Univerzita Pardubice
Fakulta elektrotechniky a informatiky

Výuková aplikace pro teorii grafů

Václav Hrubý

Bakalářská práce
2022

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Václav Hrubý**
Osobní číslo:	**I19091**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Výuková aplikace pro teorii grafů**
Zadávající katedra:	**Katedra informačních technologií**

Zásady pro vypracování

Cílem bakalářské práce je vytvořit aplikaci pro demonstrování vybraných algoritmů z oblasti teorie grafů. Aplikace bude schopna animovat elementární kroky algoritmů. Aplikace bude vytvořena v programovacím jazyce Java.

V teoretické části práce budou představena vybraná témata a typy úloh z teorie grafů. Dále bude proveden návrh výukové aplikace.

V praktické části bude vytvořena aplikace pro podporu výuky teorie grafů. Aplikace uživateli umožní vybrat úlohu z oblasti teorie grafů a následně bude krok po kroku animovat řešení úlohy. Jednotlivé kroky řešení budou zároveň okomentovány a vysvětleny. Složitost úloh bude definována podle vstupních parametrů. Úlohy mohou být vybírány z množiny konkrétních úloh, které budou již v aplikaci připraveny, nebo aplikace může úlohy generovat, případně může umožnit uživateli navrhnout vlastní graf a na něm poté demonstrovat algoritmus.

Rozsah pracovní zprávy: **40 stran**
Rozsah grafických prací:
Forma zpracování bakalářské práce: **tištěná**

Seznam doporučené literatury:

Volek, Josef. Operační výzkum I. Pardubice: Univerzita Pardubice, 2002. ISBN 80-7194-410-6.
Balakrishnan, V. K. Schaum's Outline of Theory and Problems of Graph Theory. New York (N.Y.): McGraw-Hill, 1997.

Vedoucí bakalářské práce: **Ing. Marie Nedvědová**
Katedra matematiky a fyziky

Datum zadání bakalářské práce: **17. prosince 2021**
Termín odevzdání bakalářské práce: **13. května 2022**

L.S.

Ing. Zdeněk Němec, Ph.D. v.r.
děkan

Ing. Jan Panuš, Ph.D. v.r.
vedoucí katedry

V Pardubicích dne 28. února 2022

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 2. 4. 2022

Václav Hrubý

PODĚKOVÁNÍ

Chtěl bych poděkovat Ing. Marii Nedvědové za vedení mé bakalářské práce, cenné rady a podporu při tvorbě této práce. Dále bych chtěl také poděkovat svým rodičům za podporu a trpělivost během studia.

ANOTACE

Cílem bakalářské práce je vytvořit aplikaci pro demonstrování vybraných algoritmů z oblasti teorie grafů. Aplikace bude schopna animovat elementární kroky algoritmů. Aplikace bude vytvořena v programovacím jazyce Java. V teoretické části práce budou představena vybraná témata a typy úloh z teorie grafů. Dále bude proveden návrh výukové aplikace. V praktické části bude vytvořena aplikace pro podporu výuky teorie grafů. Aplikace uživateli umožní vybrat úlohu z oblasti teorie grafů a následně bude krok po kroku animovat řešení úlohy. Jednotlivé kroky řešení budou zároveň okomentovány a vysvětleny. Složitost úloh bude definována podle vstupních parametrů. Úlohy mohou být vybírány z množiny konkrétních úloh, které budou již v aplikaci připraveny, nebo aplikace může úlohy generovat, případně může umožnit uživateli navrhnout vlastní graf a na něm poté demonstrovat algoritmus.

KLÍČOVÁ SLOVA

Teorie grafů, výuková aplikace, Dijkstrův algoritmus, Eulerovský tah, uzel

TITLE

Educational application for graph theory

ANNOTATION

The aim of this bachelor thesis is to create an application to demonstrate chosen algorithms from the graph theory field. The application will be able to animate elementary steps of algorithms. The application will be created in Java programming language. The chosen topics and types of tasks will be introduced in the theoretic part of the work. In the practical part, an application will be created to support the teaching of graph theory. The application will let the user choose a task from graph theory field and then will animate step by step the solution of the task. Each step of the solution will be commented and explained. The difficulty of the tasks will be defined by the input parameters. The tasks will be chosen from a set of specific tasks, that will already be prepared in the application, or the application can generate the tasks and even let the user propose his own graph and demonstrate the algorithm on it.

KEYWORDS

Graph theory, educational application, Dijkstra's algorithm, Eulerian path, node

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

API	Application Programming Interface
CPM	Critical Path Method
DB	Database
GPS	Global Positioning Systém
GUI	Graphic User Interface
ID	Identification
JAR	Java Archive
JDBC	Java Database Connectivity
JRE	Java Runtime Enviroment
OSPF	Open Shortest Path First
SQL	Structured Query Language

ÚVOD

Předmět teorie grafů je jedním z důležitých předmětů při studiu informačních technologií. Znalosti z tohoto předmětu se v oblasti informačních technologií hojně využívají a mají i další využití v reálném světě.

Tato práce popisuje vybrané algoritmy z teorie grafů. Například algoritmus nalezení nejkratší cesty nám umožňuje efektivněji naplánovat naši trasu, abychom dorazili do cílové destinace co možná nejrychleji a nejlevněji. Tento algoritmus má také využití v počítačových sítích v dynamickém směrovacím protokolu OSPF.

Dále se práce věnuje třeba algoritmu výpočtu kritické cesty, který se hojně využívá v managementu při plánování projektů. Umožňuje nám mimo jiné stanovit kritické činnosti, jejichž případné zpoždění by způsobilo zpoždění dokončení celého projektu.

Cílem této práce bylo vytvořit aplikaci pro studenty jako další podpůrný podklad pro studium předmětu teorie grafů. Aplikace je napsána v programovacím jazyce Java z důvodu jeho popularity na trhu a jednoduchosti spuštění aplikace na různých zařízeních. Aplikace umožňuje – podle typu zadání – buď jednotlivé úlohy přímo generovat, nebo již připravené úlohy načítat z databáze. Tyto úlohy jsou uloženy v databázi SQLite

Hlavním cílem bylo vytvořit aplikaci s jednoduchým a intuitivním ovládáním, která je schopna velmi jednoduše vysvětlit vybrané algoritmy s využitím ukázky postupu algoritmu krok po kroku, kde každý z elementárních kroků bude graficky a slovně vysvětlen. V případě, že se uživatel rozhodne úlohu vypracovat samostatně a potřebuje si pouze ověřit správnost výsledku, je možné si v aplikaci zobrazit pouze výsledek algoritmu bez postupu řešení.

V teoretické části jsou přiblíženy základní pojmy z teorie grafů. Dále jsou zde přiblíženy všechny typy úloh obsažené v aplikaci, včetně ukázky typových úloh, na kterých je daný algoritmus předveden a okomentován krok po kroku.

1 ZÁKLADNÍ POJMY TEORIE GRAFŮ

Teorie grafů se svými rozmanitými aplikacemi v přírodních a společenských vědách, a zvláště v teoretické informatice, se stává důležitou součástí učebních osnov matematiky na vysokých školách a univerzitách. [1]

Teorie grafů je oproti ostatním matematickým disciplínám velice mladá. Zatímco počátky aritmetiky nebo geometrie se datují již hluboko před počátkem letopočtu, počátek teorie grafů lze stanovit přesně, a to v 30. letech 18. století. V roce 1736 švýcarský a později pruský matematik Leonhard Euler vyřešil zvaný „Problém sedmi mostů města Královce“. Při řešení tohoto problému ale nevyužil žádné předem známe metody z jiných disciplín matematiky, ale použil, jak sám označil, „geometrii pozic“, dnes bychom tomu dali název teorie grafů. Za zakladatele teorie grafů tedy považujeme právě jeho. První učebnici teorie grafů vydal maďarský matematik Dénes Kőnig až o 200 let později, v roce 1936. V dnešní době se teorie grafů využívá při řešení mnoha praktických úloh. Hlavní výhodou teorie grafů je snadná a intuitivní možnost modelovat reálnou situaci jako množinu objektů (vrcholy grafu) a vztahy mezi nimi (hrany grafu). Převedeme-li konkrétní úlohu právě do tohoto tvaru, často se jedná o již řešený obecný problém, pro jehož řešení lze použít již známé algoritmy z jiných odvětví matematiky. Další výhodou je také jednoduché vytvoření dané úlohy pomocí různých aplikací, které tvorbu grafů značně zjednoduší. [2]

1.1 Graf

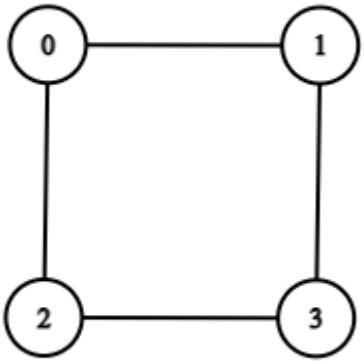
Tento pojem je jedním z ústředních pojmů současné diskrétní matematiky. V diskrétní matematice se graf definuje jako algebraická struktura, která přehledně popisuje objekty a vztahy mezi nimi. Graf si lze představit jednoduše jako několik objektů (vrcholy) a jejich spojnice (hrany). Toto přináší velkou výhodu grafů, a to přehledné a intuitivní znázornění, stačí totiž vrcholy nakreslit jako body v rovině a hrany jako přímky spojující příslušné vrcholy. Tento způsob vynaložení definice grafu nám při pečlivém rozboru úloh a implementace algoritmů ale stačit nebude. Proto zavedeme následující definici, která bude již pro tyto problémy dostačující: objekty a jejich vazby. Jestliže mezi dvěma objekty je vazba, popíšeme ji jako dvojici (dvouprvkovou množinu) příslušných objektů (vrcholů). Naopak, jestliže mezi dvěma objekty vazba není, příslušnou dvojici objektů do množiny hran nezařadíme. [2]

Definice 1: „Graf G je uspořádaná dvojice (V, E), kde V je nějaká neprázdná množina a E je množina dvoubodových podmnožin množiny V. Prvky množiny V se jmenují vrcholy grafu G a prvky množiny E hrany grafu G.“ [2]

1.2 Typy grafů

1.2.1 Neorientovaný graf

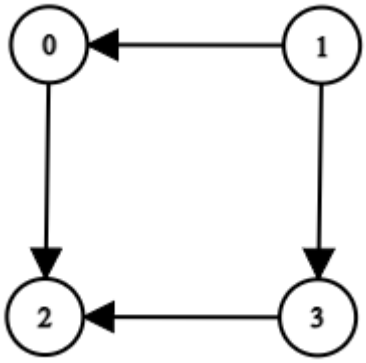
Definice 2: „Neorientovaný graf je trojice G = (V, E, ε) tvořená konečnou množinou V, jejíž prvky nazýváme vrcholy, konečnou množinou E, jejíž prvky nazýváme neorientovanými hranami a zobrazením ε (vztah incidence), které přiřazuje každé hraně e jedno nebo dvouprvkovou množinu vrcholů.“ [3]



Obrázek 1: Neorientovaný graf

1.2.2 Orientovaný graf

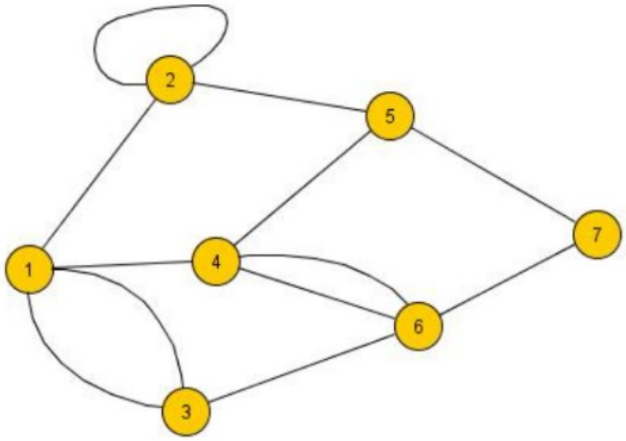
Definice 3: „Orientovaný graf je trojice $G = (V, E, \varepsilon)$ tvořená konečnou množinou prvků V, jejíž prvky nazýváme vrcholy, konečnou množinou E, jejíž prvky nazýváme orientovanými hranami a zobrazením $\varepsilon : E \to V^2$, které nazýváme vztahem incidence, a které přiřazuje každé hraně $e \in E$ uspořádanou dvojici vrcholů.“ [3]



Obrázek 2: Orientovaný graf

1.2.3 Multigraf

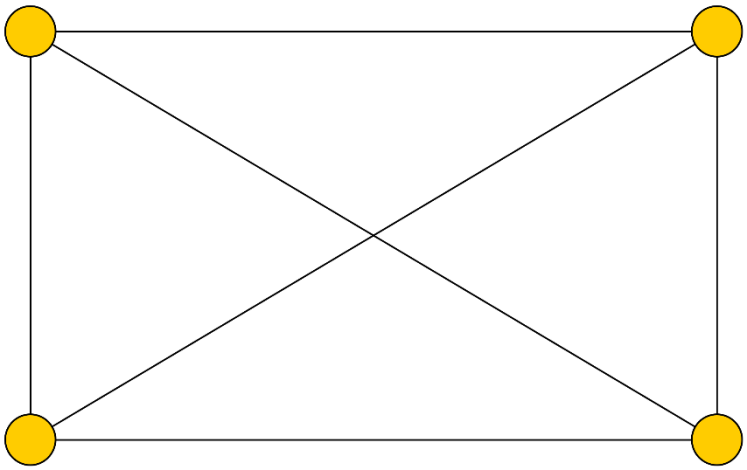
Připouští existenci dvou různých hran spojujících stejné vrcholy (nazýváme rovnoběžné hrany) i existenci hrany, která spojuje vrchol *v* s vrcholem *v* (nazývá se smyčka). Často je množina hran definována jako množina neuspořádaných dvojic vrcholů. V tom případě se incidence vypouští. Nelze pak ale definovat existenci rovnoběžných hran. Grafy bez rovnoběžných hran se nazývají prosté grafy, grafy bez smyček i rovnoběžných hran se nazývají obyčejné grafy [4]



Obrázek 3: Multigraf [4]

1.2.4 Úplný graf

Úplný graf je takový graf, kde jsou všechny vrcholy vzájemně propojeny hranou. [4]



Obrázek 4: Úplný graf

1.2.5 Souvislý graf

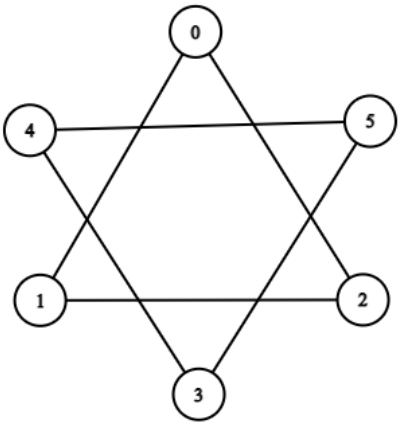
Souvislým grafem se nazývá takový graf, mezi jehož libovolnými dvěma vrcholy existuje sled (posloupnost vrcholů a hran). U orientovaných grafů rozlišujeme silnou, orientovanou a neorientovanou souvislost. [5]

Silně souvislý je graf právě tehdy, když v něm pro každou dvojici vrcholů *a*, *b* existuje orientovaná cesta *z a* do *b* a nazpět z *b* do *a*. [5]

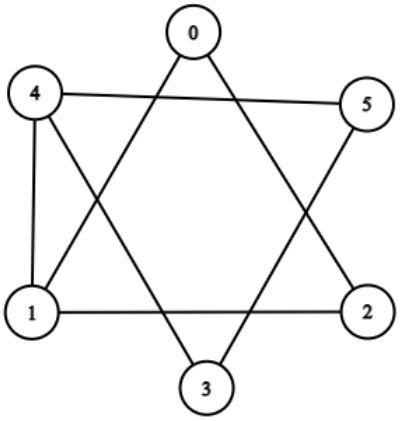
Mosty jsou hrana, které vytvoří podgraf s více propojenými komponentami, když jsou odstraněny z grafu, artikulace jsou vrcholy, které po odstranění vytvářejí podgraf s více spojitými komponentami z grafu. [33]

Orientovaně souvislý je graf tehdy, když pro každé dva vrcholy *a* a *b* existuje alespoň jedna z orientovaných cest, buď z *a* do *b* nebo z *b* do *a*.. Neorientovaně souvislý je graf takový, že

pokud z orientovaného grafu vytvoříme graf neorientovaný (budeme ignorovat směr orientace hran), tak mezi každými dvěma vrcholy *a* a *b* existuje sled. [33]



Obrázek 6: Nesouvislý graf
(inspirováno ze zdroje [5])



Obrázek 5: Souvislý graf
(inspirováno ze zdroje [5])

1.3 Sled, cesta, tah

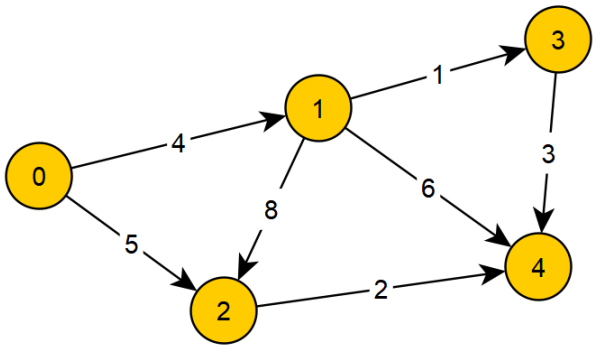
Sled je posloupnost vrcholů a hran. Pokud je počáteční vrchol shodný s koncovým, nazývá se sled uzavřený. V ostatních případech jde o sled otevřený. Sled má i orientovanou variantu, která respektuje orientaci hran. [6]

Tah je takový sled, ve kterém se neopakují žádné hrany. Tah může existovat i v orientovaném grafu. [6]

Cestu v grafu můžeme chápat jako posloupnost vrcholů a hran, kde se neopakuje žádný vrchol ani hrana. [5]

1.4 Síť

Definice 4: „Síť je hranově ohodnocený orientovaný graf G = (V, E), kde V je množina vrcholů rozdělených na zdroj, stok a vnitřní vrcholy a E je množina hran, které jsou ohodnoceny kapacitou a tokem.“ [7]



Obrázek 7: Síť

Rovinnou síť definujeme jako zobrazení, ve kterém jsou vrcholy znázorněny jako různé body v rovině a hrany jako jednoduché křivky spojující body svých koncových vrcholů. Přitom hrany se nesmí nikde křížit ani procházet jinými vrcholy. Jen málokteré grafy disponují rovinným zakreslením, ale mít rovinně nakreslený graf není důležité jen z hlediska estetiky. Například při hledání maximálního toku v síti je nutné použít rozdílný algoritmus pro rovinnou síť a pro všeobecnou síť. [8]

Kapacita hrany u sítě je funkce c: E → R0 +, to znamená, že každá hrana je ohodnocena nezáporným reálným číslem. Takové ohodnocení získáváme společně s grafem. Pro lepší pochopení si můžeme pod kapacitou hrany představit maximální propustnost kabelu u datové sítě. [7]

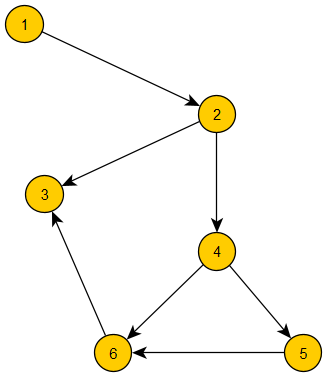
Definice 5: „Tok v síti S = (G, z, s, w) je funkce f : E(G) → R_0+ splňující

– ∀e ∈ E(G) : 0 ≤ f(e) ≤ w(e),

– ∀v ∈ V (G), v ≠ z, s : $\sum_{e \to v} f(e) = \sum_{e \leftarrow v} f(e)$. „ [9]

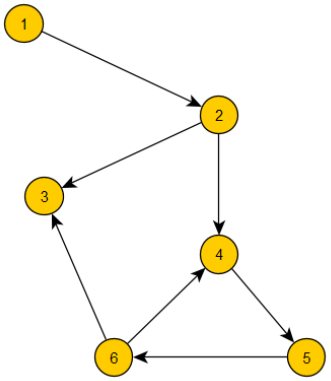
1.5 Cyklicita grafu

Cyklický graf je graf, který obsahuje alespoň jednu kružnici. O grafu, který není cyklický, se říká, že je acyklický. [10]



Obrázek 8: Acyklický graf (inspirováno ze zdroje [11])

Na obrázku můžete vidět příklad acyklického grafu (neobsahuje ani jednu kružnici).



