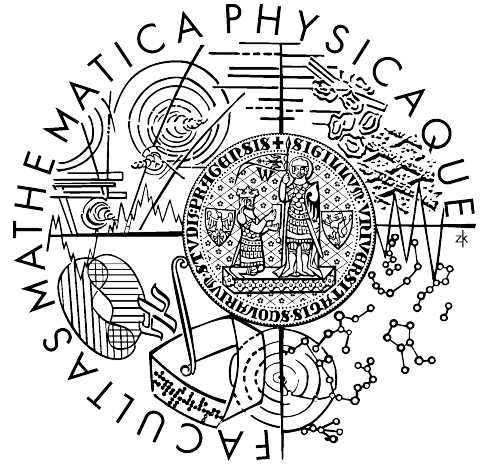
Univerzita Karlova v Praze

Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Radek Vitovják

Dynamické hašovací tabulky

Katedra softwarového inženýrství

Vedoucí práce: RNDr. Alena Koubková, CSc.

Studijní program: Informatika

Zde bych rád poděkoval své vedoucí RNDr. Aleně Koubkové, CSc. za její rady v průběhu tvorby této práce a svým rodičům za podporu při studiu.

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

V Praze dne 10. srpna 2007

Radek Vitovják

Obsah

Název práce: Dynamické hašovací tabulky

Autor: Radek Vitovják

Katedra: Katedra softwarového inženýrství

Vedoucí diplomové práce: RNDr. Alena Koubková, CSc.

E-mail vedoucího: alena.koubkova@mff.cuni.cz

Abstrakt: Cílem této práce je popsat různé metody umožňující změnu velikosti interní hašovací tabulky v závislosti na počtu vložených prvků a porovnat je na základě známých teoretických výsledků. Dále vypracovat vlastní experimentální studii chování a vzájemného porovnání vybraných metod na simulovaných datech. Závěry porovnat s teoretickými výsledky a s publikovanými výsledky předchozích experimentálních studií, pokud existují.

První část práce obsahuje popis metod implementace hašovacích tabulek a analýzu očekávaného počtu porovnání klíčů při úspěšném a neúspěšném vyhledávání. Další část pak obsahuje výsledky experimentů provedených na hašovacích tabulkách implementovaných podle popisu v první části.

Klíčová slova: interní hašovací tabulka, experimentální analýza

Title: Dynamic hash tables

Author: Radek Vitovják

Department: Department of Software Engineering

Supervisor: RNDr. Alena Koubková, CSc.

Supervisor's e-mail address: alena.koubkova@mff.cuni.cz

Abstract: The aim of the work is to describe various methods allowing the change of an internal hash table size in dependence on the number of inserted records and to compare them on the basis of known theoretical results. Then to make an experimental study of performance and mutual comparison of chosen methods on simulated data. To compare the results with theoretical findings and published precedent results, if any exist.

The first part of the work describes implementation of hash tables and the analysis of expected number of key comparison for a successful and unsuccessful search. The next part contains experimental results of the performance of hash tables implemented on the basis of description stated in the previous part.

Keywords: internal hash table, experimental analysis

1. Úvod

Programy často potřebují uchovávat bloky dat – záznamy a přistupovat k nim podle určité hodnoty – klíče. Přesněji řečeno, potřebují provádět tyto tři základní operace:

Insert – vloží určený záznam

Find – najde záznam podle klíče

Delete – odstraní záznam s daným klíčem

K těmto účelům lze použít různé datové struktury. Například:

1. Spojový seznam – umožňuje vložení a smazání záznamů v čase O(1), vyhledání v čase O(n)
2. Uspořádané pole – vložení a smazání vyžaduje čas O(n), vyhledání O(log(n))
3. Stromy – vložení, smazání a vyhledání vyžaduje typicky čas O(log(n))
4. Hašovací tabulky – vložení, smazání a vyhledání vyžaduje průměrně čas O(1), ne všechny typy tabulek však všechny tyto operace podporují.

Hašovací tabulky jsou zajímavé z několika pohledů:

Umístění záznamů v tabulce závisí na klíči, nicméně dva záznamy s podobnými klíči mohou být umístěny jakkoliv daleko od sebe.

Hašovací tabulky umožňují provádět základní operace prakticky v konstantním čase, ale existuje jistá (i když velmi malá) pravděpodobnost, že všechny prvky budou hašovací funkcí umístěny na téže místo a hašovací tabulka se pak bude chovat jako spojový seznam.

1.1 Druhy hašovacích tabulek

Na hašovací tabulky lze nahlížet z různých hledisek.

Místo uložení dat

Hašovací tabulky lze rozdělit podle toho, zda data jsou uložena v operační paměti počítače (interní hašování) nebo na pevném disku či jiném podobném médiu (externí hašování).

Hlavní rozdíl spočívá v rychlosti přístupu k datům. Přístup na disk je o několik řádů pomalejší než přístup do paměti, proto je u externího hašování nejdůležitějším kritériem hodnocení počet přístupů na disk. Výpočet hašovací funkce a ostatní operace prováděné v interní paměti tedy mohou být značně komplikované, aniž by se to znatelně projevilo na celkovém čase. Naproti tomu u interního hašování je přístup k libovolnému místu tabulky řádově stejně rychlý jako ostatní operace spojené s tabulkou, a celkový výkon závisí na rychlosti provádění všech zmíněných činností.

Statické a dynamické tabulky

Většina implementací hašovacích tabulek předpokládá, že velikost tabulky je dána při jejím vytvoření a v průběhu používání se nemění (statické hašování). Pokud pak množství dat výrazně přesáhne očekávání, chování tabulky se může značně zhoršit, nebo se její používání stane zcela nemožné. Řešením je potom vytvoření nové tabulky a přehašování všech záznamů.

Některé algoritmy umožňují plynule měnit velikost hašovací tabulky (dynamické hašování). To je výhodné zejména pro interaktivní aplikace, kdy požadujeme, aby vykonávání jakékoli operace netrvalo příliš dlouho. Správa takovýchto tabulek pak bývá o něco náročnější.

Způsob řešení kolizí

Kolize nastane, když hašovací funkce umístí dva záznamy na stejné místo v tabulce. Jedno z řešení této situace je mít u každého místa spojový seznam, do kterého jsou ukládány všechny záznamy, které sem patří. Druhá možnost je uložit kolidující záznam na jiné místo v tabulce (hašování s otevřenou adresací).

1.2 Cíl práce

Cílem této práce je popsat různé metody implementace interní hašovací tabulky umožňující změnu velikosti v závislosti na počtu vložených záznamů a porovnat je na základě známých teoretických výsledků. Dále vypracovat experimentální studii chování a vzájemného porovnání vybraných metod na simulovaných datech. Závěry porovnat s teoretickými výsledky a s publikovanými výsledky předchozích experimentálních studií, pokud existují.

2. Dynamické hašovací tabulky

V této kapitole popíšeme metody, jejichž chování budeme zkoumat. Jedná se o lineární hašování a spirálové hašování (jeho různé varianty). Tyto metody budou porovnány s implementací dynamizované standardní hašovací tabulky, která normálně změnu velikosti nepodporuje.

Všechny tyto metody používají pro řešení kolizí spojové seznamy obsahující všechny záznamy, které patří na dané místo v tabulce.

2.1 Lineární hašování

2.1.1 Popis

Lineární hašování vymyslel v roce 1980 Witold Litwin. Umožňuje plynule zvětšovat i zmenšovat hašovací tabulku, v každém kroku o jedno paměťové místo.

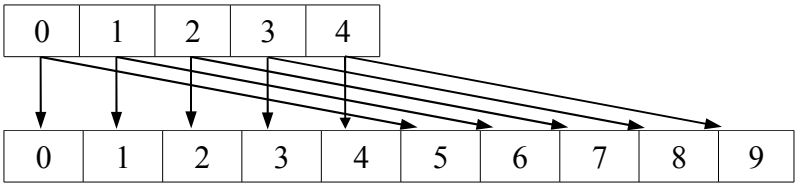
Předpokládáme, že klíč K je přirozené číslo z intervalu $[0, H]$, kde H je dostatečně velké. Pokud tomu tak není, použijeme nějakou hašovací funkci, která takové číslo každému klíči přiřadí. Dále v textu budeme používat formu zápisu, kdy K je uvedené číslo.

Na začátku má tabulka velikost M, s adresami 0, 1, ..., M-1. Každé místo v tabulce – přihrádka – obsahuje jeden ukazatel, který je začátkem spojového seznamu se všemi záznamy, které patří na tuto adresu. Jako hašovací funkci používáme $h_0(K) = K \bmod M$.

Expanze

Naším cílem je tabulku postupně zvětšovat. Nejprve se podívejme, co by se stalo, kdybychom najednou zvětšili velikost tabulky na $2M$.

Nová hašovací funkce bude $h_1(K) = K \bmod 2M$. Přitom část záznamů (přibližně polovina) se z přihrádky 0 přesune do přihrádky M, část záznamů z přihrádky 1 se přesune do přihrádky M+1, atd., až část záznamů z přihrádky M-1 se přesune do přihrádky $2M$-1. Na obrázku 1 je příklad pro $M = 5$



Obrázek 1. Zdvojnásobení velikosti tabulky

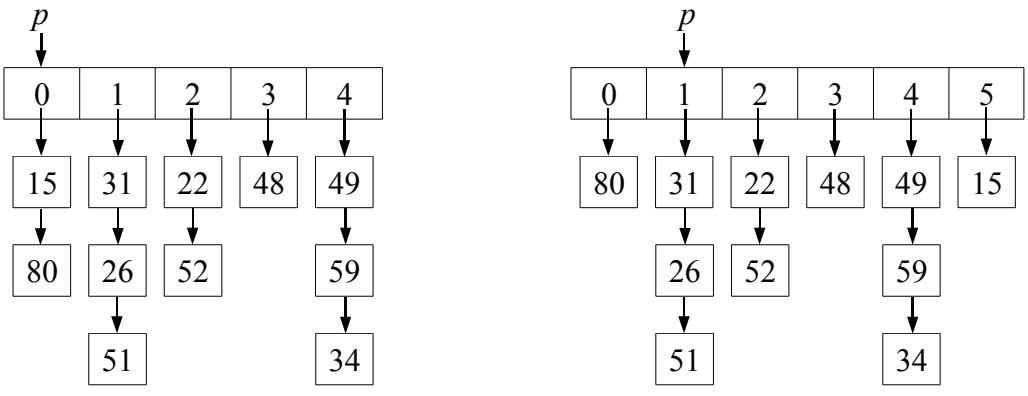
Provedení takové operace by v podstatě znamenalo přehašování všech záznamů. Taková operace by byla pro velké hodnoty M časově náročná a zablokovala by další práci s tabulkou na příliš dlouhou dobu.

Expanze po krocích

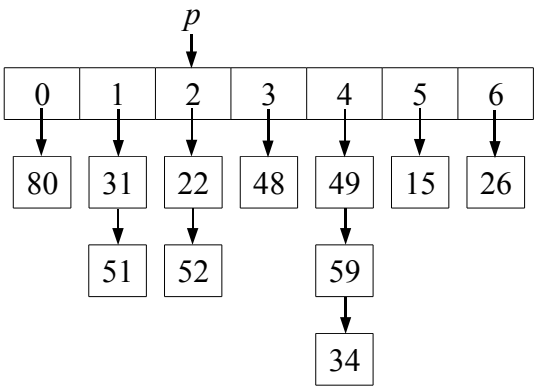
Díky vhodné volbě funkcí $h_i(x)$ můžeme expanzi provádět postupně a v každém kroku přidat právě jednu přihrádku. Mezi jednotlivými kroky bude možné s tabulkou normálně pracovat.

V prvním kroku tedy rozšíříme tabulku o přihrádku číslo M. Do ní se přesune část záznamů z přihrádky 0. Ve druhém kroku rozšíříme tabulku o přihrádku M+1, atd. V průběhu expanze si udržujeme ukazatel p na přihrádku, která se bude dělit v příštím kroku. Dále udržujeme v proměnné s počáteční velikost tabulky.

Na obrázku 2a je uveden příklad tabulky na začátku expanze a na obrázcích 2b a 2c dva kroky její expanze.



(a) (b)



(c)

Obrázek 2. Postupná expanze lineární hašovací tabulky

Adresa přihrádky, do které patří záznam s klíčem K, může být mezi jednotlivými kroky expanze zjištěna následovně: Spočítáme hodnotu $h_0(K)$. Pokud je tato hodnota menší než aktuální hodnota ukazatele p, znamená to, že přihrádka již byla rozdělena a adresa, na kterou patří K, je $h_1(K)$. Pokud je $h_0(K)$ větší nebo rovna p, přihrádka ještě nebyla rozdělena a $h_0(K)$ je správná adresa.

Příklad 1: V tabulce na obrázku 2c hledáme záznam s klíčem $K = 26$.
Spočítáme $h_0(K) = 26 \bmod 5 = 1$
Dále máme $p = 2$
Protože $h_0(K) < p$, přihrádka s hledaným záznamem již byla rozdělena a záznam je na adrese $h_1(K) = 26 \bmod 10 = 6$

Příklad 2: V tabulce na obrázku 2c hledáme záznam s klíčem $K = 52$.
Spočítáme $h_0(K) = 52 \bmod 5 = 2$
Platí $p = 2$
Protože $h_0(K) \geq p$, přihrádka s hledaným záznamem ještě rozdělena nebyla a záznam je tedy ještě stále na adrese $h_0(K) = 2$

Takto dojde expanze až do fáze, kdy se rozdělí přihrádka M-1. V tomto okamžiku se používá pro všechny záznamy hašovací funkce $h_1(K)$, a může tedy začít nová expanze.

Ukazatel p se nastaví zpět na přihrádku 0 a celý proces se opakuje. Používané hašovací funkce budou $h_1(K) = K \bmod 10$ a $h_2(K) = K \bmod 20$

Kontrakce

Velikost tabulky lze jednoduše zmenšovat postupem inverzním k expanzi. Záznamy z poslední přihrádky tabulky se přesunou do přihrádky, na kterou ukazuje ukazatel p. Hodnota ukazatele p se zmenší o 1.

Pokud hodnota p klesne pod 0, nastaví se na konec tabulky a začíná další fáze kontrakce.

Kdy provádět kontrakci a expanzi

Dosud jsme se zabývali jen tím, jak se dá velikost tabulky zvětšovat a zmenšovat, ale neřekli jsme nic o tom, kdy se tak má stát. Obvyklý postup je udržovat celkový faktor naplnění v určitých mezích. Pokud při vkládání záznamu přesáhne faktor naplnění předem stanovenou horní mez, provede se jeden krok expanze. Pokud při mazání záznamu klesne faktor naplnění pod stanovenou dolní mez, provede se jeden krok kontrakce.

