Semestrální práce z KIV/PC

Hledání maximálního toku v grafu silniční sítě

Pavel Shevnin
A21B0269P
shevnin@students.zcu.cz

27. 12. 2021

Obsah

1 Zadání

Naprogramujte v jazyce ANSI C přenositelnou konzolovou aplikaci, jejímž účelem bude nalezení maximálního toku v síti $\overrightarrow{G}$ s jedním zdrojem z a jedním stokem s. Vaše vstupní i výstupní soubory typu `csv` je možné vizualizovat pomocí aplikace *QGIS*, která je dostupná na adrese

`https://qgis.org/en/site/forusers/download.html`

Program se bude spouštět příkazem `flow.exe` s následujícími přepínači:

`-v <uzly.csv>` Parametr specifikuje vstupní soubor typu `csv`, který obsahuje informace o uzlech sítě. Při zadání nevalidní tabulky program vypíše chybovou hlášku **`„Invalid vertex file.\n"`** a skončí s návratovou hodnotou `1`. Zaručte, aby načtení uzlů grafu bylo vždy první operací.

`-e <hrany.csv>` Parametr určuje vstupní soubor typu `csv`, jehož řádky popisují hrany sítě. V případě zadání nevalidní tabulky program vypíše chybovou hlášku **`„Invalid edge file.\n"`** a skončí s návratovou hodnotou `2`.

`-s <source_id>` Parametr jednoznačně určuje zdroj v síti díky primárnímu klíči `id` tabulky *`<uzly.csv>`*. Pokud zadaný uzel v tabulce *`<uzly.csv>`* neexistuje, program vypíše chybovou hlášku **`„Invalid source vertex.\n"`** a skončí s návratovou hodnotou `3`.

`-t <target_id>` Parametr na základě primárního klíče `id` v tabulce *`<uzly.csv>`* jednoznačně určuje stok v síti. Pokud uvedený uzel v tabulce *`<uzly.csv>`* neexistuje, nebo je shodný se zdrojem *`<source_id>`*, program vypíše chybovou hlášku **`„Invalid sink vertex.\n"`** a skončí s návratovou hodnotou `4`.

`-out <out.csv>` V případě úspěšného nalezení toku nenulové velikosti jsou do `csv` souboru daného tímto *nepovinným* parametrem uloženy hrany minimálního řezu, jež odděluje zadaný zdroj a stok. Výstupní tabulka má stejnou strukturu jako vstupní (*`<hrany.csv>`*). Hrany jsou v této tabulce seřazeny dle jejich identifikátoru `id` vzestupně. Pokud je zadané umístění neplatné, program vypíše chybovou hlášku **`„Invalid output file.\n"`** a skončí

s návratovou hodnotou `5`. Pokud soubor již existuje, program jej přepíše.

`-a` — Jedná se o *nepovinný* přepínač, při jehož uvedení bude program pracovat i s hranami, které mají příznak `isvalid` nastavený na hodnotu `False`. Při spuštění bez parametru `-a` tedy program uvažuje pouze hrany s příznakem `isvalid` s hodnotou `True`.

Není-li některý z povinných parametrů uveden, aplikace končí příslušnou chybovou hláškou a návratovou hodnotou. Po úspěšném dokončení algoritmu hledání maximálního toku program vypíše hlášku **„Max network flow is |x| = <s>\n."**, kde **<s>** je velikost nalezeného toku, tj. propustnost minimálního řezu. V případě, že v síti neexistuje tok nenulové velikosti, program skončí s návratovou hodnotou `6`. Platí-li opačné tvrzení a zároveň uživatel použil parametr `-out`, pak je vytvořen výstupní soubor ***<out.csv>***, do kterého jsou zapsány hrany odpovídajícího minimálního řezu. Program následně skončí s návratovou hodnotou `EXIT_SUCCESS`.
Váš program může být během testování spuštěn například takto:

```
flow.exe -e edgs.csv -v "vertcs file.csv"-a -s 43 -t 81
```

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. Postupujte podle instrukcí uvedených na webu předmětu. Archiv nechť obsahuje všechny zdrojové soubory potřebné k přeložení programu, **makefile** pro Windows i Linux (pro překlad v Linuxu připravte soubor pojmenovaný `makefile` a pro Windows `makefile.win`) a dokumentaci ve formátu PDF vytvořenou v typografickém systému TEX(LATEX). Bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.
Další informace najděte na odkazu:

`https://www.kiv.zcu.cz/studies/predmety/pc/data/works/sw2021-03.pdf`

2 Analýza úlohy

Podle zadání lze pochopit, že hlavním úkolem semestrální práce je nalezení maximálního toku v síti s jedním zdrojem a jedním stokem. Síť bude reprezentovaná ohodnoceným orientovaným grafem. Graf to je abstraktní způsob reprezentace vztahů, např. silnice spojující města a jine druhy sítě. Grafy jsou vytvořeny ze vrcholů a hran. Vrchol to je bod, hrana spojuje dva body grafu. Podle úkolu máme orientovaný graf, to znamená, že každa hrana má jenom jeden směr. Naš graf je ohodnoceny, což znamená že každa hrana má propustnost. S propustnosti hran budeme pracovat při výpočtu maximálního toku naše sítě. Prvním problémem realizace programu semestrální práce byl výběr vhodné struktury pro uložení grafu. Existuje spousta datových struktur které využívají pro práci s grafy.

2.1 Datové strukturý pro graf

2.1.1 Matícové struktury

Jedním ze způsobů chránění grafu je matice sousednosti. To je matice kde hlavičky řádků a sloupců reprezentují vrcholy grafu, vyznam každého prvku matíce reprezentuje propustnost hrany mezí uzly hlavičky. Například, přistup k propustností hrany která jde ze vrcholu `i` do vrcholu `j`, mužeme dostat když zavoláme `i`-tý řadek `j`-tého sloupce matice Viz obrázek 2.1.

Nodes		1	2	3	4
	1	0	1	0	1
	2	0	0	1	1
	3	0	1	0	0
	4	1	0	1	0

Obrázek 2.1: Matice sousednosti

Jiným prostředkem, jak popsat graf matici je takzvaná matice incidence uzlů a hran. Když graf má n prvků a m hran, matici incidence bude obdélníková matice typu `(m, n)`. Řádky budou odpovídat vrcholům, sloupce - hranám grafu. Podle úlohy máme neorientovaný graf, proto pokud mezí dva vrcholy máme hranu a, v `i`-tem sloupci a j-tem řádku budeme mít 1 pokud `i`-tý vrchol je počátečním vrcholem hrany propustnosti `1`. Pokud tento vtchol je koncovým bodem, čísla budou záporné. Viz obrázek 2.2. Maticové typy chránění grafů mají výhodou, jsou docela rychle. Například, aby najít propustnost hrany z uzlu `i` do uzlu `j` v matici sousednosti, musíme zavolat `a(i,j)` ihned budeme mít propustnost hrany, případně `0`, když hrana neexistuje. Ale po analýze těchto datových typů a souborů s datama pochopil jsem, že maticové typy nejsou vhodné pro chránění grafu v naše úloze.

Edges

Nodes	a	b	c	d	e
1	1	1	0	0	0
2	1	0	1	1	0
3	0	0	0	1	1
4	0	1	1	0	1

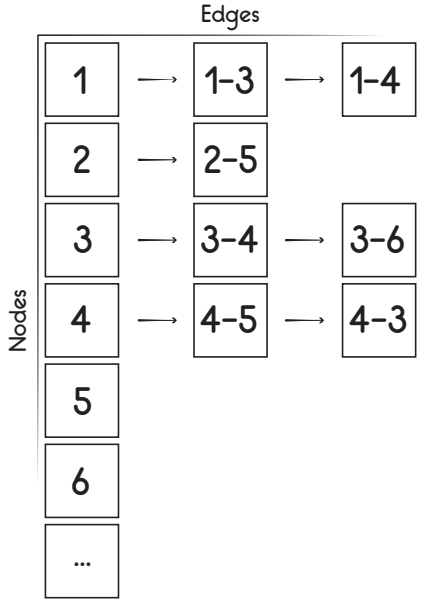
Obrázek 2.2: Matice incidence

Maticové typy jsou vhodné pro velmi husté grafy což náš graf není. Provedl jsem výpočet a pochopil, že matice sousednosti potřebuje $m \cdot m$ bytů, (kde m je počet vrcholů). místa v paměti počítače. Například matice pro soubor s testovací data *pilsen nodes.csv* potřeboval bych skoro `85` megabytů paměti počítače. Ale nebyl bych to tak velký problém, kdyby užitečná data nebyly méně než `45` kilobytů z těchto dát, to znamená, že kdyby využili jsme matice sousednosti, měli bychom velkou matici zaplněnou nulami, což není efektivní způsob chránění grafu.