Obrázek 9: Cyklický graf (inspirováno ze zdroje [11])

Ovšem pouhou změnou směru hrany vedoucí z vrcholu 4 do vrcholu 6 se z grafu stane graf cyklický, protože nám v grafu vznikne kružnice 4, 5, 6. Pro ověření, zda je graf cyklický, nebo není, lze použít jednoduchý algoritmus, který spočívá v tom, že odebíráme z grafu vždy jeden z jeho listů (vrchol ze kterého se již není možné dostat po hraně do žádného jiného vrcholu, v našem grafu by se tedy jednalo o vrchol 3). V tomto pokračujeme, dokud v grafu již není žádný list. Pokud jsme úspěšně odebrali všechny vrcholy, je graf acyklický, pokud nám nějaké vrcholy zbyly, tak je graf cyklický. [11]

2 POUŽITÉ ALGORITMY

2.1 Dijkstrův algoritmus

Dijkstrův algoritmus je nejefektivnějším algoritmem používaným k nalezení nejkratší cesty mezi známým vrcholem k ostatním vrcholům. Problém nalezení nejkratší cesty ze zadaného vrcholu *s* do našeho známého vrcholu *t* lze formulovat následovně: [12]

Dijkstrův algoritmus rozděluje vrcholy grafu do dvou skupin, na vrcholy s trvalým (definitivním, finálním) ohodnocením a dočasným (nedefinitivním) ohodnocením. Ohodnocením vrcholu zde rozumíme délku cesty (tedy součet ohodnocení hran na cestě) od počátku k danému vrcholu. Trvalé ohodnocení je takové, které už nebudeme měnit, protože o něm víme, že lepšího ohodnocení již nemůžeme dosáhnout. [12]

Algoritmus:

Celý algoritmus se dá shrnout do tří kroků (ohodnocení cesty do vrcholu X budeme značit |X|

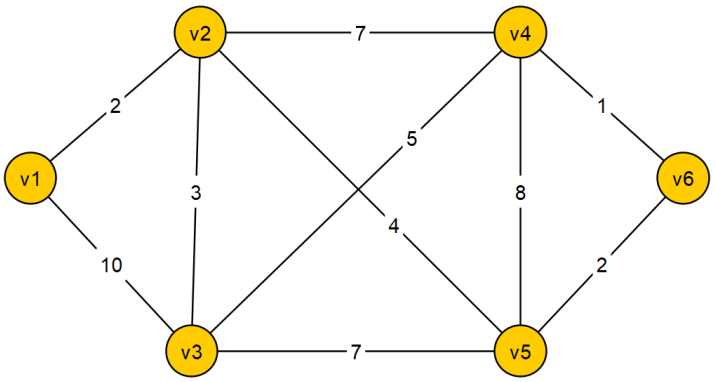
(1) nalezení vrcholu s minimálním dočasným ohodnocením (nazvěme ho M)

(2) prohlášení vrcholu M za trvalý

(3) změna ohodnocení sousedů tak, že |S| = min (|S|, |M| + ohodnocení hrany z M do S), kde S je soused M. [12]

2.1.1 Ukázka Dijkstrova algoritmu

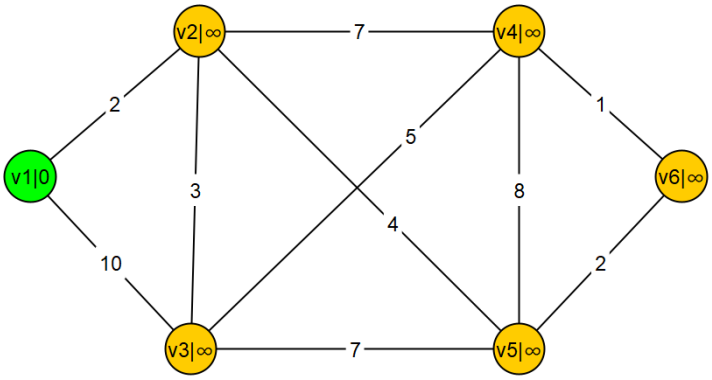
Kroky řešení Dijkstrova algoritmu si ukážeme na následujícím grafu:



Obrázek 10: Graf pro Dijkstrův algoritmus

Jako počáteční vrchol zvolíme vrchol „v1“ a konečný vrchol zvolíme vrchol „v6“.

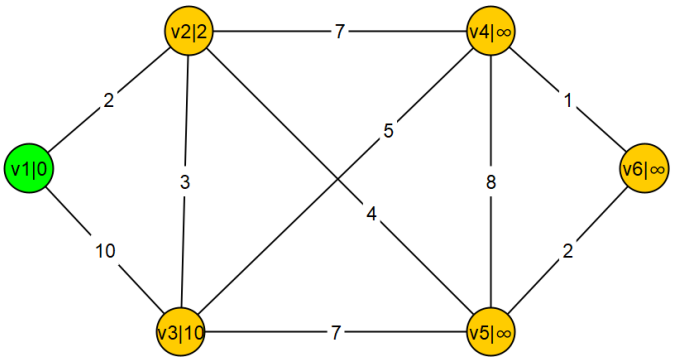
V prvním kroku ohodnotíme počáteční vrchol hodnotou 0 a trvale ho ohodnotíme, v našem případě se jedné o již zmiňovaný vrchol „v1“. Zároveň všechny ostatní vrcholy dočasně



Obrázek 11: První krok Dijkstrova algoritmu

ohodnotíme hodnotou nekonečno. Trvale ohodnocené vrcholy v grafu znázorníme pomocí zelené barvy. Ohodnocení vrcholů je v grafu zapsáno za svislou čárou hned za pojmenováním vrcholu.

V dalším kroku procházíme všechny sousední vrcholy našeho nově ohodnoceného vrcholu „v1“, které ještě nemají trvalé ohodnocení a vždy kontrolujeme, zda aktuální ohodnocení vrcholu není větší než ohodnocení, kterého dosáhneme, pokud sečteme ohodnocení hrany a ohodnocení vrcholu, ze kterého hrany vychází („v1“). Pokud je naše vypočtené ohodnocení menší než aktuální ohodnocení vrcholu, tak toto ohodnocení přepíšeme. Pokud je vypočtené ohodnocení větší, necháme původní ohodnocení.

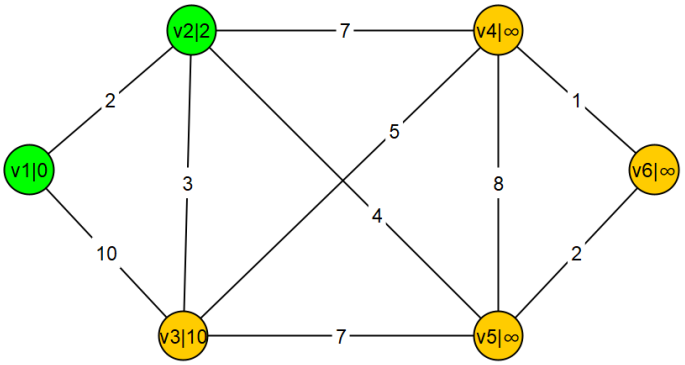


Obrázek 12: Druhý krok Dijkstrova algoritmu

Jak vidíme na obrázku nahoře, tak se nám změnila ohodnocení dvou vrcholů, a to vrcholu „v2“ a vrcholu „v3“.

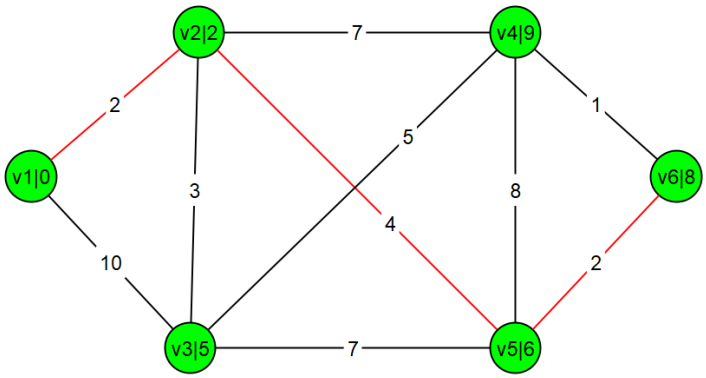
Dále projdeme všechny ještě trvale neohodnocené vrcholy (v našem příkladu se jedná o všechny žluté vrcholy) a z těchto vrcholů vybereme vrchol, který má nejmenší ohodnocení. V našem příkladu se jedná o vrchol „v2“. Tento vrchol označíme jako trvale ohodnocený.

Zbytek algoritmu se skládá z opakování předchozích dvou kroků. Ukončení algoritmu můžeme



Obrázek 13: Třetí krok Dijkstrova algoritmu

provést, pokud nastane naše stanovená podmínka. V tomto ohledu máme na výběr ze dvou možností. Můžeme ukončit provádění algoritmu již v momentě, kdy se trvale ohodnotí náš cílový vrchol, nebo můžeme algoritmus provádět, dokud trvale neohodnotíme všechny vrcholy.



Obrázek 14: Nejkratší cesta z "v1" do "v6"

Na obrázku je vidět nejkratší cesta z „v1“ do „v6“, na grafu z našeho příkladu označena červenou barvou. Je nutno podotknout, že byla zvolena podmínka ukončení algoritmu jako „ohodnocení všech vrcholů“, proto je na obrázku ohodnocen i vrchol „v4“, který by v případě zvolení podmínky „vykonávat, dokud nebude definitivně ohodnocen koncový vrchol“ ohodnocen nebyl.

2.1.2 Využití

Dijkstrův algoritmus se využívá v mnoha různých odvětvích, mezi ty nejznámější využití patří například:

Digitální mapové služby v Mapách Google: Mezi naším počátečním bodem a destinací totiž existují různé trasy/cesty, které je spojují. My však potřebujeme znát nejkratší (případně nejrychlejší) vzdálenost. Proto se k nalezení minimální vzdálenosti mezi dvěma místy používá Dijkstrův algoritmus (při hledání nejrychlejší trasy se jako ohodnocení hran v algoritmu používá čas potřebný k projetí jednotlivých hran – silnic). [13]

Aplikace sociálních sítí: Mnoho aplikací nabízí seznam přátel, které může daný uživatel znát. K výběru těchto přátel se také používá Dijkstrův algoritmus. [13]

OSPF: jedná se o směrovací protokol se stavem spojení, který se používá k nalezení nejlepší cesty mezi zdrojem a cílovým směrovačem pomocí vlastního protokolu Shortest Path First. Dijkstrův algoritmus je široce používán ve směrovacích protokolech, které vyžadují směrovače k aktualizaci své tabulky předávání. Algoritmus poskytuje nejkratší nákladovou cestu od zdrojového směrovače k ostatním směrovačům v síti. [13]

Letová agenda: Pokud například někdo potřebuje software pro vytváření agendy letů pro zákazníky. Agent má přístup k databázi se všemi letišti a lety. Kromě čísla letu, výchozího a cílového letiště je u letů uveden čas odletu a příletu. Konkrétně chce agent určit nejbližší čas příletu do cílové destinace vzhledem k výchozímu letišti a času odletu. Zde přichází ke slovu tento algoritmus. [13]

Robotická cesta: V dnešní době se začínají velice hojně používat roboti, z nichž jsou někteří plně automatičtí. Například robotické restaurace používají Dijkstrův algoritmus, aby naleznuli nejkratší cestu k zadaným stolům. [13]

2.2 Maximální dráha

Algoritmus maximální dráhy lze použít pouze na grafech, které jsou orientované, neorientovaně souvislé, hranově ohodnocené a acyklické. Dále musí splňovat nějaké podmínky také vrcholy,

které si stanovíme jako počátek a konec algoritmu. Počáteční vrchol nesmí mít prázdnou množinu hran z něho vystupujících, toto zamezí výběru vrcholu, ze kterého by nebyla možnost se dostat do žádných dalších vrcholů. Pro koncový vrchol musí platit, že existuje minimálně jedna hrana, ze které je možno se do vrcholu dostat. [14]

Algoritmus na určení maximální dráhy:

(1) Ohodnotíme všechny vrcholy hodnotou 0.

(2) Počáteční vrchol (v_0) vložíme do množiny trvale (definitivně) ohodnocených vrcholů.

(3) Vybereme takový vrchol, který ještě není trvale ohodnocený, ale všichni jeho předchůdci již trvale ohodnocení jsou. Tento vrchol tedy patří do vstupní množiny sousedů Γ+ jiného trvale ohodnoceného vrcholu (je možné se do něho dostat z nějakého již trvale ohodnoceného vrcholu) a zároveň nepatří do vstupní množiny sousedů Γ+ žádného dočasně ohodnoceného vrcholu (není možné se do něho dostat z nějakého ještě nedefinitivně ohodnoceného vrcholu).

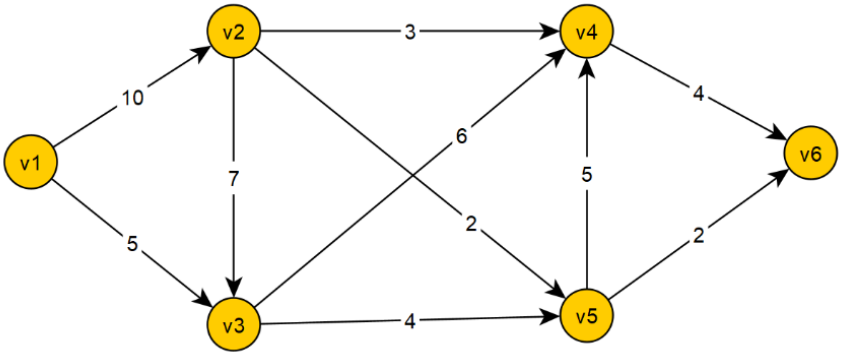
(4) Vrchol trvale ohodnotíme tak, že, porovnáme hodnotu původního ohodnocení vrcholu a hodnotu součtu ohodnocení předchůdce s ohodnocením incidující hrany (pokud je těchto hran více je nutné projít všechny), a z těchto hodnot zvolíme tu větší.

(5) Pokud ještě nejsou všechny vrcholy trvale ohodnoceny, pokračujeme znovu v kroku 3.

(6) Rekonstruujeme cestu tím, že od koncového vrcholu pokračujeme po hranách zpět k počátku, vždy tak, aby se ohodnocení následníka rovnalo součtu ohodnocení předchůdce s ohodnocením incidující hrany. [15]

2.2.1 Ukázka algoritmu maximální dráhy

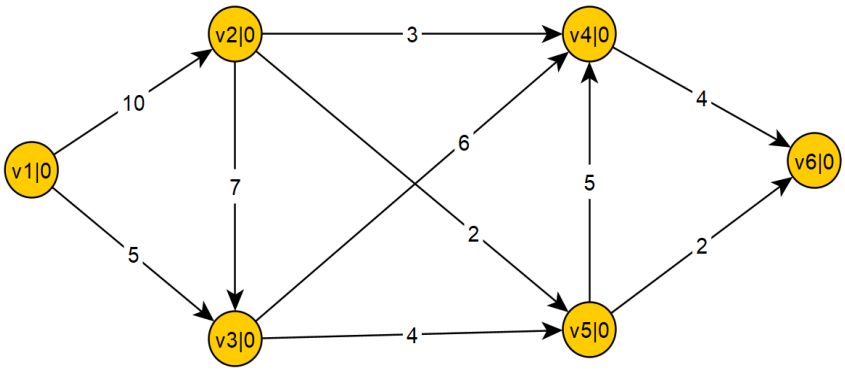
Kroky řešení algoritmu si ukážeme na následujícím grafu:



Obrázek 15: Graf pro ukázku algoritmu maximální dráhy

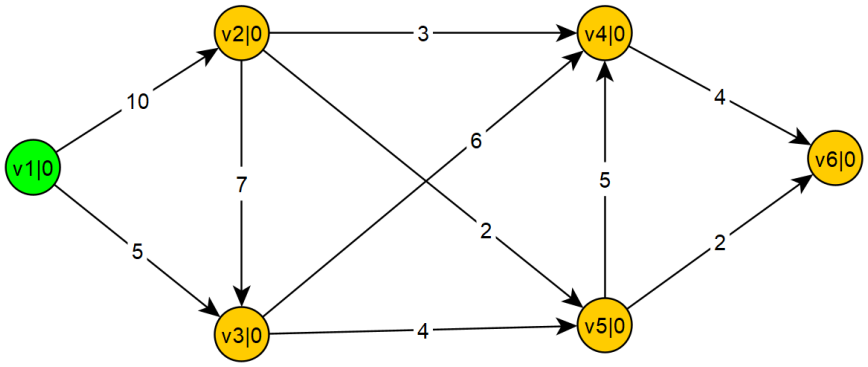
Úplně první krok, který musíme provést při hledání maximální dráhy, je stanovení požadovaného počátku a konce dráhy. V našem případě zvolíme jako počátek dráhy vrchol „v1“ a jako konec vrchol „v6“. Jakmile jsme si zvolili tyto vrcholy, dočasně ohodnotíme všechny vrcholy hodnotou 0.

Ohodnocení vrcholů je na grafu znázorněno za svislou čarou vedle označení vrcholu, stejně jak tomu je u ukázek všech ostatních algoritmů.



Obrázek 16: První krok algoritmu maximální dráhy

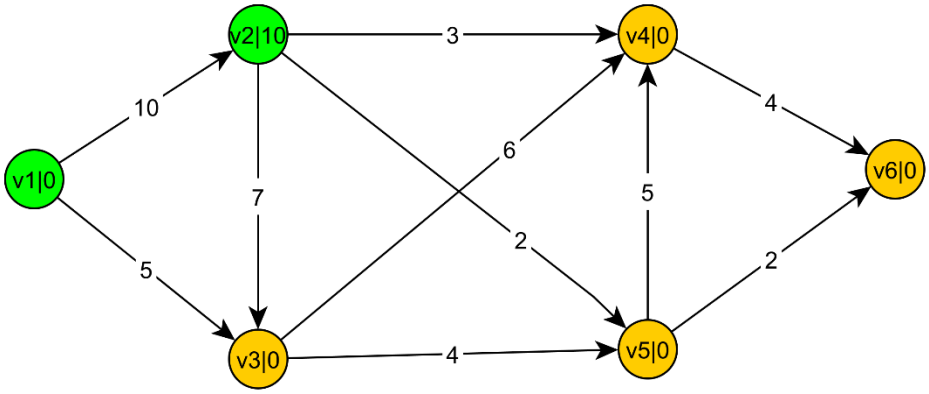
Dalším krokem je trvalé ohodnocení našeho počátečního vrcholu „v1“.



