Katedra informatiky
Přírodovědecká fakulta
Univerzita Palackého v Olomouci

DIPLOMOVÁ PRÁCE

Program na podporu výuky toků v sítích

2023

Bc. Petr Jančár

Vedoucí práce:
doc. RNDr. Miroslav Kolařík, Ph.D.

Studijní program: Aplikovaná informatika,
Specializace: Vývoj software

Bibliografické údaje

Autor:	Bc. Petr Jančár
Název práce:	Program na podporu výuky toků v sítích
Typ práce:	diplomová práce
Pracoviště:	Katedra informatiky, Přírodovědecká fakulta, Univerzita Palackého v Olomouci
Rok obhajoby:	2023
Studijní program:	Aplikovaná informatika, Specializace: Vývoj software
Vedoucí práce:	doc. RNDr. Miroslav Kolařík, Ph.D.
Počet stran:	38
Přílohy:	elektronická data v úložišti katedry informatiky
Jazyk práce:	český

Bibliographic info

Author:	Bc. Petr Jančár
Title:	Educational support application for flows in networks
Thesis type:	master thesis
Department:	Department of Computer Science, Faculty of Science, Palacký University Olomouc
Year of defense:	2023
Study program:	Applied Computer Science, Specialization: Software Development
Supervisor:	doc. RNDr. Miroslav Kolařík, Ph.D.
Page count:	38
Supplements:	electronic data in the storage of department of computer science
Thesis language:	Czech

Anotace

Byl vytvořen program na podporu výuky toků v sítích v podobě webové aplikace, zaměřený na základní problém z této oblasti – nalezení maximálního toku v síti. Program nabízí možnost krok po kroku simulovat průběh Edmondsova-Karpova, Dinicova a Goldbergova algoritmu na zadané síti. Součástí práce je výukový text, pojednávající o základních pojmech z oblasti toků v sítích a zmíněných algoritmech pro řešení problému maximálního toku.

Synopsis

An educational support application for flows in networks was developed in the form of a web application, focusing on a fundamental problem in this area – finding the maximum flow of a flow network. The application offers the possibility to simulate step by step Edmonds-Karp, Dinic and Goldberg algorithms on a given flow network. The thesis also includes a study material covering the basic concepts in the field of flows in networks and the mentioned algorithms for solving the maximum flow problem.

Klíčová slova: teorie grafů; toky v sítích; problém maximálního toku; podpora výuky; webová aplikace

Keywords: graph theory; flows in networks; maximum flow problem; educational support; web application

Děkuji doc. RNDr. Miroslavu Kolaříkovi, Ph.D. za cenné rady a trpělivost při vedení této práce. Poděkování za podporu patří také rodině a přátelům.

Odevzdáním tohoto textu jeho autor místopřísežně prohlašuje, že celou práci včetně příloh vypracoval samostatně a za použití pouze zdrojů citovaných v textu práce a uvedených v seznamu literatury.

Obsah

1 Úvod **7**

2 Teoretická část **8**
2.1 Toky v sítích 8
2.2 Algoritmy pro problém maximálního toku 12
2.2.1 Fordův-Fulkersonův algoritmus 12
2.2.2 Edmondsův-Karpův algoritmus 13
2.2.3 Dinicův algoritmus 15
2.2.4 Goldbergův algoritmus 19

3 O programu a jeho vývoji **23**
3.1 Požadavky na program 23
3.2 Volba programovacího jazyka 23
3.3 Knihovny 24
3.4 Struktura webových stránek 24
3.5 Implementace 25
3.5.1 Implementace sítě 25
3.5.2 Implementace Edmondsova-Karpova algoritmu 26
3.5.3 Implementace Dinicova algoritmu 27
3.5.4 Implementace Goldbergova algoritmu 28
3.6 Uživatelská příručka 29
3.6.1 Předpoklady pro spuštění 29
3.6.2 Uživatelské rozhraní sekce pro simulaci algoritmu 29
3.6.3 Formát DOT 30
3.6.4 Omezení v programu 31
3.6.5 Klávesové zkratky 31

Závěr **34**

Conclusions **35**

A Obsah elektronických dat **36**

Literatura **37**

Seznam obrázků

1 Síť. 9
2 Tok v síti. 9
3 Řez určený množinou vrcholů $R = \{s, a, b\}$. Hrany, které náleží do řezu, jsou vyznačeny červeně. 11
4 Edmondsův-Karpův algoritmus, příklad. Počáteční síť. 13
5 Edmondsův-Karpův algoritmus, příklad. První průchod cyklem. . 14
6 Edmondsův-Karpův algoritmus, příklad. Druhý průchod cyklem. . 14
7 Edmondsův-Karpův algoritmus, příklad. Třetí průchod cyklem. . 14
8 Schematický nákres vrstevnaté sítě. Vrcholy jsou rozděleny do vrstev podle vzdálenosti od zdroje. 16
9 Dinicův algoritmus, příklad. Počáteční síť. 18
10 Dinicův algoritmus, příklad. Vrstevnatá síť po prvním průchodu vnitřním cyklem. 18
11 Dinicův algoritmus, příklad. Vrstevnatá síť po druhém průchodu vnitřním cyklem. 18
12 Goldbergův algoritmus, příklad. Počáteční síť. 21
13 Goldbergův algoritmus, příklad. Síť po počátečním nastavení výšek vrcholů a průtoků hranami. 21
14 Goldbergův algoritmus, příklad. Síť po zvýšení vrcholu a. 22
15 Goldbergův algoritmus, příklad. Síť po protlačení toku po hraně at. 22
16 Mapa webu. 25
17 Ukázková síť se čtyřmi vrcholy. 31
18 Uživatelské rozhraní sekce pro simulaci algoritmu. 33
19 Uživatelské rozhraní sekce pro import a export sítě. 33

1 Úvod

Představme si fiktivní město. V tomto městě se nachází vodárna, která poskytuje odběratelům v celém městě pitnou vodu. Voda je od vodárny k odběratelům distribuována vodovodním potrubím. Jednotlivé části potrubí mají danou svoji maximální propustnost. Otázka zní: jaké maximální množství vody může vodárna najednou vypustit odběratelům do vodovodního potrubí tak, aby nebyla poškozena žádná část potrubí?

Odpověď na tuto otázku nám může poskytnout oblast teorie grafů, která se obecně označuje jako toky v sítích. Formálně popsaná síť nám umožňuje modelovat výše popsané vodovodní potrubí, ale například také silniční nebo železniční síť, a na této formální síti řešit různé druhy problémů, týkajících se její propustnosti. Jedná se tak o velmi univerzální nástroj s širokou škálou využití.

Tato diplomová práce poskytuje studentům informatických oborů či dalším zájemcům program na podporu výuky toků v sítích, zejména pak na podporu výuky algoritmů řešících problém maximálního toku, provázaný s učebním textem, obsahujícím potřebný teoretický základ.

Jeden z existujících programů na podporu výuky toků v sítích se nachází na webových stránkách [1]. Tato webová aplikace vznikla v rámci diplomových prací studentů z německé Technische Universität München. Aplikace umožňuje uživateli například simulovat běh Fordova-Fulkersonova algoritmu pro nalezení maximálního toku v síti. Nenachází se zde však teoretický základ toků v sítích, například formální definice samotné sítě nebo toku v síti.

2 Teoretická část

Teoretická část této práce vznikla na základě zdrojů [3], [4], [5] a je koncipována jako podpůrný výukový text pro téma toků v sítích. Nejdůležitější poznatky z této kapitoly jsou ve zkrácené formě obsaženy také v programu na podporu výuky toků v sítích. Předpokládá se čtenářova základní znalost pojmů z oblasti teorie grafů a grafových algoritmů. V případě pochybností či nejasností se lze obrátit například na publikaci [7].

2.1 Toky v sítích

V úvodní kapitole jsme uvedli motivační příklad pro formální zavedení toků v sítích, týkající se distribuce vody ve vodovodní síti. Intuitivně si tuto síť představujeme jako systém propojených potrubí, který od vodárny (zdroj) přepravuje vodu do města (stok). Předpokládáme přitom, že se voda v potrubí nikde neztrácí[1]. Nyní pro toky v sítích zavedeme formální definici.

Definice 1 (Síť)
Síť je uspořádaná čtveřice $\langle G, s, t, c\rangle$, kde

- $G = \langle V, E\rangle$ je orientovaný graf,
- vrchol $s \in V$ je *zdroj*,
- vrchol $t \in V$ je *stok* (někdy označován také jako *spotřebič*),
- zobrazení $c : E \to \mathbb{R}^+$ udává *kapacitu* jednotlivých hran.

Číslu $c\,(e)$ říkáme *kapacita* hrany e.

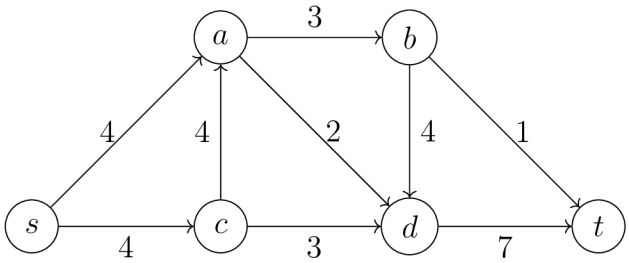
Příklad 2 (Síť)
Na obrázku 1 vidíme diagram sítě se šesti vrcholy. Vrchol s je zdroj, vrchol t je stok. Čísla u hran označují jejich kapacitu.

Definice 3 (Tok)
Tok f je zobrazení $f : E \to \mathbb{R}^+$, které splňuje

1. $\forall e \in E : \quad 0 \leq f\,(e) \leq c\,(e)$,
2. $\forall v \in V \setminus \{s, t\} : \quad \sum_{xv\in E} f\,(xv) = \sum_{vx\in E} f\,(vx)$.

Pro hrany $e = \langle x, v\rangle \in E$ využíváme alternativní značení $xv \in E$. Číslu $f\,(e)$ říkáme *průtok* hranou e. Tok ze zdroje s do stoku t explicitně značíme jako (s, t)-tok.

