Univerzita Hradec Králové

Fakulta informatiky a managementu

Katedra informatiky a kvantitativních metod

Dijkstrův algoritmus

Bakalářská práce

Autor: Jan Stránský

Studijní obor: Aplikovaná informatika

Vedoucí práce: RNDr. Andrea Ševčíková

Hradec Králové duben 2018

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 27.4. 2018 Jan Stránský

Poděkování:

Děkuji vedoucí bakalářské práce RNDr. Andree Ševčíkové za odborné vedení, za pomoc a rady při zpracování této práce. Rád bych také poděkoval rodině i všem přátelům za podporu při psaní této práce.

Anotace

Práce se zabývá konkrétní problematikou z oblasti teorie grafů, hledání minimálních cest v nezáporně ohodnoceném grafu. V práci je popsán Dijkstrův algoritmus, který je používán pro hledání nejkratších cest v grafech, kterými je možné reprezentovat různé situace z běžného života. V první části práce jsou vysvětleny základní pojmy z teorie grafů, které jsou v textu používané. Další část je věnovaná autorovi Dijkstrova algoritmu Edsgeru W. Dijkstru, je v ní popsán jeho život a práce. Poté je popsán Dijkstrův algoritmus a demonstrován v grafu a na matici. V poslední části je představena aplikace, která byla vytvořena jako součást práce.

Klíčová slova: Graf, Dijkstra, algoritmus, nejkratší cesta.

Annotation

Title: Dijkstra´s Algorithm

This bachelor thesis deals with specific problems from graphs theory, searching for the shortest ways in non-negatively rated graph. Dijkstra´s Algorithm is described in this thesis. It is used for searching of the shortest ways in graph and it is possible to represent different situation in the ordinary life. The first part of thesis explains basic terminology from the graphs theory. The next part then is devoted to the author of the Dijkstra´s Algorithm Edsger W. Dijkstra, his life and his work is described there. The next chapter is focused on explanation of Dijkstra´s Algorithm and his function is demonstrated on graph and on matrix. The last part introduces an application, which is created as part of this bachelor thesis and in which it is possible to try the algorithm.

Keywords: Graph, Dijkstra´s, algorithm, shortest path.

Obsah

1 Úvod 1
1.1 Cíl práce 2
2 Základní pojmy 2
2.1 Graf 2
2.2 Ohodnocený graf 4
2.3 Sled 5
2.4 Tah 5
2.5 Cesta 6
2.6 Definice algoritmu 7
3 Edsger Wybe Dijkstra 7
3.1 Biografie 8
3.2 Vědecké příspěvky 9
3.3 Pracovní styl 10
3.4 Dijkstrovy názory 11
4 Dijkstrův algoritmus 11
4.1 Princip Dijkstrova algoritmu 11
4.2 Pseudokód Dijkstrova algoritmu 13
4.3 Korektnost algoritmu 14
4.4 Výpočetní složitost 15
4.5 Ukázka Dijkstrova algoritmu v grafu 16
4.6 Ukázka Dijkstrova algoritmu na matici 20
4.7 Vztah Dijkstrova algoritmu s Jarníkovým algoritmem pro hledání minimální kostry 24
4.8 Použití Dijkstrova algoritmu 28
5 Implementace aplikace 29
5.1 Použité technologie 29
5.2 Návrh aplikace 30
5.3 Komponenty aplikace 30
5.3.1 Vrcholy 31
5.3.2 Hrany 32
5.3.3 Třída Graph 33
5.4 Funkce algoritmu 33

5.5 Ovládání aplikace 36
6. Testování aplikace 36
6.1 Testovací data 36
6.2 Testování 39
6.3 Vyhodnocení dat 44
7 Závěr a doporučení 44
8 Seznam použité literatury 45
A Obsah CD 49

Seznam obrázků

Obrázek 1: Neorientovaný graf 3
Obrázek 2: Orientovaný graf 3
Obrázek 3: Ohodnocený graf 4
Obrázek 4: Sled 5
Obrázek 5: Tah 6
Obrázek 6: Cesta 6
Obrázek 7: E.W.Dijkstra 7
Obrázek 8: Technická Univerzita v Eindhovenu 8
Obrázek 9:E.W.Dijkstra během konference ETH Zurich 10
Obrázek 10: Nejkratší vzdálenost d[u] 14
Obrázek 11: Počáteční stav grafu 16
Obrázek 12: Nastavení vrcholů 17
Obrázek 13: První krok Dijkstrova algoritmu 17
Obrázek 14: Druhý krok Dijkstrova algoritmu 18
Obrázek 15: Třetí krok Dijkstrova algoritmu 18
Obrázek 16: Čtvrtý krok Dijkstrova algoritmu 19
Obrázek 17: Pátý krok Dijkstrova algoritmu 19
Obrázek 18: Šestý krok Dijkstrova algoritmu 20
Obrázek 19: Výsledná nejkratší cesta 20
Obrázek 20: První krok Jarníkova algoritmus 24
Obrázek 21: Druhý krok Jarníkova algoritmu 25
Obrázek 22: Třetí krok Dijkstrova algoritmu 25
Obrázek 23: Čtvrtý krok Jarníkova algoritmu 26
Obrázek 24: Pátý krok Jarníkova algoritmu 26
Obrázek 25: Šestý krok Jarníkova algoritmu 27
Obrázek 26: Výsledná minimální kostra 27
Obrázek 27: Analytický model 30
Obrázek 28: Města - vrcholy 31
Obrázek 29: Vrchol označený 31
Obrázek 30: Vrchol neoznačený 31
Obrázek 31: Vykreslení vrcholu 32
Obrázek 32: Hrany - cesty 32

Obrázek 33: Metoda isPath 33
Obrázek 34: Třída Graph 33
Obrázek 35: Aplikace 34
Obrázek 36: Spuštění Dijkstrova algoritmu 34
Obrázek 37: Nastavení vrcholů 34
Obrázek 38: Metoda findPath 35
Obrázek 39: Metoda findPath druhá část 36
Obrázek 40: Nejkratší cesta mezi Českými Budějovicemi a Hradcem Králové 41
Obrázek 41: Informační okno 41
Obrázek 42: Nejkratší cesta mezi Prahou a Hodonínem 43
Obrázek 43: Informační okno 44

1 Úvod

V dnešním světě se elektronika a informační technologie neustále vyvíjí. S jejich pomocí je možné práci nejen urychlit, ale vyřešit i složité úkoly. Elektronikou je ovlivněn téměř každý a málokdo by si dokázal bez ní život představit. Využití informačních technologií nalezneme zejména ve stavebnictví, strojírenství, zdravotnictví a v mnoha dalších oborech. Také je velmi rozšířena v dopravě a cestování, ve kterých je možné setkat se s technologiemi jako je GPS (Global Positionig System) nebo vyhledávání na mapách. Není tomu tak dávno, co se používaly papírové mapy a trasy se musely důkladně plánovat. Pro výpočet nejkratší cesty z jednoho místa do druhého se muselo složitěji počítat, než je tomu dnes. Global Positionig System, česky Globální polohovací systém, [25] je družicový systém vybudovaný pro potřeby navigace a určování polohy na Zemi, jehož služby jsou dostupné nepřetržitě, kdykoliv a kdekoliv na zemském povrchu a přilehlém okolí. [26] V GPS navigaci se zadá start trasy (počáteční bod) a cíl trasy (koncový bod). Díky algoritmu, který vypočítává nejkratší vzdálenost, se dostane požadovaný výsledek, a to nejkratší vzdálenost mezi počátečním a koncovým bodem. Algoritmus je možné použít v počítačových sítích u směrovacího protokolu OSPF (Open shortest path first), který vypočítá nejmenší vzdálenost (metriku[1]) mezi směrovači[2]. Algoritmus používaný v GPS vynalezl nizozemský informatik Edsger Wybe Dijkstra a dnes je již znám pod názvem Dijkstrův algoritmus.

Oblast hledání nejkratší cesty je součástí matematického oboru teorie grafů. Mezi zakladatele teorie grafů je považován Leonhard Euler, který publikoval řešení příkladu Sedmi mostů města Královce. [3] Grafy pak zůstávaly na okraji zájmu matematiků. Vrátil se k nim až roku 1847 Gustav Kirchhoff, když se zabýval výpočtem proudů v elektrických sítích pomocí počtu koster grafu. [3] Významných poznatků v teorii grafů bylo dosaženo také v Československu. Mezi nejznámější matematiky je uváděn Otakar Borůvka [30] a Vojtěch Jarník. [31]

[1] Metrika sítě je číselným (obvykle celočíselným) vyjádřením náročnosti dosažení předmětné sítě ve směrování v sítích s přepínáním paketů. [27]
[2] Směrovač, neboli router je v počítačových sítích aktivní síťové zařízení, které procesem zvaným routování přeposílá datagramy směrem k jejich cíli. [28]

1.1 Cíl práce

Cílem bakalářské práce je zpracování Dijkstrova algoritmu do uceleného logického celku a vytvoření aplikace, ve které bude algoritmus použit.

První část práce obsahuje teoretický základ, ve kterém jsou definovány hlavní pojmy, které jsou v práci používané. Jedna kapitola je věnována autorovi tohoto algoritmu a jeho práci. Ve druhé polovině teoretické části práce je popsán samotný algoritmus a je předvedeno použití algoritmu v grafu a na matici.

Praktická část je zaměřena na implementaci a použití algoritmu v praxi. Součástí je i vytvoření testovacích dat, které budou použity a následně vyhodnoceny.

2 Základní pojmy

Pro lepší pochopení dané problematiky budou nejdříve vysvětleny některé základní pojmy z teorie grafů.

2.1 Graf

Graf je jedním z velmi důležitých pojmů diskrétní matematiky. [10] Graf je možné reprezentovat více způsoby. Nejčastěji je graf reprezentován graficky pomocí vrcholů a hran. Dále lze graf vyjádřit maticí sousednosti[3], maticí incidence[4] a nebo seznamem sousedních vrcholů[5].

Graf může vyjadřovat souvislost mezi objekty. [10] Vztahy mezi vrcholy jsou reprezentovány hranami. Pokud tedy jsou dva objekty ve vztahu, v grafu to bude znázorněno spojením příslušných vrcholů hranou. [11]

Kreslení vrcholů není nijak zvláště omezeno. Obvykle jsou vrcholy kresleny jako kružnice, elipsy nebo obdélníky. Hrany jsou kresleny přímými čárami, oblouky nebo lomenými čárami. Způsob kreslení vrcholů a hran je zvolen především tak, aby graf byl přehledný. [11]

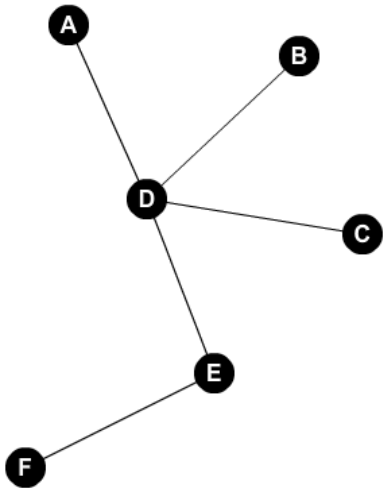
[3] http://algoritmy.eu/zga/reprezentace-grafu/
[4] https://www.algoritmy.net/article/1369/Graf
[5] http://algoritmy.eu/zga/reprezentace-grafu/

Definice grafu dle [1]:

„Graf G je uspořádaná dvojice (U, H)*, kde U je nějaká neprázdná množina a H je množina dvoubodových podmnožin množiny U. Prvky množiny U se jmenují vrcholy grafu G a prvky množiny H hrany grafu G.“*

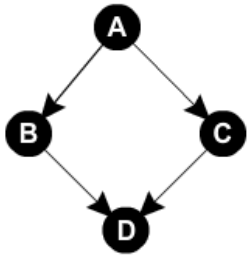
Podle typu hran jsou grafy rozděleny na:

- Neorientované - obsahují takové hrany, které reprezentují symetrické vztahy mezi vrcholy. Všechny hrany v tomto grafu jsou neorientované. [11]



Obrázek 1: Neorientovaný graf, zdroj: [35]

- Orientované - představují takové hrany, které jsou jednosměrné a mají nesymetrický vztah mezi vrcholy. Všechny hrany v tomto grafu jsou orientované. [11]



Obrázek 2: Orientovaný graf, zdroj: [35]

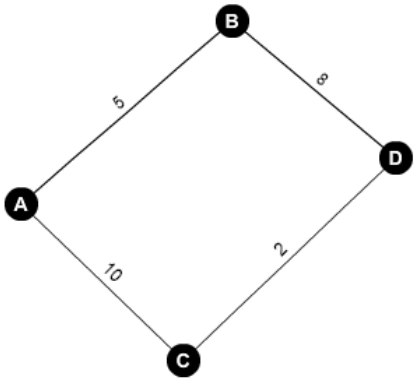
- Smíšené - obsahují neorientované i orientované hrany. [11]

Mnoho situací v matematice i v různých praktických úlohách je možné vystihnout pomocí grafu. Zde jsou některé uvedeny:

1. Automapa - vrcholy představují města a vesnice, hrany odpovídají silnicím mezi městy a vesnicemi. Jedná se o konečný neorientovaný graf. Při zobrazení jednosměrných silnic v obci se jedná o orientovaný graf. [12]
2. Lidé v jedné místnosti - lidé představují vrcholy a vztahy mezi lidmi jsou hrany, které odpovídají těm lidem, kteří se mezi sebou znají. [12]
3. Atomy znázorňující vrcholy a hrana představující chemickou vazbu mezi nimi. [12]

2.2 Ohodnocený graf

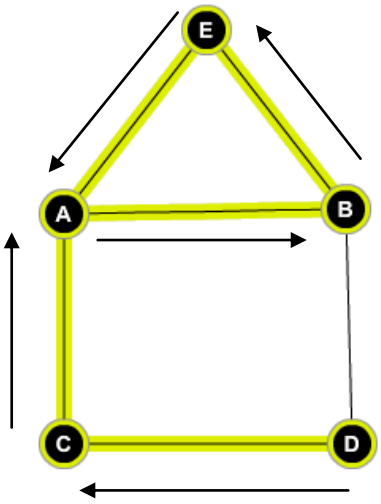
V mnoha případech obyčejný graf není dostačující pro vystihnutí problému. Jde o situace, kdy je potřeba vyjádřit i váhu vztahu. Proto se k hranám nebo vrcholům přidávají hodnoty (obvykle celočíselné), které vyjadřují délku trvání, vzdálenost nebo náklady činností. [2] Tím vznikne ohodnocený graf. Pokud tedy bude hledána nejkratší cesta z jednoho místa do druhého, není tak důležité, kolik hran tato cesta obsahuje, ale jakou váhu bude celkově mít.



