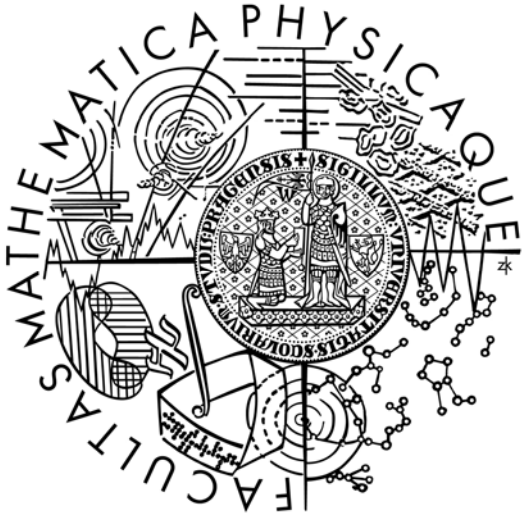
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Martin Špaňo

Vizualizace algoritmů

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D.
Studijní program: informatika, programování

2007

Poďakovanie
Ďakujem môjmu vedúcemu za jeho podporu počas vývoja projektu, jeho cenné rady ohľadom programu i textu tejto práce.

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

V Praze dne 25.5.2007

Martin Špaňo

Obsah

Název práce: Vizualizace algoritmů
Autor: Martin Špaňo
Katedra (ústav): Katedra softwarového inženýrství
Vedoucí bakalářské práce: RNDr. Filip Zavoral, Ph.D.
e-mail vedoucího: Filip.Zavoral@mff.cuni.cz

Abstrakt:

Cieľom projektu je vytvorenie prostredia, ktoré bude graficky zobrazovať priebehy rôznych algoritmov, bude ukazovať spôsob akým algoritmus z počiatočných dát (input) dospeje k výsledku (output). Dôraz je kladený na to, aby bol daný spôsob užívateľovi programu zrozumiteľný, teda aby si užívateľ prácu algoritmu rýchle osvojil. Projekt bude obsahovať modul pre distančnú výučbu, ktorý umožní užívateľovi, aby program využíval okrem vlastnej výučby aj pre účely prednášania princípov fungovania algoritmov na diaľku.

Spolu s prostredím je implementovaných niekoľko konkrétnych grafových, triediacich a textových algoritmov.

Klíčová slova: algoritmus, vizualizace, on-line přednášky

Title: Visualization of algorithms
Author: Martin Špaňo
Department: Department of software engineering
Supervisor: RNDr. Filip Zavoral, Ph.D.
Supervisor's email address: Filip.Zavoral@mff.cuni.cz

Abstract:

The aim of this project is to create an interface that visualizes the running of various algorithms, interface that shows the way how one algorithm works with initial data (input) to create result (output). The focus is on the simplicity of the visualization process that means to make user understand the principles of certain algorithm as quickly as possible. Project contains the unit for distant learning that enables the user to use the project not only for self-studying but also for on-line teaching.

The interface is accompanied by several graph, sorting and pattern-matching algorithms.

Keywords: algorithm, visualization, on-line courses

1. Kapitola - Úvod

1.1. Úvod, existujúce projekty a prínos tohto projektu

Témou tejto bakalárskej práce je vizualizácia priebehu algoritmov vhodnými grafickými prostriedkami. Je to jedna z najefektívnejších (čo sa rýchlosti osvojenia si algoritmu týka) metód výučby algoritmov.

Problémom vizualizácie algoritmov sa zaoberá viacero projektov. Na internete možno nájsť množstvo menších projektov. Odkazy na stránky s takýmito projektmi môžeme nájsť napríklad v [1]. Spoločným znakom týchto projektov je ich zameranie buď na jeden algoritmus, alebo niekoľko algoritmov náležiacich do určitej skupiny algoritmov. Používateľ sa musí najprv naučiť daný projekt (najčastejšie vo forme Java appletu) obsluhovať. Pri inom algoritme sa však znova najprv musí naučiť algoritmus používať, nakoľko odlišné projekty majú odlišné ovládanie, teda poznatky z obsluhy jedného algoritmu sú väčšinou nepoužiteľné pri obsluhe druhého algoritmu.

Podobný problém nastáva, pokiaľ by chcel upraviť zdrojový kód projektu. Často krát tento ani nie je k dispozícii. Pokiaľ aj je, býva neprehľadný a i keď si ho už programátor osvojí, to znamená rozumie mu, tieto poznatky nemôže využiť pri úprave zdrojových kódov iných projektov, keďže tie používajú odlišné programátorské prostriedky.

Medzi väčšie projekty patrí napríklad bakalárska práca kolegu Šatku [2] alebo diplomová práca kolegu Halouska [3]. Väčšie projekty majú jednotné rozhranie pre algoritmy, ktoré implementujú, ich kód je lepšie čitateľný. Väčšinou však pokrývajú iba určitú skupinu algoritmov, v prípade dvoch uvedených prác sú to grafové algoritmy respektíve kompresné algoritmy.

Osobitné postavenie medzi prácami má projekt Algovision [4] prof. Luděka Kučeru. Pokrýva množstvo algoritmov z rôznych skupín algoritmov.

Cieľom môjho projektu bolo vytvoriť prostredie pre vizualizáciu algoritmov, ktoré bude čo najobecnejšie, algoritmy budú mať jednotné ovládanie. Triedy budú navrhnuté tak, aby bolo relatívne jednoduché vložiť nový algoritmus. Prostredie bude zoskupovať algoritmy z rôznych skupín algoritmov s dôrazom na grafové a triediace algoritmy a algoritmy vyhľadávania v texte.

Súčasťou projektu bude možnosť distančnej výučby. V súčasnej dobe, kedy univerzity začínajú otvárať on-line kurzy, je obohacovanie programov o distančnú výučbu stále viac relevantnejšie. Z toho dôvodu som sa rozhodol vytvoriť prostredie pre vizualizáciu algoritmov obohatené aj o túto funkčnosť.

1.2. Stručný sprievodca po bakalárskej práci

Prvá kapitola popisuje existujúce projekty na túto tému a uvádza dôvodu pre ktoré som sa rozhodol pre túto tému.

Druhá kapitola obsahuje časť analýzy a návrhu. Vysvetľuje pojem vizualizácie algoritmov, jej realizáciu, popisuje požiadavky, ktoré boli na program kladené. Vymenováva všetky algoritmy, ktoré sú v projekte implementované, popisuje ako fungujú a ich zložitosť.

Tretia kapitola nás zoznámi s implementáciou. Vysvetľuje dôvody výberu programovacieho jazyka Java, výber knižníc pre grafiku, technológie pre sieťovú časť programu a technológie pre prenos a ukladanie dátových štruktúr a konfigurácie. Venuje sa metódam testovania. Opisuje spôsob krokovania, fungovanie distančnej výučby a iných funkcií programu z implementačného hľadiska.

Štvrtá kapitola popisuje spôsob používania programu z hľadiska obyčajného používateľa. Zaoberá sa prerekvizitami potrebnými pre beh programu, jeho inštaláciou a behom. Obsahuje stručný manuál popisujúci prácu s programom.

Piata kapitola popisuje spôsob používania programu z hľadiska programátora. Zmieňuje sa o požiadavkách, inštalácii a preklade. Popisuje prácu s dátovými štruktúrami v programe. Opisuje ako vložiť nový algoritmus do programu – a to teoreticky i prakticky v príklade. Pojednáva o možnostiach ako meniť nastavenia a lokalizovať program do iných jazykov.

Šiesta kapitola popisuje možnosti ďalšieho vývoja programu, čo by tam bolo dobré doplniť.

Siedma kapitola obsahuje popis obsahu priloženého CD.

2. Kapitola – Analýza a návrh

2.1. Úvod do problematiky vizualizácie algoritmov

Vizualizácia algoritmov je grafické zobrazenie priebehu algoritmov. Pri tomto zobrazovaní je kladený dôraz na to, aby bol daný spôsob užívateľovi programu zrozumiteľný, teda aby si užívateľ prácu algoritmu rýchlo osvojil.

Algoritmus prevezme vstupné dáta, a po sérii krokov vydá výsledok (obecne sa môže stať, že pri určitom vstupe sa algoritmus zacyklí a teda k žiadnemu konkrétnemu výsledku nedospeje, v takom prípade je výsledkom status „za daných vstupných dát nemôže algoritmus dospieť k výsledku“) . Krok algoritmu je teda prechod od jedného stavu dát algoritmu k ďalšiemu na ceste od vstupných dát k výsledku.