Obrázek 17: Druhý krok algoritmu maximální dráhy

V dalším kroku je potřeba rozhodnout, které další vrcholy nyní lze trvale ohodnotit. Tyto vrcholy mohou mít v množině svých předchůdců pouze vrcholy s trvalým ohodnocením. Nyní je tedy možné trvale ohodnotit pouze vrchol „v2“, který má v množině předchůdců pouze trvale ohodnocený vrchol „v1“. Vrchol „v3“ nyní ještě trvale ohodnotit nelze, protože má v množině předchůdců vrcholy „v1“ a „v2“ a vrchol „v2“ ještě nebyl trvale ohodnocen.

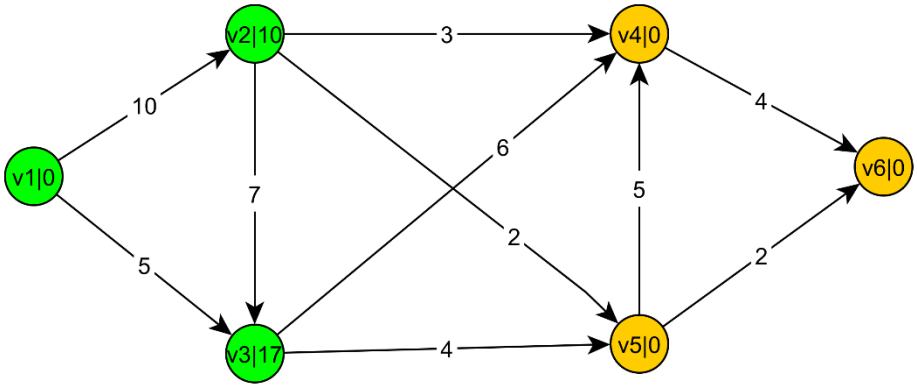
V našem případě jsme neměli na výběr a mohli jsme ohodnotit pouze vrchol „v2“ ale je nutno podotknout, že může nastat situace, kdy podmínkám bude vyhovovat více vrcholů zároveň. V případě, že se tento scénář naplní, je možné si vybrat kterýkoliv z vyhovujících vrcholů a definitivně jej ohodnotit.



Obrázek 18: Třetí krok algoritmu maximální dráhy

V dalším kroku opět hledáme takový vrchol, který lze definitivně ohodnotit. V našem případě lze nyní ohodnotit pouze vrchol v3, protože oba jeho předchůdci jsou již definitivně ohodnocení. Zde je ale už potřeba vybírat maximum mezi několika hodnotami:

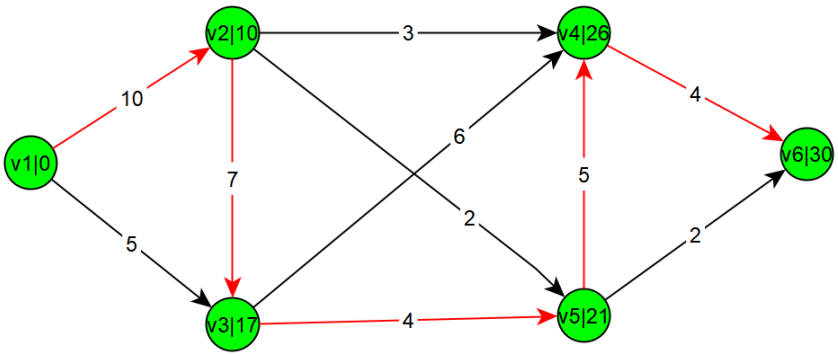
1) Původní ohodnocení vrcholu = 0
2) Ohodnocení jednoho předchůdce (v1) + ohodnocení incidující hrany = 0 + 5 = 5
3) Ohodnocení druhého předchůdce (v2) + ohodnocení incidující hrany = 10 + 7 = 17



Obrázek 19: Čtvrtý krok algoritmu maximální dráhy

Zbytek algoritmu spočívá v opakování předchozích dvou kroků, dokud se nenaplní námi zvolená podmínka. Může se tedy jednat o provádění algoritmu do té doby, dokud definitivně neohodnotíme cílový vrchol, nebo pokračujeme, dokud stále ještě existuje nějaký vrchol, který je možno trvale ohodnotit.

Jakmile se naplní námi stanovená podmínka, vykreslujeme maximální dráhu. Zpětně procházíme graf od koncového vrcholu v protisměru orientace hran tak, aby se rozdíl ohodnocení následníka a ohodnocení hrany rovnal ohodnocení předchůdce. Maximální dráhu v našem příkladu tedy začneme vykreslovat od vrcholu „v6“. Ohodnocení vrcholu „v6“ je 30. Do maximální dráhy zařadíme hranu mezi vrcholy „v4“ a „v6“, jejíž ohodnocení je 4, protože 30 – 4 nám dá výsledek 26, což je ohodnocení vrcholu „v4“. V grafu může být více než jedna maximální dráha.



Obrázek 20: Maximální dráha z vrcholu "v1" do vrcholu "v6"

Zde je vidět výsledná maximální dráha na námi vytvořeném grafu.

2.2.2 Využití

Hlavní využití algoritmu maximální dráhy je v plánování. Předpokládejme, že máme velký projekt, například stavbu domu, který se skládá z mnoha menších projektů: kopání základů, stavba zdí, připojení k plynu, elektřině a vodě, stavby střechy, interiérů, terénních úprav atd. Některé z těchto činností budou vyžadovat, aby se před nimi provedly jiné (nemůžete postavit střechu, dokud nemáte postavené stěny), zatímco jiné činnosti mohou být provedeny současně (dokončení interiérů a mezitím upravení terénu okolo domu). Každá dílčí práce má předpokládanou dobu potřebnou k jejímu dokončení a rádi byste předem věděli, jak dlouho bude celý úkol (stavba domu) trvat a kdy by měly být jednotlivé dílčí práce hotové, abyste si mohli domluvit dodavatele. [16]

2.3 Metoda kritické cesty (CPM)

Metoda CPM se používá pro časovou analýzu deterministicky ohodnocených síťových grafů, kromě určení kritických činností se zjišťují i časové rezervy činností. [17]

Původně byla metoda kritické cesty (CPM) vyvinuta za účelem zlepšení způsobů práce progresivním managementem. Byla vyvinuta na konci 50. let 20. století panem Morgan R. Walkerem ze společnosti DuPont a James E. Kellym Jr. ze společnosti Remington Rand. Během této doby prošel vývoj CPM několika fázemi a byl velmi nákladný. Tyto koncepty byly poté dále rozšiřovány a komercializovány. [18]

Oficiální prezentace vývojářů proběhla až v posledních dvou měsících roku 1959. Počátkem roku 1959 odešli James Kelley a Morgan Walker ze společnosti Du Pont a společně s Johnem Maunchlym založili společnost Maunchly Associates. [19]

Kritická cesta je definována jako (časově) nejdelší možná cesta z počátečního bodu grafu do koncového bodu grafu. Každý projekt má minimálně jednu kritickou cestu. Každá kritická cesta se skládá ze seznamu činností. Vrcholy grafu tvoří jednotlivé dílčí činnosti. Pro kritické úkoly platí, že jejich celková časová rezerva je rovna nule, tzn., že kdyby vzniklo zpoždění při provádění tohoto úkolu, bylo by zpožděno dokončení celého projektu. Kritická cesta se promítá do časového plánování a řízení projektu prakticky ve všech fázích životního cyklu projektu. [20]

Algoritmus:

(1) Provedeme již popsaný algoritmus maximální dráhy z počátečního vrcholu do koncového vrcholu. Nalezená maximální dráha je také kritickou cestou.

(2) Od koncového vrcholu začneme provádět zpětný průchod.

(3) Definitivně ohodnotíme koncový vrchol stejnou hodnotou, jakou měl po dokončení algoritmu maximální dráhy.

(4) Vybereme takový vrchol, který ještě není finálně ohodnocený, patří do výstupní množiny sousedů Γ- (je možné se z něho dostat do nějakého již finálně ohodnoceného vrcholu) a zároveň nepatří do výstupní množiny sousedů Γ- (není možné se z něho dostat do nějakého ještě finálně neohodnoceného vrcholu). Vybereme takový vrchol, který ještě není definitivně ohodnocený a v jeho množině následníků jsou pouze definitivně ohodnocené vrcholy (všichni jeho následníci jsou definitivně ohodnocení).

(5) Vrchol definitivně ohodnotíme tak, že porovnáme hodnotu původního ohodnocení vrcholu a hodnotu součtu ohodnocení následníka s ohodnocením incidující hrany (pokud je následníků více, je nutné projít všechny), a z těchto hodnot zvolíme tu menší.

(6) Pokud ještě nejsou všechny vrcholy ohodnoceny, pokračujeme znovu v kroku 4.

(7) Vypočteme volné časové rezervy pomocí vzorce: d(v) – d(u) – t(A), kde d(u) je ohodnocení počátečního vrcholu aktivity získané z prvního průchodu grafem, d(v) je ohodnocení koncového vrcholu aktivity také získané z prvního průchodu grafem a t(A) je ohodnocení hrany (doba trvání aktivity).

(8) Vypočteme celkové časové rezervy, pomocí vzorce: d´(v) – d(u) – t(A), kde d(u) je opět ohodnocení počátečního vrcholu aktivity získané z prvního průchodu grafem, d´(v) je ohodnocení koncového vrcholu aktivity získané ze zpětného průchodu grafem a t(A) je ohodnocení hrany (doba trvání aktivity).

(9) Kritické činnosti leží na kritické cestě (maximální dráze) a mají nulovou volnou i celkovou časovou rezervu. [21]

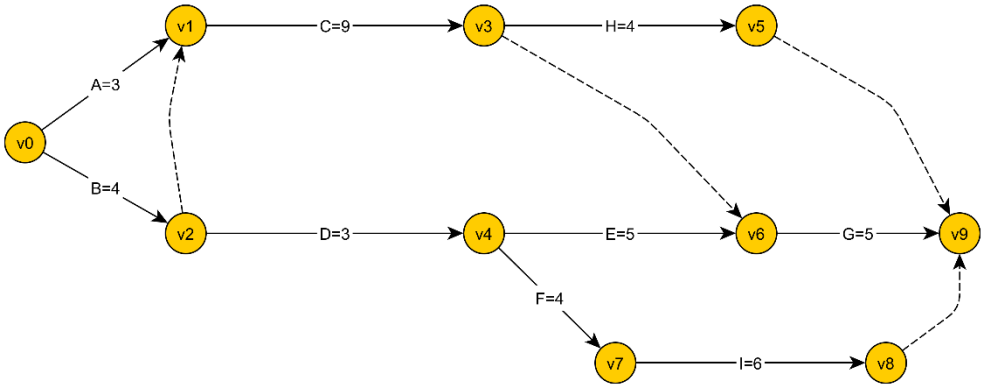
2.3.1 Ukázka metody kritické cesty

Příklady kritické cesty jsou obvykle zadávány ve formě tabulky činností. Pro náš příklad máme zadanou následující tabulku činností.

Tabulka 1: Tabulka činností pro ukázku metody kritické cesty

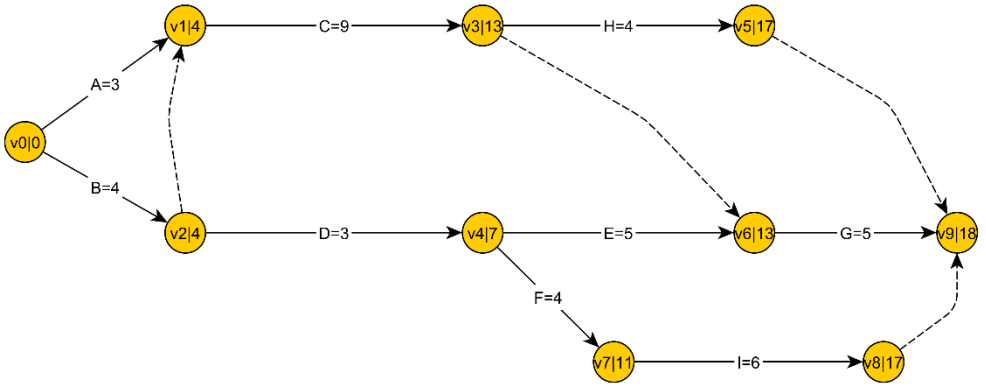
Činnost	A	B	C	D	E	F	G	H	I
Doba trvání (s)	3	4	9	3	5	4	5	4	6
Návaznost	–	–	A, B	B	D	D	C, E	C	F

Dalším krokem je vytvoření z dané tabulky činností graf. V tomto kroku si musíme dát pozor, aby činnosti správně navazovaly a abychom něco nepřehlédli. Možný tvar grafu, který budeme používat pro tuto ukázku, je zobrazen na dalším obrázku. Přerušované hrany jsou tzv. fiktivní hrany, mají nulové ohodnocení a používáme je proto, abychom se vyhnuli použití rovnoběžných hran.



Obrázek 21: Graf pro ukázku kritické cesty

Prvním krokem při algoritmu kritické cesty je určit maximální dráhu z počátečního vrcholu do koncového vrcholu. Tato část zde již popsána není, protože je již popsána právě u maximální dráhy. Graf po provedení algoritmu maximální dráhy vypadá takto:



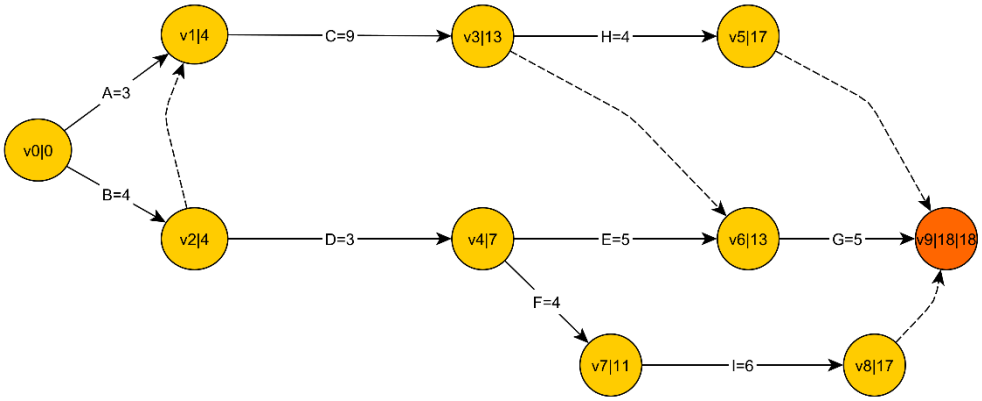
Obrázek 22: Stav grafu po provedení maximální dráhy

Nyní již jsme schopni určit kritickou dráhu (je stejná jako maximální dráha) a kritické činnosti. Následující postup využijeme k výpočtu časových rezerv všech činností.

Dalším krokem je zpětný průchod grafu a doplnění druhé hodnoty ohodnocení vrcholů. Na následujícím obrázku můžete vidět 3 hodnoty v označení vrcholů (oddělené svislými čarami). První hodnota udává název vrcholu, druhá hodnota je ohodnocení vrcholu při dopředném průchodu grafem (při hledání maximální/kritické dráhy) a třetí hodnotu budeme doplňovat nyní při zpětném průchodu a následně ji využijeme při určování časových rezerv.

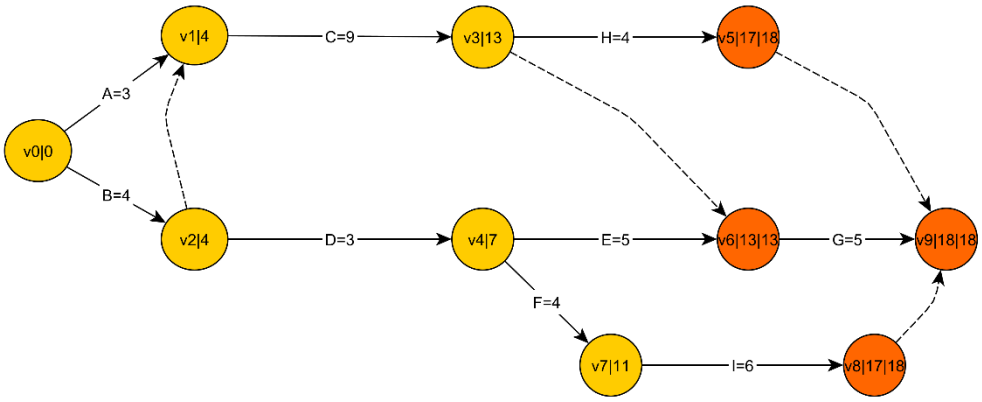
Prvním krokem nyní bude definitivní ohodnocení koncového vrcholu. V našem případě se jedná o vrchol „v9“ ohodnocený hodnotou „18“. Ohodnocení vrcholu při zpětném průchodu bude zobrazeno jako třetí hodnota – úplně vpravo. U koncového vrcholu se ohodnocení při

zpětném průchodu nezmění, opíšeme tedy původní hodnotu, kterou jsme získali z dopředného průchodu grafem.



