Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# Vyhľadávanie spojení mestskou hromadnou dopravou

Bakalárska práca

2018
Alojz Stúpal

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

# Vyhľadávanie spojení mestskou hromadnou dopravou

Bakalárska práca

| | |
|---|---|
| Študijný program: | Informatika |
| Študijný odbor: | 2508 Informatika |
| Školiace pracovisko: | Katedra informatiky |
| Školiteľ: | RNDr. Michal Forišek, PhD. |

Bratislava, 2018

Alojz Stúpal

91450249

Univerzita Komenského v Bratislave
Fakulta matematiky, fyziky a informatiky

## ZADANIE ZÁVEREČNEJ PRÁCE

**Meno a priezvisko študenta:** Alojz Stúpal
**Študijný program:** informatika (Jednoodborové štúdium, bakalársky I. st., denná forma)
**Študijný odbor:** informatika
**Typ záverečnej práce:** bakalárska
**Jazyk záverečnej práce:** slovenský
**Sekundárny jazyk:** anglický

**Názov:** Vyhľadávanie spojení mestskou hromadnou dopravou
*Finding connections using public transport*

**Anotácia:** Práca sa zaoberá vyhľadávaním a zobrazovaním možností cesty MHD.

**Cieľ:** Práca má nasledujúce ciele:
1. Spracovať prehľad základných algoritmov vhodných na vyhľadávanie spojenia v grafikone MHD.
2. Implementovať softvér umožňujúci takéto vyhľadávanie a otestovať jeho funkčnosť na reálnych dátach.
3. Pokúsiť sa navrhnúť a implementovať možné vylepšenia, ako napr.: vyhľadávanie cesty bod-bod namiesto zastávka-zastávka; vizualizácia možných alternatívnych trás; výpočet dodatočných informácií, napr. očakávaného meškania v prípade nestihnutia prestupu.

**Vedúci:** RNDr. Michal Forišek, PhD.
**Katedra:** FMFI.KI - Katedra informatiky
**Vedúci katedry:** prof. RNDr. Martin Škoviera, PhD.

**Dátum zadania:** 07.11.2017

**Dátum schválenia:** 08.11.2017

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

.......................................................... ..........................................................

študent vedúci práce

**Poďakovanie:** Tu môžete poďakovať školiteľovi, prípadne ďalším osobám, ktoré vám s prácou nejako pomohli, poradili, poskytli dáta a podobne.

# Abstrakt

Slovenský abstrakt v rozsahu 100-500 slov, jeden odstavec. Abstrakt stručne sumarizuje výsledky práce. Mal by byť pochopiteľný pre bežného informatika. Nemal by teda využívať skratky, termíny alebo označenie zavedené v práci, okrem tých, ktoré sú všeobecne známe.

**Kľúčové slová:** jedno, druhé, tretie (prípadne štvrté, piate)

# Abstract

Abstract in the English language (translation of the abstract in the Slovak language).

**Keywords:**

# Obsah

# Zoznam obrázkov

# Zoznam tabuliek

# Úvod

Cieľom tejto práce je

# Kapitola 1

# Grafy

V tejto kapitole uvedieme zopár definícií týkajúcich sa grafov, opíšeme niektoré grafové štruktúry, prípadne zavedieme rôzne názvy, ktoré budeme v ďalších častiach práce potrebovať.

## 1.1 Definícia grafu

Začneme so základnou defínicou - s definovaním pojmu *graf*. Tú prevezmeme od Reinharda Diestela [4, kapitola 0.1]:

*Graf* je dvojica $G = (V, E)$ disjunktných množín, kde prvky $E$ sú dvojprvkové podmnožiny $V$. Prvky $V$ sú *vrcholy* (prípadne *uzly* alebo *body*) grafu $G$, prvky $E$ sú jeho hrany.

Vytvorený objekt by sa mal korektne nazývať *neorientovaný graf*. Avšak my, ako aj mnohí iní, od tohto pomenovania upustíme a budeme pre jednoduchosť používať len pojem *graf*.

Hranu $\{x, y\}$, kde $x, y \in V$, budeme zasa zvyčajne označovať aj ako $(x, y)$ alebo $xy$.

## 1.2 Základné pojmy týkajúce sa grafov

V nasledujúcej podkapitole sú uvedené definície rôznych pojmov, ktoré charakterizujú a popisujú vlastnosti grafových štruktúr.

*Rád grafu* definujeme ako počet vrcholov grafu $G$.

Budeme hovoriť, že vrchol $v$ je *incidentný* s hranou $e$, ak $v \in e$. Čiže, ak je vrchol

$v$ jedným z vrcholov, ktoré hrana $e$ spája, nazveme ho incidentným s tou hranou. Ak sú dva rôzne vrcholy incidentné s tou istou hranou, budeme ich volať jej *koncovými vrcholmi.*

Dva vrcholy $x, y \in V$ sú *susedné*, ak $xy$ je hrana v $G$. Dve hrany $e, f \in E; e \neq f$ sú *susedné*, ak majú spoločný vrchol.

*Stupňom vrchola* $v \in V$ nazveme počet hrán incidentných s $v$. Alebo, inak povedané, je to počet vrcholov, ktoré sú susedné s vrcholm $v$.

### 1.2.1 Typy grafov

Teraz uvedieme niektoré základné typy grafov.

*Prázdny graf* je graf $G = (\emptyset, \emptyset)$. Označenie zjednoduchšíme na $G = \emptyset$ .

Pojmom *Triviálny graf* budeme označovať graf s rádom 0 alebo 1.

*Kompletným grafom* nazveme taký graf, ktorého všetky vrcholy sú navzájom susedné. Uvažujme $n$ vrcholov, potom kompletný graf na týchto vrcholoch označíme $K_n$. Pre predstavu môže poslúžiť prííklad kompletného graf troch vrcholov $K_3$, čo je trojuholník.

Majme graf $G$. Ak každý jeho vrchol má rovnaký stupeň $k$, potom tento graf nazveme *k-regulárnym.* V definícií môžeme upustiť od počtu vrcholov a nazvať vzniknutý objekt iba *regulárnym.*

Nech $r \in N, r \geq 2$. Graf $G = (V, E)$ voláme *r-partitný*, ak V umožňuje rozklad na r tried (partícií), a to takých, že každá hrana má svoje konce v rôznych triedach. Alebo inak, vrcholy v tej istej triede rozkladu so sebou nesmú susediť. 2-partitným grafom budeme vravieť *bipartitné.* Ľahko si bipartitný graf predstaviť uvažujúc dve množiny vrcholov, pričom sa hrany grafu nachádzajú len a len medzi týmito množinami, nie v nich.