Obrázek 3: Ohodnocený graf, zdroj: [35]

2.3 Sled

Sledem v grafu se rozumí taková posloupnost vrcholů, ve které je mezi každými dvěma po sobě jdoucími vrcholy hrana. [14] Na Obrázku 4 je znázorněn sled délky 5 s vrcholy: D, C, A, B, E, A



Obrázek 4: Sled, zdroj: [35]

Definice sledu zní:

„Nechť $G = (U, H)$ *je graf. Posloupnost* $(u_0, h_1, u_1, h_2, \dots, h_n, u_n)$ *se nazývá sled v grafu G, jestliže platí* $h_i = \{u_{i-1}, u_i\} \in E$ *pro* i *1, ...,n.“* [1]

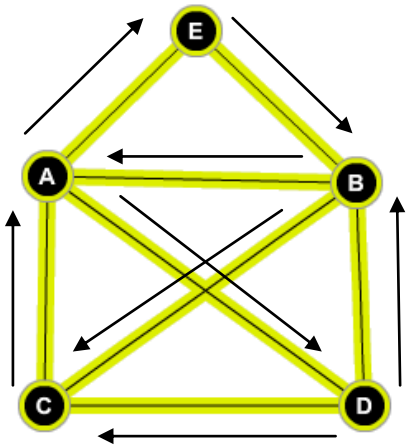
Ve sledu se mohou některé vrcholy i hrany opakovat. Sled je možné si představit jako záznam trasy bloudícího poutníka. [1]

2.4 Tah

Tahem v grafu G mezi vrcholy u a v se rozumí takový sled, ve kterém se žádná hrana nevyskytuje vícekrát. (Vrcholy se opakovat mohou) [11]

Tah je vždy sledem, ale není vždy cestou. *„Název tah je pravděpodobně odvozen od známé úlohy nakreslit ‚domeček‘ nebo jiný obrázek jedním tahem.“* [2]

Na Obrázku 5 je tento graf nakreslený jedním tahem znázorněn.

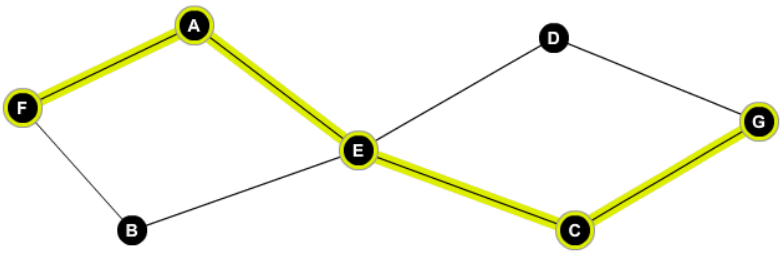


Obrázek 5: Tah, zdroj: [35]

Tah začíná ve vrcholu D a pokračuje přes tyto vrcholy: C, A, E, B, A, D, B, C

2.5 Cesta

Cesta je definována jako sled, ve kterém se neopakují vrcholy. [15]. Z toho vyplývá, že nedochází k opakování hran. Každá cesta je tedy zároveň také tahem a sledem. [2] Cesta disponuje i orientovanou variantou, která respektuje orientaci hran. Uzavřená cesta je pak nazývána kružnicí. [36] Příklad cesty délky 4, která vede přes vrcholy: F, A, E, C, G je znázorněna na Obrázku 6.



Obrázek 6: Cesta, zdroj: [35]

2.6 Definice algoritmu

Algoritmus postupně řeší určitý problém a má přesný počet definovaných kroků. Pro mnoho lidí se může jednat o neznámý pojem, ale s algoritmem se setkává každý, i když si to neuvědomuje. Je možné se s tímto pojmem setkat při vaření, u návodů a různých postupů. [4]

„ Samotné slovo algoritmus pochází ze jména perského matematika 9. století Abu Jafar Muhammada ibn Mūsā al-Chwārizmího, který ve svých dílech položil základy algebry (arabské číslice, řešení lineárních a kvadratických rovnic).“ [4]

3 Edsger Wybe Dijkstra

Dříve než bude popsán vlastní algoritmus, bude jedna kapitola věnována samotnému autorovi.

Edsger Wybe Dijsktra byl nizozemský informatik, který významně přispěl k rozvoji programovacích jazyků a roku 1972 obdržel Turingovu cenu. Dijkstra byl váženým nizozemským informatikem a mnoho pojmů, které jsou dnes známy, zavedl právě on.

Například:

- Semafory
- Bankovní algoritmus
- Algoritmus nejkratší cesty

Obrázek 7: E.W.Dijkstra, zdroj: [5]

3.1 Biografie

Edsger Wybe Dijsktra se narodil 11. května roku 1930 v Rotterdamu jako třetí dítě rodičům Douwe Wybe Dijkstra a Brechtje Cornelia Kluijver. [7]

Ve dvanácti letech začal studovat na Gymnáziu Erasminium, škole pro neobyčejně nadané studenty. Protože Dijsktra na Gymnáziu vynikal v chemii, matematice a fyzice, jeho studium dále pokračovalo na Leidenské univerzitě, kde studoval obecnou fyziku. V roce 1951 se účastnil letní školy na univerzitě v Cambridge, kde studoval programování. Roku 1952 začal pracovat v Mathematical Centre v Amsterdamu a právě tady jeho zájem o programování vzrostl. [5] V Amsterdamu se také seznámil se svojí ženou Riou. [8]

V roce 1962 se přestěhoval do Eindhovenu, kde se stal profesorem na katedře matematiky na Technické univerzitě v Eindhovenu. (viz. Obrázek 8) Poté se roku 1964 přestěhoval do nově postaveného domu v Nuenu na okraji Eindhovenu. [8]

Obrázek 8: Technická Univerzita v Eindhovenu, zdroj: [33]

„Po ukončení studia na vysoké škole a získání titulu v oboru fyziky se začal Dijkstra zabývat programováním. V té době se ale setkal s problémem, kterým byl fakt, že programování se oficialně ještě nepovažovalo za profesi. Z toho důvodu pokračoval v práci v Mathematical Centre až do roku 1970, kdy přijal pracovní místo ve výzkumu pro

Burroughs Coroporation[6] *v USA. Za necelé dva roky byl oceněn a získal ACM Turing Award, dále pak AFIPS Harry Memorial Award.“* [5]

Byl jmenován předsedou oboru informatiky na Texaské univerzitě v Austinu, ve které pracoval až do roku 1999. V únoru roku 2002 se vrátil zpět do svého domu v Nuenu, kde po půl roce zemřel na rakovinu. Zanechal po sobě manželku a 3 děti Marcuse, Femke a Rutgera. [8]

3.2 Vědecké příspěvky

Prostřednictvím svých základních příspěvků Dijkstra utvářel a ovlivňoval pole informatiky jako žádný jiný vědec. Jeho průkopnické příspěvky pokrývaly několik oblastí, včetně konstrukce překladačů, operační systémy, Disable tributed systémy, sekvenční a souběžné programování, softwarové inženýrství a grafové algoritmy. Mnoho z jeho dokumentů, často jen několik stran dlouhých, jsou zdrojem celých nových oblastí výzkumu. Některé pojmy, které nyní zcela patří do standardní výpočetní techniky, byly identifikovány právě Dijkstrou a jsou pojmenovany po něm. Například v roce 1959 vydal práci, která se týkala algoritmu pro nalezení nejkratší cesty, a zároveň vydal velmi efektivní algoritmus pro hledání nejkratší kostry. Publikoval to na dvou stránkách práce "*A note on Two Problems in Connexion with Graph*". Algoritmus pro nalezení nejkratší cesty je dnes znám jako Dijkstrův algoritmus. Mezi další nejznámější Dijkstrovy příspěvky patří i idea semaforu, nástroje pro synchronizaci vícero procesorů a programů. [8]
Jeho slavný dokument *Go To Statement Considered Harmful*[7] kritizoval použití příkazu GOTO a byl jedním z iniciátorů k jeho všeobecnému zavržení a téměř úplnému nahrazení řídícími strukturami, jakou je např. cyklus. [8]

Dijkstra byl členem týmu, který vytvářel úplně první překladač programovacího jazyka ALGOL 60. Spolu s Jaapem Zonneveldem se dohodli, že se až do dokončení projektu nebudou holit. Na rozdíl od Zonnevelda poté Dijsktra nosil bradku až do smrti. Od 70. let se Dijkstra věnoval formální verifikaci programů. Tehdejším běžným způsobem verifikace byla konstrukce matematického důkazu k již dokončenému programu. Konstrukce takových důkazů je však velice náročná a z výsledku nelze poznat myšlenky, podle kterých byl program vytvořen. Dijkstrův alternativní způsob spočívá v opačném

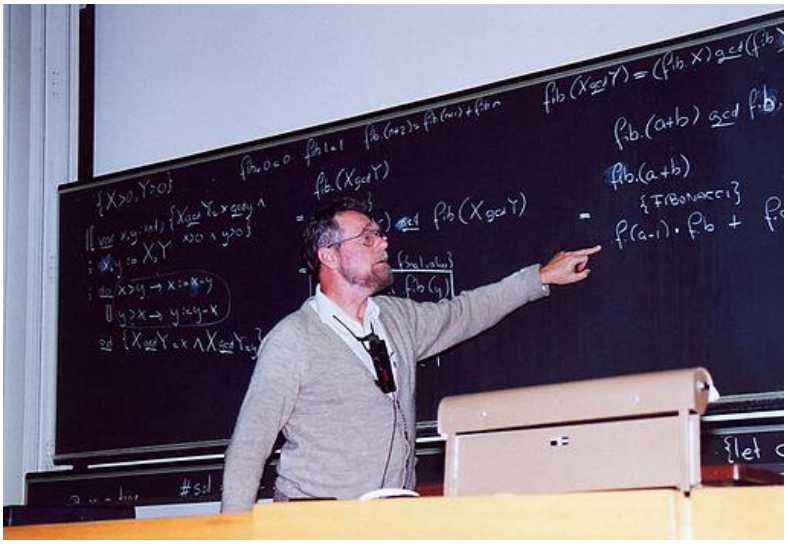
[6] Firma Burroughs byla hlavním americkým výrobcem obchodního zařízení
[7] Příkaz Go To považován za škodlivý. Název je dílem Niklause Wirtha, tehdejšího editora Communications of the ACM.

postupu. Začíná se matematicky formulovat specifikace, co a jak má program dělat, a z této specifikace se pak pomocí matematických transformací postupně vytváří program, který je poté možno spustit. Takový program je pak zaručeně správný již způsobem své konstrukce. [8]

3.3 Pracovní styl

Dijkstra nikdy nepsal své články pomocí počítače. Raději je psal na psacím stroji a později pomocí pera. Tyto články byly následně distribuovány po staru: poslal kopie několika přátelům a spolupracovníkům, kteří pak sloužili jako zdroj distribučních center. Tyto krátké články v časovém rozpětí čtyřiceti let jsou zřídka delší než patnáct stran a jsou postupně číslovány. Poslední z nich číslo 1318 je z 14. dubna 2002. [8]

Dijkstra aplikoval na svoji práci přísný postup sebehodnocení a jen malý zlomek z jeho prací byl nakonec zveřejněn v časopisech. V důsledku toho, nejsou mnohé příspěvky známé. Jeho rukopis byl tak dokonalý a zřetelný, že ve druhé polovině 80. let Luca Cardelli, poté DEC Systems Research Center, navrhli "Dijkstra" písmo pro počítače Macintosh. Brzy Dijkstra dostal dopis napsaný v tomto fontu a myslel, že byl napsán rukou, dokud se k němu nedostalo, že došlo k zavedení tohoto písma. Někteří z Dijkstrových kolegů používali toto písmo ve svých prezentací během odborového setkání v Austinu. [8]



Obrázek 9:E.W.Dijkstra během konference ETH Zurich, zdroj: [34]

3.4 Dijkstrovy názory

V osobním setkání s kolegy se jevil jako odměřený, přísný a chladný. V některých vzácných případech byl dokonce hrubý, např. na počátku 80. let v Utrechtu uprostřed výuky počítačových sítí demonstrativně odešel. Nebo o několik let později v Austinu řekl: "Díky bohu", v reakci na komentář: "já ztrácím hlas", což pronesl renomovaný počítačový vědec z MIT na konci své přednášky. Ale v neformálních soukromých setkáních ve své kanceláři byl nejvíce okouzlující, připravoval kafé pro studenty a dělal své vlastní vtipy. [8]

V důsledku svého chování a nekompromisní pozice byl neoblíbený mezi řadou kolegů, zejména v Nizozemsku. Neviděli nic jiného než jeho kousavé komentáře, které se často ozývaly ze zadní části přednáškové místnosti. Často tento skepticismus vůči jeho názorům a myšlenkám byl ze závisti, že jeho články byly ve velkém množství citovány a diskutovány. [8]

4 Dijkstrův algoritmus

Dijkstrův algoritmus slouží k nalezení všech nejkratších cest z jednoho vrcholu do všech ostatních vrcholů. Při hledání nejkratší cesty je důležité, aby graf neobsahoval hrany se zápornou hodnotou. [16] Počet průchodů je nejvýše roven počtu vrcholů. Do množiny navštívených vrcholů se v každém průchodu přidá právě jeden vrchol. Lze tedy označit tento algoritmus jako konečný, protože pro jakýkoliv konečný vstup algoritmus skončí. [17]

4.1 Princip Dijkstrova algoritmu

Dijkstrův algoritmus je vlastně obecné prohledávání grafu do šířky[8]. Avšak šíření není na základě počtu hran od zdroje, ale na základě vzdálenosti od zdroje. Vrcholy, ke kterým tedy byla nalezena nejkratší cesta, budou zachovány. [16]

V tomto algoritmu je zavedena prioritní fronta[9], ve které jsou prvky (vrcholy) řazeny dle vzdálenosti od zdroje. V první iteraci je vzdálenost zdroje nastavena na hodnotu

[8] Prohledávání do šířky je jedním ze základních grafových algoritmů.
[9] Prioritní fronta je speciální případ fronty. Prvky jsou řazeny dle priority a zařazený prvek s vyšší prioritou předbíhá méně prioritní prvek.