2.1.2 Implementační poznámky

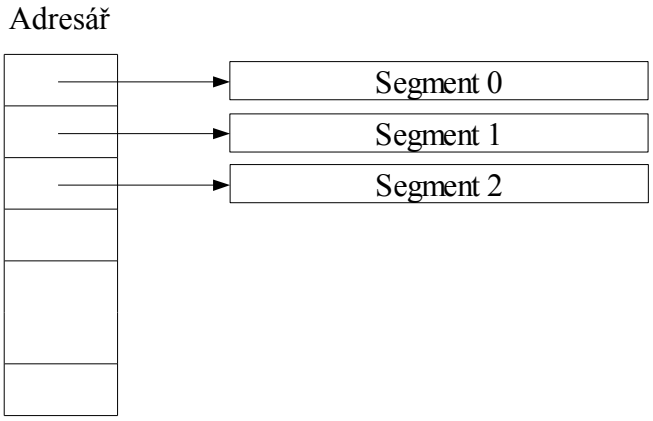
Počáteční velikost tabulky

Přestože počáteční velikost tabulky M lze zvolit libovolně, z hlediska rychlosti výpočtu je vhodné použít mocninu dvou. Pak lze místo $h_1(K) = K \bmod M$ použít rychlejší $h_1(K) = K \,\&\, (M\text{-}1)$ kde & značí bitovou operaci "and". Protože velikost tabulky se po každé expanzi zdvojnásobí, bude mít tabulka na začátku každé expanze velikost tvaru mocniny dvou a efektivnější způsob výpočtu bude možné použít pro všechny funkce h_i.

Dynamické pole

Pro implementaci lineární hašovací tabulky potřebujeme pole, jehož velikost bude možné podle potřeby zvětšovat nebo zmenšovat. Dále požadujeme, aby žádné zvětšení ani zmenšení netrvalo příliš dlouho. Takový nástroj však žádný programovací jazyk nenabízí. Existují sice funkce, které dokážou změnit velikost alokované paměti pro určenou strukturu (např. realloc v jazyku C), vždy ale připouštějí možnost, že alokují nový blok paměti a data do něj překopírují, což by u velkých polí trvalo příliš dlouho.

Proto je v této práci použita dvouúrovňová struktura (viz obrázek 3). Pole je rozdělené do segmentů pevné velikosti, pro přístup k nim slouží adresář. Když se tabulka zvětšuje, alokují se podle potřeby nové segmenty. Když se tabulka zmenšuje, nepoužívané segmenty se mohou uvolnit.



Obrázek 3. Implementace dynamického pole

2.1.3 Analýza

Provedeme asymptotickou analýzu očekávaného počtu porovnání klíčů v lineární hašovací tabulce během jejího zvětšování. Budeme předpokládat, že faktor naplnění udržujeme přibližně konstantní – tzn., že vždy, když přesáhne stanovenou mez α, provedeme jeden krok expanze. Dále budeme předpokládat, že neprovádíme operaci Delete.

Chování tabulky závisí na tom, kolik přihrádek již bylo rozděleno. Nejlépe se tabulka chová na začátku a na konci expanze, kdy je rozdělení záznamů mezi přihrádky téměř rovnoměrné.

Na tabulku se můžeme dívat, jako by se skládala ze dvou tradičních hašovacích tabulek: Jednu tabulku tvoří dosud nerozdělené přihrádky, druhou přihrádky rozdělené. V rámci každé z těchto dílčích tabulek je očekávaný počet záznamů ve všech přihrádkách stejný.

Uvažujme tradiční hašovací tabulku s faktorem naplnění α. Pak očekávaný počet porovnání klíčů při úspěšném a neúspěšném vyhledávání jsou:

$s(\alpha)=1+\alpha/2$

$u(\alpha)=\alpha$

Nechť x, $0 \leq x \leq 1$, značí podíl přihrádek, které již byly v aktuální expanzi rozděleny; dále z očekávaný počet záznamů v nerozdělené přihrádce. Pak očekávaný počet záznamů v rozdělené nebo nové přihrádce je $z/2$. Nechť v tabulce bylo na začátku expanze M přihrádek. Pak pro celkový faktor naplnění α platí:

$$\alpha=\frac{M\,z}{M(1+x)}$$

Čitatel vyjadřuje celkový počet záznamů v tabulce: Pokud bychom neprováděli aktuální expanzi (resp. kdybychom sloučili již rozdělené přihrádky), bylo by v tabulce M přihrádek, každá s očekávaným počtem záznamů z.

Jmenovatel je pak aktuální počet přihrádek v tabulce.

Uvedený vztah lze upravit na tvar

$z=\alpha(1+x)$

Očekávaný počet porovnání klíčů

Nechť $S(\alpha, x)$ značí očekávaný počet porovnání klíčů při úspěšném vyhledávání ve chvíli, kdy je podíl rozdělených přihrádek x a faktor naplnění je α. Při vyhledávání se s pravděpodobností x trefíme do rozdělené přihrádky, očekávaný počet porovnání bude $s(\alpha(1+x)/2)$. S pravděpodobností 1-x se trefíme do nerozdělené přihrádky, kde je očekávaný počet porovnání $s(\alpha(1+x))$. Využitím těchto vztahů dostaneme:

$$S(\alpha, x)=x\,s\left(\frac{\alpha(1+x)}{2}\right)+(1-x)\,s(\alpha(1+x))=1+\frac{\alpha}{4}\left(2+x-x^2\right)$$

Analogicky můžeme spočítat očekávaný počet porovnání při neúspěšném vyhledávání:

$$U(\alpha, x) = x\,u\left(\frac{\alpha(1+x)}{2}\right) + (1-x)\,u(\alpha(1+x)) = 1 + \frac{\alpha}{2}\left(2 + x - x^2\right)$$

Nejmenší očekávaný počet porovnání nastává při rovnoměrném rozdělení záznamů do všech přihrádek, tj. pro $x = 0$ nebo $x = 1$. Tehdy platí:

$$S(\alpha,0) = S(\alpha,1) = 1 + \frac{\alpha}{2}$$

$$U(\alpha,0) = U(\alpha,1) = \alpha$$

Největší očekávaný počet provonání je pro $x = 1/2$:

$$S\left(\alpha, \frac{1}{2}\right) = 1 + \frac{\alpha}{2}\frac{9}{8}$$

$$U\left(\alpha, \frac{1}{2}\right) = \alpha\frac{9}{8}$$

Průměrný očekávaný počet porovnání můžeme spočítat integrací přes očekávaný počet testů během jedné expanze:

$$\bar{S}(\alpha) = \int_0^1 S(\alpha, x)\,dx = 1 + \frac{\alpha}{2}\frac{13}{12}$$

$$\bar{U}(\alpha) = \int_0^1 U(\alpha, x)\,dx = \alpha\frac{13}{12}$$

Práce vykonaná při zvětšování tabulky

Vložení záznamu do tabulky může způsobit, že faktor naplnění přesáhne stanovenou mez. V takovém případě se provede jeden krok expanze. Zde spočítáme očekávaný počet záznamů, které jsou v takovém případě přehašovány.

Jestliže v tabulce udržujeme faktor naplnění α, pak množství záznamů, které vyvolá provedení jednoho kroku expanze, je $1/\alpha$. Provedení jednoho kroku expanze znamená projít a přehašovat záznamy z jedné (nerozdělené) přihrádky, očekávaný počet záznamů

v této přihrádce je $z = \alpha(1+x)$. Očekávaný počet přehašovávaných záznamů při vložení jednoho záznamu tedy je:

$$A(\alpha, x) = \frac{1}{\alpha}\alpha(1+x) = (1+x)$$

Průměrná hodnota během jedné expanze:

$$\bar{A}(\alpha) = \int_0^1 (1+x)\,dx = 1{,}5$$

2.2 Spirálové hašování

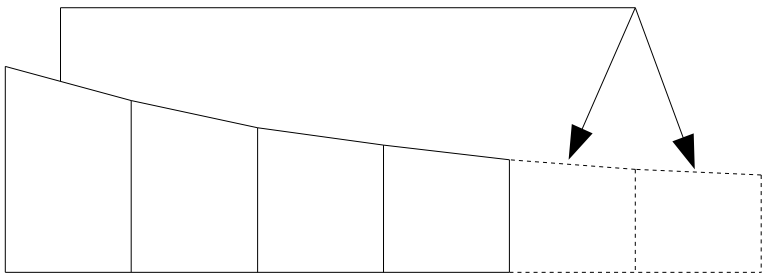
2.2.1 Popis

U lineárního hašování se očekávaný počet porovnání při vkládání, vyhledávání a mazání cyklicky mění v závislosti na fázi expanze a očekávaný počet záznamů v dosud nerozdělených přihrádkách je dvakrát větší než v rozdělených.

Spirálové hašování se snaží tyto nežádoucí vlastnosti redukovat. Využívá se zde nerovnoměrné rozmísťování záznamů do tabulky – více záznamů je umísťováno na začátek tabulky (tedy do přihrádek, které se budou dělit nejdříve), ke konci tabulky jejich množství postupně klesá.

Expanze probíhá tak, že se zruší několik přihrádek na začátku tabulky (často jen jedna) a na konci tabulky se vytvoří několik nových (a to více, než kolik se jich zrušilo). Záznamy z odstraněných přihrádek se pak rozmístí do nově vzniklých přihrádek.

Obrázek 4 ukazuje expanzi, kdy je jedna přihrádka odstraněna a vytvořeny jsou dvě nové.



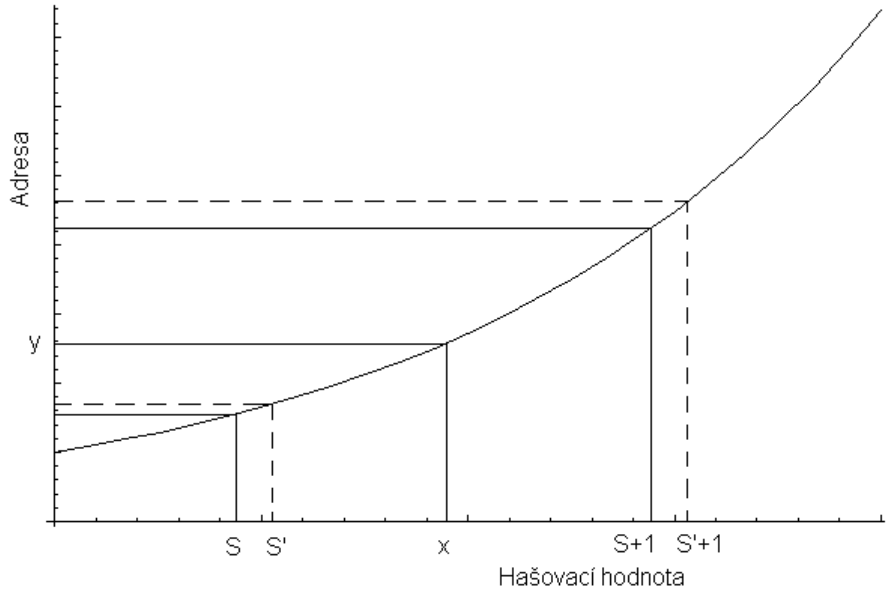
Obrázek 4. Rozmístění záznamů a expanze ve spirálovém hašování

Výpočet adresy

Spirálové hašování vyžaduje funkci $h(K)$, která klíči K přiřadí reálné číslo z intervalu [0, 1). Nechť S je (zatím blíže neupřesněné) kladné reálné číslo. Hodnota $h(K)$ je namapována na reálné číslo x z intervalu $[S, S+1)$ a to takové, že x a $h(K)$ mají stejnou desetinnou část. Takové x existuje právě jedno a lze jej vyjádřit jako $x = \lceil S - h(K) \rceil + h(K)$

Z této hodnoty je pak spočítána adresa přihrádky $y = \lfloor d^x \rfloor$, kde $d > 1$ je konstanta, která se nazývá růstový faktor, funkce d^x se nazývá expanzní (nebo růstová) funkce. Aktivní adresový prostor tedy začíná adresou $\lfloor d^S \rfloor$ a končí adresou $\lceil d^{S+1} \rceil - 1$, což je přibližně $d^S(d-1)$ adresových míst.

Odtud je zřejmé, že číslo S určuje velikost tabulky – pokud se jeho velikost zvýší z S na nějaké S', zvětší se i velikost tabulky. Viz Obrázek 5.



Obrázek 5. Výpočet adresy ve spirálovém hašování

Z obrázku 5 je dále vidět význam volby x jako čísla se stejnou desetinnou částí jako S. Pokud se totiž hodnota S změní jen málo, zůstane pro většinu záznamů v tabulce hodnota x stejná a nemusí se přehašovávat. Přehašovat je nutné pouze záznamy

z několika přihrádek ze začátku tabulky, které tímto zaniknou. S' se obvykle volí tak, že zanikne právě jedna přihrádka.

Opačným postupem pak lze velikost tabulky zmenšovat. Snížením hodnoty S zanikne několik přihrádek na konci tabulky a na začátku jich vznikne menší počet.

Spirálová hašovací tabulka s *d* = 2

Příklad: Nechť je počáteční velikost tabulky 5.

Hodnota S je pak dána velikostí tabulky – musí splňovat rovnici $2^{S+1}-2^{S}=5$, odkud vyplývá $S=\log_2(5)\approx 2{,}3219$

První aktivní adresa je pak $\lfloor 2^{2{,}2319} \rfloor = 5$, poslední $\lceil 2^{3{,}3219} \rceil - 1 = 9$.

Tabulka 1 ukazuje rozmístění záznamů do přihrádek v závislosti na jejich hašovací hodnotě. Záznamy splňující $0{,}3219 \le h(K) < 0{,}5850$ padnou do přihrádky 5; záznamy s $0{,}5850 \le h(K) < 0{,}8074$ do přihrádky 6 atd. Např. záznam s $h(K)=0{,}9$ má hodnotu $x=\lceil 2{,}3219-0{,}9 \rceil + 0{,}9 = 2{,}9$ a adresu přihrádky $y=\lfloor 2^{2{,}9} \rfloor = \lfloor 7{,}4643 \rfloor = 7$

Hodnota $h(K)$	Adresa přihrádky	Délka intervalu
[0,3219; 0,5850)	5	0,2631
[0,5850; 0,8074)	6	0,2224
[0,8074; 1,0000)	7	0,1926
[0,0000; 0,1699)	8	0,1699
[0,1699; 0,3219)	9	0,1520
[0,3219; 0,4594)	10	0,1375
[0,4594; 0,5850)	11	0,1256
[0,5850; 0,7004)	12	0,1154
[0,7004; 0,8074)	13	0,1070

Tabulka 1: Rozložení záznamů ve spirálové hašovací tabulce

Tabulka 1 dále uvádí délky intervalů hašovacích hodnot, které padnou do dané přihrádky. Protože tyto hodnoty pokrývají interval délky 1, jsou tyto hodnoty současně rovny podílu záznamů, které do dané přihrádky připadají. V přihrádce 5 tak bude přibližně 26% záznamů, v přihrádce 6 asi 22% záznamů atd.

Když budeme chtít zvětšit velikost tabulky o jednu přihrádku, zvětšíme S na S' tak, aby platilo $2^{S'+1}-2^{S'}=6$, odkud vyplývá $S'=\log_2 6=2{,}5850$. První aktivní adresa pak bude $\lfloor 2^{2{,}5850}\rfloor=6$, poslední $\lceil 2^{3{,}5850}\rceil-1=11$. Z tabulky tedy zmizela přihrádka 5 s rozsahem hašovacích hodnot [0,3219; 0,5850) a vznikly dvě nové: 10 s rozsahem [0,3219; 0,4594) a 11 s rozsahem [0,4594; 0,5850). Záznamy z přihrádky 5, které mají $h(K) < 0{,}4594$ tedy připadnou do přihrádky 10, záznamy s $h(K) \geq 0{,}4594$ do přihrádky 11.

Při dalším zvětšování tabulky se zvětší S' na $S''=\log_2 7=2{,}8074$, zmizí přihrádka 6 a vzniknou přihrádky 12 a 13.

