Semestrální práce z KIV/PC

Řešení kolizí frekvencí sítě vysílačů

Daniel Stuš
A17B0354P
stusd@students.zcu.cz

09. 10. 2020

Obsah

1 Zadání

Naprogramujte v ANSI C přenositelnou **konzolovou aplikaci**, která jako vstup načte z parametru příkazové řádky název textového souboru obsahujícího informace o parametrech a pozicích vysílačů na mapě a na jeho základě přidělí každému vysílači frekvenci tak, aby jeho signál nerušil vysílání vysílačů v jeho bezprostředním okolí.

Program se bude spouštět příkazem `freq.exe <filename>`. Symbol `<filname>` zastupuje jméno textového souboru, který obsahuje informace o rozmístění vysílačů na mapě a o dostupných vysílacích frekvencích, které jim je možné přidělit.

Váš program tedy může být během testování spuštěn například takto:
`...\>freq.exe vysilace-25.txt`

Výsledkem práce programu bude výpis do konzole, na kterém bude seznam přidělených frekvencí každému vysílači ze vstupního souboru (viz Specifikace výstupu programu). V případě chyby nebo neřešitelné situace nechť program skončí výpisem příslušné chybové hlášky (v angličtině).

Pokud nebude na příkazové řádce uveden právě jeden arguemnt, vypiště chybové hlášení a strušný návod k použití programu v angličtině podle běžných zvyklostí (viz např. ukázková semestrální práce na webu předmětu Programování v jazyce C). **Vstupem programu je pouze argument na příkazové řádce – interakce s uživatelem pomocí klávesnice ři myši v průbehu práce programu se neočekává.**

Hotovou práci odevzdejte v jediném archivu typu ZIP prostřednictvím automatického odevzdávacího a validačního systému. 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, resp. LATEX. Podrobné informace k odevzdání práce najdete na webové stránce předmětu Programování v jazyce C: `https://www.kiv.zcu.cz/studies/predmety/pc/index.php#work` – bude-li některá z částí chybět, kontrolní skript Vaši práci odmítne.

2 Analýza úlohy

Pro vyřešení zadané úlohy je potřeba vhodně načíst a uložit vstupní data, z nich sestrojit graf kolizí a ten následně obarvit.

Hlavním úkolem je tedy zvolit vhodnou strukturu pro reprezentaci vstupních dat a vhodnou strukturu pro reprezentaci grafu. Algoritmus obarvování grafu byl popsán v zadání a není tedy předmětem zkoumání.

2.1 Načtená data

Stěžejní částí načítání dat je zvolit vhodný postup jejich ukládání, neboť dopředu nevíme jejich množství. Nabízí se nám v podstatě dvou možností.

První způsob je projít vstupní data dvakrát. Při prvním průchodu si uložit počet frekvencí a vysílačů, vytvořit pole o příslušných velikostech, a při druhém průchodu tyto pole následně naplnit. Tím zajistíme, že množství alokované paměti vždy odpovídá přesně paměti potřebné. Nevýhodou jsou však dva průchody vstupními daty, které v případě velkých souborů zpomalí běh programu.

Druhý způsob je vytvoření dynamicky se zvětšujícího pole (vektoru). V takovém případě je vstupní data potřeba projít pouze jednou. Vektory se před načítáním dat nastaví na výchozí velikost a pokud je počet dat v průběhu ukládání větší než velikost vektoru, jeho velikost se jednoduše zvětší o předem zvolé množství. Nevýhodou tohoto řešení je fakt, že konečná velikost vektoru je pravděpodobně větší než množství vstupních dat a plýtváme tak pamětí.

Je tedy otázkou zda preferujeme rychlost nad množstvím využité paměti a naopak. V implementaci této práce byl zvolen druhý způsob.