0 a všechny ostatní vrcholy na nekonečno. Algoritmus vybírá z fronty ten vrchol, který má nejvyšší prioritu (nejnižší vzdálenost) a zařadí ho mezi zpracovávané vrcholy. Od tohoto vrcholu se poté procházejí všichni jeho dosud nezpracovaní potomci a přidávají se do fronty (pokud tam již nejsou) a dojde k ověření, jestli nejsou blíže zdroji, než byli před zařazením mezi zpracovávané. Pro všechny potomky je tedy ověřováno:

$$vzdálenost_{zpracov\ávaný} + délkaHrany_{zpracov\ávaný} < vzdálenost_{potomek}$$

Pokud je podmínka splněna, pak k danému potomku existuje kratší cesta a je označen za předka toho zpracovávaného vrcholu. Po ověření všech potomků je vybrán z fronty vrchol s nejvyšší prioritou a celý krok se opakuje. K ukončení algoritmu dochází, když jsou všechny vrcholy zpracovány. [16]

4.2 Pseudokód Dijkstrova algoritmu

V následujícím pseudokódu je vysvětlena funkce Dijkstrova algoritmu: Parametr *d* znázorňuje matici, která určuje délku mezi vrcholy grafu. Parametr *from* je vrcholem, ze kterého se hledají nejkratší cesty do ostatních vrcholů. [16]

Dijkstrův algoritmus

```
1: procedure int[] doDijkstra(d, from) {        // Vrchol from má vzdálenost 0, všechny ostatní nekonečno
2:     Q = Insert (from)                        // Uložení počátečního vrcholu do prioritní fronty

3:     CLOSED = {}                              // Uzavřené vrcholy - prázdná množina - vrcholy které již byly expandovány
4:   predecessors = new array[d.nodeCount]      // Pole předchůdců pro všechny vrcholy
5:       while !Q.isEmpty() do
6:           node = Q.extractMin()              // Vrať vrchol v nejnižší vzdálenosti a odstraň jej z fronty
7:           CLOSED.add(node)                   // Uzavři uzel
8:
9:               for a in Adj(node) do          // Pro všechny potomky vrcholu expanduj vrchol vybraný z fronty
10:                  if !CLOSED.contains(a)     // Pokud již nebyl potomek expandován

                         if !Q.contains(node)   // Pokud fronta neobsahuje vrchol
                             Q.insert(node)     // Přidej vrchol do fronty

                 // Vrchol je již obsažen ve frontě. Pokud je vzdálenost nově nalezené hrany menší než předešlá
11:                  else if Q[node].distance + d[node][a] < Q[a].distance

12:                      Q[a].distance = Q[node].distance + d[node][a]   // Změň prioritu vrcholu
13:                      predecessors[a] = node      // Změň předka vrcholu na node
14:
15:    return predecessors
```

4.3 Korektnost algoritmu

Nechť graf $G(U, H)$ je orientovaný s nezáporným ohodnocením hran *w*. Nechť je na něm prováděn Dijkstrův algoritmus a vrchol *s* nechť je počátečním. Poté v okamžiku výběru každého vrcholu u v algoritmu na řádku 6: (viz. kapitola 4.2) platí $node[u] = node_w(s, u)$ a tato hodnota se již nezmění. [24]

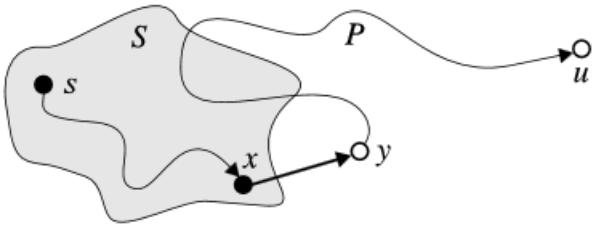
Důkaz správnosti Dijkstrova algoritmu bude proveden metodou indukce podle počtu uzavřených vrcholů, tedy vrcholů vybraných z fronty Q.

*Důkaz korektnosti:*MI (počet vrcholů)

1. Prvním vrcholem, který bude vybrán z fronty Q je vrchol *s*, pro který platí $d[s] = d_w(s, s) = 0$, tvrzení pro jeden vrchol platí.

2. *„Nechť tvrzení platí pro každý z prvních* $n \geq 1$ *uzavřených vrcholů a je uvažován* $(n + 1)$*-ní vrchol u, vybraný z fronty Q. Platnost tvrzení bude dokázána sporem. Nechť při výběru vrcholu u platí* $d[u] > d_w(s, u)$*, pak je hodnota* $d[u]$ *konečná a v G existuje nějaká orientovaná cesta z vrcholu s do vrcholu u.“* [24]

   Existuje tedy i nejkratší cesta P z *s* do *u*. Dále je označen vrchol *y* jako první vrchol cesty P, který dosud nebyl uzavřen (může být *u* = *y*) a jako *x* předchůdce vrcholu *y* na cestě P (viz. Obrázek níže).



Obrázek 10: Nejkratší vzdálenost d[u], zdroj: [24]

Protože $x \in S$, všechny relaxace[10] hran, které vystupují z x již byly provedeny při uzavření vrcholu x takže platí $d[y] = d_w(s, y)$ a tato hodnota se již nemůže změnit. Vzhledem k nezápornému ohodnocení hran se nyní dostane:

$$d[y] = d_w\,(s, y)\ \leq\ d_w(s, u)\ < d[u],$$

čím vznikne spor s podmínkou výběr u vrcholu u jako vrcholu s minimální hodnotou $d[u]$ z dosud neuzavřených vrcholů. [24]

4.4 Výpočetní složitost

Dijkstrův algoritmus tedy správně určí w-vzdálenosti od vrcholu s. Ještě je nutné určit jeho výpočetní složitost.[11] [24] Počáteční inicializace a cyklus (viz. kapitola 4.2) proběhnou v čase $O(|U|)$. Složitost Dijkstrova algoritmu závisí na tom, jak bude implementována prioritní fronta. [16] K implementaci prioritní fronty je možné použít sekvenční vyhledávání[12] nebo binární haldu[13]. V případě použití binární haldy, lze výběr vrcholu provést v čase $O(\log_2|U|)$. Celkově tedy budou operace výběru trvat $O(|U|\log_2|U|)$. Relaxace hrany je prováděna celkem $|H|$ - krát a pokud dojde ke změně hodnoty $d[u]$, je nutné umístit vrchol u na odpovídající místo v prioritní frontě, což trvá $O(\log_2|U|)$. Celkem je tedy asymptotická časová složitost:

$$O(|U|\log_2|U| + |H|\log_2|U|) = O\big((|U| + |H|)\log_2|U|\big) = O(|H|\log_2|U|).$$

Z tohoto je možné usoudit, že binární haldu bude výhodné použít pro grafy, které nemají velký počet hran (tzv. řídké grafy) a kdy je počet hran omezen počtem vrcholů. V případě použití Fibonacciho haldy[14] je dokonce možné dále snížit celkovou výpočetní složitost pro operaci výběru vrcholů na $O(|U|\log_2|U|)$ a na $O(|H|)$ pro úpravy při hledání nejkratší cesty, protože pak se jedna operace zařazení vrcholu se sníženou hodnotou $d[u]$ provádí v konstantním čase. Celková složitost v tomto případě je:

$$O(|U|\log_2|U| + |H|) \quad .$$

[10] Relaxace je technika, kterou využívají algoritmy pro hledání nejkratších cest. [29]
[11] Složitost udává, jak je daný algoritmus rychlý vzhledem k množině vstupních dat. [38]
[12] https://www.algoritmy.net/article/19/Linearni-vyhledavani
[13] https://cs.wikipedia.org/wiki/Bin%C3%A1rn%C3%AD_halda
[14] https://cs.wikipedia.org/wiki/Fibonacciho_halda

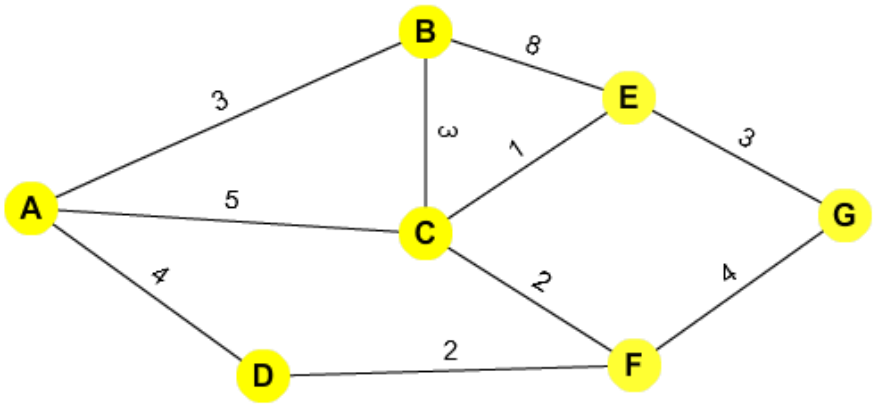
Pro grafy, které obsahují mnoho hran (tzv. husté grafy ($tj. |H| = O(|U|^2)$)), je možné použít sekvenční implementaci fronty Q v poli. Výběr jednoho vrcholu je pak proveden v čase $O(|U|)$. Celkově tedy $O(|U|^2)$. *„Naproti tomu snížení hodnoty* $d[u]$ *lze zajistit v konstantním čase.“* V tomto případě je celková výpočetní složitost Dijkstrova algoritmu tato: [24]

$$O(|U|^2 + |H|) = O(|U|^2)$$

4.5 Ukázka Dijkstrova algoritmu v grafu

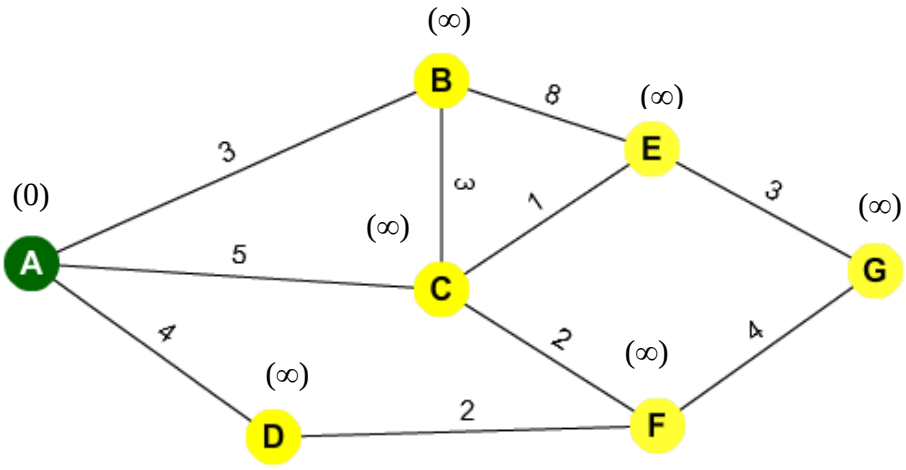
Pro lepší pochopení funkce, bude Dijkstrův algoritmus vysvětlen na příkladu grafu (Obrázek 11). Grafy byly kresleny v online programu Graph creator. [35]

Graf se skládá ze sedmi vrcholů, které jsou mezi sebou spojeny hranami. Vzdálenosti mezi jednotlivými vrcholy jsou ohodnoceny různými nezápornými hodnotami. Cílem je nalézt nejkratší cestu mezi vrcholy A a G.



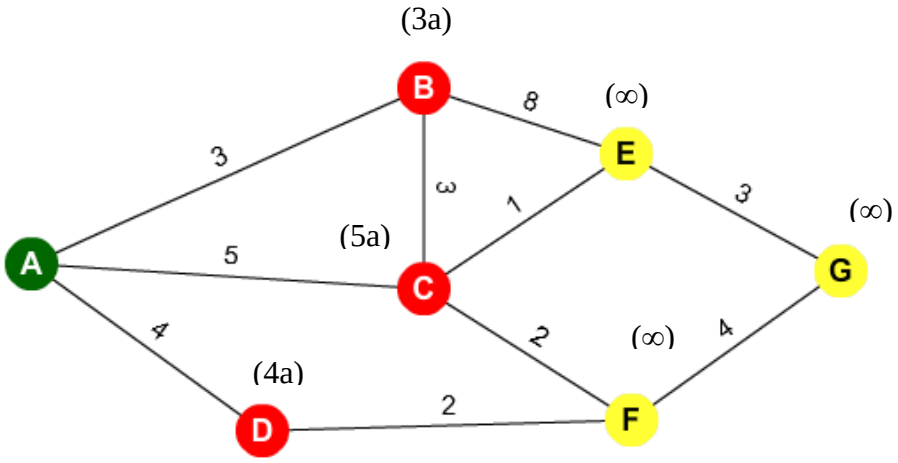
Obrázek 11: Počáteční stav grafu

Počáteční vrchol je označen zelenou barvou a je ohodnocen hodnotou 0. Všem ostatním vrcholům je nastavena vzdálenost od počátku na nekonečno.



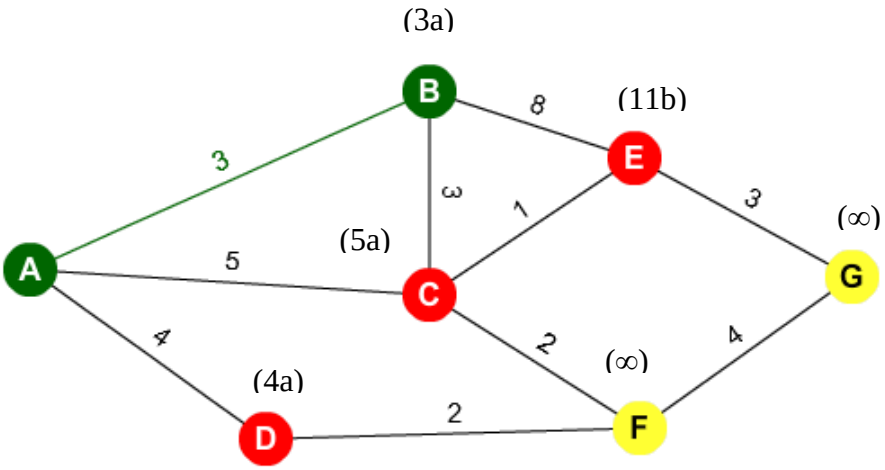
Obrázek 12: Nastavení vrcholů

V prvním kroku dojde k expanzi počátečního vrcholu, kdy se sousedním vrcholům nastaví aktuální vzdálenost od počátku. V tomto konkrétním případě B = 3, C = 5 a D = 4. Následně jsou vrcholy uloženy do prioritní fronty, kde nejvyšší prioritou je nejmenší vzdálenost. Sousední vrcholy jsou označeny červenou barvou.



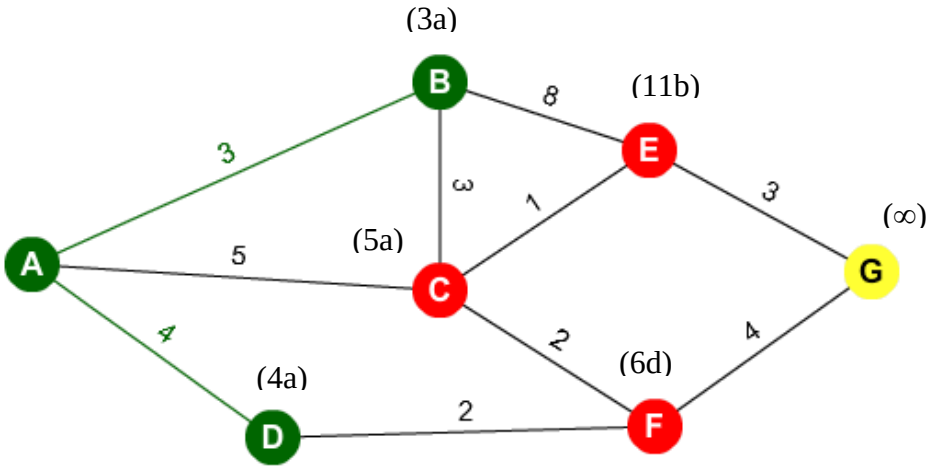
Obrázek 13: První krok Dijkstrova algoritmu

Ve druhém kroku je vybrán vrchol s největší prioritou (B) a je provedena expanze. U nalezených vrcholů je vypočtena vzdálenost od počátku (E = 11, C = 6). Do fronty je přidán vrchol E. Jelikož vrchol C je již ve frontě obsažen a jeho nově nalezená vzdálenost je větší než uložená, hodnota u tohoto vrcholu není aktualizována.