### 1.2.2 Vzťahy medzi grafmi

Nech $G = (V, E)$ a $G' = (V', E')$ sú grafy. Označíme $G \cap G' = (V \cap V', E \cap E')$ a $G \cup G' = (V \cup V', E \cup E')$. Ak prienik týchto dvoch grafov je prázdny graf $G \cap G' = \emptyset$, potom $G$ a $G'$ sú *disjunktné.* Ak $V' \subseteq V$ a $E' \subseteq E$, potom hovoríme, že $G'$ je podgraf

$G$ (alebo $G$ je nadgraf $G'$, či $G$ obsahuje $G'$) a píšeme $G' \subseteq G$.

## 1.3 Cesty a cykly

Nech $G = (V, E)$ je graf. Definujme *cestu*, ako neprázdny graf $P = (V', E')$, ktorý je podgrafom $G$, tvaru $V' = \{x_0, x_1, ..., x_k\}$ a $E' = \{x_0x_1, x_1x_2, ..., x_{k-1}x_k\}$, kde všetky $x_i$ sú navzájom rôzne. Neformálne povedané, cesta spája (pomocou hrán) dva vrcholy grafu $G$, pričom sa v nej použité vrcholy nesmú opakovať. Počet hrán cesty sa nazýva aj *dĺžkou*. Pre zjednoduchšenie budeme cestu zapisovať ako $P = x_0x_1x_2...x_{k-1}x_k$.

Majme graf $G = (V, E)$. Nech $P = (V', E')$ je cesta v $G$, $P = x_0x_1, x_1x_2, ..., x_{k-1}x_k$, pričom $k \geq 3$ a v grafe $G$ existuje hrana $x_0x_k$. Potom hovoríme, že graf $G$ obsahuje *cyklus* alebo tiež *kružnicu* $C$, kde $C$ je podgraf $G$ a platí $C = (V', E' \cup \{x_0, x_k\})$.

*Sled* v grafe $G$ je názov pre neprázdnu striedavú postupnosť $v_0e_0v_1e_1...e_{k-1}v_k$ vrcholov a hrán v G, pričom $e_i = v_iv_{i+1}$ pre všetky $i < k$. Sled teda, narozdiel od cesty, môže obsahovať rovnaké vrcholy viac krát. Preto sa dá povedať, že sled je cesta, v ktorej sa môžu vrcholy opakovať. Ale aj naopak: sled, v ktorom sú vrcholy navzájom rôzne, je cestou.

Sled, v ktorom sú hrany navzájom rôzne, sa nazýva *ťah*.

## 1.4 Súvislosť

Ak sú ľubovoľné dva vrcholy grafu $G$ spojené cestou, tento graf nazveme *súvislým*.

*Komponent grafu* $G$ je súvislý podgraf grafu $G$ taký, že nie je obsiahnutý v žiadnom väčšom súvislom podgrafe grafu $G$. Napríklad, súvislý graf má práve jeden komponent, a to samého seba.

Hrana $e \in E$ grafu $G$ sa volá *most*, ak graf $G$ má menší počet komponentov v porovnaní s grafom $G$ bez hrany $e$. Čiže, po odstránení hrany z grafu pribudne práve jeden komponent.

Vrchol $v \in V$ grafu $G$ sa nazýva *artikulácia*, ak počet komponentov grafu $G$ je menší ako ich počet po odstránení $v$ z grafu $G$. Teda, ak po odstránení vrchola z grafu pribudne aspoň jeden komponent.

## 1.5 Kostry, stromy a lesy

Nech $G = (V, E)$ je súvislý graf. *Kostrou* grafu nazveme taký graf $G' = (V', E'), G' \subseteq G$, pre ktorý platí $V' = V$ a navyše medzi každými dovma vrcholmi existuje práve jedna cesta. Ak by sme grafu $G$ postupne odnímali hrany tak, aby sme nenarušili jeho súvislosť, tak po odstránení všetkých takýchto hrán by sme získali kostru grafu $G$. Z tejto konštrukčnej definície vyplýva, že všetky hrany v kostre grafu sú mostami. Je z nej taktiež zrejmé, že každá kostra grafu je súvislá.

*Kostrový les* je taký graf $G$, pre ktorý platí, že každý z jeho komponentov je kostrou.

Acyklický graf (to jest taký, ktorý neobsahuje cyklus), ktorý je navyše súvislý, nazveme *stromom*. Všetky vrcholy stromu so stupňom 1 sú jeho *listy*. Je vhodné si povšimnúť, že, ako pri kostre, všetky hrani sú mostami. Taktiež je zaujímavé uvažovať o rozdiele medzi stromom a kostrou grafu. Možno nahliadnuť, že jediným rozdielom medzi nimi je, že strom je grafom, zakiaľ čo kostra je len podgrafom nejakého daného grafu.

*Les* je acyklický graf. Takže každý jeho komponent je stromom.

## 1.6 Špeciálne grafové štruktúry

Graf, ktorého prvky sú ohodnotené číslom určujúcim cenu, prípadne výhodnosť prechodu cezeň, nazveme *ohodnotený graf*. *Hranovo ohodnotený graf* je graf s funkciou $w : E \to R$. Teda pre ľubovoľnú hranu $e$ existuje číslo $w(e)$, ktoré nazveme *hodnotou hrany*, prípadne *cenou hrany*. Z praktického hľadiska je výhodné zadefinovať aj *kladne hranovo ohodnotený graf*. A to je taký, že $w(e) > 0$ pre všetky hrany. *Vrcholovo ohodnotený graf* je graf s funkciou $w : V \to R$. Teda pre ľubovoľný vrhol $v$ existuje číslo $w(v)$.

*Orientovaný graf* je dvojica $G = (V, E)$, kde každý prvok $xy \in E$ vyjadruje usporiadanú dvojicu. Neformálne, ak je daná hrana $xy$, ale nie $yx$, existuje cesta dĺžky 1 z vrcholu $x$ do $y$, ale z vrcholu $y$ do $x$ takáto cesta nejestvuje.

Rozdiel medzi orientovanými a neorientovanými grafmi je, ako názov napovedá v existencií orientácií hrán. Ak by sme túto skutočnosť chceli vyjadriť formálne, povedali by sme, že zatiaľ čo v neorientovanom grafe obsahuje množina $E$ dvojprvkové množiny, v orientovanom sa skladá z usporiadaných dvojíc.

