UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY
A INFORMATIKY

# VIZUALIZÁCIA ZÁKLADNÝCH GRAFOVÝCH ALGORITMOV

2011

Alena Košinárová

UNIVERZITA KOMENSKÉHO V BRATISLAVE
FAKULTA MATEMATIKY, FYZIKY
A INFORMATIKY

Evidenčné číslo: fce80521-18d2-43f4-abaf-4403ca747eef

# VIZUALIZÁCIA ZÁKLADNÝCH GRAFOVÝCH ALGORITMOV

**bakalárska práca**

Študijný program: Informatika
Študijný odbor: 921 Informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: RNDr. Jana Katreniaková, PhD

**Bratislava, 2011**

**Alena Košinárová**

---

## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Alena Košinárová
**Študijný program:** informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
**Študijný odbor:** 9.2.1. informatika
**Typ záverečnej práce:** bakalárska
**Jazyk záverečnej práce:** slovenský

**Názov:** Vizualizácia základných grafových algoritmov

**Cieľ:** Cieľom práce je zefektívniť proces pochopenia základných grafových algortimov s použitím viuálnych príkladov podrobne vysvetľujúcich ich fungovanie. Táto práca zahŕňa jednak vysvetlenie základných pojmov a grafových štruktúr a následne ich využitie v grafových algoritmoch.

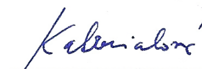
**Vedúci:** RNDr. Jana Katreniaková, PhD.
**Katedra:** FMFI.KI - Katedra informatiky

**Dátum zadania:** 26.10.2010

**Dátum schválenia:** 26.10.2010

doc. RNDr. Daniel Olejár, PhD.
garant študijného programu

.......................................
študent



.......................................
Vedúci

Chcela by som poďakovať svojej bakalárskej vedúcej, RNDr. Jane Katreniakovej, PhD., za ochotný prístup, skvelé rady a veľkú trpezlivosť pri vedení mojej práce.

# Abstrakt

Hlavným cieľom tejto bakalárskej práce je vytvorenie edukačnej aplikácie v jazyku Java, ktorej účelom je osvetlenie a priblíženie fungovania základných grafových algoritmov. Aplikácia je prístupná na internete, aby mohla poslúžiť mladým informatikom pri štúdiu týchto algoritmov. Práca obsahuje taktiež učebné texty osvetľujúce základné definície a teóriu potrebnú pre pochopenie ich fungovania.

**Kľúčové slová:** grafové algoritmy, grafy, vzdelávanie

# Abstract

Main purpose of this bachelor thesis is creation of an educational Java application, which is dedicated to explaining and describing basic graph algorithms. Application can be placed on a website in order to support the learning process of young students of computer science. Thesis also includes basic definitions and theory necessary to understand the behaviour of the algorithms.

**Keywords:** graph algorithms, graphs, education

# Obsah

**Úvod** **1**

**1 Základná teória grafov** **2**

1.1 Pojmy a definície . . . . . . . . . . 2
1.2 Typy grafov . . . . . . . . . . 5
1.3 Ohodnotené grafy . . . . . . . . . . 7

**2 Použité algoritmy** **8**

2.1 Označenia v pseudokódoch algoritmov . . . . . . . . . . 8
2.2 Mosty, artikulácie a súvislosť . . . . . . . . . . 9
2.2.1 Súvislosť . . . . . . . . . . 9
2.2.2 Mosty . . . . . . . . . . 9
2.2.3 Artikulácie . . . . . . . . . . 10
2.3 Prehľadávanie . . . . . . . . . . 11
2.3.1 Prehľadávanie do šírky . . . . . . . . . . 11
2.3.2 Prehľadávanie do hĺbky . . . . . . . . . . 12
2.4 Cesty v grafe . . . . . . . . . . 14
2.4.1 Dijkstrov algoritmus . . . . . . . . . . 14
2.4.2 Bellman-Fordov algoritmus . . . . . . . . . . 16
2.4.3 Floyd-Warshallov algoritmus . . . . . . . . . . 17
2.5 Kostra . . . . . . . . . . 19
2.5.1 Kruskalov algoritmus . . . . . . . . . . 20
2.6 Test bipartity . . . . . . . . . . 21

**3 Implementácia aplikácie** **23**

3.1 Vizualizácia grafu . . . . . . . . . . 23
3.2 Myšlienkový model . . . . . . . . . . 27
3.3 Implementačné detaily . . . . . . . . . . 29
3.4 Ovládanie . . . . . . . . . . 31
3.5 Ilustračné obrázky . . . . . . . . . . 34

**Záver** **36**

**Príloha A** **38**

# Zoznam obrázkov


1 Vizualizácia komponentov súvislosti . . . . . . . . . . . . . . . . . 9
2 Vizualizácia mostov . . . . . . . . . . . . . . . . . . . . . . . . 10
3 Vizualizácia artikulácií . . . . . . . . . . . . . . . . . . . . . . 10
4 Prehľadávanie do šírky . . . . . . . . . . . . . . . . . . . . . . 12
5 Prehľadávanie do hĺbky . . . . . . . . . . . . . . . . . . . . . . 14
6 Dijkstrov algoritmus . . . . . . . . . . . . . . . . . . . . . . . 16
7 Bellman-Fordov algoritmus . . . . . . . . . . . . . . . . . . . . 18
8 Floyd-Warshallov algoritmus . . . . . . . . . . . . . . . . . . . . 19
9 Kruskalov algoritmus . . . . . . . . . . . . . . . . . . . . . . . 21
10 Test bipartity . . . . . . . . . . . . . . . . . . . . . . . . . . 22
11 Vybrané možné rozloženia grafu . . . . . . . . . . . . . . . . . . 25
12 Rozloženie komponentov v aplikácií . . . . . . . . . . . . . . . . 29
13 Ukážky z celej aplikácie . . . . . . . . . . . . . . . . . . . . . . 34
14 Predpripravené grafy . . . . . . . . . . . . . . . . . . . . . . . 35

# Úvod

Grafy patria medzi najdôležitejšie z dátových štruktúr a ako také sa používajú v mnohých odvetviach informatiky. Preto je dôležité, aby boli podané prístupnou formou už začínajúcim študentom informatiky. Avšak občas aj najlepší výklad nestačí, a treba názornú ukážku alebo možnosť vizualizácie práve vysvetľovaných algoritmov. Cieľom tejto bakalárskej práce je vytvoriť Java aplikáciu, pomôcku pre pochopenie grafových algoritmov, v ktorej si budú môcť používatelia prečítať popisy algoritmov a hneď si ich aj pozrieť priamo "v akcií". Pevne verím, že výsledok tejto bakalárskej práce pomôže jednak študentom informatiky, ale tiež študentom stredných škôl, ktorí sa zaujímajú o informatiku. O všestrannej použiteľnosti podobných vizualizačných aplikácii som sa presvedčila sama, keď som sa dostala k bakalárskej práci Jakuba Kováča [3] z roku 2007, ktorá sa zaoberala vizualizáciou vyhľadávacích stromov. Práve táto práca ma inšpirovala k nápadu na moju bakalársku prácu.

Podobné projekty samozrejme už existujú. Najväčším z nich je, pokiaľ viem, Algovízia. Napriek tomu, že považujem Algovíziu za skvelý príklad, usudzujem tiež, že sa v nej nenachádzajú práve základy, na ktoré by som sa chcela zamerať. Keďže cieľovou skupinou sú úplní začiatočníci, má moja práca za cieľ vysvetliť dostatočne prístupne už samotné stavebné kamene týchto algoritmov. Časti práce tiež môžu slúžiť ako sprievodná literatúra. Práca je členená na tri kapitoly. Prvá sa zaoberá formálnymi definíciami obsahujúcimi základy potrebné v neskorších častiach práce. Druhá kapitola je zameraná na jednotlivé algoritmy, ktoré sú rozoberané z hľadiska nielen algoritmického, implementačného, ale aj z hľadiska zložitosti. Všetky algoritmy sú vysvetlené tiež neformálne, s praktickými príkladmi. Posledná kapitola je venovaná popisu implementácie, od počiatočných požiadaviek, cez použité vizualizačné metódy, až po ukážky samotnej aplikácie.

Na záver tohto krátkeho úvodu by som rada venovala túto bakalársku prácu Korešpondenčnému Semináru z Programovania a tejto fakulte, ktorí sa snažia rozvíjať záujem mladých ľudí o informatiku.

# 1 Základná teória grafov

V prvej tematickej kapitole sa budeme venovať teoretickým základom teórie grafov. Pôjde prevažne o základné pojmy a definície, ako aj o uvedenie rôznych typov grafov, s ktorými sa budeme stretávať v ďalších kapitolách. Cieľom tejto kapitoly je priblížiť úplnemu začiatočníkovi problematiku teórie grafov a docieliť vznik dostatočnej zásoby informácií na to, aby sme mohli prejsť k jednotlivým algoritmom.

V celej tejto kapitole chápeme a zavádzame pojmy a definície rovnako, ako ich zaviedol Diestel v jeho Graph Theory [1]. Konáme tak preto, aby sa zachovala zrozumiteľnosť a konzistencia v štruktúre definovania. Definície a vety, ktoré sa nezhodujú s Diestelovým poňatím sú označené hviezdičkou. Ide prevažne o zjednodušené definície - zredukované tak, aby dostačovali pre naše účely, ale nemiatli zbytočne čitateľa.

## 1.1 Pojmy a definície

Na úvod si zadefinujeme základné pojmy, akými sú graf, vrchol či susednosť. Potom prejdeme k definíciam zložitejším, ktoré budú ale pre účely tejto bakalárskej práce esenciálne. Medzi ne patrí okrem iného pojem most či artikulácia, s ktorými sa stretneme hneď na začiatku kapitoly venovanej algoritmom.

Celkovo poznáme dva základné druhy grafov, orientované (pri ktorých majú hrany určený smer) a neorientované (pri ktorých hrany smer nemajú). Napriek tomu, že implementačne je zväčša možné naše algoritmy využiť aj pre grafy orientované, v tejto práci sa budeme zaoberať prakticky len neorientovanými grafmi, a teda všetky potrebné pojmy si budeme definovať len pre neorientované grafy.

**Definícia 1.1.1** ***Graf*** *je dvojica $G = (V, E)$ množín, pre ktoré platí $E \subseteq V \times V$, kde prvky $E$ považujeme za neusporiadané dvojice*[1]*, a teda prvkami množiny $E$ sú dvojprvkové podmnožiny množiny $V$. Predpokladáme, že $V \cap E = \emptyset$.*

**Definícia 1.1.2** *Množinu $V$ z predošlej definície budeme nazývať* ***množina vrcholov*** *a jej prvky budeme označovať* ***vrcholy****. Množinu $E$ z tej istej definície budeme nazývať* ***množina hrán*** *a jej prvky budeme označovať* ***hrany****. Pokiaľ budeme hovoriť o množine vrcholov resp. hrán grafu $G$, budeme túto skutočnosť zapisovať ako $V(G)$ resp. $E(G)$.*

Ďalej nás budú zaujímať rôzne vlastnosti grafov, zväčša súvisiace so vzťahmi medzi vrcholmi a hranami.

**Definícia 1.1.3** ***Rádom grafu*** *$G$ budeme nazývať mohutnosť jeho množiny vrcholov $V(G)$ a budeme ho označovať $|G|$. Mohutnosť množiny hrán grafu $G$, čiže mohutnosť $E(G)$ budeme označovať $||G||$.*

**Definícia 1.1.4** *Vrchol $v$ je* ***incidentný*** *s hranou $e$ práve vtedy, keď $v \in e$. Dva vrcholy $u, v$ sú* ***susedné*** *práve vtedy, keď existuje hrana $e = \{u, v\}$ (alebo skrátene $e = uv$). Dve hrany $e, f$ sú* ***susedné*** *práve vtedy, keď existuje vrchol $v$, taký, že $v \in e$ a zároveň $u \in f$. Hranu $e = \{u, u\}$ budeme označovať* ***slučka****.*

**Definícia* 1.1.5** ***Stupňom vrcholu*** *$v$, alebo tiež $deg(v)$ označujeme počet hrán zasahujúcich do tohto vrchola. Slučky počítame dva-krát. Formálne teda $deg(v) = |\{e \in E | v \in e\}| + |\{(v, v) \in E | v \in V\}|$.*

**Definícia 1.1.6** ***Minimálny stupeň*** *$\delta(G)$ grafu $G$ je $\min \{deg(v) | v \in V\}$.* ***Maximálny stupeň*** *$\Delta(G)$ grafu $G$ je $\max \{deg(v) | v \in V\}$.*

**Definícia 1.1.7** *Vrchol so stupňom 0 nazývame tiež* ***izolovaný****.*

V ďalších definíciach sa zameriame na štruktúry ktoré vzniknú ako postupnosti vrcholov a hrán.