Rychlejší výpočet adresy

Výpočet 2^x je časově poměrně náročná operace. Čas výpočtu lze zmenšit použitím nějaké aproximační funkce, jejíž vyčíslení je rychlejší. Pro tento účel stačí mít aproximaci 2^x na intervalu [0,1]. Pak aproximaci 2^x pro libovolné x lze spočítat následovně:

$$2^x=2^{(\lfloor x\rfloor+x-\lfloor x\rfloor)}=2^{\lfloor x\rfloor}\cdot 2^{(x-\lfloor x\rfloor)}\approx 2^{\lfloor x\rfloor}\cdot f(x-\lfloor x\rfloor)$$

Výraz $x-\lfloor x\rfloor$ v podstatě značí desetinnou část čísla x. Pro zjednodušení zápisu ji označme $\{x\}$.

Martin [5] doporučuje jako funkci f použít funkci tvaru $f(x)=\frac{a}{b-x}+c,\ 0\leq x\leq 1$

Aproximační funkce pak má tvar $g(x)=2^{\lfloor x\rfloor}\cdot\frac{a}{b-\{x\}}+c$

Hodnoty parametrů a, b, c jsou částečně dány podmínkami na vlastnosti funkce f. Z požadavku na spojitost funkce $2^{\lfloor x\rfloor}\cdot f(\{x\})$ plynou dvě rovnosti: $f(0)=1$ a $f(1)=2$. Třetí podmínka pro jednoznačné určení hodnot a, b, c může být libovolná. Larson [4] doporučuje, aby f měla stejnou hodnotu jako funkce 2^x v bodě 0,5, tedy podmínku $f(0{,}5)=\sqrt{2}$. Maximální chybu pak uvádí 0,23%.

Vyřešením těchto tří rovnic obdržíme:

$b=2+\sqrt{2}\approx 3{,}4142$

$a=b(b-1)=4+3\sqrt{2}\approx 8{,}2426$

$c=2-b=-\sqrt{2}\approx -1{,}4142$

K funkci $f(x)=\frac{a}{b-x}+c$ bude pro výpočet S také potřebná funkce inverzní:

$$f^{-1}(y)=b-\frac{a}{y-c}$$

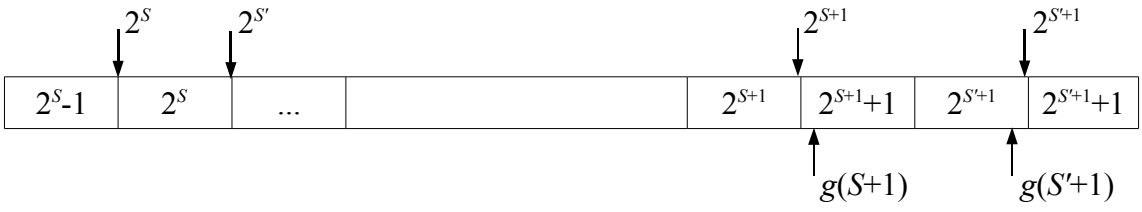
Anomálie při použití jiných expanzních funkcí

Pro spirálové hašování lze použít i jiné expanzní funkce než ty tvaru d^x.

Při použití 2^x jako expanzní funkce se velikost tabulky vždy zvětší o jednu přihrádku (jedna zanikne a dvě nové vzniknou). Ostatní růstové funkce takovou vlastnost mít nemusí. Funkce g je tedy vlastně jiná růstová funkce; chování tabulky je pak velmi podobné jako při použití 2^x – při expanzi se tabulka většinou zvětší o jednu přihrádku. Někdy se však stane, že se tabulka zvětší o dvě přihrádky, někdy se vůbec nezvětší.

Jak takovéto anomálie vznikají:

Pro dané S je obor hodnot funkce 2^x interval $[2^S, 2^{S+1})$. Pokud je 2^S celé číslo (což při výše uvedené volbě S vždy platí), je i 2^{S+1} celé číslo; nejvyšší využívaná přihrádka má tedy index $2^{S+1}-1$. Pokud je jako expanzní funkce použita nějaká aproximační funkce $g(x)$, může se stát, že místo hodnoty 2^{S+1} dostaneme hodnotu o něco větší. Některé záznamy se tak mohou dostat i do přihrádky 2^{S+1}. Při expanzi se pak tato přihrádka musí považovat za aktivní (v podstatě je "falešně aktivní", neboť do ní padne jen velmi málo záznamů) a velikost tabulky se tak zvětší o dvě přihrádky. Pokud po takovéto situaci je pro nějaké S' hodnota $g(S'+1)$ opět menší nebo rovna $2^{S'+1}$, vznikne jen jedna nová aktivní přihrádka a velikost tabulky se při expanzi se nezvětší. Viz obrázek 6.



Obrázek 6. Anomálie expanze při použití aproximační funkce

Tyto anomálie pak vyžadují ošetření při expanzi a kontrakci. Při expanzi je nutné počítat s tím, že mohou vzniknout tři nové přihrádky místo dvou; dále se pak může stát, že přehašovávané záznamy padnou i do přihrádky, která byla aktivní už po předchozí expanzi – to pokud se jednalo o "falešně aktivní" přihrádku. Při kontrakci je zase nutné přehašovat záznamy z adresy, která se po kontrakci stane nejvyšší aktivní adresou – neboť je možné, že tato přihrádka je jen "falešně aktivní" a většina záznamů z ní patří do přihrádky s nejnižší aktivní adresou.

2.2.2 Implementační poznámky

Dynamické pole

Pro implementaci spirálové hašovací tabulky je opět potřebné dynamické pole. Lze použít podobnou strukturu jako u lineárního hašování. Ve spirálové hašovací tabulce se ale navíc uvolňují přihrádky ze začátku tabulky, což umožňuje uvolňovat vyprázdněné segmenty. Na začátku adresáře tak vznikají volná místa. Ta lze využít tak, že po dosažení konce adresáře se začnou nové segmenty umísťovat zase od začátku – dokud se celý adresář nezaplní.

Velikost tabulky a proměnná *S*

V tabulce je vhodné udržovat mimo jiné tyto údaje: Hodnotu S (která určuje velikost adresového prostoru), nejnižší aktivní adresu a nejvyšší aktivní adresu.

Požadovaná velikost tabulky určuje hodnotu S a tím i nejnižší používanou adresu. To platí také naopak: Pokud je dána nejnižší adresa, je tím určena velikost tabulky. Dále se při každé expanzi nejnižší adresa zvětší o 1. Pro implementaci je výhodnější pohled, při kterém je velikost tabulky regulována nejnižší adresou. Zvětší-li se nejnižší adresa o 1, zvětší se (téměř vždy) i velikost tabulky o 1.

Nechť g je expanzní funkce a l nejnižší požadovaná aktivní adresa. Pak musí platit $l = g(S)$, tj:

$$l = 2^{\lfloor S \rfloor} \cdot \left(\frac{a}{b - \{S\}} + c \right) \qquad (1)$$

Z této rovnice je nyní nutné vypočítat hodnotu S.

Funkce $f(x)=\frac{a}{b-x}+c$ má při definičním oboru [0, 1] obor hodnot [1, 2].

Platí tedy $1\leq\frac{a}{b-x}+c\leq 2$

Dosazením do rovnosti (1) získáme:

$1\leq\frac{l}{2^{\lfloor S\rfloor}}\leq 2$

$2^{\lfloor S\rfloor}\leq l\leq 2^{\lfloor S+1\rfloor}$

Z toho vyplývá, že celá část čísla *S* je největší přirozené číslo *n* takové, že $2^n\leq l$. Nyní lze z rovnosti (1) dopočítat desetinnou část čísla *S*:

$$\{S\}=b-\frac{a}{\frac{l}{2^{\lfloor S\rfloor}}-c}$$

Výpočet hašovací funkce

Při výpočtu adresy klíče *K* se hodnota *h*(*K*) namapuje na hodnotu *x* z intervalu [*S*, *S*+1] a následně se počítá hodnota *g*(*x*). Pro urychlení výpočtu funkce *g*(*x*) je vhodné neudržovat hodnotu *S* přímo, ale uchovávat hodnoty $\{S\}$ a $2^{\lfloor S\rfloor}$. V programu jsou tyto hodnoty v proměnných x0 a y0. Funkce vracející adresu klíče *K* pak může vypadat následovně:

```
int addr(KeyType K) {
    double x_fract = h(K);
    double temp_addr = y0 * (a/(b-x_fract) + c);
    if ( x_fract < x0 ) temp_addr = temp_addr * 2;
    return (int)temp_addr;
}
```

Tato implementace nepočítá adresu přesně podle postupu popsaného výše, výsledek ale dává tentýž:

Hodnota *x* používaná při výpočtu *g*(*x*) je dána vztahem $x=\lceil S-h(K)\rceil+h(K)$, což lze upravit na $x=\lceil\lfloor S\rfloor+\{S\}-h(K)\rceil+h(K)$. Zde mohou nastat dva případy

$\{S\}-h(K)\leq 0$, pak $x=\lfloor S\rfloor+h(K)$, a tedy $g(x)=2^{\lfloor S\rfloor}\cdot\left(\frac{a}{b-h(K)}+c\right)$

$\{S\}-h(K)>0$, pak $x=\lfloor S\rfloor+1+h(K)$, a tedy $g(x)=2^{\lfloor S\rfloor+1}\cdot\left(\frac{a}{b-h(K)}+c\right)$

V obou případech je výraz $\frac{a}{b-h(K)}+c$ stejný. Jediný rozdíl je v tom, zda se tato hodnota bude násobit $2^{\lfloor S\rfloor}$ nebo $2^{\lfloor S\rfloor+1}$. Funkce tedy nejprve tedy spočítá hodnotu $2^{\lfloor S\rfloor}\cdot\left(\frac{a}{b-h(K)}+c\right)$ a pokud je $h(K)<\{S\}$, vynásobí tuto hodnota dvěma.

2.2.3 Analýza

Podobně jako u lineárního hašování provedeme asymptotickou analýzu očekávaného počtu porovnání klíčů při úspěšném a neúspěšném vyhledávání. Výpočty provedeme pro obecný růstový faktor *d*.

Nejprve předpokládejme obecnější případ: Mějme (libovolnou) hašovací tabulku s M přihrádkami obsahující N záznamů. Dále nechť pravděpodobnost, že náhodně vybraný záznam padne do přihrádky i je p_i pro $1\leq i\leq M$ a platí $\sum_{i=1}^{M} p_i=1$. Pak pravděpodobnost, že přihrádka i obsahuje právě j záznamů je $P_{Nij}=\binom{N}{j} p_i^j(1-p_i)^{N-j}$.

Nechť X je náhodná veličina s binomickým rozdělením a $P(X=j)=P_{Nij}$. Střední hodnota této veličiny je pak $E(X)=N\,p_i$, rozptyl $Var(X)=N\,p_i(1-p_i)$. Dále platí $E(X^2)=Var(X)+(E(X))^2=N\,p_i+N(N-1)\,p_i$.

Očekávaný počet porovnání klíčů

Očekávaný počet porovnání $S(N)$ při úspěšném vyhledávání jednoho záznamu je podíl celkového počtu všech porovnání při vyhledávání všech záznamů a počtu těchto záznamů N.

Přihrádka obsahující *j* záznamů přispěje do celkového počtu $j(j+1)/2$ porovnáními (neboť při vyhledávání prvního záznamu se potřebuje jedno porovnání, při vyhledávání druhého záznamu dvě porovnání atd.). Očekávaný počet porovnání, kterými přispěje přihrádka *i* pak je $\sum_{j=1}^{N} \frac{j(j+1)}{2} P_{Nij}$, všechny přihrádky dohromady pak přispějí $\sum_{i=1}^{M}\sum_{j=1}^{N} \frac{j(j+1)}{2} P_{Nij}$ porovnáními. *S*(*N*) se pak dá vyjádřit následovně:

$$\begin{aligned} S(N) &= \frac{1}{N}\sum_{i=1}^{M}\sum_{j=1}^{N} \frac{j(j+1)}{2} P_{Nij} = \frac{1}{2\mathrm{N}}\sum_{i=1}^{M}\left(\sum_{j=1}^{N} j\, P_{Nij} + \sum_{j=1}^{N} j^2\, P_{Nij}\right) \\ &= \frac{1}{2\mathrm{N}}\sum_{i=1}^{M}\left(E(X)+E(X^2)\right) = \frac{1}{2\mathrm{N}}\sum_{i=1}^{M}\left(2N\, p_i + N(N-1)\, p_i^2\right) \\ &= \sum_{i=1}^{M} p_i + \frac{N-1}{2}\sum_{i=1}^{M} p_i^2 = 1 + \frac{N-1}{2}\sum_{i=1}^{M} p_i^2 \end{aligned} \qquad (2)$$

Počet porovnání při neúspěšném vyhledávání v přihrádce s *j* záznamy je *j*. Očekávaný počet porovnání v přihrádce *i* potom je $\sum_{j=1}^{N} j\, P_{Nij}$. Očekávaný počet porovnání při neúspěšném vyhledávání v celé tabulce pak je:

$$U(N) = \sum_{i=1}^{M}\left(p_i \sum_{j=1}^{N} j\, P_{Nij}\right) = \sum_{i=1}^{M}\left(p_i\, E(X)\right) = N\sum_{i=1}^{M} {p_i}^2 \qquad (3)$$

Nyní zpět ke spirálovému hašování. Jak bylo řečeno dříve, ve spirálové hašovací tabulce jsou vždy aktivní přihrádky $\{d^S, d^S+1, \ldots, d^{S+1}-1\}$ pro nějaké *S*. Nechť počet těchto přihrádek $M = d^{S+1} - d^S = (d-1)\,d^S$ je násobkem *d*-1. Pak platí

$$d^S = \frac{M}{d-1} \text{ a } d^{S+1} = d\,\frac{M}{d-1}$$

Připomeňme, že do aktivní přihrádky *i* padnou právě ty záznamy, pro jejichž hodnotu *x* platí $i = \lfloor d^x \rfloor$, tedy

$i \le d^x < i+1$

$\log_d(i) \le x = \lfloor x \rfloor + \{x\} < \log_d(i+1)$

$\log_d(i) - \lfloor x \rfloor \le \{x\} < \log_d(i+1) - \lfloor x \rfloor$

Hodnota $\{x\}$ je (podle definice x) rovna hodnotě $h(K)$ daného záznamu K. Délka intervalu hodnot $\{x\} = h(K)$, které padnou do přihrádky i tedy je:

$$\log_d(i+1)-\lfloor x\rfloor-(\log_d(i)-\lfloor x\rfloor)=\log_d(i+1)-\log_d(i)=\log_d\left(1+\frac{1}{i}\right)$$

Protože hodnoty $h(K)$ všech záznamů pokrývají jednotkový interval, je tato délka rovna pravděpodobnosti, že záznam padne do přihrádky i.

Nyní můžeme spočítat hodnotu $\sum p_i^2$, která se objevuje ve vztazích pro očekávaný počet porovnání:

$$\begin{aligned}\sum_{i=M/(d-1)}^{dM/(d-1)-1} p_i^2 &= \sum_{i=M/(d-1)}^{dM/(d-1)-1} \log_d^2\left(1+\frac{1}{i}\right) = \frac{1}{\ln^2 d}\sum_{i=M/(d-1)}^{dM/(d-1)-1} \ln^2 d\left(1+\frac{1}{i}\right) \approx \\ &\approx \frac{1}{\ln^2 d}\sum_{i=M/(d-1)}^{dM/(d-1)-1}\left(\frac{1}{i^2}-\frac{1}{i^3}+\frac{11}{12\,i^4}-\dots\right) \approx \frac{1}{\ln^2 d}\sum_{i=M/(d-1)}^{dM/(d-1)-1}\frac{1}{i^2} \approx \\ &\approx \frac{1}{\ln^2 d}\int_{M/(d-1)}^{dM/(d-1)}\frac{1}{x^2}dx = \frac{1}{\ln^2 d}\frac{(d-1)^2}{dM} \qquad (4)\end{aligned}$$

Zde třetí rovnost využívá aproximace funkce $\ln(1+x)$ pomocí Taylorova polynomu.