### 1.6.1 Cesty a cykly pre orientované grafy

# Kapitola 2

# Algoritmy na grafoch

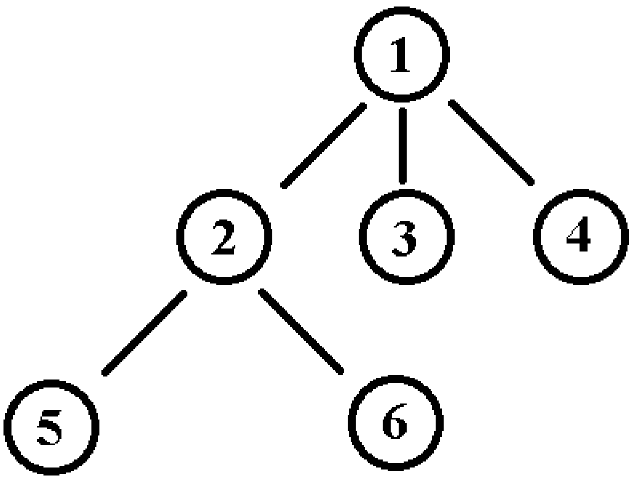
Táto kapitola obsahuje popisy algoritmov, ktoré sú aplikovateľné na grafové štruktúry a navyše využiteľné pri vyhľadávaní spojení hromadnej dopravy. Formálnu charakterizáciu algoritmu zväčša doprevádza i jeho voľnejšia interpretácia, ktorá má za účel uľahčiť porozumenie čitateľom, ktorí sa s daným algoritmom doposiaľ nestretli. Naviac, pri algoritmoch sú uvedené výhody, respektíve nedostatky, pri jeho použití na nami vytýčený cieľ. V hojnom počte budeme využívať definície z kapitoly 1.

## 2.1 Prechádzanie vrcholmi grafu

Stručne popíšeme základné algoritmy na prechádzanie vrholmi grafu. Tie však nie sú priveľmi využiteľné pri vyhľadávaní spojení MHD, môžu nám ale pomôcť k bezstarostnému prechádzaniu objemných súborov dát, ktoré bude sieť hromadnej dopravy isto obsahovať.

### 2.1.1 Prehľadávanie do šírky

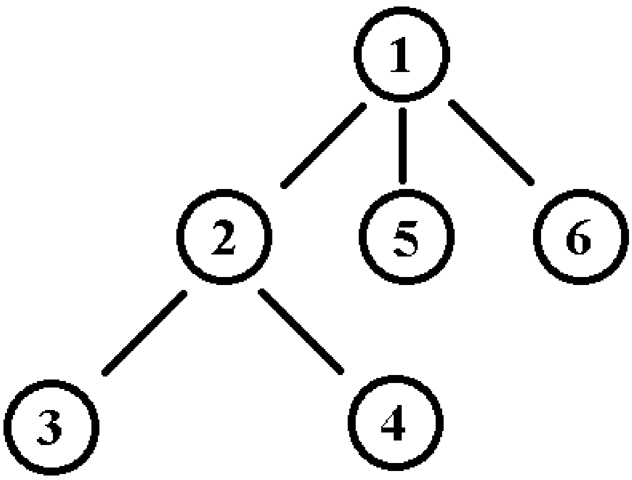
Prehľadávanie do šírky alebo skrátene BFS (z anglického Breadth-first search), dokáže pejsť cez všetky vrcholy grafu zapomoci fronty (alebo radu). Algoritmus si ukladá do nej ukladá susedov spracúvaných vrcholov, a tak prehľadá celý graf, pričom v každej iterácií prejde o jednu hranu viac vzdialené vrcholy od počiatočného.




Obr. 1: Príklad práce BFS na strome

### 2.1.2 Prehľadávanie do hĺbky

Prehľadávanie do hĺbky, skrátene DFS (z anglického Depth-first search), prechádza cez vrcholy grafu len pomocou cyklu, prípadne rekurzie. Obvykle sa v algoritme na všetkých susedov vrchola zavolá postupne rekuzívne ten istý algoritmus prehľadávania.




Obr. 2: Príklad práce DFS na strome

Je vhodné poznamenať, že na pbrázku 2 je ukázané len jedno z možných poradí práce s vrcholmi. Samozrejme, vrcholy budú vždy navštívené vo vyobrazenom poradí, avšak práca s nimi môže byť vykonaná až tromi spôsobmi. Prvý z nich súhlasí s usporiadaním na obrázku a nazýva sa *preorder*. Ďalšie dva sú *inorder* a *postorder*.

### 2.1.3 Zhrnutie

Obi dva algoritmy majú očividnú lineárnu časovú zložitosť ($O(n)$), nakoľko sa pozrú na každý vrchol grafu práve raz.

Ako sme už spomínali, v týchto algoritmoch nevidíme potenciál pri ich využití na uskutočnenie nášho cieľa. Ich znalosť však môže byť veľmi prínosná pri prechádzaní štruktúr našich grafov.

## 2.2 Najlacnejšie cesty v grafe

Pri vyhľadávaní spojení mestskou hromadnou dopravou sa zdá byť veľmi racionálne zaoberať sa nachádzaním najlacnejších ciest v grafikone MHD. Dovoľujeme si tak tvrdiť, pretože najväčší dôraz cestujúcich je kladený práve na čo najskorší príchod do požadovanej lokality. Pre úplnosť ešte dodávame, že cena cesty, ako je asi zrejmé, je súčet hodnôt hrán (respektíve vrcholov), ktoré daná cesta obsahuje. Z doposiaľ uvedeného vyplýva, že pri našich úvahách budeme používať ohodnotené grafy, kde pridelenou hodnotou bude čas medzi zastávkami. Navyše musíme nejako ošetriť i vrcholy grafu - zastávky, keďže na nich zvyčajne čakáme na prestup na ďalší spoj. Od tejto myšlienky však spočiatku upustíme a budeme sa jej venovať až neskôr. A posledná úvaha - hranami v našom grafe sú linky MHD, sme preto nútení použiť orientované grafy.

Ak uvažujeme vyhľadávanie najlacnejších ciest v grafe, musíme si najprv uvedomiť, čo presne je našim cieľom. Výledok, ktorý chceme dosiahnuť ako výstp algoritmu, má byť najlacnejšia cesta z počiatočného bodu do koncového. Avšak znalosť problematiky algoritmov na grafových štruktúrach nám ponúka riešenia iného, koplexnejšieho problému s asymptoticky rovnakou časovou zložitosťou. Týmto problémom je vyhľadávanie najlacnejších ciest z počiatočného vrcholu do všetkých ostatných vrcholov. Ľahko nahliadnuť, že riešeim tohto problému dostaneme odpoveď aj na našu počiatočnú otázku. Preto sa v nasledujúcej časti budeme zaoberať algoritmami riešiacimi túto úlohu.