Vizualizácia algoritmu je zachytená sériou diskrétnych stavov – statických obrazových zachytení rozpoložení, v akom sa algoritmus nachádza (na úvodnej obrazovke bude stav pred zahájením algoritmu, na poslednej obrazovke výsledok atď.) a prechodov medzi nimi - kroky. Pre znázornenie týchto krokov sa využíva takých prostriedkov, aby bolo zrejmé, akým spôsobom sa jeden stav algoritmu premenil na iný.

Za danej neexaktnej definície môže byť stavov jedného algoritmu príliš mnoho na to, aby bol užívateľ programu ochotný všetkými obrazovkami prejsť (stovky i tisíce), preto je nutné obmedziť ich počet na prijateľnú úroveň (maximálne 10 – 15). Toho sa dá docieliť buď obmedzením veľkosti štruktúry na ktorú daný algoritmus aplikujeme, alebo tak, že sa pri prechode od jednej obrazovky k ďalšej vykoná viacej krokov naraz, teda na jednu obrazovku bude pripadať niekoľko stavov.

Potrebná pri využívaní programu je možnosť návratu na stav pred posledným krokom, alebo na ľubovoľný iný stav tak, aby užívateľ mohol prechádzať podľa svojej vôle celým algoritmom.

Algoritmus pracuje so vstupnými dátami. Tieto vstupné dáta môže získať niekoľkými spôsobmi. Môže si ich nechať náhodne vygenerovať. Nemôže to byť úplne náhodné generovanie, ale skôr generovanie dát z prípustného rozmedzia. Algoritmus sa pri rôznych vstupných dátach môže chovať rôzne (vezmime ako príklad algoritmus triedenia dát quicksort, ktorý má zložitosť v priemernom prípade n log n, ale v najhoršom prípade n^2). Aby bolo možné vidieť priebeh algoritmu pri hraničných vstupných dátach (napríklad quicksortu v najhoršom prípade), lepšie než čakať, dokiaľ nám budú vhodné dáta náhodne vygenerované je umožniť načítanie vstupných dát zo súboru. S tým súvisí i možnosť ukladania dát do súboru tak, aby ich bolo možné znovu použiť.

Program bude možné rozširovať, teda bude schopný pridať novovytvorený algoritmus medzi už existujúce a používané algoritmy. Vlastnosti spoločné pre všetky algoritmy budú v obecnej časti tak, aby pri vložení nového algoritmu bolo treba doimplementovať iba časti špecifické pre daný algoritmus.

V súčasnosti začína byť v stále väčšej obľube výučba on-line - výučba na diaľku. To znamená, že pokiaľ chce učiteľ demonštrovať vlastnosti niektorého z algoritmov, nemusí zvolávať svojich žiakov k jednému počítaču, na ktorom tieto vlastnosti bude všetkým demonštrovať, ale bude učiteľovi umožnené na počítači spustiť „algoserver“, ku ktorému sa cez internet môžu pripojiť „algoklienti“. Potom všetko, čo učiteľ na svojom počítači vykoná, bude zobrazené i na obrazovkách klientov.

Zabezpečenie prenosu hlasu pre túto virtuálnu prednášku už nie je v kompetencii programu, dá sa toho docieliť pomocou rôznych nástrojov pre on-line hlasovú konferenciu ako napríklad Netmeeting [5] alebo Skype [6].

2.2. Programové požiadavky

Požiadavky na program vyplývajú z predchádzajúceho odseku. Grafická interpretácia musí byť prehľadná. Algoritmus sa dá ovládať krok za krokom. Počet krokov algoritmu spracovávajúceho štruktúru bežnej veľkosti nepresiahne prijateľný počet (napríklad 20). Program má byť rozšíriteľný a má umožňovať distančnú výučbu.

2.3. Popis implementovaných grafových algoritmov

2.3.1. Bellman-Fordov algoritmus

Algoritmus rieši problém hľadania najkratšej cesty v ohodnotenom grafe s hranami, ktoré môžu mať i záporné ohodnotenie bez negatívnych cyklov.

Ako algoritmus funguje:

Algoritmus funguje vo vrstvách, kedy v každej vrstve behu algoritmu skúmame hrany vychádzajúce z jedného vrcholu grafu.

Krok 1: Dĺžka cesty od počiatočného vrcholu (ďalej len dĺžka) sa pre všetky vrcholy grafu nastaví na nekonečno.

Krok 2: Dĺžka sa pre počiatočný vrchol nastaví na 0.

Krok 3: Počiatočný vrchol sa vloží do fronty.

Krok 4. Vyberie sa z fronty vrchol. Pokiaľ je počet ohrád vo fronte väčší ako počet vrcholov, algoritmus skončil neúspešne. Do fronty sa vloží ohrada. Pre každého suseda sa spočíta, či nie je dĺžka cesty od počiatočného vrcholu cez vrchol kratšia, pokiaľ áno, aktualizuje sa dĺžka a vrchol sa vloží do fronty.

Zložitosť: O(V.E), kde V je počet vrcholov, E je počet hrán

2.3.2. Dijkstrov algoritmus

Algoritmus rieši problém hľadania najkratšej cesty v ohodnotenom grafe s hranami, ktoré majú iba kladné ohodnotenie.

Ako algoritmus funguje:

Budeme si vypomáhať prioritnou frontou zoradenou podľa dĺžky cesty od počiatočného vrcholu.

Krok 1: Dĺžka cesty od počiatočného vrcholu (ďalej len dĺžka) sa pre všetky vrcholy grafu nastaví na nekonečno.

Krok 2: Dĺžka sa pre počiatočný vrchol nastaví na 0.

Krok 3: Vrcholy sa vložia do prioritnej fronty.

Krok 3: Vyberieme vrchol s najmenšou dĺžkou. Dĺžku vyhlásime za definitívnu a pre všetkých susedov zistíme, či dĺžka cesty cez tento vrchol nie je kratšia ako dĺžka ktorú máme zatiaľ. Ak áno, zmeníme ju na kratšiu.

Krok 4: Dokiaľ nie je fronta prázdna opakujeme krok 3.

Zložitosť: $O(V^2)$, kde V je počet vrcholov pri implementácii bez prioritnej fronty, O(E lg V) pri implementácii s prioritnou frontou, kde prioritná fronta je reprezentovaná haldou.

2.3.3. BFS algoritmus – prehľadávanie grafu do šírky

Algoritmus na prehľadávanie grafu.

Ako algoritmus funguje:

Krok 1: Vložiť počiatočný vrchol do fronty.

Krok 2: Vybrať vrchol z fronty, označiť ho za preskúmaný a vložiť do fronty susedov vrcholu.

Krok 3: Opakovať krok 2 dokiaľ nie je fronta prázdna.

Zložitosť: a) O(V+E), kde V je počet vrcholov, E je počet hrán pri reprezentácii grafu zoznamom následníkov

b) $O(V^2)$, kde V je počet vrcholov, E je počet hrán pri reprezentácii grafu maticou susednosti

2.3.4. DFS algoritmus – prehľadávanie grafu do hĺbky

Algoritmus prehľadávania grafu.

Ako algoritmus funguje:

Krok 1: Vložiť počiatočný vrchol do zásobníku.

Krok 2: Vybrať vrchol z fronty, označiť ho za preskúmaný a vložiť do fronty susedov vrcholu.

Krok 3: Opakovať krok 2 dokiaľ nie je zásobník prázdny.

Zložitosť: a) O(V+E), kde V je počet vrcholov, E je počet hrán pri reprezentácii grafu zoznamom nasledovníkov

b) $O(V^2)$, kde V je počet vrcholov, E je počet hrán pri reprezentácii grafu maticou susednosti

2.3.5. Jarníkov algoritmus

Algoritmus na hľadanie minimálnej kostry grafu.

Jak algoritmus funguje:

Krok 1: Vložíme ľubovoľný vrchol do minimálnej kostry grafu.

Krok 2: Vyberieme hranu grafu (u,v) s minimálnou dĺžkou takú, že u je už v minimálnej kostre grafu, v ešte nie.

Krok 3: Pridáme vrchol v a hranu (u,v) do minimálnej kostry grafu.

Krok 4: Opakujeme krok 2 dokiaľ nie sú všetky vrcholy v minimálnej kostre grafu.

Zložitosť: $O(V^2)$, kde V je počet vrcholov, E je počet hrán pri reprezentácii grafu maticou susednosti

2.4. Popis implementovaných triediacich algoritmov

2.4.1. Algoritmus bublinkového triedenia

Triediaci algoritmus.

