FAKULTA APLIKOVANÝCH VĚD
ZÁPADOČESKÉ UNIVERZITY
V PLZNI

Semestrální práce z KIV/PRO

Eulerův cyklus (Problém čínského listonoše)

Lukáš Runt (A20B0226P)

4. prosince 2021

Obsah

Obsah **1**

1 Zadání **2**

2 Existující metody **2**
2.1 Eulerizace grafu 3
2.2 Fleuryho algoritmus 4
2.3 Hierholzerův algoritmus 4

3 Navržené a zvolené řešení **4**
3.1 Zvolená řešení 4
3.1.1 Algoritmus určení a vytvoření Eulerovského cyklu . . . 4
3.1.2 Fleuryho algoritmus 5
3.1.3 Hierholzerův algoritmus 5
3.2 Navržené řešení 6

4 Experimenty a výsledky **7**

5 Závěr **8**

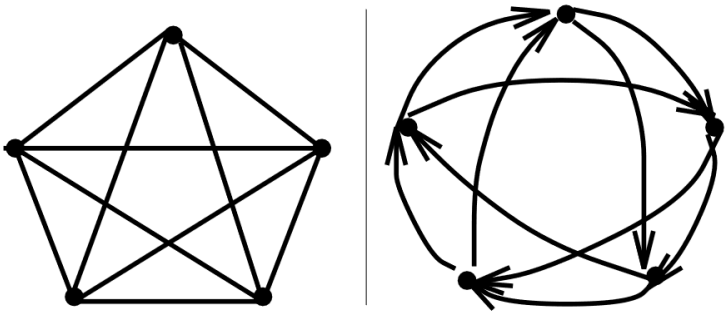
6 Reference **8**

1 Zadání

Mějme mapu města a úkol, kde máme navrhnout trasy pro popelářské vozy, sněžné pluhy nebo poštáky, tak že budou projety všechny ulice města, přičemž je v našem zájmu, aby byly trasy minimální. Zároveň se musí začínat i končit ve stejném místě. Tyto problémy patří do skupiny problémů Eulerovského cyklu. Město si můžeme představit jako graf tak, že křižovatky budou vrcholy a ulice budou hrany, naším úkolem je najít v tomto grafu Eulerovský cyklus.

Vstup: Graf $G = (V, E)$.

Výstup: Nejkratší cyklus, který projde všemi vrcholy grafu G nejméně jednou.



Obr 1: Příklad vstupu a výstupu. Zdroj: [1]

2 Existující metody

Před popisem existujících metod je dobré vysvětlit následující pojmy [2]:

- **Eulerovský tah** je cesta v grafu, která navštíví každou hranu právě jednou.
- **Eulerovský cyklus** je Eulerovský tah, který začíná a končí ve stejném vrcholu.
- **Most** je taková hrana, jejíž odstranění z grafu rozpojí graf na 2 spojité komponenty. Most nikdy nemůže být součástí cyklu.

Existují podmínky pro určení, zda graf obsahuje Eulerův cyklus nebo tah:

- Neorientovaný graf obsahuje Eulerovský cyklus, jestliže je spojitý a každý vrchol má sudý stupeň.
- Neorientovaný graf obsahuje Eulerovský tah, jetliže je spojitý a všechny vrcholy kromě dvou jsou sudého stupně. Tyto dva vrcholy budou počátečním a koncovým vrcholem cesty.
- Orientovaný graf obsahuje Eulerovský cyklus, pokud je silně spojitý a každý vrchol má stejný počet vstupních a výstupních hran.
- Orientovaný graf obsahuje Eulerovský tah z x do y, jestliže všechny ostatní vrcholy mají stejný počet vstupních a výstupních hran, přičemž x a y jsou vrcholy, které mají počet vstupních hran o jednu menší než počet výstupních hran (respektive o jednu vstupní hranu méně u y). Eulerovský tah pak vede z x do y.

Pro nalezení Eulerovského cyklu v grafu musíme mít na vstupu graf, který Eulerovský cyklus obsahuje. Na vstupu by tedy měl být Eulerovský graf. Pokud graf Eulerovský není, musí se z původního graf upravit pomocí zdvojení některých hran (tzv. Eulerizace grafu, viz. 2.1), čímž se docílí, že graf bude Eulerovský.