**Definícia 1.1.8** *Cesta je neprázdny graf $P = (V, E)$, ktorý spĺňa nasledujúce podmienky $V = \{x_0, x_1, x_2, \ldots, x_k\{$ a $E = \{x_0x_1, x_1x_2, \ldots, x_{k-1}x_k\}$ kde všetky $v_i$ sú navzájom rôzne. Túto cestu nazývame tiež $x_0 - x_k$ cesta. Vrcholy $v_0$ a $v_k$ sú* ***spojené*** *cestou $P$ a nazývame ich* ***konce cesty*** *$P$. Ostatné vrcholy nazývame* ***vnútorné vrcholy****.*

[1] V origináli definície sa použil ekvivalentný zápis $E \subseteq [V]^2$

**Definícia 1.1.9** *Dve cesty nazývame* ***nezávislé****, pokiaľ nemajú spoločný žiadny vnútorný vrchol.*

**Definícia 1.1.10** *Cestu, ktorej počiatočný a koncový vrchol sa zhodujú, a má dĺžku aspoň 3, budeme nazývať* ***cyklus****. Alternatívne, ide o cestu, ku ktorej pridáme hranu spájajúcu jej koncové vrcholy.*

**Definícia 1.1.11** *Hranu, ktorá spája dva rôzne vrcholy cyklu, ale nepatrí medzi hrany samotného cyklu, budeme označovať* ***tetiva****.*

**Definícia 1.1.12** *Pokiaľ graf obsahuje podgraf vo forme cyklu, nazývame tento graf* ***cyklický****. V opačnom prípade ho označujeme ako* ***acyklický****.*

Na záver tohto teoretického úvodu sa pozrieme na súvislosť grafov, pod čím si môžeme predstaviť tiež otázku - na koľko samostatných častí sa nám graf "rozpadá"?

**Definícia* 1.1.13** *Graf $G = (V, E)$ nazývame* ***súvislý****, pokiaľ pre každé dva vrcholy $u, v \in V$ existuje $u - v$ cesta v grafe $G$. Pokiaľ graf nie je súvislý, voláme ho* ***nesúvislý*** *a vieme ho rozdeliť na súvislé podgrafy, ktoré nazývame* ***komponenty súvislosti****.*

**Definícia 1.1.14** *Graf $G = (V, E)$ nazývame $k$-**súvislý** (resp. občas tiež $k$-**vrcholovo-súvislý**), kde $k \in \mathbb{N}$, pokiaľ je súvislý, má aspoň $k$ vrcholov a po odobratí ľubovoľnej množiny $S$ vrcholov grafu $G$, kde $|S| < k$, bude graf $G - S$ stále súvislý. Alternatívne, že žiadne dva vrcholy $G$ nemôžu byť oddelené menej ako $k$ vrcholmi. Najväčšie také $k$, pre daný graf, je označované tiež* ***súvislosť grafu*** *$G$, alebo tiež $\kappa(G)$.*

**Definícia 1.1.15** *Graf $G = (V, E)$ nazývame $\ell$-**hranovo-súvislý**,kde $\ell \in \mathbb{N}$, pokiaľ je súvislý, má aspoň 1 vrchol a po odobratí ľubovoľnej množiny $F$ hrán $G$, kde $|F| < \ell$, bude graf $G - F$ stále súvislý. Alternatívne, že žiadne dva vrcholy $G$ nemôžu byť oddelené menej ako $\ell$ hranami. Najväčšie také $\ell$, pre daný graf, je označované tiež* ***hranová súvislosť grafu*** *$G$, alebo tiež $\lambda(G)$.*

**Definícia* 1.1.16** *Pokiaľ každá cesta vedúca medzi množinami $A, B \subseteq V$ obsahuje vrchol alebo hranu z $S$, hovoríme, že $S$* ***oddeľuje*** *množiny $A$ a $B$ v $G$. Vrchol, ktorý takto rozdeľuje komponent grafu na dva samostatné komponenty nazývame* ***artikulácia****. Hranou s rovnakou vlastnosťou nazývame* ***most****.*

## 1.2 Typy grafov

V tejto podkapitole sa budeme venovať definíciam rôznych typov grafov. Ich vlastnostiam sa budeme venovať len zriedka a stručne, pretože väčšina z nich nie je podstatná pre pochopenie neskorších algoritmov. Napriek tomu sa však objavia v texte aj vety, ktoré poukazujú na informacie, ktoré budeme využívať. Dôkazy viet vynechávame, pretože nie sú v našom prípade potrebné. V prípade záujmu je možné ich vyhľadať v zdrojoch uvedených v bibliografií.

**Definícia 1.2.1** ***Prázdny graf*** *$G = (\emptyset, \emptyset)$ budeme označovať $\emptyset$ a ma rád 0.*

**Definícia 1.2.2** *Za* ***triviálny graf*** *budeme považovať graf s rádom 0 alebo 1.*

**Definícia 1.2.3** ***Kompletným grafom*** *$K_n$ budeme nazývať graf, ktorý má hranu medzi každými dvoma rôznymi vrcholmi, čiže ak sú každé dva vrcholy susedné.*

**Definícia 1.2.4** ***Komplementárny graf*** *$\bar{G} = (V', E')$ (alebo tiež* ***komplement****) je graf, ktorý vznikne z grafu $G = (V, E)$ nasledujúcim spôsobom: $V' = V$; $E' = [V]^2 \setminus E$ (tzn. hrana $xy$ je v $\bar{G}$ práve vtedy, keď nie je v $G$).*

**Veta* 1.2.5** *Komplementom kompletného grafu je graf zložený z izolovaných vrcholov.*

**Definícia 1.2.6** ***Hranovým grafom*** *$L(G)$ grafu $G$ nazývame taký graf na množine $E$, v ktorom sú vrcholy $x, y \in E$ susedné v $L(G)$ práve vtedy, keď sú hrany $x, y$ susedné v $G$.*

**Definícia 1.2.7** ***$K$-regulárnym grafom*** *nazývame graf, ktorého všetky vrcholy majú rovnaký stupeň rovný $k$. 3-regulárny graf nazývame* ***kubický****.*

**Definícia 1.2.8** *Graf $G = (V, E)$ je* ***bipartitný*** *práve vtedy, keď môžeme množinu vrcholov $V$ rozdeliť na dve disjunktné podmnožiny, pre ktoré platí, že žiadne dva vrcholy z tej istej podmnožiny nie sú susedné.*

**Definícia 1.2.9** *Graf $G = (V, E)$ je* ***$r$-partitný*** *práve vtedy, keď môžeme množinu vrcholov $V$ rozdeliť na $r$ disjunktných podmnožín, pre ktoré platí, že žiadne dva vrcholy z tej istej podmnožiny nie sú susedné.*

**Veta 1.2.10** *Graf je bipartitný práve vtedy, keď neobsahuje žiaden cyklus nepárnej dĺžky.*

**Definícia 1.2.11** *Acyklický graf $G$ nazývame tiež* ***les****. Súvislý acyklický graf nazývame* ***strom****.* ***Les*** *sa teda skladá zo* ***stromov****. Vrcholy* ***stromu****, ktoré majú stupeň 1 nazývame* ***listy****.* ***Koreňom*** *stromu môžeme nazvať ľubovoľný vrchol stromu. Inak povedané, môžeme strom zakoreniť v ľubovoľnom vrchole.*

**Veta 1.2.12** *Nasledujúce tvrdenia sú ekvivalentné pre graf $G$:*

- *$G$ je strom;*
- *každé dva vrcholy grafu $G$ sú spojené práve jednou cestou v $G$;*
- *$T$ je minimálne súvislý, t.j. $G$ je súvislý, ale $T - \{e\}$ nie je súvislý pre ľubovoľnú hranu $e$ z $G$;*
- *$T$ je maximálne acyklický, t.j. $G$ neobsahuje žiaden cyklus, ale $T + \{xy\}$ už cyklus obsahuje, pre ktorékoľvek dva nesusedné vrcholy $x$, $y$ z $G$;*

**Veta 1.2.13** *Súvislý graf s $n$ vrcholmi je strom práve vtedy, keď má $n - 1$ hrán.*

**Definícia* 1.2.14** ***Kostra grafu*** *$G = (V, E)$ je každý taký jeho podgraf $T = (V', E')$, pre ktorý platí nasledujúce: $V = V'$, $E' \subseteq E$ a $T$ je strom.*

**Veta 1.2.15** *Každý súvislý graf obsahuje kostru, ktorá môže byť zakorenená v ľubovoľnom vrchole.*

**Definícia* 1.2.16** *Graf, ktorý vieme zakresliť do roviny tak, aby sa žiadne dve jeho hrany nepretínali na inom mieste ako v ich spoločnom vrchole, nazývame* ***planárny graf****. Pokiaľ už tak nakreslený je, používa sa preň občas tiež označenie* ***rovinný graf****.*

**Definícia* 1.2.17** *Uzavretý ťah (cesta, v ktorej sa môžu vrcholy opakovať), ktorý obsahuje každú hranu grafu $G$ práve raz, nazývame tiež* ***eulerovský ťah****. Inituitívne môžeme prirovnať k uzavretému kresleniu obrazca jedným ťahom.*

**Definícia 1.2.18** *Graf, v ktorom existuje Eulerovský ťah, nazývame* ***eulerovský graf****.*

**Veta 1.2.19** *Spojitý graf je eulerovský práve vtedy, keď majú všetky jeho vrcholy párny stupeň.*

**Definícia* 1.2.20** *Kružnicu, ktorá obsahuje každý vrchol grafu $G$ práve raz, nazývame tiež* ***hamiltonovská kružnica****. Cestu, ktorá obsahuje každý vrchol grafu $G$ práve raz, nazývame* ***hamiltonovská cesta****.*

**Definícia 1.2.21** *Graf, v ktorom existuje hamiltonovská kružnica, nazývame* ***hamiltonovský graf****.*

## 1.3 Ohodnotené grafy

Je mnoho situácií z reálneho i akademického života, kedy nám grafy ako dátová štruktúra nedostačujú. Ako príklad uvedieme jednak cenovo rozdielne lety medzi letiskami, ktoré tvoria pri vhodne zvolenej forme zápisu graf, alebo tiež rôzne dlhé vzdialenosti kľúčových miest na mape pri tvorbe okružnej cesty. V oboch prípadoch potrebujeme priradiť hranám grafu určitú hodnotu, či už ide o cenu letu, dĺžku trasy, alebo iné kritérium. Ďalším prípadom ohodnotenia je priradenie hodnôt jednotlivým vrcholom, napríklad v prípade, kedy sa jedná o poplatky na mieste reprezentovanom jedným vrcholom grafu, alebo napríklad o zisk, ktorý plynie z návštevy daného vrcholu.

V tejto podkapitole sa pokúsime definovať uvedené pojmy čo najjednoduchšie a zároveň tak, aby sme zachovali konzistenciu s predošlými definíciami. Avšak nasledujúce definície nie sú konkrétne podložené v Diestelovej Graph Theory [1], a teda vytvoríme vlastné definície.