Obrázek 23: Druhý krok metody kritické cesty

V dalším kroku je potřeba rozhodnout, které další vrcholy nyní lze trvale ohodnotit. Tyto vrcholy mohou mít v množině svých následníků pouze vrcholy s trvalým ohodnocením (sledujeme pouze ohodnocení teď ze zpětného průchodu, tedy třetí hodnotu za svislou čarou). Nyní je tedy možné trvale ohodnotit vrcholy „v5“, „v6“ a „v8“, které mají v množině následníků pouze vrchol „v9“, který je už trvale ohodnocený. Jako ohodnocení přiřadíme vrcholům rozdíl hodnot trvale ohodnoceného následníka („v9“) a incidující hrany (ohodnocení hrany je 0 v případě „v5“ a „v8“ a 5 v případě „v6“). Po provedení tohoto kroku bude graf vypadat následovně:

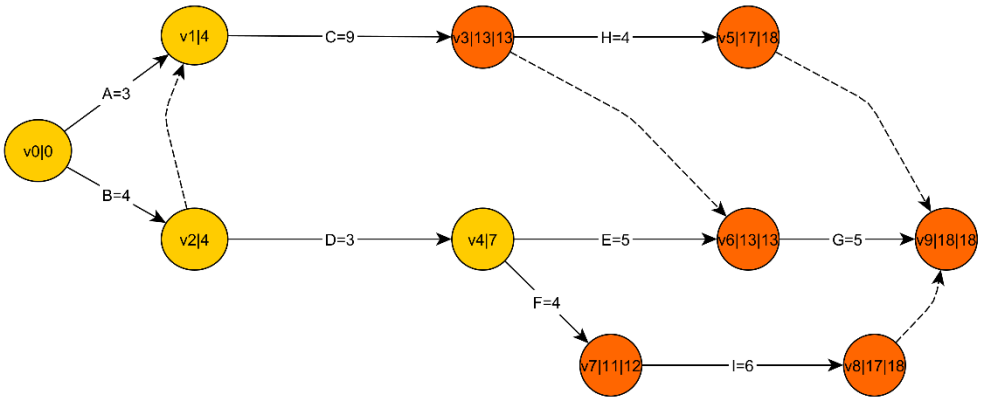


Obrázek 24: Třetí krok metody kritické cesty

V dalším kroku je potřeba rozhodnout, které další vrcholy nyní lze trvale ohodnotit. Tyto vrcholy mohou mít v množině svých následníků pouze vrcholy s trvalým ohodnocením. Nyní je tedy možné trvale ohodnotit vrcholy „v3“ a „v7“. Vrchol „v4“ ještě v tomto kroku ohodnotit

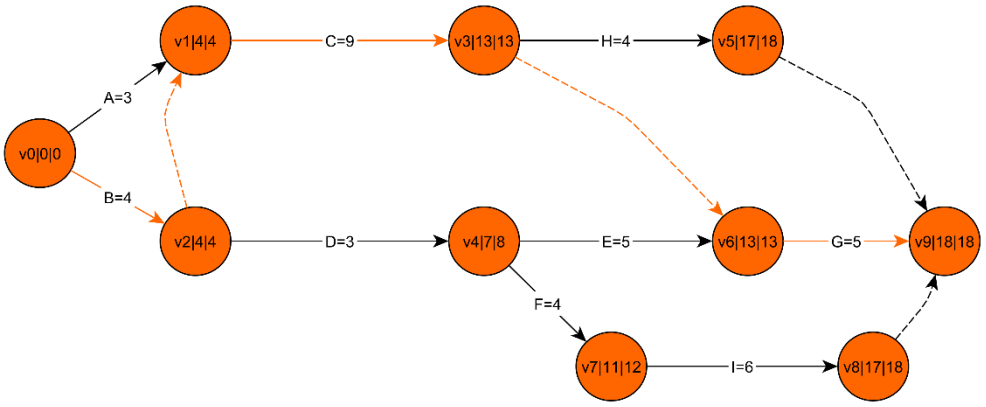
nelze, protože nejsou trvale ohodnoceni všichni jeho následníci (potřebujeme nejdříve ohodnotit vrchol „v7“.

Vrchol „v7“ má v množině následníků pouze vrchol „v8“, který je už trvale ohodnocený. Vrcholu „v7“ tedy přiřadíme ohodnocení 12 (rozdíl ohodnocení vrcholu „v8“ a incidující hrany). Vrchol „v3“ má však v množině následníků dva trvale ohodnocené vrcholy, „v5“ a „v6“. Zde tedy bude potřeba vybírat z několika hodnot. Určíme tedy rozdíly ohodnocení následníků a incidujících hran. Z výsledků vybereme **minimum**. Budeme tedy volit mezi hodnotami 18 – 4 = **14** vzhledem k následníkovi „v5“ a 13 – 0 = **13** vzhledem k následníkovi „v6“. Minimum z hodnot je 13, vrcholu „v3“ tedy přiřadíme ohodnocení 13.



Obrázek 25: Čtvrtý krok metody kritické cesty

Po provedení čtvrtého kroku pouze opakujeme kroky 3 a 4, než nebudeme mít definitivně ohodnoceny všechny vrcholy. Náš graf po ohodnocení všech vrcholů budete vypadat následovně.



Obrázek 26: Výsledný graf po vykonání zpětného průchodu metody kritické cesty

Na obrázku můžete vidět výsledný graf po vykonání zpětného průchodu algoritmu metody kritické cesty. Kritická cesta je vyznačena červenou barvou (jedná se o maximální dráhu z „v0“ do „v9“).

Posledním krokem je dopočítání časových rezerv. Vzorec pro spočítání těchto rezerv je již napsán v předchozí části, při popisu algoritmu, proto zde jen budou zobrazeny výsledné časové rezervy.

Tabulka 2: Časové rezervy pro zadaný příklad

Aktivita	A	B	C	D	E	F	G	H	I
FF	1	0	0	0	1	0	0	1	1
TF	1	0	0	1	0	0	0	1	1

FF v tabulce je zkratka pro free float (volnou časovou rezervu), TF je zkratka pro total float (celkovou časovou rezervu).

2.3.2 Využití

Tato metoda může sloužit jako nástroj zejména pro odhad doby trvání projektu. Používá se u přímočarých projektů, kde se dají velice přesně předpokládat doby trvání jednotlivých činností, toto platí například u stavebního průmyslu. Doby trvání pro činnosti projektu jsou známy obvykle podle minulých zkušeností nebo zkušeností z minulých projektů. Doby trvání nejsou statisticky určeny. Mimo stavební průmysl se lze s metodou setkat i v oblasti logistiky a dopravy. [20]

2.4 Maximální tok

Typickou aplikací grafů je jejich použití k reprezentaci sítí infrastruktury, např. pro distribuci vody, elektřiny nebo dat. Pro zachycení omezení sítě je užitečné opatřit hrany v grafu kapacitami, které modelují, kolik zdrojů lze daným spojením přenést. Častou vlastností, kterou chceme v těchto typech sítí znát, je, kolik zdroje lze současně přepravit z jednoho bodu do bodu jiného neboli problém maximálního toku. Tok v síti je důležitý, protože jej lze použít k vyjádření široké škály různých druhů problémů. Využitím dobrých algoritmů pro řešení síťového toku okamžitě získáme algoritmy pro řešení mnoha dalších problémů. [22]

Každá hrana grafu má maximální množství prostředků, které lze po této hraně přesunout, tato vlastnost se nazývá kapacita hrany. Nasycená hrana je taková hrana, u které se její tok rovná její kapacitě [22]

Naším cílem je přenést v grafu co největší tok z bodu s do bodu t. Pravidla jsou taková, že žádná hrana nesmí mít tok přesahující její kapacitu, a pro každý vrchol kromě *s* a *t* se tok do vrcholu musí rovnat toku z vrcholu. [23]Pro řešení maximálního toku si představíme dva algoritmy, které pro řešení tohoto problému můžeme využít. Jeden z nich lze použít pouze na rovinné sítě, nazývá se „algoritmus nejhořejší cesty“, a ten si představíme jako první.

2.4.1 Algoritmus nejhořejší cesty

Pro rovinné grafy lze použít velice jednoduchý způsob nalezení maximálního průtoku. Tento způsob se nazývá algoritmus nejhořejší cesty. Jako vstup potřebujeme mít rovinnou síť, ve které jsou zdroj a ukončení (stok, ústí) na obvodu grafu.

Algoritmus:

(1) Inicializujeme tok všech hran na hodnotu 0.

(2) Nalezneme nejvýše položenou cestu grafu, pokud žádná neexistuje, ukončíme provádění algoritmu.

(3) Nalezneme hranu, u které zbývá nejméně jednotek, aby byla zcela nasycená, a počet těchto jednotek si zapamatujeme.

(4) Zvětšíme nasycení všech hran, které se nacházejí na naší nejhořejší cestě, o jednotku získanou z bodu 3.

(5) Z grafu odebereme všechny hrany, které jsou zcela nasycené, a pokračujeme znovu krokem 2.

Po ukončení algoritmu hodnotu maximálního toku, získáme tak, že sečteme tok vycházející ze zdroje. [23]

2.4.2 Ford-Fulkersonův algoritmus

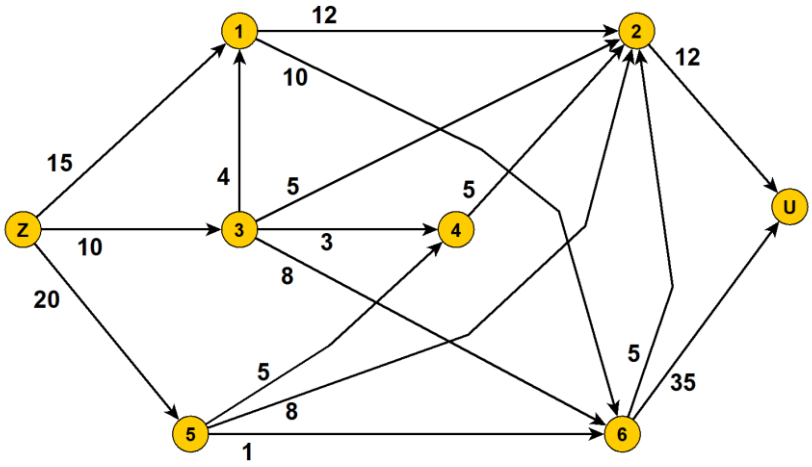
Fordův-Fulkersonův algoritmus vypočítá maximální tok ve všeobecné síti. Lze jej tedy použít i u grafů, u nichž se hrany při zakreslení kříží. U těchto grafů nelze použít algoritmus nejhořejší cesty, protože při křížení hran v grafu ne vždy nejhořejší cesta existuje. Algoritmus se skládá ze dvou částí:

1. Nalezení úplného toku – úplný tok nemusí být vždy maximální. Postup je lehce podobný s algoritmem nejhořejší cesty, avšak cestu vybíráme podle odlišných kritérií.

2. Nalezení maximálního toku. V předchozím kroku jsme nalezli úplný tok. Ten však může být menší než tok maximální. Proto je potřeba zjistit, jestli se úplný tok nedá ještě „vylepšit“. K tomu nám slouží zlepšující cesty. V síti tedy budeme hledat zlepšující cesty, ve kterých budeme upravovat tok.

2.4.3 Ukázka Ford-Fulkersonova algoritmu

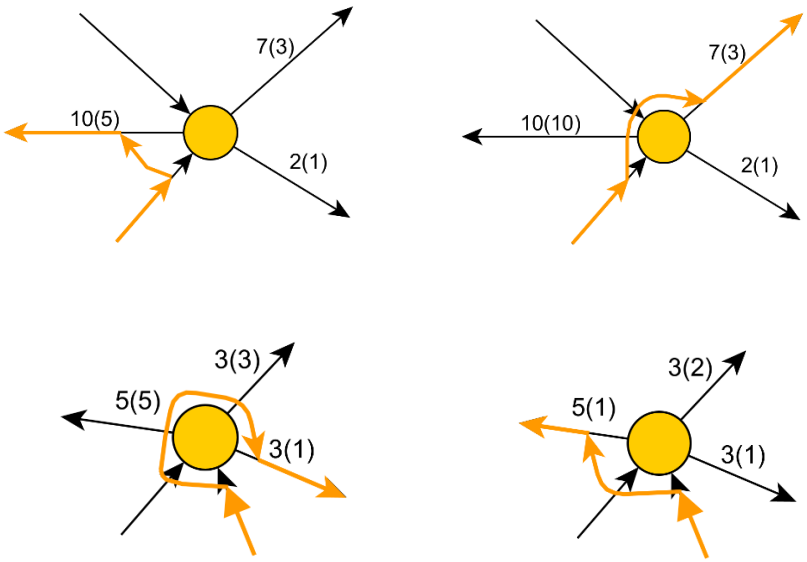
V následující všeobecné síti nyní nalezneme maximální tok.



Obrázek 27: Síť pro ukázku Ford Folkersonova algoritmu

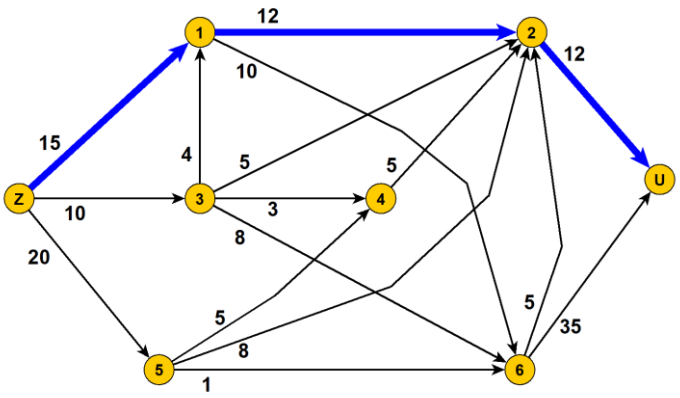
1) Nalezení úplného toku

Podobně jako při hledání úplného toku v rovinné síti budeme postupně vybírat cesty (dráhy) ze zdroje do ústí. Protože však ve všeobecné síti nemůžeme vybírat nejhořejší cestu, je potřeba si určit jinou podmínku, podle které budeme do cesty zařazovat hrany. Začneme v počátečním vrcholu, tedy ve zdroji, a v prvním kroku vybereme hranu, která směřuje nejvíce nahoru, což je hrana ze *z* do *v1*. V místě, kde jsme vstoupili do vrcholu *v1*, vrchol „obcházíme“ po směru hodinových ručiček a jakmile narazíme na první hranu, kterou lze poslat nějaký tok (vychází z vrcholu ven a není nasycená), zařadíme ji do cesty. Níže je na několika obrázcích znázorněn princip, jakým vybíráme hrany.



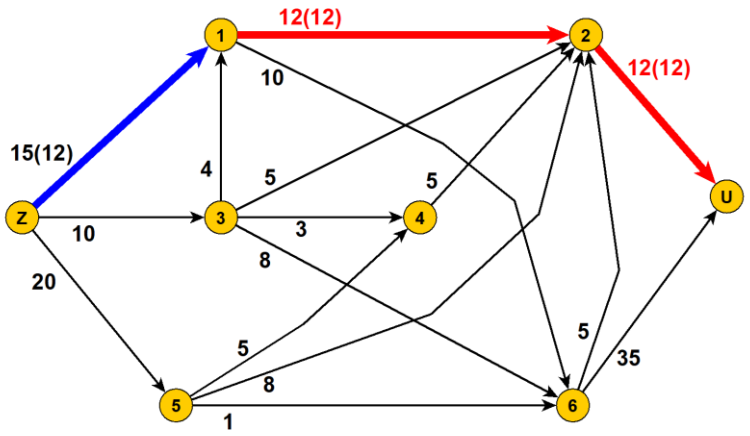
Obrázek 28: Příklady principu výběry hrany

První cesta ze zdroje do ústí, kterou jsme vybrali, tedy bude vypadat takto:



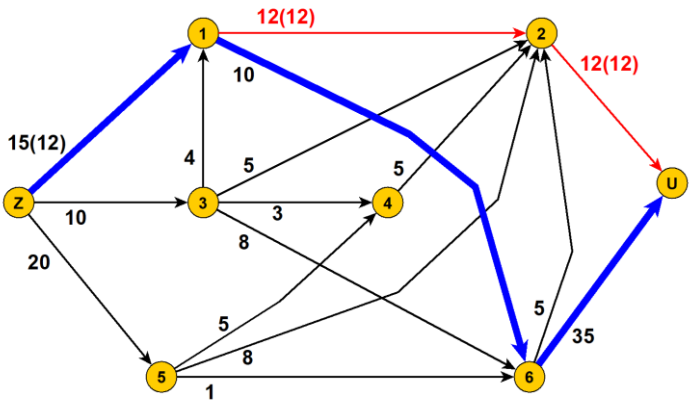
Obrázek 29: První krok Ford-Fulkersonova algoritmu

Nyní touto cestou povedeme takový minimální počet jednotek toku, aby došlo k nasycení alespoň jedné hrany. Nasycené hrany jsou znázorněny červeně.

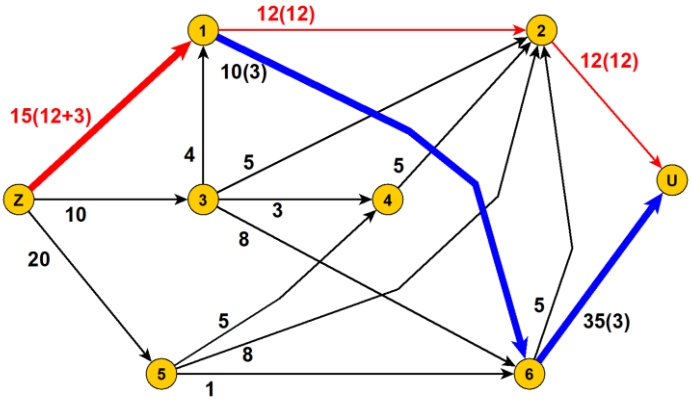


Obrázek 30: Druhý krok Ford-Fulkersonova algoritmu

V dalším kroku opět budeme podle výše definovaných pravidel hledat cestu ze zdroje do ústí, přes kterou pak pošleme takový minimální tok, aby došlo k nasycení alespoň jedné hrany v cestě.

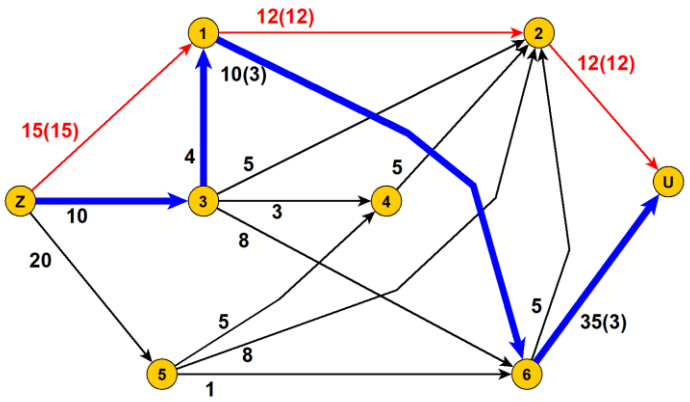


Obrázek 31: Třetí krok Ford-Fulkersonova algoritmu

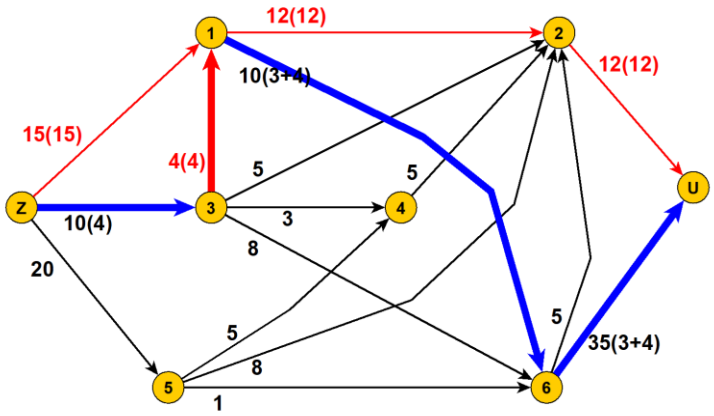


Obrázek 32: Čtvrtý krok Ford-Fulkersonova algoritmu

Obdobným způsobem pokračujeme dále tak dlouho, dokud takto nevybereme všechny cesty ze zdroje do ústí.