Ako algoritmus funguje:

Porovnávame vždy dve susedné hodnoty. V prípade, že nie sú v správnom poradí, vymeníme ich. Pri každom priechode sa jedna hodnota dostane na správne miesto. Po n priechodoch sú všetky hodnoty na správnych miestach.

Zložitosť: V najhoršom prípade má algoritmus bublinkového triedenia zložitosť $O(n^2)$, kde n je veľkosť zoznamu čísel na triedenie. V najlepšom prípade, kedy je už zoznam čísel zotriedený, má algoritmus zložitosť O(n).

2.4.2. Triedenie zlievaním

Triediaci algoritmus.

Ako algoritmus funguje:

Rozdelíme postupnosť na dve polky, obe polky zotriedime a zotriedené polky zlúčime.

Zložitosť: V priemernom i najhoršom prípade má algoritmus zložitosť O(n log n), kde n je veľkosť zoznamu čísel na triedenie.

2.4.3. Quicksort

Triediaci algoritmus.

Ako algoritmus funguje:
Krok 1: Vyberieme pivota (napríklad posledný prvok v postupnosti) a zoradíme prvky podľa tohto prvku, naľavo budú prvky menšie, napravo prvky väčšie.
Krok 2: Opakujeme krok 1 s postupnosťou prvkov naľavo od pivota.
Krok 3: Opakujeme krok 1 s postupnosťou prvkov napravo od pivota.

Zložitosť: V priemernom prípade má algoritmus zložitosť O(n log n), v najhoršom prípade $O(n^2)$, kde n je veľkosť zoznamu čísel na triedenie.

2.5. Popis implementovaných algoritmov vyhľadávania reťazca v texte

2.5.1. Rabin-Karpov algoritmus

Algoritmus vyhľadávania v texte.

Ako algoritmus funguje:

Prechádzam od začiatku text v ktorom hľadám. Skúmam vždy časť textu, ktorá má rovnakú dĺžku ako je dĺžka hľadaného textu. Vypočítam si hash skúmanej časti textu a porovnám s hashom hľadaného textu – pokiaľ sa zhodujú, pozriem sa, či sa texty zhodujú, ak áno, našiel som text, pokiaľ nie, zatiaľ som nenašiel. Pokračujem, skúmam ďalšie výskyty.

Zložitosť: V priemernom prípade má algoritmus zložitosť O(n), v najhoršom prípade O(nm), kde n je dĺžka textu v ktorom hľadám a m je dĺžka hľadaného textu.

2.5.2. Aho-Corasick

Algoritmus vyhľadávania v texte.

Ako algoritmus funguje:
S použitím reťazcov, ktoré chcem v texte vyhľadať, vyrobím automat - vyhľadávací stroj. Pomocou tohto automatu potom prehľadávam text a oznamujem, či som našiel alebo nenašiel hľadané reťazce.

Zložitosť: Výroba automatu trvá O (d*AB), kde d je súčet dĺžok všetkých vyhľadávaných slov a AB je počet písmen abecedy, automat prehľadá text za O(n), kde n je dĺžka textu, teda celková zložitosť algoritmu je O(n + d*AB).

2.6. Zdrojový kód k algoritmom

Zdrojový kód k algoritmom som získaval buď vo forme pseudokódu, ktorý som si následne prepísal do kódu v jazyku Java alebo pokiaľ to bolo možné, hľadal som priamo neproprietárny kód v jazyku Java a ten som si prispôsoboval tak aby pracoval s mojimi dátovými štruktúrami. Užitočný pseudokód a zdrojový kód k rôznym algoritmom a dátovým štruktúrami možno nájsť v [7], [8] a [9].

3. Kapitola - Implementácia

3.1. Úvod do implementácie

3.1.1. Výber platformy a programovacieho jazyka

Medzi hlavné požiadavky pri analýze projektu patrila požiadavka na spustitelnosť programu na rôznych operačných systémoch bez nutnosti prispôsobovania kódu danému operačnému systému. Ďalšou požiadavkou bola okrem možnosti spustiť program lokálne možnosť komunikovať po sieti s inštanciou programu spustenou na inom počítači. Z týchto dôvodov bol vybraný programovací jazyk Java, nakoľko program napísaný v jazyku Java je spustiteľný pod akýmkoľvek operačným systémom na ktorom beží Java Virtual Machine. Jazyk Java zároveň obsahuje prostriedky pre prácu po sieti.

3.1.2. Grafika

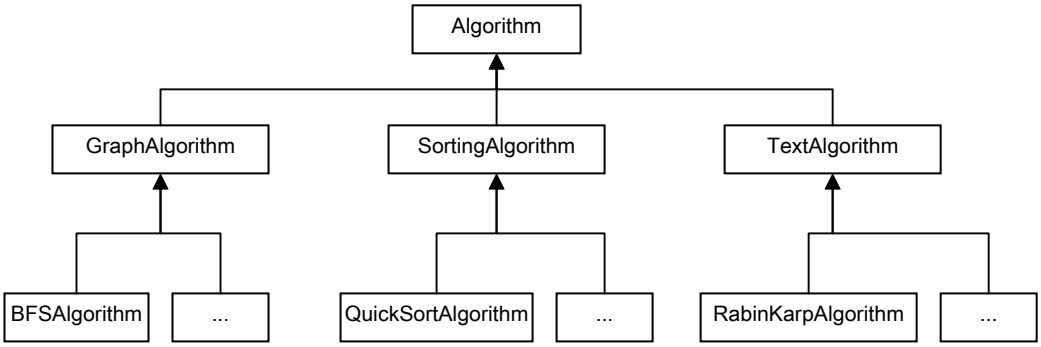
Pre účely grafického zobrazenia programu sú použité JFC (Java Foundation Classes) Swing a Java 2D API. JFC Swing je súbor nástrojov grafického užívateľského rozhrania (GUI). Obsahuje podporu pre rôzne druhy tlačidiel, textových polí, menu, panelov a iné. Java 2D je aplikačné programové rozhranie (API) pre kreslenie 2-D grafiky pomocou programovacieho jazyka Java. Podporuje vykresľovanie rôznych tvarov, prácu s farbami, druhmi písma a iné.

Viac sa o JFC Swing dočítate v [10], o Java 2D API v [11].

3.2. Implementácia hlavných častí projektu

3.2.1. Hierarchia tried reprezentujúcich algoritmy

Prarodičom všetkých tried reprezentujúcich algoritmus je abstraktná trieda Algorithm. Túto triedu rozširujú triedy, ktoré reprezentujú skupiny algoritmov (GraphAlgorithm, SortingAlgorithm, TextAlgorithm). Tie sú opäť abstraktné. Uvedená hierarchia je graficky vyobrazená na obrázku 1.



Obrázok 1: Hierarchia dedičnosti tried reprezentujúcich algoritmus.

3.2.2. Triedy reprezentujúce skupiny algoritmov

Každá skupina algoritmov má v projekte triedu, ktorá ju reprezentuje. Programátor nemusí pri implementácii konkrétneho algoritmu dediť od triedy, reprezentujúcej skupinu algoritmov do ktorej algoritmus patrí, môže dediť priamo od Algorithm, ale podedením od triedy reprezentujúcej skupinový algoritmus (ďalej len trieda) si uľahčí implementáciu nového algoritmu.

V triede sa nachádza štruktúra zoznam, do ktorého sa vždy pri behu algoritmu generujú podoby dátovej štruktúry reprezentujúcej danú skupinu algoritmov.

Trieda obsahuje metódy, ktoré konvertujú dátovú štruktúru z a do podoby XML. Túto štruktúru vykresľuje. Pre účely zachytenia stavu štruktúry má definovanú metódu capture(), ktorá si vytvorí kópiu súčasného stavu štruktúry. Ďalej sa v triede nachádzajú metódy pre naplnenie štruktúry defaultnými alebo náhodnými dátami.

Jednou z najdôležitejších metód v triede je metóda, ktorá má na starosti inicializáciu algoritmu. Resetuje číslo aktuálneho kroku na nulu, zachytí podobu štruktúry ešte pred tým, ako sa spustil algoritmus, spustí algoritmus, zachytí štruktúru tesne po tom ako sa ukončil algoritmus. Nakoniec zavolá čistiacu metódu, ktorá odstráni duplicity v krokoch.