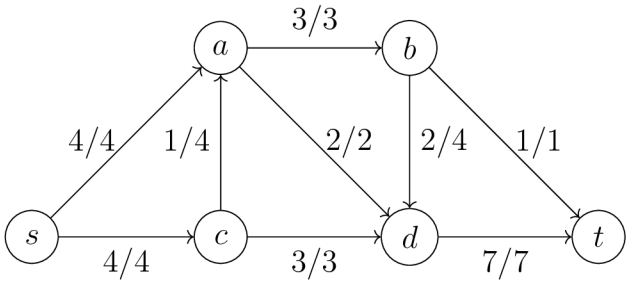
[1] Tento předpoklad v reálných vodovodních sítích neplatí. Společnost Pražské vodovody a kanalizace na svých webových stránkách uvádí, že v roce 2020 se v pražské vodovodní síti ztrácelo téměř 13 procent vody, viz [2].



Obrázek 1: Síť.

První podmínka v předchozí definici říká, že průtok hranou e je zdola omezený nulou a shora omezený kapacitou hrany e. Druhé podmínce se říká také *zákon o zachování toku*[2]. Tento název je odvozen od skutečnosti, že pro každý vrchol mimo zdroj a stok platí, že přítok do vrcholu je roven odtoku z vrcholu.

Příklad 4 (Tok v síti)
Na obrázku 2 vidíme tok v síti z předchozího příkladu. U každé hrany e je uveden její průtok a kapacita ve tvaru $f(e) \,/\, c(e)$.



Obrázek 2: Tok v síti.

Definice 5 (Bilance vrcholu)
Mějme libovolný vrchol $v \in V$. Číslu $b(v) = \sum_{xv \in E} f(xv) - \sum_{vx \in E} f(vx)$ říkáme *bilance vrcholu* v.

Bilance vrcholu tedy vyjadřuje rozdíl mezi přítokem a odtokem vrcholu. Z druhé podmínky definice 3 je zřejmé, že pro všechny vrcholy kromě zdroje a stoku platí, že jejich bilance je rovna nule. Nyní můžeme pomocí bilance vrcholu snadno definovat velikost toku.

[2]U elektrických obvodů je tento zákon znám jako *Kirchhoffův zákon*.

Definice 6 (Velikost toku)
Číslu $|f| = -b(s) = b(t)$ říkáme *velikost toku* f.

Velikost toku lze tedy vyjádřit jako zápornou hodnotu bilance zdroje nebo jako hodnotu bilance stoku. Největšímu možnému toku v dané síti říkáme *maximální tok*.

Příklad 7 (Velikost toku)
Tok v síti na obrázku 2 má velikost $|f| = b(t) = 8$. Jedná se o maximální tok v této síti.

Nalezení maximálního toku v síti je základní problém související s toky v sítích. S tímto problémem úzce souvisí pojem *řezu v síti*, jak dále uvidíme.

Definice 8 (Řez)
Mějme orientovaný graf $G = \langle V, E \rangle$, množinu $R \subseteq V$ a její doplněk $\overline{R} = V \setminus R$. Množinu orientovaných hran $\delta(R) = \left\{ vw \in E \mid v \in R \wedge w \in \overline{R} \right\}$ nazveme *řezem* určeným množinou R. Řez v síti $\langle G, s, t, c \rangle$ nazveme (s,t)-řez, pokud $s \in R$ a $t \notin R$.

Definice 9 (Kapacita řezu)
Kapacitu řezu $\delta(R)$ definujeme jako $c(\delta(R)) = \sum_{vw \in \delta(R)} c(vw)$.

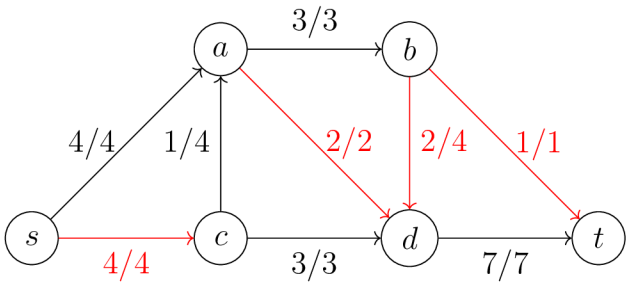
Řez nám tedy graf rozděluje na dvě disjunktní množiny vrcholů. V případě (s,t)-řezu platí, že odděluje zdroj od stoku. Kapacitu řezu vypočítáme jednoduše jako součet kapacit všech hran řezu.

Příklad 10 (Řez v síti)
Na obrázku 3 jsou v síti vyznačeny červenou barvou hrany, které náleží do (s,t)-řezu určeného množinou vrcholů $R = \{s, a, b\}$. Kapacita tohoto řezu je rovna číslu 11.

Lemma 11
Pro každý (s,t)*-tok* f *a každý* (s,t)*-řez* $\delta(R)$ *platí* $f(t) \leq c(\delta(R))$.

Věta 12 (O maximálním toku a minimálním řezu)
Pokud v síti existuje maximální (s,t)*-tok, pak* $\max\{|f| \mid f \text{ je } (s,t)\text{-tok}\} = \min\{c(\delta(R)) \mid \delta(R) \text{ je } (s,t)\text{-řez}\}$.

Lemma 11 nám říká, že velikost každého (s,t)-toku je shora omezena kapacitou libovolného (s,t)-řezu. Navíc díky větě 12 víme, že velikost maximálního (s,t)-toku je rovna kapacitě minimálního (s,t)-řezu. Nalezneme-li tedy řez a tok stejné velikosti, je jisté, že se jedná o minimální řez a maximální tok.



Obrázek 3: Řez určený množinou vrcholů $R = \{s, a, b\}$. Hrany, které náleží do řezu, jsou vyznačeny červeně.

Zamysleme se nyní, jakým způsobem bychom mohli hledat a zlepšovat toky v síti. Začněme s nulovým tokem. Postupně budeme tento tok zlepšovat tak, že pokud nalezneme cestu orientovanou směrem od zdroje ke stoku, pro jejíž každou hranu platí, že její průtok je menší než kapacita, pak maximálně možně zvětšíme průtok touto cestou a tím zvětšíme i průtok celého toku. Tímto přímočarým postupem bychom však nedosáhli nalezení maximálního toku pro všechny sítě. Háček je v hranách, které jsou orientovány směrem od stoku ke zdroji a mají nenulový průtok. Tyto hrany musíme při hledání maximálního toku také vzít v potaz. Pro takové hrany můžeme tok ve směru od zdroje ke stoku „simulovat“ odečtením od toku ve směru od stoku ke zdroji.

Pro zjednodušení budeme dále předpokládat, že ke každé hraně uv existuje opačná hrana vu. Pokud opačná hrana neexistuje, můžeme si ji s nulovou kapacitou snadno dodefinovat. Toto rozšíření grafu (které nijak nezmění výši maximálního toku) nám umožní snadněji definovat následující pojmy.

Definice 13 (Rezerva hrany)
Rezervu hrany uv definujeme jako $r(uv) = (c(uv) - f(uv)) + f(vu)$.

Definice 14 (Vylepšující cesta)
Vylepšující cesta je taková orientovaná cesta od zdroje ke stoku, jejíž všechny hrany mají nenulovou rezervu.

Lemma 15
Pokud v síti existuje vylepšující cesta pro tok f vedoucí od zdroje ke stoku, pak tok f není maximální.

Přirozeným důsledkem lemma 15 je fakt, že tok f je maximální, právě když neexistuje vylepšující cesta pro f.

Na základě uvedených poznatků a vlastností toků v sítích již můžeme představit algoritmus pro hledání maximálního toku v síti, který publikovali v roce 1956 Lester Randolph Ford Jr. a Delbert Ray Fulkerson [8].

2.2 Algoritmy pro problém maximálního toku

2.2.1 Fordův-Fulkersonův algoritmus

U Fordova-Fulkersonova algoritmu začínáme s nulovým tokem v síti. Postupně budeme v síti hledat vylepšující cesty a zvětšovat po nich aktuální tok (připomeňme, že můžeme využít i hrany orientované ve směru od stoku ke zdroji). Jakmile se dostaneme do situace, kdy již nebude existovat žádná další vylepšující cesta, pak podle lemma 15 víme, že jsme nalezli maximální tok v síti.

Procedure Ford-Fulkerson($\langle G, s, t, c \rangle$)

```
1 f = 0
2 while existuje vylepšující cesta do
3     nechť P je vylepšující cesta
4     vylepši tok podél P
5     aktualizuj f
6 return f
```

Jak je to s konečností Fordova-Fulkersonova algoritmu? Začněme situací, kdy všechny kapacity hran v síti jsou celočíselné. V takovém případě je hodnota f v každém kroku algoritmu navýšena minimálně o jedna. Protože součet kapacit všech hran je konečný, snadno vidíme, že algoritmus po konečném počtu kroků zastaví. To platí i pro síť s racionálními kapacitami. Jednoduchým přenásobením kapacit jejich nejmenším společným jmenovatelem totiž získáme opět síť s celočíselnými kapacitami hran. Situace se změní, pokud v síti povolíme iracionální kapacity hran. V takovém případě již není zaručeno, že algoritmus po konečném počtu kroků skončí. Může totiž dojít k situaci, kdy se algoritmus zacyklí a bude donekonečna konvergovat k jisté hodnotě. Příklad sítě, na které může dojít k zacyklení Fordova-Fulkersonova algoritmu, ukázal Uri Zwick [9].

Zamysleme se nyní nad situací po ukončení algoritmu. Vydal algoritmus opravdu maximální tok v síti? Podle věty 12 víme, že ověřit nalezení maximálního toku můžeme tím, že nalezneme řez s kapacitou stejné velikosti. Takový řez nalezneme tak, že po ukončení Fordova-Fulkersonova algoritmu nalezneme množinu C vrcholů, které jsou ze zdroje dosažitelné po hranách, které mají nenulovou rezervu. Do množiny C jistě náleží zdroj a nenáleží stok, $\delta(C)$ tedy tvoří (s,t)-řez a $c(\delta(C)) = f$. To znamená, že je zaručeno, že algoritmus vydá maximální tok v síti (za předpokladu, že nedojde k zacyklení).