**Definícia* 1.3.1** ***Hranovo ohodnoteným grafom*** *budeme nazývať taký graf $G = (V, E, c)$, pre ktorý existuje* ***hranová ohodnocovacia funkcia*** *$c : E \rightarrow \mathbb{R}$, ktorá každej hrane $e \in E$ priradí reálne číslo $c(e)$ - jej cenu, resp. váhu.*

**Definícia* 1.3.2** ***Kladne hranovo ohodnotený graf*** *je taký graf, pre ktorého hranovú ohodnocovaciu funkciu $c$ platí, že $\forall e \in E : c(e) > 0$.*

**Definícia* 1.3.3** ***Záporný cyklus*** *na hranovo ohodnotenom grafe $G = (V, E, c)$ je taký cyklus na jeho vrcholoch - $v_0 v_1 v_2 \ldots v_k$ - pre ktorý platí $c(v_0 v_1) + c(v_1 v_2) + \ldots + c(v_k v_0) < 0$.*

**Definícia* 1.3.4** ***Vrcholovo ohodnotený graf*** *je taký graf $G = (V, E, w)$, pre ktorý existuje* ***vrcholová ohodnocovacia funkcia*** *$w : V \rightarrow \mathbb{R}$, ktorá každému vrcholu $v \in V$ priradí reálne číslo $w(v)$ - jeho cenu.*

**Definícia* 1.3.5** ***Kladne vrcholovo ohodnotený graf*** *je taký graf, pre ktorého vrcholovú ohodnocovaciu funkciu platí, že $\forall v \in V : w(e) > 0$.*

# 2 Použité algoritmy

Po tom, čo sme vybudovali dostatočný teoretický základ a funkčný aparát z teórie grafov, môžeme prejsť k popisom konkrétnych problémov a algoritmov na ich vyriešenie. Ako zistíme aj neskôr, nie vždy vieme alebo chceme zrealizovať najoptimálnejšie riešenie toho-ktorého problému, pretože môžu presahovať naše momentálne znalosti. Avšak pokúsime sa venovať širšej palete problémov a algoritmov a od jednoduchších sa dostať až k zložitejším. Nad rámec týchto materiálov budeme predpokladať u čitateľa aspoň základnú predstavu o časovej zložitosti - na účely pochopenia efektivity jednotlivých algoritmov.

Táto kapitola bude mať štruktúru odlišnú od predošlej. Vedeli by sme si ju pomyseľne rozdeliť na teoretickú a implementačnú časť popisu problémov a algoritmov. Najprv sa v teoretickej príprave vždy pozrieme na definíciu problému, ktorý chceme riešiť a skusíme ho neformálne ozrejmiť na jednom či dvoch praktických príkladoch, následne si uvedieme algoritmus, alebo algoritmy, ktoré sa používajú na jeho riešenie, taktiež dodáme pseudokód a časovú zložitosť každého z nich. V implementačnej časti sa pozrieme okrem neodmysliteľného ilustračného obrázku z behu daného algoritmu sa pozrieme tiež na potenciálne úskalia a detaily implementácie. V prípade, že by si čitateľ chcel preštudovať algoritmy podrobnejšie, chceli by sme poukázať na Introduction to Algorithms[2], ako na vhodnú odporúčanú literatúru pre tieto účely.

## 2.1 Označenia v pseudokódoch algoritmov

V rámci jednotnosti pseudokódov budeme používať tieto označenia:

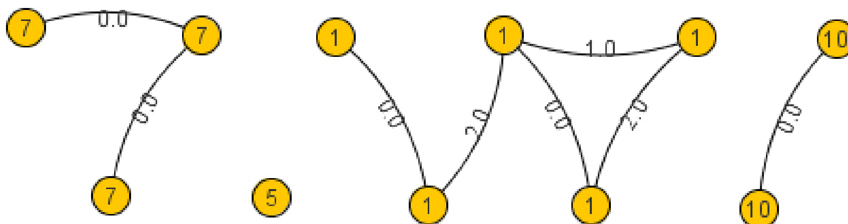
| | |
|---|---|
| $x \leftarrow 4$ | Priradenie do $x$ |
| $x \rightarrow F$ | Pridanie prvku do dátovej štruktúry |
| $x \leftarrow F$ | Vybratie prvku z dátovej štruktúry a vloženie do $x$ |
| $N_G[v]$ | Množina vrcholov susedných s $v$ v grafe $G$ |
| $\text{Weight}(e)$ | Cena hrany $e$ |

## 2.2 Mosty, artikulácie a súvislosť

Po tom, čo sme si ujasnili označenia, môžeme prejsť k podkapitole, ktorá sa ešte nevenuje priamo algoritmom, ale je pomôckou pre pochopenie niektorých pojmov a definícií z predchádzajúcej kapitoly. Konkrétne sa budeme venovať zobrazeniu komponentov súvislosti, mostov a artikulácií v ľubovoľnom grafe.

### 2.2.1 Súvislosť

Napriek tomu, že by sa mohlo zdať, že pochopenie toho, čo je komponent súvislosti, môže byť na prvý pohľad triviálne, rozhodli sme sa zaradiť názornú ukážku. Samotný algoritmus, ktorý sme zvolili, ukazuje tiež iný dôležitý postup, ktorý budeme využívať aj v nasledujúcich algoritmoch. Ide o dvojicu operácií Union / Find [2], ktoré sa vykonávajú s množinami, v našom prípade vrcholov. Využívame tu princíp, pri ktorom sa každý vrchol (pokiaľ sa tak ešte nestalo) snažíme pripojiť do tej istej množiny, do ktorej patria jeho susedia. Tým zabezpečíme, že v konečnom dôsledku budeme mať samostatné množiny vrcholov zodpovedajúce práve komponentom súvislosti v grafe. Ilustráciou vizualizácie je Obr.1.

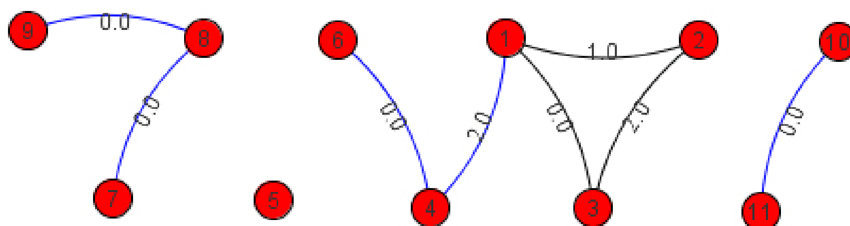



Obr. 1: **Vizualizácia komponentov súvislosti.** Finálny stav spracovania grafu. Na obrázku vidíme 4 komponenty, každý z nich označený viac-menej náhodným číslom vrchola, ktorý doň patrí.

### 2.2.2 Mosty

Ďalším častým problémom býva hľadanie mostov v zadaných grafoch. Opäť nemáme na mysli problém algoritmický, i keď to samozrejme tiež, ale

[2] O ktorých sa môžeme dozvedieť viac napríklad v knihe Introduction to Algorithms [2], v kapitole 21.




Obr. 2: **Vizualizácia mostov.** Finálny stav spracovania grafu.

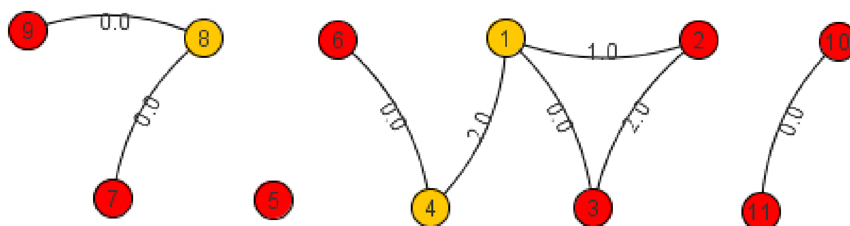
problém s chápaním pojmu most. Samotné optimálne známe riešenie, v čase $O(|V|+|E|)$ tohto algoritmického problému presahuje náročnosť tohto textu a na naše účely - vizualizácia existujúcich mostov v grafe - nie je potrebné. Pozorný čitateľ si v prípade záujmu môže vyhľadať tento algoritmus podľa jeho autora, Roberta Tarjana. Na naše účely postačila implementácia založená na princípe Union-Find a vynechávania jednotlivých hrán. Pokiaľ sa po odobratí hrany zvýši počet komponentov, jedná sa presne o definíciu mostu. Pri vizualizácií sa vysvietená hrana buď prefarbí na šedo, pokiaľ nie je mostom, alebo na modro, pokiaľ je mostom. Ilustráciou vizualizácie je Obr.2.

### 2.2.3 Artikulácie

Posledným z trojice čisto vizualizačných problémov je vizualizácia artikulácií. Tak ako pri mostoch, zvolili sme aj tu algoritmus, ktorý je úrovňou vhodný pre náročnosť tohto textu. Funguje na tom istom princípe ako algoritmus pre vizualizáciu mostov. Prechádza totiž cez všetky vrcholy a kontroluje, či ide o artikuláciu tak, že skontroluje, či by sa zvýšil celkový počet komponentov v prípade, že tento vrchol odstránime. Tu je dôležité si uvedomiť, že odstránenie izolovaného vrcholu nám nezvyšuje počet komponentov, a že takýto vrchol nie je artikuláciou. Ilustráciou vizualizácie je Obr.3.

## 2.3 Prehľadávanie

Medzi prvé typy algoritmov, na ktoré sa v tomto texte pozrieme, patria prehľadávania grafu. Ako názov napovedá, pôjde o systematické prechádzanie vrcholov grafu podľa ich dosiahnuteľnosti z počiatočného vrchola. Názov je jednoducho odvoditeľný od situácií akými sú hľadanie predmetu v blu-




Obr. 3: **Vizualizácia artikulácií.** Finálny stav spracovania grafu.

disku či labyrinte. Istotne si vieme rýchlo spomenúť na možnosti ako "pravou rukou popri stene". Aj to je jeden z možných algoritmov, akými sa môžeme pokúšať o prehľadávanie labyrintu zapísaného vo forme grafu. Ako však čitateľ rýchlo zistí, práve tento uvedený algoritmus nám ukazuje istý problém a tým je možnosť sa zacykliť. Preto sa pozrieme na iné algoritmy, ktoré tento problém nemajú, a sú teda z hľadiska teórie grafov pre nás zaujímavejšie. Tradičným účelom prehľadávacích algoritmov je buď hľadanie konkrétneho vrchola, alebo snaha obsiahnuť všetky vrcholy niektorého komponentu.

### 2.3.1 Prehľadávanie do šírky

Prehľadávanie do šírky, anglicky Breadth-First Search (BFS), je, ako už bolo spomínané pred okamihom, systematické prechádzanie cez všetky dostupné vrcholy. Dôležitým faktorom je práve faktor výberu poradia vrcholov, v ktorom ich budeme prechádzať. V prípade prehľadavania do šírky je postup založený na dátovej štruktúre typu FIFO[3], čiže napríklad, ako a v našom prípade, fronta[4]. Vždy, keď spracovávame niektorý z vrcholov, tak pridáme na koniec fronty všetkých jeho ešte nespracovaných susedov. Následne prejdeme na ďalší vrchol, ktorý je na začiatku fronty a opakujeme postup.

Algoritmus má časovú zložitosť v závislosti od vstupného grafu $G = (V, E)$ práve $O(|V| + |E|)$, pretože pri najhoršom prípade sa pokúsi každou hranou prejsť raz a spustiť sa v každom vrchole.

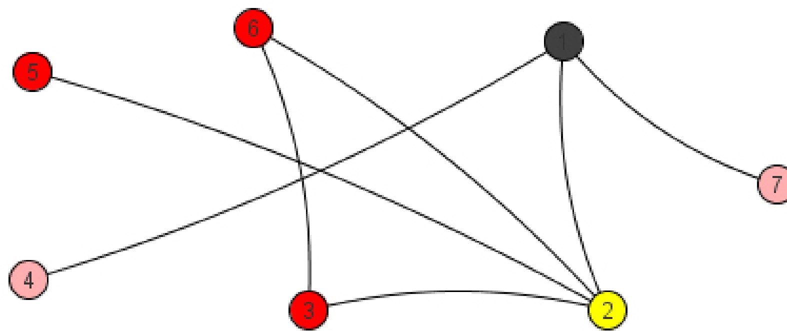
V našej aplikácií sme frontu implementovali pomocou bežne dostupnej štruktúry v Jave - `LinkedList`. Označovanie sme zvolili dobre viditeľné - tmavým sme označili vrcholy, ktoré sú už prejdené a plne spracované. Ružová

[3]First in, first out - alebo tiež kto prv príde, ten prv melie. V našom prípade tiež prv odíde.

[4]V slovenčine oficiálne nazývaná rad

**Algoritmus 1** Prehľadávanie do šírky

**procedure** BFS($G, z$)
  vytvor frontu $F$
  $z \to F$
  označ $z$
  **while** $F$ nie je prázdne **do**
    $v \leftarrow F$
    **for all** $w \in N_G[v]$ **do**
      **if** $w$ nie je označené **then**
        označ $w$
        $w \to F$




Obr. 4: **Prehľadávanie do širky.** V tomto momente je vrchol 1 spracovaný, vrchol 2 aktuálne spracovávaný, a vrcholy 4 a 7 vo fronte čakajú na spracovanie.

farba zodpovedá vrcholom, ktoré sa momentálne nachádzajú vo fronte, a pôvodná farba vrcholov je zachovaná pre ešte nespracované vrcholy. Taktiež máme možnosť vidieť práve aktívny vrchol. Ilustráciou behu algoritmu je Obr.4.