2.1.2 Seznam sousedností

Dalším způsobem chránění grafů je seznam sousedností. Obrázek č `3`. Tento datový typ je reprezentován seznamem, který obsahuje všechny vrcholy grafu. A každý vrchol má svůj seznam. V seznamech vrcholu se nachází hrany, které se začínají v tomto vrcholu. Tento datový typ využívá paměť mno-

hem efektivněji, bude mít v pamětí jen `n + m` prvků, kde n je počet vrcholů a `m` je počet hran. Není tento způsob nejrychlejším pro hledání prvků a hran. Aby najít hranu, která jde z uzlu `x` do uzlu `y`, potřebujeme projít všechny hrany uzlu x, a skontrolovat jejích cílový uzel, ale není to velkým problémem, protože máme není moc hustý graf a každý vrchol bude mít jen několik hran. Jako seznamy pro chránění vrcholů a hran budu využívat vektor to je dynamické realokující pole.



Obrázek 2.3: Seznam sousedností

2.2 Odstranění duplicit

Dalším problémem při napsání úlohy bylo ignorování duplicit při načtení hran a vrcholů z *.csv* souborů. Prvním způsobem který existuje je nejjednodušší procházení všech dosud načteny prvků, a kontrola zda index nového prvků dříve už byl načten, ale tento způsob není vhodný proto, že data které načítání ze souboru mají složitější strukturu seznamů sousedností. To jest pro ověření hran potřebujeme projít celou strukturu.

Ještě jedním způsobem odstranění duplicit je generace pole délkou maximálního indexu nalezeného v souboru. Toto celé pole bude vyplněné nulami, při nalezení a přidání nového prvků ze souboru index pole ktery se rovná indexu načteného prvků, měníme za nulu. Výhodou tohoto způsobu je to, že nepotřebujeme pořád procházet celé pole již načtených indexu, stačí prostě

zavolat prvek načteného indexu, a zkontrolovat zda existuje a v případě když už existuje, ignorovat tento element ze souboru. Ale tento způsob má i nevýhodu, která spočívá v tom, že data která načítání ze souboru *.csv* mohou mít velké mezery, to znamená že v případě výběru daného způsobů budeme využívat paměť počítače pro chránění nul.

Nejvhodnějším pro řešení tohoto problému způsobem zdál se mi jednoduchý, ale přesto paměťové efektivní. Každý index nově načteneho prvků zapisujeme do vektoru. Pak, při načtení nového prvků kontrolujeme zda prvek s takovým indexem jíž byl načten do struktury. Funguje to pro vrcholy `i` pro uzly grafu.

2.3 Algoritmus

Tento algoritmus začíná tak, že všem hranám přiřadí tok hodnoty nula, neboli nic sítí na počátku neprotéká. Dále pak začne testovat nalezení zlepšující cesty, jestliže tato cesta bude nalezena tak se tok zvýší. Naopak jestliže cesta nalezená už nebude, to znamená že zlepšující cesta už neexistuje (viz obecná teorie k algoritmu) je nalezen maximální tok.

Na začátku přiřadí všem uzlům grafu jako stav uzlu hodnotu `FRESH`. To znamená,
že tyto uzly jsou čisté neboli ještě nepoužité. Uzlu `S` se přiřadí `p[s]` směr kterým jde hrana a poté ještě delta s jako nekonečno jelikož je to uzel předaný parametrem a tento uzel se otevře. Poté startuje cyklus který na svém začátku vybere uzel který je otevřený (stav uzlu == otevřeno) a uzavře ho
a poté pro všechny jeho následníky zjistí že jestliže je následník, uzel se stavem `FRESH` a jeho tok je nižší než kapacita hrany jež spojuje uzel s tímto následníkem, tak ho otevře, přiřadí mu směr kterým je orientovaná hrana a vypočte pro něj delta. Delta se vypočítává právě pro to aby bylo možné zjistit o kolik se dá navýšit tok v hranách orientovaných směrem od zdroje k spotřebiči a snížit tok na hraně směřující od spotřebiče ke zdroji (bilance zůstává stejná). Když se takto projedou všechny následovníci tak se začnou procházet všichni předchůdci tohoto uzlu. U nich se zjišťuje zda jsou `FRESH` a zda hrana spojující tyto dva uzly má vyšší tok než `0`. Když je tato podmínka splněna nastaví se předchůdce na stav `OPEN` nastaví se mu orientace hrany a určí delta. A toto celé se opakuje dokud existuje alespoň nějaký otevřený uzel či dokud testovaný uzel u se nebude rovnat spotřebiči. A také na základě této naposledy zmiňované rovnosti bude funkce vracet návratovou hodnotu true nebo false.