Vraťme se nyní k pseudokódu algoritmu Ford-Fulkerson. Zbývá vyřešit otázku, jak vybírat vylepšující cestu na řádku 3 tak, abychom dosáhli výhodné časové složitosti. S přímočarým řešením přišli v roce 1972 Jack Edmonds a Richard Manning Karp [10].

2.2.2 Edmondsův-Karpův algoritmus

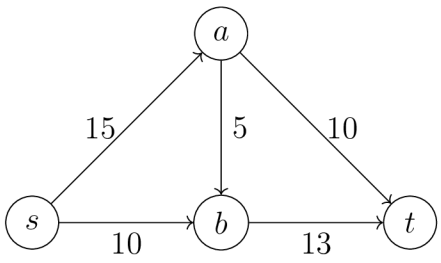
Úprava Fordova-Fulkersonova algoritmu, se kterým přišli Edmonds a Karp, spočívá v definování pořadí, ve kterém vybíráme vylepšující cesty na třetím řádku pseudokódu Ford-Fulkerson. Konkrétně vždy vybereme nejkratší vylepšující cestu, tedy cestu s minimálním počtem hran, viz pseudokód Edmonds-Karp.

Procedure Edmonds-Karp($\langle G, s, t, c \rangle$)

```
1 f = 0
2 while existuje vylepšující cesta do
3     nechť P je nejkratší vylepšující cesta
4     vylepši tok podél P
5     aktualizuj f
6 return f
```

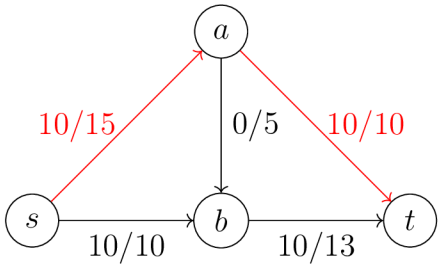
Při implementaci se k nalezení nejkratší vylepšující cesty využívá algoritmus prohledávání grafu do šířky, dále jen BFS (z anglického *Breadth-First Search*). Díky využití BFS lze algoritmus Edmonds-Karp implementovat s časovou složitostí $O(nm^2)$, kde n je počet vrcholů a m je počet hran grafu[3].

Příklad 16 (Edmondsův-Karpův algoritmus)
Na obrázku 4 je síť, na které si ukážeme průchod algoritmem Edmonds-Karp. Obrázek 5 zachycuje síť po prvním průchodu `while` cyklem. Červené hrany označují vylepšující cestu mezi vrcholy s, a, t, která byla vybrána pomocí BFS. Jako další byla vybrána vylepšující cesta mezi vrcholy s, b, t, viz obrázek 6. V dalším průchodu `while` cyklem byla vybrána vylepšující cesta mezi vrcholy s, a, b, t, viz obrázek 7 (všimněme si, že tato vylepšující cesta má více hran než cesta, která byla vybrána v minulém průchodu). Protože již další vylepšující cesta v síti neexistuje, další průchod `while` cyklem neproběhne. Snadno vidíme, že maximální tok v této síti má velikost 23.

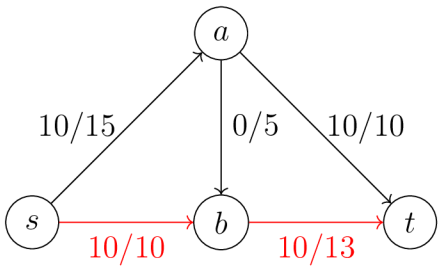


Obrázek 4: Edmondsův-Karpův algoritmus, příklad. Počáteční síť.

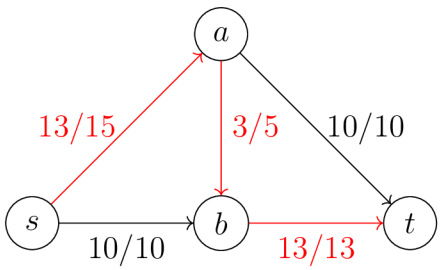
[3]Důkaz časové složitosti algoritmu Edmonds-Karp je k dispozici ve knize [6], kapitola 2.7.



Obrázek 5: Edmondsův-Karpův algoritmus, příklad. První průchod cyklem.



Obrázek 6: Edmondsův-Karpův algoritmus, příklad. Druhý průchod cyklem.



Obrázek 7: Edmondsův-Karpův algoritmus, příklad. Třetí průchod cyklem.

2.2.3 Dinicův algoritmus

S myšlenkou vylepšení časové složitosti Fordova-Fulkersonova algoritmu pomocí vybírání nejkratších vylepšujících cest přišel ve skutečnosti jako první sovětský vědec Yefim Dinitz[4], a to už v roce 1970 [11]. Nezávisle na něm publikovali algoritmus Edmonds a Karp v již zmíněném roce 1972, ale protože se jednalo o první publikaci v anglickém jazyce, algoritmus se obvykle označuje jako Edmondsův-Karpův [5]. Dinitz nicméně přišel s dalšími vylepšeními, díky kterým je jeho algoritmus ještě časově efektivnější, než výše popsaný Edmondsův-Karpův algoritmus.

Dinicův algoritmus je založen na následující myšlence. Namísto hledání vylepšujících cest jako ve Fordově-Fulkersonově algoritmu můžeme hledat a přičítat rovnou vylepšující toky. Podstatné přitom je, aby vylepšující toky dostatečně vylepšily stávající tok, ale zároveň šly jednoduše nalézt (snadněji než maximální tok). Zavedeme proto nyní pojem *blokujícího toku*, který tato kritéria splňuje.

Definice 17 (Blokující tok)
Blokující tok je takový tok, že každá orientovaná (s,t)-cesta obsahuje alespoň jednu hranu e takovou, že $r(e) = 0$.

Hraně e takové, že $r(e) = 0$ říkáme, že je *nasycená*. Orientovanou cestu nazveme *nasycenou*, obsahuje-li nasycenou hranu.

U definice blokujícího toku je důležité zdůraznit, že se hovoří o *orientovaných* cestách ze zdroje do stoku. Na rozdíl od předchozího algoritmu tak neuvažujeme i hrany a jejich rezervy v protisměru. Z toho vyplývá, že blokující tok nemusí být vždy tokem maximálním, protože může existovat vylepšující cesta obsahující hrany v protisměru.

Pro efektivní hledání blokujícího toku nyní definujeme dvě pomocné sítě – *síť rezerv* a *vrstevnatou síť* (často označovaná také jako *čistá síť*).

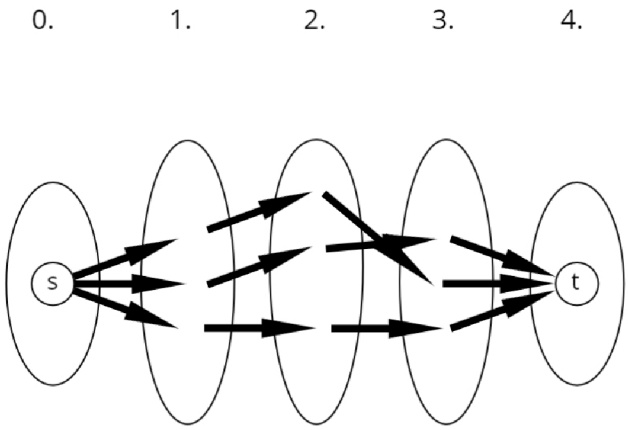
Definice 18 (Síť rezerv)
Nechť $S = \langle G, s, t, c\rangle$ je síť a f je tok v síti S. *Síť rezerv* příslušná k síti S a toku f je síť $S_R = \langle G, s, t, r\rangle$.

Síť rezerv tedy odpovídá původní síti s tím rozdílem, že hrany již nejsou ohodnoceny kapacitami, ale rezervami. Podotkněme, že při implementaci Edmondsova-Karpova algoritmu můžeme síť rezerv využít k nalezení nejkratší vylepšující cesty pomocí BFS.

Definice 19 (Vrstevnatá síť)
Nechť S_R je síť rezerv. *Vrstevnatá síť* S_V je podgraf sítě S_R takový, že obsahuje pouze hrany, které leží na některé nejkratší (s,t)-cestě.

[4]Dinitz bývá často překládán jako *Dinic*, odtud název *Dinicův algoritmus*.

Vrstevnatou síť lze zkonstruovat s využitím BFS. Vrcholy sítě jsou při konstrukci pomocí BFS rozděleny do vrstev podle vzdálenosti od zdroje (proto název *vrstevnatá* síť). Pro každou hranu uv sítě S_V platí, že vrchol v leží ve vrstvě o jedna vyšší než vrchol u.



Obrázek 8: Schematický nákres vrstevnaté sítě. Vrcholy jsou rozděleny do vrstev podle vzdálenosti od zdroje.

Procedure Dinic($S = \langle G, s, t, c \rangle$)

```
1  f = 0
2  spočítej síť rezerv S_R
3  while existuje vylepšující cesta v S_R do
4      spočítej vrstevnatou síť S_V
5      while existuje (s,t)-cesta v S_V do
6          nechť P je (s,t)-cesta v S_V
7          vylepši tok podél P
8          uprav S_V
9      aktualizuj f
10 return f
```

Podívejme se nyní na hlavní proceduru algoritmu Dinic. Podobně jako u předchozích algoritmů začínáme s nulovým tokem v síti. Na řádku 2 poté vypočítáme síť rezerv S_R.

V hlavní smyčce algoritmu (první smyčka `while` na řádku 3) kontrolujeme, zda ještě existuje vylepšující cesta v S_R. Pokud ne, pak podle lemma 15 víme, že jsme již nalezli maximální tok. Pokud vylepšující cesta byla nalezena, pokračujeme na řádku 4 a spočítáme vrstevnatou síť S_V na základě aktuální S_R.

