Semestrální práce z KIV/PC

Vyhledávání cest v grafu technikou DFS

Radek Juppa

Student: A16B0039K

20. 1. 2018

Obsah

1. Zadání

Naprogramujte v ANSI C přenositelnou konzolovou aplikaci, která bude procházet graf technikou DFS (Depth-First Search). Vstupem aplikace bude soubor s popisem grafu. Výstupem je pak odpovídající výčet všech cest mezi požadovanými uzly grafu.

Program␣se␣bude␣spouštět␣příkazem␣dfs.exe␣<soubor-grafu>␣<id1>␣<id2>␣<maxD>

- Symbol <soubor-grafu> zastupuje parametr - název vstupního souboru se strukturou grafu.
- Následují identifikátory (dále jen id) dvou uzlů v grafu <id1> a <id2>, mezi kterými bude spouštěn proces hledání cest.
- <maxD>je parametr popisující maximální délku cest, které mají být nalezeny.
- Vámi vyvinutý program tedy bude vykonávat následující činnosti.
    1. Při spuštění bez potřebných parametru vypíše nápovědu pro jeho správné spuštění a ukončí se.
    2. Při spuštění s parametry načte zadaný vstupní soubor do vhodné struktury reprezentující graf a mezi zadanými uzly najde všechny cesty, jejichž délka nepřekročí konstantu nastavenou posledním parametrem.

1.a) Specifikace výstupu programu

Program bude na standardní výstup vypisovat jednotlivé cesty. Vždy na jeden řádek právě jednu cestu. Cesty budou popsány posloupností id jednotlivých uzlů oddělených pomlčkou (tj. znakem minus). Následovat bude středník a popisky jednotlivých hran v grafu oddělené čárkou (viz Příloha 1). Za posledním středníkem bude uvedena hodnota sekundární metriky - relevance cesty, podle které budou cesty seřazeny (viz Řazení výstupu).
Například tedy pro hledání cest mezi uzly A a B:

A-B;h1;m1
A-F-B;h2,h3;m2,3
A-W-B;h4,h5;m4,5
A-F-O-B;h2,h6,h7;m2,6,7

Kde A,B,F,W,F,... jsou popisky uzlů grafu, hn jsou ohodnocení hran a mc1,c2...cn je hodnota metrika relevance nalezené cesty.