2.1 Eulerizace grafu

Graf lze Eulerizifikovat přidáním příslušných hran do grafu G a to tak, aby byl graf souvislý (v případě orientovaného grafu silně souvislý) a graf obsahoval pouze vrcholy se sudým stupněm. Nejdříve tedy propojíme souvislé komponenty grafu a následně budeme přidávat nejkratší cesty mezi vrcholy s lichým stupněm, čímž se dané vrcholy stanou vrcholy stupně sudého a tím se graf G přiblíží Eulerovskému grafu. Při řešení musíme zohlednit orientaci grafu, neboť u orientovaného musíme narozdíl od neorientovaného hlídat, aby vytvořená hrana vedla z vrcholu se zápornou δ do vrcholu s kladnou δ. Pro nalezení nejkratší cesty mezi dvěma vrcholy lze použít Dijkstrův nebo Floyd-Warshallův algoritmus.

Pro určení Eulerovského cyklu jsou nejznámější algoritmy Fleuryho a Hierholzerův.

2.2 Fleuryho algoritmus

Fleuryho algoritmus [3, 4, 2] se používá pro nalezení Eulerovského cyklu nebo tahu ve spojitém grafu. Jedná se o celkem neefektivní algoritmus, který se většinou používá pro neorientované grafy. Na vstupu se očekává spojitý graf s žádnými nebo dvěma lichými vrcholy. Základní myšlenka algoritmu je taková, že most by měl být poslední hranou, kterou budeme procházet, neboť kdybychom jsme překročili most, aniž bychom navštívili všechny hrany v první složce, museli bychom se přes most vracet.

Ačkoliv je složitost procházení grafu $O(|E|)$, celý algoritmus zpomaluje složitost detekce mostů pomocí Trajanova algoritmu [5], která je $O(|E|^2)$, nebo alternativním dynamickým Thorupovo algoritmem [6] se složitostí $O(|E| \cdot \log^3 |E| \cdot \log\log |E|)$. Oba algoritmy logicky vedou ke zhoršení celkové složitosti.

2.3 Hierholzerův algoritmus

Hierholzerův algoritmus [4, 7] pro nalezení Eulerovského cyklu nebo tahu v grafu, který očekává na vstupu eulerovský graf. Většinou se používá pro orientované grafy. Základní myšlenkou je postupná konstrukce Eulerova cyklu spojováním disjunktivních kružnic. Jelikož algoritmus každou hranu projde pouze jednou, má algoritmus lineární složitost $O(|E|)$.

3 Navržené a zvolené řešení

3.1 Zvolená řešení

3.1.1 Algoritmus určení a vytvoření Eulerovského cyklu

Jelikož víše zmíněné algoritmy potřebují na vstupu Eulerovský graf, budeme muset nejdříve určit zda graf obsahuje Eulerovský cyklus a následně neodpovídající grafy Eulerizifikovat 2.1. K určení, zda graf obsahuje Eulerovský cyklus, musíme zjistit, zda je graf souvislý, a to pomocí prohledávání do šířky nebo do hloubky. Následně se spočte počet lichých vrcholů. Jestliže mají všechny vrcholy sudý stupeň, graf obsahuje Eulerův cyklus a tím pádem máme i minimální délku, protože každá hrana je navštívena právě jednou. Avšak tato možnost, že vstupní data budou splňovat podmínky Eulerova

cyklu, je velmi nepravděpodobná. Většina vstupních grafů se tedy Eulerizifikuje.

V rámci implementace jsem našel na internetu program [8] na určení, zda graf obsahuje Eulerovský cyklus, Eulerovský tah nebo není Eulerovský. Následně jsem naimplementoval vlastní algoritmus, s využitím Dijkstrova algoritmu, k přidání nejkratších cest, v případě, že graf není eulerovský.

3.1.2 Fleuryho algoritmus

Pro Fleuryho algoritmus jsem využil již připravený program z internetu [9]. Algoritmus programu je následující:

1. Zjistíme, zda má graf 0 nebo 2 liché vrcholy.
2. Vytvoříme kopii vstupního grafu, se kterou se bude pracovat.
3. Zvolíme počáteční vrchol. Pokud má graf 0 lichých vrcholů, můžeme začít kdekoli. Pokod máme liché vrcholy 2, začneme u jednoho z nich. Počáteční vrchol nazveme v.
4. Procházíme hrany grafu následujícím způsobem:
   - Pokud nemá vrchol v žádné sousedy, algoritmus končí.
   - Pokud má vrchol v právě jednoho souseda u. Nastaví se v na u a hrana se odstraní.
   - Pokud má vrchol v více než jednoho souseda vybírá se takový vrchol, který není mostem. Následně se nastaví v na u a hrana se odstraní.