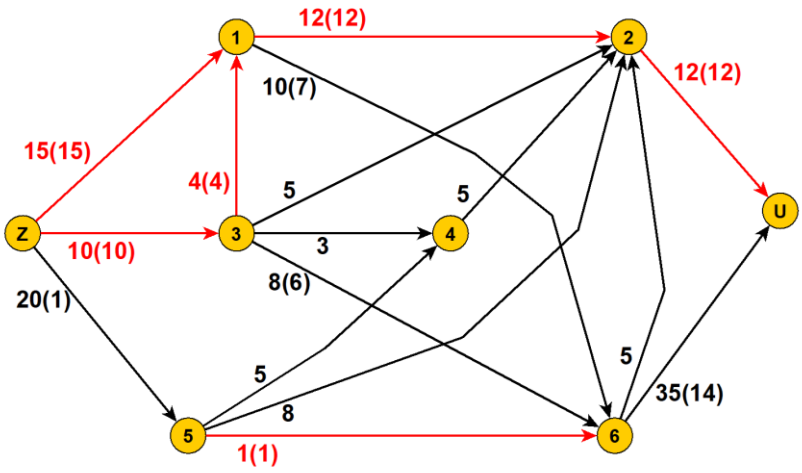


Obrázek 33: Pátý krok Ford-Fulkersonova algoritmu



Obrázek 34: Šestý krok Ford-Fulkersonova algoritmu

Takto vypadá síť po nalezení úplného toku. Ne vždy je však úplný tok stejný jako maximální. Proto v dalším kroku budeme hledat zlepšující cesty.



Obrázek 35: Sedmý krok Ford-Fulkersonova algoritmu

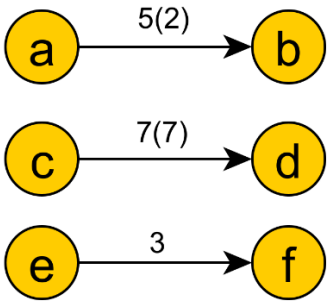
2) Hledání zlepšujících cest.

Zlepšující cesta vede ze zdroje do ústí. Hranu do zlepšující cesty můžeme přidat tehdy, pokud jí lze projít podle následujících kritérií:

Hranou se můžeme pokusit projít dopředně i zpětně (proti směru orientace):

- Při dopředném projití hranou **přidáváme** tok, při zpětném projití tok **odebíráme**.
- Hranou lze projít, pokud při dopředném průchodu lze přidat tok, nebo pokud při zpětném průchodu lze odebrat tok. Jinak hranou projít nelze.

Na následujícím obrázku je možné vidět několik příkladů. Hranou z „a“ do „b“ lze projít dopředně, protože lze přidat ještě 3 jednotky toku do nasycení hrany. Touto hranou lze projít i zpětně, protože z hrany můžeme odebrat 2 jednotky toku. Hranou z „c“ do „d“ nelze projít dopředně, protože hrana je nasycená a nelze do ní již přidat žádný tok. Touto hranou lze však projít zpětně, jelikož z hrany můžeme 7 jednotek toku odebrat. Poslední hranou z „e“ do „f“ lze projít dopředně, v hraně neteče žádný tok, můžeme tedy přidat ještě 3 jednotky toku. Avšak zpětně hranou projít nelze, protože v hraně není tok a nemůžeme tedy žádný tok odebrat.

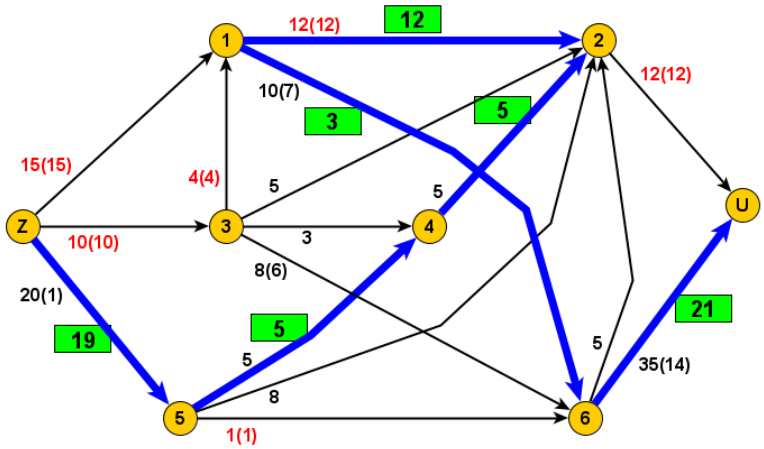


Obrázek 36: Příklady možných variant hran

V síti, ve které jsme nalezli úplný tok, teď tedy budeme podle těchto pravidel hledat zlepšující cesty. Těch zde existuje hned několik. Například:

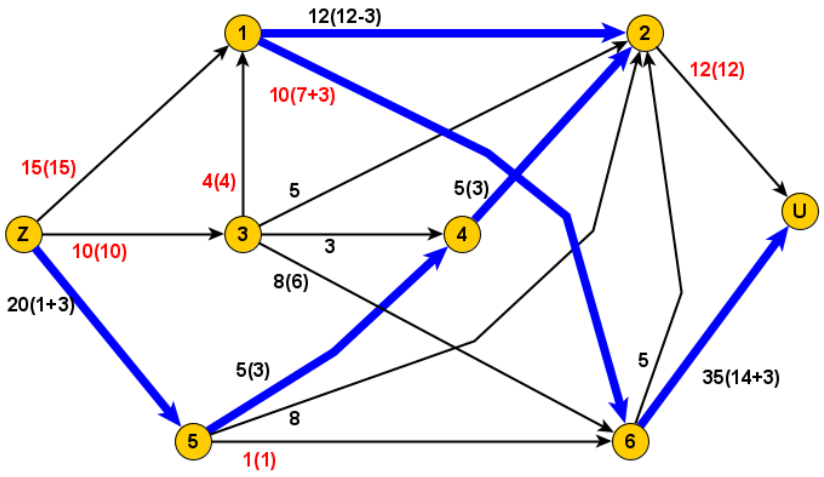
- z →v5→v4→v2←v1→v6→u
- z →v5→v4→v2←v1←v3→v6→u
- z →v5→v2←v1→v6→u
- z →v5→v2←v1←v3→v6→u

Vybereme si jednu ze zlepšujících cest a upravíme v ní tok. Na obrázku níže je u každé hrany ve zlepšující cestě v zeleném rámečku zobrazeno, o kolik jednotek lze v dané hraně tok upravit (kolik jednotek toku lze při dopředném/zpětném průchodu hranou do hrany přidat/odebrat).

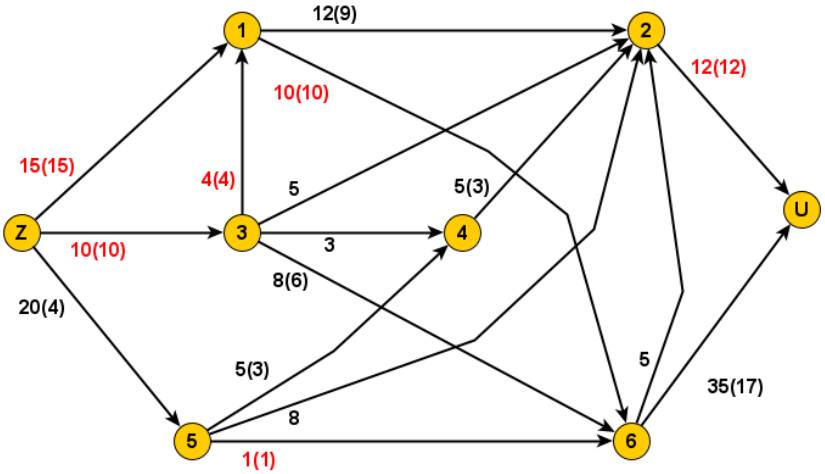


Obrázek 37: Osmý krok Ford-Fulkersonova algoritmu

Z těchto hodnot následně vybereme minimum (tedy hodnotu 3) a o tuto hodnotu opravíme tok v celé cestě. Hodnotu toku budeme v jednotlivých hranách buď přidávat (pokud jsme hranou procházeli dopředně), nebo odebírat (pokud jsme procházeli zpětně).

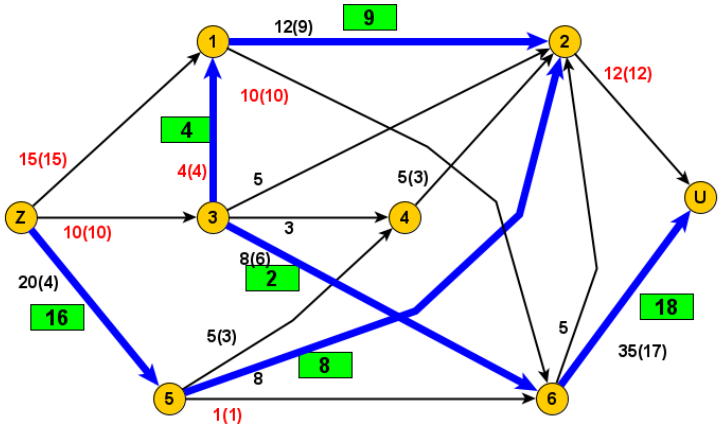


Obrázek 38: Devátý krok Ford-Fulkersonova algoritmu

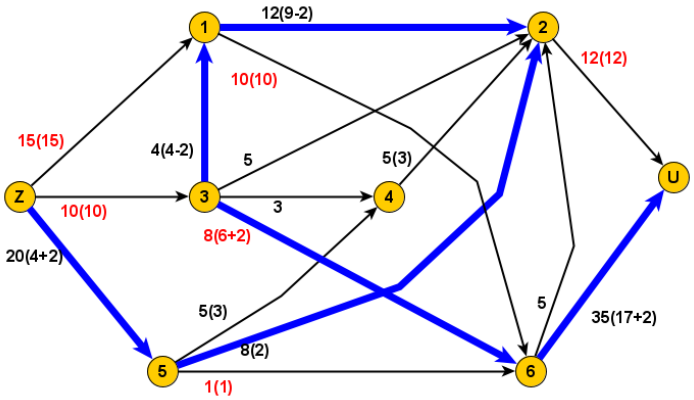


Obrázek 39: Desátý krok Ford-Fulkersonova algoritmu

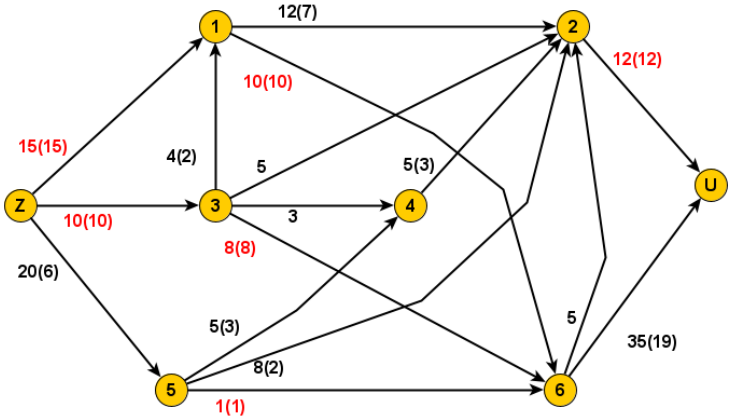
A v grafu opět hledáme zlepšující cestu, ve které opravíme tok. Takto pokračujeme tak dlouho, dokud v grafu existuje nějaká zlepšující cesta.



Obrázek 40: Jedenáctý krok Ford-Fulkersonova algoritmu



Obrázek 41: Dvanáctý krok Ford-Fulkersonova algoritmu



Obrázek 42: Třináctý krok Ford-Fulkersonova algoritmu

Nyní již v grafu žádná další zlepšující cesta neexistuje, nalezený tok je tedy maximální. Hodnotu toku určíme jako součet toků vycházejících ze zdroje. Tedy 15 + 10 + 6 = 31.

2.4.4 Využití

Maximální tok má spoustu využití, pojďme si tedy přiblížit alespoň některé z nich.

Vyřazení basebalového týmu: Jedná se o problém, u kterého chceme zjistit, zda je stále možné v současném stavu tabulky týmů zjistit, zda náš daný tým má stále ještě šanci skončit v tabulce první (nebo se dělit o první místo). Podrobněji se snažíme vybrat výherce každé další konané hry, abychom dosáhli námi požadovaného výsledku. Jelikož chceme vybrat vítěze každého zápasu, tak začneme vytvářet bipartitní graf (graf, jehož vrcholy můžeme rozdělit na 2 množiny, které nejsou spojeny žádnou hranou). A snažíme se dostat do stavu, kdy jsme schopni naplnit všechny hrany vycházející ze zdroje, pokud se nám toto podaří, tak náš daný tým je schopen skončit na konci sezóny první. [24, 25]

2.5 Eulerovský tah

Eulerovský tah je tah, který obsahuje každou hranu grafu právě jednou. Rozlišujeme typy eulerovského tahu:

Uzavřený Eulerovský tah: začíná i končí ve stejném vrcholu

Otevřený Eulerovský tah: počáteční a koncový vrchol tahu se liší

Ne v každém grafu Eulerovský tah existuje. Aby v grafu existoval uzavřený Eulerovský tah, musí být všechny vrcholy grafu sudého stupně (tedy 2., 4., 6. atd.). Otevřený Eulerovský tah existuje v takovém grafu, kde právě dva vrcholy jsou lichého stupně a všechny ostatní vrcholy jsou sudého stupně. V jednom z dvojice lichých vrcholů pak tah začíná a ve druhém končí.

Pokud v grafu existuje Eulerovský tah, pak takový graf nazýváme Eulerovským grafem.

Algoritmus:

(1) Ujistěte se, že graf má buď žádný, nebo 2 liché vrcholy.

(2) Pokud nemá graf žádný lichý vrchol, můžeme zvolit začátek Eulerova tahu kdekoliv, pokud v grafu existují 2 liché vrcholy, musíme začít u jednoho z nich.

(3) Vyberte si libovolnou hranu vycházející z vrcholu, na kterém se právě nacházíte. Máte-li na výběr mezi mostem a hranou co není most, vždy zvolte hranu, co není most.

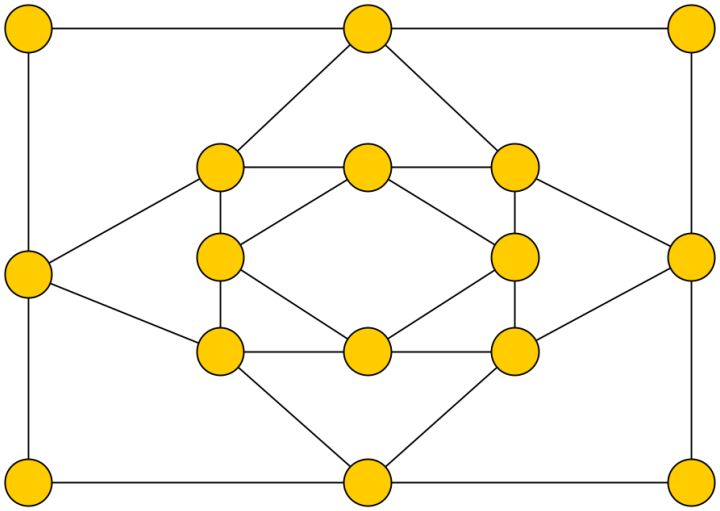
(4) Přesuňte se na vrchol na druhé straně Vámi vybrané hrany.

(5) Odstraňte Vámi vybranou hranu z grafu.

(6) Opakujte kroky 3 až 5, dokud Vám nedojdou hrany. [26]

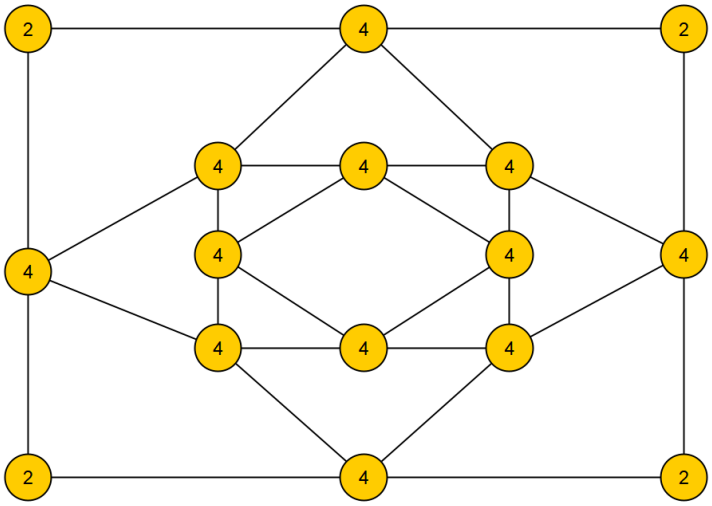
2.5.1 Ukázka Eulerova tahu

Kroky řešení Eulerova tahu si ukážeme na následujícím grafu:



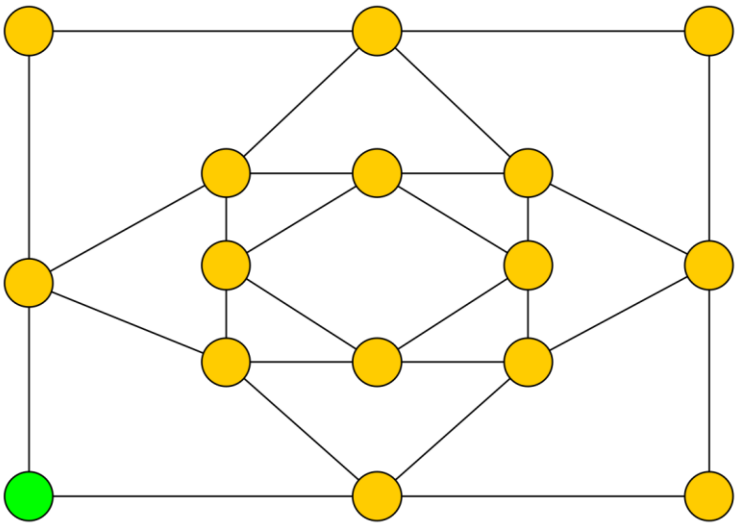
Obrázek 43: Graf pro ukázku řešení Eulerova tahu

Prvním krokem Eulerova tahu je zjištění počtu lichých vrcholů. Jedná se tedy o vrcholy, které mají lichý počet sousedních vrcholů. Spočtěme tedy stupně vrcholů.



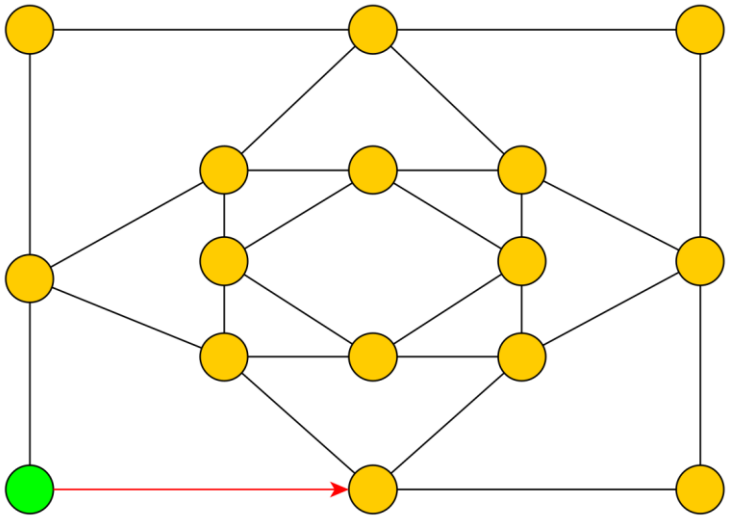
Obrázek 44: První krok Eulerova tahu

Jak můžeme z obrázku vidět, v našem grafu se nenachází žádné vrcholy s lichým stupněm, proto můžeme přejít k druhému kroku algoritmu, který spočívá ve stanovení počátečního vrcholu Eulerova tahu Jelikož jsou všechny vrcholy sudé, můžeme tento vrchol volit libovolně.



