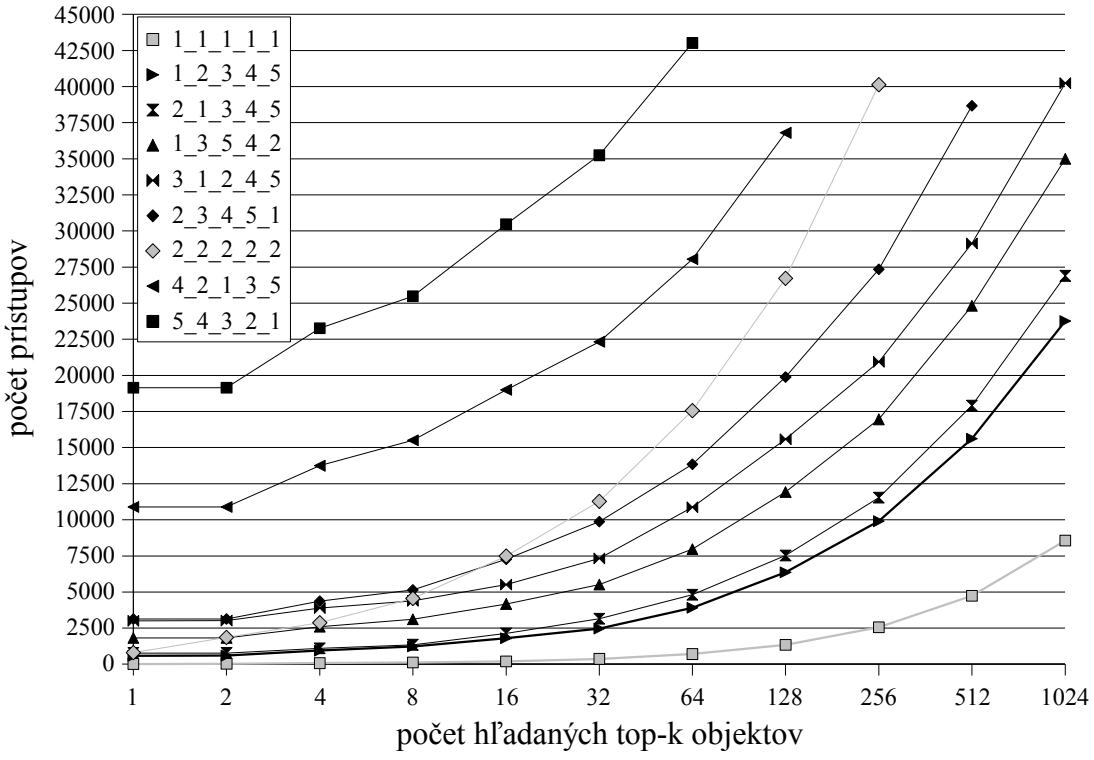
Graf 9.4.1.b znázorňuje výsledky algoritmu NRA v piatich testovaných množinách objektov. Algoritmus NRA taktiež dosahoval približne rovnaký počet prístupov pre všetky veľkosti domén atribútov pri hľadaní rovnakého počtu top-k objektov. V prípade algoritmov TA a NRA teda veľkosť domén atribútov minimálne ovplyvňuje počet prístupov vykonaných počas ich výpočtu.

Na grafe 9.4.1.c sú znázornené výsledku algoritmu VB. Oproti algoritmom TA a NRA je zrejmé, že veľkosť domén atribútov podstatne ovplyvňuje výpočet. VB algoritmus dosahuje pre množiny objektov tým lepšie výsledky, čím sú veľkosti domén atribútov týchto objektov menšie. ktorých veľkosti domén atribútov sú malé.

Napríklad pre malú veľkosť domén (10) vykonal VB algoritmus len 17 prístupov pri hľadaní najlepšieho objektu a pri veľkej veľkosti domén (63 150) potreboval vykonať viac ako 40 000 prístupov. Ďalej pri hľadaní 1024 najlepších objektov pri malej veľkosti domén (10) potreboval vykonať 8 556 prístupov a pri veľkej veľkosti domén (63 150) potreboval vykonať 263 956 prístupov.

## 9.4.2 Rozloženie veľkostí domén atribútov

V predchádzajúcom teste boli použité tabuľky, ktoré mali veľkosti domén všetkých atribútov rovnaké. Pre tento test sa vytvorilo desať rôznych tabuliek s rôznymi rozloženiami veľkostí domén atribútov. Prvá tabuľka bola vytvorená tak, aby veľkosť domény prvého atribútu bola 10 (hodnoty atribútov boli vygenerované v rozmedzí 1 až 10), druhého atribútu $10^2$, tretieho $10^3$, štvrtého $10^4$ a piateho $10^5$. Ostatné tabuľky vznikli z prvej tabuľky zmenou poradia poradia atribútov (zmena poradia stĺpcov). Celkovo by bolo možné vytvoriť 5! = 120 rôznych tabuliek, ale pre účely tohoto testu sa použilo len desať reprezentatívnych tabuliek.




*Graf 9.4.2, Počet prístupov vykonaný algoritmom VB*

Z konštrukcie algoritmov TA a NRA vyplýva, že pre rôzne zoradenie atribútov ich výpočet dosiahne rovnaké výsledky (viz kapitola 4.2 a 4.3). Mierne rozdiely môže spôsobiť len heuristika, podľa ktorej sa vyberajú dvojice (objekt s hodnotou atribútu) z jednotlivých vstupných zoznamov. Preto v sa tomto teste skúmal len algoritmus VB.

Výsledky testu sú znázornené na grafe 9.4.2. Veľkosti domén atribútov sú vyjadrené ako postupnosť exponentov čísla 10. Napríklad pre prvú testovanú tabuľku, označenú ako 1_2_3_4_5, sú veľkosti jej domén porade $10^1$, $10^2$, $10^3$, $10^4$, $10^5$. Medzi výsledkami sú pre väčšiu názornosť uvedené (svetlejšie krivky) aj počty prístupov pre tabuľky z predchádzajúceho testu s veľkosťou všetkých domén atribútov 10 a 100.

Test jednoznačne ukázal, že pre rozloženie veľkostí domén atribútov dosahuje VB algoritmus najlepšie výsledky vtedy, keď veľkosti domén atribútov sú vzostupne radené vzhľadom k poradiu atribútov.

Podľa zvolenej tabuľky s dátami o objektoch sa pred spustením VB algoritmu vytvorí VB-strom, v ktorom sú hodnoty *i*-tého atribútu všetkých objektov uložené ako kľúče v $B^+$-stromoch *i*-tej úrovne VB-stromu. Podľa výsledkov testov je teda pre výpočet VB algoritmus výhodné, ak sú atribúty s menšou veľkosťou domény umiestnené vo vyšších a atribúty s väčšou veľkosťou domény umiestnené v nižších úrovniach VB-stromu.

