KIV/PC
Zadání č.3
Hledání Minimální Kostry Grafu

Jiří Veselý

January 8, 2021

Contents

1 Zadání **2**

2 Analýza úlohy **3**
2.1 Rozbor . . . 3
2.1.1 Ohodnocený neorientovaný graf . . . 3
2.1.2 Reprezentace datové struktury Graf . . . 4

3 Popis implementace **6**
3.1 Přehled . . . 6
3.1.1 edge.h . . . 6
3.1.2 vertex.h . . . 7
3.1.3 graph.h . . . 7
3.1.4 usefc.h . . . 7
3.2 Rozbor funkcí . . . 8
3.2.1 edge . . . 8
3.2.2 vertex . . . 9
3.2.3 graph . . . 9
3.2.4 usefc . . . 10
3.3 main . . . 10

4 Uživatelská příručka **11**
4.1 Spuštění programu . . . 11

5 Závěr **12**

Chapter 1

Zadání

Detailní znění zadání je na stránce
https://www.kiv.zcu.cz/studies/predmety/pc/doc/work/sw2020-03.pdf

Chapter 2

Analýza úlohy

2.1 Rozbor

Úlohu lze rozdělit do tří částí. Prvním krokem je načtení dat ze souborů hran a vrcholů. Druhou částí spuštění kruskalova algoritmu na hledání minimální kostry grafu. Poslední částí je export nalezených hran do daného souboru, podle zadaného parametru -mst nebo -mrn. Úloha vede k řešení pomocídatové struktury ohodnoceného neorientovaného grafu.

2.1.1 Ohodnocený neorientovaný graf

Ohodnocený neorientovaný graf se chápe jako množina vrcholů a hran. Hrany definují spojení mezi jednotlivými vrcholy a cenu tohoto spojení. Na směru hrany nezáleží. Například, pokud je hrana mezi vrcholy 1 a 2, tak je i hrana mezi vrcholy 2 a 1.

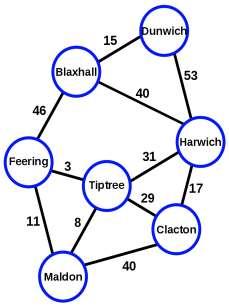

Figure 2.1: Příklad Ohodnoceného Neorientovaného grafu

2.1.2 Reprezentace datové struktury Graf

Pro řešení datové struktury Graf existují dvě nejčastější řešení. První využívá datovou strukturu Spojový seznam a druhá využívá Matici souslednosti.

Matice Sousednosti

Matice sousednosti je matice A definována, jako NxN matice, kde N reprezentuje počet vrcholů. Indexy i a j v reprezentují vrcholy, tudíž pro: $A_{ij} = x$, x může být libovolný znak a reprezentuje hranu z vrcholu i do j. Tento znak u ohodnoceného grafu reprezentuje cenu této hrany.

Implementace tohoto řešení není složitá, neboť spočívá ve vytvoření dvourozměrného pole o rozměrech NxN, kde N reprezentuje počet vrcholů. Zásadním problémeme tohoto řešení je jeho paměťová neefektivita a náročnost. V datech poskytnutých pro tuto úlohu je přibližně 26000 validních vrcholů a 28000 validních hran. To znamená, že by bylo $((26000)^2 - 28000)$ nevyužitých prvků.

$$A = \begin{bmatrix} a_{11} & 0 & a_{13} & \dots & a_{1K} \\ \vdots & \ddots & \dots & \dots & \vdots \\ \vdots & \ddots & \dots & \dots & \vdots \\ a_{K1} & 0 & \dots & a_{KK-1} & 0 \end{bmatrix} \tag{2.1}$$

Výhoda matice sousednosti, je rychlý přístup k prvku se složitostí *O(1)*. Takže toto řešení by bylo vhodné pro malé grafy. Pro data, která byla spokytnutá k této úloze je toto řešení velice nevhodné.

Spojový seznam

Narozdíl od matice sousednosti, *Spojový seznam*, neplýtvá pamětí na nepoužité prvky. Spojový seznam je struktura, která obsahuje atribut, ve kterém je uložen odkaz na další prvek seznamu, tím vzniká spojení (hrana).