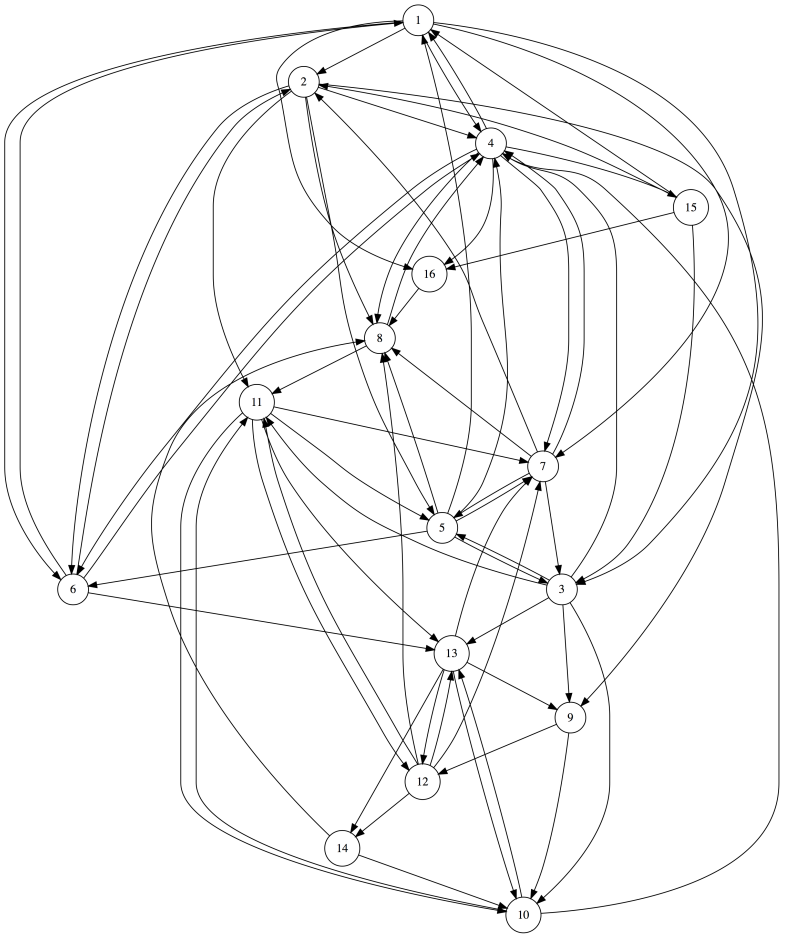
1.b) Řazení výstupu

Cesty budou na standardním výstupu seřazeny primárně podle jejich délky. Pokud bude nalezeno více cest stejné délky, budou seřazeny podle jejich relevance následujícím způsobem. Popisky v grafu nesou informaci o kalendářním datu ve formátu YYYY-MM-DD. Každá cesta bude tedy ohodnocena celým číslem, které bude odpovídat rozdílu v počtu dní mezi nejstarším a nejnovějším datem, následně budou podle tohoto čísla cesty se shodnou délkou seřazeny vzestupně.

2. Analýza úlohy

2.a) Analýza grafu

Nejprve se pojďme podívat, co to je za graf. Na první pohled je vidět, že je to velký graf s několika tisíci uzly. Vizualizace celého grafu by byla dost obtížná. Proto si vyberu jen nějaký podgraf. Použil jsem k tomu *Python* a knihovnu *networkx*. Vybral jsem si prvních 16 různých vrcholů a vzal jsem v úvahu jen 8 sousedů každého vrcholu. Vizualizaci tohoto podgrafu jsem provedl pomocí knihovny *graphviz*.

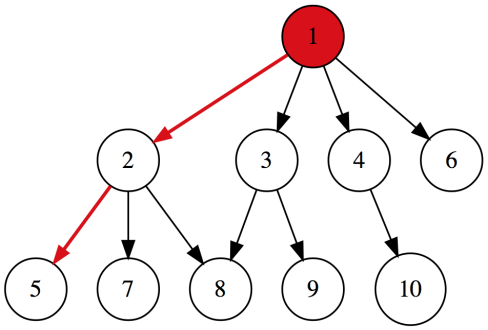


Obrázek 1: podgraf

Na obrázku vidíme, že se jedná o orientovaný graf s kružnicemi. My ale máme s grafem zacházet jako s orientovaným. To znamená, že s každou hranou přidáme i hranu opačnou. Délka cesty je počet hran. Cesta nesmí obsahovat jeden vrchol vícekrát. Například cesta 1-2-6 má délku 2. Cesta 1-6 má délku 1. A sled 1-2-8-4-2-6 není cestou, protože obsahuje vrchol 2 více než jednou.

2.b) Metoda procházení grafu

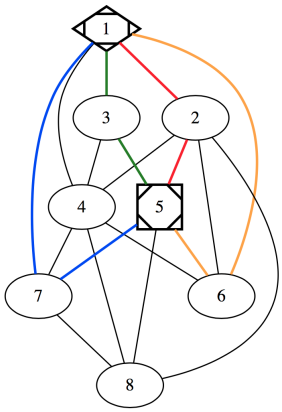
Pro prohledávání grafu existují 2 základní metody *Breadth First Search - BFS* a *Deep First Search - DFS*. My máme použít techniku DFS, což je prohledávání do hloubky. Konkrétně to znamená, že při prohledávání postupujeme po listech do hloubky. Tedy na obecném grafu, který je zobrazen na obrázku 2 to bude sekvence vrcholů [1-2-5-7-8-3..] Kdežto BFS prochází nejdříve nejbližší uzly. BFS sekvence by vypadala následovně: [1-2-3-4-6-5-7...]