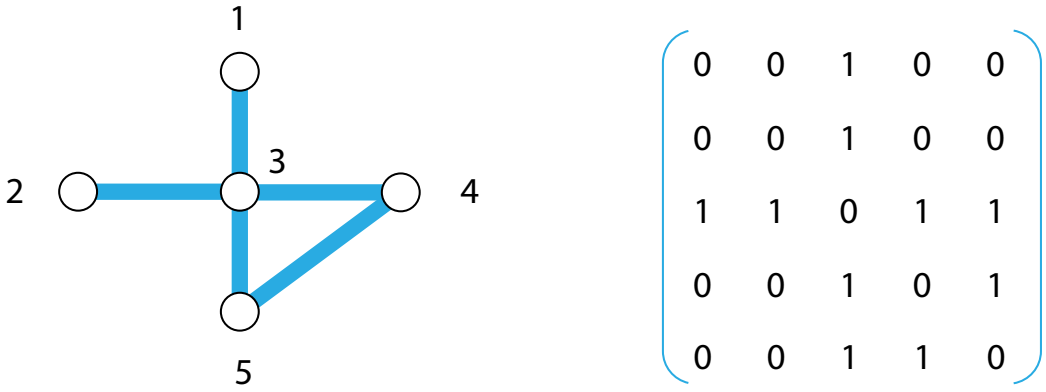
2.2 Graf

Graf kolizí je speciálním případem neorientovaného grafu s hranami u překrývajících se vysílačů. Neorientovaný graf je obecně definován jako dvojice $G =< V, E >$, kde `V` je neprázdná množina vrcholů a `E` neprázdná množina dvouprvkových množin vrcholů. V informatice existuje mnoho způsobů jak takový graf reprezentovat, z nichž každý má své případy použití. Nejčastější reprezentace, jejich slabé a silné stránky níže nastíním.

2.2.1 Matice sousednosti

Nejtypičtějším reprezentace grafu je zápis pomocí matice. Nejčastěji pak matice sousednosti. Řádky i sloupce matice představují uzly grafu. Hodnoty na jednotlivých pozicích matice nabívají pouze hodnot 1 a 0. Souřadnice daná řádkem m a sloupcem n bude jednotková právě tehdy, když mezi vrcholy m a n vede hrana. V opačném případě bude hodnota na daných souřadnicích nulová. V případě neorientovaného grafu je matice symetrická.

Paměť potřebná pro reprezentaci grafu tímto způsobem je rovna druhé mocnině počtu vrcholů V^2, neboť ukládáme i neexistující hrany. Výhodou této reprezentace je fakt, že jako jediná umožňuje ověřit existenci hrany v konstantním čase, a to až $O(1)$. Naopak nevhodná se tato reprezentace její u řídkých grafů a velkého množství uzlů, neboť plýtváme pamětí.



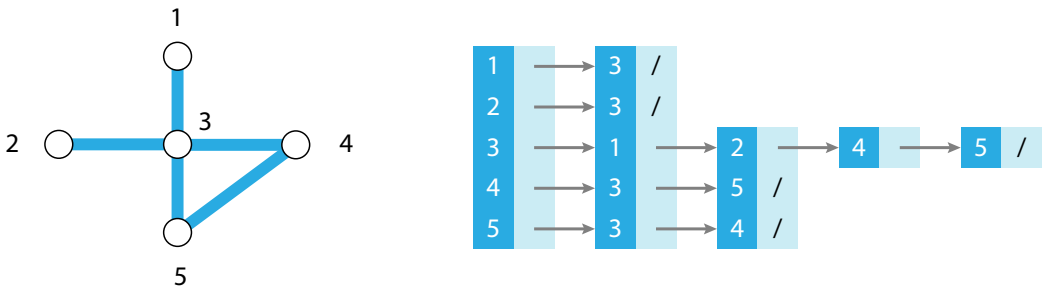
Obrázek 2.1: Příklad grafu a matice sousednosti

2.2.2 Spojový seznam

Dalším způsobem reprezentace grafu je pomocí spojového seznamu. Základní myšlenkou je, že každý z vrcholů si vnitřně ukládá seznam, který obsahuje všechny další vrcholi k němu připojené. Neuchováváme tedy žádné redundantní informace jako tomu je v případě matic.

Paměť, která je potřebná k uložení takové struktury je dána počtem hran, tedy $O(E)$. K zjištění, zda mezi dvěmi vrcholy vede hrana, je potřeba projít jeden seznam, tedy $O(V)$. Tato reprezentace je zejména vhodná pokud často procházíme seznam sousedů daného vrcholu a pokud pracujeme s řídkými grafy.

Vzhledem k povaze úlohy je realizován ve výsledné implementaci graf pomocí spojového seznamu.



Obrázek 2.2: Příklad grafu a spojového seznamu

3 Implementace

Aplikace je strukturována do několika souborů. Jmenovitě: `collission_graph.c, vector.c, stack.c a main.c`. Soubor `collission_graph.c` obsahuje reprezentaci grafu a funkce s ní spojené, soubor `vector.c` obsahuje reprezentaci vektoru (pole s dynamickou velikostí) s potřebnými funkcemi, soubor `stack.c` obsahuje reprezentaci zásobníku a funkce pro jeho ovládání a soubor `main.c` načítá data a řídí program voláním příslušných funkcí.

3.1 Popis struktur

3.1.1 Dynamické pole – Vektor

Frekvence a vysílače se ukládají do dynamicky se zvětšujících polí (vektorů). Ty jsou řešeny obecně (aby mohli uchovávat data libovolného formátu) jako struktura, která si uchovává informace o počtu aktuálně uložených prvků, své vlastní velikosti a ukazatel na pole ukazatelů typu `void`.