Použití spojového seznamu je velmi vhodné na použití poskytnutých dat, protože by spojení vrcholů a hran je řídké. Pokud by matice souslednosti byla plně zaplněná, nemá smysl používat spojový seznam, protože paměťová náročnost by byla stejná, ale vypočetní náročnost větší.

Chapter 3

Popis implementace

Data ze souboru se ukládají do pole struktur hran a vrcholů. Před jejich vložením do polí se filtrují podle pokynů v zadání. Implementace Kruskalova algoritmu se provadí pomocí spojového seznamu. Původně jsem chtěl použít řešení pomocí matice souslednosti, ale kvůli paměťové náročnosti jsem od tohoto řešení upustil.

Implementace je pomocí 4 knihoven a ty jsou *edge*, *vertex*, *graph* a *usefc*.

3.1 Přehled

3.1.1 edge.h

První knihovna je *edge*. Jak už název napovídá, jedná se o strukturu, která obsahuje atributy hrany. [language=C] edge *edge$_{l}oad(constchar filename[], uint*
$datasize); intedge_{c}ompar_{f}n_{b}y_{i}d(constvoid*p1, constvoid*p2); intedge_{c}ompar_{f}n_{b}y_{c}len(constvoid*$
$p1, constvoid*p2); intedge_{c}ompar_{f}n_{b}y_{i}d_{d}own(constvoid*p1, constvoid*p2); intedge_{c}ompar_{f}n_{b}y_{c}le$
$p1, constvoid*p2); voidedge_{p}rint(edge*edge_{d}ata, uintdatasize); voidedge_{e}xport_{m}st(edge*$
$edge_{d}ata, uintdatasize, char*filename); voidedge_{e}xport_{m}rn(edge*edge_{d}ata, uintdatasize, char*$
$filename);$

3.1.2 vertex.h

Další knihovnou je *vertex*. Tato knihovna je velmi podobná knihovně *edge*, ale řeší uložení vrcholů. [language=C] vertex *vertex$_{l}oad(constchar filename[], uint*$ $datasize); intvertex_{c}ompar_{f}n(constvoid*p1, constvoid*p2); voidvertex_{p}rint(vertex*$ $vertex_{d}ata, uintdatasize); intvertex_{g}et_{k}ey_{b}y_{i}d(vertex*vertex_{d}ata, uintdatasize, uintid);$

3.1.3 graph.h

Další knihovnou je *graph*. Tato knihovna je zajišťuje vytvoření grafu a řeší kruskalův algoritmus. [language=C] graph* createGraph(uint vertex$_{d}len, uintedge_{d}len, edge*$ $edges); uintfind(subsetsubsets[], uinti); voidUnion(subsetsubsets[], intx, inty); edge*$ $KruskalMST(graph * graph_{t}emp, vertex * vertex_{d}ata, uint * edge_{m}st_{l}en);$

3.1.4 usefc.h

Další knihovnou je *usefc*. Tato knihovna obashuje užitečné funkce, které se hodili při řešení úlohy. [language=C] void printArrayString(int size, char **values); int inArrayString(int size, char **values, char *search);

3.2 Rozbor funkcí

V této sekci budou vysvětleny dané funkce a jejich účel.

3.2.1 edge

edge_load

Tato funkce načte data ze souboru hran a vytvoří pole struktur hran. Při načítání se řeší filtrování podle pokynů v zadání. Nejprve se ověří hlavička souboru hran. Poté se připravý paměť pro hranu na aktuálním indexu. Dalším krokem je vykopírování atributu WKT z řádky souboru, aby se mohla řádka dále rozdělit podle znaku ','. Pokud se má daná hrana ignorovat, nezvýší se index uvolní se paměť pro atribut WKT. Fuknce vratí pointer na první index v poli hran.

edge_compar_fn_by_id

Tato funkce slouží k porovnání dvou prvků podle id. Používá se pro seřazení hran podle id vzestupně.