### 2.3.2 Prehľadávanie do hĺbky

Prehľadávanie do hĺbky, anglicky Depth-First Search (DFS), sa od vyššie uvedeného algoritmu líši v poradí, v akom prehľadáva vrcholy. Na rozdiel od predošlého algoritmu, ktorý sa "rozširoval", prehľadávanie do hĺbky sa vždy snaží ísť prvou možnou neznámou cestou. To už intuitívne naznačuje, že na rozdiel od prehľadávania do šírky a FIFO fronty, bude prehľadávanie

do hĺbky používať LIFO [5] typ štruktúry - zásobník. Vždy, keď spracovávame niektorý vrchol, pridáme rovno prvého jeho nespracovaného suseda do zásobníku a začneme spracovávať jeho namiesto potenciálnych starších nájdených vrcholov. Je dôležité si uvedomiť, že vrchol zo zásobníku zatiaľ neodstraňujeme, keď sa posúvame na niektorého z jeho susedov.

Tento postup opakujeme, kým sa nám nestane, že niektorý vrchol už nemá susedov, ktorí by neboli objavení. Vtedy ho označíme za definitívne spracovaný, vyberieme ho z vrchu zásobníku a vrátime sa cestou, ktorou sme k nemu prišli (to nám zabezpečuje práve zásobník a fakt, že jeho predchodca sa jednoznačne nachádza v zásobníku tesne pod ním). V tomto prípade nám namiesto samostatného zásobníku poslúži rekurzia, ktorá nám zabezpečí rovnaké usporiadavanie jednotlivých volaní, aké by sme dosiahli bez rekurzie pomocou zásobníku. Kvôli lepšej zrozumiteľnosti preto uvediem pseudokód rekurzívnej implementácie.

Algoritmus má časovú zložitosť v závislosti od vstupného grafu $G = (V, E)$ práve $O(|V| + |E|)$, pretože každou hranou [6] a každým vrcholom.

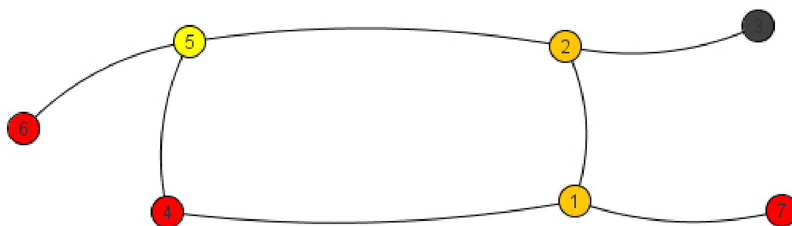
**Algoritmus 2** Prehľadávanie do hĺbky

**procedure** DFS($G, z$)
  označ $z$ za objavený
  **for all** $v \in N_G(z)$ **do**
    **if** $w$ nie je objavený ani spracovaný **then**
      DFS($G, w$)
  označ $z$ za spracovaný

Zásobník sme pri potrebe krokovať program (kedy je rekurzia viac než nežiadaná) implementovali taktiež pomocou `LinkedList`. Grafické označenie vrcholov sa oproti predošlému algoritmu jemne líši. Definitívne spracované vrcholy síce tiež označujeme tmavou farbou, avšak vrcholy, cez ktoré sme už prešli (označili ich) a znova cez ne ešte plánujeme prejsť označujeme oranžovou. Zvýraznenie aktívneho vrcholu žltou farbou v rámci animácie zostáva taktiež nezmenené. Ilustráciou behu algoritmu je Obr.5.

[5] Last in, first out - kto neskoro chodí, prvého ho vybavia.

[6] Maximálne dvakrát - tam a nazad. Smerom k novému vrcholu, a potom pri tom ako sa z už definitívne spracovaného vrcholu vraciame k jeho predchodcovi.




Obr. 5: **Prehľadávanie do hĺbky.** Zatiaľ sme prešli postupne 1 - 2 - 3 - 2 - 5, preto je vrchol 3 v tomto momente už definitívne spracovaný, do vrcholov 1 a 2 sa ešte vrátime, a cez vrchol 5 práve prechádzame.

## 2.4 Cesty v grafe

V tejto podkapitole sa pozrieme na problémy súvisiace prevažne s ohodnotenými grafmi. Problém najlacnejších ciest v grafe jedným z tradičných problémov z praxe, či už ide o najlacnešiu možnú prepravu medzi mestami v niektorej oblasti, alebo o najrychlejší prenos dát po linkách medzi počítačmi v sieti. Môže sa stať, že nás zaujímajú len cesty z niektorého vrcholu do iných, alebo sa môže stať, že nás zaujíma kompletná informacia o každých dvoch vrcholoch. Problém, pri ktorom by sme hľadali najkratšiu cestu medzi konkrétnymi dvoma vrcholmi, sa ukázal ako rovnako náročný, ako problém hľadania vzdialeností všetkých vrcholov od jedného, a teda sa ním nebudeme zaoberať, keďže sa jedná o rovnaké spôsoby riešenia. K týmto problémom je možné pristupovať rôznymi spôsobmi, a vieme využiť tiež princípy dynamického programovania, či takzvané "pažravé" algoritmy.

### 2.4.1 Dijkstrov algoritmus

Prvým algoritmom, ktorému sa budeme pri tomto probléme venovať bude Dijkstrov algoritmus, pomocou ktorého vieme vypočítať dĺžky najkratších ciest z počiatočného vrcholu do všetkých ostatných. Jedná sa o "pažravý" algoritmus, čiže algoritmus, ktorý vyberá lokálne najlepšie riešenia za účelom nájdenia globálne najlepšieho riešenia. Pre jednoduchosť vysvetlenia predpokladajme, že pokiaľ medzi dvomi vrcholmi nie je v grafe hrana, majú tieto vrcholy pomyseľnú hranu s cenou $\infty$. Taktiež predpokladáme, že cena slučiek je nulová.

Dijstrov algoritmus postupuje takto: V každom kroku označí momen-

tálne najbližší vrchol a potom prepráta pre všetky neoznačené vrcholy, či do nich neexistuje lepšia (lacnejšia) cesta cez novooznačený vrchol. Tento postup opakuje, dokým nie su označené všetky vrcholy. Vtedy sa budú ceny najlacnejších ciest pre každý vrchol $v$ nachádzať v poli ako $D[v]$. Je dôležité poznamenať, že Dijkstrov algoritmus má určité obmedzenia, čo sa vstupu týka. Presnejšie, je nutné, aby boli ceny hrán nezáporné reálne čísla. V opačnom prípade totiž nastáva možnosť, že by vrchol susediaci s touto hranou po označení ovplyvnil dĺžky ciest medzi už označenými vrcholmi. Problém teda nastáva vďaka tomu, že ide o "pažravý" algoritmus. Implementačne je taktiež možné získať konkrétne najkratšie cesty tým, že si ku každému vrcholu budeme pamätať tiež jeho predchodcu v najkratšej zatiaľ nájdenej ceste. Avšak základný algoritmus túto dodatočnú funkcionalitu nemá a teda sme ju pri implementácií vynechali.

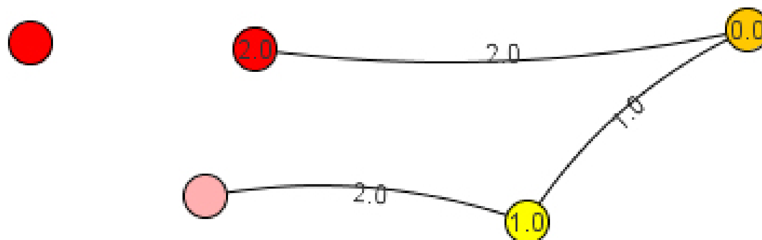
Dijkstrov algoritmus má časovú zložitosť v závislosti od vstupného grafu $G = (V, E)$ buď $O(|V|^2)$, v prípade triviálnej implementácie cez pole, alebo časy podobné $O(|E| + |V|\log(|V|))$ v prípade implementácie s prioritnou frontou, minhaldou, a pod.

**Algoritmus 3** Dijkstrov algoritmus

```
procedure DIJKSTRA(G, z)
    označ z za objavený
    D[z] ← 0
    for all v ∈ V\{z} do
        D[v] ← Weight(zv)
    while nie sú označené všetky vrcholy v G do
        vyber neoznačený v ∈ V s minimálnou D[w]
        označ w
        for all neoznačený v ∈ V do
            D[v] ← min{D[v], D[w] + Weight(vw)}
```

V našej implementácií sme namiesto $\infty$, ktoré sa zle technicky realizuje používali iné jednoznačné označenie pre cenu neexistujúcich hrán, presnejšie, nechavali sme tietohodnoty `null`-ové. Tým sa nám stal síce samotný program trošku menej elegantný, ale výslednej efektivite ani funkcionalite to neubralo. Taktiež namiesto čísel vrcholov sme zvolili zobrazovanie hodnoty $D[v]$ pre každý vrchol $v$. Grafické znázornenie už označených vrcholov je s pomocou oranžovej farby, aktuálne najlepší nájdený vrchol (pre pridanie v

ďalšom kroku) je žltej farby, a práve prezeraný vrchol (a porovnávaný so žltým vrcholom) je ružovej farby. Ilustráciou behu algoritmu je Obr.6.




Obr. 6: **Dijkstrov algoritmus.** Spracovaný je len počiatočný vrchol a nachádzame sa vo fáze hľadania budúceho pridaného vrcholu. Tie hodnoty $D[v]$, ktoré nie sú $\infty$, sú zobrazené vo vrcholoch.

### 2.4.2 Bellman-Fordov algoritmus

Druhý algoritmus, ktorý rieši rovnaký problém ako Dijkstrov algoritmus je algoritmus od pánov Bellmana a Forda. Na rozdiel od Dijkstrovho algoritmu je schopný pracovať aj so zápornými hranami, avšak má isté obmedzenia. Presnejśie, nevie pracovať s grafmi, ktoré obsahujú záporné cykly, t.j. cykly so záporným súčtom cien hrán. Ďalším rozdielom je fakt, že sa nejedná o "pažravý" algoritmus. Naproti tomu sa tento algoritmus opakovane pokúša o vylepšenie existujúcej vzdialenosti použitím niektorej z hrán. Kontrola, či graf neobsahuje záporné cykly je súčasťou algoritmu.

Napriek tomu, že nám to môže spôsobiť "zbytočnú" prácu, nachádza sa táto kontrola až na konci algoritmu. Dôvod je triviálny a tým je, že na konci algoritmu je možne vykonať kontrolu výrazne jednoduchšie a tiež s menšou časovou zložitosťou. Dôležitou súčasťou algoritmu z implementačného hladiska je tiež fakt, že si pamätáme tiež predchodcu každého vrchola v jeho zatiaľ najlepšej ceste. Náš pseudokód je pre pôvodný Bellman-Fordov algoritmus, ktorý funguje na orientovaných grafoch. Naša implementácia je upravená potreby použitia pre neorientované grafy.

Bellman-Fordov algoritmus má časovú zložitosť v závislosti od vstupného grafu $G = (V, E)$ práve $O(|E|.|V|)$, pretože maximálna dĺžka cesty zo zadaného začiatočného vrcholu je rádovo $|V|$ a na jej vybudovanie sa snažíme $|V|$-krát zlepšiť cesty použitím každej z $|E|$ hrán.