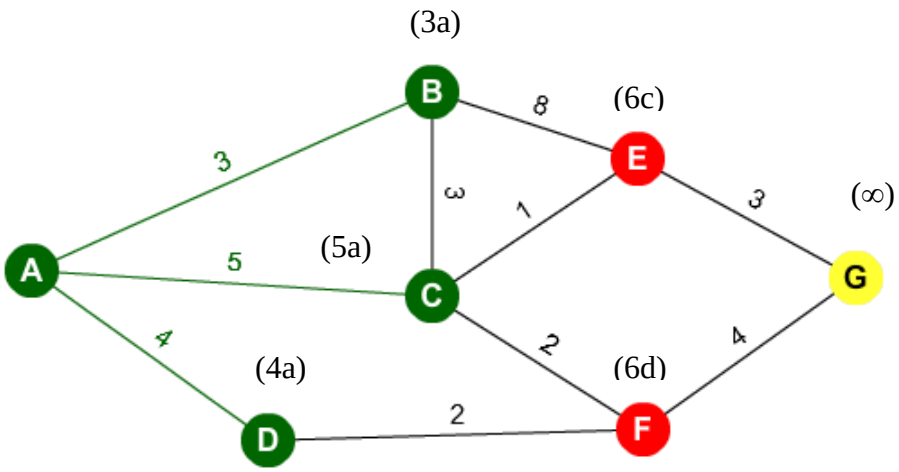
Obrázek 14: Druhý krok Dijkstrova algoritmu

Algoritmus opět vybere z fronty vrchol s nejvyšší prioritou (D) a provede expanzi. Do fronty je uložen vrchol F s hodnotou 6.



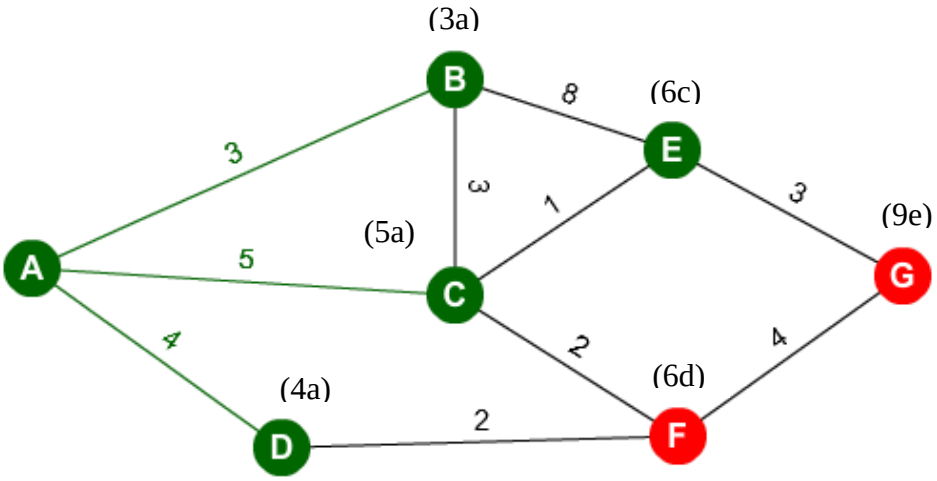
Obrázek 15: Třetí krok Dijkstrova algoritmu

Algoritmus opět vybere z fronty vrchol s největší prioritou (C) a provede expanzi. Jelikož vrchol E je již ve frontě obsažen a jeho nově nalezená vzdálenost je menší, hodnotu u tohoto vrcholu aktualizujeme. Ve frontě je také obsažen vrchol F, ale nově nalezená hodnota je větší, proto hodnota u tohoto vrcholu není aktualizována.



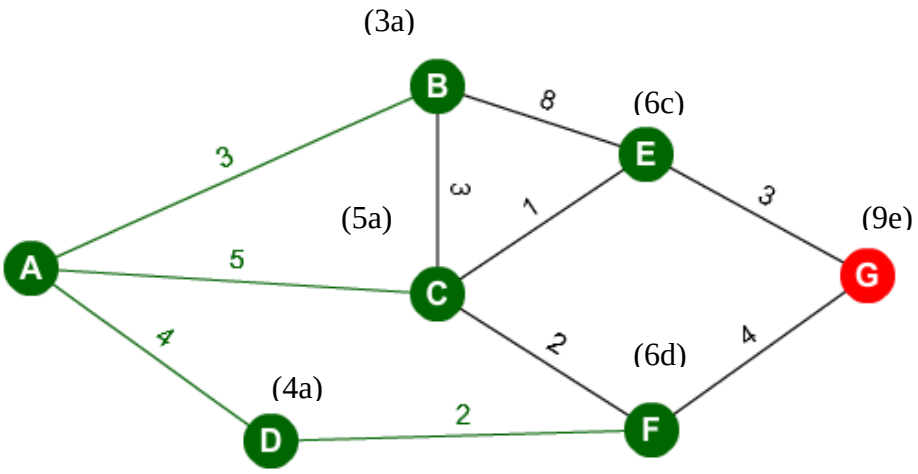
Obrázek 16: Čtvrtý krok Dijkstrova algoritmu

Algoritmus opět vybere z fronty vrchol s největší prioritou. Jelikož vrcholy E a F jsou ve frontě a mají stejnou hodnotu, vyšší prioritu dostane ten vrchol, který byl vložen do fronty dříve. Tedy bude vybrán vrchol E a provede se expanze. Do fronty je uložen vrchol G s hodnotou 9.



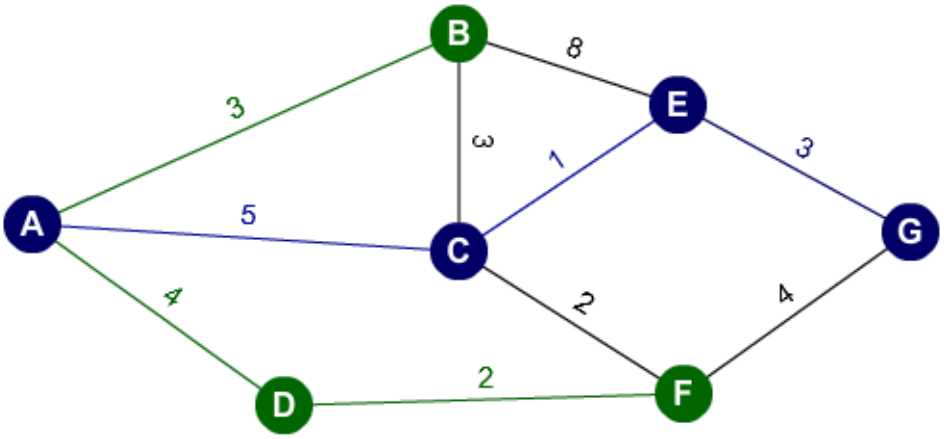
Obrázek 17: Pátý krok Dijkstrova algoritmu

Algoritmus opět vybere z fronty vrchol s největší prioritou (F) a provede expanzi. Jelikož vrchol G je již ve frontě a jeho nově nalezená vzdálenost je větší, hodnotu u tohoto vrcholu neaktualizujeme.



Obrázek 18: Šestý krok Dijkstrova algoritmu

Následně je algoritmus ukončen, protože našel nejkratší cestu z vrcholu A do vrcholu G. Nejkratší cesta prochází přes vrcholy A, C, E, G a její velikost je 9.



Obrázek 19: Výsledná nejkratší cesta

4.6 Ukázka Dijkstrova algoritmu na matici

Výchozím grafem pro ukázku Dijkstrova algoritmu na matici bude graf z kapitoly 4.5. Obrázek 11. V následujícím grafu zadaném maticí bude určena nejkratší cesta z

vrcholu ***a*** do vrcholu ***g***. Každému vrcholu je přiřazena informace o vzdálenosti od ***a***, a předcházející vrchol na nejkratší cestě z vrcholu ***a***.

- *b* - 3*a* (sousedí s hranou ***a*** o velikosti 3)
- *c* - 5*a* (sousedí s hranou ***a*** o velikosti 5)
- *d* - 4*a* (sousedí s hranou ***a*** o velikosti 4)
- *e,f,g* - nesousedí s hranou ***a***, proto je velikost nastavena na nekonečno

Vybrán bude nejmenší 3*a* - vrchol ***b***

Vybrán bude vrchol ***b*** a u něj se zjistí, zda nějaký vrchol nemá menší vzdálenost od ***a*** přes vrchol ***b***.

	a	*b*	*c*	*d*	*e*	*f*	*g*		
a		3	5	4					--
b	3		3		8				3*a*
c	5	3			1	2			5*a*
d	4					2			4*a*
e		8	1				3		∞
f			2	2			4		∞
g					3	4			∞

- c - dosavadní vzdálenost od ***a*** do ***c*** je 5. Přes vrchol ***b*** by vzdálenost byla ***ab*** + ***bc***, tedy 3+3 = 6, což je větší než cesta ***ac***. Proto hodnota u vrcholu bude ponechána.
- e - dosavadní vzdálenost je ∞, velikost hrany ***ab*** je 3, velikost hrany ***be*** je 8. To je menší než ∞, proto hodnota u vrcholu ***e*** bude změněna na 11*b*.

Vybrán bude vrchol ***d*** a dochází k dalšímu porovnávání.

	a	*b*	*c*	*d*	*e*	*f*	*g*			
a		3	5	4					--	--
b	3		3		8				3*a*	--
c	5	3			1	2			5*a*	5*a*
d	4					2			4*a*	4*a*
e		8	1				3		∞	11*b*
f			2	2			4		∞	∞
g					3	4			∞	∞

Jediným dalším možným sousedem vrcholu ***d*** je vrchol ***f***.

- *f* - dosavadní vzdálenost je ∞. Velikost hrany ***ad*** je 4 a velikost hrany ***df*** je 2. Tedy 4+2 < ∞. Proto hodnota u vrcholu ***f*** bude změněna na 6*d*.

Vybrán bude vrchol ***c*** a dochází k dalšímu porovnávání.

	a	*b*	*c*	*d*	*e*	*f*	*g*				
a		3	5	4					--	--	--
b	3		3		8				3*a*	--	--
c	5	3			1	2			5*a*	5*a*	5*a*
d	4					2			4*a*	4*a*	--
e		8	1				3		∞	11*b*	11*b*
f			2	2			4		∞	∞	6*d*
g					3	4			∞	∞	∞

- *e* - dosavadní vzdálenost je 11. Velikost hrany ***ac*** je 5 a velikost hrany ***ce*** je 1. Tedy 5+1 < 11. Proto hodnota u vrcholu ***e*** bude změněna na 6*c*.
- *f* - dosavadní vzdálenost je 6. Velikost hrany ***ac*** je 5 a velikost hrany ***cf*** je 2. Tedy 5+2 > 6. Proto hodnota u vrcholu bude ponechána.

Hodnota vrcholu ***e*** a ***f*** je stejná. Vybrán bude vrchol ***e***, protože byl do fronty přidán jako první.

	a	*b*	*c*	*d*	*e*	*f*	*g*					
a		3	5	4					--	--	--	--
b	3		3		8				3*a*	--	--	--
c	5	3			1	2			5*a*	5*a*	5*a*	--
d	4					2			4*a*	4*a*	--	--
e		8	1				3		∞	11*b*	11*b*	6*c*
f			2	2			4		∞	∞	6d	6*d*
g					3	4			∞	∞	∞	∞

Jediným možným vrcholem je vrchol ***g***.

- *g* - dosavadní vzdálenost je ∞. Velikost hrany ***ac*** je 5, velikost hrany ***ce*** je 1 a velikost hrany ***eg*** je 3. Tedy 5+1+3 < ∞. Proto bude hodnota u vrcholu ***g*** změněna na 9*g*.

Vybrán bude vrchol ***f*** a dochází k dalšímu porovnávání.

	a	*b*	*c*	*d*	*e*	*f*	*g*							
a		3	5	4					--	--	--	--	--	
b	3		3		8				3*a*	--	--	--	--	
c	5	3			1	2			5*a*	5*a*	5*a*	--	--	
d	4					2			4*a*	4*a*	--	--	--	
e		8	1				3		∞	11*b*	11*b*	6*c*	--	
f			2	2			4		∞	∞	6*d*	6*d*	6*d*	
g					3	4			∞	∞	∞	∞	9*e*	

Jediným dalším možným sousedem vrcholu ***f*** je vrchol ***g***.

- *g* - dosavadní vzdálenost je 9. Velikost hrany ***ad*** je 4, velikost hrany ***df*** je 2 a velikost hrany ***fg*** je 4. Tedy 4+2+4 > 9. Proto hodnota u vrcholu bude ponechána.

	a	*b*	*c*	*d*	*e*	*f*	*g*							
a		3	5	4					--	--	--	--	--	--
b	3		3		8				3*a*	--	--	--	--	--
c	5	3			1	2			5*a*	5*a*	5*a*	--	--	--
d	4					2			4*a*	4*a*	--	--	--	--
e		8	1				3		∞	11*b*	11*b*	6*c*	--	--
f			2	2			4		∞	∞	6*d*	6*d*	6*d*	--
g					3	4			∞	∞	∞	∞	9*e*	9*e*

Tento krok byl poslední. Nejkratší cesta z vrcholu ***a*** do vrcholu ***g*** je 9 a cesta vede přes vrcholy: ***a, c, e, g***

Myšlenka Dijkstrova algoritmu je přímá. [19] Postupně je tvořena cesta pomocí expandování vrcholů. Díky ukládání vrcholů do prioritní fronty na základě vzdáleností, jsou expandovány vždy ty vrcholy, ke kterým již nemůže být nalezena kratší cesta.

4.7 Vztah Dijkstrova algoritmu s Jarníkovým algoritmem pro hledání minimální kostry

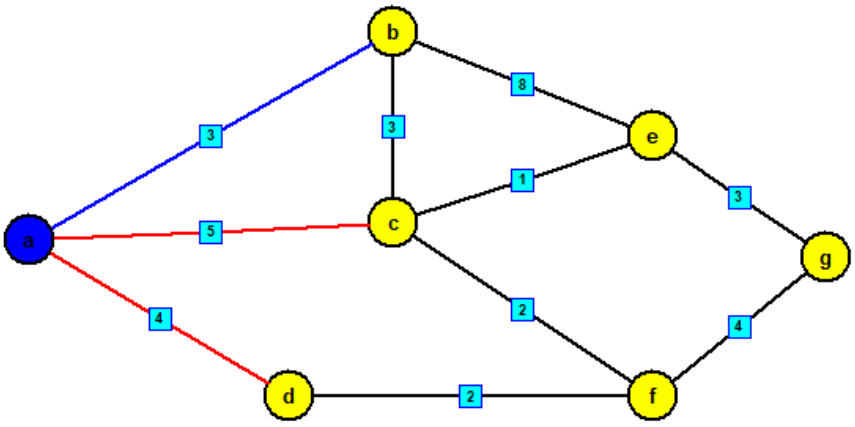
Při zabývání určitého problému, se často výzkumníci snaží o přezkoumání z více než jednoho hlediska. Pokud je to možné, diskutují o různých přístupech k jeho řešení. Na jedné straně může existovat mnoho metod, které lze použít k řešení stejného problému, zatímco na druhé straně, pomocí účinných modifikací jednoho algoritmu se můžou navrhnout metody k řešení různých dalších úkolů. [39] V článku [39] je popsán vztah mezi Dijkstrovým algoritmem a Jarníkovým algoritmem pro hledání minimální kostry.