Totižto pri generovaní algoritmu sa stáva, že sú niektoré kroky algoritmu graficky neviditeľné, to znamená, že algoritmus urobil krok, ale štruktúru to graficky nezmenilo. Preto po vykonaní algoritmu môžu byť niektoré kroky graficky identické, čo má za následok pocit u užívateľa, že sa nič nedeje. Aby sa tomu predišlo, duplicity sú zmazané a tak každý krok algoritmu spôsobuje grafickú zmenu.

3.2.3. Sieťová časť programu - distančná výučba

3.2.3.1. Úvod do sieťovej časti

Po spustení funguje program defaultne v roli samouk. To znamená, že užívateľ má možnosť spustiť si akýkoľvek algoritmus z výberu a pracovať s ním bez interakcie ďalšieho užívateľa - počítač odpovedá na podnety užívateľa.

Ďalšia z rolí, do ktorej môže byť program uvedený, je rola server. Z hľadiska užívateľa pracujúceho s programom v tejto roli nie sú oproti roli samouk žiadne zmeny (ak nerátame drobné vizuálne zmeny sieťového ovládacieho panelu a statusu programu). Na program v roli server sa pomocou siete môže pripojiť iná inštancia programu, tá musí byť v roli klient (rola klient je popísaná v nasledujúcom odseku). Potom grafické udalosti, ktoré sa budú diať na serveri, budú vykonávané i na klientovi. Na server je možné pripojiť viac ako jedného klienta.

V roli klient má program zablokované ovládanie (okrem možnosti pre ukončenie role klienta), užívateľ môže iba pasívne prihliadať dianiu na obrazovke.

Role server a klient a ich vzájomná komunikácia, teda sieťová časť programu je implementovaná pomocou technológie RMI. Beh RMI má síce nenulovú réžiu, ale kód napísaný s využitím tejto technológie je čitateľnejší ako kód s využitím socketov.

Taktiež je jednoduchšie a rýchlejšie napísať program v RMI a rýchlosť, jednoduchosť a prehľadnosť boli hlavnými kritériami, ktoré predurčili RMI pre použitie v mojom prípade. Viac o RMI v [12].

3.2.3.2. Komunikácia medzi klientom a serverom

Komunikácia medzi klientom a serverom prebieha cez vzdialené rozhrania AlgoClientBase a AlgoServerBase. Tieto rozhrania majú nasledujúci zdrojový kód:

```
public interface AlgoServerBase extends Remote {
	/** Pridá klienta do zoznamu klientov, ktoré server obsluhuje. */
	public void addClient(AlgoClientBase client) throws RemoteException;
	/** Odstráni klienta zo zoznamu klientov, ktoré server obsluhuje. */
	public void removeClient(AlgoClientBase client) throws RemoteException;
	/** Vráti referencii na spojový zoznam s históriou. Dôležité pre
synchronizáciu */
	public LinkedList getHistory() throws RemoteException;
	/** Zistí číslo verzie na severi. */
	public String getVersionNo() throws RemoteException;
}

public interface AlgoClientBase extends Remote {
	/** Zpracovává serverem zaslaný objekt. */
	public void processAction(Object e) throws RemoteException;
}
```

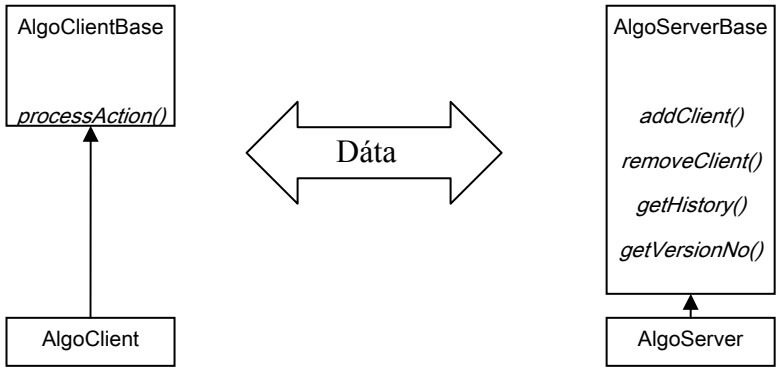
Tieto rozhrania sú implementované triedami AlgoClient a AlgoServer.

Na počítači, ktorý bude mať rolu server, spustíme server (spôsob, akým to spraviť, je popísaný v užívateľskej príručke, kapitola 4.4.4). Vytvorí sa nám tým inštancia triedy AlgoServer. Táto inštancia bude zachytávať udalosti, ktoré sa budú na serveri diať a preposielať ich všetkým registrovaným klientom. Aby to bolo možné vykonať, musia sa klienti nejakým spôsobom registrovať, to znamená server musí mať referenciu na ich vzdialené rozhranie AlgoClientBase.

Našej inštancii AlgoServer-a, ktorá implementuje vzdialené rozhranie AlgoServerBase, priradíme meno (ďalej len meno) v RMI registry. Klient pozná adresu počítača, na ktorom beží server a podľa tejto adresy a zároveň podľa mena si vyžiada od RMI registry referenciu na vzdialené rozhranie serveru teda referenciu na AlgoServerBase. Pokiaľ túto referenciu úspešne získa, môže sa pomocou metódy addClient u serveru zaregistrovať. Registráciou rozumieme, že klient zašle serveru referenciu na svoje vzdialené rozhranie. Po registrácii už môže server klientovi odosielať dáta.

Dôležité je poznamenať, že klient si pri pripojovaní na server najprv zistí, či majú programy na serveri a klientovi rovnaké číslo verzie a iba v prípade zhody komunikácia pokračuje. Inak sa program ukončí. Kontrolovaním čísel verzii sa zabezpečí kompatibilita tak aby nevznikali chyby.

Pozornému čitateľovi neujde, že AlgoServerBase má ešte jednu vzdialenú metódu, ktorú sme nespomenuli a to getHistory(). Totižto keď sa klient pripojí k serveru, server už môže byť nejaký čas zapnutý a vykonať už niekoľko akcií, o ktorých však klient nemôže vedieť, pretože v čase ich vykonania nebol pripojený k serveru. Preto sa klient po pripojení musí najprv zosynchronizovať, to znamená dostať do stavu, v akom je server. Práve pre účely tejto synchronizácie sa využíva getHistory(). Obrázok 2 graficky zobrazuje hierarchiu tried implementujúcich sieťovú časť programu.



Obrázok 2: Hierarchia tried (respektíve rozhraní) implementujúcich sieťovú časť programu.

3.2.3.3. Druhy dát zasielané medzi klientom a serverom

Server posiela klientovi niekoľko druhov dát. Zaslaním inštancie triedy WndOper server oznamuje klientovi, že na serveru prebehla operácia s oknom algoritmu, otvorenie nového okna s algoritmom, respektíve zatvorenie okna s algoritmom.

Zaslaním inštancie triedy ActionEvent server oznamuje klientovi, že server krokoval algoritmus pomocou jednej zo štyroch krokovacích metód popísaných v kapitole 4.4.3.

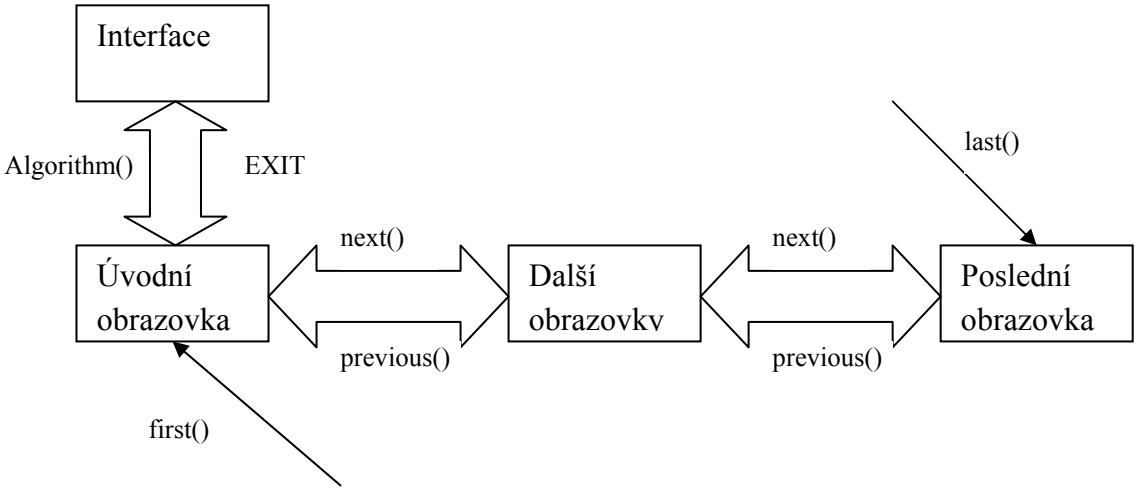
Zaslaním inštancie triedy String server oznamuje klientovi, že mu posiela reťazec s XML podobou dátovej štruktúry reprezentujúcej algoritmus, touto dátovou štruktúrou si má nahradiť súčasnú dátovú štruktúru.