Obrázek 45: Druhý krok Eulerova tahu

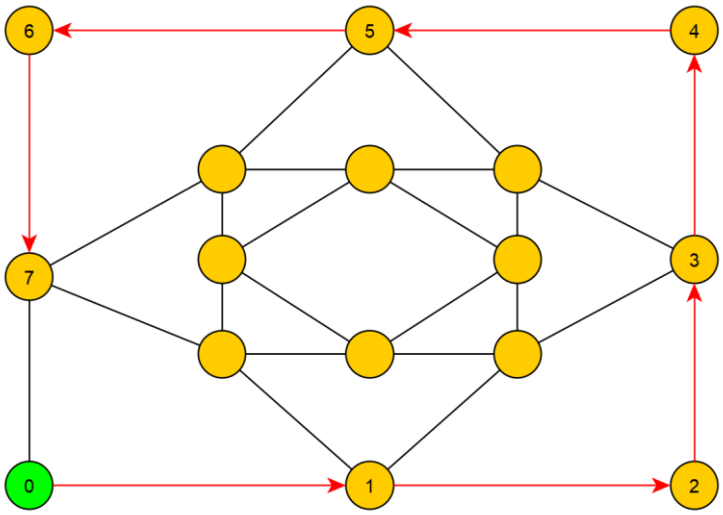
Třetí krok algoritmu spočívá ve výběru takové hrany vycházející z vrcholu, která není mostem. V našem grafu máme na výběr mezi dvěma hranami, co nejsou mosty, proto vybereme jednu z nich.



Obrázek 46: Třetí krok Eulerova algoritmu

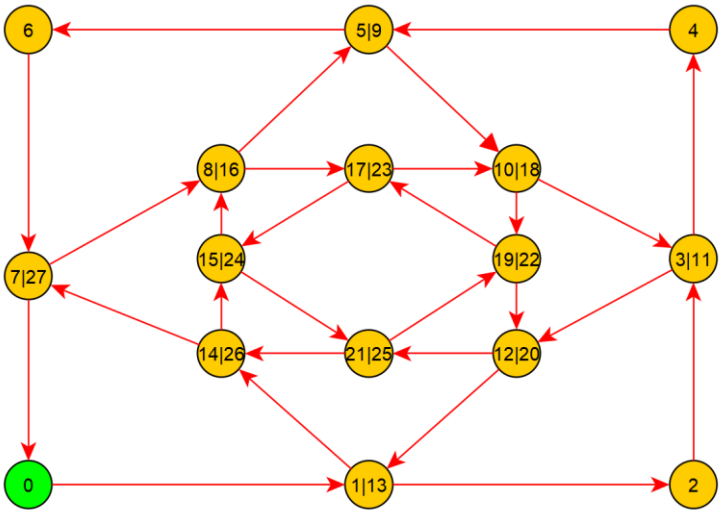
Poté se přesuneme na další vrchol a vybranou hranu z grafu odstraníme. Pro lepší zobrazení nebudeme hrany z obrázků odstraňovat, ale pouze je zvýrazníme červenou barvou a vykreslíme šipku, která bude orientována směrem, kterým jsme hranou prošli.

Dále postupně provádíme předchozí krok algoritmu, dokud nám v grafu již nezbydou žádné hrany.



Obrázek 47: Nutnost odebírat nejdříve hrany co nejsou mostem

Musíme si ale dát pozor, abychom skutečně ctili danou podmínku „vybírání hran co nejsou mosty předtím, než budeme odebírat hrany co jsou mosty“. Na obrázku výše vidíte příklad, kdy jedna z našich možností je jít hranou přímo k počátečnímu vrcholu. Toto ale nemůžeme, protože bychom si vytvořili 2 nesouvislé komponenty souvislosti a potom by nebylo možné dokončit Eulerovský tah, proto je nutné si na tuto situaci dávat pozor.



Obrázek 48: Výsledný stav po provedení Eulerova tahu

Na obrázku výše je vidět jeden z mnoha možných způsobu provedení Eulerova tahu na námi zadaném grafu.

2.5.2 Využití

Eulerovy tahy mají mnoho praktických aplikací. V matematice lze grafy využít k řešení mnoha složitých problémů, jako je například problém Königsberského mostu. Poštovní doručovatelé mohou navíc využívat Eulerovy tahy, aby měli trasu, na které nemusí opakovat své předchozí kroky. V širším spektru jsou Eulerovy tahy užitečné pro malíře, popeláře, piloty letadel, vývojáře GPS, nebo obchodníky distribuující reklamu. Eulerovské tahy tedy může použít téměř každý, kdo používá jakékoliv cesty. [26]

3 NÁVOD PRO POUŽITÍ APLIKACE

3.1 Prerekvizity pro spuštění aplikace

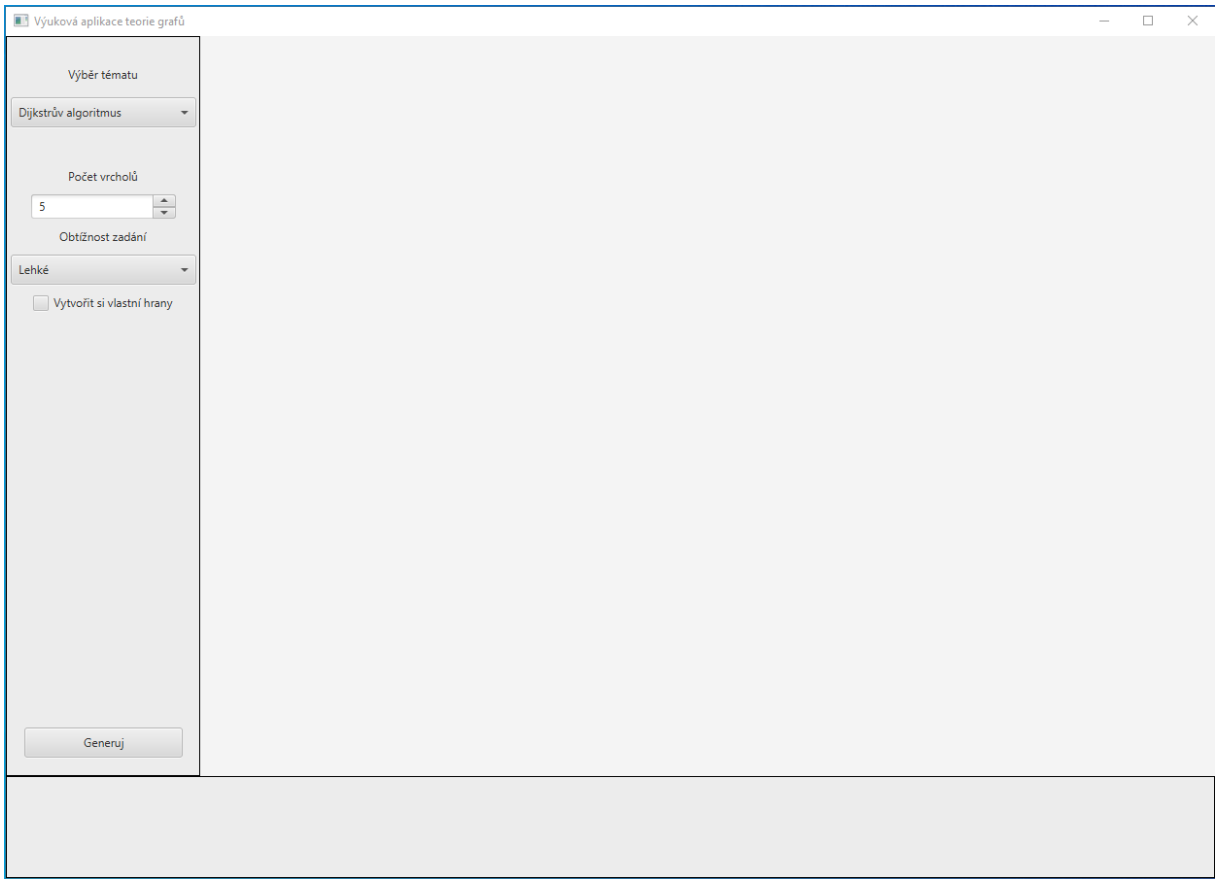
Aplikace se jednoduše spustí otevřením exe souboru *vyukovaAplikace.exe* v přiloženém zip archivu.

Důležitou podmínkou je také mít nainstalované JRE minimálně ve verzi 1.8, nicméně aplikace byla testována pouze ve verzi 1.8, proto je primárně doporučeno používat právě tuto.

3.2 Orientace v aplikaci

3.2.1 Hlavní okno

Po spuštění aplikace se nám zobrazí následující okno:



Obrázek 49: Hlavní okno aplikace

Hlavní okno se skládá ze tří hlavních částí:

- Levá část obsahuje základní informace o tom, jaký graf se má generovat.
- Spodní část obsahuje informace o specifikaci problému, který chceme řešit, obsahuje tedy mimo jiné výběr počátečního a koncového vrcholu algoritmu.

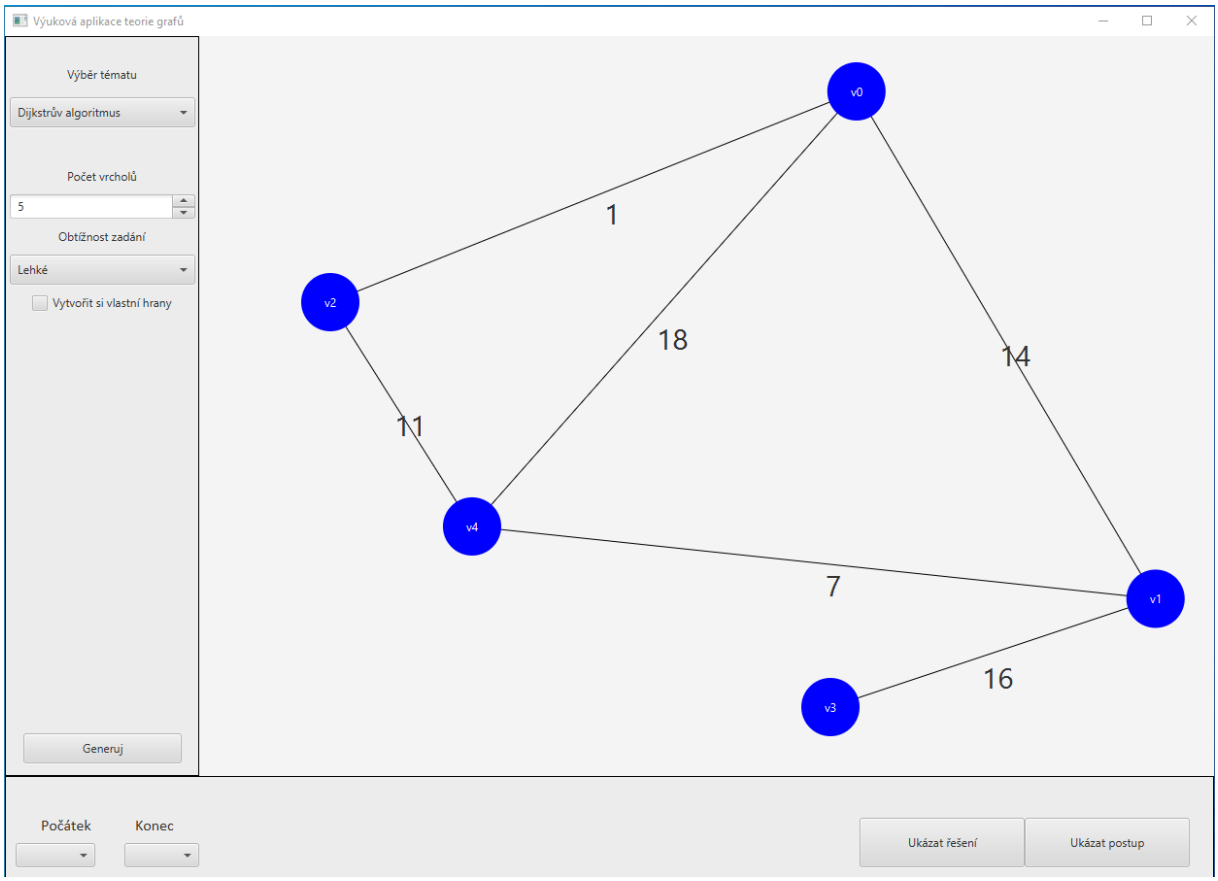
- Hlavní část umístěná uprostřed obrazovky obsahuje samotný graf.

Levou část si můžeme popsat rovnou, protože většina z možných funkcionalit je viditelná již na prvním obrázku. První výběrové tlačítko nám umožňuje vybrat téma. Témata podporovaná touto aplikací jsou:

- Dijkstrův algoritmus
- Nejspolehlivější cesta
- Maximální dráha
- Kritická cesta
- Ford-Fulkersonův algoritmus
- Nejhořejší cesta
- Eulerovský tah

Dále je u některých z témat, u kterých je povoleno generování příkladů, umožněno vybrat počet vrcholů, kterými bude graf disponovat po jeho vygenerování. Je zde také možnost zaškrtnout výběr „Vytvořit si vlastní hrany“. Toto nám umožní přesně definovat hrany, které chceme, aby byly v grafu přítomny. Poslední výběrové tlačítko, které tato část obsahuje, je tlačítko, které umožní uživateli vybrat obtížnost generovaného zadání. Toto tlačítko je dostupné u všech témat a jeho funkčnost se liší. U témat, kde je povolena úplná generace příkladů (to jsou témata: Dijkstrův algoritmus, nejspolehlivější cesta, maximální dráha a Eulerovský tah), se volba obtížnosti promítne na počtu generovaných hran. U ostatních témat, kde se příklady generují z databáze, jsou příklady zařazeny do kategorií, které reflektují jejich obtížnost. Poslední funkcionalitou, kterou tato část disponuje, je tlačítko generuj, které uživateli vygeneruje samotný graf a vykreslí ho na obrazovku.

Nechme si tedy teď vygenerovat graf, abychom si mohli popsat funkčnosti zbylých dvou částí. Vygenerujeme si jednoduchý graf s 5 vrcholy na téma Dijkstrův algoritmus. Výsledek můžete vidět na dalším obrázku.



Obrázek 50: Ukázka generování grafu

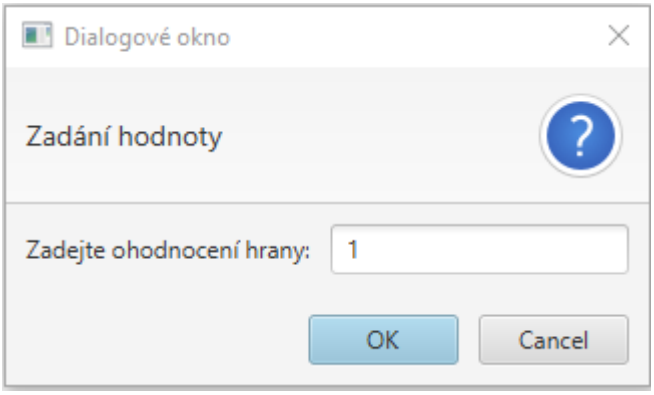
V prostřední části můžeme vidět vygenerovaný graf. Hrany jsou popsány jejich ohodnoceními, která se také náhodně vygenerují. S vrcholy je možno libovolně pohybovat tak, že na ně najedeme myší, podržíme prostřední nebo pravé tlačítko myši a přesuneme vrchol do námi požadované pozice.

Poslední část je část, která se nachází na spodní části obrazovky. Zde je možno upravovat počáteční a u některých algoritmů i jejich koncový vrchol. Nalezneme zde také tlačítka, která nám rovnou zobrazí řešení algoritmu, nebo ukáží postup řešení.

3.2.2 Vytvoření grafu s vlastními hranami

Pokud před stlačením tlačítka „Generuj“ uživatel zaškrtne možnost „Vytvořit si vlastní hrany“, tak se vygenerují pouze vrcholy. Uživatel pak může pomocí kliknutí levého tlačítka myši na vrcholy vybrat vrcholy, mezi kterými se vytvoří hrany. Po kliknutí na první vrchol vrchol zezelená, aby uživateli znázornil, že tento vrchol je počáteční vrchol jeho nové hrany. Jakmile

uživatel klikne na další vrchol, ukáže se mu nové okno, které ho požádá o zadání ohodnocení hrany.[1]



Obrázek 51: Dialogové okno pro zadání ohodnocení

Pokud klikneme na tlačítko „OK“, aplikace nejprve zkontroluje, zda je zadané ohodnocení hrany validní a pokud ano, vytvoří se nová hrana mezi vrcholy, které si uživatel zvolil, s odpovídajícím ohodnocením. Pokud uživatel klikne na tlačítko „Cancel“, tak se žádná hrana nevytvoří a uživatel je vrácen zpět na hlavní okno programu.

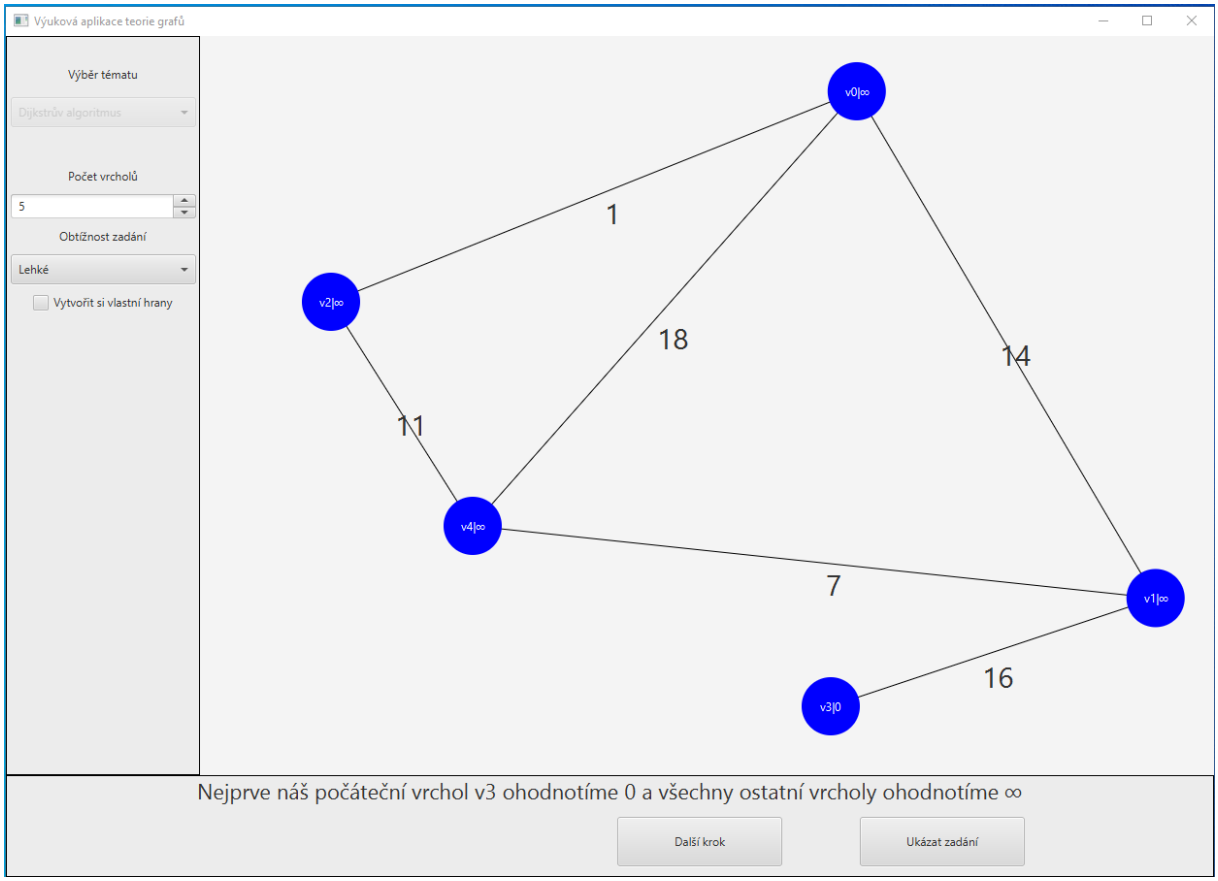
Jakmile uživatel vytvoří svůj požadovaný graf, stačí kliknout na tlačítko „Dokončit editaci“ nacházející se v levé části hlavního okna. Aplikace poté nejprve zkontroluje, zda je graf v takovém tvaru, aby na něm mohl být proveden odpovídající algoritmus, a pokud některá z podmínek pro provedení algoritmu není splněna, je o tom uživatel informován výstražným oknem s popisem, kterou podmínku graf nesplňuje.