Vyhľadávané objekty majú väčšinou spojité a aj diskrétne atribúty. Diskrétne atribúty objektov majú typicky malú veľkosť domény, preto je pre VB algoritmus vhodnejšie, ak sú diskrétne atribúty umiestnené vo vyšších úrovniach VB-stromu.

Tieto výsledky testu dokonca potvrdzujú pôvodnú myšlienku, na základe ktorej autori článku [5] navrhli VB-stom ako novú štruktúru pre realizáciu rozsahových dotazov.

## 9.5 Testovanie užívateľských preferencií

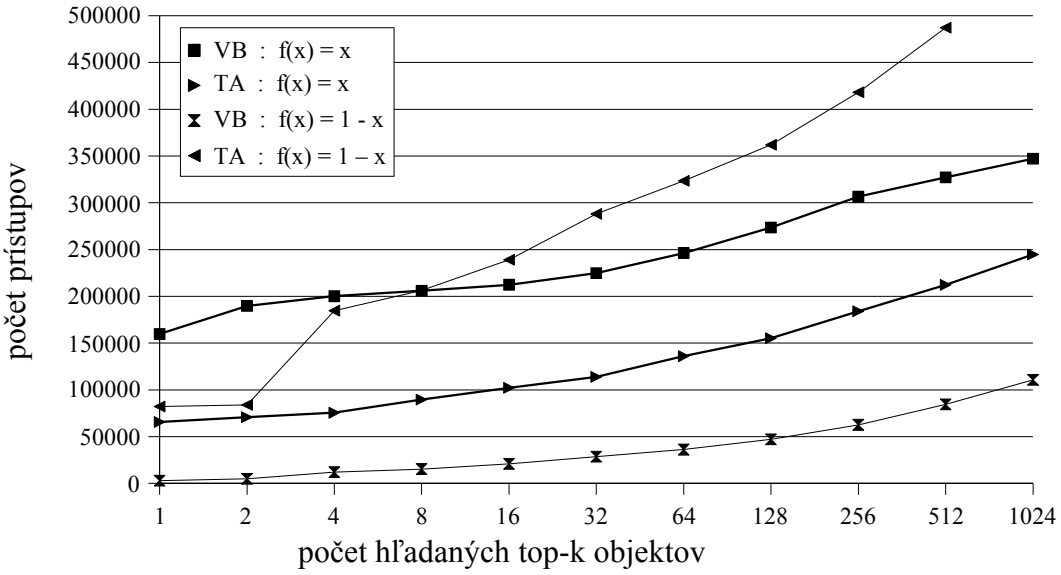
Pred každým vyhľadaním *K* najlepších objektov musí užívateľ nastaviť lokálne preferencie pre jednotlivé atribúty a agregačnú funkciu s váhami jednotlivých atribútov. Nad jednou množinou objektov s viacerými hodnotami atribútov existuje veľa možností ako nastaviť užívateľské preferencie. Táto podkapitola sa zameriava na porovnanie počtu prístupov jednotlivých top-k algoritmov vzhľadom na tieto užívateľské požiadavky.

### 9.5.1 Vplyv zmeny užívateľských preferencií

V predchádzajúcich testoch na tabuľkách s normálnym a rovnomerným rozložením hodnôt atribútov dosahoval algoritmus VB najlepšie výsledky. V prípade exponenciálneho rozloženia dosahoval lepšie výsledky algoritmus TA. Tento test sa preto zaoberal vzťahom lokálnych preferencií ku exponenciálnemu rozloženiu hodnôt atribútov.

Ako lokálne užívateľské preferencie boli vtedy použité fuzzy funkcie tvaru $f(x) = x$ (čím vyššia hodnota atribútu tým lepšie). V tomto teste sa ako lokálne užívateľské preferencie pre všetky atribúty použili fuzzy funkcie opačného tvaru tvaru $f(x) = 1 – x$ (čím nižšia hodnota atribútu tým lepšie pre užívateľa).

Na nasledujúcom grafe 9.5.1 sú znázornené výsledky testu algoritmov VB a TA pri exponenciálnom rozložení hodnôt atribútov pre dve rôzne nastavenia lokálnych preferencií (pre lokálne preferencie tvaru sú výsledky rovnaké ako na grafe 9.3.1.c).




*Graf 9.5.1, Počet prístupov vykonaný algoritmom VB a TA*

Z výsledkov testu je zrejmé, že rozdielne užívateľské preferencie ovplyvňujú výsledky top-k algoritmov pri vyhľadávaní top-k objektov v konkrétnej množine objektov. Pre odlišne nastavené užívateľské preferencie vykonal VB algoritmus mnohonásobne menej prístupov ako TA algoritmus a to hlavne pre malý počet hľadaných top-k objektov.

### 9.5.2 Použitie reálnych dát

Na testovanie užívateľských preferencií sa v tejto práci ďalej použili reálne dáta, na ktorých je možné oveľa prirodzenejšie sledovať závislosti medzi zvolenými užívateľskými preferenciami a výsledkami top-k algoritmov. Pre testovanie bolo potrebné zvoliť vhodnú množinu objektov tak, aby táto množina obsahovala čo najviac objektov rovnakého druhu, aby objekty mali čo najviac atribútov použiteľných na ohodnocovanie týchto objektov, a aby bolo možné túto množinu objektov získať a uložiť v relačnej databáze.

Pre účely testovania sa podarilo získať množinu vyše 8 824 bytov na prenájom v Prahe. Vo výslednej testovanej tabuľke mali tieto byty 19 atribútov, z ktorých bolo možné použiť 9 atribútov na nastavovanie užívateľských preferencií. Atribúty cena za mesiac, poplatky za mesiac a plocha bytu boli celočíselného typu a ostatné atribúty sa použili ako diskrétne atribúty (viz kapitola 8.3.2).

### 9.5.3 Pozorovanie

Po vykonaní mnohých testov sa ukázalo, že testovanie rôznych užívateľských preferencií je veľmi problematické. Výsledky testov sú výrazne závislé na konkrétnej množine prehľadávaných objektov a konkrétnych užívateľských preferenciách. Navyše pre potreby testovania je vo všeobecnosti veľmi problematické zvoliť niekoľko reprezentatívnych nastavení užívateľských preferencií, pre ktoré by sa napríklad vykonali výpočty top-k algoritmov nad rovnakou množinou objektov. Aj keby sa podarilo zvoliť niekoľko typických nastavení, výsledok takého testu by bol veľmi

konkrétny. Napriek tomu je možné vyvodiť niekoľko záverov o vplyve užívateľských preferencií na výsledky top-k algoritmov.