Pátá rovnost je aplikace Eulerova součtového vzorce.

Tento výsledek lze použít v rovnostech (2) a (3). Indexy v sumách u p_i sice nejsou stejné, ve všech třech případech se ale sčítalo přes všechny přihrádky. $S(N)$ a $U(N)$ tedy lze vyjádřit následovně:

$$S(N)\approx 1+\frac{N-1}{2}\left(\frac{1}{\ln^2 d}\frac{(d-1)^2}{dM}\right) = 1+\frac{(N-1)(d-1)^2}{2\,dM\ln^2 d}$$

$$U(N)\approx N\left(\frac{1}{\ln^2 d}\frac{(d-1)^2}{dM}\right)=\frac{N(d-1)^2}{dM\ln^2 d}$$

Tyto vztahy lze ještě upravit dosazením $\alpha = N/M$

$$S(\alpha)\approx 1+\alpha\frac{(d-1)^2}{2d\ln^2 d}$$

$$U(\alpha)\approx\alpha\frac{(d-1)^2}{d\ln^2 d}$$

Pro speciální případ $d = 2$ platí:

$$S(\alpha) \approx 1 + \alpha \frac{1}{4 \ln^2 2} \approx 1 + 0{,}5203\,\alpha$$

$$U(\alpha) \approx \alpha \frac{1}{2 \ln^2 2} \approx 1{,}0407\,\alpha$$

Práce vykonaná při zvětšování tabulky

Spočítáme očekávaný počet záznamů, které je nutno přehašovat při vložení jednoho záznamu do tabulky.

Nechť přihrádka, která se bude dělit, má číslo i, v tabulce je N záznamů a M přihrádek. Připomeňme několik faktů:

- Vždy dělí aktivní přihrádka s nejnižším číslem, platí tedy $i = d^S$.
- $M = d^S (d-1) = i\,(d-1)$
- Pravděpodobnost, že záznam padne do přihrádky i je $\log_d \left(1 + \frac{1}{i}\right)$

Očekávaný počet záznamů v přihrádce i tedy je:

$$N \log_d \left(1 + \frac{1}{i}\right) = \alpha M \log_d \left(1 + \frac{1}{i}\right) = \alpha (d-1)\, i \log_d \left(1 + \frac{1}{i}\right) =$$

$$= \alpha (d-1) \log_d \left(1 + \frac{1}{i}\right)^i \approx \alpha (d-1) \log_d e = \alpha \frac{d-1}{\ln d}$$

Při rozdělení jedné přihrádky se jejich celkový počet v tabulce zvýší o d-1. Počet záznamů, které se mohou vložit do tabulky, než faktor naplnění opět přesáhne stanovenou mez, je tedy $(d\text{-}1)\alpha$. Tedy jen $\frac{1}{(d-1)\alpha}$ vkládaných záznamů vyvolá expanzi. Očekávaný počet přehašovávaných záznamů na jeden vložený záznam tedy je:

$$A(\alpha) = \frac{1}{(d-1)\alpha} \alpha \frac{d-1}{\ln d} = \frac{1}{\ln d}$$

Speciálně pro $d = 2$ pak platí: $A(\alpha) = \frac{1}{\ln 2} \approx 1{,}4427$

2.3 Dynamizovaná standardní hašovací tabulka

Nejedná se tedy o skutečně dynamickou hašovací tabulku, ale o pseudodynamickou strukturu. Nelze ji tedy brát jako rovnocennou alternativu, bude ale sloužit pro orientační porovnání s dynamickými metodami.

2.3.1 Popis

Pod pojmem standardní hašovací tabulka budeme v této práci rozumět následující strukturu: Vždy je alokována tabulka určité velikosti – s pevným počtem přihrádek. Přihrádky jsou realizovány opět jako spojové seznamy. Expanze (resp. kontrakce) tabulky probíhá následovně: Nejprve se alokuje nová tabulka – dvakrát větší (resp. menší) než stávající; následně jsou do ní přehašovány všechny záznamy. Stará tabulka je pak dealokována.

Expanze a kontrakce probíhají při dosažení stanovených mezních faktorů naplnění.

2.3.2 Analýza

Opět provedeme asymptotickou analýzu očekávaného počtu porovnání klíčů při vkládání a vyhledávání.

Nechť se expanze tabulky provede vždy, když faktor naplnění dosáhne hodnoty α. Po provedení expanze tedy faktor klesne na hodnotu $\alpha/2$. Aktuální faktor naplnění mezi dvěma expanzemi tedy lineárně roste z $\alpha/2$ na α. Průměrný faktor naplnění udržovaný v tabulce při jejím zvětšování je tedy $3\alpha/4$.

Nechť se kontrakce tabulky provede vždy, když faktor naplnění klesne pod hodnotu β. Po provedení kontrakce faktor naplnění stoupne na hodnotu 2β. Aktuální faktor naplnění mezi dvěma kontrakcemi tedy lineárně klesá z 2β na β, průměrný faktor naplnění udržovaný v tabulce při jejím zmenšování je tedy $3\beta/2$.

Pokud tedy má být faktor naplnění ve standardní hašovací tabulce udržován přibližně na hodnotě γ, je nutné, aby mezní faktor naplnění pro expanzi měl hodnotu $\alpha = 4\gamma/3$ a mezní faktor pro kontrakci $\beta = 2\gamma/3$.

Očekávaný počet porovnání klíčů

Pokud faktor naplnění v tabulce je γ, pak očekávané počty porovnání klíčů při úspěšném a neúspěšném vyhledávání jsou:

$S(\gamma) = 1 + \gamma/2$

$U(\gamma) = \gamma$

Práce vykonaná při zvětšování tabulky

Spočítáme očekávaný počet záznamů přehašovávaných při expanzi připadající na jeden vložený záznam.

Nechť tabulka obsahuje N záznamů a právě byla provedena expanze. Při této expanzi bylo přehašováno všech N záznamů. Předchozí expanze proběhla v okamžiku, kdy tabulka obsahovala $N/2$ záznamů a tyto záznamy byly tehdy přehašovány. Při každé předcházející expanzi tak bylo přehašováno poloviční množství záznamů. Celkové množství přehašovaných záznamů právě po expanzi tedy je:

$$N + \frac{N}{2} + \frac{N}{4} + \frac{N}{8} + ... + N_0 \approx 2\,N$$

kde N_0 je počáteční velikost tabulky. Pokud budeme považovat N_0 za zanedbatelné vzhledem k N, pak očekávaný počet přehašovávaných záznamů na jeden vložený záznam po provedení expanze je $\frac{2\,N}{N} = 2$. Pokud budou vkládány další záznamy, bude tabulka těsně před další expanzí obsahovat $2N$ záznamů a očekávaný počet přehašovávaných záznamů na jeden vložený záznam bude $\frac{2\,N}{2\,N} = 1$.

3. Realizace experimentů

3.1 Implementace hašovacích tabulek a testů

Pro tuto práci byly vytvořeny vlastní implementace hašovacích tabulek.

Lineární hašovací tabulka

Implementace lineární hašovací tabulky vychází z programu uvedeného v [7], je ale obecnější. Autor se zde snažil dosáhnout co nejrychlejší implementace, ale někdy za cenu značných omezení. (Například meze, ve kterých je udržován faktor naplnění, musely být celočíselné.)

Spirálová hašovací tabulka

Implementace vycházejí z částí kódu uvedených v [4]. Spirálová hašovací tabulka byla realizována pro čtyři různé růstové funkce:

1. 2^x.
2. Aproximace 2^x tvaru $g(x)=2^{\lfloor x \rfloor}\cdot\frac{a}{b-\{x\}}+c$. Hodnoty a, b, c jsou spočítány postupem uvedeným výše.
3. d^x pro obecné $d \geq 2$.
4. Aproximace 3^x tvaru $g(x)=2^{\lfloor x \rfloor}\cdot\frac{a}{b-\{x\}}+c$. Hodnoty a, b, c jsou spočítány analogicky jako v případu 2.

Standardní hašovací tabulka

Implementace dynamizované standardní hašovací tabulky je provedena podle výše uvedeného popisu.

3.1.1 Programovací jazyk

Jako programovací jazyk byl zvolen C++, a to z několika důvodů:

- O správu paměti se stará programátor, vznik a zánik objektů tedy nastává ve stanovených místech programu a běh není narušován těžko předvídatelnými zásahy garbage-colectoru.
- Pomocí maker lze jednoduchými změnami efektivně měnit chování programu. Toho je využito mimo jiné pro oddělení měření času a počítaní operací provedených programem.
- Šablony umožňují psát kód obecně a přitom velmi efektivně. Ve většině testů prováděných pro tuto práci byla použita jako klíče celá čísla, implementace ale umožňuje použít tabulku pro jakýkoliv jiný typ, který má korektně definované operátory `=` (přiřazení), `==` (porovnání) a operátor přetypování na hodnotu `unsigned int` (získání hašovací hodnoty). Jeden z testů byl prováděn na řetězcích; pro tento účel byla vytvořena třída `HashString`.

3.1.2 Kompilování

Program může být zkompilován jak pro operačním systém Windows, tak pro Linux. Pro korektní ošetření částí kódu závislých na operačním systému slouží makro `WIN32`, které musí být definováno na Windows a nesmí být definováno na Linuxu.

Při kompilování je vždy do výsledného spustitelného souboru vložen jen jeden typ hašovací tabulky. Tento typ je určen makrem `TABLE_ID`, jehož hodnota musí být číslo z rozsahu 1 až 6. Význam těchto čísel je následující:

`TABLE_ID`	Typ tabulky
1	Standardní hašovací tabulka
2	Lineární hašovací tabulka
3	Spirálová hašovací tabulka – aproximace 2^x
4	Spirálová hašovací tabulka – 2^x
5	Spirálová hašovací tabulka – d^x
6	Spirálová hašovací tabulka – aproximace 3^x

Pokud je hodnota `TABLE_ID` rovna 5 (spirálová hašovací tabulka s růstovou funkcí d^x), musí být dále definováno makro `SPIRAL_D`, jehož hodnota určuje hodnotu růstového faktoru d.

Pro testování je tedy nutné vytvořit pro každou tabulku zvláštní spustitelný soubor.

Tento způsob realizace nevypadá příliš elegantně a pohodlně, ale shledal jsem jej jako nejlepší: Další možnost realizace byla pomocí virtuální dědičnosti. Všechny implementace tabulek by byly odvozeny od společného předka, při spuštění testu by se identifikace požadované tabulky předala programu jako parametr a na začátku testu by se jednoduše zvolil požadovaný typ tabulky. Tato možnost však s sebou nese režii spojenou s virtuální dědičností a zbytečně tak zpomaluje běh programu. Navíc při použití hašovacích tabulek v praxi je skutečně použit jenom jeden typ tabulky a žádná dědičnost použita není.

Pokud je dále při kompilování definováno makro `CMP_COUNT`, pak při spuštění programu není měřen čas běhu, ale jsou zjišťovány počty porovnání klíčů při vkládání, vyhledávání a mazání záznamů. Tímto způsobem je zajištěno, že při měření času není běh programu zdržován výpočty, které do něj vlastně nepatří.

Pro kompilaci pod Windows byl vytvořen projekt pro Microsoft Visual Studio 2005. Tento projekt slouží zejména pro ladění a neumožňuje pohodlně provádět testy.

Pro překlad pod Linuxem lze použít Makefile, který umožňuje najednou vygenerovat spustitelné soubory pro všechny tabulky. Pro každou tabulku jsou vytvořeny soubory hash_*jméno_tabulky* a hash_*jméno_tabulky*_stats. První ze souborů slouží k měření doby běhu programu, druhý zaznamenává počty porovnání. Jména tabulek a jejich význam je následující:

Jméno tabulky	Význam
standard	Standardní hašovací tabulka
linear	Lineární hašovací tabulka
spiral2x	Spirálová hašovací tabulka – 2^x
spiral2xa	Spirálová hašovací tabulka – aproximace 2^x
spiral3x	Spirálová hašovací tabulka – 3^x
spiral3xa	Spirálová hašovací tabulka – aproximace 3^x

3.1.3 Typy testů

Pro testování efektivnosti hašovacích tabulek bylo implementováno pět typů testů, u každého z testů je možné nastavovat různé parametry. Testy 1-4 používají jako klíče náhodně generovaná 64-bitová celá čísla, test 5 používá řetězce načítané ze souboru. Testy se spouštějí příkazem:

jméno_programu číslo_testu parametry_testu

Číslo testu může být 1 až 5, význam parametrů je popsán u příslušných testů.

Některé parametry společné pro všechny testy:

- Počet běhů – u všech testů je to první parametr. Test je spouštěn opakovaně na stejných datech a pro každý běh je zaznamenán čas. Spuštění testu s počtem běhů *n* je rychlejší než spustit test *n*-krát, neboť není nutné pokaždé provádět inicializaci a generovat testovací data.
- Minimální faktor naplnění vynásobený 100 – určuje minimální faktor naplnění udržovaný v tabulce. Pokud při mazání poklesne faktor naplnění pod danou mez, je provedena kontrakce tabulky. Důvod pro tento způsob zadávání (vynásobení 100) je důsledkem kompromisu mezi efektivností a flexibilitou: Výpočty s reálnými čísly jsou značně pomalejší než výpočty s celými čísly. V tabulkách je proto výhodné udržovat požadovaný faktor naplnění jako celé číslo. Takové omezení by ale bylo příliš velké. Kompromisem je pracovat s faktorem naplnění vynásobeným nějakou hodnotou (z hlediska zadávání jsou vhodné mocniny 10). Pak je možné faktor naplnění dostatečně jemně škálovat a přitom stále pracovat s celými čísly.
- Maximální faktor naplnění vynásobený 100 – určuje maximální faktor naplnění udržovaný v tabulce. Pokud při vkládání vzroste faktor naplnění nad danou mez, je provedena expanze tabulky.
- Inicializační číslo pro náhodný generátor – přirozené číslo od 0 do 4 294 967 295 (2^{32}). Pokud je test opakovaně spouštěn se stejnými parametry, běží vždy na stejných datech.

Test 1 - Vkládání a mazání

Tento test začíná s prázdnou tabulkou. Postupně do ní vloží určený počet záznamů a změří celkový spotřebovaný čas. Potom záznamy v náhodném pořadí smaže a opět změří spotřebovaný čas. Vkládané prvky a pořadí jejich mazání jsou vygenerovány před započetím vkládání a tudíž neovlivňují naměřený čas.

Parametry testu 1:

1. Počet běhů
2. Počet záznamů – určuje, kolik záznamů je do tabulky vloženo a následně smazáno.
3. Minimální faktor naplnění vynásobený 100
4. Maximální faktor naplnění vynásobený 100
5. Inicializační číslo pro náhodný generátor

Test 2 - Vyhledávání v průběhu expanze

Test začíná s prázdnou tabulkou. Postupně do ní vkládá určený počet záznamů, po každém vložení provede určený počet úspěšných vyhledání náhodně zvolených záznamů. Měřen je celkový čas potřebný pro všechna vkládání a vyhledání.