3 Implementace programu

3.1 Modul `main`

Na začátku modul provádí „include“ standardních knihovén: `stdlib.h`, `stdio.h` take je vložena knihovna pro práce s čarou: `string.h`. Potom jsou vložené lokální soubory potřebné pro běh programu: `parameters.h`, `loader.h`, algorithm.h a `out_file.h`. Modul `main` obsahuje eponymní funkce to je hlavní řidicí funkce programu. Jako parametry přijima `int argc`, což je počtem parametrů zadaných uživatelem z příkazové řádky a `argv` což je pole ukazatelů na samotné parametry z příkazové řádky. Funkce main volá všichni potřebné funkce pro provedení všech načtení a výpočtů celeho programu, pak kontroluje návratové hodnoty volaných funkcí, a v případě když hodnota kterou vrací funkce liší od nuly, to znamená, že došlo k nějakému selhání, program se zastavuje, a `main` vrací nenulovou hodnotu, která závisí na druhu chyby. První funkce kterou volá `main` se nachází v modulu `parameters`

3.1.1 Modul `parameters`

V tomto modulu jsou vložené standardní knihovný: `stdlib.h`, `stdio.h`, `string.h` a knihovnu `loader.h` Tento modul obsahuje funkce `parameters_processing`, kéra přijímá jako parametry z `main int argc` a `char *argv` a slouží hlavně k rozdělení, zpracování a kontrole vstupních parametrů. Na začátku, `parameters_processing` kontroluje, že počet parametrů je větší než jeden (když parameter je jenom `1` chápeme, že určitě chybí první z povinných parametrů, konkrétně – soubor uzlů sítě). Pak procházíme všichni vstupní parametry cyklem for od `1` do `argc`, a porovnáváme všichni parametry pomocí standardní funkce `strcmp` s předem definovanými označení. Pokud označení se rovná nějakému parametru, bereme `id + 1` a takovým způsobem získáváme potřebný parameter. Pak v případě když parameter musí být ve formátu `.csv`, kontrolujeme spravnost tohoto formátu pomocí funkce `format_check`, která kontroluje aby parameter měl na konci `.csv`, označení a v případě když tak to není, vrací `1`. Když soubor má požadovaný formát, vkládáme ho do proměnné které pro všichni vstupní parametry jsou `extern` a nachází v souboru `parameters.h`. Když název má špatný format, proměnnou necháváme v původním stavu to jest s řadkem „undefined“. Ale pro výstupní soubor to funguje jinak. Pokud soubor nebyl nalezen, necháváme proměnnou s názvem „undefined“, ale pokud byl nalezen a má špatný formát, vkládáme do proměnné prázdný řetězec. Když všichni

parametry mají správný formát, po ukončení této funkce, budeme mít externí proměnné s názvy souborů.

3.1.2 Struktury

V programu mám několik struktur `vector`, `queue`, `edge` a `node`. Vector to je standardní implementace vektoru, ve struktuře jsou uložené `count` – počet prvků vektoru, `capacity` – velíkost vektoru, `item_size` – Velikost jednoho prvku vektoru a `deallocator`, to je ukazatel na funkci, která umí uvolnit dynamicky alokované prvky obsažené ve vektoru. Queue to je standartní implementace fronty. Struktura node reprezentuje vrchol grafu. Node obsahuje id, což je indexem vrcholu načtení z .csv souboru; Color, který potřebujeme pro činnost algoritmu nalezení maximalního toku; Před který je indexem předchozího vrcholu, při nalezení cesty; `wkt` to je řetězec souřadnic načteny z `.csv` souboru; `edges` – to je vektor obsahující seznam hran, které začínají tímto vrcholem. Struktura `edge` to je reprezentace hran grafu. Obsahuje: id; source – je indexem počátečního vrcholu hrany; target – je indexem koncového vrcholu hrany; capacity to je propustnost hrany; `is_valid` to je validnost hrany; `flow` – je tokem, který aktuálně hranou běží, potřebujeme pro výpočet maximálního toku sítě; `*wkt` to je řetězec souřadnic načteny z `.csv` souboru.