Obrázek 2: DFS

2.c) Algoritmus

Vraťme se teď opět k našemu grafu. Vezmu si ještě menší podgraf (Vyberu podgraf s vrcholy: 1, 2, 4, 5, 6, 7, 8) a zkusím najít všechny cesty mezi vrcholy 1 a 5 s maximální délkou cesty 2. viz obrázek 3



Obrázek 3: všechny cesty z 1 do 5 (maxD=2)

Zde je tabulka všech cest o maximální délce 2 z vrcholu 1:

1-2-4 NE	1-3-5 ANO	1-4-8 NE	1-7-5 ANO	1-6 NE
1-2-5 ANO	1-4-2 NE	1-6-2 NE	1-7-8 NE	1-7 NE
1-2-6 NE	1-4-3 NE	1-6-4 NE	1-2 NE	
1-2-8 NE	1-4-6 NE	1-6-5 ANO	1-3 NE	
1-3-4 NE	1-4-7 NE	1-7-4 NE	1-4 NE	

Tabulka 1: cesty z vrcholu 1 do vrcholu 5 (maxD = 2)

Je vidět, že jen 4 cesty vedou do vrcholu 5. Budu tedy používat techniku *DFS*. Vrcholy, jimiž jsem prošel, si budu ukládat do zásobníku. Tak budu postupovat, dokud budu mít kam vstupovat nebo dokud nevstoupím do cílového vrcholu nebo nedosáhnu maximální délky cesty. V opačném případě se vrátím do vrcholu, ze kterého jsem přišel. Pokud vstoupím do cílového vrcholu, cestu si uložím jako jedno z řešení. Na konci pak všechny nalezené cesty seřadím a vytisknu.

3. Popis implementace

Aby bylo možno graf procházet, je třeba ho nejprve uložit do nějaké vhodné struktury. Samotný graf tvoří množina vrcholů a hran. Vrchol je tvořen jednou integer hodnotou a hrana spojuje 2 vrcholy a je ohodnocena řetězcem znaků. Vrcholy i hrany uložím do spojového seznamu. Již jsem zmínil, že pracujeme s grafem neorientovaným, proto každou hradu uložím v obou směrech.

Struktura pro vrchol *VERTEX* bude vypadat následovně:

```
typedef struct vertex{       // VERTEX (as linked list)
    int id;                  // id of the vertex
    EDGE *first_edge;        // reference to the first edge (as linked list)
    struct vertex *next;     // next item
} VERTEX;
```

Hrany uložím do struktury *EDGE*:

```
typedef struct edge{        // EDGE (as linked list)
    int id;                 // id of vertex creating an edge with its parent
    char *data;             // data(label) assign to the edge
    struct edge *next;      // next item
} EDGE;
```

Tak, jak budu grafem procházet, budu si jednotlivé vrcholy ukládat do zásovníku *STACK*:

```
typedef struct stack_item{
    int id;                  // id of a vertex
    char *data;              // label of an edge
    struct stack_item *next; // reference to next item in linked list
} STACK_ITEM;

typedef struct stack{
    STACK_ITEM *top;         // the first vertex
    int count;               // count of items in the stack
} STACK;
```

Zásobník je implementován ve svém vlastním modulu nazvaném *stack.c*. Bude ukládat aktuální cestu a budu k tomu používat dvě typické metody:

```
void stack_push(STACK *stack, int id, char *data);
int stack_pop(STACK *stack);
```

Pokud aktuální cesta dosáhne cílového vrcholu, bude uložena do seřazeného spojového seznamu úspěšných cest typu *RESULT*. Tuto strukturu definuji následovně:

```
typedef struct result{       // RESULT (as sorted linked list)
    int *vertexes;           // array of vertex ids
    struct tm *labels;       // array of labels (datetime struct)
    int length;              // length of path
    int score;               // number of days between the first and the last date
    struct result *next;     // next item
} RESULT;
```

Pro uložení celkové struktury grafu včetně aktuální cesty a výsledků bude sloužit struktura *GRAPH*:

```
typedef struct graph{
    VERTEX *first_vertex;
    STACK *current_path;
    int count;
    int start_vertex_id;
    int target_vertex_id;
    int limit;
    RESULT *result;
} GRAPH;
```