3.2.4. Krokovanie algoritmu

Po zavolaní konštruktoru na konkrétny algoritmus sa spustí inicializácia štruktúr potrebných pre vykonanie algoritmu a následne sa algoritmus vykoná. Počas vykonávania algoritmu dátovú štruktúru okrem kódu algoritmu menia i tzv. vizualizačné vsuvky (napríklad takým spôsobom, že sa nám vyznačí určitý vrchol grafu inou farbou ako ostatné). Takto zmenenú štruktúru je možné „zachytiť“ a tým vytvoriť ďalší krok algoritmu. Krokovanie teda spočíva v tom, že skočíme na ďalšiu zaznamenanú podobu štruktúry algoritmu. Nevyhnutnosť takéhoto prístupu pri

implementácii krokovania (kedy zachytávame podobu celej štruktúry v každom kroku) spočíva v tom, že niektoré algoritmy sú nedeterministické. U takéhoto spôsobu by nebolo jasné, ako vykonať krok späť. A práve tento problém rieši vyššie spomenuté zachytávanie podoby štruktúry v jednotlivých krokoch.

Pre účely krokovania slúži hlavný ovládací panel, ktorý je podrobne popísaný v kapitole 4.4.2. Schematické vyobrazenie krokovania algoritmu je na obrázku 3.



Obrázok 3: Krokovanie algoritmu.

3.3. Implementácia ostatných častí programu

3.3.1. Automatické zisťovanie názvov algoritmov a názvov skupín algoritmov

Každá trieda predstavujúca skupinu algoritmov má definovaný statický reťazec algorithmGroupName, ktorý popisuje názov skupiny algoritmu. Podobne každý konkrétny algoritmus má definovaný statický reťazec algorithmName, ktorý určuje názov algoritmu.

Konkrétny algoritmus obsahuje vnútornú triedu AlgInfo implementujúcu rozhranie AlgInfoBase

```
protected interface AlgInfoBase{
        public String getAlgName();
        public String getAlgGroupName();
}
```

Pretože každá vnútorná trieda je pri preklade pomenovaná nasledovným spôsobom – NázovTriedy$NázovVnútornejTriedy, čo v našom prípade bude predstavovať NázovTriedyAlgoritmu$AlgInfo, znamená to, že pokiaľ si z adresára načítame len triedy, ktoré sa končí na „AlgInfo“, vytvoríme si ich inštancie podľa mena, získame tak informácie o všetkých implementovaných algoritmoch. Výhodou takéhoto prístupu je skutočnosť, že sa nám menu so zoznamom algoritmov vytvorí

dynamicky, nie je teda nutné, aby sme ho menili vždy pri pridaní, respektíve odobraní algoritmu.

Dôvod, prečo je nutné vkladať do každej konkrétnej triedy rovnaký kód pre implementáciu rozhrania AlgInfoBase, ktoré sa nachádza v triede Algorithm a nie je možné použiť dedičnosť, je ten, že v prípade dedičnosti by nám prekladač nevytvoril súbory s bytekódom vnútornej triedy AlgInfo vo vyššie spomenutom tvare NázovTriedyAlgoritmu$AlgInfo, čo je, ako bolo v predošlom odseku vysvetlené, veľmi dôležité pri automatickom zisťovaní názvov algoritmov.

3.3.2. Prenos, ukladanie a načítanie dátových štruktúr a konfigurácie

Dáta sa ukladajú a prenášajú v tvare podľa technológie XML (informácie o XML možno nájsť v [13]). Táto reprezentácia je dobre čitateľná, je výhodná i na prenos dát po sieti, ľahko sa s ňou manipuluje.

Algoritmus má definovanú metódu, ktorá má na starosti previesť reprezentáciu algoritmu do XML podoby takým spôsobom, aby sa dáta z tejto podoby dali znovu bez straty zrekonštruovať. Pri ukladaní vstupu do súboru sa teda zavolá táto metóda a takto konvertovaná reprezentácia algoritmu do podoby reťazca je uložená do súboru.

Každá úložná štruktúra má definovanú metódu, ktorá ju načíta z XML podoby. Pri načítaní vstupu sú na vstup spustené dve úrovne parsovania. Prvá úroveň získa z XML reprezentácie obecné informácie o algoritme (počet krokov), druhá úroveň, ktorá je závislá od skupiny algoritmov, načíta dáta do dátovej štruktúry.

3.3.3. Slideshow

Po naštartovaní SlideShow je v novom vlákne spustená trieda SlideShow (potomok Thread), ktorá v pravidelných intervaloch emituje ActionEvents, ktoré spôsobujú rovnaké chovanie, ako keby sme na hlavnom ovládacom paneli stlačili tlačidlo pre krok dopredu. Rýchlosť týchto zmien je nastaviteľná, udáva sa ako parameter konštruktoru triedy Slideshow. Nutnosť spustiť slideshow v novom vlákne vyplýva zo skutočnosti, že v opačnom prípade by nám program vykreslil po dlhom čakaní až konečný stav algoritmu.

3.4. Testovanie

Testovanie funkčnosti programu bolo vykonávané niekoľkými spôsobmi. Jeden zo spôsobov obsahoval skúšku pridania nového algoritmu do projektu. Testovala sa priamočiarosť pridania.

Medzi ďalšie spôsoby testovania patrilo používanie programu - vyskúšanie funkčnosti všetkých možností programu. Program bol uvedený do rôznych kritických situácii, napríklad pri diaľkovom spojení medzi klientom a serverom bol nečakane prerušený server, potom nečakane prerušený klient, nastala úmyselná chyba v sieti

(odpojenie sieťového kábla). Reakcie programu na tieto podnety boli skúmané a následne vhodne upravované a znovu testované.

Ďalšia metóda testovania obsahuje vytvorenie veľkej štruktúry pre každú skupinu algoritmov. Štruktúru nazveme veľkou, pokiaľ počet jej jednotlivých súčastí (vrcholov v prípade grafu, čísel v prípade triediacich algoritmov a písmen v prípade textových algoritmov) rádovo presiahne počet súčastí štruktúry, ktorá je použitá v programe, tým sa myslí štruktúra, ktorá je defaultne vygenerovaná alebo štruktúra, ktorá je vygenerovaná po zadaní príkazu na náhodný vstup. Pre potreby vytvorenia veľkých štruktúr bola vytvorená trieda Generator, ktorá generuje veľkú štruktúru do súboru.

Náhodnú veľkú štruktúru vytvoríme nasledujúcim postupom. Presunieme sa do adresára, kde máme nainštalované súbory (inštalácia je popísaná v kapitole 4.2. Zadáme nasledujúci príkaz:

```
java vizualizace.Generator TYPE FILENAME
```

TYPE je typ algoritmu (hodnota môže byť `graph`, `sort`, alebo `text`) a FILENAME je názov súboru, do ktorého chceme vygenerovaný vstup uložiť. Takto vygenerovaný vstup môžem načítať do projektu a zistiť ako sa program správa.

Všetky testy boli úspešné až na test načítania veľkej štruktúry u triediaceho algoritmu bublinkového triedenia, kde nastal problém s nedostatkom pamäti kvôli príliš veľkému počtu krokov potrebných na zotriedenie postupnosti. Tento problém súvisí s tým, že triedenie pomocou bublinkového algoritmu je oproti iným triediacim algoritmom neefektívne (pokiaľ za kritérium efektivity považujeme zložitosť algoritmu).

4. Kapitola – Užívateľská dokumentácia

4.1. Systémové požiadavky

Pre beh program je nevyhnutná inštalácia Java JRE SE [14] verzie minimálne 1.5. Pokiaľ chce užívateľ využívať program v pozícii serveru, musí mať nainštalované RMI registry (to je súčasťou Java JRE SE). RMI beží defaultne na porte 1099, teda užívateľ musí zabezpečiť, aby na tomto porte nebežala žiadna iná služba (aby sme mohli spustiť server na tomto porte) a taktiež musíme nastaviť firewall tak, aby umožňoval komunikáciu cez tento port kvôli komunikácii medzi klientom a serverom.