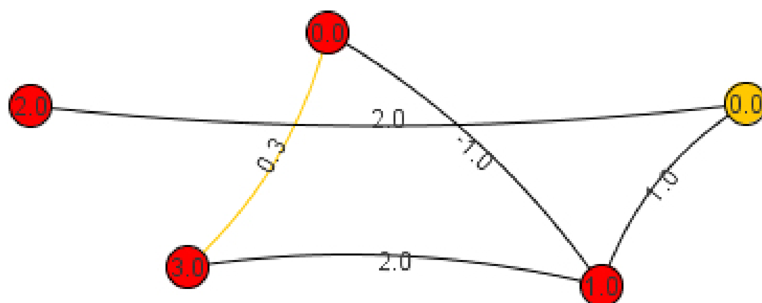
**Algoritmus 4** Bellman-Fordov algoritmus

```
procedure BELLMAN-FORD(G, z)
    D[z] ← 0
    predecessor[z] ← null
    for all v ∈ V ∖ {z} do
        D[v] ← ∞
        predecessor[v] ← null
    for i = 1 to |V| − 1 do
        for all uv ∈ E do
            u ← začiatočný vrchol uv
            v ← koncový vrchol uv
            if D[u]+Weight(uv)< D[v] then
                D[v] ← D[u]+Weight(uv)
                predecessor[v] ← u
    for all uv ∈ E do
        if D[u]+Weight(uv)< D[v] then
            Chyba: Obsahuje záporný cyklus.
```

Hlavným faktorom implementácie tohto algoritmu je, že sme si ho upravili na symetrický - vhodný pre neorientované grafy. Ubralo to síce algoritmu na jednoduchosti, ale nezmenilo to jeho pointu a ani časovú zložitosť. Opäť sme namiesto $\infty$, používali `null`. Farebné zvýrazňovanie sa v tomto algoritme zredukovalo na zvýraznenie našeho začiatočného vrchola a aktuálne spracovávanej hrany. Aktualizácia dát počas výpočtov je viditeľná taktiež v jednotlivých vrcholoch. Ilustráciou behu algoritmu je Obr.7.

### 2.4.3 Floyd-Warshallov algoritmus

Tretím a posledným, tu rozoberaným, algoritmom na hľadanie najlacnejších ciest v grafe je Floyd-Warshallov algoritmus. Na rozdiel od dvoch vyššie uvedených algoritmov, Floyd-Warshall vypočíta ceny najlacnejších ciest medzi každou dvojicou vrcholov v grafe. Na začiatku je vzdialenosť každých dvoch vrcholov rovná cene hrany medzi nimi, pokiaľ taká existuje. Ak nie, je rovná $\infty$. Dôležitým faktom tiež je, že cena slučiek je nulová. Fungovanie algoritmu spočíva v postupe, pri ktorom sa pre každú dvojicu vrcholov opakovane pokúša zlepšiť ich vzájomnú vzdialenosť tým, že, pokiaľ sa to oplatí, vedie cestu medzi nimi cez niektorý ďalší vrchol. Keďže každá cesta




Obr. 7: **Bellman-Fordov algoritmus.** Nachádzame sa uprostred spracovávania a zlepšovania jednotlivých najlacnejších ciest. Aktuálne je spracovávaná žlto zafarbená hrana, ktorá v ďalšej chvíli upravi hodnotu spodného vrchola.

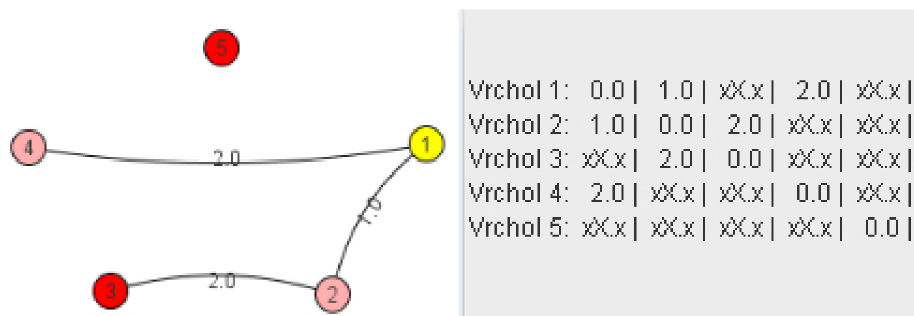
môže viesť maximálne cez všetky ostatné vrcholy, algoritmus naozaj dospeje do štádia, že pre každú dvojicu nájde optimálnu cestu. Tým vidíme skôr podobnosť so systematickým Bellman-Fordovym algoritmom než s "pažravým" Dijkstrovym algoritmom. Ďalšou podobnosťou je obmedzenie, ktoré sa týka len záporných cyklov. Na rozdiel od záporných hrán ako takých, ktoré algoritmu nevadia, záporný cyklus spôsobuje napríklad, že vzdialenosť vrcholu od samého seba sa stane po niekoľkých iteráciach záporná. To nám ale umožnuje tiež odhadliť prípadný záporný cyklus v grafe. Taktiež je možnosť konštrukcie konkrétnych ciest s pomocou metódy ukladania predchodcu.

Floyd-Warshallov algoritmus má očividnú časovú zložitosť, v závislosti od vstupného grafu $G = (V, E)$, práve $O(|V|^3)$. Tým vidíme zase, čo sa zložitosti týka, určitú podobnosť s Dijkstrovym algoritmom.

Pri implementácií sme už tradične namiesto $\infty$ použili `null`. Kvôli množstvu počítaných dát sme zvolili externé zobrazovanie priebežných výsledkov na paneli s popisom algoritmu. Tieto výsledky sú aktualizované podľa behu algoritmu. Farebné značenie nám taktiež indikuje, či sa najlacnejšia cesta medzi aktuálnou ružovo zvýraznenou dvojicou vrcholov dá zlacniť prechodom cez žlto zvýraznený vrchol. Pokiaľ áno, je táto skutočnosť indikovaná dlhším zvýraznením na oranžovo a následnou zmenou vypočítaných dát, pokiaľ nie, tak sú vrcholy na krátko zafarbené na tmavo. Ilustráciou behu algoritmu je Obr.8.

**Algoritmus 5** Floyd-Warshallov algoritmus

**procedure** Floyd-Warshall($G, z$)
  **for** $u = 1$ to $|V|$ **do**
    **for** $v = 1$ to $|V|$ **do**
      **if** $uv \in E$ **then**
        $P[u][v] \leftarrow$ Weight$(uv)$
      **else**
        $P[u][v] \leftarrow \infty$
  **for** $k = 1$ to $|V|$ **do**
    **for** $i = 1$ to $|V|$ **do**
      **for** $j = 1$ to $|V|$ **do**
        $P[i][j] = \min(P[i][j], P[i][k] + P[k][j])$




Obr. 8: **Floyd-Warshallov algoritmus.** Vpravo môžeme vidieť zatiaľ vypočítané hodnoty v prehľadnej tabuľkovej forme. Zatiaľ neexistujúce cesty sú výrazne vyznačené. Vľavo je vidno aktuálne spracovávaný “prechodový” vrchol, zafarbený žlto, a dvojicu vrcholov, ktorých vzdialenosť sa snažíme zlepšiť.

## 2.5 Kostra

[7]Táto kapitola bude riešiť problém nájdenia najlacnejšej kostry v danom hranovo-ohodnotenom grafe. V praxi je tento problém použiteľný napríklad pri potrebe spojiť viaceré mestá autobusovými linkami tak, aby sa dalo dostať z každého mesta do každého, ale zároveň aby náklady na beh týchto liniek boli čo najmenšie. Je preto dôležité, aby existovala práve jedna cesta medzi každými dvoma vrcholmi, čo je v skutočnosti priamo poukazujúce na problém hľadania najlacnejšej kostry.

[7]Naozaj sa nebudeme zaoberať pozostatkami, ako by sa mohlo na prvý pohľad zdať.

### 2.5.1 Kruskalov algoritmus

Tento problém vieme riešiť pomocou Kruskalovho algoritmu na hľadanie najlacnejšej kostry. Jeho základom je, že sa jedná opäť [8] o "pažravý" algoritmus. V každom okamihu pridáva do budúcej kostry najlacnejšiu hranu, ktorá spĺňa jedinú zásadnú podmienkou. Touto podmienkou je, aby po jej doplnení do kostry v kostre nevznikol cyklus. Pokiaľ by mal vzniknúť, hranu preskočíme. Tak isto sa dá veľmi triviálne skončiť aj v prípade, že je už v kostre $|V|$-1 hrán. Tak ako mnoho algoritmov, funguje Kruskalov algoritmus priamočiaro len v prípade, že ide o súvislý graf. V nesúvislom grafe je výstupom algoritmu les kostier jednotlivých komponentov súvislosti.

Dôležitou súčasťou behu algoritmu je tiež spôsob, akým je overovaná potenciálna prítomnosť cyklu v grafe po pridaní tej-ktorej hrany. V prípade Kruskalovho algoritmu je odporúčanou metódou Union-Find, s ktorou sme sa mali možnosť stretnúť už v rámci tejto kapitoly. Pre pripomenutie, jej princíp spočíva v tom, že si udržiavame množiny vrcholov, ktoré sú navzájom spojené, a vždy keď chceme niektoré dva nové vrcholy spojiť, skontrolujeme, či nie sú v tej istej množine. Efektivita tohto prístupu závisí na konkrétnej implementácií, bežné sú triviálna implementácia v $O(|V|)$, alebo tiež o niečo komplexnejšia v $O(\log|V|)$.

Kruskalov algoritmus má časovú zložitosť, v závislosti od vstupného grafu $G = (V, E)$ a od konkrétnej implementácie Union-Find, $O(|E|\log|E|)$ resp. $O(|E|\log|V|)$ [9] kvôli zložitosti triedenia hrán na začiatku algoritmu, ktorá je $O(|E|\log|E|)$. Táto zložitosť je dosiahnutá, pokiaľ zvolíme implementáciu Union-Find s pomocou porovnávania veľkostí jednotlivých množín pri optimalizácií spájania množín. V prípade vyššie spomínanej triviálnej implementácie bude výsledná implementácia približne $O(|E|.|V|)$

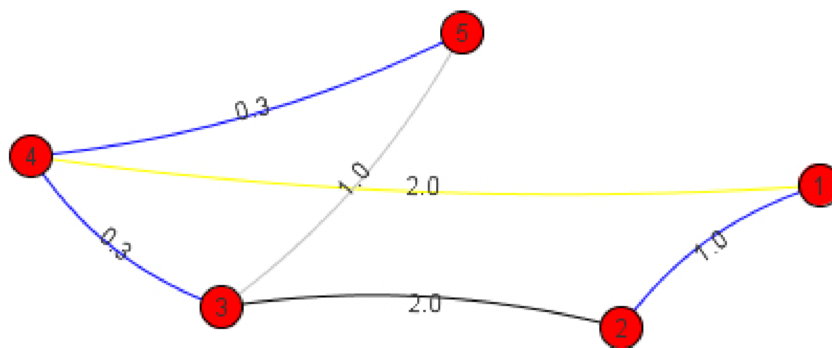
Čo sa implementácie týka, zvolili sme na usporiadanie hrán algoritmus, ktorý je poskytnutý v bežných knižniciach jazyka Java. Union-Find sme v našom prípade implementovali s použitím vyvažovania a teda sme dosiahli naozaj predpokladanú časovú zložitosť. Grafické znázornenie spočíva v tom, že v každom kroku sa vyznačí na žlto aktuálne najlacnejšia nespracovaná hrana, a nastane jeden z dvoch prípadov. V prvom sa prefarbí na modro, pokiaľ sme ju vložili do výslednej kostry, alebo na šedo, pokiaľ by vytvorila cyklus, a teda sme ju nepridali do kostry. Ilustráciou behu algoritmu je Obr.9.

[8] Podobne ako Dijkstrov algoritmus

[9] Dá sa dokázať, že ide o tú istú zložitosť.

**Algoritmus 6** Kruskalov algoritmus

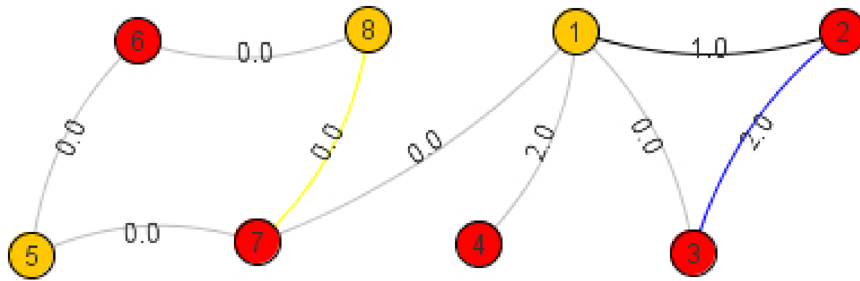
```
procedure KRUSKAL(G, z)
    for all e ∈ E do
        vytvor {e}
    usporiadaj hrany podľa ceny, neklesajúco
    T ← ∅
    while |T| < |V| − 1 do
        (u, v) ← najlacnejšia nepoužitá hrana
        kontrola, či pridanie (u, v) nevytvorí cyklus
        if find(u) ≠ find(v) then
            (u, v) → T
            union(find(u), find(v))
    return T
```




Obr. 9: **Kruskalov algoritmus** Tri hrany sú už vo výslednej kostre, jedna hrana je zamietnutá a ďalšia sa aktuálne spracováva. Vo výsledku bude táto hrana pridaná a doplní kompletne kostru, a zvyšná hrana bude zamietnutá.