Ve vnitřní smyčce algoritmu (druhá smyčka `while` na řádku 5) hledáme blokující tok v S_V následujícím způsobem. Nejprve hledáme v S_V (s,t)-cestu (díky vlastnostem vrstevnaté sítě víme, že se jedná o cestu vylepšující). Při implementaci se obvykle pro hledání využívá algoritmus prohledávání grafu do hloubky, dále jen DFS (z anglického *Depth-First Search*). Maximálně možně vylepšíme tok podél nalezené (s,t)-cesty a poté upravíme S_V. Vnitřní smyčku provádíme, dokud v S_V ještě existuje nějaká (s,t)-cesta. Nakonec zvýšíme globální hodnotu toku v síti f na základě dílčích toků, které jsme ve vnitřní smyčce vylepšili a které dohromady tvoří blokující tok.

Zbývá uvést, jakým způsobem probíhá úprava vrstevnaté sítě S_V na řádku 8. Protože se při vylepšení toku podél nalezené cesty některé hrany nasytí (jejich rezerva klesne na 0), musíme tyto hrany z S_V odebrat. Odebráním hran ale mohou vzniknout „slepé uličky“ – cesty vedoucí do vrcholů, ze kterých již nevede hrana do žádného dalšího vrcholu. Takové vrcholy (a hrany do nich vedoucí) je také potřeba z S_V odebrat, protože by nám komplikovaly nalezení (s,t)-cest v dalších iteracích vnitřní smyčky. Podotkněme ještě, že vylepšením toku podél (s,t)-cesty v S_V nemohou vzniknout nové cesty v S_R, které by měly kratší délku.

Skutečnost, že Dinicův algoritmus vydá maximální tok plyne z faktu, že algoritmus končí ve chvíli, kdy v síti již neexistuje další vylepšující cesta, jak jsme uvedli výše. Díky lemma 15 víme, že v ten moment jsme již nalezli maximální tok.

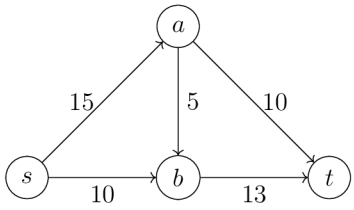
Dinicův algoritmus lze implementovat s časovou složitostí $O(n^2m)$, kde n je počet vrcholů a m je počet hran grafu[5]. Pro úplnost uveďme variantu Dinicova algoritmu, pro kterou se vžil název Algoritmus tří Indů [12]. Ten publikovali v roce 1978 Indové Malhotra, Kumar a Maheshwari. Ukázali efektivnější způsob pro hledání blokujícího toku, který vylepšuje výslednou časovou složitost algoritmu na $O(n^3)$ kde n je počet vrcholů grafu.

Příklad 20 (Dinicův algoritmus)

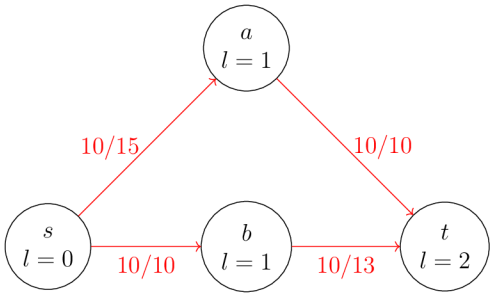
Průběh algoritmu Dinic si ukážeme na síti z obrázku 4. Zaměříme se přitom na hledání blokujícího toku ve vrstevnaté síti. Na začátku algoritmu rezervy hran odpovídají kapacitám hran. Na základě sítě rezerv je vypočítána vrstevnatá síť na obrázku 10. Hodnota l vrcholu přitom odpovídá vrstvě, ve které se daný vrchol nachází. Ve vnitřním cyklu algoritmu jsou nalezeny dvě (s,t)-cesty – první mezi vrcholy s,b,t a druhá mezi vrcholy s,a,t. Tyto dvě cesty dohromady tvoří blokující tok nalezený v prvním průchodu vnitřním cyklem. Na obrázku 10 je blokující tok znázorněn červenými hranami. Protože stále existuje vylepšující cesta v síti rezerv, algoritmus pokračuje. Jelikož došlo k nasycení hran sa a sb, je vypočítána nová vrstevnatá síť na obrázku 11. Všimněme si, že vrcholy b a t se tentokrát nachází v jiných vrstvách. Vrstevnatá síť tak nyní obsahuje i hranu ab. Ve vnitřním cyklu algoritmu je nalezena (s,t)-cesta mezi vrcholy s,a,b,t. Tato cesta tvoří blokující tok nalezený v druhém průchodu vnitřním cyklem. Na obrázku 11 je blokující tok znázorněn červenými hranami. Protože již neexistuje

[5]Důkaz časové složitosti algoritmu Dinic je k dispozici ve knize [6], kapitola 4.1.

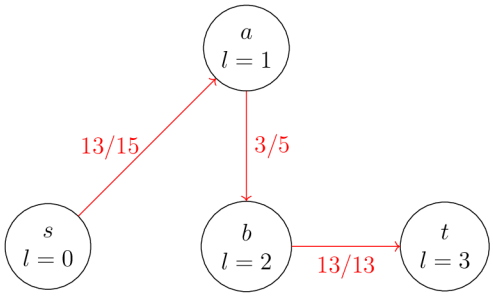
další vylepšující cesta v síti rezerv, algoritmus končí.



Obrázek 9: Dinicův algoritmus, příklad. Počáteční síť.



Obrázek 10: Dinicův algoritmus, příklad. Vrstevnatá síť po prvním průchodu vnitřním cyklem.



Obrázek 11: Dinicův algoritmus, příklad. Vrstevnatá síť po druhém průchodu vnitřním cyklem.

2.2.4 Goldbergův algoritmus

Posledním algoritmem pro nalezení maximálního toku v síti, který představíme, je Goldbergův algoritmus[6]. Publikoval jej v roce 1986 Andrew Vladislav Goldberg společně s Robertem Tarjanem pod názvem „A new approach to the maximum flow problem“ [13]. Už z názvu jejich publikace je zřejmé, že se jedná o odlišný přístup k řešení problému, než v případě předchozích dvou algoritmů.

U Fordova-Fulkersonova (respektive Edmondsova-Karpova) i Dinicova algoritmu jsme začínali s nulovým tokem v síti, který jsme postupně vylepšovali. Po celou dobu běhu těchto algoritmů tak byl udržován korektní tok. V případě Goldbergova algoritmu však začínáme s ohodnocením hran, které vůbec nemusí tvořit tok. Zavedeme pro něj označení *předtok*[7].

Definice 21 (Předtok)
Předtok f je zobrazení $f : E \to \mathbb{R}^+$, které splňuje

1. $\forall e \in E : \quad 0 \leq f(e) \leq c(e)$,
2. $\forall v \in V \setminus \{s, t\} : \quad b(v) \geq 0$.

Z definice předtoku vidíme, že každý tok je zároveň předtok. Předtok se stane tokem ve chvíli, kdy pro každý vrchol kromě zdroje a stoku platí, že jeho bilance je rovna nule.

Na začátku Goldbergova algoritmu začínáme s předtokem takovým, že $f(e) = c(e)$ pro všechny hrany $e \in E$, které vedou ze zdroje s. Ostatním hranám nastavíme nulový průtok.

Základní myšlenkou algoritmu je snižování bilance vrcholů na nulu pomocí operace *protlačení toku*. Protlačení toku po hraně $uv \in E$ lze provést, pokud platí $b(u) > 0$ a zároveň $r(uv) > 0$. Protlačením toku po hraně $uv \in E$ pošleme po hraně $\min(b(u), r(uv))$ jednotek toku (přičtením po směru hrany, nebo odečtením proti směru hrany).

Operací protlačení budeme chtít postupně protlačit tok ve směru od zdroje ke stoku a tím snížit bilanci vrcholů na nulu. Pokud při procesu protlačování dojdeme až do zdroje a některé vrcholy stále budou mít nenulovou bilanci, bude potřeba provést protlačování toku v opačném směru, zpět do zdroje. Proces protlačování toku tak v jistém smyslu může připomínat šíření vlny, která se v první fázi šíří směrem od zdroje ke stoku, kde se odrazí a šíří se zpět opačným směrem.

Při procesu protlačování je ovšem potřeba dávat pozor na to, abychom se nezacyklili. Proto vrcholům $v \in V$ přiřadíme číslo $h(v) \in \mathbb{N}$, kterému říkáme *výška vrcholu*. Na začátku algoritmu přiřadíme zdroji výšku n (počet vrcholů grafu) a všem ostatním vrcholům výšku 0. Operaci protlačení toku po hraně $uv \in E$ pak provedeme pouze pokud platí $h(u) > h(v)$. Intuitivně si můžeme představit,

[6] V anglicky psané literatuře se Goldbergův algoritmus častěji označuje jako *Push-Relabel algorithm* nebo také *Preflow-Push algorithm*. Názvy vyplývají ze základních operací algoritmu.

[7] Název *předtok* vychází z anglického výrazu *preflow*. V české literatuře se lze setkat také s pojmy *pratok* nebo *vlna*.

že se tekutina přelije pouze z vyššího vrcholu do nižšího. V případě, kdy z vrcholu s nenulovou bilancí nevede žádná nenasycená hrana směrem k nižšímu vrcholu, provedeme operaci *zvýšení vrcholu*. Zvýšením vrcholu $v \in V$ zvýšíme hodnotu $h(v)$ o 1. V procesu zvyšování budeme pokračovat, než bude vrchol dostatečně vysoko, aby z něj bylo možné protlačit tok do některého nižšího vrcholu, viz pseudokód Goldberg.

Procedure Goldberg($S = \langle G, s, t, c \rangle$)

```
1  h(s) = n
2  pro všechny v ≠ s :  h(v) = 0
3  f = 0
4  pro všechny v takové, že sv ∈ E :  f(sv) = c(sv)
5  while existuje u ∈ V \ {s, t} takový, že b(u) > 0 do
6      if existuje uv ∈ E taková, že r(uv) > 0 a h(u) > h(v) then
7          protlač tok po hraně uv
8      else
9          h(u) = h(u) + 1
10 return b(t)
```

Z popisu Goldbergova algoritmu nemusí být na první pohled zřejmé, zda ve chvíli, kdy algoritmus zastaví, skutečně vydá korektní tok. Pro předtok platí stejné omezení na kapacity hran jako pro tok, stačí tedy ukázat, že platí také zákon o zachování toku. Ten platí, protože algoritmus končí ve chvíli, kdy je bilance všech vrcholů kromě zdroje a stoku nulová.