Program nie je spätne kompatibilný s nižšími verziami JRE, pretože využíva niektoré vlastnosti jazyka Java pridané až vo verzii 1.5 (generické typy atď.).

4.2. Inštalácia

Program je distribuovaný v zip formáte v súbore vizualizace.zip, ktorý sa nachádza v adresári /bin na CD. Inštaláciou rozumieme rozbalenie súborov zo zip súboru do ľubovoľného adresára, napríklad /temp.

4.3. Spustenie programu

Prejdeme do adresára, kde sa nachádzajú rozbalené súbory (v predošlom prípade /temp). Program spustíme príkazom

```
java vizualizace.Interface
```

Program je taktiež možné spustiť spustením batch súboru start.bat, ktorý sa nachádza v adresári programu.

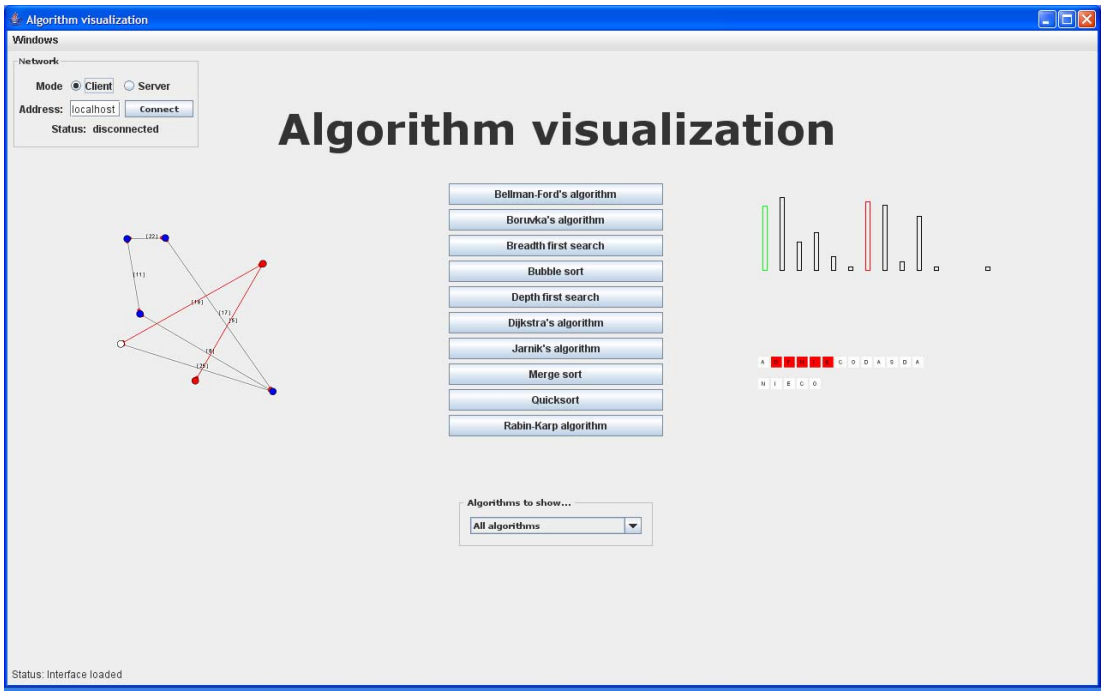
4.4. Stručný sprievodca programom

4.4.1. Možnosti behu programu

Možnosti behu programu sú popísané v kapitole 3.2.3.1.

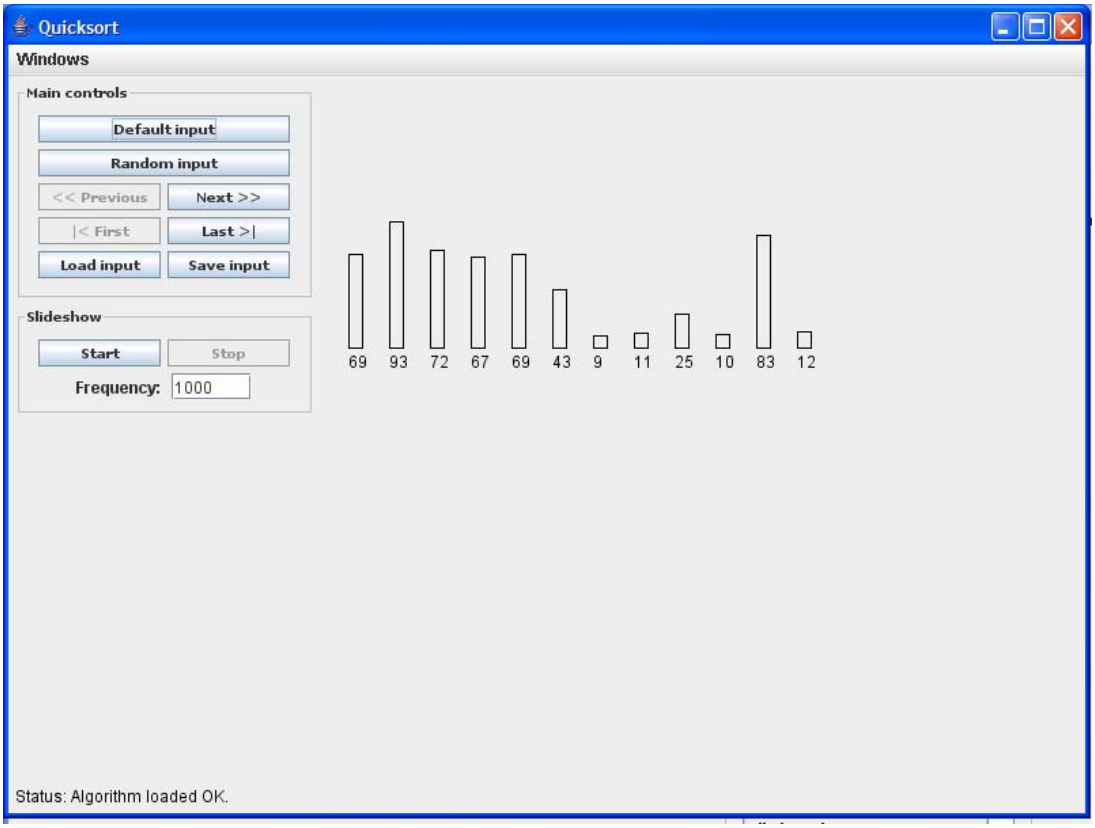
4.4.2. Úvodná obrazovka

Úvodnú obrazovku programu tvorí hlavné menu so zoznamom implementovaných algoritmov, Combobox, ktorý triedi zobrazované algoritmy po jednotlivých skupinách a ovládací panel pre sieť. Screenshot úvodnej obrazovky môžeme vidieť na obrázku 4.



Obrázok 4: Screenshot úvodnej obrazovky programu.

Po kliknutí na tlačidlo s názvom algoritmu je zobrazené nové okno do ktorého sa nám vybraný algoritmus načíta. Screenshot úvodnej obrazovky jedného z implementovaných algoritmov je na obrázku 5.



Obrázok 5: Screenshot úvodnej obrazovky algoritmu quicksort.

4.4.3. Ovládací panel Main controls

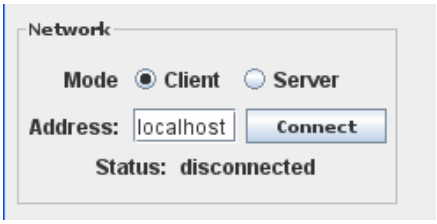
Ovládací panel Main controls (obrázok 6) slúži na krokovanie algoritmu, generovanie náhodného vstupu, návrat k defaultnému vstupu, načítanie a ukladanie vstupu.

Obrázok 6: Hlavný ovládací panel

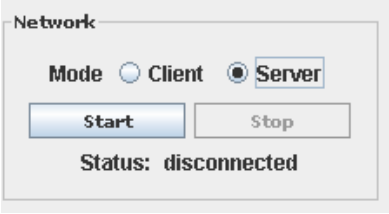
Popis významu jednotlivých tlačidiel na hlavnom ovládacou paneli:
First = Skok na prvý krok vizualizácie algoritmu
Last = Skok na posledný krok vizualizácie algoritmu
Next = Skok na ďalší krok vizualizácie algoritmu
Previous = Skok na predchádzajúci krok vizualizácie algoritmu
Default input = Načítanie defaultného inputu
Random input = Vygenerovanie náhodného inputu
Load input = Načítanie inputu zo súboru
Save input = Uloženie inputu do súboru

4.4.4. Ovládací panel Network

Ovládací panel pre sieť (obrázky 7 a 8) slúži na uvedenie programu do role server alebo klient, popísaných v kapitole 3.2.3.1.