## 2.6 Test bipartity

Posledným algoritmom, ktorým sa budeme zaoberať v tejto práci je algoritmus na testovanie bipartitnosti grafu. Vo svojej podstate ide o ten istý algoritmus, akým sa dá overiť tiež, či graf obsahuje nepárny cyklus. Na začiatku si vypočítame pre každý vrchol vzdialenosti od ľubovoľného (avšak pevne určeného) vrchola, a následne podľa parity tejto vzialenosti rozdelíme




Obr. 10: **Test bipartity.** Aktuálne vidíme väčšinu hrán skontrolovanú s pozitívnym výsledkom, nájdenú problematickú hranu zvýraznenú namodro, a tiež jednu nespracovanú a jednu aktívnu hranu.

vrcholy do dvoch množín. Pokiaľ je graf bipartitný, budú tieto množiny rovno tými, ktoré zodpovedajú definícií bipartity. Ak však obsahuje graf nepárny cyklus, stane sa, že niektoré dva susedné vrcholy budú mať rovnakú farbu. Preto prejdeme cez všetky hrany, a skontrolujeme koncové vrcholy každej z hrán. Časová zložitosť tejto časti algoritmu je $O(|E|)$, a teda celková zložitosť algoritmu závisí od konkrétnej metódy na zistenie parity vzdialenosti od niektorého vrchola.

**Algoritmus 7** Test bipartity

```
procedure Bipartity(G)
    for all v ∈ V do
        if parita vzdialenosti v of v_0 je 0 then
            označ v
    if ∃uv ∈ E, že u, v sú obe rovnako (ne)označené then
        graf nie je bipartitný
```

Implementácia v tomto prípade zahŕňa prevažne voľbu metódy na zistenie parity vzdialenosti od niektorého vrchola, aby sme sa vedeli pokúsiť o striedavé vyfarbenie vrcholov. V tomto prípade sme zvolili znovuvyužitie zjednodušeného algoritmu založeného na základe Bellman-Fordovho algoritmu. Následne je prechádzanie po jednotlivých hranách vizualizované cez zvýraznenie žltou farbou pre aktívnu hranu a následné zošednutie v prípade vyhovujúcej, alebo zmodrenie, v prípade nevyhovujúcej hrany. Ilustráciou behu algoritmu je Obr.10.

# 3 Implementácia aplikácie

V tejto kapitole sa budeme venovať samotnej implementácií aplikácie GraphViz, a teda implementácií všetkého "okolo" algoritmov. Tým máme na mysli prostredie v ktorom sa algoritmy vizualizujú, fungovanie samotnej aplikácie na vyšších úrovniach než samotné algoritmy, knižnice použité pri tvorbe tejto aplikácie, ako aj teoretické programátorské modely, ktoré stoja za týmto všetkým. V prvej podkapitole sa budeme venovať vizualizačnej časti, ktorá nie je hlavnou náplňou tejto práce, ale napriek tomu je jej neodmysliteľnou a nutnou súčasťou.

V druhej podkapitole sa zameriam na myšlienkový model a štruktúru, ktorá sa skrýva za celou aplikáciou. Systém, akým fungujú jednotlivé vrstvy aplikácie a distribúciu zodpovednosti medzi jednotlivé časti aplikácie. Na záver sa potom pozrieme na jednotlivé detaily a konkrétnu realizáciu predtým popísaného myšlienkového modelu ako aj na úskalia, ktoré nastali počas jeho implementácie.

## 3.1 Vizualizácia grafu

Základom každej vizualizačnej práce, či už bakalárskej, diplomovej, alebo akejkoľvek inej je práve samotná vizualizácia. Pod týmto pojmom chápeme spôsob, akým sú jednotlivé dáta, v našom prípade vrcholy, hrany, ich ohodnotenie a pod., vykresľované na obrazovku. Pre vizualizáciu grafov existuje samozrejme viacero prístupov. Rozlišujú sa optimalitou podľa rôznych kritérií. Naším cieľom ale nebolo zobrazovať graf optimálne, ale hlavne prehľadne, aby neprehľadnosť grafu nezhatila náš zámer vysvetliť jednoducho a pochopiteľne, čo sa s daným grafom práve deje.

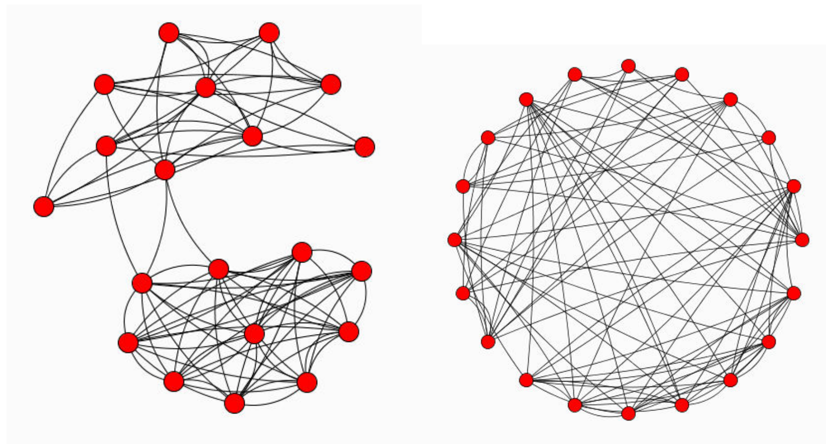
Vzhľadom na fakt, že cieľom tejto práce je zamerať sa na algoritmy a ich ozrejmenie čitateľovi a nie na rôzne možnosti a spôsoby samotnej vizualizácie grafov. Preto sa možnosť použitia niektorej už existujúcej knižnice či

frameworku ukázala ako optimálna. Stále však ostávalo otázkou, ktorú voľne dostupnú a použiteľnú knižnicu si vybrať. V tom pomohli aj nasledujúce kritériá:

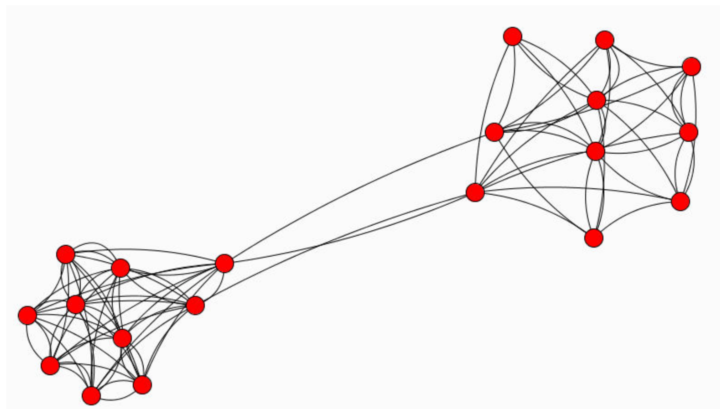
- Prispôsobiteľnosť vzhľadu grafu
- Prispôsobiteľnosť rozloženia vrcholov
- Rôzne implementované vizualizačné algoritmy
- Vhodná dátová štruktúra najnižšej úrovne
- Možnosť implementácie ohodnotených grafov
- Možnosť dynamickej zmeny grafu
- Aspoň aká-taká dokumentácia
- Čo najväčšia jednoduchosť, alebo zjednodušiteľnosť (možnosť vybrať si len niektoré podknižnice a zvyšné vynechať)
- Vhodná licencia pre voľné použitie

Tieto kritéria spĺňalo viac rôznych voľne dostupných knižníc, ale ako najpoužiteľnejšie sa zdali **Prefuse Visualization Toolkit**[8] a **JUNG - Java Universal Network/Graph Framework**[9]. Obe sa zdali spĺňať aspoň väčšinu z našich vyššie uvedených kritérií, ale pri podrobnejšom prezretí sa ukázalo, že Prefuse, napriek svojej väčšej variabilite, čo sa oblastí využitia týka, je až zbytočne zložitý a komplexný na naše účely. Ďalšou nevýhodou, ktorá sa prejavila, je nedostatok poriadnej dokumentácie, napriek tomu, že má internetové fóra, ktoré sú aktívne a je možnosť získať tam radu či tipy.

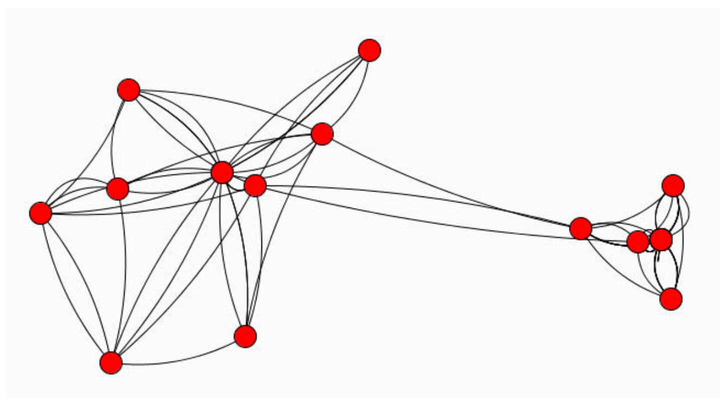
Ukázalo sa, že pre naše účely bude vhodnou voľbou framework JUNG, verzia 2.0, ktorý poskytuje nielen možnosť jednoducho a rýchlo implementovať prispôsobiteľnosť grafu myšou, ale tiež hneď niekoľko rôznych algoritmov na vykresľovanie. Samozrejme, tie sú len určitým odrazovým mostíkom, spôsobom, ktorým sa vykreslí graf na začiatku, predtým, ako si ho používateľ upraví podľa ľubovôle. Spomedzi viacerých implementovaných sme pre našu aplikáciu zvolili štyri, ktoré sa ukázali ako dostačujúco rozmanité. Presnejšie, ide o Kamada-Kawaiov algoritmus, Fruchterman-Rheingoldov algoritmus, Meyerov prístup založený na samo-usporiadavajúcich sa grafov a nakoniec jednoduché kruhové grafy, kedy sú vrcholy náhodne rozmiestnené na obvode kruhu. Ukážky týchto rozložení sú na Obr. 11.



(a) Kamada-Kawai (b) Kruhové rozloženie



(c) Fruchterman - Rheingold



(d) Meyerove samo-usporiadavajúce sa mapy

Obr. 11: Vybrané možné rozloženia grafu

Po zoznámení sa s dokumentáciou JUNGu [5] sa ukázalo, že síce taktiež obsahuje rozličné časti, ktoré nebudeme potrebovať, ale sú vhodne jednoducho oddeliteľné a teda zjednodušujú prácu s podstatnými časťami. Samozrejme je na mieste poznamenať, že v JUNGu sú implementované aj niektoré z nami zvolených algoritmov. Prečo sme teda implementovali znovu? Dôvodov je hneď niekoľko. Prvý je, že my sa chceme zamerať na názornosť, nielen na získanie výsledku. Ďalším je fakt, že sme chceli tiež možnosť krokovať si algoritmy v záujme lepšieho pochopenia toho, ako presne fungujú.

Ďalšou výhodou, ktorá sa pri podrobnejšej práci s JUNGom prejavila bola jednoduchá transformovateľnosť farebných schém jednotlivých vrcholov a hrán a tiež ich popisov, čo nám zjednodušilo okrem zvýrazňovania tiež zobrazovanie podstatných informácií. Priamo implementovaná vizualizácia v rôznych rozloženiach vrcholov a hrán ako i triviálne pridávanie, hľadanie a odoberanie vrcholov aj hrán z grafu vytvorili dobrý základ pre dátovú štrúktúru - graf - ktorú sme potrebovali. Na nej sme potom mohli budovať našu komplexnejšie reálne použité dátové štruktúry.