edge_compar_fn_by_clen

Tato funkce slouží k porovnání dvou prvků podle clen. Používá se pro seřazení hran podle id sestupně.

edge_compar_fn_by_id_down

Tato funkce slouží k porovnání dvou prvků podle id. Používá se pro seřazení hran podle id vzestupně.

edge_compar_fn_by_clen_down

Tato funkce slouží k porovnání dvou prvků podle clen. Používá se pro seřazení hran podle id sestupně.

edge_print

Vypíše obsah předaného pole hran.

edge_export_mst

Exportuje hrany do souboru podle argumentu -mst.

edge_export_mrn

Exportuje hrany do souboru podle argumentu -mrn.

3.2.2 vertex

vertex_load

Obdobná funkce jako funkce edge_load akorát, že načte vrcholy.

vertex_compar_fn

Porovnávací fuknce vrcholů podle jejich id. Slouží k seřazení vrcholů podle id vzestupně.

vertex_print

Tato fuknce vypíše obsah předaného pole vrcholů.

vertex_get_key_by_id

Funkce vratí na jakém indexu se nachází vrchol v poli vrcholů s předaným atributem id. Pokud není nalezen, vrátí -1.

3.2.3 graph

createGraph

Funkce vytvoří strukturu graph a načte do ní předané hrany.

find

Rekurzivně projde předané pole subsets a vratí root prvek prvku i.

Union

Zajišťuje spojení hran, které jsou navíc. Slouží k imlementaci kruskalova algoritmu.

KruskalMST

Funkce vrátí pole struktur hran. Tyto hrany tvoří minimální kostru grafu.

3.2.4 usefc

printArrayString

Tato funkce vypíše obsah pole stringů.

inArrayString

Vrátí id, kde se nachází hledaný prvek. Pokud pole prvek neobsahuje, vratí -1.

3.3 main

Ještě zbývá funkce main, která je vstupním bodem aplikace. Funkce nejprve ošetří argumenty z příkazové řádky. Poté načte vrcholy, hrany a seředí je podle id vzestupně. Toto seřezení není nutné, ale pro lepší orientaci v datech jsem jej implementoval.Pokud zadané argumenty říkají, že se má řešit minimální kostra, tak se spustí kruskalův algoritmus a exportuje nalezené hrany do požadovaných souborů. Pokud se minimální kostra řešit nemá, program skončí s návratovou hodnotou *EXIT_SUCCESS*. Posledním krokem je uvolnění alokované paměti.

Chapter 4

Uživatelská příručka

Pro přeložení projektu je třeba spustit soubor Makefile. Za předpokladu absence fatálních chyb se tímto projekt přeloží a vytvoří se spustitelný soubor graph.

4.1 Spuštění programu

Program očekává dva povinné parametry následované souborem obsahující data. Prvním je -v, tento argument říká, že dalším argumentem je datový soubor vrcholů. Druhým je -e, tento argument říká, že dalším argumentem je datový soubor hran.
Následují dva nepovinné argumenty. Prvním je -mst. Tento argument říká, že program má ze zadaných dat vytvořit minimální kostru grafu a uložit ji do souboru definovaným v následujícím argumentu. Druhým je -mrn. Tento argument říká, že program má ze zadaných dat vytvořit minimální kostru grafu a uložit ji do souboru definovaným v následujícím argumentu. Formát uložení dat se liší v seřezení a atributech nation, cntryname (viz. zadání).

Chapter 5

Závěr

Myslím, že jsem nalezl optimální řešení danné úlohy. Ovšem musím konstatovat, že program by mohl být více optimalizovaný. Například implementace dynamického alokování paměti pro výsledné hrany minimální kostry nebo volbou mezi použitím spojového seznamu a matice souslednoti na základě vstupních dat.
Práce je nahraná ve službě github.com na adrese https://github.com/teamSPSE/PCsemestralGraph.
Práce je bohužel soukromá, aby nemohlo dojít k případnému plagiátu. Pro zpřístupnění stačí napsat mail s údaji na veselyj@students.zcu.cz