Semestrální práce z KIV/PC

VYHLEDÁVÁNÍ CEST V GRAFU TECHNIKOU DFS

Jan Brtek
A15B0281P
brtekj@students.zcu.cz

1. 1. 2017

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.Vámi vyvinutý program tedy bude vykonávat následující činnosti.

Při spuštění bez potřebných parametru vypíše nápovědu pro jeho správné spuštění a ukončí se. 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. **Podrobné zadání úlohy se nachází na webové stránce předmětu Programování v jazyce C.**

2 Analýza úlohy

2.1 Rozdělení problému

Po prvotní analýze je jasné, že si problém rozčleníme do více částí, které budeme postupně řešit. Rozhodl jsem se pro tyto:

1. Načtení dat ze souboru
2. Implementace a volba vhodné reprezentace grafu
3. Depth-First Search implementace
4. Nalezení a uložení VŠECH cest
5. Výpis a výstup

2.2 Načtení souboru

2.2.1 Soubor se strukturou grafu

Soubor můžeme číst po řádkách či znacích. Jelikož dopředu nemůžeme vědět, jak dlouhá bude řádka, rozhodl jsem se číst soubor po znacích a rovnou při načítání rozdělovat a ukládat informace informace podle klíčových znaků.

2.3 Implementace a volba vhodné reprezentace grafu

Graf lze implementovat dvěma způsoby. Pomocí matice sousednosti a nebo pomocí spojového seznamu. Zvolil jsem spojový seznam, jelikož je mi práce s ním bližší.

1. Struktura s grafem obsahuje ukazatel na spojový seznam prvků grafu a celočíselné údaj o maximálním počtu hran, počet hran a počet cest.
2. Struktura s prvkem grafu uchovává ID prvku, ID rodiče a pointery na strukturu Hrana a na příští prvek.
3. Struktura pro hranu grafu obsahuje celočíslené údaje o propojených hranách a její hodnotu.

2.4 Struktura grafu

2.4.1 Graf

Jelikož jsem se pozhodl pro implementaci spojovým seznamem, struktura bude obsahuje ukazatel na ukazatel na prvek. Dále budeme potřebovat základní informace o grafu, které nám v budoucnu pomohou pro práci s ním. Určitě se nám bude hodit počet prvků a počet cest. Oba tyto číselné údaje budou nezáporné, můžeme proto použít datový typ unsigned int. Struktura by tedy mohla mít tuto podobu:

```
typedef struct Graf {
   unsigned int pocetPrvku, pocetCest;
   prvek **ppPrvek;
} Graf;
```

2.4.2 Prvek grafu

Pro prvek grafu se hodí znát jeho ID, ukazatel na hranu, kterou je propojen s dalším prvkem a další prvek v seznamu.

```
typedef struct Prvek {
   unsigned int id
   hrana *pHrana;
   prvek *pPrvek;
} Prvek;
```

2.4.3 Hrana

Struktura pro hranu grafu bude uchovávat ID obou propojených uzů, hodnotu hrany a retezec, ktery hrana reprezentuje.

```
typedef struct Hrana {
   unsigned int id_OD;
   unsigned int id_DO;
   unsigned int vaha
   char *string;
} Hrana;
```

2.5 Nalezení všech cest

2.5.1 Zasobnik

Jelikož dle zadání je třeba najít více než jednu cestu, bude se hodit datová struktura pro uložení již nalezených cest. Pro tento případ se bude hodit práce se zásobníkem. Zde stačí klasický zásobník s následující strukturou:

```
typedef struct Zasobnik {
   unsigned int velikost;
   unsigned int pozice;
   prvek** ulozenyPrvek;
} Zasobnik;
```

2.5.2 Cesta

Datová struktura pro uložení cesty a základních informací k ní. `typedef struct Cesta {`

```
   unsigned int delka;
   unsigned int vaha;
   unsigned int maxDelka;
   hrana **pHrana;
   uint *pPrvek;
} Cesta;
```

3 Popis implementace

Program jsem rozdělil do více modulů z důvodu přehlednosti a funkční logiky.

3.1 Executable

Tento modul obsahuje pouze funkci `main`, která zajišťuje chod celého programu. Ověřuje správný počet argumentů,volá funkci pro načtení souboru, vytvoření grafu a nalezení příslušných cest. Poté zavolá funkci na seřazení cest a pomocí dalšího volání cety vypíše.

3.2 soubor

Soubor shromažďuje funkce pro práci se vstupním souborem. Umožňuje otevření a načtení souboru, jeho rozparsování dle kriterií a případné zvětšení bufferu. Obsahuje funkce:

- `hrana **rozdel(FILE *soubor)`
- `hrana **nacti_soubor(char *jmeno)`
- `char *zvetsi_buffer(char *pole, int velikostPole)`

Funkce nacti_soubor slouží k otevření souboru, na jejím konci je volání funkce rozdel, která čte a ukládá vstupní informace.

3.3 prvek

Obsahuje strukturu `prvek`, která si uchovává informace o prvku grafu a ukazatele na jeho hrany a dalsi prvek v seznamu. Vypadá takto:

```
typedef struct {
    struct edge *pHrana;
    struct node *pDalsi;
    unsigned int id;
    unsigned int id_min;
    int isDone;
} prvek;
```

Součástí jsou také následující funkce:

- `prvek *vytvor_prvek(unsignedi int idPrvku, hrana *pHrana)`
- `void pridej_souseda(uint idPrvku, prvek *pPrvek, hrana *pHrana)`
- `int jeKam_jit(prvek *pPrvek)`
- `void uvolni_prvek(prvek **ppPrvek)`
- `void command_execute(parameters *par, node **actual)`

Funkce `vytvor_prvek` vytvoří nový uzel grafu a uloží mu příslušené informace. Podobnou funkci zastupuje `vytvor_souseda`, která ale již existujícímu prvku vytvoří souseda a nastaví parametry. Funkce `jeKam_jit` zjišťuje, zdali daný prvek má ještě nějaké sousedy, kteří nebyli zatím při hledání cesty využiti.