Ľahko spozorovať, že by mohlo byť v určitých situáciách výhodné vypočítať ceny a nájsť najlacnejšie cesty pre všetky dvojice vrcholov. Hlavne, keď graf obsahuje malé množstvo vrcholov - zastávok. Mnoho miest má len malú sieť mestskej hromadnej dopravy, a teda by bolo rozumné vypočítať všetky potrebné údaje naraz na začiatku, a potom, pri prijímaní dotazu na vyhľadanie spojenia, jednoducho vypísať už vypočítanú odpoveď. Z tohto dôvodu uvedieme i algoritmy riešiace tento problém.

### 2.2.1 Dijkstrov algoritmus

Holandský informatik, po ktorom je tento algoritmus pomenovaný, dokázal, okrem iného, vyriešiť aj nami nastolený problém. V jeho riešení je ale potrebné, aby boli ceny hrán grafu nezáporné reálne čísla, čo súhlasí s našou predstavou aplikovania algoritmu na grafikon MDH. Algoritmus je navrhnutý tak, že dostane ako vstup graf $G = (V, E)$, počiatočný vrchol $v_0$ a hodnotiacu funkciu $h : V \times V \to R_0^+$. Predpokladáme, že ak hrana $uv \notin E$, potom $h(u, v) = \infty$, ďalej, že $h(u, u) = 0$ a nakoniec, že funkciu $h$ je možné vypočítať v konštantnom čase $O(1)$. Máme taktiež jeden predpoklad na vrcholy grafu, a to, že sú reprezentované celými číslami $1, 2, ..., k$ (kde $k$ je počet vrcholov grafu). Nakoniec, výsledkom algoritmu bude pole čísel $D$, v ktorom bude pre každý vrchol $v \in V$ vypočítaná hodnota $D[v]$, čo je cena najlacnejšej cesty z počiatočného vrchola $v_0$ do vrchola $v$.

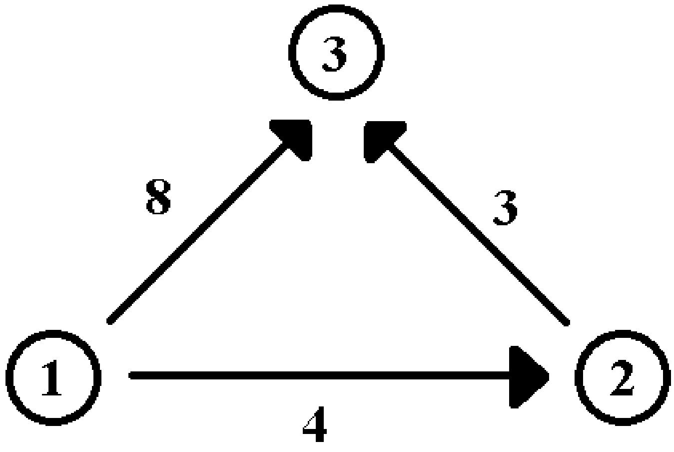
Tieto požiadavky, predpoklady, ba i vstup a výstup funkcie sú uvažované v horeuvedenom tvare len aby sme mohli predviesť implementáciu algoritmu od Pavla Ďuriša, ktorú možno nájsť aj v jeho knihe [10, kapitola 2.2.1].

**begin**
- $S \leftarrow \{v_0\}$
- $D[v_0] \leftarrow 0$
- *pre každý vrchol* $v \in V \setminus \{v_0\}$*:* $D[v] \leftarrow h(v_0, v)$
- **while** $S \neq V$ **do begin**
  - *vyber* $w \in V \setminus S$ *taký, že hodnota* $D[w]$ *je minimálna*
  - $S \leftarrow S \cup \{w\}$
  - *pre každý* $v \in V \setminus S$*:*
    - $D[v] \leftarrow \min\{D[v], D[w] + h(w, v)\}$
- **end**

**end**

Algoritmus 1: Dijkstrov algoritmus

Časová zložitosť dijkstrovho algoritmu je $O(|V|^2)$, čo je dokázané spolu s jeho korektnosťou v už uvedenom zdroji od Pavla Ďuriša [10, kapitola 2.2.1].




Obr. 3: Orientovaný graf s ohodnotenými hranami

Priebeh dijkstrovho algoritmu vysvetlíme na jednoduchom príklade. Máme orientovaný graf ako je zobrazený na obrázku 3. Našou úlohou je nájsť najkratšiu cestu z vrcholu 1 do ostatných vrcholov. Učiníme tak teda zapomoci dijkstrovho algoritmu. Čiže, najprv si vytvoríme pole, označme ho $D$, v ktorom si pre každý vrchol budeme pamätať hodnotu doň najkratšej cesty z vrcholu 1. Na počiatku bude táto hodnota pre všetky vrhcoly nekonečno, respektíve nejaká jeho rozumná náhrada. My budeme používať nekonečno. Nastavíme $D[1] = 0$. Začíname teda s počiatočným vrcholom 1. Zoberieme jeho všetkých zatiaľ nenavštívených susedov (vrcholy 2 a 3), a porovnáme hodnoty takto: najprv pre vrchol 2, zistíme či je jeho hodnota v poli $D$ (to jest nekonečno) menšia ako hodnota vrchola, z ktorého vychádzame, plus hodnota hrany (to jest $0 + 4$). To ale nie je pravda, takže prepíšeme hodnotu $D[2]$ na 4. Pre vrchol 3 zasa porovnávame nekonečno s hodnotou $0 + 8$ alebo aj, ako je v algoritme 1 naznačené, minimum z týchto dvoch hodnôt zapíšeme do $D[3]$. Vrchol 1 nemá viac susedov, takže ho označíme ako navštívený a pokročíme. Ďalším vrcholom v poradí bude ten s najmenšou hodnotou $D[i]$ spomedzi doposiaľ nenavštívených, čiže vrchol 2. Ten má iba jedného ešte nenavštíveného suseda, a to vrchol 3. Do $D[2]$ preto zapíšeme minimum z hodnôt 8 a $4 + 3$. Vrchol 2 je týmto vybavený a označený za navštívený. Zostáva posledný vrchol - vrchol 3. Väčšina implementácií dijkstrovho algoritmu pri dosiahnutý finálneho vrcholu končí v snahe urýchliť vyhľadávanie medzi dvoma vrcholmi. Ak však túto podmienku nezakomponujeme do nášho algoritmu, program zráta hodnoty najkratších ciest z daného vrcholu do všetkých ostatných vrcholov. V našom prípade nie je veľmi podstatné, ktorú variantu zvolíme, keďže algoritmus tak či tak vo vrchole 3 skončí, nakoľko ten už nemá žiadne incidentné hrany a v grafe už nejestvujú ďalšie nenavštívené vrcholy. Dosiahli sme teda výsledok: najkratšie cesty z vrcholu 1 sú uložené v poli $D$. Najkratšia cesta do vrcholu 1 je $D[1] = 0$, do vrcholu 2 je $D[2] = 4$ a pre vrchol 3 je odpoveď $D[3] = 7$.