Z konštrukcie použitých top-k algoritmov vyplýva, že výpočet každého z nich skončí v okamihu, keď už je zrejmé, že nie je potrebné vykonávať ďalšie prístupy do použitých stromových dátových štruktúr, pretože už nie je možné nájsť objekty s lepším ohodnotením ako v tom okamihu *K* najlepších objektov. Čím skôr nastane tento okamih, tým lepší výsledok top-k dosiahne (počet vykonaných prístupov, resp. čas výpočtu).

Pre dosiahnutie čo najlepšieho výsledku je preto potrebné aby tento okamih nastal čo najskôr. To je možné dosiahnuť práve vhodnou voľbou užívateľských preferencií. Čím sú užívateľské preferencie presnejšie a optimálnejšie zadané, tým je výsledok top-k algoritmu lepší.

### 9.5.4 Presnosť užívateľských preferencií

Presnosť užívateľských preferencií sa v tomto zmysle chápe ako také nastavenie užívateľských preferencií, vzhľadom k počtu top-k hľadaných objektov, ktoré vedie ku rýchlejšiemu nájdeniu požadovaných objektov.

Nech napríklad užívateľ potrebuje nájsť $K = 5$ najlepších objektov. Ak užívateľ nastaví svoje preferencie tak, že z množiny objektov bude mať práve 5 objektov vysoké ohodnotenie a ohodnotenie ostatných objektov (počnúc šiestym najlepším objektom podľa jeho preferencií) bude nízke, potom dokážu top-k algoritmy nájsť týchto 5 najlepších objektov s veľmi dobrými výsledkami.

V takýchto prípadoch pre malé hodnoty *K* algoritmus TA a hlavne algoritmus VB dosahujú až neuveriteľne dobré výsledky. V krajných prípadoch stačí týmto top-k algoritmom vykonať menej ako 1 % zo všetkých možných prístupov a časy výpočty sú menšie ako 1 ms aj pri množinách obsahujúcich desiatky tisícov objektov.

Z vykonaných testov je zrejmé, že v takýchto prípadoch algoritmus VB dosahuje niekoľkonásobne lepšie výsledky ako algoritmus TA . Algoritmus NRA v tomto prípade svojimi výsledkami zaostáva za algoritmami VB a TA.

### 9.5.5 Optimálnosť užívateľských preferencií

Optimálnosť užívateľských preferencií sa v tomto zmysle chápe ako nastavenie užívateľských preferencií vzhľadom ku konkrétnym hodnotám atribútov hľadaných objektov, ktoré je logické vzhľadom na danú množinu objektov.

Nech napríklad užívateľ potrebuje nájsť $K = 5$ najlepších bytov v Prahe na prenájom (viz kapitola 9.5.2). Ak užívateľ nastaví svoje preferencie tak, že bude preferovať byty s veľkou plochou (300 $m^2$) a malou cenou (5 000 Kč/mesiac), top-k algoritmy budú prehľadávať veľké množstvo bytov, pretože byt s obidvoma preferovanými vlastnosťami asi ťažko existuje. Top-k algoritmy sa budú snažiť hľadať medzi veľkým množstvom objektov kompromis, čiže 5 objektov s najvyšším ohodnotením. Tým pádom top-k algoritmy musia vykonať veľké množstvo prístupov.

Ak však užívateľ nastaví svoje preferencie tak, že bude preferovať byty s primeranou plochou (100 $m^2$) a primeranou cenou (15 000 Kč/mesiac), tak top-k algoritmy dokážu nájsť takýchto 5 bytov veľmi rýchlo, pretože takéto byty sa pravdepodobne budú v množine vyskytovať.

Vykonané testy ukázali že najlepšie výsledky v prípade použitia optimálnych užívateľských preferencií top-k algoritmy skutočne dosahujú veľmi dobré výsledky (počet vykonaných prístupov, resp. čas výpočtu). V tomto prípade taktiež dosahoval VB algoritmus mnohonásobne lepšie výsledky oproti algoritmom TA a NRA.

### 9.5.6 Váhy atribútov

Pri použití niektorých agregačných funkcií, ako napríklad vážený priemer, môže užívateľ nastaviť váhy jednotlivých atribútov. Rôzne nastavenia váh atribútov taktiež ovplyvňujú výsledky top-k algoritmov.

Podľa vykonaných testov je na voľbe váh jednotlivých atribútov najvýraznejšie závislí VB algoritmus. Ukázalo sa, že VB algoritmus dosahuje najlepšie výsledky, keď atribúty ktoré zodpovedajú vyšším úrovniam VB stromu majú priradené vyššie váhy.

Tento záver vyplýva z konštrukcie VB algoritmu (viz kapitola 7.3). Procedúra `searchTopK` v takomto prípade oveľa častejšie končí v $B^+$-stromoch vo vyšších úrovniach VB-stromu. Preto sa nemusí tak často pristupovať do $B^+$-stromov v nižších úrovniach VB-stromu resp. do polí objektov a tým pádom stačí VB algoritmu vykonať malý počet prístupov.

## 9.6 Zhrnutie testov

Vykonané testy preukázali, že implementované algoritmy dokážu nájsť *K* najlepších objektov bez prejdenia všetkých objektov podľa zadaných užívateľských preferencií. Užívateľské preferencie boli pritom zadávané podľa zavedeného modelu užívateľských preferencií (viz kapitola 2.4). Implementované algoritmy, narozdiel od pôvodných Faginových algoritmov, teda dokážu efektívne vyhľadávať top-k objekty aj s podporou lokálnych užívateľských preferencií.

Vykonané testy ďalej preukázali, že VB algoritmus, ktorý sa podarilo vyvinúť vrámci tejto diplomovej práce, je plnohodnotným top-k algoritmom. VB algoritmus dokonca dosahuje až na niekoľko výnimočných prípadov najlepšie výsledky v počte vykonaných prístupov. Pri vhodnom zadaní užívateľských preferencií a pri vhodnom rozložení hodnôt atribútov prehľadávaných objektov dokonca VB algoritmus dosahuje mnohonásobne lepšie výsledky v počte prístupov ako algoritmy TA a NRA.

V prípade niektorých množín objektov je možné vopred predpokladať rozloženie hodnôt atribútov v jednotlivých doménach atribútov. Potom je vhodné upraviť poradie atribútov v jednotlivých úrovniach VB-stromu (viz kapitola 9.4.2). Taktiež je možné predpokladať určité závislosti pri voľbe užívateľských preferencií (viz kapitola 9.5.5). Podľa vykonaných testov (hlavne na reálnymi dátami) je v takýchto prípadoch vhodné použiť VB algoritmus, ktorý hlavne pre malé *K* dosahuje veľmi dobré výsledky v počte vykonaných prístupov (často menej ako 1 % možných prístupov), teda aj v čase výpočtu algoritmu.