Víme tedy, že Goldbergův algoritmus po zastavení vydá korektní tok. Jedná se však zároveň o tok maximální? Dokážeme sporem. Předpokládejme, že tok, který vydá Goldbergův algoritmus, není maximální. Z lemma 15 víme, že v síti tak musí existovat vylepšující cesta. Uvažme některou vylepšující cestu. Z pseudokódu Goldberg je zřejmé, že zdroj je po celou dobu běhu algoritmu ve výšce 0 a stok je ve výšce n. Vylepšující cesta tedy překonává výškový rozdíl n, ale může mít maximálně $n-1$ hran. Víme tedy, že se v ní nachází jedna hrana s výškovým rozdílem mezi vrcholy o velikosti alespoň 2. Protože je tato hrana součástí vylepšující cesty, je jistě nenasycená. To je ale spor, protože z pseudokódu Goldberg očividně nemůže existovat hrana uv, která by měla kladnou bilanci a výškový rozdíl mezi jejími vrcholy $h(u) - h(v)$ vetší než 1. Tok, který vydá Goldbergův algoritmus, je tedy maximální.

Časová složitost Goldbergova algoritmu závisí na způsobu, kterým vybíráme vrchol na řádku 5. Základní variantu Goldbergova algoritmu, tak jak jsme ji popsali výše, lze implementovat s časovou složitostí $O(n^2 m)$, kde n je počet vrcholů a m je počet hran grafu. V praxi se díky výhodnější časové složitosti často používá varianta *Maximum-distance*, ve které na řádku 5 vybíráme vždy nejvyšší vrchol s kladnou bilancí. Tato úprava zajistí časovou složitost $O(n^3)$,

kde n je počet vrcholů grafu[8].

Příklad 22 (Goldbergův algoritmus)

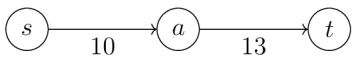
Protože již u sítí s několika málo vrcholy vyžaduje výpočet maximálního toku pomocí Goldbergova algoritmu poměrně vysoký počet kroků, jsou v tomto příkladu demonstrovány základní operace Goldbergova algoritmu na jednoduché síti se třemi vrcholy z obrázku 12.

Na začátku algoritmu je nastavena výška zdroje na počet vrcholů v síti a výška všech ostatních vrcholů na 0. Poté je nastaven průtok každou hranou na 0. Protože hrana sa vede ze zdroje, je průtok touto hranou nastaven na hodnotu její kapacity, tedy 10. Tím byly provedeny kroky 1 až 4 pseudokódu Goldberg. Stav sítě po provedení těchto kroků je zachycen na obrázku 13. U každého vrcholu je pod jeho názvem uvedena hodnota bilance b tohoto vrcholu a dále jeho výška h.

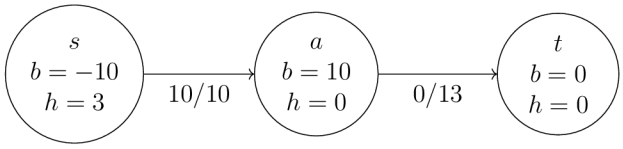
Ve hlavní smyčce algoritmu je poté vybrán vrchol a, jehož bilance je 10. Protože vrchol a má stejnou výšku jako vrchol t, nelze zatím protlačit tok po hraně st. Je tak provedena operace zvýšení vrcholu a, viz obrázek 14.

V další iteraci hlavní smyčky algoritmu je opět vybrán vrchol a. Protože již má vrchol a výšku 1, je možné z něj protlačit tok o velikosti 10 po hraně at do vrcholu t, který má výšku 0. Stav sítě po provedení operace protlačení toku je zachycen na obrázku 15.

V další iteraci hlavní smyčky algoritmu již není nalezen žádný vrchol (kromě zdroje a stoku) s nenulovou bilancí. Algoritmus tak končí a vydává velikost maximálního toku v síti, vypočítanou jako bilanci stoku.

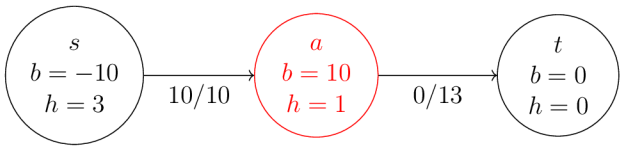


Obrázek 12: Goldbergův algoritmus, příklad. Počáteční síť.

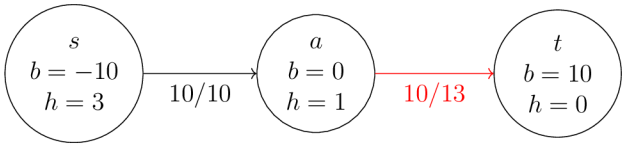


Obrázek 13: Goldbergův algoritmus, příklad. Síť po počátečním nastavení výšek vrcholů a průtoků hranami.

[8]Důkaz časové složitosti základní varianty algoritmu Goldberg a varianty *Maximum-distance* je k dispozici ve knize [4], kapitoly 14.5 a 14.6.



Obrázek 14: Goldbergův algoritmus, příklad. Síť po zvýšení vrcholu a.



Obrázek 15: Goldbergův algoritmus, příklad. Síť po protlačení toku po hraně at.

3 O programu a jeho vývoji

V této kapitole pojednám o vývoji hlavního výstupu této diplomové práce – programu na podporu výuky toků v sítích. Objasním požadavky na program, volbu programovacího jazyka, stručně představím použité technologie a knihovny a popíši důležité části týkající se návrhu a implementace programu. Sekce 3.6 slouží jako uživatelská příručka.

3.1 Požadavky na program

Požadavky na program vychází ze zadání diplomové práce, zahrnují následující:

- možnost vytvoření sítě,
- v zadané síti možnost vyřešení problému úlohy nalezení maximálního toku v síti třemi různými algoritmy,
- možnost sledovat průběh algoritmu.

3.2 Volba programovacího jazyka

Volba programovacího jazyka je zásadní moment při vývoji jakéhokoliv programu. Je potřeba zvážit celou řadu aspektů, v případě tohoto programu zejména:

- vhodnost programovacího jazyka pro daný typ programu,
- dostupnost potřebných knihoven v daném jazyce,
- jednoduchost distribuce programu k uživateli.

U praktických implementací algoritmů pro problém maximálního toku bývá přirozeným požadavkem také výkon implementace. U výukového programu, kde se pro přehlednost předpokládá běh na malých sítích (například do 10 vrcholů), však lze před výkonností upřednostnit přehlednost a jednoduchost použití.

Aby bylo pro uživatele co nejjednodušší začít program používat, byla zvolena forma webové aplikace, která nevyžaduje instalaci na klientský počítač a je nezávislá na operačním systému. Z tohoto důvodu budu dále program označovat slovem aplikace, které je ve webovém prostředí běžnější. K běhu aplikace postačí běžný internetový prohlížeč. Logickou volbu při výběru programovacího jazyka pro vývoj webových aplikací je *JavaScript*. Jedná se o široce rozšířený multiplatformní jazyk, využívaný na straně klienta i na straně serveru. V případě tohoto programu je JavaScriptový kód interpretován na straně klienta po stažení webové stránky.

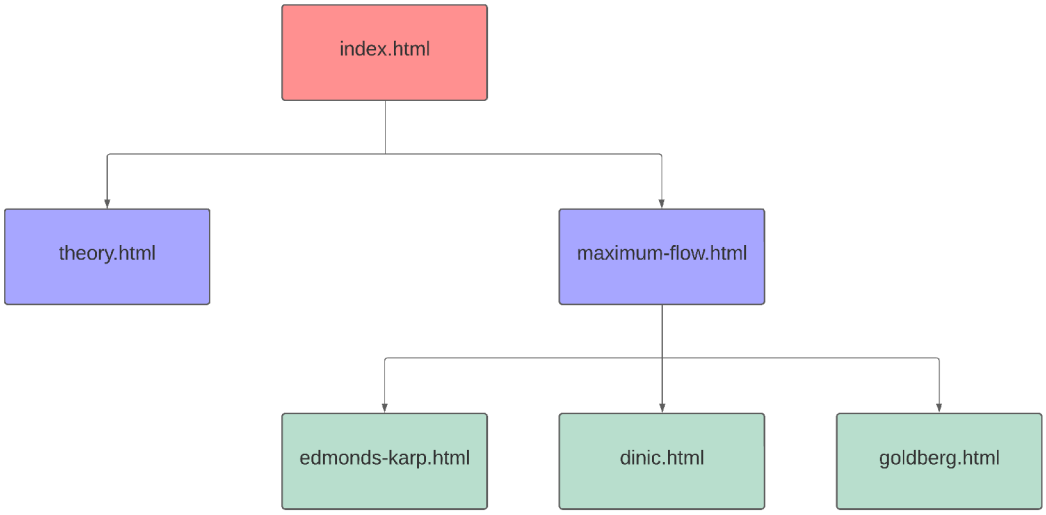
3.3 Knihovny

Nespornou výhodou vývoje aplikací v JavaScriptu je dostupnost velkého množství kvalitních knihoven třetích stran. Pro vývoj této aplikace jsem využil následující knihovny:

- **Bootstrap** [14] **(verze 5.1.3)** – open-source front-end toolkit, který umožňuje rychlé vytváření responzivních webových stránek za pomoci sady předstylovaných UI komponent. Výhodou je jednoduchost použití, rychlost vývoje aplikace a široká sada předstylovaných UI komponent.
- **Font Awesome** [15] **(verze 6.4.0)** – open-source knihovna poskytující velké množství kvalitních ikon. Existuje placená i neplacená verze, které se liší zejména v množství ikon. Pro vývoj této aplikace jsem využil neplacenou verzi.
- **MathJax** [16] **(verze 3.2.2)** – open-source knihovna pro zobrazování matematických symbolů na webové stránce. Podporuje podmnožinu matematické notace LaTeX–u.
- **jQuery** [17] **(verze 3.7.0)** – open-source knihovna, která usnadňuje použití jazyka JavaScript při vývoji webových stránek. Zjednodušuje například manipulaci s HTML a CSS nebo použití asynchronního JavaScript–u a XML (AJAX).
- **vis.js** [18] **(verze 9.1.0)** – open-source knihovna, umožňující dynamické zobrazování a interakci s grafy na webové stránce. Knihovnu jsem zvolil, protože je dostatečně flexibilní a umožňuje tak dále rozšiřovat její funkcionalitu. V neposlední řadě obsahuje detailní dokumentaci.