Napriek týmto mnohým prednostiam práce s JUNGom sa ukázali aj niektoré problémy pri práci s ním. Medzi hlavné patrila prehnaná "inteligencia" niektorých rozložení, ktorá spôsobovala, že si vizualizačná procedúra občas nevedela vybrať, aké umiestnenie vrcholov je práve to najvhodnejšie, čo spôsobovalo, takpovediac, roztrasenosť obrazu, kedy sa vrcholy pohybovali aj bez vonkajšieho pričinenia, zvyčajne medzi dvomi rôznymi polohami. Ďalším z problémov bolo počiatočné očakávanie, že ohodnotený graf bude priamo implementovaný v JUNGu. Ako sa ukázalo po hlbšom bádaní, v najnovšej verzií nebol, ale z dobrých dôvodov a tými je možnosť vytvorenia si dostatočne komplexného modelu vrcholu aj hrany, čo nám priamo ponúka možnosť vytvoriť si nielen jednoducho ohodnotené hrany, ale tiež hrany s viac než jedným parametrom, čo by bolo využiteľné napríklad pri tokoch.

Na záver dodám niekoľko príjemných prekvapení, ktoré priniesol JUNG v podobe mnohých ukážkových programov so zdrojovým kódom, ktoré často ozrejmovali použitie tých ktorých knižníc výrazne lepšie než samotná dokumentácia, nehovoriac o tom, že odhaľovali aké všetky možnosti nám JUNG poskytuje. Taktiež veľmi užitočným sa ukázal manuál [6], ktorý bol veľmi dobrým návodom ako začať s JUNGom.

## 3.2 Myšlienkový model

Každému programu a aplikácií musí predchádzať určitý myšlienkový model či predstava, inak by išlo o náhodné kombinácie syntakticky, a prípadne aj sémanticky správnych príkazov a nie o komplexný program. Tak isto aj za touto prácou sa nachádzala určitá myšlienková štruktúra. Jej základom samozrejme bol účel, pre ktorý táto aplikácia vzniká, čiže **odovzdanie informácií čitateľovi** resp. **používateľovi**, ktorý nemá ešte prax s teóriou grafov. Preto medzi prvé požiadavky na budúcu aplikáciu patrili nasledujúce:

- Krokovateľnosť rôznych algoritmov
- Možnosť pridávať ku krokom deskriptívne popisy toho, čo sa práve deje s grafom
- Možnosť preskočiť krokovanie a získať rovno výsledok, v prípade, že si chce používateľ overiť vlastný výsledok
- Konzistencia v ovládaní

Vďaka týmto kritériam mohla vzniknúť základná štruktúra aplikácie, pozostávajúca z hlavného, najväčšieho, panelu, kde sa zobrazuje samotný graf, bočného panelu so zobrazeným popisom a navigáciou pre používaný algoritmus a horného panelu s navigáciou a všetkými nastaveniami.

Ďalším kritériom, ktoré sa objavilo okamžite po nápade na vytvorenie takejto aplikácie bol fakt, že ide o vizualizačnú prácu a teda je potrebné zachovať **prehľadnosť a prispôsobiteľnosť** celého grafu. Vďaka tomu teda vznikli ďalšie požiadavky na výsledný program:

- Vyhnúť sa priamej implementácií vizualizácie, akožto tématike, ktorá je mimo zameranie tejto bakalárskej práce. Využiť teda niektorú na to vhodnú a uspôsobenú knižnicu, ktorá už existuje. [10]
- Dynamické farebné odlišovanie aktuálne menených dát
- Možnosť prispôsobenia vzhľadu grafu na globálnej aj lokálnej úrovni. Prevažne ide o posúvanie vrcholov podľa používateľovej ľubovôle a možnosť nastaviť rôzne počiatočné rozloženia vrcholov.

[10]tomuto bodu sme sa v podrobnej miere venovali v predošlej kapitole

Vhodný výber knižníc nám samozrejme vedel zabezpečiť všetky uvedené body, čo nám dovolilo sa zamerať na tretiu skupinu požiadaviek, ktoré poukazujú na prácu s grafom ako s **dátovou štruktúrou**, ktorá by teda mala byť schopná spracovať klasické množinové operácie, ako aj niekoľko našich ďalších požiadaviek z tejto oblasti:

- Možnosť pridať nový vrchol alebo hranu do grafu
- Možnosť odobrať existujúcu hranu alebo vrchol (a v prípade pokusu o odobratnie neexistujúcej neurobiť nič)
- Možnosť vytvorenia náhodného grafu, resp. pridania väčšieho počtu vrcholov a náhodných hrán do existujúceho grafu
- Existencia prednadstavených známych grafov

Po vytvorení tejto základnej sady požiadaviek na možnosti a schopnosti našej aplikácie je nutnosťou zamerať sa tiež na spôsoby, akými zabezpečiť aspoň istú mieru prehľadnosti samotného programu a jednoduchosti práce pri samotnej implementácií. Vďaka tomu sa ukázala príležitosť zamerať sa tiež na samotnú **štruktúru programu z hľadiska programátora** a nielen z hľadiska používateľa:

- Oddelenie našej rozšírenej a komplexnejšej dátovej štruktúry od hlavnej triedy aplikácie v rámci prehľadnosti.
- Výzor a navigáciu aplikácie prenechať podľa možnosti hlavnej triede.
- Ako najvnodnejšie riešenie samotných algoritmov sa ukázal návrhový vzor Strategy (v jemne upravenej implementačnej verzií), ktorý nám zabezpečí jednoduchosť používania a striedania algoritmov, keďže ich konkrétne implementácie budú o úroveň nižšie než časti aplikácie starajúce sa o ich beh.
- Jednoduchá rozšíriteľnosť o ďalšie algoritmy.
- Existencia samostatného vlákla pre obsluhu aktuálne bežiaceho algoritmu
- Implementácia krokovania, spracovania popisov algoritmov a pod. na vyššej úrovni - globálnejšie - než pri každom algoritme zvlášť.

Tým uzatvárame základný zoznam myšlienok, ktoré stáli za vznikom štruktúry programu, a ktoré zabezpečujú hlavnú časť plynulého behu aplikácie.

## 3.3 Implementačné detaily

Po spísaní a premyslení jednotlivých podmienok a kritérií, ktoré má spĺňať budúca aplikácia, prišla na rad jej implementácia. V rámci zmysluplného vytvárania jednotlivých častí aplikácie sme sa na jej časti ako na tri viac či menej nezávislé komponenty:

- Panel nástrojov a textu pre algoritmy - vpravo
- Panel nástrojov na úprava grafu - hore
- Vizualizačný panel pre graf

Obr. 12: Rozloženie komponentov v aplikácií

Intuícia nám hovorí, že logické poradie na tvorbu jednotlivých komponentov je opačné, a teda sme začali samotnou vizualizáciou grafu. Základom je trieda myGraph, ktorá je rozšírením základnej dátovej štruktúry poskytovanej JUNGom. Obsahuje okrem základných možností úpravy dát v grafe, taktiež všetky informácie nutné na jeho vykresľovanie. Vďaka možnosti použiť knižnice z JUNGu bola implementácia tejto triedy rýchla a bezproblémová, a preto sa zameriame radšej na zvyšné dva komponenty.

V poradí druhým na pláne bol panel na úpravu grafu. Ovládanie pre pridávanie a odoberania hrán a vrcholov funguje na princípe ActionListener-ov aplikovaných na príslušné tlačidlá. Okrem toho sem spadá možnosť nastavenia iniciálneho rozloženia vrcholov, ktorú zabezpečuje vyrolovateľný JComboBox, ktorý posunie vhodný parameter nášmu grafu. Ten sa prekreslí s novým nastavením, opäť automaticky vďaka JUNGu. Poslednou významnou časťou je možnosť nastavenia módu práce s grafou za pomoci myši. Ide o jednu z vecí, ktorá je priamo implementovaná v JUNGu, a teda zodpovedá nasledujúcim dvom riadkom programu:

```
JComboBox modeBox = graphMouse.getModeComboBox();
modeBox.addItemListener(graphMouse.getModeListener());
```

Do dokončenia základnej štruktúry a kostry aplikácie chýbala teda už len časť venovaná ovládaniu algoritmov. Pre začiatok, v rámci toho, aby sa dianie v grafe vykresľovalo priebežne, ukázalo sa použitie priame ActionListener-ov problematické, a bolo nutné siahnuť po vláknach. Zaviedli sme teda vlákno, ktoré sa staralo o sledovanie prípadnej zmeny požadovaného algoritmu a tiež o jeho vytvorenie a spustenie. To nám umožnilo aj počas behu algoritmu zmeniť názor a vybrať si iný algoritmus namiesto neho, čím sme pôvodný zrušili, a spustili nový.

Keď sa vnoríme hlbšie do štruktúry tohto komponentu narazíme na, v predošlej časti spomínaný, návrhový vzor Strategy, kompletne implementovaný v balíku `algorithms`. Jeho implementácia je jemne netypická, s ohľadom na fakt, že sme sa snažili o čo najjednoduchšiu možnosť úpravy a pridávania jednotlivých algoritmov (teda aby samotné triedy algoritmov boli čo najjednoduchšie). Základom sa teda stáva rozhranie `AlgorithmInterface`, ktoré obsahuje základné funkcie potrebné pre každý algoritmus. Jednu pre súvislé neanimované prebehnutie algoritmu, niekoľko pre krokovanie a jednu pre nastavovanie popisu k jednotlivým fázam algoritmu.

Avšak okolo týchto základných algoritmických funkcií potrebujeme, jednotných pre každý algoritmus, taktiež množstvo iných funkcií, ktoré sa postarajú o samotný beh a vizualizáciu algoritmu. Práve vďaka ich jednotnosti sme zvolili implementáciu, pri ktorej používame tiež triedu `Algorithm`, ktorá nám zjednocuje algoritmy a slúži ako určitý balík okolo nich. Všetky konkrétne implementácie algoritmov sú následne už len rozšírenia tejto triedy, ktoré implementujú len najnutnejšie funkcie (tie, ktoré sa nachádzajú tiež v rozhraní `AlgorithmInterface`). O všetko ostatné sa totiž postará trieda `Algorithm`, ktorej vlastnosti a funkcie zdedili. Preto tu máme nielen rozhranie, ale tiež

triedu, ktoré nám zjednocujú jednotlivé algoritmy, a teda nejde o celkom priamočiaru implementáciu Strategy, i keď myšlienka ostáva zachovaná.

Tým sme dokončili popis hlavnej štruktúry našej aplikácie a zvyšok je považovateľný naozaj za jednotlivé programátorské nuansy. Namiesto vymenovávania jednotlivých knižníc či tried použitých v aplikácií sa budeme venovať popisu ovládania aplikácie v praxy, čo bude pre náš čitateľa pravdepodobne prínosnejšie pri využívaní tejto aplikácie.

## 3.4 Ovládanie

Neskôr si ukážeme aj niekoľko ilustračných obrázkov zobrazujúcich aplikáciu, takpovediac, "v akcii", ale pre tento moment sa najprv pozrieme na rôzne možnosti ovládania.

**Úprava dát grafu:**

- Pridávanie vrcholov:
  - Priamym stlačením tlačidla `Add Vertex`
  - Uvedením čísla vrcholu, ktorý chceme do grafu pridať a následným stlačením tlačidla `Add Vertex`
- Pridávanie hrany - Uvedením čísel koncových vrcholov do prvých dvoch polí (pokiaľ chceme, tak ceny hrany do tretieho poľa) a následným stlačením tlačidla `Add Edge`.
- Odoberanie vrcholov:
  - Uvedením čísla vrcholu, ktorý chceme odobrať a stlačením tlačidla `Remove Vertex`.
  - Označením zvolených vrcholov a kliknutím na tlačidlo `Remove Selected`.
- Odoberanie hrán:
  - Uvedením čísel koncov hrany do prvých dvoch polí a následným stlačením tlačidla `Remove Edge`.
  - Označením vybraných hrán a následným kliknutím na tlačidlo `Remove Selected`.

- Hromadné pridávanie:
  - Uvedením požadovaného počtu nových vrcholov (ktorých poradové čísla budú vygenerované automaticky) do prvého poľa, čísla `0` do druhého poľa a stlačením tlačidla `Add Random Graph`
  - Uvedením požadovaného počtu nových hrán (ktorých koncové vrcholy budu vygenerované náhodne) do druhého poľa, čísla `0` do prvého poľa a stlačením tlačidla `Add Random Graph`
  - Hrán aj vrcholov - Uvedením požadovaného počtu nových vrcholov a hrán do prvého a druhého poľa a stlačením tlačidla `Add Random Graph`
- Resetovanie grafu - Ponechanie jediného izolovaného vrcholu v grafe dosiahneme stlačením tlačidla `Reset`

**Úprava vzhľadu grafu:**

- Voľba initiálneho rozloženia vrcholov v rolovateľnom výbere.
- Možnosť vypnúť/zapnúť popis hrán/vrcholov cez zaškrtávacie políčka.
- Možnosť výberu z predpripravených grafov. Výber v oblasti pre masové pridávanie.