3.1.3 Hierholzerův algoritmus

Hierholzerův algoritmus jsem též našel na internetu [10] , avšak algoritmus byl konstruován pro orientované grafy, takže jsem ho trochu upravil, abych jsem ho mohl testovat na stejných datech jako Fleuryho algoritmus. Postup algoritmu je následující:

1. Vybereme náhodný uzel, ve kterém budeme začínat.
2. Vybíráme nenavštívené hrany k sousedním vrcholům dokud se nevrátíme zpátky do výchozího vrcholu. Tímto vytvoříme uzavřený sled. Pokud jsme prošli všechny hrany, našli jsme Eulerovský cyklus a algoritmus končí. Pokud ne pokračujeme krokem 3.

3. Vybíráme vrcholy s nevštívenými hranami a nacházíme další uzavřené sledy, dokud nejsou navštíveny všechny hrany.

4. Spojíme všechny uzavřené sledy dohromady.

3.2 Navržené řešení

Rozhodl jsem se vytvořit vlastní řešení problému. Napadlo mě využít backtacking a to tak, že budu brát hrany grafu, které si zároveň budu označovat jako navštívené. Pokud se dostanu do vrcholu, ve kterém jsou všechny hrany navštívené, vrátím se o krok zpět a zkusím vzít jinou hranu.

Algorithm 1 My algorithm

```
1: Spočítej počet hran
2: Aktuální vrchol ← vrchol 0
3: while je počet hran větší než počet navštívených hran do
4:     while je k dispozici nevyzkoušená hrana z aktuálního vrcholu do
5:         if hrana nebyla navštívena then
6:             Označ hranu za navštívenou
7:             Aktuální vrchol ← vrchol, kam vede hrana
8:             Inkerementuj počet navštívených hran
9:             break
10:        else
11:            Označ hranu za vyzkoušenou
12:        end if
13:    end while
14:    if všechny hrany z aktuálního vrcholu jsou vyzkoušeny then
15:        Označ hranu z aktuálního do předchozího vrcholu za nenavštívenou
16:        Aktuální vrchol ← Předchozí vrchol
17:        Dekrementuj počet navštívených hran
18:    end if
19: end while
```

Jedná se o celkem hloupý algoritmus (brute-force), kde záleží jaké štěstí budeme mít při vybírání hran. U této metody bude složitost v nejlepším případě $O(|E|)$ a to právě, když se nikdy nebudeme vracet. Bohužel v nejhorším případě se dosahuje složitosti až $O(|E!|)$ a to, když bude potřba vyzkoušet všechny možné kombinace, než se dojde ke správnému výsledku.

4 Experimenty a výsledky

Pro experimenty jsem si vytvořil program, kam se pokusil implementovat a zkopírovat všechny metody z 3. Výsledný program je dostupný na mém GitHubu [12].

Aby byly experimenty co nejvíce vypovídající, vytvořil jsem si metody na generování a načítání dat . Pro testování jsem volil různě veliké grafy s různou hustotou hran. Narazil jsem, zde na problém implementace Hierholzerova algoritmu, neboť je to primárně algoritmus pro orientované grafy. V testovaném programu jsem nechal variantu pro orientovaný graf (neorientovaná verze nefungovala), a tak Hierholzerův algoritmus prochází 2x více hran než ostatní algoritmy a teoreticky by měli být všechny naměřené časy tohoto algoritmu poloviční. Dále jsem se setkal s problémem přetečení zásobníku u testování větších grafů s velkým počtem hran u Fleuryho algoritmu. Výsledky měření můžeme nalézt v tabulce 4.

Tabulka 1: Doba běhů různých metod v různých grafech na procesoru Intel Core i5-7300HQ

Počet vrcholů	Počet hran	Fleury	Hierholzer	Vlastní
5	8	1 ns	1 ns	1 ns
10	14	2 ns	1 ns	1 ns
36	290	35 ns	8 ns	4 ns
50	49	4 ns	5 ns	10 ns
50	50	3 ns	1 ns	2 ns
50	1225	34 ns	17 ns	16 ns
95	755	84 ns	22 ns	7 ns
191	2360	141 ns	41 ns	44 ns
256	6320	- [1]	68 ns	55 ns
450	8169	-	83 ns	64 ns
450	17425	-	166 ns	> 20 min
900	307350	-	6778 ns	3786 ns

Pro zjištění, který graf je efektivní ve kterém typu grafu jsem vygeneroval 3 grafy s 50 vrcholy. První graf se 49 vrcholy představuje jednu cestu z vrcholu 1 do vrcholu 49 (algoritmus se následně musí vrátit do počátečního bodu). Druhý graf představuje kružnici, tedy oproti předchozímu grafu má navíc hranu mezi vrcholem 1 a 49. Poslední graf má pak hrany mezi všemi vrcholy.