3.2.3 Ukázání postupu algoritmu

Mimo ukázky řešení algoritmu umožňuje tato aplikace také postupné zobrazování kroků řešení, aby mohl uživatel lépe pochopit, jak daný algoritmus funguje a naučit se, jak se sám dostat k výsledku.

Postup řešení u daného algoritmu si můžeme ukázat pomocí kliknutí na tlačítko „Ukázat postup“. Ze spodní části okna zmizí některá tlačítka a naopak se objeví jiná, a to tlačítka „Další krok“, tlačítko „Ukázat zadání“ a poté také tlačítko „Předchozí krok“, které se objeví pouze pokud se nenacházíme na prvním kroku postupu. Po kliknutí na tlačítko „Ukázat postup“ bude vypadat hlavní okno takto:

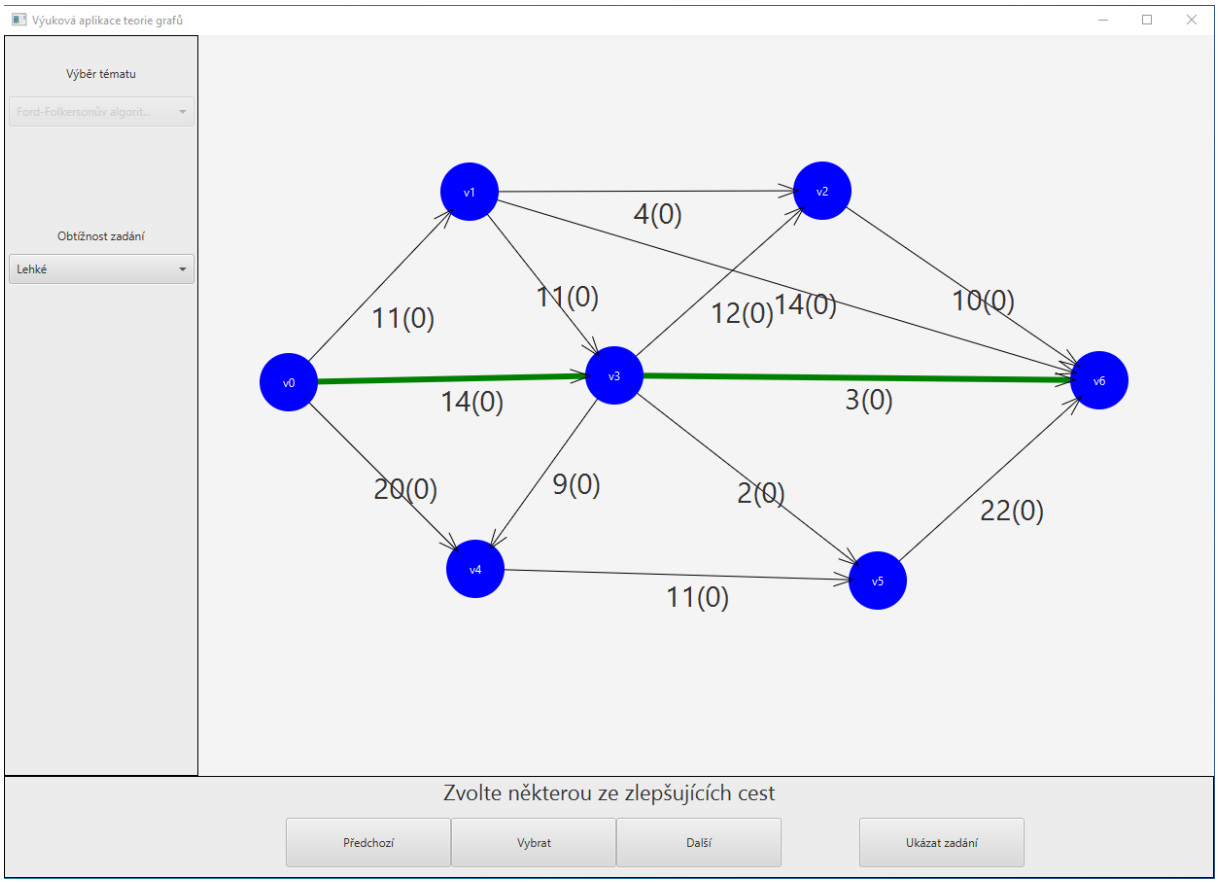
[1] Toto neplatí u grafů, které mají jako téma zvolený Eulerovský tah, u tohoto tématu totiž není třeba aby hrana měla jakékoli ohodnocení, proto se žádné nové okno neobjeví.



Obrázek 52: Ukázka procházení postupu

V dolní části obrazovky se objeví popisek, který popisuje aktuální krok a zároveň se tento krok promítne i do grafu. Kroky lze libovolně procházet pomocí tlačítek „Další krok“ a „Předchozí krok“.

Speciální případ u procházení postupu je možno pozorovat u Ford-Fulkersonova algoritmu. Existuje zde totiž možnost si vybrat, jakou zlepšující cestu algoritmus použije. Pokud uživatel dojde do kroku, kde se vybírá zlepšující cesta, tak se mu v dolní liště objeví 3 nová tlačítka: „Předchozí“, „Vybrat“, a „Další“. Aktuální zlepšující cesta se uživateli zvýrazní zeleně a po kliknutí na tlačítko „Vybrat“ algoritmus pokračuje s tím, že použije zlepšující cestu, kterou si uživatel zvolil. Proces volby zlepšující cesty je možno vidět na obrázku níže.



Obrázek 53: Vybírání zlepšující cesty

4 TECHNICKÁ DOKUMENTACE

4.1 Použité technologie

4.1.1 Java

Java je programovací jazyk, který vyvinuli James Gosling, Patrick Naughton, Chris Warth, Ed Frank a Mike Sheridan ve společnosti Sun Microsystems, Inc. v roce 1991. Vývoj první funkční verze trval 18 měsíců. Původně se jazyk nejmenoval Java, ale nesl název "Oak", na dnešní název „Java“ byl přejmenován až v roce 1995. Hlavní motivací pro tvorby Javy byla potřeba platformově nezávislého systému, jazyka, který by se dal použít k vytváření softwaru pro zabudování do různých zařízení spotřební elektroniky jako jsou mikrovlnné trouby a dálkové ovladače. [27]

4.1.2 JavaFX

JavaFX je open source framework založený na Javě pro vývoj bohatých klientských aplikací. Je srovnatelný s jinými frameworky na trhu, jako jsou Adobe Flex a Microsoft Silverlight. JavaFX je také považován za nástupce Swingu v oblasti technologie vývoje grafických uživatelských rozhraní (GUI) na platformě Java. Knihovna JavaFX je k dispozici jako veřejné aplikační programové rozhraní (API) jazyka Java. JavaFX obsahuje několik funkcí, které z ní činí preferovanou volbu pro vývoj bohatých klientských aplikací. [28]

4.1.3 SQLite

SQLite je vestavěná relační databáze s otevřeným zdrojovým kódem. Původně byla vydána v roce 2000 a je určena k poskytování aplikacím pohodlný způsob správy dat bez režie, která je často spojena se specializovanými systémy pro správu relačních databází. SQLite má zaslouženou reputaci jako vysoce přenosná, snadno použitelná, kompaktní, efektivní a spolehlivá databáze. [29]

4.1.4 Launch4J

Launch4j je multiplatformní nástroj pro balení aplikací Java distribuovaných jako JAR soubory do lehkých nativních spustitelných souborů systému Windows. Spustitelný soubor lze nakonfigurovat tak, aby vyhledával určitou verzi JRE nebo používal přiloženou verzi, a je možné nastavit volby spouštění, například počáteční/maximální velikost haldy. Wrapper také poskytuje lepší uživatelské prostředí prostřednictvím ikony aplikace, nativní úvodní obrazovky před JRE a stránky pro stažení Javy v případě, že nelze nalézt odpovídající JRE. [30]

4.2 Databázová vrstva aplikace

Aplikace využívá databázi za účelem ukládání příkladů u témat, kde je složité generování grafů, u kterých se dané algoritmy mohou aplikovat. Jedná se tedy o témata: Ford-Fulkersonův algoritmus, kritická cesta a nejhořejší cesta.

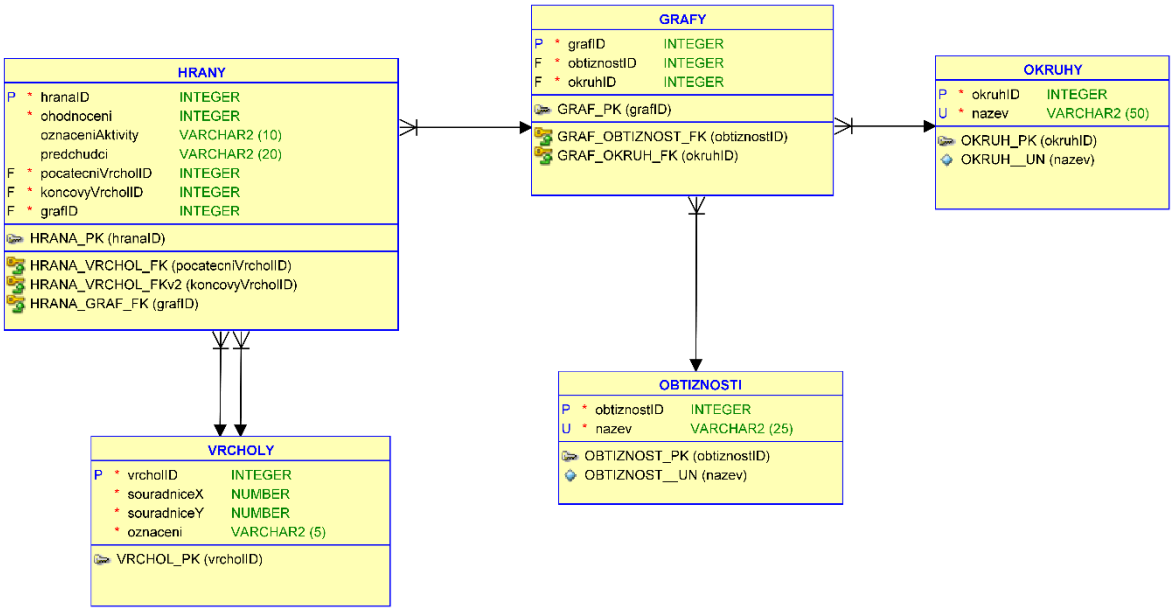
Požadavek na databázi byl takový, aby se jednalo o vestavěnou databázi, se kterou se jednoduše pracuje, bez nutnosti složitých SQL dotazů, pro naše účely totiž budeme pouze vybírat specifické příklady z databáze. Z těchto požadavků nám vzešly 2 hlavní kandidáti, a to Java Derby Database a SQLite. Z důvodu jednoduchosti používání byla zvolena databáze SQLite.

4.2.1 Relační model databáze

Relační model databáze byl vytvořen v programu Oracle SQL Developer Data Modeler.

Oracle SQL Developer Data Modeler je bezplatný grafický nástroj, který zvyšuje produktivitu a zjednodušuje úlohy modelování dat. Pomocí nástroje Oracle SQL Developer Data Modeler mohou uživatelé vytvářet, prohlížet a upravovat logické, relační, fyzické, vícerozměrné modely a modely datových typů. Data Modeler také umožňuje dopředné a zpětné inženýrství a podporuje společný vývoj prostřednictvím integrované kontroly zdrojového kódu. Data Modeler je také možno používat v cloudových prostředích. [31]

Tento program jsem zvolil kvůli předchozím zkušenostem s prací právě v tomto programu a pro naše účely jsou jeho funkce postačující. Níže můžete vidět výsledný relační model databáze, který si následně popíšeme.



Obrázek 54: Relační model databáze

Relační model obsahuje 5 tabulek:

Tabulka hrany – Jedná se o hlavní tabulku databáze, uchovává vždy ohodnocení hrany, které je povinné, dále samozřejmě jaký má hrana počáteční a koncový vrchol a také do jakého grafu patří. Poslední 2 vlastnosti (označení aktivity a předchůdci) se vztahují pouze na grafy, u kterých se pracuje s kritickou cestou, za účelem zobrazení tabulky aktivit.

Tabulka vrcholy – Jedná se o tabulku ukládající informace o vrcholech. Obsahuje údaje o souřadnici x a souřadnici y vrcholu, aby bylo možné zobrazit při načítání graf přesně tak, jak vypadal v původním zadání. Tabulka také obsahuje informaci o označení vrcholu, aby bylo vrcholy možno lépe identifikovat.

Tabulka grafy – Jedná se o tabulku, která uchovává informace o jednotlivých grafech. Obsahuje pouze cizí klíč na tabulku obtížnosti a cizí klíč na tabulku okruhy.

Tabulka obtížnosti – Jedná se o jednoduchý číselník, který obsahuje pouze ID a název obtížnosti, v našem případě se jedná pouze o varianty: lehké, středně náročné a těžké.

Tabulka okruhy – Také se jedná o jednoduchý číselník, který obsahuje pouze ID a název okruhu. Jsou zde zahrnuty všechny okruhy, které již byly popsány výše.

4.2.2 Použití databáze

Databáze je uložená v jediném souboru pojmenovaném *Grafy.db*, který se při spuštění aplikace načte do paměti a je s ním tak možno jednoduše pracovat. Zároveň to ale také znamená, že jakékoliv změny provedené v databázi se po ukončení programu do databáze nepropíšou a při dalším spuštění programu bude databáze v původním stavu, což nás v aktuální verzi aplikace netrápí, protože se z databáze pouze čte a žádným způsobem nezapisuje, ale je to věc, kterou musíme mít na mysli při případném rozšíření aplikace o tuto funkčnost.

Pro vytvoření a spravování databáze byl použit program *DB Browser for SQLite.*

DB Browser for SQLite (DB4S) je vysoce kvalitní, vizuální, open source nástroj pro vytváření, navrhování a úpravu databázových souborů kompatibilních s SQLite. DB4S je určen pro uživatele a vývojáře, kteří chtějí vytvářet, prohledávat a upravovat databáze. DB4S používá známé rozhraní podobné tabulkovému procesoru a je tak intuitivní a uživatel nepotřebuje mít znalost jazyka SQL. [32]

4.3 Aplikační vrstva aplikace

4.3.1 Rozvržení aplikace

Zdrojový kód aplikace je rozdělen do 4 hlavních balíčků: *algoritmy*, *data*, *gui* a *utils*.

V balíčku *gui* se nachází třídy, které zodpovídají za vzhled a funkčnost grafického zobrazení. Nachází se zde kódově nejobjemnější třída celého projektu, a to třída *FXMLHlavniOknoController*, která se stará o funkčnost celého hlavního okna programu. Aplikace disponuje pouze dvěma okny, proto se zde nachází pouze jedna další třída vytvářející nové okno, a to třída *FXMLTabulkaAktivitController,* která se stará o logiku při zobrazování tabulky aktivit Pokud pracujeme s grafem, tato třída se využívá pouze u tématu kritická sekce. Poslední třída, kterou tento balíček obsahuje, je hlavní třída nazvaná *VyukovaAplikace,* což je třída, která pouze nastaví základní parametry okna a chová se jako vstupní blok aplikace.

V balíčku *data* se nachází všechny třídy, které pracují s daty, v této aplikaci se tedy jedná o třídy:

- *Graf* – obsahuje informace o celkovém grafu, tj. drží seznam všech vrcholů a hran v grafu
- *Hrana* – obsahuje informace o dané hraně, mezi tyto informace patří například ohodnocení hrany, informace, zda se jedná o orientovanou hranu či ne, nebo odkazy na počáteční a koncový vrchol

- *Vrchol* – obsahuje informace o daném vrcholu, mezi tyto informace patří například označení vrcholu, dále souřadnice vrcholu v aktuálním grafu, nebo seznam všech jeho hran

Balíček *algoritmy* obsahuje rozhraní *Algoritmus*, které implementují všechny další algoritmy obsažené v tomto balíčku, kromě třídy *FordFulkerson,* u níž bylo nutné zvolit jiný způsob řešení kvůli umožnění uživateli vybrat si zlepšující cestu. Tento balíček také obsahuje další balíček s názvem *pomocne,* který obsahuje pomocné algoritmy, které se používají u hlavních algoritmů za účelem zpřehlednit kód.

V balíčku *utils* se nachází veškeré pomocné třídy, které se nehodí do žádných z předchozích balíčků, aby se celkově zpřehlednil kód. Nachází se zde například třída *DatabaseConnection*, která se stará o připojení do databáze. Dále se zde vyskytují třídy, které pomáhají vykreslovat graf do uživatelského rozhraní nebo konstanty, které se často používají v kódu.

4.4 Popis nejdůležitějších částí zdrojového kódu

4.4.1 Generování grafů z databáze

Jak již bylo popsáno při popisu databázové vrstvy, v této aplikaci existují 3 možnosti, jak je možno vygenerovat pro uživatele graf.