**Úprava vzhľadu grafu myšou, mód `Picking`:**

- Kliknutie:
  - Na vrchol - označí práve daný vrchol
  - Mimo vrchola - odznačí všetky vrcholy
  - Na vrchol spoločne s tlačidlom `Shift` - označí alebo odznačí daný vrchol
  - Na vrchol spoločne s tlačidlom `Ctrl` - označí práve daný vrchol a vycentruje naň obrazovku
- Držanie ľavého tlačidla na myši a ťahanie:
  - Na vrchol - pohne všetkými označenými vrcholmi
  - Mimo vrcholu - označí vrcholy vo vyznačenom obdĺžniku
  - Spoločne s tlačidlom `Shift` - označí (pridá) všetky vrcholy vo vyznačenom obdĺžniku

- Scrollovanie - približuje alebo odďaluje graf

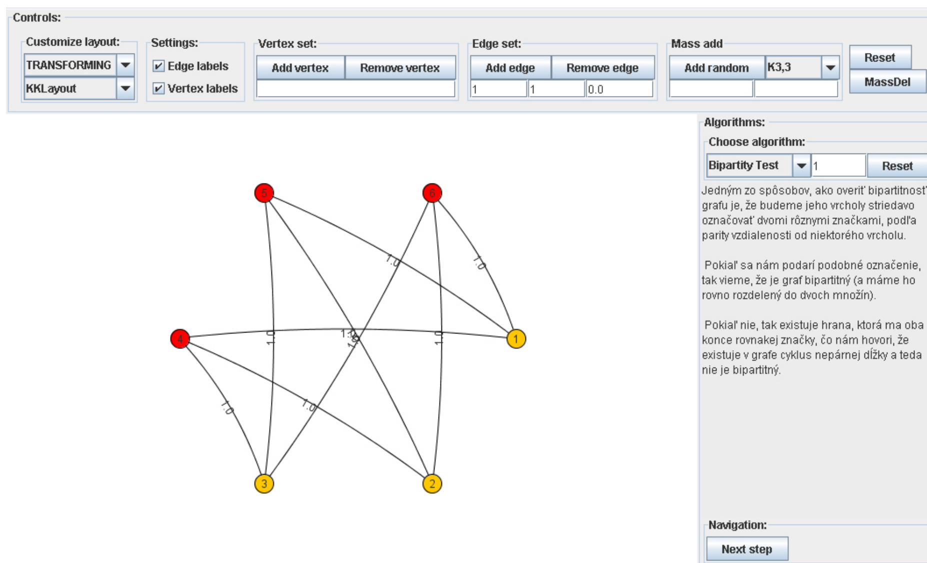
**Úprava vzhľadu grafu myšou, mód `Transforming`:**

- Držanie ľavého tlačidla na myši a ťahanie:
  - Samostatne - posúva graf
  - Spoločne s tlačidlom `Shift` - rotuje graf
  - Spoločne s tlačidlom `Ctrl` - naťahuje graf
- Scrollovanie - približuje alebo odďaluje graf

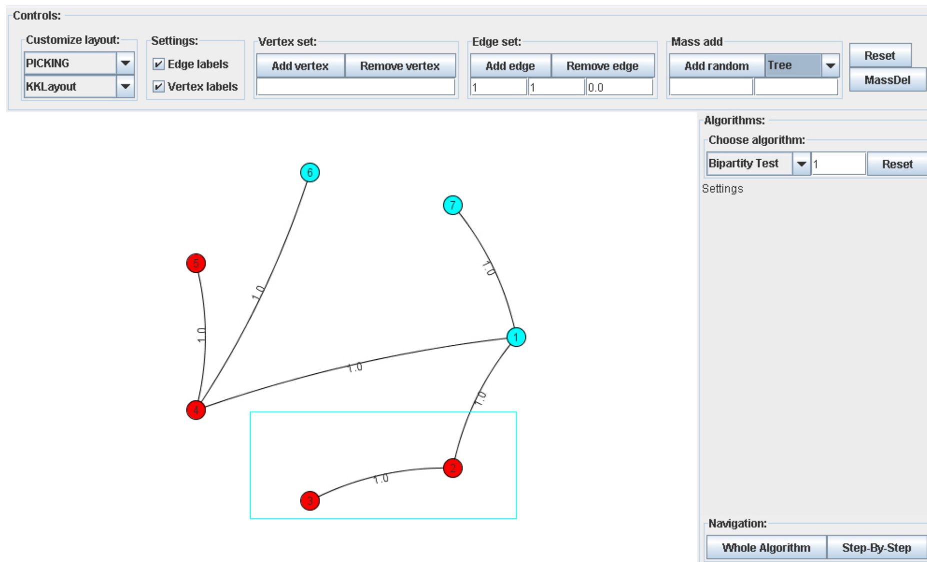
**Navigácia v rámci algoritmov**:

- Voľba algoritmu výberom z rolovateľného zoznamu a kliknutím na vybranú položku
- Voľba vrchola, z ktorého má byť algoritmus spustený, pokiaľ je to relavantné
- Prezretie si výsledku, ktorý algoritmus dosiahne - kliknutím na `Whole algorithm`
- Beh algoritmu krok po kroku:
  - Spustenie kliknutím na `Step-by-step`
  - Prejdenie na ďalší krok cez `Next step` resp. `Final step`
- Resetovanie behu algoritmu na stav pred jeho začatím - kliknutím na `Reset`

## 3.5 Ilustračné obrázky

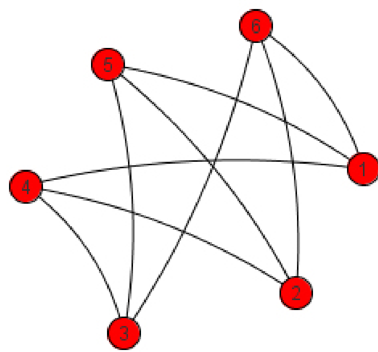



(a)

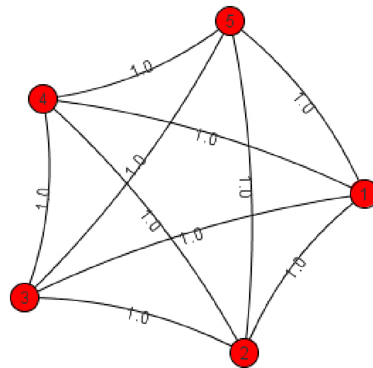



(b)

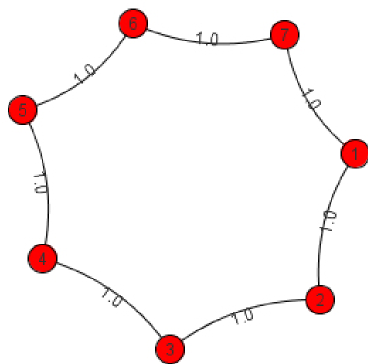
Obr. 13: Ukážky z celej aplikácie



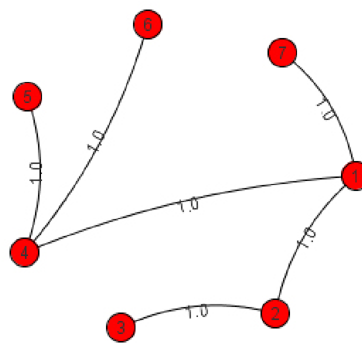

(a) $K_{3,3}$



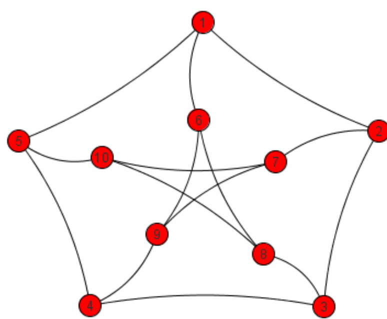

(b) $K_5$



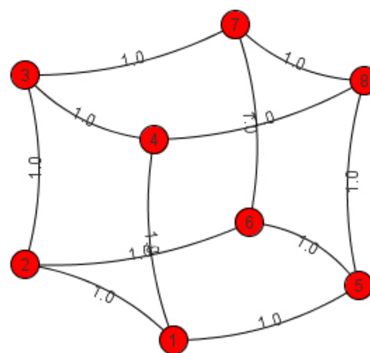

(c) Kružnica




(d) Strom




(e) Petersenov graf




(f) Kocka

Obr. 14: Predpripravené grafy

# Záver

V tejto práci sme najprv vysvetlili a zhrnuli základné definície a poznatky teórie grafov tak, aby boli sebestačným materiálom a aby boli vhodné pre úplného začiatočníka v tejto oblasti. Zachovali sme tiež konzistenciu a pokúsili sa o dostatočnú zrozumiteľnosť s minimálnymi predpokladanými znalosťami. V druhej kapitole sme sa zamerali na vysvetlenie konkrétnych algoritmov, ich časovej zložitosti a presného fungovania. Taktiež sme uviedli niektoré detaily implementácie týchto algoritmov, ako aj ilustračné obrázky priamo z aplikácie, ktorá je súčasťou tejto práce.

V tretej kapitole sme sa venovali najprv postupu akým bola naša aplikácia vytvorená. Po pôvodnom myšlienkovom modeli nasledovali konkrétne fakty ohľadne implementácie, a potom celá podkapitola venovaná ovládaniu aplikácie. Dôležitou časťou bola tiež kapitola venovaná metóde vizualizácie, o ktorú sa v tomto prípade staral voľne dostupný JUNG. Na záver sme uviedli tiež obrázky samotnej aplikácie, ktorú je možné nájsť okrem priloženého CD aj na internete, na webovej adrese: `http://people.ksp.sk/~allie/bakalarka`

Na prácu je možné nadviazať napríklad v oblasti pridávania ďalších, pokročilých algoritmov aj s ich vysvetlením (toky, párovania, a mnohé iné), prípadne pridávanie funkcionalít ako nahrávanie vytvoreného grafu, či dynamická možnosť pridávať nové algoritmy a prispievať k zdokonaľovaniu aplikácie.

Pevne veríme, že táto práca pomôže nielen študentom, ktorí začínajú s teóriu grafov, či už na niektorom z predmetov na našej fakulte, alebo mimo školských lavíc, ale aj tým, ktorí svoj záujem o informatiku ešte len rozvíjajú.

# Literatúra


[1] DIESTEL, R. 2010, Graph Theory, Fourth Edition, Springer-Verlag, Heidelberg, Graduate Texts in Mathematics, Volume 173, ISBN 978-3-642-14278-9

[2] CORMEN, T. H. – LEISERSON, Ch. E. – RIVEST, R. L. – STEIN C. 2009, Introduction to Algorithms, Third Edition, The MIT Press, ISBN 978-0-262-03384-8

[3] KOVÁČ, J. 2007, Vyhľadávacie stromy a ich vizualizácia, bakalárska práca pod vedením RNDr. J. Katreniakovej, PhD.

[4] GROSS, J. L. - YELLEN, J. 2005, Graph Theory and Its Applications, Second Edition, Chapman and Hall/CRC ISBN 978-1-584-88505-4

[5] O'MADADHAIN J. - FISHER D. - WHITE S. 2009, JUNG API Specification, http://jung.sourceforge.net/doc/api/

[6] BERNSTEIN G., 2009, JUNG 2.0 Tutorial http://www.grotto-networking.com/JUNG/JUNG2-Tutorial.pdf

[7] Oracle 1993-2011, Java$^{TM}$ Platform Standard Ed. 6 API Specification, http://download.oracle.com/javase/6/docs/api/

[8] Prefuse Visualization Toolkit, 13.4.2011 http://prefuse.org/

[9] Java Universal Network/Graph Framework, 17.4.2011 http://jung.sourceforge.net/

# Príloha A

K práci je priložené CD, na ktorom sa nachádza zdrojový kód aplikácie, sada knižníc JUNG, ako aj skompilovaný JAVA-applet a jednoduchá webová stránka na jeho spustenie.