[1] Došlo k přetečení zásobníku DFS

Naměřené výsledky vycházejí podle předpokladů. Dokázalo se, že Fleuryho algorimus je celkem neefetivní, díky složitému hledání mostů. U větších dat navíc kvůli rekurzi v metodě prohledávání do hloubky, přetéká zásobník. Můj vlastní algoritmus mě celkem překvapil, že je ve většině případů stejně rychlý, ne-li rychlejší než Hierholzerův, což je nejspíše způsobeno procházením dvojnásobného počtu hran, kvůli používání orientovaného grafu v Hierholzerově metodě. Také se prokázalo, že vlastní algoritmus není spolehlivý a záleží na náhodě jak se vybírají hrany. U grafu s 450 vrcholy a 17425 hranami, se začala vybírat úplně špatná cesta, což zapříčinolo dlouhou dobu trvání programu. Za nejsplehlivější bych jsem označil Hierholzerův algoritmus, který měl čas úměrný k počtu vrcholu a hran.

5 Závěr

Ohledně problému jsem si zjsitil informace a našel existující implementace problému čínského pošťáka, zároveň jsem se pokusil vymyslet svůj algoritmus. V rámci experimentů byl vytvořen jednochý program, do kterého se implemetovaly 3 metody na řešení daného problému. Z experimentů vyplynulo, že nejrychleší a nejspolehlivější je Hierholzerův algoritmus. Vlastní algoritmus funguje povětšinou dobře srovnatelně s Hierholzerův, avšak pro některá data trvá nesmírně dlouho. Fleuryho se prokázal jako celkem neefektivní algoritmus.

6 Reference

[1] The algorithm design manual, Skiena, Steven S, 2, 1998, Springer.

[2] Fleury's algorithm, `https://slaystudy.com/fleurys-algorithm/`, SlayStudy, 8, Sarthak Jain February, 2020, Jul.

[3] PROBLEM WYZNACZENIA CYKLU EULERA W GRAFIE NIESKIEROWANYM, STACHNIK, Paulina and ŚLIWA, Michał.

[4] Eulerian path `https://en.wikipedia.org/wiki/Eulerian_path`, Wikipedia, Wikimedia Foundation, 2021, Oct.

[5] Tarjan's strongly connected components algorithm, `https://en.wikipedia.org/wiki/Tarjan's_strongly_connected_components_algorithm`, Wikipedia, Wikimedia Foundation, 2021, Aug.

[6] Speeding up dynamic shortest path algorithms, Buriol, Luciana S and Resende, Mauricio GC and Thorup, Mikkel, Rapport technique TD-5RJ8B, AT&T Labs Research, 41, 2003, Citeseer.

[7] Hierholzer's Algorithm, https://slaystudy.com/hierholzers-algorithm/, SlayStudy, 12, Nurgazy Nazhimidinov, July, 2021, Mar.

[8] Eulerian path and circuit for undirected graph, https://www.geeksforgeeks.org/eulerian-path-and-circuit/?ref=lbp, GeeksforGeeks, 2021, Aug.

[9] Fleury's Algorithm for printing Eulerian Path or Circuit, https://www.geeksforgeeks.org/fleurys-algorithm-for-printing-eulerian-path/?ref=lbp, GeeksforGeeks, 2021, Sep.

[10] Hierholzers Algorithm with Implementation in Java, https://www.thecrazyprogrammer.com/2021/04/hierholzers-algorithm.html, The Crazy Programmer, Sinha, Vijay, 2021, Apr.

[11] Chinese postman problem, Resources available, http://www.harold.thimbleby.net/cpp/index.html.

[12] 2. Semestralni práce PRO, PRO/SP-2 at main · LRunt/PRO, https://github.com/LRunt/PRO/tree/main/SP-2, GitHub, Runt, Lukáš.

[13] Graph Coloring Instances, https://mat.gsia.cmu.edu/COLOR/instances.html#XXSGB.