3.1.3 Modul `loader`

Další funkce, kterou volame v mainu je `node_loader`, kéra se nachází v modulu `loader`. Jako parameter přijímá `char *path`, to je cesta do `.csv` souboru s úhly grafu. V případě úspěchu, výsledkem činnosti teto funkci bude vector zaplněny vrcholy, které jsou načtené ze souboru. Za běhu programu bude pomocí standardní funkce strtok rozdělovat vstupní řetězec do správných proměnných take bude zapisovat již načtené indexy do vektoru indexes, pak, pomocí funkce `is_dublikate` bude kontrolovat jedinečnost každého nově načteného vrcholu. V případě chyb, funkce uzavří otevřený soubor, uvolní paměť vektorů a skonči s navratovou hodnotou `1`. Příští volána v mainu funkce je `edge_loader` take se nachází v modulu `loader`. Je velmi podobná funkce node_loader s jedním rozdílem, vkládám načtené hrany do různých vektorů, které nachází ve vrcholech indexy kterých jsou stejné s hodnotou source nově načtené hrany. Funkce edge_to_node prochází cyklem `for` všichni vrcholy, a ukladá hranu do spravného indexu. Duplicity odstraňují stejně jako při načtení vrcholů. V případě selhání funkce uvolňuje vektory,
a vrací hodnotu `2` Dále v mainu volají podobné funkce `source_in_graph`

a `target_in_graph`. Tyto funkce slouží ke kontrole vstupních hodnot zdroju a stoku. Prochází graf cyklem `for`, a kontroluje zda existují indexy zadaných uživatelem zdroju a stoku. Pokud zdroj neexistuje funkce `_in_graph`, končí s návratovou hodnotou `3`. Pokud neexistuje stok, funkce `target_in_graph` končí s návratovou hodnotou `4`.

3.1.4 Modul `algorithm`

Potom ve funkci `main` voláme funkce `ford_fulkerson`, z modulu algorithm, která jako parametry přijímá `source_id` což je indexem zdroju s `target_id` což je indexem stoku. Na začátku funkce kontroluje že indexy zdroju a stoku najsou stejná čisla, v případě když jsou stejná, vrací nulu, protože nenulový tok určitě nemůže existovat. Pak generuje vektor `out_edges`, do kterého budeme zapisovat hrány minimálního řezu pro výpis do výstupního `.csv` souboru programů. Ford–Fulkersonův algoritmus je implementován tak, že v cyklu while volá funkce `bfs`. Funkce bfs přijímá jako parametry start a end, a hledá novou cestu od start do end. Na začátku `bfs` prochází všichni node,
a barví je do bíle barvy, což znamená že jsou volná. Pak začne přidávat prvky do fronty, když prvek vkladá do fronty, přebarví do šedé barvy. Dale v cyklu while jde po frotě a odstraňuje první prvek, přebarví ho do černé barvy a zároveň v cyklu for prochází všechny hrany aktuálního uzlu hledá příští prvek bíle barvy, který ještě má volny tok `(capacity - flow) > 0`, přidává ho také do fronty. To opakuje pokud existují bíle prvky s možností zvětšit tok. Na konci kontroluje jestli poslední prvek byl přebarven do černé barvy, to znamená ž cesta byla nalezena, algoritmus vrací `1`. V opačném případě vrací `0`. Dale funkce `ford_fulkerson` prochází nově nalezenou cestou od posledního prvků do prvního, pomocí proměnné pred, ze struktury node. A spočítá increment, pak ještě jednou prochází tu cestu a zvyšuje flow všech hran o increment. Pak kontroluje, aby increment nebyl nulou, a nedochazelo k zacyklení. Na konci cyklu `max_flow_graph` zvyšuje o increment nalezené cesty. Do vektoru `out_edges` vkládáme nalezenou hranu minimálního řezu. To všechno opakuje pokud bfs vrací `1` to jest muže najít novou cestu.