Týmto príkladom sme chceli jemne ozrejmiť postup pracovania algoritmu, keďže pre tých, čo sa s ním ešte nestretli, ho môže byť dosť obtiažne z pseudokódu 1 vybadať.

Keďže je našim cieľom nie ani tak zistiť cenu najlacnejšej cesty, čo je hlavnou úlohou popísaného algoritmu, ale skôr takéto cesty nájsť, dijkstrov algoritmus by sme potrebovali jemne modifikovať, a to tak, aby sme vedeli spätne zrekonštruovať nájdené cesty. Teda so znalosťou konečného vrchola by sme chceli vedieť vygenerovať postupnosť vrcholov, cez ktoré sme sa do finálneho vrchola dostali. Budeme si preto pamätať pri každom vrchole informáciu, z ktorého vrchola sme sa doň dostali.

Dijkstrov algoritmus sa zdá byť najvhodnejším kandidátom pre účely našej práce. Jeho časová zložitosť je príjemná, implementácia nenánorčná a je dosť ľahké sa v nej orientovať, takže si ju budeme môcť poľahky modifikovať, prispôsobiť našim potrebám. Budeme teda schopný vypísať dodatočné informácie pre používateľov aplikácie, prípadne vyhľadávanie spresniť či rozšíriť podľa ich potrieb.

### 2.2.2 Bellman–Fordov algoritmus

Ďalším algoritmom na výpočet najlacnejších ciest v grafe je Bellman–Frodov algoritmus. Hoci ako prvý prišiel s týmto riešením Alfonso Shimbel v roku 1955, algoritmus bol pomenovaný po Fordovi, ktorý ho publikoval v roku 1956 a po Bellmanovi, ktorého publikácia je z roku 1958. Ten istý algoritmus zverejnil i Moore v roku 1957, a preto niekedy môžeme počuť aj o Bellman–Ford–Moorovom algoritme.

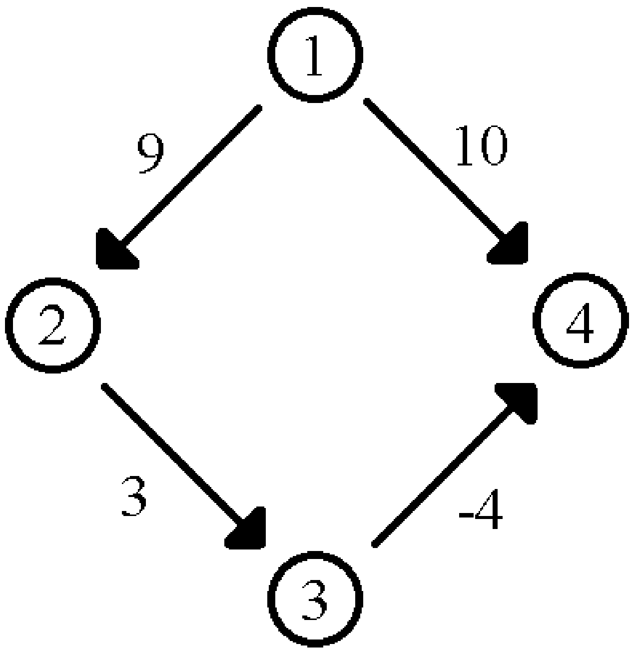
Samotný algoritmus, narozdiel od dijkstrovho, dokáže pracovať i so záporne ohodnotenými hranami. Avšak predpokladá sa, že graf neobsahuje záporne cykly, to jest cykly, ktorých celková cena má zápornú hodnotu. Ak by v grafe existoval aspoň jeden, mohli by sme po ňom prejsť ľubovoľne veľa krát, čím by sme ustavične znížovali najlacnejšiu cenu do viacerých vrcholov. Bellman–Fordov algoritmus je však navrhnutý tak, že zakaždým vykoná konečný počet krokov, takže i pri grafe obsahujúcom záporný cyklus skončí, nezacyklí sa, iba jeho výsledné hodnoty nebudú pravdivé. Navyše, mnohé implementácie zahŕňajú i detekciu takéhoto cyklu a pri jeho nájdení vyhlásia chybu. Naša implementácia takúto kontrolu obsahovať nebude, nakoľko nie je potrebná pre vysvetlenie funkcionality a fungovanie algoritmu.

Náš algoritmus predpokladá, že na vstupe dostane graf $G = (V, E)$ neobsahujúci záporné cykly, hodnotiacu funkciu $h : V \times V \to R$ a počiatočný vrchol $v_0$. Ďalšie predpoklady sú, ako u dijkstru, že ak $uv \notin E$, potom $h(u,v) = \infty$, taktiež $h(u,u) = 0$ a napokon, že funkciu $h$ je možné vypočítať v čase $O(1)$. Aby sa nám ľahšie pracovalo, opäť povedzme, že vrcholy na vstupe musia byť reprezentované celými číslami $1, 2, ..., n$, kde $n$ je počet vrcholov grafu. Našim výsledkom bude pole čísel $D$, v ktorom bude pre každý vrchol $v \in V$ vypočítaná cena najlacnejšej cesty z počiatočného vrchola $v_0$ do $v$, označená $D[v]$. Nasledovná implementácia je zostavená s pomocou knihy od Bannistera a Eppsteina [2].

$D[v_0] \leftarrow 0$
*pre každý vrchol* $v \in V \setminus \{v_0\}$: $D[v] \leftarrow \infty$
**for** $i \leftarrow 1$ to $n-1$ **do**
  **for** each edge $wv$ in $G$ **do**
    $D[v] \leftarrow \min\{D[v], D[w] + h(w,v)\}$

Algoritmus 2: Bellman–Fordov algoritmus

Časová zložitosť algoritmu je očividne $O(|V| \cdot |E|)$. Jeho korektnosť je možné nájsť napríklad v knihe [1, kapitola 3.3.4]. Toto dielo navyše obsahuje aj implementáciu spomínaného drobného vylepšenia, a to zisťovanie prítomnosti záporného cyklu.