# 10 Záver

Cieľom tejto práce bolo implementovať Faginove algoritmy tak, aby bolo možné vyhľadávať *K* najlepších objektov podľa preferencií viacerých užívateľov v spoločnej dátovej štruktúre založenej na B+-stromoch resp. vo VB-strome.

Teoretická časť práce sa najskôr venuje modelovaniu užívateľských preferencií. Lokálne preferencie sú pritom modelované pomocou fuzzy funkcií a globálne pomocou agregačnej funkcie. Potom sa práca zaoberá aplikáciou lokálnych preferencií vo Faginových algoritmoch. Hlavným prínosom tejto časti je nový model vstupných zoznamov Faginových algoritmov, ktorý je založený na indexácii pomocou $B^+$-stromov. Ako prínos práce je možné vnímať aj samotný text práce, ktorý umožní zorientovať sa čitateľovi v problematike preferenčného dotazovania v slovenčine.

Ďalej sa teoretická časť práce zaoberá použitím VB-stromu na vyhľadávanie *K* najlepších objektov. Keďže Faginove algoritmy nie je možné použiť vo VB-strome, bolo potrebné vytvoriť nový algoritmus, ktorý dokáže vo VB-strome vyhľadať *K* najlepších objektov bez prejdenia všetkých objektov, podobne ako Faginove algoritmy. Vyvinutý VB algoritmus je najvýznamnejším prínosom tejto práce.

Praktická časť práce sa zaoberá implementáciou Faginových algoritmov s podporou lokálnych užívateľských preferencií a implementáciou VB algoritmu a ich testovaním. Bol vytvorený nástroj, pomocou ktorého je možné implementované algoritmy testovať na rôznych testovacích dáta. Ako vedľajší produkt implementácie vznikla aj aplikácia TreeTopK, pomocou ktorej je možné interaktívne zadávať užívateľské preferencie a následne vyhľadávať *K* najlepších objektov, testovať použité algoritmy.

Pomocou vykonaných testov sa podarilo ukázať, že VB algoritmus je plnohodnotným top-k algoritmom, ktorý dokonca dosahuje lepšie výsledky ako Faginove algoritmy. Testy ďalej ukázali, že za istých podmienok je možné zlepšiť výsledky implementovaných algoritmov, ako napríklad voľbou vhodných užívateľských preferencií alebo v prípade VB algoritmu vhodným poradím atribútov vo VB-strome.

Motiváciu pre ďalšiu prácu môže byť riešenie top-k problému s podporou lokálnych preferencií priamo v databázovom systéme pomocou použitia Faginových algoritmov alebo pomocou VB algoritmu.

Ako motivácia pre ďalší výskum môžu poslúžiť rôzne heuristiky použité vo Faginových algoritmoch napríklad uvedené v prácach [4] a [9]. Napríklad je možné vo VB-strome sledovať rozloženie $B^+$-stromov v jeho úrovniach alebo sledovať celkovú vyváženosť VB-stromu. Vyvinutím nových heuristík by tak bolo možné optimalizovať VB algoritmus. Taktiež by bolo možné vyvinúť heuristiky vychádzajúce z predpokladaných vlastností množiny objektov uložených vo VB-strome.

Ďalšou motiváciou pre budúci výskum môže byť kombinovanie Faginovho algoritmu s VB algoritmom. Takéto riešenie by mohlo byť napríklad použité v distribuovanom prostredí, kde by sa na jednotlivých serveroch vyhľadávali objekty pomocou VB algoritmov a potom by sa výsledky týchto vyhľadávaní spojili pomocou Faginovho algoritmu, resp. opačne.

# 11 Literatúra


[1] Fagin, R., Lotem, A., Naor, M.: Optimal aggregation algorithms for middleware. Journal of Computer and System Sciences 66 (2003) 614–656.

[2] Pokorný, J., Žemlička, M.: Základy implementace souborů a databází. Skripta UK, Vydavatelství Karolinum, 2004.

[3] Eckhardt, A., Pokorny, J., Vojtas, P.: A system recommending top-k objects for multiple users preference. In Proc. of 2007 IEEE International Conference on Fuzzy Systems, July 24-26, 2007, London, England, pp. 1101-1106.

[4] Gurský, P.: Algoritmy na vyhľadávanie najlepších k objektov bez priameho prístupu. Proceedings of Znalosti 2006, pp. 95-105.

[5] Scheuerman, P., Ouksel, M.: Multidimensional B-trees for associative searching in database systems. Information systems, Vol. 34, No. 2, 1982.

[6] P. Gurský: Towards Better Semantics in the Multifeature Querying. In Proc. Of the Dateso 2006, Czech Republic, April 26-28,2006. Edited by: Vaclav Snasel, Karel Richta, Jaroslav Pokorný Published on CEUR-WS: Vol-176

[7] Vojtáš, P.: Fuzzy logic aggregation for Semantic Web search for the best (top-k) answers, in FLSW - Fuzzy logic and the semantic web, E. Sanchez ed. Capturing Intelligence Series, 1, Elsevier (2006) 341-360

[8] Bayer, R., McCreight, E.: Organization and Maintenance of Large Ordered Indices, Acta Informatica, Vol. 1, Fasc. 3, 1972 pp. 173-189.

[9] Gurský, P., Lencses, R., Vojtáš, P.: Algorithms for user dependent integration of ranked distributed information. Proceedings of TED Conference on e-Government (TCGOV 2005), Pages 123-130, 2005

[10] Chaudhuri, S., Gravano, L., Marian, M.: Optimizing Top-k Selection Queries over Multimedia Repositories. IEEE Trans. On Knowledge and Data Engineering, August 2004 (Vol. 16, No. 8) pp. 992-1009

# A Obsah CD

Priložený CD disk obsahuje všetky súbory potrebné k spusteniu nástroja TreeTopK. CD ďalej obsahuje všetko potrebné na zopakovanie testov, všetky zdrojové kódy a testovacie dáta, a samozrejme aj text diplomovej práce vo formáte pdf.