3.4 Struktura webových stránek

Hierarchická struktura webových stránek, neboli mapa webu, je zachycena na obrázku 16. První stránka, kterou uživatel uvidí, je `index.html`. Jedná se o úvodní stranu s informacemi o diplomové práci a jejím účelu. Z úvodní strany je možné přejít na stránku `theory.html`, která obsahuje shrnutí základních teoretických pojmů, nebo na stránku `maximum-flow.html`. Tato stránka slouží jako rozcestník pro výběr algoritmů pro řešení problému maximálního toku. Na výběr jsou tři algoritmy – Edmondsův-Karpův (`edmonds-karp.html`), Dinicův (`dinic.html`) a Goldbergův (`goldberg.html`). Stránky věnované algoritmům obsahují základní teoretický popis včetně odkazů do literatury a především část pro sledování průběhu algoritmu. Rozložení uživatelského rozhraní této části (zejména ovládací panel) je inspirováno aplikací [1].



Obrázek 16: Mapa webu.

3.5 Implementace

3.5.1 Implementace sítě

Jak již bylo uvedeno v přehledu použitých knihoven, pro zobrazování sítě jsem využil knihovnu *vis.js*. Tato knihovna obsahuje několik komponent, z nichž jsem využil dvě – komponenty `Network` a `DataSet`.

`Network` je objekt, který slouží k vizualizaci grafů na webové stránce. Obsahuje množinu vrcholů a množinu hran, reprezentované v podobě objektů `DataSet`.

`DataSet` je objekt, který slouží k uchovávání nestrukturovaných dat a manipulaci s nimi. Umožňuje naslouchat změnám v datech a na základě změny podniknout uživatelem definovanou akci.

Uživatel má na výběr tři možnosti, jak zadat síť. První možnost je využít předpřipravené sítě. Ty jsou interně uloženy v DOT formátu, který je blíže představen v sekci 3.6.3. Vis.js obsahuje funkci, která z řetězce v DOT formátu vytvoří objekt `Network`. Druhá možnost je zadání sítě v DOT formátu (ručně nebo nahráním souboru), kde se využívá stejné funkce. Třetí možnost je vytvoření sítě ručním zadáváním vrcholů a hran na webové stránce. Zde bylo nutné implementovat přidávání vrcholů a hran do objektu `Network`, včetně omezení na zadávané hodnoty a zobrazování dialogů, nebo některých klávesových zkratek.

Síť reprezentovaná objektem `Network` je uložena v globální proměnné nazvané `network`. Množina vrcholů je reprezentována objektem `DataSet` a uložena v globální proměnné `networkVertices`. Obdobně je množina hran reprezentována objektem `DataSet` a uložena v globální proměnné `networkEdges`. Deklarace zmíněných proměnných a související funkcionalita se nachází v souboru `base.js`.

3.5.2 Implementace Edmondsova-Karpova algoritmu

Implementace Edmondsova-Karpova algoritmu se nachází v souboru `edmonds-karp.js`.

Pro interní reprezentaci stavu, ve kterém se algoritmus právě nachází, slouží objekt `AlgData`, který je uložen v globální proměnné `algData` a vytváří se vždy na začátku běhu algoritmu, tedy po kliknutí uživatele na tlačítko „Start“. Interně jsou pro potřeby algoritmu vrcholy a hrany sítě reprezentovány objekty `Vertex` a `Edge`. Ty jsou pak v poli uloženy ve vlastnostech `vertices` a `edges` objektu `AlgData`. V mapě, která je uložena ve vlastnosti `vertexIndexMap` objektu `AlgData`, je každému vrcholu přiřazeno přirozené číslo, které je poté použito jako index v interní maticové reprezentaci grafu sítě, uložené taktéž v objektu `AlgData`. V případě Edmondsova-Karpova algoritmu se využívá matice kapacit `capacityMatrix` a matice rezerv `residualMatrix`. Hodnota maximálního toku v síti, která je průběžná vypočítávána, je uložena ve vlastnosti `maxFlow` objektu `AlgData`.

Samotný Edmondsův-Karpův algoritmus je v této implementaci rozdělen do několika stavů:

1. `init` – počáteční fáze algoritmu (vybírání zdroje a stoku),
2. `initFlow` – inicializace nulového toku v síti,
3. `mainLoop` – nalezení nejkratší vylepšující cesty pomocí BFS,
4. `showPath` – zvýraznění nejkratší vylepšující cesty,
5. `pushFlow` – vylepšení toku podél nejkratší vylepšující cesty,
6. `updateFlow` – navýšení hodnoty aktuálního toku v síti,
7. `finished` – fáze po dokončení běhu algoritmu.

Stavy 2 až 7 odpovídají jednotlivým řádkům pseudokódu Edmonds-Karp. Aktuální stav algoritmu je uložen ve vlastnosti `algState` objektu `AlgData`.

Při kliknutí uživatele na tlačítko „Vpřed“ se na základě aktuálního stavu provede příslušná operace voláním funkce `algMainLoop()`, v případě potřeby se aktualizuje vizualizace sítě a algoritmus přejde do následujícího stavu.

V případě, že uživatel klikne na tlačíko „Na konec“, se v cyklu volá funkce `algMainLoop()`, dokud algoritmus není ve stavu `finished`.

V objektu `AlgData` je ve vlastnosti `mainLoopStep` uloženo číslo, které značí, kolik kroků algoritmu již bylo provedeno. Tato znalost je užitečná v případě, kdy uživatel klikne na tlačítko „Zpět“. Řekněme, že v momentu kliknutí uživatele bylo provedeno n kroků algoritmu. Algoritmus je poté znovu restartován od začátku a je provedeno $n - 1$ kroků, respektive volání funkce `algMainLoop()`.

Po kliknutí uživatele na tlačítko `Reset` je navrácena vizualizace sítě do původního stavu (před spuštěním algoritmu) a objekt `algData` nastaven na `null`.

3.5.3 Implementace Dinicova algoritmu

Implementace Dinicova algoritmu se nachází v souboru `dinic.js`.

Obdobně jako u Edmondsova-Karpova algoritmu slouží pro reprezentaci stavu, ve kterém se algoritmus nachází, objekt `AlgData`, který je uložen v globální proměnné `algData`. V objektu `AlgData` je v případě Dinicova algoritmu navíc kromě matice kapacit a matice rezerv uložena také maticová reprezentace vrstevnaté sítě a dále pole `level`, ve kterém jsou uloženy vrstvy, ve kterých leží jednotlivé vrcholy. Kromě proměnné `maxFlow` je v objektu `AlgData` uložena také proměnná `blockingFlow`, ve které se postupně ukládá velikost hledaného blokujícího toku.

Dinicův algoritmus je v této implementaci rozdělen do následujících stavů:

1. `init` – počáteční fáze algoritmu (vybírání zdroje a stoku),
2. `initFlow` – inicializace nulového toku v síti,
3. `computeResidualGraph` – výpočet sítě rezerv,
4. `mainLoop` – kontrola, zda v síti rezerv existuje vylepšující cesta (implementováno pomocí BFS),
5. `computeLevelGraph` – výpočet vrstevnaté sítě,
6. `innerLoop` – kontrola, zda ve vrstevnaté síti existuje (s, t)-cesta (implementováno pomocí DFS),
7. `showPath` – zvýraznění nalezené (s, t)-cesty,
8. `pushFlow` – vylepšení toku podél nalezené (s, t)-cesty,
9. `updateLevelGraph` – úprava vrstevnaté sítě,
10. `updateFlow` – navýšení hodnoty aktuálního toku v síti o velikost nalezeného blokujícího toku,
11. `finished` – fáze po dokončení běhu algoritmu.

Stavy 2 až 11 odpovídají jednotlivým řádkům pseudokódu Dinic. Aktuální stav algoritmu je uložen ve vlastnosti `algState` objektu `AlgData`.

Logika při kliknutí na tlačítka „Vpřed“, „Na konec“, „Zpět“ a „Reset“ je obdobná jako u výše popsané implementace Edmondsova-Karpova algoritmu.

3.5.4 Implementace Goldbergova algoritmu

Implementace Goldbergova algoritmu se nachází v souboru `goldberg.js`.

Podobně jako u předchozích dvou algoritmů slouží pro interní reprezentaci stavu, ve kterém se algoritmus nachází, objekt `AlgData`, který je uložen v globální proměnné `algData`. Stejně jako v případě implementace Edmondsova-Karpova algoritmu je v objektu `AlgData` uložena matice kapacit, navíc objekt obsahuje pole `height`, ve kterém jsou uloženy výšky vrcholů, a dále pole `excess`, ve kterém jsou uloženy bilance vrcholů. Velikost maximálního toku v dané síti je uložena ve vlastnosti `maxFlow` objektu `AlgData` a je vypočítána v posledním kroku algoritmu jako bilance stoku.