Účelem tohoto testu je zjistit chování tabulky za situace, kdy nedochází k častým změnám obsahu tabulky.

Parametry testu 2:

1. Počet běhů
2. Počet vkládaných záznamů
3. Počet vyhledání po každém vložení
4. Minimální faktor naplnění vynásobený 100
5. Maximální faktor naplnění vynásobený 100
6. Inicializační číslo pro náhodný generátor

Test 3 - Největší prodlevy při vkládání a mazání

Test zjišťuje nejdelší časy potřebné na vkládání a mazání jednoho záznamu. Tento test byl navržen zejména pro zjištění prodlev standardní (pseudodynamické) hašovací

tabulky při přehašovávání záznamů při expanzi a kontrakci. U ostatních (dynamických) tabulek se dá očekávat, že tyto časy budou velmi malé.

Test začíná s prázdnou tabulkou a postupně do ní vkládá určený počet záznamů. Čas potřebný na každé vložení je změřen a je udržován stanovený počet nejvyšších časů. Potom jsou záznamy s tabulky postupně mazány a jsou měřeny časy jednotlivých mazání. Opět je udržován stanovený počet nejvyšších časů.

Parametry testu 3:

1. Počet běhů
2. Počet záznamů
3. Počet nejvyšších časů
4. Minimální faktor naplnění vynásobený 100
5. Maximální faktor naplnění vynásobený 100
6. Inicializační číslo pro náhodný generátor

Test 4 - Náhodné posloupnosti operací

Tento test umožňuje provádět náhodné posloupnosti operací vkládání, mazání a vyhledávání se zadaným poměrem počtů těchto operací.

Test nejprve do tabulky vloží určený počet záznamů a poté z ní (jiný) určený počet záznamů smaže. Teprve nyní začíná měření času, při kterém je provedena posloupnost operací dané délky. Každá operace je náhodně vybraná ze čtyř možností: vkládání, mazaní, úspěšné vyhledání, neúspěšné vyhledání záznamu. Pravděpodobnosti zvolení operací jsou dány parametry testu.

Dále je celkový počet záznamů v tabulce udržován ve stanovených mezích. To znamená, že pokud by měl počet záznamů klesnout pod dolní hranici, operace Delete se neprovede, místo ní je náhodně zvolena jiná. Analogicky se neprovede operace Insert, pokud by měl počet záznamů v tabulce překročit horní hranici.

Parametry testu 4:

1. Počet běhů
2. Počet operací
3. Počet záznamů vložených do tabulky před započetím měření času
4. Počet záznamů smazaných před započetím měření času
5. Minimální počet záznamů udržovaný v tabulce

6. Maximální počet záznamů udržovaný v tabulce
7. Podíl operací Insert (v procentech)
8. Podíl operací Delete (v procentech)
9. Podíl operací Find - úspěšný případ – vyhledání náhodně zvoleného záznamu, který se v tabulce nachází
10. Podíl operací Find - neúspěšný případ – vyhledání náhodného záznamu, který se v tabulce nenachází
11. Minimální faktor naplnění vynásobený 100
12. Maximální faktor naplnění vynásobený 100
13. Inicializační číslo pro náhodný generátor

Test 5 – vkládání, mazání a vyhledávání na řetězcích

Test nejprve načte řetězce ze zadaného souboru do poměti. Pak tyto záznamy vkládá do tabulky a měří potřebný čas. Potom provede určený počet úspěšných vyhledávání náhodně zvolených záznamů a opět změří spotřebovaný čas. Nakonec změří čas potřebný na smazání záznamů z tabulky. Záznamy jsou mazány v náhodném pořadí.

Parametry testu 5:

1. Počet běhů
2. Jméno souboru s řetězci
3. Počet vyhledávání
4. Minimální faktor naplnění vynásobený 100
5. Maximální faktor naplnění vynásobený 100
6. Inicializační číslo pro náhodný generátor

Výstupy testů

Testy ukládají své výsledky do souborů. Když test běží v režimu měření času, píše naměřené časy do souboru s následujícím jménem:

*typ_tabulky_číslo_testu_jméno_testu*_times*_parametry_testu*.txt

Pokud test zjišťuje počty porovnání, zapisuje zjištěné hodnoty do souboru

*typ_tabulky_číslo_testu_jméno_testu*_stats*_parametry_testu*.txt

3.1.5 Náhodný generátor

Jako náhodný generátor byl použit Mersenne Twister [8]. Tento generátor byl shledán rychlejší než standardní generátor náhodných čísel jazyka C++, navíc generuje 32-bitová čísla (oproti 16-bitovým).

3.2 Měření času

Měření času běhu programu je náročnější, než by se mohlo na první pohled zdát a v podstatě neexistuje univerzální návod, jak tuto úlohu řešit. Existují různé metody, jejichž vhodnost a použitelnost závisí na použitém hardwaru, operačním systému a povaze úlohy.

Nabízejí se dvě základní možnosti: Přerušení časovače a čítač cyklů procesoru.

Přerušení časovače

Operační systémy počítají pro každý proces čas strávený na procesoru, aby mohly procesor spravedlivě přidělovat. Tento čas lze zjistit pomocí systémového volání. K počítání tohoto času využívá operační systém přerušení časovače, které je generováno v určitých intervalech, obvykle 1 až 100 ms. V těch okamžicích připočítá časový interval procesu, který je právě prováděn, a může také rozhodnout o jeho přeplánování.

Přerušení, při kterých mohou být procesy přeplánovány, vznikají i mimo tyto časy. Může tedy nastat situace, kdy proces běží jen část intervalu, ale je mu připočítán celý, nebo naopak může být přeplánován těsně před přerušením časovače a interval mu přičten není. Takto zjištěný čas tedy může být větší nebo menší než čas skutečně strávený na procesoru. S největší pravděpodobností se ale vzniklé odchylky budou kompenzovat a zjištěný čas se nebude příliš lišit od skutečného.

Výhodou této metody je, že se dá použít i na zatíženém systému, nevýhodou nepříliš velká přesnost.

Čítač cyklů procesoru

Většina procesorů má speciální registr, jehož hodnota se v každém cyklu procesoru zvýší o 1. Tuto hodnotu lze zjistit pomocí speciální instrukce. Metoda je vhodná

zejména pro krátké úlohy, kdy běh procesu není přerušen. Pokud úloha trvá méně než přibližně 1 milisekundu, lze měření provádět i na zatíženém systému. Nevýhodou této metody je, že ne všechny procesory tuto funkci poskytují.

Při experimentech prováděných pro tuto práci byla použita první popsaná metoda (využívající přerušení časovače), neboť jednotlivé úlohy trvaly řádově desítky sekund a metoda měří skutečný čas strávený na procesoru.

3.3 Problémy zjištěné během testování

Během testování se objevily některé problémy a neočekávané výsledky, které musely být ošetřeny v dalším testování.

3.3.1 Alokace a dealokace paměti

Při zkušebním spuštění testu 1 (Vkládání a mazání) byly u spirálových hašovacích tabulek zjištěny velké rozdíly mezi časy vkládání a mazání – časy mazání byly několikrát větší. U lineární hašovací tabulky byl čas potřebný na mazání jen mírně větší než čas na vkládání.

K tomuto jevu docházelo jen na operačním systému Linux. Když byl stejný test spuštěn na témže počítači pod Windows, všechny tabulky potřebovaly na mazaní záznamů čas mírně kratší na vkládání. Toto byl výsledek, který se dal očekávat, neboť při mazání záznamů se tabulka zmenšuje a přemisťování záznamů při kontrakci není tak náročné jako při expanzi.

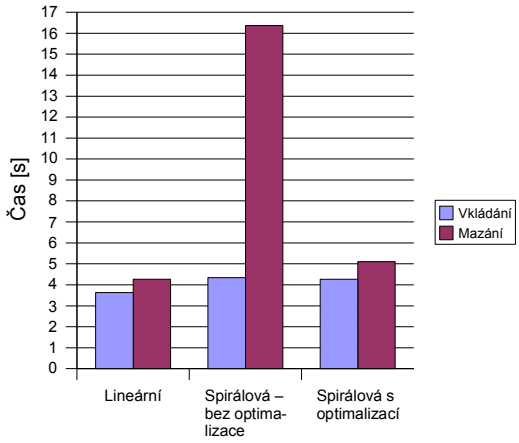
K vysvětlení problému spirálových hašovacích tabulek vedlo vyzkoušení jednoduché optimalizace. U spirálových hašovacích tabulek totiž dochází k posunu aktivního adresového prostoru. Proto při zvětšování tabulky jsou nejen alokovány nové bloky paměti, ale některé jsou i uvolňovány. Naopak při zmenšování tabulky jsou i některé bloky paměti alokovány.

Optimalizace spočívala v uchovávání jednoho záložního bloku paměti, který by se “mohl hodit“: Pokud má být nějaký blok paměti uvolněn, nejprve se zkontroluje, jestli je k dispozici záložní blok. Pokud ne, uloží se tento blok jako záložní. Pouze když je k dispozici jiný záložní blok, je nepotřebný blok uvolněn. Při potřebě nového bloku se

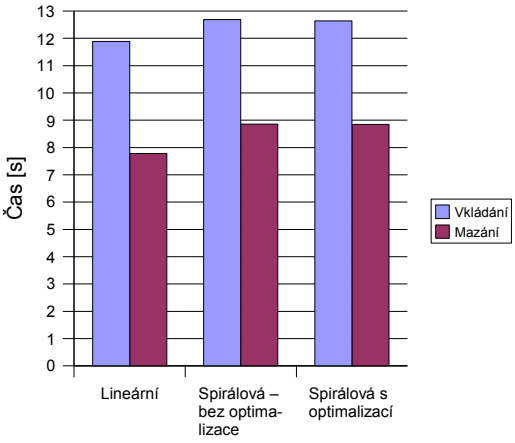
pak nejprve zjišťuje, jestli je k dispozici nějaký záložní. Tato optimalizace je možná díky tomu, že všechny takto alokované a dealokované bloky mají stejnou velikost.

Použitím této optimalizace klesly u spirálových hašovacích tabulek časy mazání mírně nad časy potřebné pro vkládání. Linux tedy zřejmě používá podstatně jiný systém správy paměti než Windows a při určité kombinaci požadavků na alokaci a dealokaci se jeho chování značně zhoršuje.

Grafy 1 a 2 zobrazují časy naměřené při použití testu 1 s parametry počet záznamů 4 194 304, minimální a maximální faktor naplnění 1 na Windows a Linuxu. (Tyto testy byly prováděny na jiném počítači než ostatní testy.)



Graf 1. Časy vkládání a mazání na Linuxu



Graf 2. Časy vkládání a mazání na Windows

Všechny spirálové hašovací tabulky tedy používají uvedenou optimalizaci.

3.3.2 Číselné chyby ve spirálových hašovacích tabulkách

Při výpočtu adres ve spirálových hašovacích tabulkách se pracuje s reálnými čísly a je zde tudíž prostor k určitým nepřesnostem. U spirálových hašovacích tabulek používajících jako expanzní funkci 2^x nebo 3^x (nikoliv aproximace) občas dochází k chybám, které vyúsťují ve snahu zařadit záznam do přihrádky, která již nemá být aktivní.

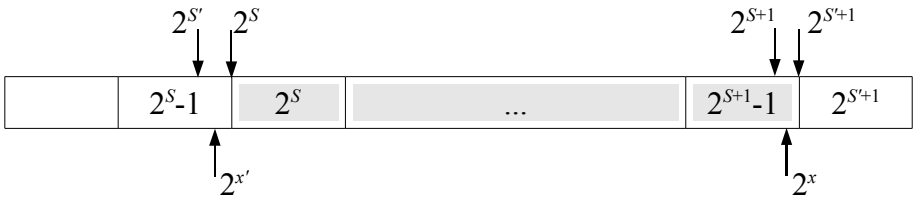
Chyby vznikají při výpočtu proměnné $S = \log_d(low\_addr)$, která určuje aktivní adresový prostor. Místo správné hodnoty S vyjde nějaká hodnota S', která je o něco menší. Pokud hodnota $h(K)$ nějakého vkládaného záznamu K padne do intervalu $[\{S'\}, \{S\})$ (kde $\{A\}$ značí desetinnou část čísla A) * je chybně spočítána hodnota

$$x' = \lceil \lfloor S' \rfloor + \{S'\} - h(K) \rceil + h(K)$$

namísto správného

$$x = \lceil \lfloor S \rfloor + \{S\} - h(K) \rceil + h(K).$$

Protože čísla S a S' mají stejnou dolní celou část, liší se tyto dva výrazy jen v hodnotách $\{S\}$ a $\{S'\}$. Protože ale $\{S'\} - h(K) \leq 0$ a $\{S\} - h(K) > 0$, je hodnota x' o jedničku menší než x. Potom x neleží v intervalu $[S, S+1)$ a následně spočítaná adresa $y' = \lfloor d^{x'} \rfloor$ pak ukazuje na přihrádku, která je již považována za neaktivní. Situaci v tabulce zachycuje následující obrázek (aktivní adresový prostor je šedě podbarven).



Obrázek 7. Spirálová hašovací tabulka při chybném výpočtu adresy přihrádky

K chybám tohoto typu docházelo zejména u velkých tabulek (obsahujících alespoň 16 000 000 záznamů). Protože však na testovacím počítači zbývalo při vytvoření tak velkých tabulek jen velmi málo volné fyzické paměti, testy byly prováděny s menšími tabulkami. Během těchto testů došlo k uvedené chybě jen jednou. Protože v této situaci dojde ke zhroucení programu, měření bylo opakováno s odlišnou inicializací generátoru náhodných čísel.

Uvedený jev byl pozorován jen u spirálových hašovacích tabulek, které používají jako expanzní funkci d^x a ne její aproximaci. K nepřesnosti tedy dochází zřejmě jen u výpočtu náročnějších funkcí, jako je logaritmus, a to u větších tabulek, kde více záleží i

* Zde se předpokládá, že hodnoty S a S' mají stejnou dolní celou část. Uvedená situace by mohla nastat, i kdyby tomu tak nebylo. Popis situace by pak byl jiný, vedl by ale ke stejnému výsledku.

na méně významných desetinných místech spočítaných čísel. U aproximačních funkcí používajících jen základní aritmetické operace žádné takovéto jevy pozorovány nebyly a přesnost se tedy zdá být dostačující.

Všichni autoři dostupné literatury používají jen aproximační funkce a o možnosti vzniku uvedeného problému se nezmiňují.

Pro odstranění tohoto problému by možná pomohlo přesnější počítání funkce logaritmus, což by ale znamenalo větší výpočetní náročnost.

Jinou možností je připustit nepřesnost výpočtu a považovat dotčenou přihrádku za aktivní. To by ale zkomplikovalo expanzi a kontrakci tabulky, a tyto tabulky by tak ztratily výhodu, kterou mají oproti tabulkám s aproximační funkcí.