Štruktúra adresárov CD je nasledujúca:

- `\text`
  - Obsahuje text práce vo formáte pdf, súbor `dip.pdf`.
- `\treetopk`
  - Obsahuje spustiteľný súbor `treetopk.bat`, pomocou ktorého je možné spustiť nástroj TreeTopK (bez argumentov sa spusti grafické rozhranie).
  - Obsahuje konfiguračné súbory použitých testov vo formáte xml.
- `\treetopk\src`
  - Zdrojové kódy aplikácie / nástroja TreeTopK.
- `\treetopk\lib`
  - Použité knižnice vo formáte jar.
- `\treetopk\bin`
  - Obsahuje skompilovanú (spustiteľnú) verziu aplikácie / nástroja TreeTopK.
- `\treetopk\dll`
  - použité dynamické knižnice (Microsoft Windows) vo formáte dll, ktoré sú potebné pre spustenie grafického rozhrania nastroja TreeTopK.
- `\treetopk\doc`
  - Obsahuje Javadoc dokumentáciu zdrojových kódov vo formáte html, ktorú je možné spustiť súborom `index.html`.
- `\data`
  - adresár obsahuje súbory pomocou, ktorých je možné vytvoriť MySQL databázu, ktorá sa v tejto práci použila ako zdroj testovacích dát. (viz súbor `\data\readme.txt`).
- `\data\test`
  - adresár obsahuje priamo obsah databázy test. Tento adresár je možné priamo skopírovať do príslušného adresára MySQL servru.
- `\install\JRE`
  - Adresár obsahuje inštalačné súbory pre JRE - Java Runtime Environment.
- `\install\MySQL`
  - Adresár obsahuje inštalačné súbory pre MySQL server.

# B Užívateľská dokumentácia nástroja TreeTopK

Cieľom tejto práce bolo implementovať top-k algoritmy popisované v tejto diplomovej práci, ktoré vyhľadávajú *K* najlepších objektov, pričom dáta o objektoch sú uložené v relačnej databáze. Cieľom bolo skúmať ich výhody, nevýhody, a porovnať vzájomný vzťah týchto riešení.

Popisované top-k algoritmy aj s podporou užívateľských preferencií sú implementované v nástroji *TreeTopK*. Nástroj TreeTopK umožňuje testovať použité top-k algoritmy. Okrem toho obsahuje aj grafické užívateľské rozhranie v ktorom je možné pohodlne zadávať užívateľské preferencie a vyhľadávať tak *K* najlepších objektov pre rôznych užívateľov.

## B.1 Inštalácia

Pre správnu funkčnosť TreeTopK nástroja je na PC potrebné nainštalovať JAVA prostredie a nainštalovať databázový server MySQL.

### B.1.1 Java

Pre potreby nástroja je potrebné nainštalovať na počítači JRE - Java Runtime Environment, prostredie určené na spúšťanie JAVA programov. CD obdsahuje súbor

`\install\JRE\jre-1_5_0_15-windows-i586-p.exe`

pomocou ktorého je možné v prostredí operačného systému MS Windows nainštalovať JRE (prípadne ďalší priložený súbor pre PC s x64 architektúrou). V prípade iného operačného systému je potrebné použiť príslušný inštalačný postup, ktorý je možné nájsť na adrese `http://java.sun.com/javase/`.

### B.1.2 MySQL

Nástroj TreeTopK používa ako zdroj dát databázu MySQL. V prípade ak používateľ nástroja TreeTopK nemá prístup k databáze MySQL, je potrebné nainštalovať databázový server na použitom PC. Priložené CD obsahuje inštalačný súbor

`\install\JRE\mysql-5.0.45-win32-Setup.exe`,

pomocou ktorého je možné v prostredí operačného systému MS Windows nainštalovať databázový server MySQL (CD obsahuje aj súbor pre PC s x64 architektúrou). Počas inštalácie je vhodné zadať „Standard Configuration“ a neupravovať bespečnostne nastavenia „Modify Security Settings“ (štandardne je prístupové meno „root“ a heslo „prázdne slovo“). V prípade iného operačného systému je potrebné použiť príslušný inštalačný postup, ktorý je možné nájsť na adrese `http://www.mysql.com/`.

Po nainštalovaní serveru MySQL je nutné pre potreby testovania nástroja TreeTopK vytvoriť testovacie dáta. Testovacie dáta tvoria jednu databázu „`test`“. Na CD je súbor

`\data\test-export.sql`,

ktorý obsahuje SQL skript. Pre vytvorenie testovacej databázy postačí, ak sa tento skript spustí v MySQL serveri (viz `http://dev.mysql.com/doc/`).

Testovaciu databázu je možné vytvoriť v MySQL serveri aj „rýchlejším“ spôsobom. V tomto prípade postačí adresár `\data\test` z priloženého CD aj s jeho obsahom nakopírovať do príslušného adresára MySQL serveru (štandardne je to adresár `c:\Program Files\MySQL\MySQL Server 5.0\data\` ).

CD obsahuje aj súbor `\data\CREATE.SQL`, v ktorom je štruktúra testovaných tabuliek a v `\data\tab-delimiter` sú exportované dáta týchto tabuliek.

## B.2 Testovacie rozhranie

Na spustenie nástroj TreeTopK v testovacom rozhraní, je vhodné skopírovať obsah adresára `\treetopk` na pevný disk, pretože nástroj potrebuje zapisovať výsledky testov do súboru. Pre testovacie rozhranie je potrebné spustiť súbor `treetopk.bat` s dvoma parametrami. Prvý parameter je názov konfiguračného xml súboru s nastaveniami cestu a druhý parameter je názov výstupného súboru, do ktorého sa zapíšu výsledky testov. Príklad:

```
treetopk.bat settings.xml output.txt
```

### B.2.1 Konfiguračný súbor

Pre účely testovania umožňuje nástroj TreeTopK načítať nastavenie požadovaného testu zo súboru xml. Konfiguračný súbor má nasledujúcu štruktúru:

```
<Settings>
  <Database>
        <Type>MySQL</Type>
        <Host>localhost</Host>
        <Port>3306</Port>
        <Name>test</Name>
        <Username>root</Username>
        <Password></Password>
  </Database>
  <Tables>
     <Table>test_data_nor</Table>
     <Table>test_data_reg</Table>
     <Table>test_data_exp</Table>
  </Tables>
  <TopKs>
     <K>1</K>
     <K>10</K>
     <K>100</K>
  </TopKs>
  <Algorithms>
     <Algorithm>VB</Algorithm>
     <Algorithm>TA</Algorithm>
     <Algorithm>NRA</Algorithm>
  </Algorithms>
  <Aggregations>
    <Aggregation>Weighted avg.</Aggregation>
    <Aggregation>Euclid distance</Aggregation>
    <Aggregation>Minimum</Aggregation>
    <Aggregation>Maximum</Aggregation>
  </Aggregations>
  <Multiplicator>1</Multiplicator>
</Settings>
```

Ako je na prvý pohľad zrejmé, značka `Database` a jej podznačky určujú konfiguráciu pripojenia na databázový server MySQL.

Značka `Tables` a jej podznačky určujú, nad ktorými tabuľkami príslušnej databázy sa majú spustiť testované algoritmy. Aký počet top-k objektov majú tieto algoritmy hľadať určuje značka `TopKs` a jej podznačky. Nakoniec značka `Algorithms` a jej podznačky určujú, ktoré algoritmu majú byť testované.