Problém minimální kostry je založen na projití grafu všemi vrcholy s co nejmenším hranovým ohodnocením. Tento příklad lze ilustrovat na příkladech z reálného světa. Např.: rozvod elektřiny mezi městy tak, aby trasa elektrického vedení byla co nejkratší. [37]

Pro přehledné znázornění vztahu obou algoritmů bude ilustrován Jarníkův algoritmus na stejném grafu, který byl uveden v kapitole 4.5 – ilustrace Dijkstrova algoritmu na grafu. Grafy byly nakresleny v programu GrAlg [40], který byl vytvořen v rámci diplomové práce za účelem výuky.

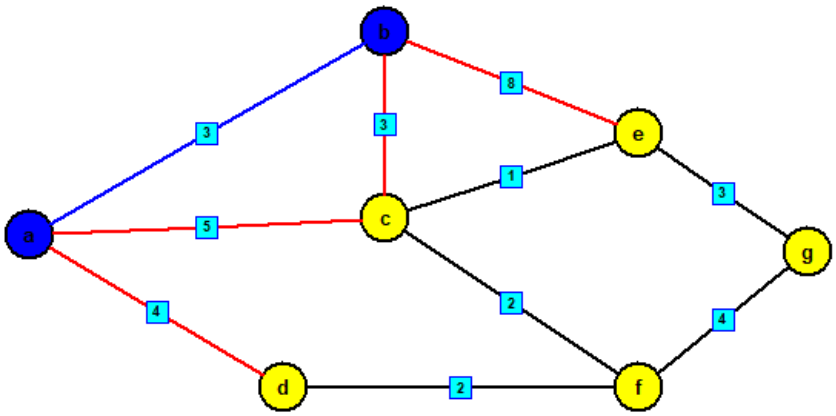
Zadání: Nechť je zadán souvislý graf G = (V, E) s *n* vrcholy a *m* hranami. Pro každou hranu *e* grafu G nechť je dáno reálné číslo *w*(e), tzv. ohodnocení hrany *e*. [39]

V prvním kroku Jarníkova algoritmu je označen startovací vrchol ***a*** a algoritmus vybírá ze všech hran, které vedou z vrcholu ***a***. Ohodnocení hrany ***ab*** = 3 a je nejmenší, a proto tato hrana bude vybrána.



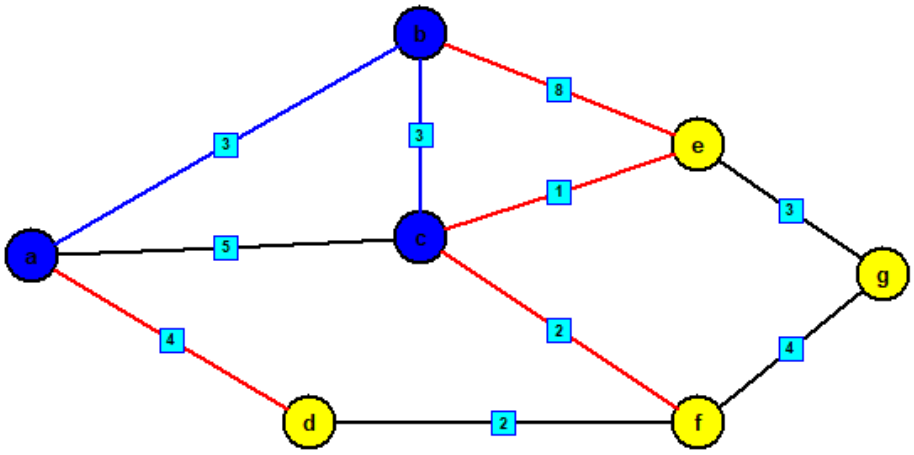
Obrázek 20: První krok Jarníkova algoritmus

Ve druhém kroku algoritmu je již označen vrchol ***b*** a algoritmus vybírá ze všech hran, které vedou z vrcholu ***a*** a ***b***. Hrana ***bc*** = 3 a je z vybíraných hran nejmenší, proto bude vybrána a vrchol ***c*** bude označen.



Obrázek 21: Druhý krok Jarníkova algoritmu

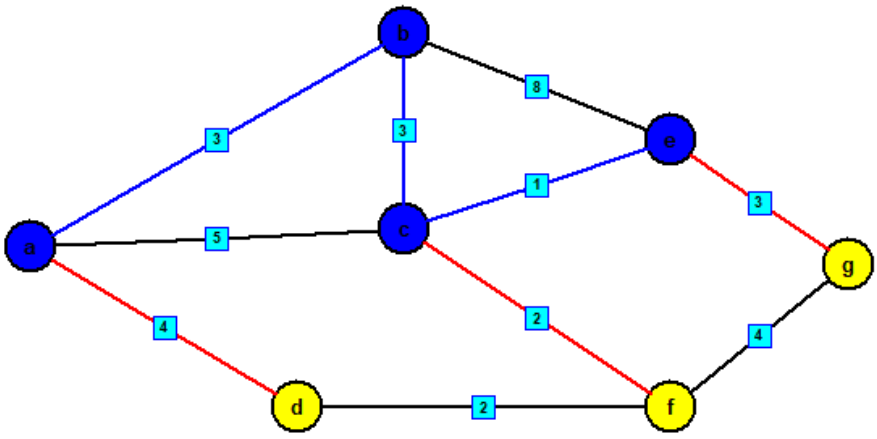
Ve třetím kroku je již označen vrchol ***c*** a algoritmus hledá nejmenší ohodnocení hrany z vrcholů ***a***,***b*** a ***c***. Není však už hledáno mezi vrcholem ***ac***, protože již hrana do vrcholu ***c*** existuje. Hrana ***ce*** = 1 je z vybíraných hran nejmenší, a proto bude vybrána a vrchol ***e*** označen.



Obrázek 22: Třetí krok Dijkstrova algoritmu

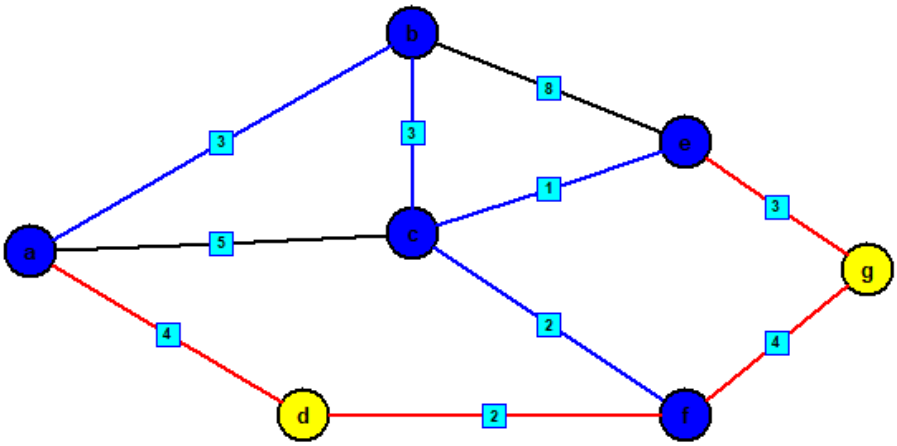
Ve čtvrtém kroku je již označen vrchol ***e*** a algoritmus pokračuje v hledání nejmenší ohodnocené hrany z vrcholů ***a***,***c*** a ***e***. Není však už hledáno mezi vrcholy ***be***, protože již

hrana do vrcholu ***e*** existuje. Hrana ***cf*** = 2 je z vybíraných hran nejmenší, a proto bude vybrána a vrchol ***f*** označen.



Obrázek 23: Čtvrtý krok Jarníkova algoritmu

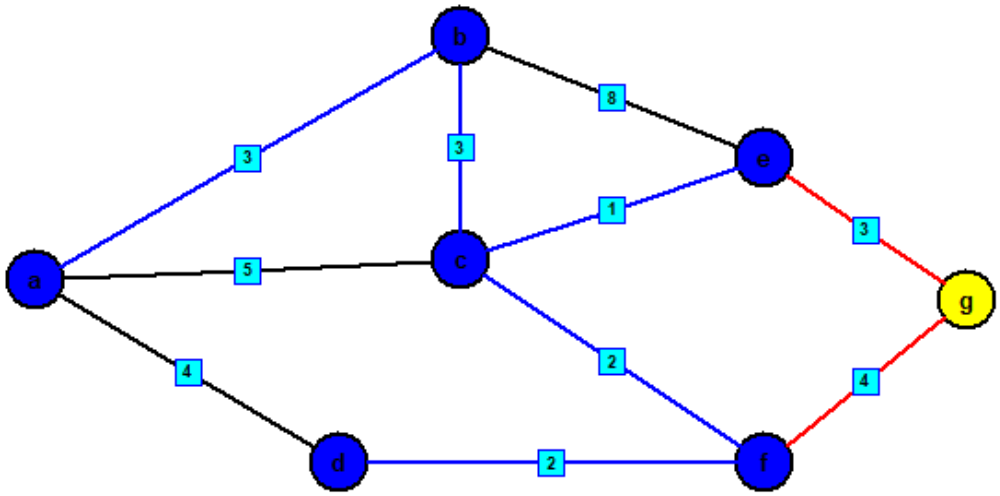
V pátém kroku je již označen vrchol ***f*** a algoritmus pokračuje v hledání nejmenší ohodnocené hrany z vrcholů ***a***,***e*** a ***f***. Hrana ***fd*** = 2 je z vybíraných hran nejmenší, a proto bude vybrána a vrchol ***d*** označen.



Obrázek 24: Pátý krok Jarníkova algoritmu

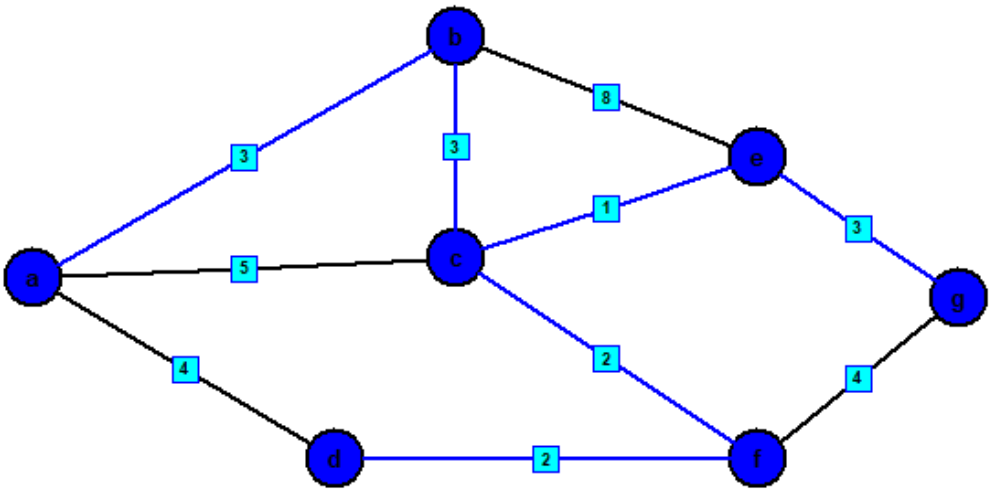
V šestém kroku je již označen vrchol ***d*** a algoritmus pokračuje v hledání nejmenší ohodnocené hrany z vrcholů ***e*** a ***f***. Není však už hledáno mezi vrcholy ***ad***, protože již hrana

do vrcholu $\boldsymbol{d}$ existuje. Hrana $\boldsymbol{eg} = 3$ je z vybíraných hran nejmenší, a proto bude vybrána a vrchol $\boldsymbol{g}$ označen.



Obrázek 25: Šestý krok Jarníkova algoritmu

Následně je algoritmu ukončen, protože minimální kostra obsahuje všechny vrcholy. Minimální kostra $= \{\boldsymbol{ab} - 3, \boldsymbol{bc} - 3, \boldsymbol{ce} - 1, \boldsymbol{cf} - 2, \boldsymbol{fd} - 2, \boldsymbol{eg} - 3\}$, cena $= 3 + 3 + 1 + 2 + 2 + 3 = 14$



Obrázek 26: Výsledná minimální kostra

Z ilustrací obou algoritmů je zcela zřejmá souvislost, která z uvedených příkladů vyplývá. V případě algoritmu E.W. Dijkstra je v každém kroku z nezařazených vrcholů vybrán ten, který je počátečnímu vrcholu nejblíže. V případě algoritmu Vojtěcha Jarníka, je vybíráno z těch nezařazených vrcholů, které jsou nejblíže přes minimální hranu.

4.8 Použití Dijkstrova algoritmu

Jak již bylo uvedeno v úvodu práce, Dijkstrův algoritmus má mnoho využití. Lze se s tímto algoritmem setkat nejen v dopravě (plánování cest na mapě), ale je i velmi rozšířen v oboru počítačových sítích, konkrétně u směrování.

Komunikace mezi jednotlivými počítačovými sítěmi je prováděna technikou zvanou routing[15]. K tomu jsou používány různé routovací protokoly. Routing je jednou ze základních částí komunikace v internetu. [22]

Routovací protokol, ve kterém je Dijkstrův algoritmus použit, se nazývá OSPF. Vytvoření protokolu OSPF (Open Shortest Path First) organizací IETF se datuje přibližně v letech 1988 -1991. Tento protokol je zařazen do skupiny směrovacích protokolů IGP[16]. Používá se v lokálních sítích, nebo je používán osobami, které zprostředkovávají internetové připojení.

OSPF patří do kategorie směrovacích protokolů zvaných Link State[17]. V paměti směrovače je tedy vytvářena kompletní mapa celé sítě, označována jako topologická databáze (neboli Link State Database). Nad touto databází se poté provádí výpočty, které jsou potřebné k nalezení nejvýhodnější cesty do jednotlivých sítí. Tyto výpočty jsou prováděny pomocí Dijkstrova algoritmu.

Ve velmi zjednodušené podobě je možné funkci protokolu OSPF popsat následovně:

1. *„Směrovač vysílá přes svá rozhraní tzv. Hello pakety. Pokud se dva navzájem propojené routery pomocí těchto paketů dohodnou na určitých společných parametrech, stávají se sousedy.*
2. *Mezi některými ze sousedů se vytvářejí užší vazby. Tyto routery se pak označují jako přilehlé.*

[15] Routing je proces výběru cesty pro provoz v síti nebo přes více sítí. [21]
[16] IGP - Interior Gateway Routing Protocols slouží k výměně směrovacích informací mezi směrovači v rámci autonomního systému.
[17] Link-state je jedním ze dvou hlavních kategorií směrovacích protokolů používaných v paketových sítích.