Obrázok 7: Ovládací panel pre sieť – časť pre pripojenie/odpojenie klienta



Obrázok 8: Ovládací panel pre sieť – časť pre spustenie/zastavenie serveru

Pomocou rádio tlačidla si vyberieme mód, v ktorom sa bude naša inštancia programu nachádzať – buď server alebo klient, oba naraz nie je možné spustiť. Podľa tohto výberu sa zmení i výzor ovládacieho panelu Network.

Mód klient:
Address form = Formulár pre adresu serveru
Connect/Disconnect = Pripojenie na server na adrese z Address formu
Status = Udáva status pripojenia klienta (pripojený/odpojený)

Mód server:
Start = spustenie serveru
Stop = zastavenie serveru
Status = Udáva status pripojenia serveru (spustený/zastavený)

4.4.5. Ovládací panel Slideshow

Pokiaľ si chceme prehliadnuť beh algoritmu bez nutnosti preklikávať sa do konca pomocou Next, môžeme využiť služieb ovládacieho panelu Slideshow (obrázok 9).

Obrázok 9: Ovládací panel Slideshow

Popis ovládacieho panela Slideshow:
Start = spustí slideshow
Stop = zastaví slideshow
Period = Perióda prekresľovania krokov algoritmu v milisekundách

5. Kapitola – Programátorská dokumentácia

5.1. Systémové požiadavky

Pre účely kompilácie projektu je nutné aby bola v systéme nainštalovaná Java JDK SE [14] verzie minimálne 1.5 a Apache Ant [15].

5.2. Inštalácia

Pokiaľ nemáme Ant nainštalovaný, nainštalujeme ho. Podrobný postup inštalácie môžeme nájsť v [16].

Rozbalíme zdrojové súbory z jar súboru do ľubovoľného adresára, napríklad /temp. Predpokladáme, že cesta k programu Ant patrí do systémovej premennej PATH, pokiaľ nie, pridáme ju tam. Prejdeme do adresára /temp a spustíme príkaz:

```
ant -f build.xml compile,rmi,javadoc
```

5.3. Vizualizácia dátových štruktúr – trieda Visualizer

Pri vytváraní krokov algoritmu sa stretneme s potrebou graficky zvýrazniť ako daný algoritmus funguje. Príkladom môže byť zmena farby vrcholu ktorý preskúmavame, nebo farebné označenie dvoch hodnôt v postupnosti, ktoré algoritmus bublinkového triedenia vymení. Pre tieto účely sa využíva statických metód v triede Visualizer. Týmto metódam je daná referencia na objekt, ktorý majú graficky upraviť (typicky vstup algoritmu) a ďalšie parametre potrebné pre vykonanie zmeny. Detaily používania jednotlivých metód triedy Visualizer sa nachádzajú v Javadoc-u na priloženom CD.

5.4. Vloženie nového algoritmu – všeobecné informácie

Nasledujúci postup popisuje spôsob, akým do projektu vložíme nový algoritmus.

Pokiaľ chceme do projektu vložiť algoritmus zo skupiny, ktorá sa tam nenachádza (teda nie je grafový, triediaci alebo textový), budeme si programovaciu logiku tejto novej skupiny algoritmov musieť naprogramovať sami. My sa teraz budeme zaoberať druhou možnosťou, a to keď algoritmus bude buď algoritmom grafovým, triediacim alebo algoritmom vyhľadávania v texte.

Nová trieda, ktorú pre algoritmus vytvoríme, bude dediť od abstraktnej triedy reprezentujúcej skupinu, do ktorej algoritmus patrí.

Nadefinujeme do triedy statický reťazec algorithmName, ktorý bude obsahovať názov algoritmu. Triedu algoritmu obohatíme o vnútornú triedu AlgInfo presne v nasledujúcej podobe.

```
public static class AlgInfo implements AlgInfoBase {
        public String getAlgName() {
                return algorithmName;
        }

        public String getAlgGroupName() {
                return algorithmGroupName;
        }
}
```

Dôvody, prečo programátor vždy musí vkladať tento kód a nie je možné vyriešiť tento problém dedičnosťou vysvetľuje kapitola 3.3.1.

Keďže naša novovzniknutá trieda je prvá konkrétna trieda v „rodokmeni“, to znamená rodičovská trieda a zároveň prarodičovská trieda Algorithm sú abstraktné, musí naša neabstraktná trieda implementovať všetky zdedené abstraktné metódy. V našom prípade je to metóda algorithm(), do ktorej sa zapíše samotný algoritmus. Kód v metóde algorithm bude používať štruktúru určenú pre danú skupinu algoritmov. Na vhodných miestach (podľa uváženia programátora) budú v tejto metóde použité nástroje triedy Vizualizer.

5.5. Špecifiká vloženia nového grafového algoritmu

Ústrednou štruktúrou grafových algoritmov je graf. V projekte ho reprezentuje trieda GraphInput. Konštruktor určí koľko vrcholov bude mať graf a či to bude orientovaný nebo neorientovaný graf. Do grafu je možno pridávať vrcholy a hrany. Vrchol grafu je indexovaný poradím jeho pridania do grafu. Hrana je indexovaná indexmi dvoch vrcholov ktoré spája. Užívateľ má k dispozícii funkciu, ktorá umožňuje pre každý vrchol prechádzať jeho susedmi. Graf je možné načítať zo súboru a z reťazca vo formátu XML.

5.6. Špecifiká vloženia nového triediaceho algoritmu

Analogickou štruktúrou ku grafu pre triediace algoritmy je SortInput. Obsahuje pole triedených prvkov a pivota. Umožňuje získať a uchovávať informácie o triedených prvkoch potrebné pre beh algoritmu. Obsahuje možnosť načítania a uloženia svojej podoby do XML.

5.7. Špecifiká vloženia nového algoritmu pre vyhľadávanie v texte

Textové algoritmu nie sú navzájom také podobné ako algoritmy triediace alebo grafové. Textový algoritmus pracuje so štruktúrou reprezentovanou triedou TextInput. Tá však obsahuje iba dátové štruktúry potrebné pre vizualizáciu algoritmu Rabin-Karp a preto je potrebné doplniť TextInput pri implementácii nového textového algoritmu o nevyhnutné štruktúry, napríklad o tabuľku v prípade implementácie algoritmu Knutt-Morris-Pratt, respektíve vytvoriť novú štruktúru (ako som to urobil pri implementácii algoritmu Aho-Corasick).

5.8. Vloženie nového algoritmu – príklad

Popísali sme obecný postup, ako do projektu vložiť nový algoritmus. Teraz si to vyskúšame na príklade. Vložíme do projektu algoritmus triedenia vkladaním (insertion-sort).

Do balíka vizualizace.extensions vložíme novú triedu, nazveme ju napríklad InsertionSortAlgorithm. Podedíme tuto triedu od triedy reprezentujúcej triediace algoritmy (trieda SortingAlgorithm). Nadefinujeme vhodne algorithmName, napríklad na hodnotu „Insertion sort“. Vložíme kód triedy AlgInfo. Nájdeme zdrojový kód pre insertion sort, napríklad na Wikipedii [17]. Našli sme niečo ako...

```
insert(array a, int length, value) {
     int i = length - 1;
     while (i >= 0 && a[i] > value) {
         a[i + 1] = a[i];
         i = i - 1;
     }
     a[i + 1] = value;
 }

 insertionSort(array a, int length) {
     for (int i = 0; i < length; i = i + 1) insert(a, i, a[i]);
 }
```

Prepíšeme tento kód tak, aby pracoval s dátovou štruktúrou z nášho projektu a vložíme ho do metódy algorithm():

```
private void insert(int length, ListItem value){
        int i = length - 1;

        while (i >= 0 && input.values[i].value > value.value) {
          input.values[i+1] = input.values[i];
          i = i - 1;
     }
     input.values[i + 1] = value;
}

public void algorithm(){
        int length = input.length();

        for (int i = 0; i < length; i = i + 1)
                        insert(i, input.values[i]);
}
```