Konfiguračný súbor ďalej obsahuje značku `Aggregations`, ktorá podznačkami určuje, pre aké typy agregačných funkcií sa majú požadované algoritmy spustiť. Nakoniec konfiguračný súbor obsahuje značku `Multiplicator`. Tá je doplnená kvôli potrebám testovania času výpočtu algoritmov. Určuje koľko krát sa má konkrétne spustenie algoritmu zopakovať. Potom sa čas výpočtu algoritmu určí ako priemer týchto opakovaní výpočtu.

Samotné testovanie prebieha tak, že sa v každej požadovanej tabuľke spustí každý požadovaný algoritmus s každou agregačnou funkciou pre každý počet top-k hľadaných objektov. Po každom takomto výpočte top-k algoritmu sa zapíše jeho výsledok do výstupného súboru ako jeden riadok. Riadky vyzerajú nasledovne:

| | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| VB | 1 | 100000 | 5 | 64022 | 0 | 0 | 63893 | 129 | 206940267 | 1 |
| VB | 2 | 100000 | 5 | 72540 | 0 | 0 | 72358 | 182 | 142537669 | 1 |
| VB | 8 | 100000 | 5 | 100834 | 0 | 0 | 100310 | 524 | 188225598 | 1 |
| TA | 1 | 100000 | 5 | 93067 | 74448 | 18619 | 0 | 18612 | 124074149 | 1 |
| TA | 2 | 100000 | 5 | 99954 | 79956 | 19998 | 0 | 19989 | 91901065 | 1 |
| TA | 4 | 100000 | 5 | 116475 | 93168 | 23307 | 0 | 23292 | 106241157 | 1 |
| NRA | 1 | 100000 | 5 | 121429 | 0 | 121429 | 0 | 0 | 10762914560 | 1 |
| NRA | 2 | 100000 | 5 | 145455 | 0 | 145455 | 0 | 0 | 10345200298 | 1 |
| NRA | 4 | 100000 | 5 | 178114 | 0 | 178114 | 0 | 0 | 17447909720 | 1 |

V jednom riadku výstupného súboru sú uvedené nasledujúce údaje v tomto poradí:

1. názov testovaného algoritmu
2. počet hľadaných top-k objektov
3. počet prehľadávaných objektov
4. počet ohodnocovaných atribútov
5. celkový počet prístupov vykonaný top-k algoritmom
6. počet prístupov vykonaný top-k algoritmom v poli $B^+$-stromov (NRA a TA)
7. počet prístupov vykonaný top-k algoritmom v asociatívnom poli (iba TA)
8. počet získaných kľúčov z $B^+$-stromov vo VB-strome top-k algoritmom (iba VB)
9. počet získaných ()identifikátorov) objektov top-k algoritmom (NRA a VB)
10. čas výpočtu algoritmu v nanosekundách
11. nastavenie multiplikátora, počet opakovaní výpočtu

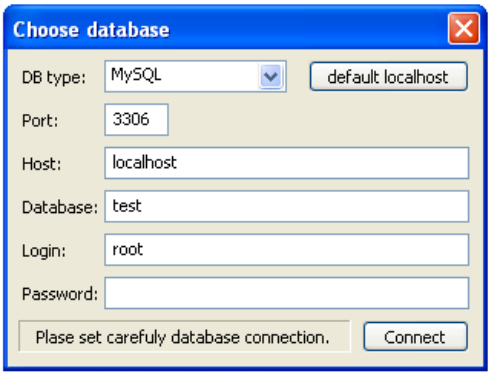
Veľkou výhodou takto formátovaného výstupného súboru je možnosť otvoriť takýto súbor alebo priamo skopírovať a vložiť napríklad do programu MS Excel a ďalej tak pracovať s výsledkami testov top-k algoritmov.

## B.3 Grafické užívateľské rozhranie

Na spustenie TreeTopK nástroja v režime grafického užívateľského rozhrania stačí v adresári `/treetopk` na CD spustiť súbor `treetopk.bat` bez argumentov.

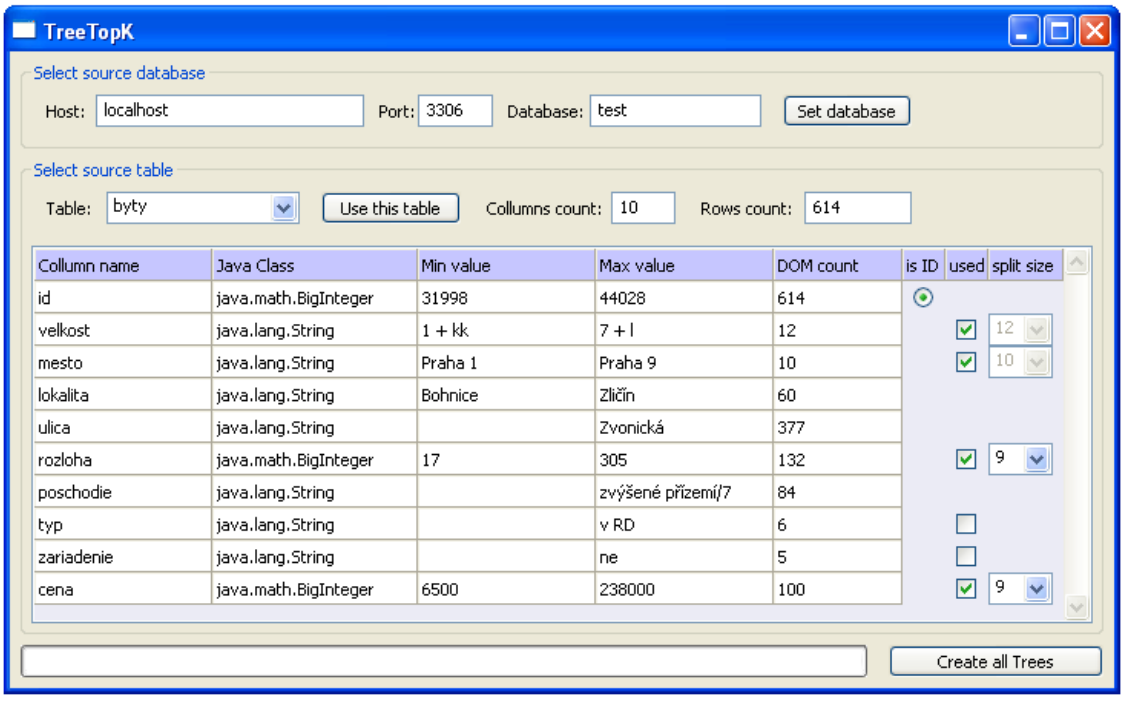
### B.3.1 Nastavenie zdroja dát

Štandardne sa aplikácia TreeTopK pripojí na prednastavenú testovaciu databázu. Ak sa nepodarí na túto databázu pripojiť po stlačení tlačítka „Set database“ (viz obrázok B.3.1.b) je možné nakonfigurovať požadované pripojenie (viz obrázok B.3.1.a).