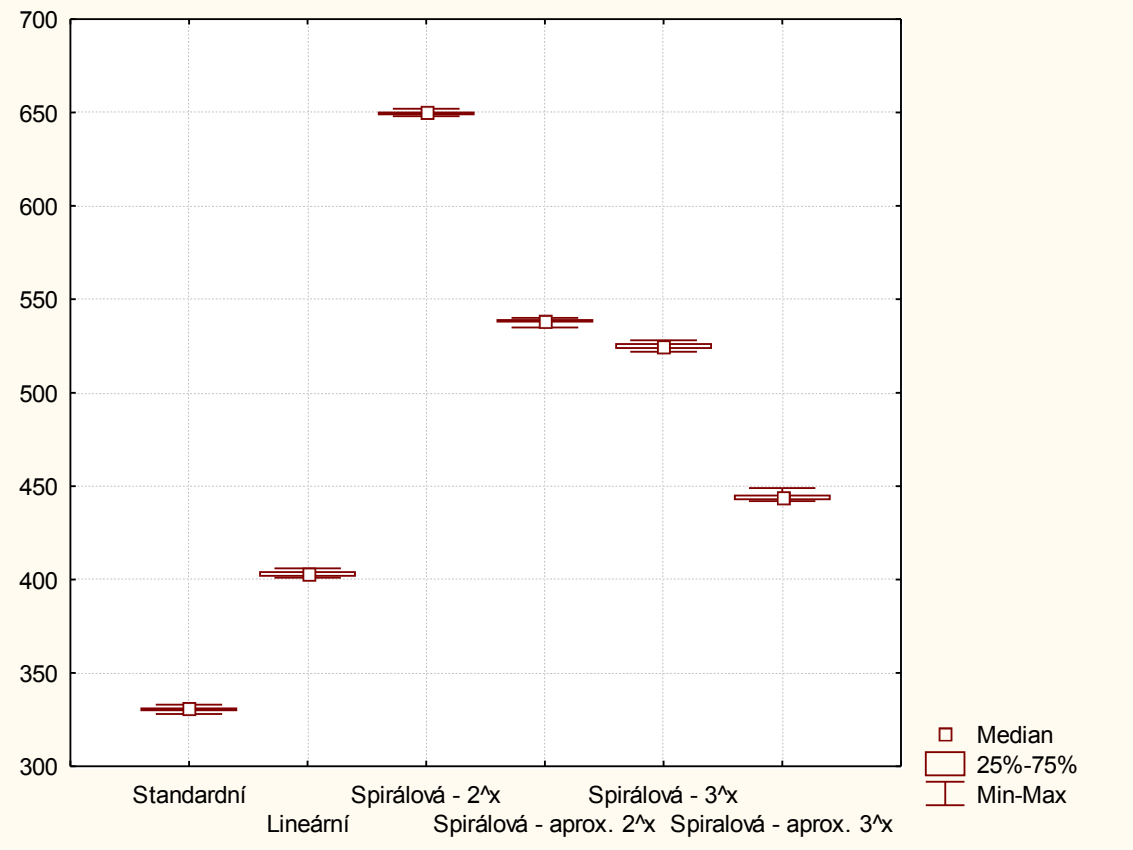
3.4 Realizované testy

Původním plánem bylo provádět každý test pro větší množství inicializací náhodného generátoru (tedy různá vstupní data) a ostatní parametry stejné a porovnávat rozmezí, ve kterých se získané časy pohybují. Získané časy se však ukázaly být velmi vyrovnané.

Jako příklad mohou posloužit časy testu 1 potřebné na vkládání. Test byl spouštěn s počtem vkládaných záznamů 8 388 608 pro pět různých faktorů naplnění – pokaždé pro 25 různých inicializací náhodného generátoru. Graf 3 zobrazuje krabicový diagram s rozmezími časů potřebných pro vkládání při faktoru naplnění 1,0.

Podobně vyrovnaných výsledků bylo dosaženo i pro ostatní faktory naplnění i jiné testy. Vzhledem k tomuto faktu a časové náročnosti testů byl u dalšího testování snížen počet inicializací náhodného generátoru a dále se počítá jen se středními hodnotami získaných časů.

Následující testy jsou tedy spouštěny (pokud není uvedeno jinak) pro pět různých inicializací náhodného generátoru. Pokaždé jsou provedena čtyři měření (počet běhů je čtyři), z těchto časů se vždy vybere minimum. Výsledný čas je pak brán jako průměr těchto pěti minim.



Graf 3. Časy vkládání 8 388 608 záznamů při faktoru naplnění 1,0

Parametry testovacího počítače

Testy byly prováděny na počítači s následující konfigurací:

Procesor: AMD 64 3200+ (2.2GHz) – socket 754

Operační paměť: 1024 MB

Operační systém: Gentoo Linux

Faktor naplnění u standardní hašovací tabulky

Pokud je nějaké dynamické hašovací tabulce předán jako minimální (resp. maximální) faktor naplnění α, je v této tabulce při mazání (resp. vkládání) průměrný faktor naplnění udržován na této hodnotě. U standardní hašovací tabulky je však pro udržování faktoru naplnění α nutné zadávat jako minimální faktor naplnění $2\alpha/3$ a maximální faktor naplnění $4\alpha/3$ (jak bylo popsáno v příslušné analýze). Tato zásada je

dodržena u všech testů. Např. tedy výsledek lineární hašovací tabulky s minimálním a maximálním faktorem naplnění 1,5 je vždy srovnáván s výsledkem standardní hašovací tabulky s minimálním faktorem naplnění 1,0 a maximálním faktorem naplnění 2,0.

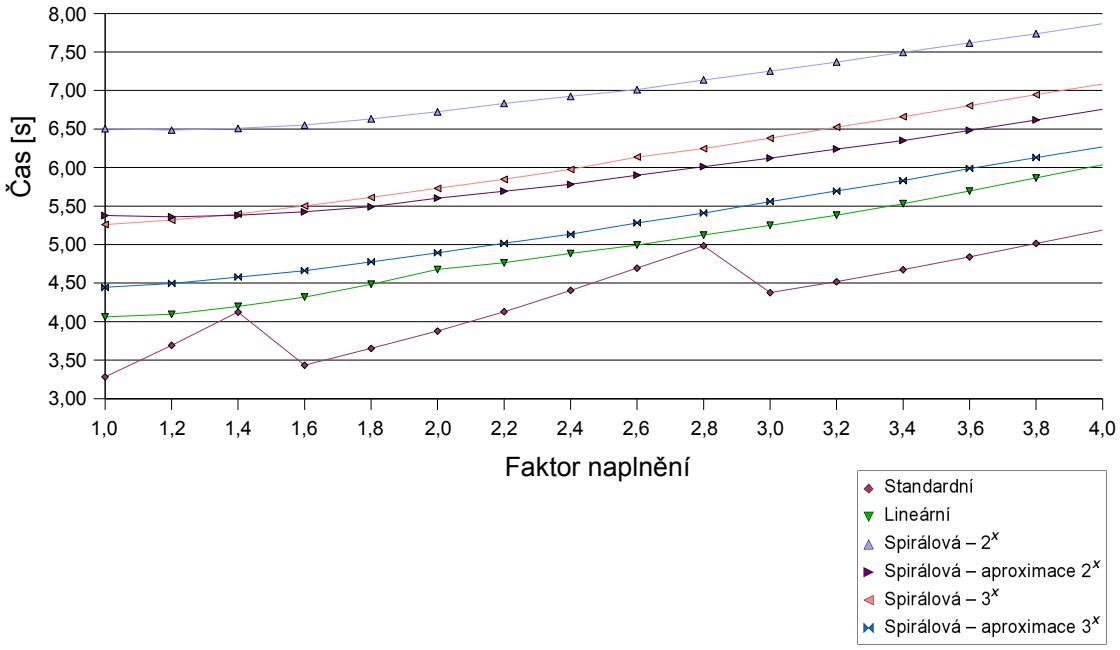
V tomto textu jsou u většiny testů uváděny jen závěry a některé vybrané hodnoty, kompletní výsledky se nacházejí na přiloženém CD.

3.4.1 Vkládání

Test 1 byl spouštěn pro konstantní počet záznamů (8 388 608) a různé hodnoty faktoru naplnění. Výsledky týkající se vkládání jsou následující:

Spotřebovaný čas

Závislost času běhu na faktoru naplnění zobrazuje následující graf.



Graf 4. Závislost času vkládání na faktoru naplnění

Z grafu lze vyčíst následující informace:

Z hlediska rychlosti je pro všechny tabulky nejlepší udržovat nízký faktor naplnění (kolem 1), s rostoucím faktorem naplnění se mírně zvyšují spotřebované časy.

Nejlépe dopadla standardní hašovací tabulka, která ale nedokáže svou velikost zvětšovat plynule. Dále jsou u této tabulky patrné poklesy spotřebovaného času i při jistých zvětšení faktoru naplnění. To je způsobené tím, že s rostoucím faktorem naplnění se odkládají i expanze. Při určitém malém zvýšení faktoru naplnění se pak poslední expanze již neprovede a ušetří se tak nezanedbatelný čas.

Z dynamických tabulek dopadla nejlépe lineární, která byla přibližně o 10-20% pomalejší než standardní.

Časy všech verzí spirálových hašovacích tabulek pak byly ještě větší. To lze odůvodnit náročnějším způsobem výpočtu adres záznamů. Dále je vidět, že obě spirálové tabulky používající pouze aproximaci funkce d^x jsou rychlejší než ty, které používají přesnou funkci d^x pro dané d. Jednodušší způsob výpočtu adresy se tedy vyplácí i při komplikovanější expanzi (viz kapitola 2.2.1 Popis spirálového hašování, sekce Anomálie při použití jiných expanzních funkcí).

Jako nejlepší expanzní funkce pro spirálovou hašovací tabulku se v tomto testu jeví aproximace 3^x.

Počty porovnání klíčů

Následující tabulka zachycuje očekávané a zjištěné počty porovnání klíčů při vkládání záznamů. Při vkládání záznamu se nejprve kontroluje, zda záznam v tabulce již není, počet porovnání klíčů by tedy měl být stejný jako u neúspěšného vyhledávání. Vybrány jsou jen některé hodnoty faktoru naplnění, kompletní výsledky se nacházejí na přiloženém CD.

		Faktor naplnění				
		1,0	1,4	2,0	3,0	4,0
Standardní	spočítaný	1,0000	1,4000	2,0000	3,0000	4,0000
	pozorovaný	0,9423	1,3744	1,8913	3,0002	3,7756
Lineární	spočítaný	1,0833	1,5167	2,1667	3,2500	4,3333
	pozorovaný	1,0835	1,5140	2,1671	3,2363	4,3332
Spirálová – 2^x	spočítaný	1,0407	1,4570	2,0814	3,1221	4,1627
	pozorovaný	1,0408	1,4571	2,0817	3,1222	4,1630
Spirálová – aproximace 2^x	spočítaný	1,0407	1,4570	2,0814	3,1221	4,1627
	pozorovaný	1,0404	1,4573	2,0811	3,1235	4,1622
Spirálová – 3^x	spočítaný	1,1047	1,5466	2,2094	3,3141	4,4189
	pozorovaný	1,1047	1,5466	2,2095	3,3141	4,4187
Spirálová – aproximace 3^x	spočítaný	1,1047	1,5466	2,2094	3,3141	4,4189
	pozorovaný	1,1036	1,5495	2,2105	3,3115	4,4267

Tabulka 2. Počty porovnání klíčů při vkládání

Z tabulky je vidět, že pozorované hodnoty kromě šedě podbarvených hodnot velmi dobře odpovídají očekávaným hodnotám. U standardní hašovací tabulky platí očekávané hodnoty pro případ, kdy je právě provedena expanze. To platí jen pro určité kombinace faktoru naplnění a počtu vkládaných záznamů. Některé další odhady nevycházejí přesně, protože při zadávání mezních faktorů naplnění dochází k jejich přepočítávání a tudíž i zaokrouhlování (na dvě desetinná místa). U lineární hašovací tabulky, podobně jako u standardní, platí spočítané hodnoty pro situaci, kdy je právě dokončena expanze. Šedě podbarvené hodnoty jsou tedy pouze orientační.

Nejmenší počty porovnání má standardní hašovací tabulka. Tohoto výsledku dosahuje díky stále rovnoměrnému rozdělení záznamů mezi všechny přihrádky, které dynamické metody nemají.

Z dynamických tabulek má nejmenší počty porovnání spirálová hašovací tabulka s expanzní funkcí 2^x (resp. s její aproximací), těsně následovaná lineární hašovací tabulkou a spirálovou tabulkou s expanzní funkcí 3^x.

Přestože nejmenší počty porovnání ze spirálových hašovacích tabulek má ta s expanzní funkcí 2^x (resp. s její aproximací), umístila se až na posledním místě. Nepříliš velké rozdíly v počtu porovnání klíčů tedy zřejmě nemají dominantní vliv na výsledný čas.

Počty přehašovávaných záznamů při expanzích

Následující tabulka zachycuje očekávané a zjištěné počty přehašovávaných záznamů při expanzích přepočítané na jeden vložený záznam.

		Faktor naplnění				
		1,0	1,4	2,0	3,0	4,0
Standardní	spočítaný	1,0000	1,0000	1,0000	1,0000	1,0000
	pozorovaný	1,3300	1,8700	1,3350	1,0000	1,3325
Lineární	spočítaný	1,5000	1,5000	1,5000	1,5000	1,5000
	pozorovaný	1,5000	1,4142	1,5000	1,4166	1,5000
Spirálová – 2^x	spočítaný	1,4427	1,4427	1,4427	1,4427	1,4427
	pozorovaný	1,4428	1,4427	1,4428	1,4427	1,4428
Spirálová – aproximace 2^x	spočítaný	1,4427	1,4427	1,4427	1,4427	1,4427
	pozorovaný	1,4426	1,4430	1,4426	1,4421	1,4426
Spirálová – 3^x	spočítaný	0,9102	0,9102	0,9102	0,9102	0,9102
	pozorovaný	0,9102	0,9102	0,9102	0,9102	0,9102
Spirálová – aproximace 3^x	spočítaný	0,9102	0,9102	0,9102	0,9102	0,9102
	pozorovaný	0,9135	0,9126	0,9057	0,9135	0,9135

Tabulka 3. Počty přehašovávaných záznamů při expanzi na jeden vložený záznam

Počty přehašovávaných záznamů také velmi dobře odpovídají teoretickým výpočtům (až na některé hodnoty). U stadnardní a lineární hašovací tabulky opět platí očekávané počty jen pro situace, kdy je dokončena expanze.