Obr. 4: Orientovaný graf s ohodnotenými hranami

Funkcionalitu algoritmu si opäť predvedieme na jednoduchom príklade. Nech je vstupným grafom algoritmu graf vyobrazený na obrázku 4. Naším cieľom bude nájsť najkratšiu cestu z vrcholu 1 do ostatných vrcholov. Pod týmto popisom sa nachádza aj tabuľka (tabuľka 1) ukazujúca kľúčové hodnoty počas vykonávania algoritmu na našom príklade. Iterácia číslo 0 zaznamenáva stav vrcholov po inicializácií a pred začatím vykonávania cyklu, preto má počiatočný vrchol v tomto momente hodnotu $D[1] = 0$ a všetky ostatné $D[j] = \infty$. Počas iterácie $i = 1$ sa ideme pozerať na všetky hrany v grafe. Tu zavisí od ich usporiadania, ktorá z nich sa vezme v akom poradí. V našom príklade, aby sme demonštrovali čo najviac možností, povedzme, že ako prvú určíme na porovnávanie hranu medzi vrcholmi 2 a 3. Keďže je ale hodnota $D[2]$ aj $D[3]$ rovná nekonečnu, zapíšeme do $D[3]$ opäť nekonečno. Nech je ďalšou hranou v poradí vyhodnocovania hrana medzi vrcholmi 1 a 2. Tu je už situácia príznačnejšia. Nakoľko poznáme hodnotu $D[1]$, môžeme zaznačiť, že $D[2] = 9$. Pozorné oko si isto všimne, čo by sa stalo, ak by boli vyhodnocovania týchto dvoch hrán v opačnom poradí. Totiž, v takom prípade by sme ihneď vedeli vypočítať i hodnotu $D[3]$. Pokračujme hranou medzi vrcholmi 1 a 4. Zisťujeme, že $D[4] = 10$. Hrana medzi vrcholmi 3 a 4 dovŕši prvú iteráciu algortmu. Tu sa ale nič nezmení, keďže porovnávame nekonečno s hodnotou 10. V nasledujúcej iterácií $i = 2$ sa pre hrany spájajúce vrcholy $1, 2$ a $1, 4$ nestane nič zaujímavé. Pozrime sa teda na zvyšné hrany. Pre hranu medzi vrcholmi 2 a 3 sa upraví hodnota $D[3]$ na 12 a následne sa pre poslednú hranu (medzi vrcholmi 3 a 4) vykoná porovnanie hodnôt $(D[4] = 10)$ a $((D[3] = 12) + (-4))$. Po druhej iterácií sme už dospeli k optimálnemu riešeniu, čo ale náš algoritmus nemôže vedieť, a preto vykoná i poslednú, tretiu iteráciu. V nej sa ale už nič nezmení. Ak v dvoch stavoch po sebe nastala rovnaká situácia, repsektíve neprišlo k žiadnej zmene, je jasné, že algoritmus dosiahol výsledok. Ak by teda bola na pláne ešte ďalšia iterácia či nebodaj viac, môžeme vykonávanie programu ukončiť bez strachu pre nesprávny výsledok. Mnohé implementácie sú preto väčšinou doplnené o toto jemné vylepšenie. V našom príklade by nám však nepomohlo, keďže by program tak či tak zastavil po tretej iterácií.

Tabuľka 1: Priebeh algoritmu

| iterácia = i | D[1] | D[2] | D[3] | D[4] |
|---|---|---|---|---|
| 0 | 0 | $\infty$ | $\infty$ | $\infty$ |
| 1 | 0 | 9 | $\infty$ | 10 |
| 2 | 0 | 9 | 12 | 8 |
| 3 | 0 | 9 | 12 | 8 |

V prípade Bellman–Fordovho algoritmu je nám opäť na obtiaž jeho strohá funkcionalita, a teda zisťovanie cien najlacnejších ciest, zatiaľ čo my žiadame poznať celú nájdenú cestu. Modifikácia kódu je ale jednoduchá, rovnaká ako pre dijkstru - pre každý vrchol si navyše zapamätáme, z ktorého vrcholu sme sa doň dostali.

Bellman–Fordov algoritmus má mnoho pozitív - aplikovateľnosť i na hrany so záporným ohodnotením. Jeho časová zložitosť je taktiež potešujúca a implementácia prehľadná. Avšak jeho funkčnosť i nad zápornými ohodnoteniami nám nijako nepomôže pri jeho aplikácií na grafikon hromadnej dopravy. Navyše, jeho časová zložitosť $O(|V| \cdot |E|)$ bude pri riešení nášho problému iba príťažou, nakoľko predpokladáme, že sieť liniek MHD bude obsahovať veľké množstvo hrán. Netreba však zabúdať, že algoritmus je pri iných problémoch široko využiteľný a výborný pre dobre zvolené dátové štruktúry.

### 2.2.3 Floyd–Warshallov algoritmus

Posledným algortimom slúžiaci na hľadanie najlacnejších ceist, ktorý uvedieme, nesie názov Floyd–Warshallov algoritmus. Opäť, ako v predošlom prípade, sa ale ani Floyd, ani Warshall nemôžu pýšiť prvenstvom. Predbehol ich Roy publikáciou z roku 1959, zatiaľ čo Floydova i Warshallova boli obe z roku 1962. Warshall dokonca v jeho práci nadväzuje na Kleeneho algoritmus z roku 1956. Finálna verzia, ktorú aj mi predvedieme, dostala svoju podobu až na konci roku 1962 zásluhou Ingermana. Vďaka mnohým publikáciám sa tento algoritmus nazýva rôznymi menami. My si ale vystačíme s jedným.

Výsledok Floyd–Warshallovho algoritmu sa značne odlišuje od predošlých dvoch. Zatiaľ čo tie spoľahlivo zistili hodnoty najlacnejších ciest z daného vrcholu do všetkých ostatných vrcholov, Floyd–Warshallov ich vyráta z každého vrcholu do každého. Samozrejme, to by sme vedeli dosiahnuť aj použitím predošlých algoritmov na všetkych vrcholoch. Vyžadovalo by sa však mnoho réžie navyše a kód by sa pravepodobne priveľmi zneprehľadnil. Naopak, Floyd–Warshallov algoritmus je na pohľad až príliš jednoduchý, no nesmierne účinný.