3.1.5 Modul `out_file`

Když algoritmu podařilo najít nenulovou cestu, voláme funkce `out_write` z modulu `out_file`, která v případě když uživatel přidal k argumentum název výstupního `.csv` souboru ve správném formátu, vygeneruje tento soubor a zapíše tam hrany minimálního řezu. Pokud najít nenulový maximální tok nepodařilo, program končí s vypísem: `Max network flow is |x| = 0`

a návratovou hodnotou `6`. Pokud funkce main doběhne dokonce, zavolá definovanou předem část: `clean_and_exit_main`, která uvolní vektor `out_edges`, pak na konci uvolní vektor node, a vrátí `0`.

4 Uživatelská příručka

4.1 Překlad programu

Program pro nalezení maximalního toku je konzolovou aplikací. Před spuštěním musí byt přeložen. Pro překlad na operačním systému Windows existuje soubor `Makefile.win`. Pomocí přikazu **`make -f Makefile.win`** v příkazové řadce ze složky s programem.

Na operačním systému `Linux` program lze přeložit přikazem make z příkazové řadky. Přeložení provedé soubor `Makefile`

4.2 Spuštění programu

Pozor na to, že přikaz musí obsahovat všichní povinne parametry: soubor hran, soubor uzlu, index zdroje a stoku. Také jsou nepovinné parametry: parameter validnosti: texttt-a, a parameter obsahující nazev vystupního souboru do kterého budou uložene hrany minímalního řezu v případě úspěšného nalezení maximalního toku. Program lze spustít nasledujicím příkazém:

```
flow.exe -e pilsen_edges.csv -v pilsen_nodes.csv -s 1 -t 83
```

4.3 Běh programu

```
C:\Users\pshev\git\PC_SP>flow.exe -e pilsen_edges.csv
 -v pilsen_nodes.csv -s 1 -t 3000 -out out.csv
Max network flow is |x| = 2300.
```

Obrázek 4.1: Vystup programu při úspěšném výpočtu maximalního toku sítě

```
C:\Users\pshev\git\PC_SP>flow.exe -e pilsen_edges.csv
 -v pilsen_nodes.csv -s 1 -t 1 -out out.csv
Max network flow is |x| = 0.
```

Obrázek 4.2: Vystup programu při neexistujicím toku nenulové velíkosti

```
C:\Users\pshev\git\PC_SP>flow.exe -e -v pilsen_nodes.
csv -s 1 -t 3000 -out out.csv
Invalid edge file.
```

Obrázek 4.3: Vystup programu při chybějicím souboru s hranama

```
C:\Users\pshev\git\PC_SP>flow.exe -e pilsen_edges.csv
 -s 1 -t 3000 -out out.csv
Invalid vertex file.
```

Obrázek 4.4: Vystup programu při chybějicím souboru s uzly

```
C:\Users\pshev\git\PC_SP>flow.exe -e pilsen_edges.csv
 -v pilsen_nodes.csv -t 1 -out out.csv
Invalid source vertex.
```

Obrázek 4.5: Vystup programu při chybějicím zdroju sítě

```
C:\Users\pshev\git\PC_SP>flow.exe -e pilsen_edges.csv
 -v pilsen_nodes.csv -s 1 -out out.csv
Invalid sink vertex.
```

Obrázek 4.6: Vystup programu při chybějicím stoku sítě

5 Závěr

Během práci nad semestrální práce vytvořil jsem úplně splňující zadání algoritmus. Můj program načita data z .csv souboru a generuje z těchto dat graf pak hleda v tomto grafu maximální tok mezí zdrojem a stokem, indexy kterých píše uživatel do příkazové řádky. Program běží docela rychle, například v testovacích datech pro Plzeň: `pilsen_edges.csv` a `pilsen_nodes.csv` vyhledává maximální tok mezí zdrojem č. `1` a stokem č. `3000` za `3` sekundy. Ale algoritmus má i prostor pro vylepšení. Například: když odstraňují duplicity při načtení dat, procházím vektor do kterého jsou zapsány jíž načtené indexy a nevyužívám rychlejší algoritmy, to mohlo bych ještě zrychlít můj algoritmus. Během práce nad algoritmem narazil jsem na spoustu problémů, většina z ních byla způsobena tím že psal jsem v jazyce `ANSI C` poprvé a po Java ve kterém mám největší zkušenosti jazyk `C` je velmi nepřátelský k uživateli. Ale samozřejmě ma i spoustu velkých výhod.