Graf je implementován ve svém vlastním modulu nazvaném *graph.c*. Nejdůležitejší metodou, která prochází graf a hledá cesty je metoda:

```
void graph_dfs(GRAPH *g)
```

4. Uživatelská příručka

Program je dodán v podobě zdrojového kódu a proto je nutno jej nejprve přeložit. Přeložení programu je shodné pro všechny hlavní platformy: Windows, Linux, MacOS

Předpokladem úspěšného přeložení je přítomnost překladače jazyka C. V kořenovém adresáři je soubor *makefile*, který zajištuje překlad.
Překlad se provede příkazem: *make*

```
$ make
```

Příkaz make vypíše zprávu o spuštění překladače a vytvoří soubor *dfs.exe* V případě problému se nejprve přesvěčte, že compiler jazyka C funguje. Napište příkaz pro vypsání verze překladače: *gcc -version* Pokud nezobrazi verzi, pak pravděpodobně není nainstalován nebo k němu chybí cesta v proměnné PATH. Pokud se Vám program podařilo přeložit a vytvořil se soubor *dfs.exe*, pak jej můžete spustit následujícím způsobem:

```
$ ./dfs.exe <cesta> <id_start> <id_target> [maxDistance=5]
```

Pokud nezadáte povinné vstupní parametry, program vypíše nápovědu:

```
$ ./dfs.exe
***** Search for Path using DFS ******
* Seminar work of 'Programming in C' *
*  Copyright (c) Radek Juppa, 2017   *
*************************************
Usage:  dfs.exe <filename> <id1> <id2> [maxD]
Example:
  dfs.exe graph.csv 1 2 3
```

Příklad spuštění programu:

```
$ ./dfs.exe sw2017-02-data.csv 1 29 2

1-6-29;2007-06-14,2007-06-11;3
1-25-29;2007-02-04,2007-02-01;3
1-2-29;2007-02-16,2007-04-30;73
1-3-29;2007-10-26,2008-01-18;84
1-882-29;2007-11-02,2007-08-03;91
1-4698-29;2007-05-29,2007-10-04;128
1-21935-29;2007-01-31,2007-06-26;146
1-780-29;2007-10-29,2007-05-30;152
1-67-29;2007-08-07,2008-01-08;154
1-7-29;2007-08-14,2007-02-23;172
1-67-29;2007-05-08,2008-01-08;245
1-25-29;2007-10-31,2007-02-01;272
1-2-29;2008-02-15,2007-04-30;291
1-3306-29;2008-01-17,2007-02-27;324
1-777-29;2008-02-14,2007-03-13;338
1-16-29;2008-03-05,2007-03-09;362
1-2917-29;2008-09-09,2007-08-05;401
1-1230-29;2007-01-15,2008-03-19;429
```

Program načetl soubor *sw2017-02-data.csv* z aktuálního adresáře a vyhledal všechny cesty mezi vrchly 1 a 29, které jsou dlouhé maximálně 3 hrany. Vidíme, že program našel 18 cest a vypsal je seřazené podle jejich délky a hodnoty hran.

5. Závěr

Tato úloha mi dala příležitost si vyzkoušet dynamické vytváření datovách struktur jako je spojový seznam, zásobník či graf pomocí jazyka ANSI C. Dále jsem si vyzkoušel nástroj pro kontrolu uvolňování paměti programem *Valgrind*. A naučil jsem se napsat jednoduchý *makefile* pro různé platformy. Program jsem testoval v prostředí Linuxu s překladačem *GCC* a v prostředí Windows s překladačem *MinGW*. Na platformě Windows jsem narazil na problém s kompilaci. Chyběla tam funkce *strptime()*, kterou jsem původně používal k parsování datumu. Tuto funkci jsem nahradil svojí vlastní funkcí, využívající funkci *sscanf*. Potom už to prošlo i na Windows. Úkoly a překážky, se kterými jsem se během práce setkal považuji za solidní úvod do programování v jazyce ANSI C. Dokumentaci jsem vytvořil v programu LYX.