*Obrázok B.3.1.a, Konfigurácia požadovaného pripojenia*

Po pripojení na požadovanú databázu sa aplikácia TreeTopK načíta všetky názvy tabuliek v tejto databáze. Ako zdroj dát pre vyhľadávanie top-k objektov je potrebné jednu tabuľku vybrať a načítať informácie o jej atribútoch tlačítko „Use this table“




| Collumn name | Java Class | Min value | Max value | DOM count | is ID | used | split size |
|---|---|---|---|---|---|---|---|
| id | java.math.BigInteger | 31998 | 44028 | 614 | ● | | |
| velkost | java.lang.String | 1 + kk | 7 + l | 12 | | ☑ | 12 |
| mesto | java.lang.String | Praha 1 | Praha 9 | 10 | | ☑ | 10 |
| lokalita | java.lang.String | Bohnice | Zličín | 60 | | | |
| ulica | java.lang.String | | Zvonická | 377 | | | |
| rozloha | java.math.BigInteger | 17 | 305 | 132 | | ☑ | 9 |
| poschodie | java.lang.String | | zvýšené přízemí/7 | 84 | | | |
| typ | java.lang.String | | v RD | 6 | | ☐ | |
| zariadenie | java.lang.String | | ne | 5 | | ☐ | |
| cena | java.math.BigInteger | 6500 | 238000 | 100 | | ☑ | 9 |

*Obrázok B.3.1.b, Konfigurácia zdroja dát*

Na obrázku B.3.1.b je znázornená situácia pre tabuľku „byty“ ktorá obsahuje 614 bytov ktoré majú 10 atribútov. Pre každý atribút sa načíta jeho meno, jeho typ (java trieda), maximálna a minimálna hodnota, a veľkosť domény tohto atribútu.

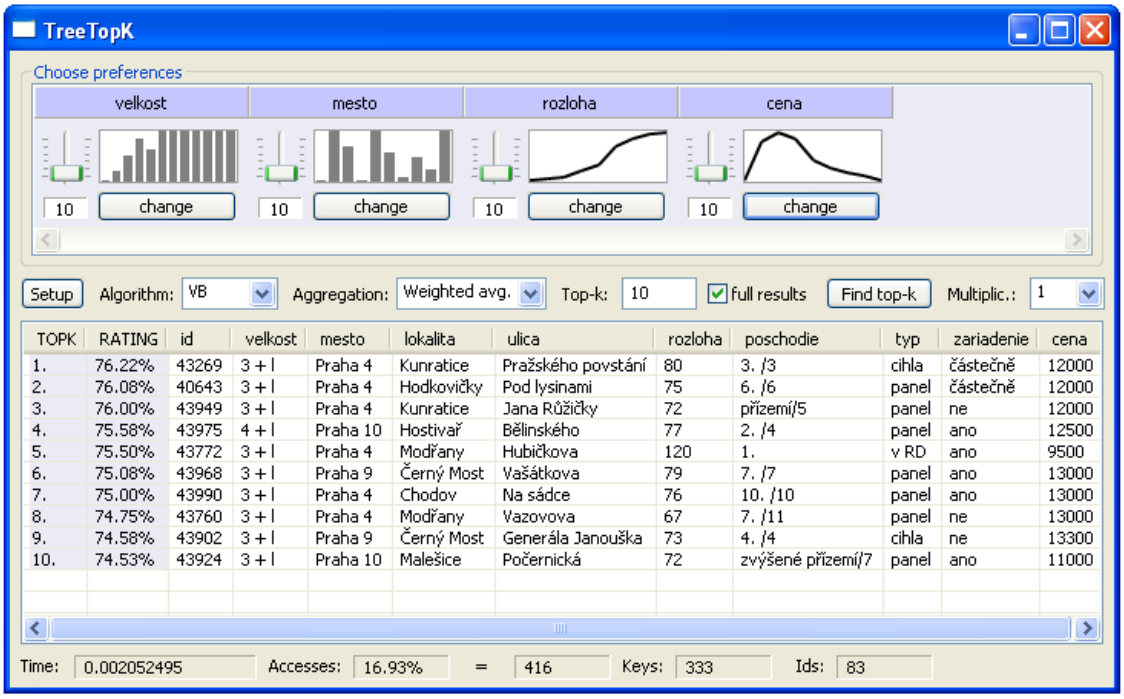
Jeden z atribútov musí byť vybraný ako identifikátor vyhľadávaných objektov (radio button). Ako identifikátor môže byť použitý iba atribút s veľkosťou domény, ktorá sa rovná počtu objektov. Ak je niektorý z ostatných atribútov možné použiť na ohodnocovanie objektov pomocou užívateľských preferencií, objaví sa pri tomto atribúte možnosť voľby (check button), pomocou ktorého je možné vybrať požadované ohodnocovacie atribúty.

V prípade spojitého atribútu je možné zvoliť jemnosť rozdelenia pre nastavovanie užívateľských preferencií (viz obrázok B.3.3.a a B.3.3.b). Ak je atribút diskrétny (veľkosť domény atribút je v tomto prípade maximálne 15) nie je možné zvoliť jemnosť rozdelenia pre nastavovanie užívateľských preferencií. Táto jemnosť je daná veľkosťou domény atribútu, pre každú hodnotu atribútu je potrebné nastaviť ohodnotenie (viz obrázok B.3.2.a).

Po nastavení ohodnocovacích atribútov sa po stlačení tlačítka „Create all Trees“ vytvoria v operačnej pamäti potrebné stromové dátové štruktúry a aplikácia TreeTopK prejde do ďalšieho zobrazenia.

## B.3.2 Vyhľadávanie top-k objektov

Na obrázku B.3.2 je zobrazenie aplikácie TreeTopK, v ktorom už je možné nastaviť užívateľské preferencie. Ďalej, v strednej časti okna, požadovaný top-k algoritmus, požadovaný typ agregačnej funkcie, počet top-k objektov a prípadne kompletnosť zobrazenia výsledkov. Podľa týchto nastavení potom aplikácia vyhľadá top-k objektov.




*Obrázok B.3.2, Konfigurácia zdroja dát*

Voľba „full results“ je štandardne vybraná. V takomto prípade sa po vyhľadaní top-k objektov získajú hodnoty všetkých atribútov objektov z databázy pomocou ďalších SQL dotazov. V opačnom prípade sa zobrazia len hodnoty atribútov získané zo stromových štruktúr počas vyhľadávania top-k objektov.