Pokiaľ by sme teraz spustili náš projekt, zistili by sme, že má vizualizačná časť algoritmu dva kroky – úvodný (pôvodné zoradenie hodnôt) a konečný (zoradené hodnoty). Teraz sa hodí do algoritmu pridať vizualizačné prvky a na vhodných miestach zaznamenať kroky, napr.:

```
private void insert(int length, ListItem value){
        int i = length - 1;
```

```
        while (i >= 0 && input.values[i].value > value.value) {
                input.values[i+1] = input.values[i];
               i = i - 1;
}
     input.values[i + 1] = value;
}

public void algorithm(){
        int length = input.length();

        for (int i = 0; i < length; i = i + 1){
                Visualizer.setValueStatus(input,i,2);
                capture();
                insert(i, input.values[i]);
                capture();
                Visualizer.resetValuesStatuses(input);
        }
}
```

Statická metóda ***Visualizer.setValueStatus(input,i,2)*** vezme input, v našom prípade inštanciu triedy SortInput reprezentujúcej input pre triediace algoritmy a status položky na mieste i nastaví na 2. To spôsobí zmenu farby čísla i obdĺžnika reprezentujúceho položku na mieste i na červenú. Metóda ***capture()*** „zachytí“ túto zmenu, to znamená pridá krok do krokovania algoritmu, v ktorom bude dátová štruktúra zvýraznená spôsobom popísaným v predchádzajúcich riadkoch. Statická metóda ***Visualizer.resetValuesStatuses(input)*** spôsobí zmenu statusov všetkých položiek inputu na pôvodnú hodnotu, čo znamená že všetky čísla i obdĺžniky budú mať čiernu farbu. Algoritmus je vizualizovaný.

5.9. Nastavenia a lokalizácia

Rôzne nastavenia - hodnoty textov v programu, súradnice panelov, začiatku vykresľovania algoritmov atď., farebných odlíšení stavov prvkov štruktúr a texty chybových hlášok sú uložené v konfiguračnom súbore „config“. Odtiaľ sú pri inicializácii programu načítané. Konfiguračný súbor má formát XML, na jeho načítanie sa používa XML parser. Priamym zásahom do súboru možno konfiguráciu meniť, ale treba mať na zreteli, že nesprávny zásah vedie k dysfunkcii programu.

Pretože sú v konfiguračnom súbore zoskupené všetky popisné texty, ktoré sa v programu nachádzajú, jeho úpravou môžeme docieliť lokalizáciu programu do iného jazyka.

6. Kapitola - Záver

6.1. Zhodnotenie

Projekt vytvára prostredie pre vizualizáciu algoritmov. Je v ňom vizualizovaných 11 rôznych algoritmov a obsahuje časť pre distančnú výučbu.

Napriek tomu, že je zameraný na grafové a triediace algoritmy a algoritmy vyhľadávania v texte, jeho vhodnou úpravou dokáže vizualizovať i iné druhy algoritmov. Časti ktoré majú všetky algoritmy spoločné, sú implementované v triede ktorú dedia všetky algoritmy. Časti spoločné pre skupinu algoritmov sú zasa implementované v triede od ktorej dedí každý algoritmus v danej skupine. Vďaka tomuto postupu sa programátor pri rozširovaní programu musí zaoberať iba časťami, ktoré sú špecifické pre daný algoritmus, čím je pridanie nového algoritmu relatívne jednoduché, ako bolo vidieť v kapitole 5.8.

Algoritmy majú jednotné ovládanie, aby po práci s jedným algoritmov používateľ bol schopný používať i ostatné algoritmy. Vstupy sú ukladané do súborov vo formáte XML. Sieťová časť funguje kvôli jednoduchosti a prehľadnosti kódu pomocou technológie RMI.

Všetky texty v programe (textové popisy na tlačidlách, text chybových hlásení atď.) sa načítavajú z konfiguračného súboru, jednoduchou editáciou tohto súboru je možné program lokalizovať do iných jazykov.

Pokiaľ porovnáme tento projekt s menšími projektmi, zistíme, že jeho výhoda spočíva v univerzálnosti spomínanej v predchádzajúcim odsekoch. Ani väčšie projekty nie sú tak univerzálne ako tento projekt, nakoľko i väčšie projekty sa zameriavajú väčšinou iba na jednu skupinu algoritmov. Výnimkou je projekt Algovision. V porovnaní s projektom Algovision obsahuje môj projekt naviac sieťovú časť. To znamená program môže okrem role „samouk“, kedy si používateľ osvojuje algoritmy samostatnou prácou s programom, fungovať v roli server a klient. Pokiaľ je program v roli server je k nemu pripojený jeden alebo viac klientov, grafické udalosti na serveri sa budú vykonávať i na všetkých klientoch. Nakoľko v súčasnosti začína byť výučba on-line (on-line prednášky) stále viac populárna, je obohatenie programu o túto možnosť významným doplnením funkčnosti programu.

Projekt bol implementovaný presne tak, ako bol na začiatku navrhnutý, obsahuje všetku funkčnosť stanovenú v špecifikácii. Je pripravený na využitie používateľmi, ktorý sa chcú oboznámiť s tým, ako fungujú niektoré algoritmy, prednášajúcimi, ktorý chcú prednášať svoje prednášky obsahujúce tému fungovania algoritmov on-line i programátormi, ktorý chcú doplniť projekt o nové algoritmy prípadne o novú funkčnosť.

6.2. Ďalší vývoj

Námetov pre ďalší vývoj je viacero. Projekt je možné doplniť o triedy reprezentujúce ďalšie skupiny algoritmov, ako napríklad aritmetické alebo

geometrické algoritmy. Následne možno každú skupinu algoritmov obohatiť minimálne o jeden nový algoritmus. Takisto možno o nové algoritmy obohatiť už existujúce skupiny algoritmov.

Sieťovú časť by bolo vhodné doplniť o prenos zvuku tak, aby boli všetky nástroje pre virtuálnu prednášku v jednom programe a užívateľ nemusel na prenos hlasu používať iné programy.

7. Kapitola – Obsah CD

Program je distribuovaný v zip formáte v súbore vizualizace.zip, ktorý sa nachádza v adresári /bin. Zdrojové kódy sa nachádzajú v adresári /source. PDF verzia tejto bakalárskej práce sa nachádza v adresári /text v súbore bakalarskaPraca.pdf. Vygenerovaný javaDoc sa nachádza v adresári /javadoc.

CD obsahuje i inštaláciu Java JRE verzie 1.5 (verzia pre Windows a verzia pre Linux), Java JDK 1.5 (verzia pre Windows a verzia pre Linux), Apache Ant 1.7.0 (binárne distribúcie, zdrojové distribúcie), všetko v adresári /install. Podporu pre ostatné operačné systémy možno nájsť v [14] a [15]

Literatúra

[1] Single-Purpose Algorithm Visualizations:
http://www.cs.sjsu.edu/faculty/khuri/inv_links2.html .
[2] Šatka Milan: Vizualizace grafových algoritmů, bakalárska práca, Univerzita Karlova 2006.
[3] Hanousek Ondřej: Kompresní algoritmu, jejich implementace a vizualizace, diplomová práca, ČVUT 2005.
[4] Kučera Luděk Prof. RNDr., DrSc.:
http://kam.mff.cuni.cz/~ludek/AlgovisionPage.html.
[5] Microsoft Corporation: Microsoft NetMeeting,
http://www.microsoft.com/windows/netmeeting/ .
[6] Skype Limited: Skype, http://www.skype.com .
[7] Thomas H. Cormen a kol.: Introduction to algorithms, Second edition, The MIT Press 2001.
[8] Drake Peter: Data structures and algorithms in Java, Prentice Hall 2005.
[9] Sedgewick Robert: Algorithms in Java, Third edition, Addison Wesley 2003.
[10] Kathy Walrath a kol.: JFC Swing Tutorial, The: A Guide to Constructing GUIs, Second Edition, Addison Wesley, 2004.
[11] Knudsen Jonathan: Java 2D graphics, O'Reilly Media 1999.
[12] William Grosso: Java RMI, O'Reilly 2001.
[13] Sun Java tutorials: Working with XML, The Java XML Tutorial,
http://java.sun.com/xml/tutorial_intro.html .
[14] Sun Microsystems: Java 2 Standard Edition version 5.0,
http://java.sun.com/j2se/1.5.0/ .
[15] The Apache Software Foundation: The Apache Ant Project,
http://ant.apache.org/ .
[16] The Apache Software Foundation: http://ant.apache.org/manual/install.html .
[17] Wikipedia, The Free Encyclopedia, http://www.wikipedia.org .