3. *Přilehlé routery si vzájemně vyměňují pakety obsahující LSA informace. Ty popisují stav rozhraní směrovače nebo seznam směrovačů připojených k dané síti.*
4. *Všechny směrovače si ukládají přijaté LSA do své lokální topologické databáze a zároveň je přeposílají na ostatní přilehlé směrovače. Tím se informace postupně rozšíří mezi všechny směrovače v síti. Výsledkem bude shodná topologická databáze na všech směrovačích.*
5. *Po naplnění databáze každý směrovač provede výpočet pomocí Dijkstrova algoritmu. Jeho výsledkem bude nalezení nejkratší cesty do každé známé sítě a odstranění smyček v topologii sítě.*
6. *Na základě vypočtených dat je možné naplnit směrovací tabulku routeru.* [23]
7. *Pokud dojde ke změně topologie sítě, směrovač, na kterém ke změně došlo, odešle přilehlým směrovačům informaci v podobě LSA datových položek v OSPF paketu."* [23] Ty se postupně rozšíří po celé síti a každý směrovač upraví svou topologickou databázi a provede nový výpočet Dijkstrova algoritmu. [23]

5 Implementace aplikace

Praktická část práce je věnována tvorbě a použití algoritmu v aplikaci. V aplikaci je vytvořena mapa České republiky, ve které jsou větší města znázorněna jako vrcholy a cesty mezi nimi jako hrany. Na této mapě je demonstrován Dijkstrův algoritmus. Tato kapitola popisuje použité technologie, návrh aplikace, použité komponenty a také ovládání.

5.1 Použité technologie

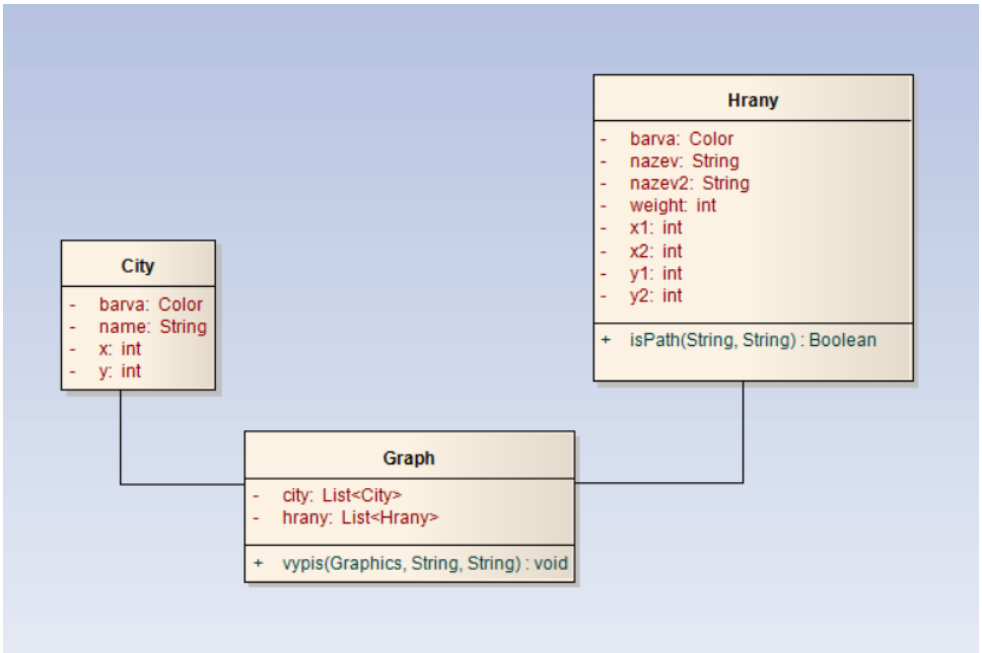
Aplikace je vytvořená v programovacím jazyku Java, ve verzi 8. Pro implementaci bylo zvoleno vývojové prostředí Eclipse. Vývojové prostředí Eclipse pochází ze světově známé společnosti IBM. V současnosti je distribuována pod licencí EPL[18] (Eclipse Public License) a v různých aspektech se odlišuje od známe licence GNU GPL. Prostředí Eclipse je postaveno na programovacím jazyku Java, to zaručuje velmi dobrou přenositelnost mezi ostatními platformami. Toto vývojové prostředí nepodporuje vizuální grafické rozhraní, ale díky různým modulům (plug-inům) je možné tuto funkci přidat. Kromě grafického

[18] EPL je open source software licence, používaná ve společnosti Eclipse.

rozhraní lze pomocí modulů do Eclipsu přidat: nové typy editorů, podporu pro další programovací jazyky, ladící nástroje a mnoho dalších funkcí. [32]

5.2 Návrh aplikace

Návrh aplikace je znázorněn na Obrázku 27. Aplikace je tvořena třídami, kterými jsou vrcholy (City) a hrany. Dohromady spolu tvoří graf.



Obrázek 27: Analytický model

Analytický model je vytvořen v programu Enterprise architekt verze 7.1. [19]

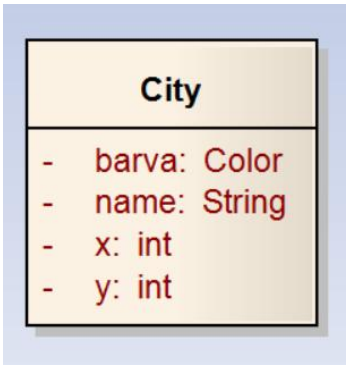
5.3 Komponenty aplikace

Hlavní komponenty v aplikaci jsou vrcholy a hrany. Vrcholy v aplikaci reprezentují města a hrany cesty mezi nimi.

[19] https://cs.wikipedia.org/wiki/Enterprise_Architect

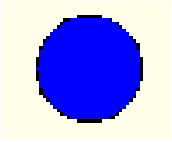
5.3.1 Vrcholy

Třída City označuje vrcholy (viz. Obrázek 28). Vrcholy jsou definovány celočíselnými proměnnými x a y, které určují jejich polohu. Vrcholy mají přiřazen název, který odpovídá názvu města.



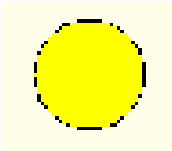
Obrázek 28: Města - vrcholy

Posledním atributem vrcholu je barva, která se mění v závislosti na tom, zda-li je vrchol označen. Pokud je vrchol označen, má barvu modrou. (Obrázek 29)



Obrázek 29: Vrchol označený

Pokud vrchol není označen, má barvu žlutou (Obrázek 30).



Obrázek 30: Vrchol neoznačený

Komponenta vrcholu je tvořena pomocí metody drawOval a fillOval, které jsou ze třídy Graphics a Graphics2D z knihovny grafických prvků java.awt. Proto jeho vykreslení není náročné. (viz. Obrázek 31)

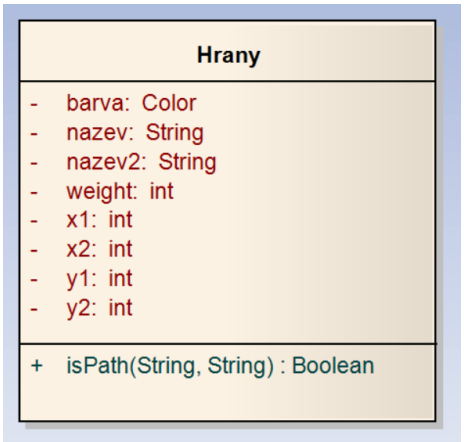
```
g.setColor(Color.BLACK);
g.drawOval(city.get(i).x-15, city.get(i).y-15, 30, 30);

g.setColor(city.get(i).barva);
g.fillOval(city.get(i).x-15, city.get(i).y-15, 30, 30);
```

Obrázek 31: Vykreslení vrcholu

5.3.2 Hrany

Třída hrany (viz. Obrázek 32) označuje spojení mezi dvěma vrcholy. Atributy ve třídě Hrany jsou podobné jako ve třídě City. Barva se mění v závislosti na tom, zda přes tuto hranu vede nejkratší cesta. Pokud ano, barva hrany je modrá, pokud ne, hrana má barvu červenou. Atributy *nazev* a *nazev2* definují, které dva vrcholy jsou touto hranou spojeny. Atribut *weight* značí váhu hrany, v tomto případě je to vzdálenost mezi dvěma vrcholy. Posledními atributy zde jsou proměnné x_1, y_1, x_2, y_2, které určují polohu hrany.



Obrázek 32: Hrany - cesty

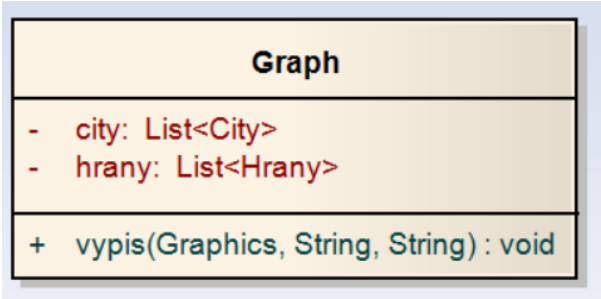
Třída Hrany také obsahuje metodu isPath (Obrázek 33), která kontroluje, zda mezi vrcholy existuje hrana.

```
public boolean isPath(String city, String city2) {
    if (nazev.equals(city)) {
        return (nazev2.equals(city2));
    }
    if (nazev.equals(city2)) {
        return (nazev2.equals(city));
    }
    return false;
}
```

Obrázek 33: Metoda isPath

5.3.3 Třída Graph

Ve třídě Graph jsou vytvořeny a uloženy všechny vrcholy a hrany. Vrcholy a hrany jsou uloženy do datové struktury, která se nazývá ArrayList, neboli dynamické pole. Toto pole je velmi výhodné z důvodu přístupu k jeho uloženým hodnotám.

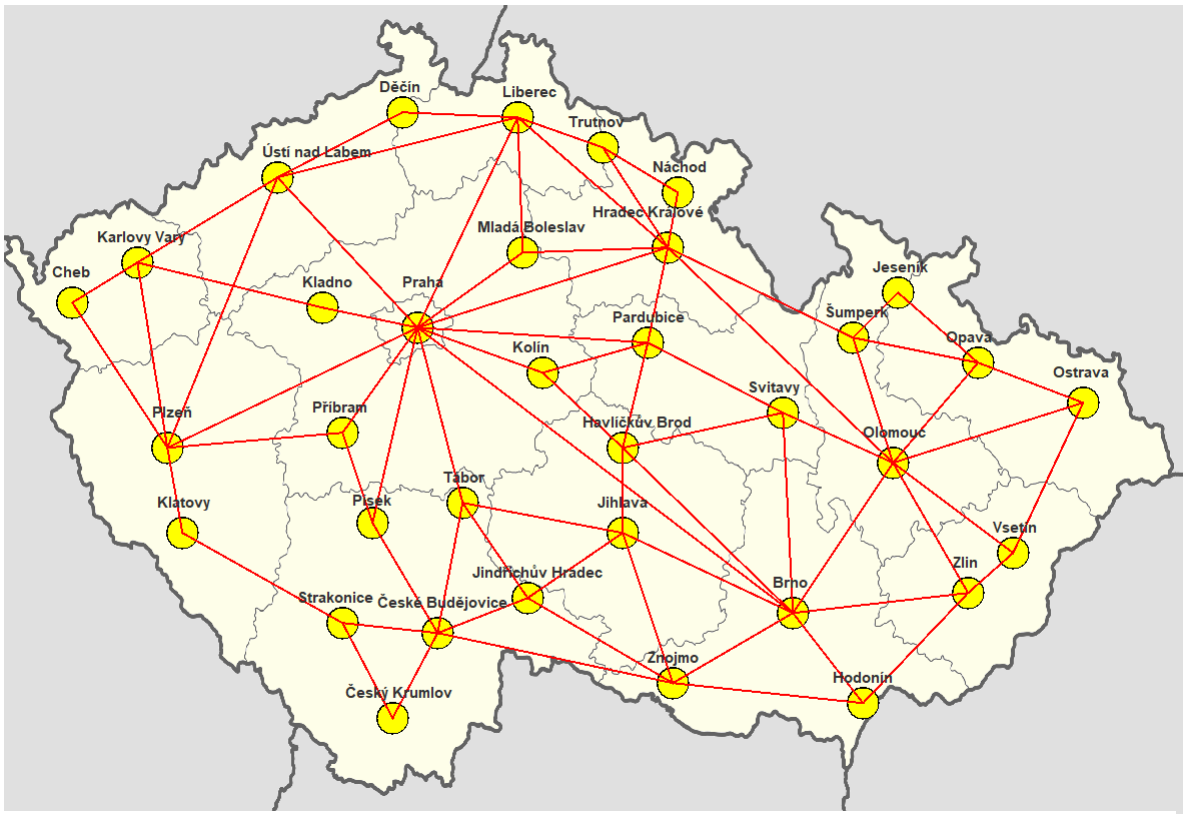


Obrázek 34: Třída Graph

Třída Graph také obsahuje metodu *vypis*, která po obdržení místa odkud a kam, spouští celý Dijkstrův algoritmus.

5.4 Funkce algoritmu

V této kapitole je popsána funkce algoritmu tak, jak je implementována v aplikaci. Aplikace je zobrazena na Obrázku 35. Prvním kliknutím na vrchol, je vybrán počáteční vrchol, od kterého se bude hledat nejkratší cesta. Druhým kliknutím na jiný vrchol je spuštěn Dijkstrův algoritmus.



Obrázek 35: Aplikace

Dijkstrův algoritmus začíná v metodě *vypis* ve třídě Graph. Nejdříve je vytvořeno dynamické pole (ArrayList) *listMest*, do kterého se budou ukládat města. Poté je spuštěn cyklus for-each, ve kterém je zavolána metoda *findPath*, s parametry měst *prvni* a *druhy*.

```
List<String> listMest = new ArrayList<>();
for (String entry : e.findPath(prvni, druhy)) {
     listMest.add(entry);
     System.out.println(entry);
}
```

Obrázek 36: Spuštění Dijkstrova algoritmu

V metodě *findPath* jsou nejdříve nastaveny vrcholy na nekonečno a ten vrchol, ze kterého bude hledána nejkratší cesta (*prvni*), se nastaví na nulu. (viz. Obrázek 37)

```
public List<String> findPath(String prvni, String druhy) {
    for (String node : getNodes()) {
        vzdalenost.put(node, Integer.MAX_VALUE);
    }
    vzdalenost.put(prvni, 0);
```

Obrázek 37: Nastavení vrcholů

Dále pak v metodě *findPath* probíhá samotný algoritmus. Ukládání předchůdců je realizováno datovou strukturou HashMap[20]. Následuje vytvoření proměnné, do které se budou ukládat aktuální vrcholy. Zatím je tato proměnná nastavena na null.

Cyklus while vrátí vrchol, který má nejmenší vzdálenost, a bude prováděn tak dlouho, dokud metoda *findNearestNode* nevrátí všechny nejbližší vrcholy. V průběhu cyklu, dojde k porovnání vzdáleností mezi nově objeveným vrcholem a vrcholem ve frontě. Jestli-že je vzdálenost (*possiblyBetterDistance*) menší, než vzdálenost vrcholu, který je ve frontě, pak dochází k vylepšení cesty a nastavení předchůdců.

Po ukončení cyklu while dojde k odstranění vrcholu z grafu a z fronty vzdáleností.

```
Map<String, String> predecessors = new HashMap<String, String>();
String currentNode = null;

while ((currentNode = findNearestNode()) != null) {
    for (String adjacentNode : findAdjacentNodes(currentNode)) {
        int possiblyBetterDistance = vzdalenost.get(currentNode)
                + distance(currentNode, adjacentNode);
        if (possiblyBetterDistance < vzdalenost.get(adjacentNode)) {
            vzdalenost.put(adjacentNode, possiblyBetterDistance);
            predecessors.put(adjacentNode, currentNode);
        }
    }
    removeNode(currentNode);
    vzdalenost.remove(currentNode);
```

Obrázek 38: Metoda findPath

Ve druhé části metody *findPath* dojde k vytvoření nového listu, ve kterém bude uložena výsledná cesta. Nejdříve je přidán cílový vrchol a poté jsou postupně přidáváni jeho předchůdci. Nakonec je metodou vrácena nejkratší cesta mezi dvěma vrcholy. (viz. Obrázek 39).