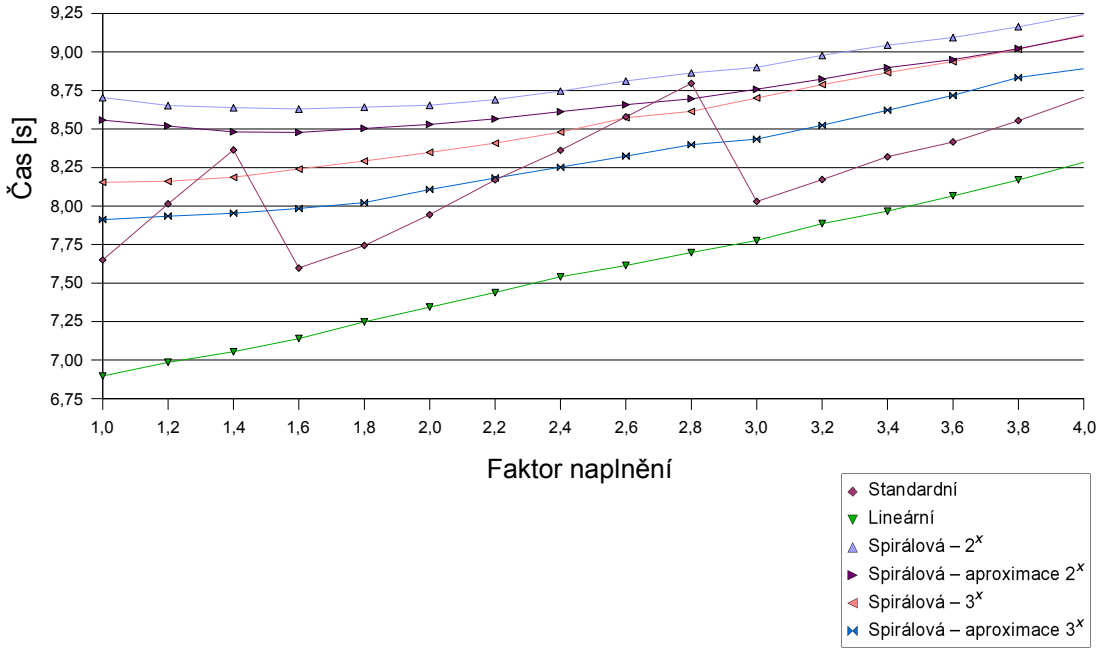
Nejmenší počty přehašovávaných záznamů má jednoznačně spirálová hašovací tabulka s expanzní funkcí 3^x (resp. s její aproximací). Díky tomu dosahuje lepších časů než spirálová tabulka s funkcí 2^x. Náročnost výpočtu hašovací funkce spirálové tabulky je však zřejmě stále příliš velká na to, aby byla rychlejší než lineární hašovací tabulka.

3.4.2 Mazání

Následující výsledky opět pocházejí z testu 1 pro počet záznamů 8 688 608 a různé faktory naplnění.

Spotřebovaný čas

Čas potřebný pro smazání všech záznamů z tabulky v závislosti na udržovaném faktoru naplnění zobrazuje následující graf



Graf 5. Závislost času mazání na faktoru naplnění

Graf vypadá podobně jako u vkládání, rozdíl je zejména v podstatně horším chování standardní hašovací tabulky. U té opět dochází k poklesům spotřebovaného času i při zvýšení faktoru naplnění (v důsledku neprovedení poslední expanze). Kromě toho však tentokrát dopadla jednoznačně hůře než lineární hašovací tabulka a někdy i hůře než některé spirálové tabulky. Tento výsledek je překvapivý vzhledem k tomu, že při vkládání dopadla standardní hašovací tabulka jednoznačně lépe než lineární a při tom u obou tabulek je kontrakce mírně jednodušší než expanze.

Jako vysvětlení se nabízí způsob správy paměti. Standardní hašovací tabulka totiž při kontrakci nejen uvolňuje, ale i alokuje novou paměť, což, jak bylo popsáno v kapitole 3.1.3, může značně zvětšit potřebný čas. Tento problém nelze řešit podobným způsobem jako u spirálových tabulek, neboť zde se pracuje s bloky paměti různé velikosti.

Ostatní výsledky jsou podobné jako u vkládání. Z dynamických tabulek dopadla nejlépe lineární. Ze spirálových tabulek jsou opět rychlejší ty s aproximačními expanzními funkcemi než ty s přesně exponenciálními. Spirálové tabulky s expanzní funkcí 3^x resp. její aproximací jsou rychlejší než ty s 2^x.

Počty porovnání klíčů

Následující tabulka zachycuje očekávané a zjištěné počty porovnání klíčů při mazání záznamů. Při mazání se nejprve musí záznam v tabulce vyhledat, počet porovnání klíčů by tedy měl být stejný jako při úspěšném vyhledávání. Vybrány jsou jen některé hodnoty faktoru naplnění, kompletní výsledky se nacházejí na přiloženém CD.

		Faktor naplnění				
		1,0	1,4	2,0	3,0	4,0
Standardní	očekávaný	1,5000	1,7000	2,0000	2,5000	3,0000
	pozorovaný	1,4745	1,6826	1,9424	2,4999	2,8913
Lineární	očekávaný	1,5417	1,7583	2,0833	2,6250	3,1667
	pozorovaný	1,5418	1,7570	2,0837	2,6184	3,1670
Spirálová – 2^x	očekávaný	1,5203	1,7285	2,0407	2,5610	3,0814
	pozorovaný	1,5203	1,7287	2,0405	2,5609	3,0810
Spirálová – aproximace 2^x	očekávaný	1,5203	1,7285	2,0407	2,5610	3,0814
	pozorovaný	1,5202	1,7289	2,0404	2,5618	3,0811
Spirálová – 3^x	očekávaný	1,5524	1,7733	2,1047	2,6571	3,2094
	pozorovaný	1,5522	1,7732	2,1047	2,6567	3,2093
Spirálová – aproximace 3^x	očekávaný	1,5524	1,7733	2,1047	2,6571	3,2094
	pozorovaný	1,5518	1,7748	2,1051	2,6556	3,2132

Tabulka 4. Počty porovnání klíčů při mazání

Zjištěné počty porovnání klíčů i zde velmi dobře odpovídají očekávaným hodnotám.

U standardní a lineární hašovací tabulky platí očekávané hodnoty opět jen pro určitou kombinaci počtu vkládaných záznamů a faktoru naplnění, kdy při zahájení mazání byla právě dokončena expanze.

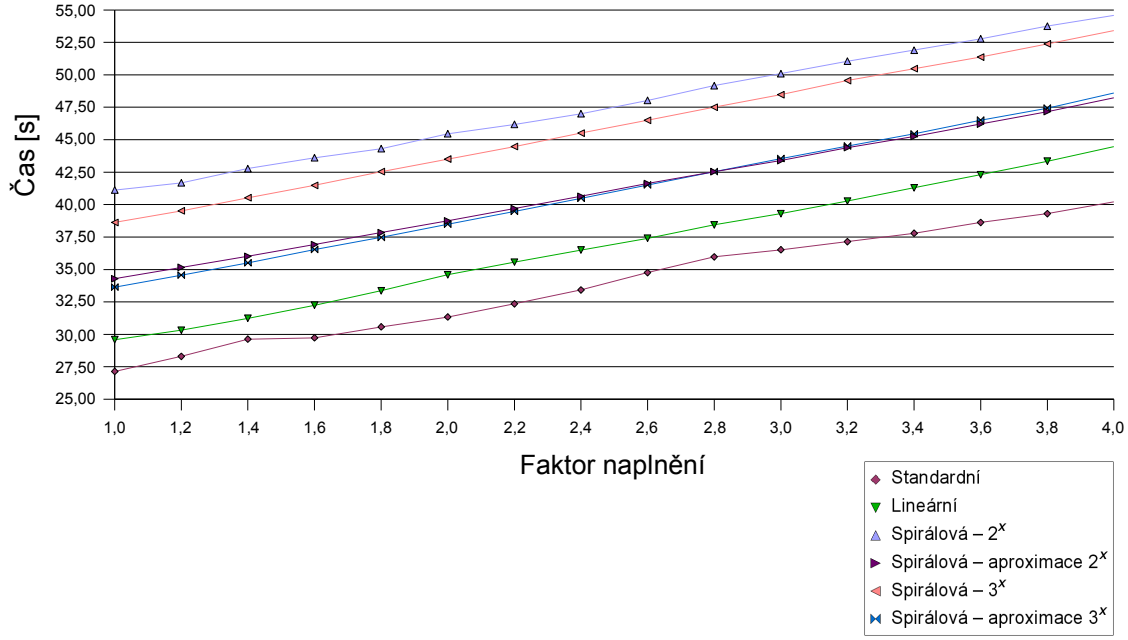
Nejmenší počty porovnání má opět standardní hašovací tabulka díky stále rovnoměrnému rozmisťování záznamů do tabulky. Následuje spirálová hašovací tabulka s expanzní funkcí 3^x (resp. její aproximací), dále lineární hašovací tabulka a spirálová hašovací tabulka s expanzní funkcí 2^x.

3.4.3 Vyhledávání

Výsledky testu 2 pro počet vkládaných záznamů 8 388 608, počet vyhledání po každém vkládání 10 a různé faktory naplnění.

Spotřebovaný čas

Závislost času běhu na faktoru naplnění zobrazuje následující graf.



Graf 6. Závislost času vyhledávání na faktoru naplnění

Při vyhledávání se více projevuje význam nízkého faktoru naplnění – s jeho růstem se zvětšuje i spotřebovaný čas u všech tabulek přibližně lineárně. Jinak jsou výsledky podobné jako u vkládání a mazání.

Nejlepších časů dosahuje opět standardní hašovací tabulka, z dynamických tabulek pak lineární hašovací tabulka. Ze spirálových tabulek dopadly nejlépe ty s aproximačními expanzními funkcemi, které dosahují přibližně stejných časů.

3.4.4 Maximální časy expanze a kontrakce

Tento test zkoumá, k jakým nejdelším prodlevám dochází při expanzi a kontrakci, přesněji řečeno měří maximální časy vkládání (a tím i expanze) a mazání (a tím i kontrakce). Zde jsou uvedeny jen časy pro standardní hašovací tabulku. U dynamických tabulek byly největší naměřené časy 0,01s, což je nejmenší nenulová hodnota, kterou lze použitou metodou měření času zjistit, tudíž ve skutečnosti jsou tyto časy nejspíš ještě menší.

Výsledky byly získány pomocí testu 3 s počtem běhů 10, počtem vkládaných záznamů 4 500 000 a minimálním a maximálním faktorem naplnění 1,0.

Při těchto parametrech se dosahovaly největší časy vkládání 0,29 až 0,30 sekundy, největší časy mazání 0,71 až 0,77 sekundy. Takové hodnoty již jsou při běhu programu postřehnutelné a ukazuje se zde význam dynamických hašovacích tabulek, které takové prodlevy nemají.

Časy mazání vycházejí větší než časy vkládání, přestože kontrakce je jednodušší než expanze; což opět nejspíše ukazuje na speciální způsob správy paměti na Linuxu.

3.4.5 Náhodné posloupnosti operací

Test 4 byl spouštěn s následujícími parametry:

Počet operací: 67 554 432

Počet záznamů vložených před započetím měření času: 4 197 304

Počet záznamů smazaných před započetím měření času: 0

Minimální počet záznamů udržovaný v tabulce: 524 288

Maximální počet záznamů udržovaný v tabulce: 7 864 320

Podíly operací vkládání, mazání, úspěšné vyhledávání a neúspěšné vyhledávání: každá z operací 25%.

Dále různé hodnoty minimálního a maximálního faktoru naplnění a různé poměry mezi nimi. Tyto poměry nabývají jedné ze tří hodnot: 1:1, 1:1,5 a 1:2.

Při konstantním poměru mezi maximálním a minimálním faktorem naplnění rostly spotřebované časy spolu s faktory naplnění. Následující tabulka zobrazuje časy pro minimální faktor naplnění 1,0, detailní výsledky jsou na přiloženém CD.

Typ tabulky	Maximální faktor naplnění		
	1,0	1,5	2,0
Standardní		21,94	28,51
Lineární	23,04	25,10	26,26
Spirálová – 2^x	34,98	30,97	32,94
Spirálová – aproximace 2^x	30,05	27,66	29,22
Spirálová – 3^x	34,36	31,76	33,50
Spirálová – aproximace 3^x	30,06	27,83	30,25

Tabulka 5. Časy testu 4 při minimálním faktoru naplnění 1,0

Standardní hašovací tabulka potřebuje v situacích, kdy se střídavě vkládají a mažou záznamy, aby byl mezi minimálním a maximálním faktorem naplnění nějaký rozdíl. Jinak může docházet k tomu, že vložením jednoho záznamu dojde k expanzi a následným smazáním opět ke kontrakci, což u velkých tabulek prakticky způsobí jejich nepoužitelnost. Proto u ní nejsou prováděny testy se stejným minimálním a maximálním faktorem naplnění.

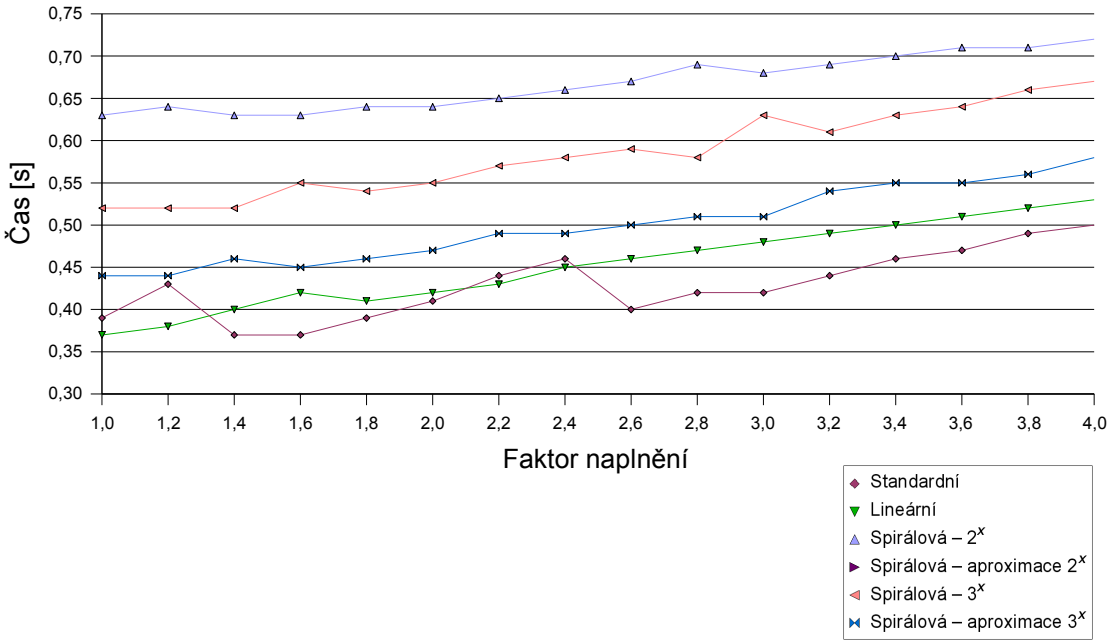
V tomto testu nejlépe vychází standardní hašovací tabulka pro poměr 1:1,5, z dynamických tabulek pak lineární s poměrem 1:1. U všech spirálových tabulek se jeví jako nejlepší poměr 1:1,5. Ze spirálových tabulek zde vychází nejlépe ta s expanzní funkcí aproximace 2^x, těsně následovaná aproximací 3^x.