Nami uvedená implementácia bude pracovať s maticami. Na vstupe dostane algoritmus graf $G = (V, E)$ bez záporných cyklov a incidenčnú maticu $W = (w_{ij})$ rozmerov $|V| \times |V|$. Maticu $W$ si vieme jednoducho zostrojiť z hodnotiacej funkcie takto: Nech sú ceny hrán hodnotené hodnotiacou funkciou $h : V \times V \to R$, kde, ako obvykle, $h(u, u) = 0$, $uv \notin E \implies h(u, v) = \infty$ a $h(u, v) \in O(1)$. Potom $w_{ij} = h(i, j)$ pre všetky $i, j$. Príklad takejto matice $W$ je vyobrazený na 2.1. Výsledkom algoritmu je posledná matica, a tou je matica $C^{(|V|)}$.

**begin**

$n \leftarrow |V|$

$C^{(0)} \leftarrow W$

**for** $k \leftarrow 1$ **to** $n$ **do**

  **for** $i \leftarrow 1$ **to** $n$ **do**

    **for** $j \leftarrow 1$ **to** $n$ **do**

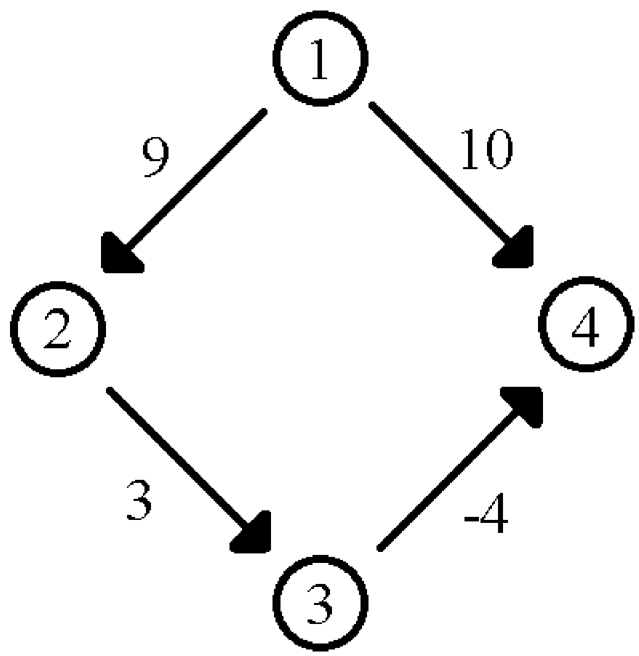
      $c_{ij}^{(k)} \leftarrow MIN(c_{ij}^{(k-1)}, c_{ik}^{(k-1)} + c_{kj}^{(k-1)})$

**return** $C^{(n)}$

**end**

Algoritmus 3: Floyd–Washallov algoritmus

Ľahko nahliadnuť, že časová zložitosť Floyd–Warshallovho algoritmu je $O(|V|^3)$. Jeho korektnosť je taktiež pohľadom badateľná. Pre istotu pridávame možnosť overiť si naše tvrdenie v knihe od Pavla Ďuriša, odkiaľ pochádza aj implementácia algoritmu [10, kapitola 2.2.2].




Obr. 5: Orientovaný graf s ohodnotenými hranami

Ako príklad nám poslúži ten istý graf ako v prípade Bellman–Fordovho algoritmu. Avšak výpočet algoritmu by bol na popis pridlhý, preto uvedieme iba zopár vyhodnocovaní a zaujímavé idey, ktoré je vhodné si uvedomiť. Prvý cyklus algoritmu, pracujúci s premennou $k$, vytvorí zakaždým celú novú maticu $C^{(k)}$. Pre každý riadok a stĺpec (premenné $i$ a $j$) tejto novovznikajúcej matice sa vykoná porovnanie, ktoré vypočíta hodnotu momentálneho prvku. Napríklad, ak by sme chceli zrátať hodnotu v matici $C^{(1)}$ na pozícií $i = 2, j = 2$, s ľahkosťou využijeme vzorec použitý v algoritme, a teda $c^{(1)}_{2,2} = MIN(c^{(0)}_{2,2}, c^{(0)}_{2,1} + c^{(0)}_{1,2})$, po dosadení $c^{(1)}_{2,2} = MIN(0, \infty + 9) = 0$. Lepším príkladom môže byť zmena hodnoty v po sebe nasledujúcich maticiach, ako napríklad $c^{(3)}_{2,4} = MIN(c^{(2)}_{2,4}, c^{(2)}_{2,3} + c^{(2)}_{3,4}) = MIN(\infty, 3 + (-4)) = -1$.

$$C^{(0)} = W = \begin{pmatrix} 0 & 9 & \infty & 10 \\ \infty & 0 & 3 & \infty \\ \infty & \infty & 0 & -4 \\ \infty & \infty & \infty & 0 \end{pmatrix} \tag{2.1}$$

$$C^{(1)} = \begin{pmatrix} 0 & 9 & \infty & 10 \\ \infty & 0 & 3 & \infty \\ \infty & \infty & 0 & -4 \\ \infty & \infty & \infty & 0 \end{pmatrix} \tag{2.2}$$

$$C^{(2)} = \begin{pmatrix} 0 & 9 & 12 & 10 \\ \infty & 0 & 3 & \infty \\ \infty & \infty & 0 & -4 \\ \infty & \infty & \infty & 0 \end{pmatrix} \tag{2.3}$$

$$C^{(3)} = \begin{pmatrix} 0 & 9 & 12 & 8 \\ \infty & 0 & 3 & -1 \\ \infty & \infty & 0 & -4 \\ \infty & \infty & \infty & 0 \end{pmatrix} \tag{2.4}$$

$$C^{(4)} = \begin{pmatrix} 0 & 9 & 12 & 8 \\ \infty & 0 & 3 & -1 \\ \infty & \infty & 0 & -4 \\ \infty & \infty & \infty & 0 \end{pmatrix} \tag{2.5}$$

Opäť je namieste podotknúť, že nami požadované vylepšenie na poznanie i predchodcov s cieľom rekonštrukcie nájdených najlacnejších ciest treba pridať ešte ďalších $|V|$ matíc, ktoré si danú informáciu budú počas výpočtu ukladať. Ich definíciu je možné uzrieť v už menovanom zdroji [10, kapitola 2.2.2].

Taktiež je vhodné poznamenať, že pre väčšie vstupy má algoritmus potenciál skonzumovať príliš veľa pamäte. Preto je v takom prípade výhodné algoritmus upraviť tak, aby používal iba dve matice, nakoľko pri algoritme je potrebné držať referenciu len na momentálnu a predchádzajúcu maticu ($C^{(k)}$ a $C^{(k-1)}$). To isté možno tvrdiť aj o maticiach predchodcov.