[20] https://cs.wikipedia.org/wiki/Asociativn%C3%AD_pole

```
List<String> path = new ArrayList<String>();
String predecessor;
path.add(destinationNode);
String node = destinationNode;
while ((predecessor = predecessors.get(node)) != null) {
    path.add(0, predecessor);
    node = predecessor;
}
return path;
```

Obrázek 39: Metoda findPath druhá část

Po vyhledání nejkratší cesty jsou vykresleny hrany mezi vrcholy a je zobrazen výsledek s vypočtenou vzdáleností a seznamem procházených měst.

5.5 Ovládání aplikace

Ovládání aplikace je velmi jednoduché a intuitivní. Pro spuštění aplikace je nutné mít nainstalovanou v počítači Javu (JRE). Javu je možné stáhnout z webových stránek www.oracle.com a poté nainstalovat.

Aplikace se spustí dvojklikem na jar. soubor Dijkstrův algoritmus. Následně je zobrazena mapa České republiky s městy, která jsou spojena „silnicemi“.
Prvním kliknutím na město je označeno město, odkud bude hledána nejkratší cesta. Druhým kliknutím na jiné město je vypsána do nového okna trasa a délka nejkratší cesty mezi těmito městy. Po zavření informačního okna lze pokračovat v hledání nejkratších cest mezi dalšími dvěma městy.

6. Testování aplikace

Aplikace byla testována ručně za použití testovacích dat. Byla ověřena správnost výpočtu a určení nejkratší cesty mezi dvěma městy v aplikaci.

6.1 Testovací data

Testovacími daty pro vytvořenou aplikaci je graf, jehož vrcholy představují města, která jsou spojena hranami o různých délkách.

Vzdálenost je udána v kilometrech a její hodnoty přesně neodpovídají skutečnosti. Přehled testovacích dat je zobrazen v Tabulce 1.

Počáteční město	Cílové město	Vzdálenost (km)
Plzeň	Praha	100
Plzeň	Ústí nad Labem	150
Praha	Hradec Králové	110
Praha	Liberec	80
Liberec	Hradec Králové	100
Ústí nad Labem	Liberec	90
Praha	Ústí nad Labem	80
Praha	Pardubice	100
Pardubice	Hradec Králové	30
Jihlava	Brno	90
Brno	Olomouc	75
Olomouc	Ostrava	95
Praha	Brno	205
Zlín	Olomouc	60
Hradec Králové	Olomouc	150
Brno	Zlín	95
Ústí nad Labem	Karlovy Vary	120
Karlovy Vary	Plzeň	90
Cheb	Karlovy Vary	43
Cheb	Plzeň	97
Ústí nad Labem	Děčín	90
Děčín	Liberec	70
Liberec	Trutnov	86
Trutnov	Hradec Králové	50
Praha	Mladá Boleslav	60
Mladá Boleslav	Hradec Králové	85
Praha	Kolín	60
Kolín	Pardubice	46
Plzeň	Příbram	65
Plzeň	Klatovy	22
Příbram	Praha	60
Klatovy	Strakonice	51
Strakonice	Český Krumlov	72

Český Krumlov	České Budějovice	26
Strakonice	České Budějovice	57
České Budějovice	Tábor	60
České Budějovice	Písek	50
Písek	Praha	105
Tábor	Praha	87
Příbram	Písek	51
Tábor	Jihlava	110
České Budějovice	Znojmo	135
Znojmo	Jihlava	51
Znojmo	Brno	67
Znojmo	Hodonín	93
Hodonín	Brno	60
Hodonín	Zlín	67
Zlín	Vsetín	37
Vsetín	Olomouc	75
Vsetín	Ostrava	73
Ostrava	Opava	35
Opava	Šumperk	90
Šumperk	Hradec Králové	105
Olomouc	Opava	72
Olomouc	Šumperk	54
Olomouc	Svitavy	70
Pardubice	Svitavy	68
Liberec	Mladá Boleslav	50
Brno	Svitavy	68
Náchod	Hradec Králové	45
Náchod	Trutnov	34
Karlovy Vary	Kladno	103
Praha	Kladno	27
České Budějovice	Jindřichův Hradec	54
Jihlava	Jindřichův Hradec	86
Tábor	Jindřichův Hradec	53
Znojmo	Jindřichův Hradec	96
Jihlava	Havlíčkův Brod	26
Pardubice	Havlíčkův Brod	66
Kolín	Havlíčkův Brod	63
Brno	Havlíčkův Brod	104

Svitavy	Havlíčkův Brod	80
Šumperk	Jeseník	46
Opava	Jeseník	79

Tabulka 1: Testovací data

6.2 Testování

V aplikaci bude nyní otestována implementace Dijkstrova algoritmu na několika trasách a výsledky aplikace budou porovnány s výsledky referenčních řešení.

Úloha 1: Je hledána nejkratší cesta z města České Budějovice do města Hradec Králové.

Byly vypočteny všechny možné trasy z Českých Budějovic do Hradce Králové a seřazeny dle délek. Prvních několik nejkratších cest je uvedeno níže v tabulkách. Je předpokládáno, že algoritmus nalezne tu nejkratší cestu.

1. Cesta vede z Českých Budějovic přes Tábor, Prahu do Hradce Králové. Z tabulky lze vypočítat délku trasy.

Trasa	Vzdálenost (km)	Mezisoučet (km)
České Budějovice - Tábor	60	60
Tábor - Praha	87	147
Praha - Hradec Králové	110	257

Tabulka 2: První cesta

Celkem je tedy cesta dlouhá 257 km.

2. Cesta vede z Českých Budějovic přes Jindřichův Hradec, Jihlavu, Havlíčkův Brod, Pardubice do Hradce Králové. Z tabulky lze vypočítat délku trasy:

Trasa	Vzdálenost (km)	Mezisoučet (km)
České Budějovice – Jindřichův Hradec	54	54
Jindřichův Hradec - Jihlava	86	140
Jihlava - Havlíčkův Brod	26	166
Havlíčkův Brod - Pardubice	66	232
Pardubice – Hradec Králové	30	262

Tabulka 3: Druhá cesta

Celkem je tedy cesta dlouhá 262 km.

3. Cesta vede z Českých Budějovic přes Písek, Prahu do Hradce Králové. Z tabulky lze vypočítat délku trasy:

Trasa	Vzdálenost (km)	Mezisoučet (km)
České Budějovice - Písek	50	50
Písek - Praha	105	155
Praha - Hradec Králové	110	265

Tabulka 4: Třetí cesta

Celkem je tedy cesta dlouhá 265 km.

4. Cesta vede z Českých Budějovic přes Písek, Příbram, Prahu do Hradce Králové. Z tabulky lze vypočítat délku trasy:

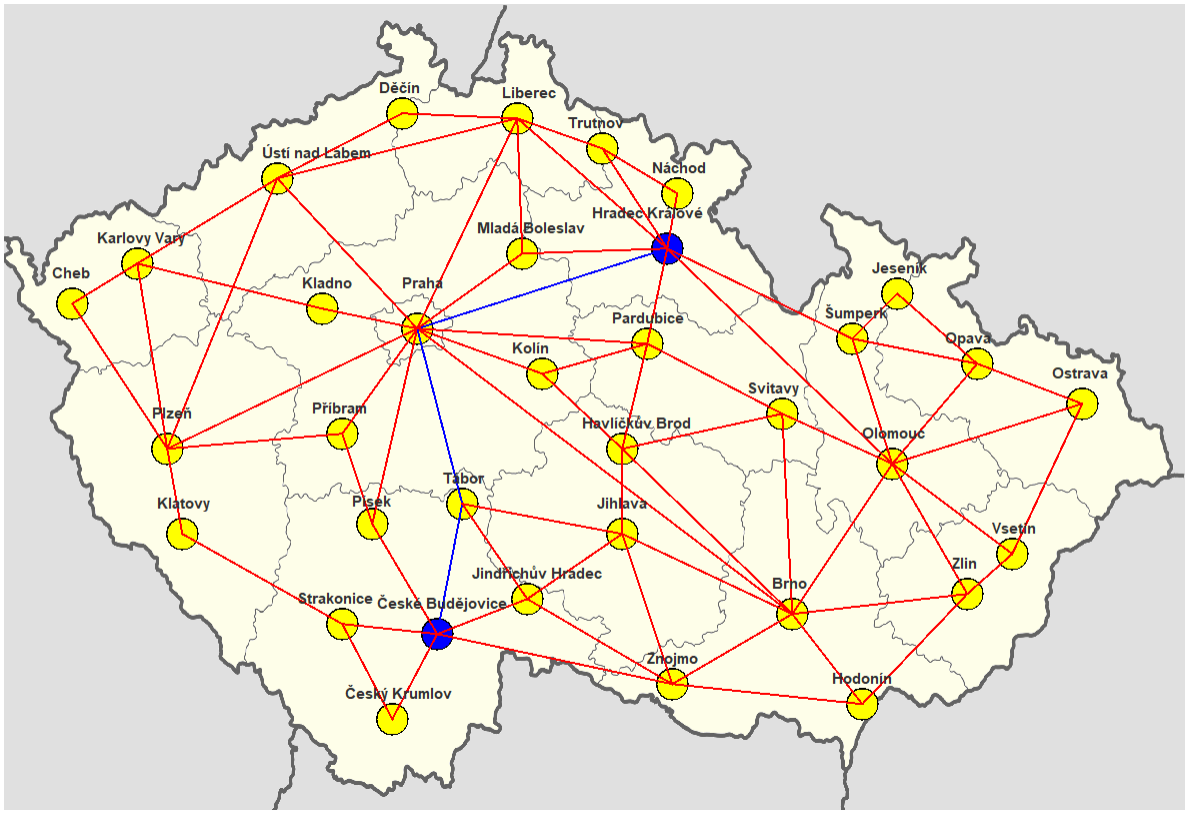
Trasa	Vzdálenost (km)	Mezisoučet (km)
České Budějovice - Písek	50	50
Písek - Příbram	51	101
Příbram - Praha	60	161
Praha - Hradec Králové	110	271

Tabulka 5: Čtvrtá cesta

Celkem je tedy cesta dlouhá 271 km.

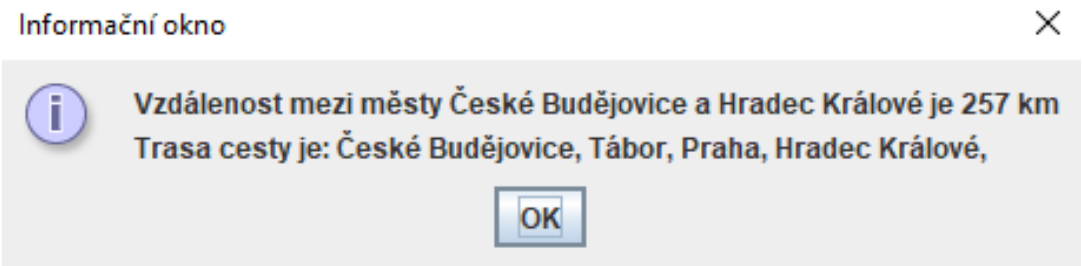
Referenční řešení: Dle první tabulky je nejkratší cesta z Českých Budějovic do Hradce Králové dlouhá 225km.

Po spuštění aplikace bylo otestováno, zda se výsledné cesty v aplikaci shodují s referenčním řešením. V aplikaci bylo zadáno startovní a cílové město. Výsledek je možné si prohlédnout na Obrázku 40. Algoritmus postupoval stejným principem, jak je uvedeno v ukázce v kapitole 4.5.



Obrázek 40: Nejkratší cesta mezi Českými Budějovicemi a Hradcem Králové

Vypočtená vzdálenost a trasa je vypsána v informačním okně na následujícím Obrázku 41.



Obrázek 41: Informační okno

Úloha 2: Je hledána nejkratší cesta z města Praha do města Hodonín.

Byly vypočteny všechny možné trasy z Prahy do Hodonína a seřazeny dle délek. První tři nejkratší cesty jsou znázorněny v tabulkách. Je předpokládáno, že algoritmus najde tu nejkratší cestu.

1. Cesta vede z Prahy přes Brno do Hodonína. Z tabulky lze vypočítat délku trasy:

Trasa	Vzdálenost (km)	Mezisoučet (km)
Praha - Brno	205	205
Brno - Hodonín	60	265

Tabulka 6: První cesta

Celkem je tedy cesta dlouhá 265 km.

2. Cesta vede z Prahy přes Kolín, Havlíčkův Brod, Brno do Hodonína. Z tabulky lze vypočítat délku trasy:

Trasa	Vzdálenost (km)	Mezisoučet (km)
Praha - Kolín	60	60
Kolín – Havlíčkův Brod	63	123
Havlíčkův Brod – Brno	104	227
Brno - Hodonín	60	287

Tabulka 7: Druhá cesta

Celkem je tedy cesta dlouhá 287 km.

3. Cesta vede z Prahy přes Tábor, Jindřichův Hradec, Znojmo do Hodonína. Z tabulky lze vypočítat délku trasy:

Trasa	Vzdálenost (km)	Mezisoučet (km)
Praha – Tábor	87	87
Tábor – Jindřichův Hradec	53	140
Jindřichův Hradec – Znojmo	96	236
Znojmo - Hodonín	93	329

Tabulka 8: Třetí cesta

Celkem je tedy cesta dlouhá 329 km.