Goldbergův algoritmus je v této implementaci rozdělen do následujících stavů:

1. `init` – počáteční fáze algoritmu – vybírání zdroje a stoku,
2. `initSourceHeight` – inicializace výšky zdroje,
3. `initOtherVerticesHeight` – inicializace výšek ostatních vrcholů kromě zdroje,
4. `initFlow` – inicializace nulového toku v síti,
5. `initFlowPush` – inicializace průtoku hran vedoucích ze zdroje,
6. `findHighestVertex` – kontrola, zda existuje vrchol s kladnou bilancí (pokud jich existuje více, je vybrán ten nejvyšší),
7. `canPush` – kontrola, zda existuje hrana, po které by šel protlačit tok z nalezeného vrcholu,
8. `push` – operace protlačení toku,
9. `relabel` – operace zvýšení vrcholu,
10. `finished` – fáze po dokončení běhu algoritmu.

Stavy 2 až 10 odpovídají jednotlivým řádkům pseudokódu Goldberg. Aktuální stav algoritmu je uložen ve vlastnosti `algState` objektu `AlgData`.

Logika při kliknutí na tlačítka „Vpřed“, „Na konec“, „Zpět“ a „Reset“ je obdobná jako u výše popsané implementace Edmondsova-Karpova algoritmu.

3.6 Uživatelská příručka

Tato část slouží jako uživatelská příručka k programu na podporu výuky toků v sítích. Je popsáno uživatelské rozhraní aplikace a možnosti, které aplikace uživateli nabízí, včetně základního popisu jazyka DOT, který lze využít pro definici sítě.

3.6.1 Předpoklady pro spuštění

Program na podporu toků v sítích je webová aplikace, která pro provoz vyžaduje připojení k internetu a internetový prohlížeč. Doporučené prohlížeče, které byly testovány během vývoje, jsou *Chrome* od společnosti Google, *Edge* od společnosti Microsoft a *Firefox* od společnosti Mozilla. Aplikaci lze spustit na desktopových i mobilních zařízeních, pro optimální uživatelský zážitek je doporučen desktop.

Dále je předpokládána základní znalost uživatele v oblasti teorie grafů a algoritmizace.

3.6.2 Uživatelské rozhraní sekce pro simulaci algoritmu

Struktura webových stránek je popsána v sekci 3.4. Jádro aplikace tvoří webové stránky umožňující sledování průběhu algoritmů (Edmondsův-Karpův, Dinicův a Goldbergův). Každá ze stránek věnovaná jednomu z algoritmů obsahuje sekci s nadpisem „Vyzkoušet algoritmus“, kde je uživateli umožněno sledovat průběh vybraného algoritmu na jím vybrané či sestavené síti. Uživatelské rozhraní této sekce je zachyceno na obrázku 18, kde jsou červenými číslicemi označeny následující části uživatelského rozhraní:

1. Ovládací panel – část rozhraní obsahující tlačítka „Start“, „Reset“, „Zpět“, „Vpřed“ a „Na konec“, které slouží ke spuštění či resetování algoritmu a přechodu na předchozí krok, následující krok či úplně na konec algoritmu.

2. Záložky „Popis“, „Pseudokód“ a „Stav proměnných“. Záložka „Popis“ obsahuje textový popis právě vykonávaného kroku algoritmu. Po rozkliknutí záložky „Pseudokód“ je uživateli prezentován pseudokód daného algoritmu, kde je modře vyznačen řádek pseudokódu, který je právě vykonáván. V záložce „Stav proměnných“ se nachází tabulka s vybranými proměnnými a jejich hodnotami.

3. Oblast s tlačítky sloužícími k úpravě sítě. Pokud není kliknutím vybrán žádný vrchol ani hrana, jsou zobrazena tlačítka „Přidat vrchol“ a „Přidat hranu“. Při kliknutí na některý vrchol se navíc zobrazí tlačítka „Upravit vrchol“ a „Smazat“, sloužící k upravení či smazání vybraného vrcholu. Obdobně se při kliknutí na některou hranu zobrazí navíc tlačítka „Upravit hranu“ a „Smazat“, sloužící k upravení či smazání vybrané hrany.

   Po spuštění algoritmu tlačítkem „Start“ není možná editace sítě a tlačítka v oblasti 4 jsou tak skryta až do stisknutí tlačítka „Reset“.

4. Oblast, ve které je zobrazena síť. V horní části navíc obsahuje tlačítko „Smazat síť“, po jehož stisknutí dojde ke smazání všech hran a vrcholů sítě. V levé dolní části se nachází směrová tlačítka sloužící k posunu sítě v rámci oblasti. V pravé dolní části se nachází tlačítka pro přiblížení či oddálení sítě a pro vycentrování sítě na střed oblasti.

V horní části uživatelského rozhraní sekce pro simulaci algoritmu se navíc nachází dropdown pro výběr předpřipravené sítě a sekce pro import a export sítě. Rozložení uživatelského rozhraní oblastí 1 a 2 bylo inspirováno aplikací [1].

Na obrázku 19 je zobrazeno uživatelské rozhraní sekce „Import a export sítě“ po kliknutí na rozbalovací komponentu „Import a export sítě“. Zde má uživatel na výběr dvě možnosti. V horní části sekce se nachází možnost nahrát síť ze souboru či uložit síť do souboru, s příslušnými tlačítky „Nahrát“ a „Uložit“. Níže má uživatel možnost zadat síť ručně. Červenou číslicí 1 je označeno textové pole, do kterého uživatel může zadat síť ve formátu DOT. Formát DOT je blíže popsán v následující sekci. Po zadání sítě do textového pole ji lze do aplikace importovat pomocí tlačítka „Import“. Stiskne-li uživatel tlačítko „Export“, do textového pole bude exportována aktuální síť ve formátu DOT. Stisknutím tlačítka „Vložit do schránky“ je uživateli následně umožněno vložit síť v DOT formátu z textového pole do schránky.

3.6.3 Formát DOT

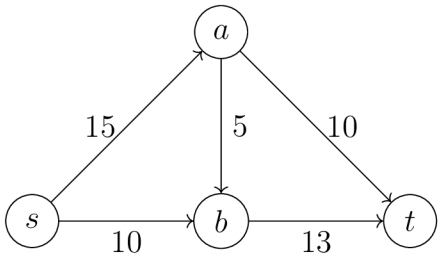
DOT [19] je jazyk určený pro popis grafů. Lze využít například v open-source nástroji Graphviz, který umožňuje vizualizovat různé typy grafů. V programu na podporu výuky toků v sítích je podporována určitá podmnožina jazyka DOT, která je demonstrována na následujícím příkladě.

Příklad 23 (Formát DOT)

Na obrázku 17 je zobrazena jednoduchá síť se čtyřmi vrcholy. Zápis této sítě v jazyce DOT je následující.

```
digraph {
    s;
    a;
    b;
    t;
    s -> a [label="15"];
    a -> b [label="5"];
    a -> t [label="10"];
    s -> b [label="10"];
    b -> t [label="13"];
}
```

Zápis začíná vždy klíčovým slovem `digraph`. Poté následují složené závorky, ve kterých jsou definovány vrcholy a hrany sítě. Vrchol v se souřadnicemi $[a, b]$ lze definovat jako `v [x=a, y=b];`. Zápis souřadnic u vrcholu je nepovinný. Pokud uživatel nechce zadávat souřadnice, je možné úplně vynechat definice vrcholů. Síť je pak určena zápisem hran. Hranu v_0v_1 s kapacitou n lze definovat jako `v0 -> v1 [label="n"];`.



Obrázek 17: Ukázková síť se čtyřmi vrcholy.

3.6.4 Omezení v programu

Pro optimální uživatelský zážitek a bezproblémový běh programu byly zavedeny následující omezení:

- názvy vrcholů se nesmí opakovat, mohou obsahovat pouze písmena a číslice a délkou jsou omezeny na 10 symbolů,
- síť může obsahovat maximálně 20 vrcholů,
- hranám je možné nastavit pouze celočíselné kapacity.

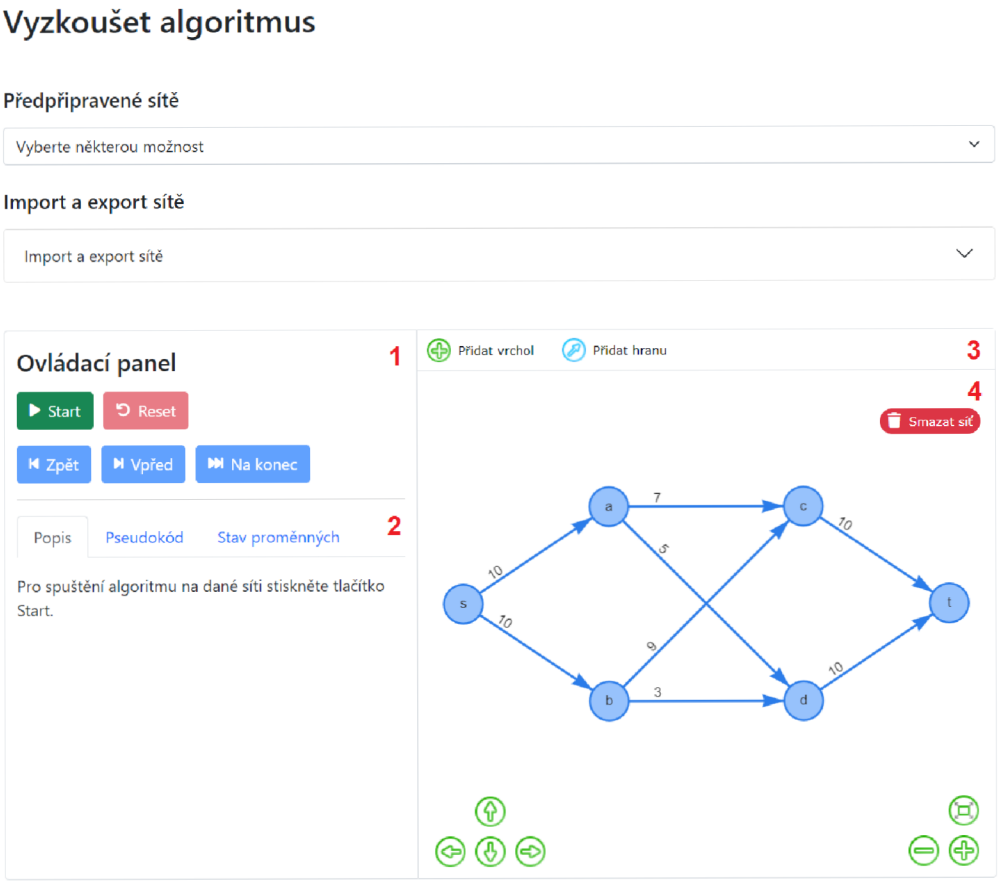
Omezení na maximálně 20 vrcholů v síti bylo zavedeno s ohledem na výukový účel programu, kdy vyšší počet vrcholů v síti není vhodný – klesá přehlednost a schopnost uživatele sledovat průběh algoritmu, zároveň rostou požadavky na výkon.