3.4.6 Testy na řetězcích

Test 7 byl spouštěn s parametry počet běhů 10, soubor vstupních dat dict1.txt a různé faktory naplnění

Vkládání

Závislost času běhu na faktoru naplnění zobrazuje následující graf:

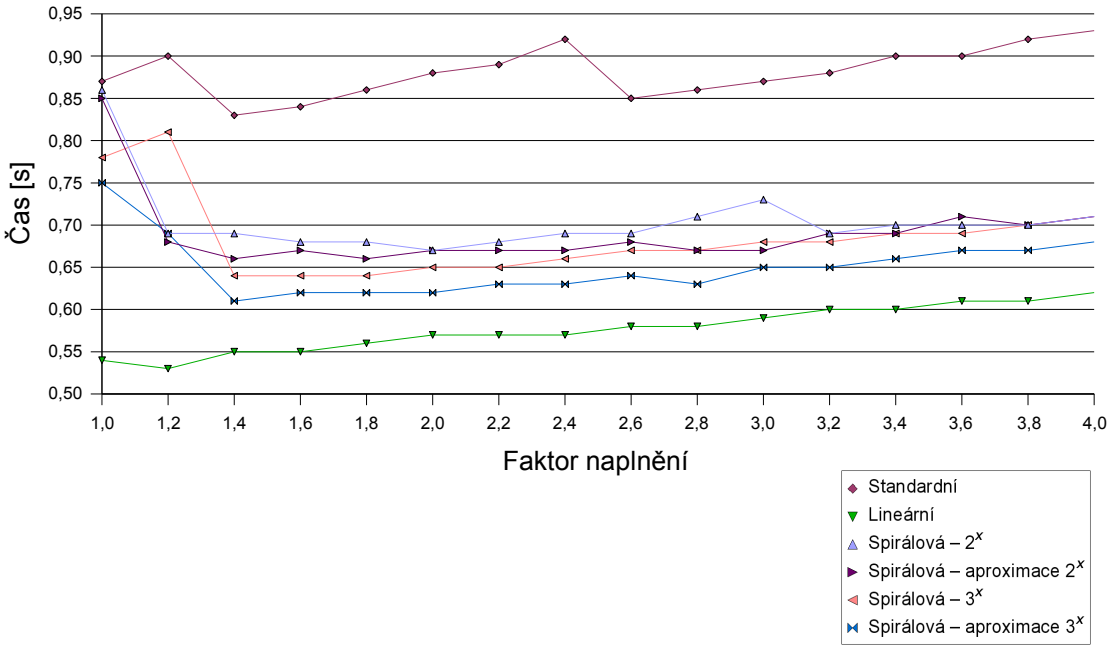


Graf 7. Závislost času vkládání na faktoru naplnění

Výsledky jsou podobné jako u experimentu s číselnými klíči, časové křivky jsou ale více zvlněné. Příčinou je ale nespíš fakt, že měřené časy byly hodně malé a projevila se nepřesnost měření. Jako nejlepší se opět jeví standardní hašovací tabulka, následovaná lineární tabulkou. Ze spirálových tabulek je nejlepší ta s expanzní funkcí aproximace 3^x.

Mazání

Závislost času mazání na faktoru naplnění zobrazuje následující graf.



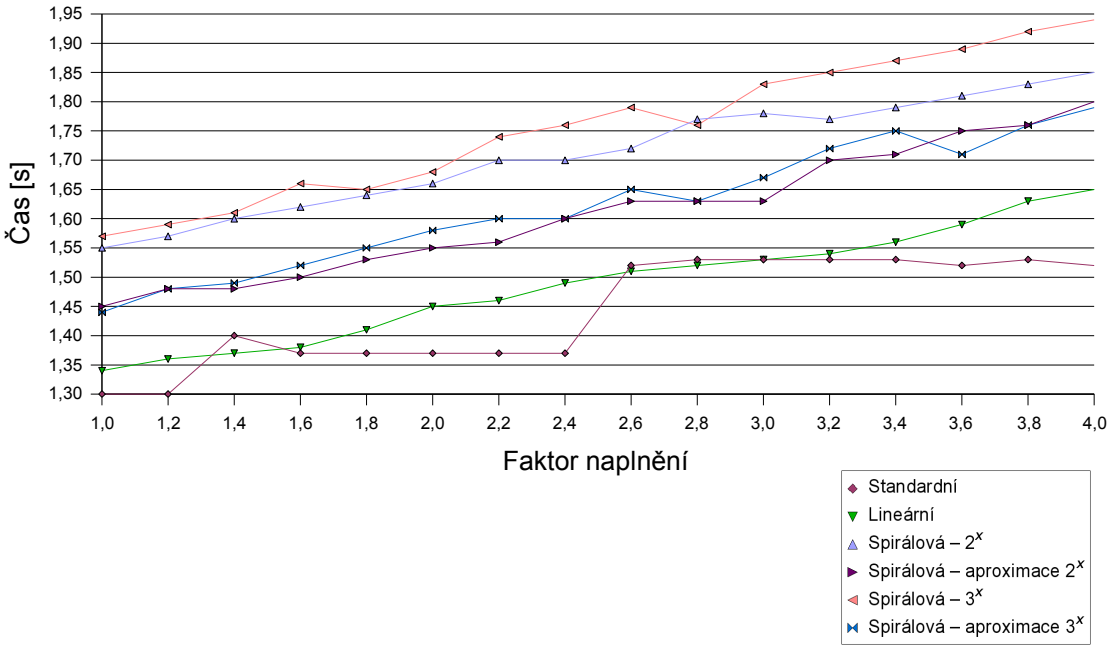
Graf 8. Závislost času mazání na faktoru naplnění

Zde se výsledky značně liší od výsledků s číselnými klíči. Standardní hašovací tabulka dopadla jednoznačně nejhůře. Nejlépe dopadla lineární a další pořadí je již podobné jako dříve.

Překvapivé jsou poměrně vysoké časy spirálových tabulek pro nízké faktory naplnění – u experimentů s číselnými klíči k takovým jevům nedocházelo, časy se vždy s rostoucím faktorem naplnění zvětšovaly. Pro toto odlišné chování jsem nenašel žádné vysvětlení. Při nižším faktoru naplnění totiž dochází jen častěji ke kontrakcím. Protože hašovací hodnota je uložena v tabulce spolu se záznamem, nedochází už k jejímu opětovnému přepočítávání, a samotná práce při kontrakci je stejná jako u číselných klíčů. Naopak při práci s řetězci je porovnání klíčů náročnější, a tudíž by se dalo očekávat, že se více projeví význam nízkého faktoru naplnění. Ani zjištěné počty porovnání klíčů při mazání a počty procházených záznamů při kontrakci se příliš nelišily od teoretických výsledků. Opakování experimentu vedlo k přibližně stejnému výsledku.

Vyhledávání

Závislost času vyhledávání na faktoru naplnění zobrazuje následující graf:



Graf 9. Závislost času vyhledávání na faktoru naplnění

Zde ve většině případů dopadla nejlépe standardní hašovací tabulka. Její graf připomíná po částech konstantní funkci, což je zcela v souladu s očekáváním: Velikost tabulky se totiž mění skokově – pro podobné hodnoty faktoru naplnění se provede stejný počet expanzí a tabulka má pak vždy stejnou velikost. Vzhledem ke stále stejnému počtu vkládaných záznamů je reálný faktor naplnění i při různých hodnotách udržovaného faktoru naplnění v okamžiku vyhledávání stejný, a mělo by tedy být i stejné chování tabulky. Ke skokovým zvýšením času dochází právě u těch faktorů naplnění, kdy u vkládání a mazání dochází ke snížení času. V tyto okamžiky se totiž snižuje počet provedených expanzí (resp. kontrakcí), což urychluje plnění a vyprazdňování tabulky, ale zvyšuje průměrný faktor naplnění, což zpomaluje vyhledávání.

U ostatních tabulek se spotřebovaný čas s rostoucím faktorem naplnění plynule zvyšuje. Z dynamických tabulek dopadla opět nejlépe lineární.

U spirálových tabulek je opět patrná výhoda rychlejšího výpočtu aproximačních expanzních funkcí. Na rozdíl od vkládání nebo mazání se zde projevuje význam rovnoměrného rozdělení záznamů v tabulce – tabulka s expanzní funkcí 2^x (resp. její aproximací) dopadla o trochu lépe než ta s expanzní funkcí 3^x (resp. její aproximací).

4. Srovnání s předchozími experimenty

Studií zkoumajících různé metody implementace dynamických hašovacích tabulek v interní paměti je dostupných velmi málo. Většina dynamických tabulek je zkoumána na externích pamětech.

Experimenty Larsona

Zde se pokusíme srovnat výsledky této práce s výsledky, ke kterým dospěl ve své studii Larson [4]. Ten z dynamických tabulek zkoumal jen lineární a spirálovou s expanzní funkcí aproximace 2^x a výsledky porovnával s nevyvažovaným binárním stromem a dvojitým hašováním. Testy prováděl na řetězcích, a to se třemi soubory dat:

Soubor A: Uživatelská jména, 10 000 záznamů

Soubor B: Anglický slovník pro kontrolu pravopisu, 20 000 záznamů

Soubor C: Čísla knihovních volání, 10 000 záznamů

Zjišťoval průměrné časy na vkládání a vyhledávání. Dospěl k následujícím výsledkům:

	Vkládání			Vyhledávání		
	$\alpha = 1$	$\alpha = 5$	$\alpha = 10$	$\alpha = 1$	$\alpha = 5$	$\alpha = 10$
Soubor A	0,88	0,97	1,15	0,34	0,41	0,50
Soubor B	0,94	1,02	1,20	0,36	0,44	0,53
Soubor C	1,06	1,23	1,53	0,41	0,53	0,69

Tabulka 6. Průměrné časy (v milisekundách) vkládání a vyhledávání jednoho záznamu v lineární hašovací tabulce

	Vkládání			Vhledávání		
	$\alpha = 1$	$\alpha = 5$	$\alpha = 10$	$\alpha = 1$	$\alpha = 5$	$\alpha = 10$
Soubor A	1,25	1,14	1,34	0,41	0,48	0,57
Soubor B	1,26	1,20	1,37	0,42	0,49	0,59
Soubor C	1,40	1,43	1,71	0,47	0,59	0,75

Tabulka 6. Průměrné časy (v milisekundách) vkládání a vyhledávání jednoho záznamu ve spirálové hašovací tabulce

Spirálová hašovací tabulka zde vychází při vkládání a faktoru naplnění 1 přibližně o 30-40% pomalejší, u vyšších faktorů naplnění jen o 15-20% pomalejší než lineární. Při

vyhledávání je pomalejší o 10-20%, s rostoucím faktorem naplnění se rozdíly opět snižují.

Vlastní výsledky

Následující tabulka shrnuje časy vkládání a vyhledávání hašovacích tabulek vytvořených pro tuto práci. Hodnoty pocházejí z testu s řetězci pro faktor naplnění 1,0. Časy jsou dále přepočítány na procentuální podíl času lineární hašovací tabulky.

	Čas vkládání		Čas vyhledávání	
Typ tabulky	v sekundách	% lineární tab.	v sekundách	% lineární tab.
Standardní	0,39	105	1,30	97
Lineární	0,37	100	1,34	100
Spirálová – 2^x	0,63	170	1,55	116
Spirálová – aproximace 2^x	0,52	141	1,45	108
Spirálová – 3^x	0,52	141	1,57	117
Spirálová – aproximace 3^x	0,44	119	1,44	107

Tabulka 7. Časy vkládání a vyhledávání

Na vkládání vychází spirálová hašovací tabulka s expanzní funkcí aproximace 2^x přibližně o 41% pomalejší tabulka lineární, což přibližně odpovídá Larsonovým výsledkům. Při vyhledávání je pak pomalejší asi o 8%. Zde Larson zaznamenal větší rozdíl.

5. Závěr

Přínos práce

V této práci byly prozkoumány různé dynamické hašovací tabulky a vytvořeny jejich implementace. Dále byly vytvořeny testy, které umožňují prakticky ověřit funkčnost a rychlost těchto implementací a zjistit skutečné počty porovnání klíčů při různých operacích. Chování dynamických tabulek bylo porovnáno s pseudodynamickou standardní hašovací tabulkou.

Ve většině testů dopadla nejlépe standardní hašovací tabulka, která ale nedokáže měnit svou velikost plynule. Při expanzích a kontrakcích u ní docházelo k prodlevám v řádu až desetin sekundy, což může v některých situacích vadit.

Při požadavku, aby žádná operace s tabulkou netrvala příliš dlouho, se jeví jako nejlepší volba lineární hašovací tabulka, která ve všech testech dopadla lépe než všechny implementace spirálových tabulek. Ty jsou pomalejší zřejmě zejména kvůli náročnějšímu výpočtu adres a nepomůže ani fakt, že některé mají menší průměrný počet porovnání klíčů při manipulacích s tabulkou.

U spirálových hašovacích tabulek se projevuje volba expanzní funkce jako faktor významně ovlivňující jejich chování. Tabulky s aproximačními funkcemi používajícími pouze základní matematické operace jsou znatelně rychlejší než ty s přesnými exponenciálními funkcemi. Dále tabulka se strmější expanzní funkcí 3^x (resp. její aproximací) dopadla lépe při testech s větším množstvím vkládání nebo mazání lépe než tabulka s expanzní funkcí 2^x – díky tomu, že v jednom kroku expanze (resp. kontrakce) přehašovává více záznamů. Díky tomu má menší průměrný počet přehašovávaných záznamů na jeden vložený (resp. smazaný) záznam. Při testech na vyhledávání dopadla díky menšímu počtu porovnání o trochu lépe tabulka s expanzní funkcí 2^x (resp. její aproximací).

Spirálové hašovací tabulky se tedy neukázaly příliš efektivní, větší efektivity snad mohou dosahovat u externích pamětí, kde tolik nezáleží na náročnosti výpočtu adres, ale hlavně na počtu přístupů na médium.

Dále bylo zjištěno, že počty porovnání klíčů u všech tabulek a všech operacích velmi dobře odpovídají teoretickým výsledkům.

Zkušenosti z praktických testů

Při spouštění testů se ukázalo, že cesta od teoretického popisu k praktickým výsledkům může přinést nečekané komplikace. Už samotné měření času není triviální záležitost. Dále způsob správy paměti na různých systémech se ukázal jako faktor významně ovlivňující naměřené časy. Projevilo se také nebezpečí nepřesností při práci s reálnými čísly.

Náměty pro další zkoumání

Jistě by bylo zajímavé provést větší množství testů na reálných datech a ne jen na náhodně generovaných číslech. Vzhledem k šablonovitému návrhu tabulek stačí naprogramovat příslušné datové typy a získat zdroje dat, do implementace tabulek není nutné nijak zasahovat.

Přestože se spirálové hašovací tabulky ukázaly být pomalejší než lineární tabulka, mohly by spirálové tabulky s ještě strmější expanzní funkcí (4^x, 5^x, ...) konkurovat lineární tabulce v situacích, kdy se rychle mění velikost tabulky a málo se vyhledává. Spirálové tabulky s těmito expanzními funkcemi mají totiž nezanedbatelně menší počet přehašovávaných záznamů na jeden vložený (resp. smazaný) záznam. Tento fakt měl znatelný vliv na naměřené časy.

U spirálových hašovacích tabulek by bylo dále zajímavé prozkoumat, za jakých okolností se mohou projevit nepřesnosti při počítání adres záznamů – jestli např. existuje možnost nesprávného výpočtu adresy i u tabulek s (jednoduchými) aproximačními expanzními funkcemi. To ale přesahuje rámec této práce.

Použitá literatura

[1] Bryant R. E., O'Hallaron D. R. (2003): Computer Systems: A Programmer's Perspective. *Prentice Hall*

[2] Enbody R. J., Du H. C. (1988): Dynamic Hashing Schemes. *ACM, 20, 85-113*

[3] Chu J-H, Knott G. D. (1994): An Analysis of Spiral Hashing. *The Computer Journal, 37, 715-719*

[4] Larson, P-Å (1988): Dynamic Hash Tables. *ACM, 31, 446-457*

[5] Martin G. N. N. Spiral Storage: Incrementally augmentable hash addressed storage. *Theory of Computation*

[6] Mehlhorn K. (1984): Data Structures and Algorithms I. Sorting and Searching. *Springer - Verlag*

[7] Pettersson M. (1993): Main-Memory Linear Hashing – Some Enhancements of Larson's Algorithm.

[8] Wagner R. J.: Mersenne Twister Random Number Generator http://www-personal.engin.umich.edu/~wagnerr/MersenneTwister.html