Výsledky vyhľadávania sú ešte doplnené o poradové číslo top-k objektov a ich globálne ohodnotenie vyjadrené v tvare percenta. V spodnej časti okna, pod výsledkami, sa nachádzajú informácie o čase výpočtu algoritmu, počte vykonaných prístupov do príslušných stromových štruktúr.

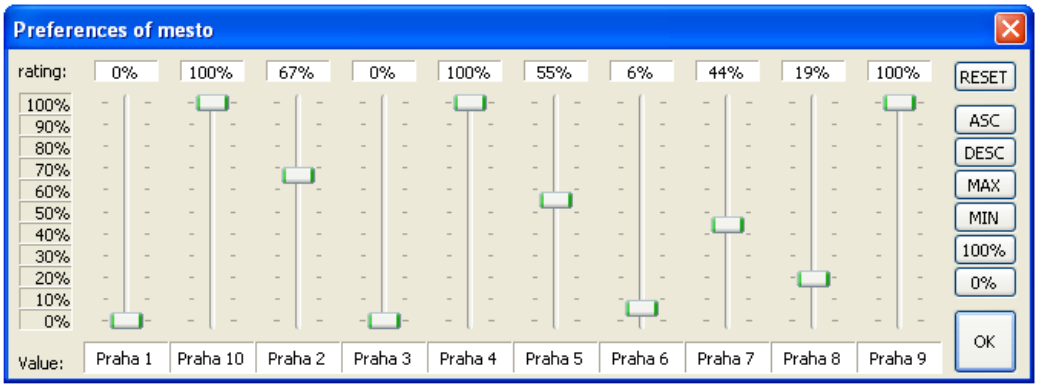
### B.3.3 Voľba užívateľských preferencií

Podľa modelu užívateľských preferencií použitých na vyhľadávanie top-k objektov v tejto práci užívateľ nastavuje svoje globálne preferencie pomocou agregačnej funkcie. Okrem toho že užívateľ nastaví požadovaný typ agregačnej funkcie, môže v prípade váženého priemeru a euklidovskej vzdialenosti nastaviť aj váhy jednotlivých atribútov.

Štandardne majú všetky atribúty rovnakú váhu (v tomto prípade 10). Váhu jednotlivých atribútov je však možné zmeniť pomocou vertikálnych posuvných stupníc (scale) pri každom z ohodnocovaných atribútov (viz obrázok B.3.2). Váhu konkrétneho atribútu je možné zvýšiť na hodnotu 100 alebo znížiť až na hodnotu 0. V tomto krajnom prípade váhy atribútu rovnej 0 hodnoty tohto atribútu neovplyvnia ohodnotenie objektu.

Lokálne preferencie jednotlivých atribútov sú v aplikácii TreeTopK znázornené graficky pri každom ohodnocovanom atribúte. Lokálne preferencie diskrétnych atribútov sú znázornené ako stĺpcový graf a spojitých atribútov ako graf funkcie (viz obrázok B.3.2).

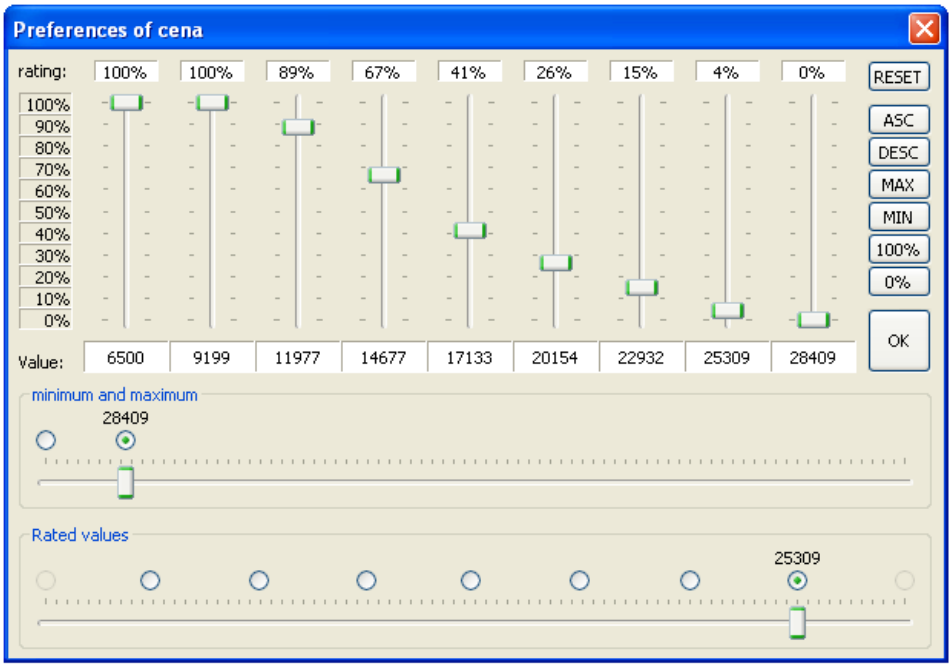
Lokálne preferencie pre jednotlivé atribúty je možné zmeniť (tlačítko „change“). Po stlačení tohto tlačítka sa spustí ďalšie okno aplikácie TreeTopK. Pre diskrétny atribút je toto okno znázornené na obrázku B.3.3.a.




*Obrázok B.3.3.a, Nastavenie lokálnych preferencií diskrétno atribútu*

Užívateľ tak môže jednoducho a pohodlne ohodnotiť všetky možné hodnoty atribútov. Pre názornosť sa ohodnotenie zobrazuje v percentách pomocou zvislých posuvných stupníc (scale). Okrem toho je možné použiť pomocné tlačítka na pravej strane okna, ktoré automaticky nastavia stupnice podľa potrieb užívateľa. Tlačítko „RESET“ vráti stupnice do pôvodného stavu.

Po stlačení tlačítka „OK“ sa nastavené lokálne preferencie uložia, okno sa zavrie a v pôvodnom okne (viz obrázok B.3.2) sa zmení grafický znázornenie preferencií príslušného atribútu.




*Obrázok B.3.3.b, Nastavenie lokálnych preferencií spojitého atribútu*

Pre spojité atribúty je situácia obdobná. Okno nastavovania preferencií je rozšírené o ďalšie dve horizontálne stupnice (viz obrázok B.3.3.b). Pomocou nich môže užívateľ nastaviť, ktoré hodnoty atribútov chce hodnotiť. Toto vylepšenie ponúka užívateľovi oveľa väčšie možnosti nastaviť preferencie podľa jeho predstáv.