3.4 hrana

Slouží k reprezentaci hran grafu. Obsahuje strukturu `hrana`, která si uchovává informace o hranách grafu. Vypadá následovně:

```
typedef struct {
   unsigned int id_DO
   unsigned int id_DO
   char *string
   unsigned int pocet_dni
} hrana; } hrana;
```

Součástí jsou také následující funkce:

- `hrana *vytvor_hranu(unsigned int id_OD, unsigned int id_DO, char *string, unsigned int pocet_dni)`
- `void hrana_uvolni(hrana **pp_Hrana)`

3.5 graf

Slouží k reprezentaci celého grafu. Obsahuje strukturu `graf`, která si uchovává informace o grafu jako celku. Vypadá následovně:

```
typedef struct {
   unsigned int pocetCest
   unsigned int maxPrvku
   unsigned int pocetPrvku
```

```
   struct node **prvkyGrafu;
} graf;
```
Součástí jsou také následující funkce:

- `graf *vytvor_graf(hrana **ppHrany, int pocetHran)`
- `void uvolni_graf(graf **ppGraf)`
- `void void pridejPrvek(graf *pGraf, hrana *pHrana)`
- `int vratPocetPrvku(hrana **ppHany, int pocet_hran)`

3.6 Cesta

Slouží k reprezentaci cest uvnitř celého grafu.

```
typedef struct {
   unsigned int delka
   unsigned int vaha
   unsigned int maxDelka
   struct node **ppHrana;
   struct node *pPrvek;
} graf;
```
Součástí jsou také následující funkce:

- `graf *vytvor_graf(hrana **ppHrany, int pocetHran)`
- `void uvolni_graf(graf **ppGraf)`
- `void void pridejPrvek(graf *pGraf, hrana *pHrana)`
- `int vratPocetPrvku(hrana **ppHany, int pocet_hran)`

3.7 DeepFirstSearch

Slouží k realizaci a implementaci Deep-First-Search. Neobsahuje žádnou datovou strukturu, obsahuje pouze funkce pro vyhledávání a výpis:

- `cesta *vytvorCestu(uint maximalni_cesta)`
- `cesta **zvetsiPole(cesta **ppPole, int velikost)`
- `void pridejPrvekDoCesty(cesta *pCesta, prvek *pPrvek)`
- `void pridejHranuDoCesty(cesta *pCesta, hrana *pHrana)`

- `int porovnejCesty(cesta *pCesta, cesta *pCesta_druha)`
- `int jeDuplicitni(cesta *pCesta, cesta **ppCesty, int pocet_cest)`
- `void vypoctiVzdalenost(cesta *pCesta)`
- `void uvolniCestu(cesta **ppCesta)`

4 Uživatelská příručka

4.1 Přeložení programu

Pro úspěšný překlad musíme mít nainstalovaný překladač gcc a také musíme mít k dispozici všechny tyto soubory:

- `Executable.c`
- `DeepFirstSearch.c`
- `DeepFirstSearch.h`
- `cesta.c`
- `hrana.h`
- `prvek.c`
- `prvek.h`
- `cesta.h`
- `cesta.c`
- `hrana.h`
- `hrana.c`
- `soubor.c`
- `soubor.h`
- `zasobnik.c`
- `zasobnik.h`
- `makefile`

Překládáme v konzoli v adresáři, kde máme výše vypsané soubory, příkazem odlišném v:

- GNU/Linux

  příkaz: `make`

- Microsoft Windows

  příkaz: `mingw32-make`

4.2 Spuštění programu

- GNU/Linux

  `./dfs.exe <vstsup.csv> <id1> <id2> <maxCesta>`

- Microsoft Windows

  `dfs.exe<vstsup.csv> <id1> <id2> <maxCesta>`

4.3 Výstup programu

Výstup programu je specifikovaný v zadání na stánkách Programování v jazyce C.

Ukázkový výstup může vypadat například:

1-6;2000-01-01;0
1-5-6;2000-01-02,2000-01-03;1
1-7-6;2000-01-01,2001-01-02;367
1-8-6;2007-10-11,2000-01-02;2839
1-8-7-6;2007-10-11,2007-06-14,2001-01-02;2473
1-4-5-6;2007-02-16,2007-03-16,2000-01-03;2629
1-7-8-6;2000-01-01,2007-06-14,2000-01-02;2721

5 Závěr

Program prošel správně validací a funguje v plném rozsahu zadání. Během psaní kódu jsem lehce pozměnil datové struktury, dle potřeb v danou chvíli. Původní analýza problému byla poměrně přesná. Během práce jsem ještě při řazení hran začal používat algoritmus Bubble Sort, z důvodů úspory času.

Největším problémem bylo nakonec paradoxně využívání kvalitního IDE, jelikož Visual Basicu při práci nevadila deklarace proměnné uvnitř cyklu. Při odevzdávání ovšem validátor práci odmítal a tak jsem musel práci ještě na poslední chvíli upravit a hledat v ní chybu. Bohužel mi kvůli tomuto problému nezbylo dostatek času na kvalitnější zpracování dokumentace.

Aplikaci jsem testoval na počítači s operačním systémem Microsoft Windows 10 Home, s pamětí RAM 12 GB a procesorem Intel® Core™ i7-4447U s frekvencí procesru 4,2 GHz.