4. Cesta vede z Prahy přes Tábor, Jihlavu, Brno do Hodonína. Z tabulky lze vypočítat délku trasy:

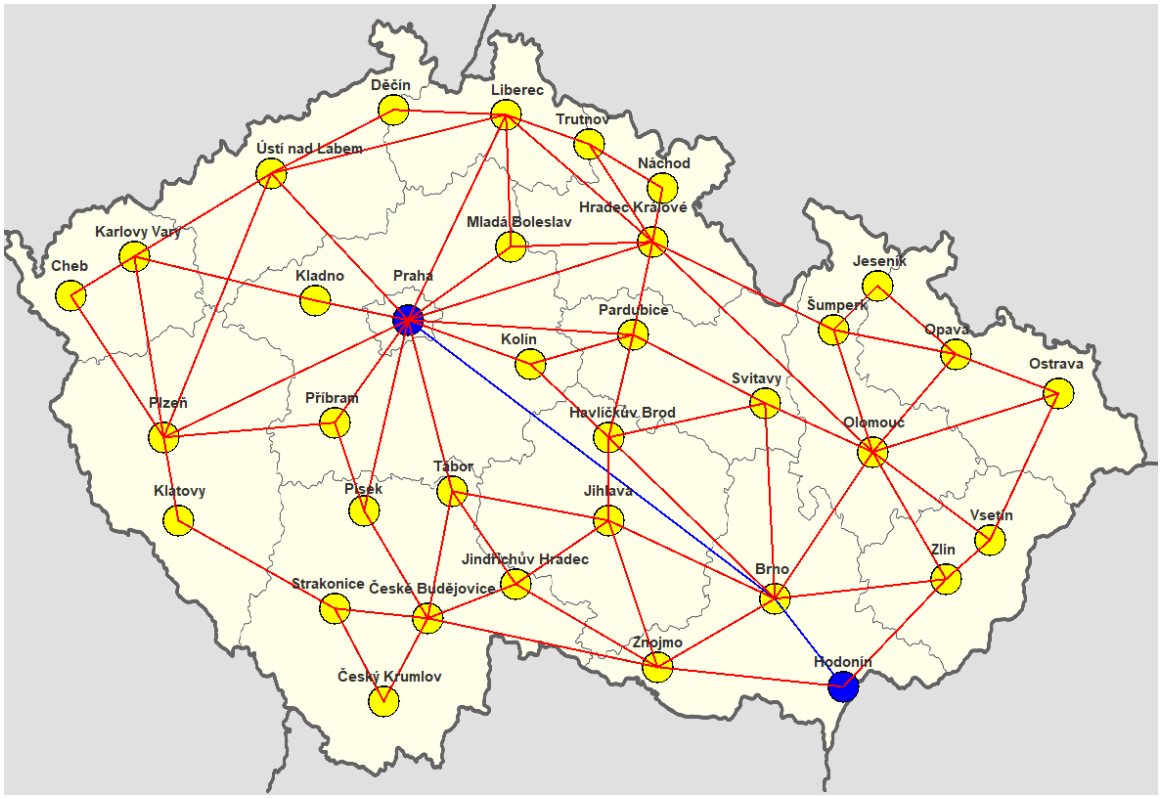
Trasa	Vzdálenost (km)	Mezisoučet (km)
Praha – Tábor	87	87
Tábor – Jihlava	110	197
Jihlava - Brno	90	287
Brno - Hodonín	60	347

Tabulka 9: Čtvrtá cesta

Celkem je tedy cesta dlouhá 347 km.

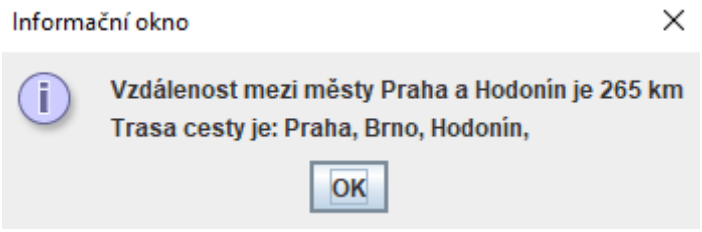
Referenční řešení: Dle první tabulky je nejkratší cesta z Prahy do Hodonína dlouhá 265 km.

Po spuštění aplikace bylo otestováno, zda se výsledné cesty v aplikaci shodují s referenčním řešením úlohy 2. V aplikaci bylo zadáno startovní a cílové město. Výsledek je možné si prohlednout na Obrázku 42. Stejně jako v prvním případě, algoritmus postupoval stejným principem, jak je uvedeno v kapitole 4.5.



Obrázek 42: Nejkratší cesta mezi Prahou a Hodonínem

Vypočtená vzdálenost a trasa je vypsána v informačním okně na následujícím Obrázku 43.



Obrázek 43: Informační okno

6.3 Vyhodnocení dat

Testování proběhlo na vzorku 60 příkladů. Ve všech případech bylo dosaženo požadovaného výsledku. Referenční řešení se shodovalo s výsledky aplikace.

7 Závěr a doporučení

Cílem bakalářské práce bylo zpracování Dijkstrova algoritmu do uceleného logického celku a vytvoření aplikace, ve které bude algoritmus použit. Nejdříve byly v práci vysvětleny základní pojmy z teorie grafů a byl představen autor tohoto algoritmu i jeho život. Dále byl popsán princip Dijsktrova algoritmu a byla ověřena jeho správnost a složitost. Ve druhé polovině teoretické části byla ukázána funkce algoritmu v grafu a na matici. Proběhlo také porovnání Dijkstrova algoritmu s Jarníkovým algoritmem. Poté bylo uvedeno použití Dijkstrova algoritmu v počítačových sítích.

V praktické části práce byla vytvořena aplikace, ve které byl algoritmus použit a otestován. V této části práce byla popsána jeho implementace, použité komponenty, seznámení s ovládáním aplikace a také vytvořena testovací data. Testování aplikace proběhlo korektně a bylo docíleno požadovaného chování aplikace. Cíl bakalářské práce byl tedy splněn. Za osobní přínos se dá pokládat bližší seznámení s autorem a jeho prací. Aplikace se může použít při výuce Dijkstrova algoritmu jako ukázka použití algoritmu v praxi.

Do budoucna je možné aplikaci rozšířit o více měst a silnic. Zajímavým rozšířením by bylo umožnit uživateli přidávat města a silnice a manipulovat s městy tak, aby se měnila jejich vzdálenost (váha) v závislosti na tom, jak daleko se nachází od jiného města spojeného „silnicí“.

8 Seznam použité literatury

[1]NEŠETŘIL Jaroslav, MATOUŠEK Jiří, Kapitoly z diskrétní matematiky. 2000. ISBN 80-246-0084-6.

[2]DEMEL Jiří, Grafy a jejich aplikace.2002.ISBN: 80-200-0990-6.

[3]JIROVSKÝ Lukáš,teorie-grafu.cz[online].[cit. 2017-03-18] Dostupné z: http://teorie-grafu.cz/uvod/historie.php

[4]PAVEL Mička, Algoritmy.net [online]. 2008-2014,[cit.2017-03-22] Dostupné z: http://www.algoritmy.net/article/1240/Algoritmus

[5]Wikipedie: Otevřená encyklopedie: Edsger_Dijkstra[online].[cit. 2017-03-22]. Dostupné z: https://cs.wikipedia.org/wiki/Edsger_Dijkstra

[6]Unsung Heroes in Dutch Computing History[online].[cit.2017-03-25]. Dostupné z: http://www-set.win.tue.nl/UnsungHeroes/heroes/dijkstra.html

[7]O'CONNOR,J.J , ROBERTSON,E.F, School of Mathematics and Statistics University of St Andrews, Scotland[online].2008,[cit.2017-03-25].Dostupné z: http://www-history.mcs.st-and.ac.uk/Biographies/Dijkstra.html

[8]KRZYSZTOF,R. Edsger Wybe Dijkstra (1930 - 2002): A Portrait of a Genius[online].2008,[cit.2017-03-25] Dostupné z: http://homepages.cwi.nl/~apt/ps/dijkstra.pdf

[9] Wikipedia:The Free Encyclopedia:Burroughs Corporation[online].[cit.2017-03-25]. Dostupné z: https://en.wikipedia.org/wiki/Burroughs_Corporation

[10] HLÍNĚNÝ Petr, Grafy[online].[cit.2017-04-14].Dostupné z: http://www.fi.muni.cz/~hlineny/Vyuka/GT/Grafy-lect--1.pdf

[11]VEČERKA Arnošt, Grafy a grafové algoritmy[online].[cit.2017-04-14]. Dostupné z: http://phoenix.inf.upol.cz/esf/ucebni/Grafy_a_grafove_algoritmy.pdf

[12]MAŠTEROVÁ Ludmila, Bakalářská práce - Teorie grafů[online].[cit.2017-04-14]. Dostupné z https://is.muni.cz/th/409120/prif_b/BP_L._Masterova.pdf

[13]RYJÁČEK Zdeněk, Skripta Diskrétní matematika[online].[cit.2017-04-14]. Dostupné z: http://www.cam.zcu.cz/~ryjacek/students/DMA/skripta/12.pdf

[14] Wikipedie: Otevřená encyklopedie: Sled [online].[cit.2017-04-14].Dostupné z : https://cs.wikipedia.org/wiki/Sled_(graf)

[15]KOVÁŘ Petr, Sled,Tah,Cesta[online].[cit.2017-04-14]. Dostupné z: http://homel.vsb.cz/~kov16/animations/sled_tah_cesta.pdf

[16] PAVEL Mička, Algoritmy.net [online]. 2008-2014,[cit.2017-04-15] Dostupné z: https://www.algoritmy.net/article/5108/Dijkstruv-algoritmus
[17] Wikipedie: Otevřená encyklopedie: Dijkstrův algoritmus [online].[cit.2017-04-15]. Dostupné z https://cs.wikipedia.org/wiki/Dijkstr%C5%AFv_algoritmus

[18]Wikisofia: Datové struktury [online].[cit.2017-04-15].Dostupné z: https://wikisofia.cz/wiki/Datov%C3%A9_struktury
[19] FUCHS, Eduard, Diskrétní matematika a teorie množin pro učitele.2000. ISBN 80-210-2463-1.

[20] Wikipedie: Otevřená encyklopedie: Paket [online].[cit.2017-04-16]. Dostupné z: https://cs.wikipedia.org/wiki/Paket
[21] Wikipedia: The Free Encyclopedia: Routing [online].[cit.2017-04-16].Dostupné z: https://en.wikipedia.org/wiki/Routing
[22]BOUŠKA Petr, Cisco Routing 3 - OSPF - Open Shortest Path First [online].[cit.2017-04-16]. Dostupné z: http://www.samuraj-cz.com/clanek/cisco-routing-3-ospf-open-shortest-path-first/

[23]GRYGÁREK Petr, SPS [online].[cit.2017-04-16]. Dostupné z: http://www.cs.vsb.cz/grygarek/SPS/lect/OSPF/ospf.html

[24] KOLÁŘ Josef, Teoretická informatika.[online].[cit.2017-04-19]. Dostupné z: http://www.exfort.org/2005l/4/x36tin/pdf/ti.pdf

[25]Wikipedie: Otevřená encyklopedie: Global Positionig systém [online].[cit.2017-04-20]. Dostupné z https://cs.wikipedia.org/wiki/Global_Positioning_System

[26] VALENTA Svatoslav, Diplomová práce - Validita přístroje GPS Polar G3 pro měření vzdáleností překonaných lokomocí člověka s aplikací do sportovních her[online].[cit.2017-04-20]. Dostupné z: http://theses.cz/id/xsd10h/DIPLOMOV_PRCE_-_VALIDITA_PSTROJE_GPS_POLAR_G3_PRO_MEN_VZD.pdf

[27]Wikipedie: Otevřená encyklopedie: Metrika sítě[online].[cit.2017-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/Metrika_s%C3%ADt%C4%9B

[28]Wikipedie: Otevřená encyklopedie:Router[online].[cit.2017-04-20]. Dostupné z: https://cs.wikipedia.org/wiki/Router

[29]ČERNÁ Ivana,Algoritmy a datové struktury: nejkratší cesty[online].[cit.2017-04-27] Dostupné z: https://is.muni.cz/el/1433/jaro2016/IB002/um/IB002_2016_slajdyV.pdf

[30]Wikipedie: Otevřená encyklopedie: Otakar Borůvka[online].[cit.2017-04-27]. Dostupné z: https://cs.wikipedia.org/wiki/Otakar_Bor%C5%AFvka

[31]Wikipedie: Otevřená encyklopedie: Vojtěch Jarník[online].[cit.2017-04-27]. Dostupné z: https://cs.wikipedia.org/wiki/Vojt%C4%9Bch_Jarn%C3%ADk

[32]TIŠNOVSKÝ Pavel, Eclipse - integrované vývojové prostředí pro Javu i další programovací jazyky[online].[cit.2017-04-27].Dostupné z: https://mojefedora.cz/eclipse-integrovane-vyvojove-prostredi-pro-javu-i-dalsi-programovaci-jazyky/

[33]Wikipedia The Free Encyclopedia: Eindhoven University of technology [online].[cit.2017-04-27].
Dostupné z: https://en.wikipedia.org/wiki/Eindhoven_University_of_Technology

[34]Wikipedia The Free Encyclopedia :Edsger W. Dijkstra[online].[cit.2017-04-27]. Dostupné z: https://en.wikipedia.org/wiki/Edsger_W._Dijkstra

[35] National Council Of Teachers Of Mathematics: Graph Creator[online].[cit.2017-04-27]. Dostupné z: http://illuminations.nctm.org/Activity.aspx?id=3550

[36]HORDĚJČUK Vojtěch, Teorie grafů [online].[cit.2018-03-30]. Dostupné z: http://voho.eu/wiki/graf/

[37] JIROVSKÝ Lukáš,teorie-grafu.cz [online].[cit. 2018-04-14] Dostupné z: http://teorie-grafu.cz/vybrane-problemy/minimalni-kostra.php

[38]PAVEL Mička, Algoritmy.net [online]. 2008-2014,[cit.2018-04-24] Dostupné z: https://www.algoritmy.net/article/102/Asymptoticka-slozitost

[39]MILKOVÁ Eva, Combinatorial optimization: mutual relations among graph algorithms, WSEAS Transactions on Mathematics, v.7 n.5, p.293-302, May 2008

[40]ŠITINA Jiří, Grafové algoritmy a jejich vizualizace. 2010. Diplomová práce. Univerzita Hradec Králové. Vedoucí práce Eva Milková.

Seznam příloh

A. Obsah CD.

A Obsah CD

K práci je přiloženo CD, na kterém je aplikace Dijkstrova algoritmu. Na CD je také dostupný zdrojový kód a textový soubor README.

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Jan Stránský**
Studium:	I1700649
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Dijkstrův algoritmus**
Název bakalářské práce AJ:	Dijkstra's algorithm

Cíl, metody, literatura, předpoklady:

Cíl práce: Cílem práce je přehledně shrnout oblast z teorie grafů - hledání nejkratších cest v grafech a Dijsktrův algoritmus. Součastí práce bude vytvoření aplikace pro nalezení nejkratších cest pomocí Dijkstrova algoritmu. Osnova: 1.Nastudovat potřebnou část z teorie grafů 2.Historie Dijkstrova algoritmu 3.Popis Dijkstrova algoritmu 4.Využití Dijkstrova algoritmu 5.Pseudokód algoritmu 6.Korektnost algoritmu 7.Složitost algoritmu 8.Tvorba aplikace 9.Dokumentace aplikace 10.Shrnutí výsledků a závěr

NEŠETŘIL Jaroslav, MATOUŠEK Jiří, Kapitoly z diskrétní matematiky. 2000. ISBN 80-246-0084-6. DEMEL Jiří, Grafy a jejich aplikace.2002.ISBN: 80-200-0990-6. Unsung Heroes in Dutch Computing History[online]. Dostupné z: http://www-set.win.tue.nl/UnsungHeroes/heroes/dijkstra.html FUCHS, Eduard, Diskrétní matematika a teorie množin pro učitele.2000. ISBN 80-210-2463-1. KRZYSZTOF,R. Edsger Wybe Dijkstra (1930 - 2002): A Portrait of a Genius[online].2008. Dostupné z: http://homepages.cwi.nl/~apt/ps/dijkstra.pdf Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein: Introduction to Algorithms, Third Edition, 2009 Massachusetts Institute of Technology JIROVSKÝ Lukáš,teorie-grafu.cz[online]. Dostupné z: http://teorie-grafu.cz/uvod/historie.php

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	RNDr. Andrea Ševčíková
Datum zadání závěrečné práce:	14.1.2015