První možnost se vyskytuje u témat, kde je nutné mít předem definované příklady, jelikož generování příkladů, kde by bylo možné daný algoritmus provést, by bylo velice náročné. O tuto možnost se stará následující kód:

```
private void generovaniPrikladuZDatabaze() {
        tabulkaKritickeCesty.clear();
        SerializaceGrafu serializace = new SerializaceGrafu(pane,
zadani, stredyVrcholu, graf, vykreslovani, comboBoxTema,
cbObtiznostZadani);
        this.graf = serializace.nacteniGrafuZDatabaze(conn);
        ukazNode(hboxReseni, true);
        if (comboBoxTema.getValue().equals("Kritická cesta")) {
           this.tabulkaKritickeCesty=serializace.getTabulkaAkitivit();
            ukazNode(btnZadaniAktivit, true);
        } else {
            ukazNode(btnZadaniAktivit, false);
        }

spPocetVrcholu.getValueFactory().setValue(graf.getVrcholy().size());
        zadani.addAll(pane.getChildren());
        if (comboBoxTema.getValue().equals("Ford-Folkersonův
algoritmus")) {
            fordFolkersonuvAlgoritmus.setGraf(graf);
        } else {
            algoritmus.setGraf(graf);
        }
choiceBoxZacatekCesty.setValue(graf.getVrcholy().get(0).getOznaceni());
```

```
            choiceBoxKonecCesty.getSelectionModel().selectLast();
        }
```

V první části kódu se nejprve vyčistí tabulka kritické cesty, poté se vybere samotný graf z databáze. Zobrazí se *hBoxŘešení,* což je v uživatelském rozhraní hlavní ovládací prvek na spodní straně obrazovky. Pokud je zvoleno téma kritická cesta, tak se spolu s grafem ještě načte tabulka aktivit a zobrazí se tlačítko s možností zobrazit si nové okno se zadáním tabulky aktivit. Dále se nastaví spinner *spPocetVrcholu* na hodnotu počtu vrcholů v daném grafu. Celý graf se uloží do seznamu *zadani*, který se využije, pokud v budoucnu budeme chtít znovu zobrazit původní variantu grafu. Poté pouze nastavíme algoritmu graf, který bude používat. Poslední věc, kterou ještě musíme nastavit, je počáteční a koncový vrchol u daného algoritmu.

4.4.2 Generování zcela náhodných grafů

Druhou možnost generování grafu využívá aplikace u témat, u kterých je možné generovat zcela náhodný graf. O toto se stará následující kód:

```
private void generovaniGrafu() {
        ProhlidkaDoHloubky dfs = new ProhlidkaDoHloubky();
        pane.getChildren().clear();
        zadani.clear();
        stredyVrcholu.clear();
        graf = new Graf();
        generovaniNahodnychVrcholu();
        generovaniNahodnychHran();
        if (comboBoxTema.getValue().equals("Maximální dráha")) {
            while (testCyklicity(graf) || dfs.prohlidkaDoHloubky(graf)
!= spPocetVrcholu.getValue()) {
                generovaniNahodnychHran();
            }
        } else if (comboBoxTema.getValue().equals("Eulerovský tah")) {
            while (!jeMozneProvestEulerovskyTah(graf) ||
dfs.prohlidkaDoHloubky(graf) != spPocetVrcholu.getValue()) {
                generovaniNahodnychHran();
            }
        } else {
            while (dfs.prohlidkaDoHloubky(graf) !=
spPocetVrcholu.getValue()) {
                generovaniNahodnychHran();
            }
        }
        for (Hrana hrana : graf.getHrany()) {
            vykresleniGenerovaneHrany(hrana);
        }
        for (Node node : pane.getChildren()) {
            zadani.add(node);
        }
    }
```

První věc, kterou tento kód dělá, je vytvoření instance algoritmu prohlídky do hloubky. Tento algoritmus nám bude sloužit k ověření toho, zda je daný graf spojitý. Poté se pročistí všechna

data, která nějakým způsobem souvisela s eventuálním předchozím grafem, který již byl v aplikaci vygenerován.

Dále se zavolá metoda *generovaniNahodnychVrcholu*, která na náhodných souřadnicích vytvoří uživatelem zvolený počet vrcholů s tím, že kontroluje, aby se vrcholy navzájem nepřekrývaly.

Další metoda, která je v této metodě volána, je metoda *generovaniNahodnychHran*. Tato metoda nejprve vygeneruje náhodný počet hran podle úrovně obtížnosti a počtu vrcholů, poté se začnou generovat samotné hrany s tím, že probíhá kontrola, zda vygenerovaná hrana již v grafu není přítomna. Pokud se tak stane, tak se vygeneruje hrana nová. Jakmile se nám vygenerují všechny hrany, tak pokračujeme v metodě dál.

V další fázi se nám kontrolují všechny podmínky, které graf u daného tématu musí splňovat, například u grafu maximální dráhy musí být graf acyklický. Poté, co graf vyhovuje všem podmínkám stanoveným pro daný algoritmus, se může přejít na vykreslování všech hran a vrcholů.

4.4.3 Generování grafu s výběrem vlastních hran

Poslední možností, jak vygenerovat v aplikaci graf, je za pomoci vytvoření vlastních hran. O toto se stará následující kód:

```
private void generovaniVlastnichHran() {
        comboBoxTema.setDisable(true);
        spPocetVrcholu.setDisable(true);
        ukazNode(hBoxReseni, false);
        pane.getChildren().clear();
        zadani.clear();
        stredyVrcholu.clear();
        graf = new Graf();

        generovaniNahodnychVrcholu();
        editaceHran = true;
        algoritmus.setGraf(graf);
        ukazNode(btnGeneruj, false);
        ukazNode(btnDokoncitEditaci, true);
    }
```

První věc, která se musí zajistit, je znemožnění uživateli změnit téma, dokud nedokončí editaci, aby se předešlo nechtěným změnám. To samé také platí se spinnerem *spPocetVrcholu*, lišta na spodní straně obrazovky se také zneviditelní, dokud není editace dokončena. Opět se vyčistí všechna data o eventuálním předchozím grafu, aby byla aplikace připravena na nový graf. Na zobrazovací plochu, kde je umístěn graf, se vygenerují pouze vrcholy a uživateli je dána možnost si vytvořit vlastní hrany.

Po vygenerování vrcholů se objeví tlačítko „Dokončit editaci“, které má na sobě navázaný následující kód, který se spustí po stlačení tlačítka:

```
@FXML
    private void btnDokoncitEditaciOnAction(ActionEvent event) {
        if (comboBoxTema.getValue().equals("Eulerovský tah") &&
!jeMozneProvestEulerovskyTah(graf)) {
            zobrazChyboveOkno("Graf nemá požadovaný počet lichých
vrcholů!");
        } else if (comboBoxTema.getValue().equals("Maximální dráha") &&
testCyklicity(graf)) {
            zobrazChyboveOkno("Vytvořený graf je cyklický! Vytvořte graf
co cyklický není");
        } else {
            ProhlidkaDoHloubky dfs = new ProhlidkaDoHloubky();
            if (dfs.prohlidkaDoHloubky(graf) !=
spPocetVrcholu.getValue()) {
                zobrazChyboveOkno("Vytvořený graf je nesouvislý!
Vytvořte souvislý graf");
            } else {
                List<Hrana> noveHrany = eventListeners.getHrany();
                for (int i = 0; i < noveHrany.size(); i++) {
                    this.graf.getHrany().add(noveHrany.get(i));
                }
                zadani.clear();
                for (Node node : pane.getChildren()) {
                    zadani.add(node);
                }
                algoritmus.setGraf(graf);
                editaceHran = false;
                ukazNode(btnGeneruj, true);
                ukazNode(btnDokoncitEditaci, false);
                ukazNode(hBoxReseni, true);
                comboBoxTema.setDisable(false);
                spPocetVrcholu.setDisable(false);
            }
        }
    }
```

Tato část kódu se zejména věnuje kontrole, zda uživatelem vytvořený graf splňuje potřebné podmínky pro možnost provedení algoritmu na daném grafu. Pokud nějaká z podmínek není splněna, je zobrazeno chybové okno s odpovídající chybovou hláškou. Dále se nastaví hrany, zadání a také proměnná *editaceHran.* Poslední krok je zakrytí a zobrazení tlačítek, u kterých je to vyžadováno.

ZÁVĚR

Cílem bakalářské práce bylo vytvořit výukovou aplikaci pro teorii grafů. Myslím si, že tento cíl byl splněn, aplikace umožňuje procházet řešení algoritmů nebo přímo ukázat výsledek po provedení algoritmu. Myslím si, že se jedná o užitečnou pomůcku pro studenty při studování předmětu teorie grafů.

Na aplikaci bych v budoucnu vylepšil vizuální vzhled, aby vypadala pro uživatele z vizuálního hlediska více pohledněji, ale zároveň neztratila svou jednoduchost. Možné vylepšení by také bylo přidání možnosti testů, kde by se automaticky generovaly příklady, které by student musel vyřešit (například naklikat pořadí vrcholů, kterými by procházela nejkratší cesta u Dijkstrova algoritmu). Tyto testy by se mohly ukládat do databáze, aby mohly být použity pro více studentů a byla tak zachována stejná obtížnost pro všechny studenty. Další možností, jak vylepšit aplikaci, by bylo přidání funkce generování takových příkladů, které jsou zatím předem připravené a jsou pouze načítány z databáze. Nejsem si jistý, zda je tato možnost reálná, proto si myslím, že je ze všech možných vylepšení až na posledním místě.

Dle mého názoru má tato aplikace vysoký potenciál a v budoucnu bych ji chtěl dál vylepšovat, aby tohoto potenciálu skutečně dosáhla.

ZDROJE

[1] BALAKRISHNAN, *V.K. Schaum's Outline of Theory and Problems of Graph Theory* [online]. 1997 edition. New York: McGraw-Hill Education, 1997 [cit. 2022-03-23]. ISBN 0-07-005489-4. Dostupné z: https://kashanu.ac.ir/Files/GraphTheoryBalakrishnan1.pdf

[2] KOVÁŘ, Petr. *Úvod do Teorie grafů* [online]. Ostrava, 2012 [cit. 2022-03-23]. Dostupné z: https://mi21.vsb.cz/sites/mi21.vsb.cz/files/unit/uvod_do_teorie_grafu.pdf. Příprava přednášek a cvičení. Vysoká škola báňská – Technická univerzita Ostrava a Západočeská univerzita v Plzni.

[3] *Teorie grafů* [online]. Ostrava: Vysoká škola Báňská, 21.6.2006 [cit. 2022-03-24]. Dostupné z: http://books.fs.vsb.cz/SystAnal/texty/21.htm

[4] RAK, Josef. *Teorie grafů*. Univerzita Pardubice, c2011-2020 s. 1–5.

[5] JIROVSKÝ, Lukáš. *Teorie grafů ve výuce na střední škole*. Praha, 23.9.2010. Diplomová práce. Univerzita Karlova, Matematicko-fyzikální fakulta. Vedoucí práce RNDr. Pavla Pavlíková, Ph.D.

[6] Teorie grafů. *Voho* [online]. Vojtěch Hordejčuk, c2008-2022 [cit. 2022-03-28]. Dostupné z: http://voho.eu/wiki/graf/

[7] *Toky v sítích* [online]. Praha: Jakub Kárný, 2019 [cit. 2022-04-04]. Dostupné z: https://www2.karlin.mff.cuni.cz/~tuma/Aplikace19/Prace/Jakub_Karny.pdf

[8] HLINĚNÝ, Petr. *Základy Teorie Grafů: pro (nejen) informatiky* [online]. Brno, 19.3.2010 [cit. 2022-04-04]. Dostupné z: https://is.muni.cz/do/1499/el/estud/fi/js10/grafy/Grafy-text10.pdf. Výukový text. Masarykova univerzita, Fakulta informatiky.

[9] HLINĚNÝ, Petr. Toky v sítích [online]. Brno, [2007] [cit. 2022-04-05]. Dostupné z: https://is.muni.cz/el/1433/podzim2007/MA010/um/Grafy-lect--6.pdf

[10] Cyclic graph. *Wolfram MathWorld* [online]. Boston: Wolfram Research, 2014 [cit. 2022-04-10]. Dostupné z: https://mathworld.wolfram.com/CyclicGraph.html

[11] KELLER, Robert. M. Testing whether a grap his acyclic [online]. Claremont, [1998] [cit. 2022-04-10]. Dostupné z: https://www.cs.hmc.edu/~keller/courses/cs60/s98/examples/acyclic/

[12] KOLÁŘ, Josef. Teoretická informatika. 2. vyd. Praha: Česká informatická společnost, 2000. ISBN 80-900-8538-5.

[13] *Applications of Dijkstra's shortest path algorithm* [online]. Noida: GeeksforGeeks, 21.8.2020 [cit. 2022-04-16]. Dostupné z: https://www.geeksforgeeks.org/applications-of-dijkstras-shortest-path-algorithm/

[14] RAK, Josef. *Teorie grafů*. Univerzita Pardubice, c2011-2020 s 26–28.

[15] NEDVĚDOVÁ, Marie. *Maximální dráha*. Univerzita Pardubice, c2011-2020.

[16] MAS341: *Graph Theory* [online]. Sheffield: Paul Johnson, [2014] [cit. 2022-04-17]. Dostupné z: https://ptwiddle.github.io/Graph-Theory-Notes/s_graphalgorithms_longest-paths.html

[17] Brázdová, M. . (2007). VYUŽITÍ NĚKTERÝCH METOD TEORIE GRAFŮ PŘI ŘEŠENÍ DOPRAVNÍCH PROBLÉMŮ. *Perner's Contacts*, 2(1). Získáno z https://pernerscontacts.upce.cz/index.php/perner/article/view/1412

[18] Kelley, J. E., Walker, M. R., & Sayer, J. S. (1989). The origins of CPM: a personal history. *PM Network, 3*(2), 7–22.

[19] Project management through the 1950s [online]. Alan Stretton, 2011 [cit. 2022-04-21]. Dostupné z: https://projectmanager.com.au/project-management-through-the-1950s/2/

[20] *Metoda kritické cesty – CPM (Critical Path Method)* [online]. Česká republika: Management Mania, 2019 [cit. 2022-04-21]. Dostupné z: https://managementmania.com/cs/metoda-cpm

[21] Metoda kritické cesty [online]. San Francisco (CA): Wikipedia: the free encyclopedia, 22.3.2022 [cit. 2022-04-22]. Dostupné z: https://cs.wikipedia.org/wiki/Metoda_kritické_cesty

[22] *The Maximum Flow Problem* [online]. München: Quirin Fischer, 2015 [cit. 2022-04-25]. Dostupné z: https://algorithms.discrete.ma.tum.de/graph-algorithms/flow-ford-fulkerson/index_en.html

[23] RYBIČKOVÁ, Alena. *Network flow* [online]. Praha, 2019 [cit. 2022-04-25]. Dostupné z: https://www.fd.cvut.cz/personal/rybicale/TGA_04_Network_flow.pdf. Učební materiál. České vysoké učení technické v Praze, Fakulta dopravní.

[24] Bipartite graph. *Wolfram MathWorld* [online]. Boston: Wolfram Research, 11.11.2015 [cit. 2022-04-26]. Dostupné z: https://mathworld.wolfram.com/BipartiteGraph.html

[25] *Algorithms* [online]. Illinois: Jeff Erickson, 2019 [cit. 2022-04-26]. ISBN 978-1792644832 s. 361–365. Dostupné z: https://archive.org/details/Algorithms-Jeff-Erickson

[26] Eulerian Cycles: Why Are They So Unique, and Are They Significant to Us in the 21st Century?. Towards Data Science [online]. Indie: Jaival Patel, 2019 [cit. 2022-04-26]. Dostupné z: https://towardsdatascience.com/eulerian-cycles-why-are-they-so-unique-and-are-they-any-significant-to-us-in-the-21st-century-3ca489af585c

[27] SCHILDT, Herbert. *Java™: The Complete Reference*. Seventh Edition. Illinois: McGraw-Hill Osborne Media, 1.12.2006. ISBN 978-0072263855.

[28] HASHAN, Kishori. *Learn JavaFX 8*. Alabama: Apress, 2015. ISBN 978-1484211434.

[29] ALLEN, Grant a Michael OWENS. *The definitive guide to SQLite* [online]. 2nd ed. [New York]: Apress, c2010 [cit. 2022-05-01]. Expert's voice in open source (Apress). ISBN 978-1430232254. str. 23

[30] *JAR to EXE: Java program to Windows executable* [online]. Indie: Genuine Coder, 1.7.2022 [cit. 2022-05-01]. Dostupné z: https://genuinecoder.com/convert-java-jar-to-exe/

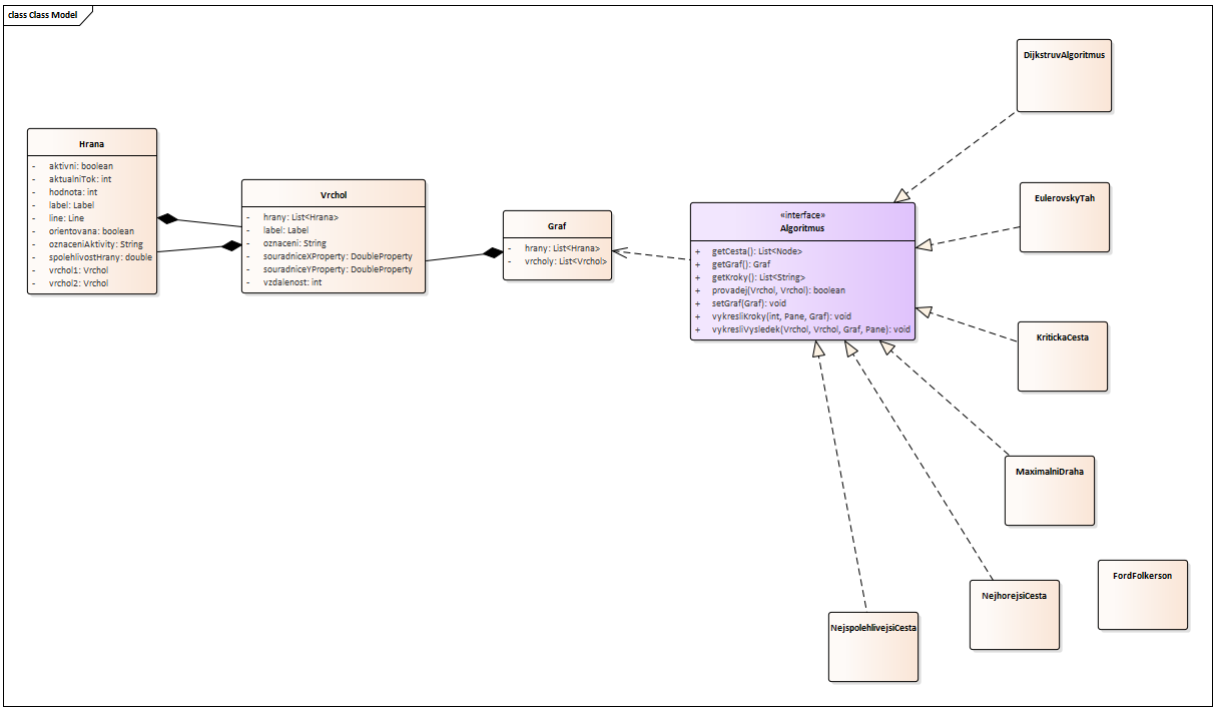
[31] *Data Modeling with Oracle SQL Developer* [online]. Austin: Oracle, 2018 [cit. 2022-05-02]. Dostupné z: https://www.oracle.com/cz/database/sqldeveloper/technologies/sql-data-modeler/

[32] *DB Browser for SQLite* [online]. New York: DigitalOcean, 2021 [cit. 2022-05-03]. Dostupné z: https://sqlitebrowser.org

[33] BAEK, Kyungim a Kazuo SUGIHARA. ICS 241: Discrete Mathematics II (Spring 2015) [online]. Hawaii, 2015 [cit. 2022-08-24]. Dostupné z: http://courses.ics.hawaii.edu/ReviewICS241/morea/graphs/Graphs4-QA.pdf. Učební materiál. University of Hawai‘i at Mānoa.

PŘÍLOHY

Příloha A – UML diagram soustředěný na interface Algoritmus



Příloha B – UML diagram obsahující detaily GUI