3.6.5 Klávesové zkratky

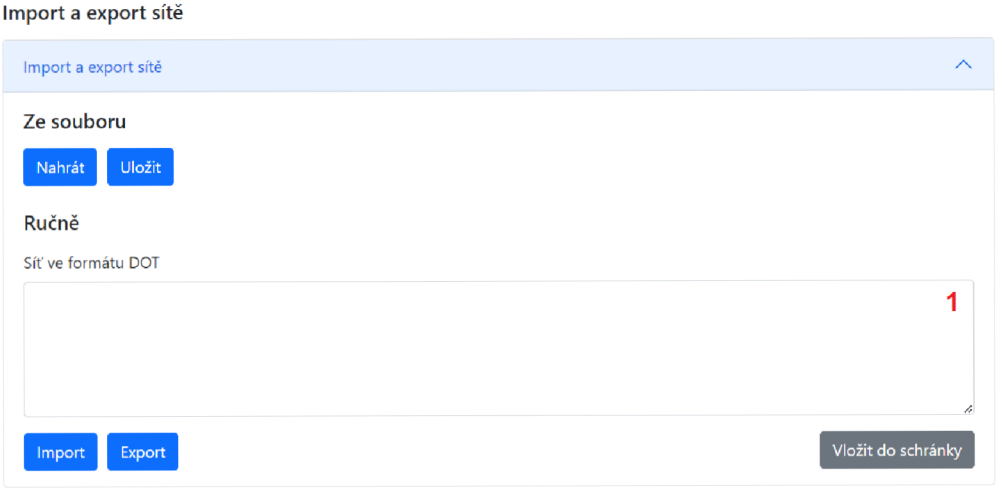
Pro pohodlnější práci s programem jsou uživateli k dispozici následující klávesové zkratky:

- dialog při přidávání a editaci vrcholu nebo hrany lze potvrdit stiskem klávesy „Enter“ a zrušit stiskem klávesy „Esc“,
- po kliknutí na vrchol či hranu lze vybraný objekt smazat kombinací kláves „Alt + Delete“,

- „Alt + S“ – spuštění algoritmu,
- „Alt + R“ – reset algoritmu,
- „Alt + B“ – krok zpět,
- „Alt + N“ – krok vpřed,
- „Alt + M“ – přejít na konec algoritmu.



Obrázek 18: Uživatelské rozhraní sekce pro simulaci algoritmu.



Obrázek 19: Uživatelské rozhraní sekce pro import a export sítě.

Závěr

Byl vytvořen program na podporu výuky toků v sítích v podobě webové aplikace, která uživateli nabízí možnost krok po kroku simulovat průběh *Edmondsova-Karpova*, *Dinicova* a *Goldbergova* algoritmu na zadané síti. Pro simulaci algoritmu je možné využít některou z předpřipravených sítí, síť v programu ručně vytvořit, nebo ji importovat ve formátu *DOT*.

Kapitola 2 slouží jako podpůrný výukový text pro toky v sítích a zmíněné algoritmy pro problém maximálního toku. Princip každého algoritmu je vysvětlen a poté demonstrován na jednoduchém příkladu. Požadavky na program, vývoj programu a použité technologie jsou stručně shrnuty v kapitole 3. V sekci 3.6 je dále zdokumentována uživatelská příručka, poskytující návod k používání programu.

Jako možné rozšíření této práce se nabízí implementace dalších algoritmů pro problém maximálního toku nebo dalších algoritmů pro problémy souvisejícími s toky v sítích, jako je například problém maximálního toku s minimální cenou.

Conclusions

An educational support application for flows in networks was created in the form of a web application that offers user the possibility to simulate step by step the *Edmonds-Karp*, *Dinic* and *Goldberg* algorithms on a given network. To simulate the algorithm, it is possible to use one of the predefined networks, create the network manually in the program, or import it in *DOT* format.

Chapter 2 serves as a study material for flows in networks and the mentioned algorithms for the maximum flow problem. The principle of each algorithm is explained and then demonstrated through a simple example. The program requirements, program development and the technologies used are briefly summarized in chapter 3. In the section 3.6 the user manual is also documented, providing instructions on using the program.

As a possible extension of this thesis, there is the possibility of implementing additional algorithms for the maximum flow problem or additional algorithms for problems related with flows in networks, such as the minimum cost flow problem.

A Obsah elektronických dat

bin/

Kompletní adresářová struktura webové aplikace Program na podporu výuky toků v sítích (v ZIP archivu) pro zkopírování na webový server. Adresář obsahuje i všechny runtime knihovny a další soubory potřebné pro bezproblémový provoz webové aplikace na webovém serveru.

doc/

Text práce ve formátu PDF, vytvořený s použitím závazného stylu KI PřF UP v Olomouci pro závěrečné práce, včetně všech příloh, a všechny soubory potřebné pro bezproblémové vygenerování PDF dokumentu textu (v ZIP archivu), tj. zdrojový text textu, vložené obrázky, apod.

src/

Kompletní zdrojové texty webové aplikace Program na podporu výuky toků v sítích se všemi potřebnými zdrojovými texty, knihovnami a dalšími soubory potřebnými pro bezproblémové vytvoření adresářové struktury pro zkopírování na webový server.

readme.txt

Instrukce pro nasazení webové aplikace Program na podporu výuky toků v sítích na webový server, včetně všech požadavků pro její bezproblémový provoz, a webová adresa, na které je aplikace nasazena pro účel testování při tvorbě posudků práce a pro účel obhajoby práce.

Literatura

[1] Webová aplikace – *Visualizations of Graph Algorithms*. Department of Mathematics, Technical University of Munich. Dostupné z: https://algorithms.discrete.ma.tum.de/flow.

[2] *Ztráty vody loni dosáhly v Praze opět historického minima*. Společnost Pražské vodovody a kanalizace, 2021. Dostupné z: https://pvk.cz/aktuality/ztraty-vody-loni-dosahly-v-praze-opet-historickeho-minima.

[3] MAREŠ, M. *Krajinou grafových algoritmů: průvodce pro středně pokročilé*. Praha: ITI, 2007. ISBN 978-80-239-9049-2. Dostupné z: http://mj.ucw.cz/vyuka/ga.

[4] MAREŠ, M. a VALLA, T. *Průvodce labyrintem algoritmů*. Praha: CZ.NIC, 2022. ISBN 978-80-88168-66-9. Dostupné z: https://knihy.nic.cz/files/edice/pruvodce_labyrintem_algoritmu_v2.pdf.

[5] ČERNÝ, J. *Základní grafové algoritmy*. Praha: MFF UK, 2010. Dostupné z: https://kam.mff.cuni.cz/~kuba/ka/ka.pdf.

[6] WILLIAMSON, D. P. *Network flow algorithms*. New York, NY, USA: Cambridge University Press, 2019. ISBN 9781316636831. Dostupné z: http://www.networkflowalgs.com.

[7] VEČERKA, A. *Grafy a grafové algoritmy*. Univerzita Palackého v Olomouci, 2007. Dostupné z: https://phoenix.inf.upol.cz/esf/ucebni/Grafy_a_grafove_algoritmy.pdf.

[8] FORD, L. R. a FULKERSON, D. R. *Maximal Flow Through a Network*. Canadian Journal of Mathematics. 1956, 8, 399–404. ISSN 0008-414X. Dostupné z: https://doi.org/10.4153/CJM-1956-045-5.

[9] ZWICK, U. *The smallest networks on which the Ford-Fulkerson maximum flow procedure may fail to terminate*. Theoretical Computer Science. 1995, 148(1), 165–170. ISSN 03043975. Dostupné z: https://doi.org/10.1016/0304-3975(95)00022-O.

[10] EDMONDS, J. a KARP, R. M. *Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems*. Journal of the ACM. 1972, 19(2), 248–264. ISSN 0004-5411. Dostupné z: https://doi.org/10.1145/321694.321699.

[11] DINITZ, Y. *Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation*. Doklady Akademii Nauk SSSR. 1970, 11, 1277–1280. Dostupné z: https://www.researchgate.net/publication/228057696_Algorithm_for_Solution_of_a_Problem_of_Maximum_Flow_in_Networks_with_Power_Estimation.

[12] MALHOTRA, V. M. a KUMAR, M. P. a MAHESHWARI, S. N. *An $O(|V|^3)$ algorithm for finding maximum flows in networks.* Information Processing Letters. 1978, 7(6), 277–278. ISSN 00200190. Dostupné z: https://doi.org/10.1016/0020-0190(78)90016-9.

[13] GOLDBERG, A. V. a TARJAN, R. E. *A new approach to the maximum flow problem.* Proceedings of the eighteenth annual ACM symposium on Theory of computing – STOC '86. New York, USA: ACM Press, 1986, 136–146. ISBN 0897911938. Dostupné z: https://doi.org/10.1145/12130.12144.

[14] Knihovna – *Bootstrap*. Dostupné z: https://getbootstrap.com.

[15] Knihovna – *Font Awesome*. Dostupné z: https://fontawesome.com.

[16] Knihovna – *MathJax*. Dostupné z: https://www.mathjax.org.

[17] Knihovna – *jQuery*. Dostupné z: https://jquery.com.

[18] Knihovna – *vis.js*. Dostupné z: https://visjs.org.

[19] Dokumentace jazyka *DOT* na webu nástroje Graphviz. Dostupné z: https://graphviz.org/documentation.