Floyd–Warshallov algoritmus je taktiež dobrým kandidátom pre riešenie nášho problému. Implementácia je opäť jednoduchá, časová zložitosť vítaná. Znovu ale, možnosť využitia algoritmu i na záporných hranách nie je pre nás nijako prínosná. Naviac, pri vyhľadávaní spojenia medzi dvomi zastávkami v grafikone MHD by bolo nutné vyrátať hodnoty najlacnejších ciest z každého vrcholu do všetkých vrcholov, čo môže trvať prdlho. Samozrejme, po skončení hľadania budú už nasledujúce dopyty uspokojené v konštantnom, resp. lineárnom (ak uvažujeme i rekonštrukcie ciest), čase.

### 2.2.4 Zhrnutie

Všetky tri prezentované algoritmy sú vhodnými adeptami pri riešení problému vyhľadávaní spojení v grafikone MHD. Každý z nich má svoje pozitíva, ale i nedostatky. Ktorí z algoritmov bude najvhodnejší, zavisí hlavne od dát, nad ktorými bude pracovať. Ako sme už spomínali, Bellman–Fordov algoritmus bude mať výhodu pri grafoch obsahujúcimi menšie množstvo hrán, čo, žiaľ, nebude prípad liniek MHD. Floyd–Warshallov algoritmus je zasa zostavený tak, aby vypočítal všetky najlacnejšie cesty. Preto bude jeho priebeh pri väčších vstupoch neprimerane dlhý. Jeho využitie preto vidíme pri grafikonom MHD malých miest, kde sa vyrátajú vsetky potrebné hodnoty už pri inicializácií, najlepšie na druhom vlákne, aby mohol používateľ zadávať dopyt už počas výpočtu. Využitie algoritmu by bolo v tomto prípade priam žiadané, keďže ostatné dva algoritmy môžu svoje výpočty pri väčšine postupností dopytov opakovať, čo ich môže nažiadane zdržovať. A nakoniec, Dijkstrov algoritmus sa zdá byť najvšeobecnejší a najpoužiteľnejší pre ľubovoľné dáta.

Existujú aj iné, pokročilejšie algoritmy, ktoré sú schopné nájsť najlacejšie cesty v grafe. Zvyčajne však ide o kombináciu nami uvedených algoritmov, prípadne ich optimalizácia alebo nejaká rafinovaná modifikácia. Na predstavu poslúži Johnsonov algoritmus, ktorého výstup je rovnaký ako výsledok Floyd–Warshallovho algoritmu, len jeho priebeh je iný - využíva kombináciu Dijkstrovho a Bellman–Fordovho algoritmu, čím dosahuje na riedkych grafoch výsledok rýchlejšie. Spomínaným pokročilejším algoritmom sa však podrobnejšie v našej práci venovať nebudeme.

# Kapitola 3

# Softvér

Táto kapitola obsahuje základné informácie o nami implementovanom softvére. Popisujeme v nej, ako program funguje, na čo slúži a ako sa ovláda. Uvádzame rôzne funkcie softvéru, prípadne aj príklady ich použitia.

# Kapitola 4

# Implementácia softvéru

V tejto kapitole uvedieme, ako sme postupovali pri implementácií softvéru. Odhalíme, aké postupy, štruktúry či algoritmy sme zvolili, pričom popíšeme aj dôvody nášho výberu. Neuvádzame celý kód, ale len tie časti, ktoré sa nám zdali najviac zaujímavé a zodpovedajúce obsahu našej práce.

# Záver

Na záver už len odporúčania k samotnej kapitole Záver v bakalárskej práci podľa smernice [?]: „V závere je potrebné v stručnosti zhrnúť dosiahnuté výsledky vo vzťahu k stanoveným cieľom. Rozsah záveru je minimálne dve strany. Záver ako kapitola sa nečísluje.“

Všimnite si správne písanie slovenských úvodzoviek okolo predchádzajúceho citátu, ktoré sme dosiahli príkazmi `\glqq` a `\grqq`.

V informatických prácach niekedy býva záver kratší ako dve strany, ale stále by to mal byť rozumne dlhý text, v rozsahu aspoň jednej strany. Okrem dosiahnutých cieľov sa zvyknú rozoberať aj otvorené problémy a námety na ďalšiu prácu v oblasti.

Abstrakt, úvod a záver práce obsahujú podobné informácie. Abstrakt je kratší text, ktorý má pomôcť čitateľovi sa rozhodnúť, či vôbec prácu chce čítať. Úvod má umožniť zorientovať sa v práci skôr než ju začne čítať a záver sumarizuje najdôležitejšie veci po tom, ako prácu prečítal, môže sa teda viac zamerať na detaily a využívať pojmy zavedené v práci.

# Literatúra


[1] Jørgen Bang-Jensen and Gregory Z Gutin. *Digraphs: theory, algorithms and applications*. Springer Science & Business Media, 2008.

[2] Michael J Bannister and David Eppstein. Randomized speedup of the bellman–ford algorithm. In *2012 Proceedings of the Ninth Workshop on Analytic Algorithmics and Combinatorics (ANALCO)*, pages 41–47. SIAM, 2012.

[3] Richard Bellman. On a routing problem. *Quarterly of applied mathematics*, 16(1):87–90, 1958.

[4] Reinhard Diestel. Teória grafov, 2000. [Citované 2017-12-4] Dostupné z `http://www.dcs.fmph.uniba.sk/~haviarova/uktg/#materialy`.

[5] Shimon Even. *Graph algorithms*. Cambridge University Press, 2011.

[6] Jana Katreniaková. Vizualizácia a kreslenie pekných grafov, 2014. [Citované 2017-12-4] Dostupné z `http://sccg.sk/~ferko/VizualizaciaPreGrafikov.pdf`.

[7] S Meena Kumari and N Geethanjali. A survey on shortest path routing algorithms for public transport travel. *Global Journal of Computer Science and Technology*, 9(5):73–75, 2010.

[8] Jakub Novák. Dynamická navigácia osôb so spätnou väzbou v mestskej hromadnej doprave. Bakalárska práca, Univerzita Komenského v Bratislave, 2016.

[9] José Luis Santos. k-shortest path algorithms. 2007. [Citované 2017-12-4] Dostupné z `https://estudogeral.sib.uc.pt/jspui/bitstream/10316/11305/1/k-Shortest%20path%20algorithms.pdf`.

[10] Pavol Ďuriš. *Tvorba efektívnych algoritmov*. Knižničné a edičné centrum FMFI UK, 2009.