V případě frekvencí je dynamické pole naplněno ukazatelemi na hodnoty typu `int`.

V případě vysílačů se pak v poli ukládají jako ukazatel na vlastní strukturu obsahující dvě reálná čísla pro uložení souřadnic a jedno celé číslo, které představuje index frekvence přiřazené vysílači.

Funkce vektoru

vector_init Inicializace vektoru. Nastaví úvodní velikost a vytvoří úvodní pole.

vector_resize Umožňuje změnit velikost vektoru

vector_add Umožňuje přidat prvek do vektoru

***vector_get** Vrací prvek na požadované pozici

vector_free Uvolňuje paměť celého vektoru

3.1.2 Graf

Graf je složen ze dvou základních struktur. Struktura `graphnode` představuje jednotlivé uzly grafu a obsahuje celé číslo, reprezentující hodnotu uzlu a ukazatel na sousední uzel. Struktura `graph` představuje obecný graf a uchovává

si počet vrcholů obsažených v grafu a pole ukazatelů na uzly grafu (seznamy sousedů vrcholů).

Funkce grafu

***create_graph_node** Vytvoří uzel grafu

create_graph Vytvoří strukturu grafu

append_edge Přidá hranu na konec seznamu sousedů

add_collisions Přiřadí uzlům seznam sousedů (kolidujících vysílačů)

free_graph Uvolňuje paměť celého grafu

3.1.3 Zásobník

Zásobník slouží pro uchování indexů vysílačů čekajících na obarvení. Jeho struktura si tedy ukládá pouze celé číslo reprezentující vysílač a ukazatel na další prvek obsažený v zásobníku. Poslední vložený prvek je zároveň prvním prvkem, který je ze zásobníku vyjmut.

Funkce zásobníku

***create_stack_node** Vytvoří uzel zásobníku (prvek v zásobníku)

is_empty vrací 1, pokud je zásobník prázdný, jinak 0

push Vloží prvek na vrchol zásobníku

pop Odebere prvek z vrcholu zásobníku

in_stack Zjišťuje zda prvek není obsažen v zásobníku. Vrací 1 pokud ano, jinak 0

3.2 Načtení dat

Načtení dat probíhá v modulu `main`, ve funkci `parse_file`. Té se předají ukazatele na dvojici vektorů pro uložení frekvencí a vysílačů a ukazatel na proměnnou pro uložení poloměru. Soubor se prochází po řádcích a pomocí konstrukce `switch` se zjišťuje ve které části souboru se aktuálně nacházíme a podle toho provádíme uložení příslušných dat.

3.3 Vytvoření grafu kolizí

Graf kolizí se vytváří ve funkci `add_collisions`, které se předá ukazatel na graf, ukazatel na vektor vysílačů a poloměr. Funkce vypočítá vzdálenost mezi každým ze dvou vysílačů a pokud je tato vzdálenost menší než dvojnásobek poloměru vysílaného signálu (signáli vysílačů spolu kolidují), přiřadí příslušnému vysílači (uzlu v grafu) hranu ke kolidujícímu vysílači.

3.4 Obarvování grafu

Obarvování grafu probíhá dle pseudokódu uvedeného v zadání. Základem algoritmu je cyklus, který prochází postupně všechny vysílače a pokud narazí na vysílač, který ještě nemá přidělenou frekvenci (hodnota frekvence je rovna -1), vloží jej do zásobníku a tím ho předá ke zpracování. Při každém průchodu tímto cyklem se také zkontroluje jestli zásobník již neobsahuje vysílač určen ke zpracování, pokud ano, přejde se do vnořeného cyklu, který se opakuje dokud není zásobník prázdný.

Uvnitř vnořeného cyklu se postupně snažíme přiřadit vrcholu barvu (index do pole frekvencí) počínaje nejmenší hodnotou. Při každém přiřazení další vnořený cyklus projde všechny sousedy vrcholu (kolidující vysílače) a zjistí, zda některý z nich již nemá hodnotu frekvence, kterou se snažíme vrcholu přiřadit. Pokud tomu tak je, zvýší se přiřazovaná hodnota barvy o jedna a cyklus se opakuje (znovu se projde seznam všech sousedů). Pokud hodnota zkoumané frekvence přesáhne počet frekvencí, které můžeme přidělit, algoritmus prohlásí úlohu za neřešitelnou a ukončí program.

Pokud je barva úspěšně nalezena, přiřadí se aktuálně zkoumanému vrcholu a algoritmus přejde k cyklu, který naplní zásobník sousednímy vrcholy, které ještě nejsou obarveny a které ještě nejsou obsaženy v zásobníku.

Po úspěšném obarvení všech vrcholů se výsledek vypíše na konzoli, uvolní se veškerá přidělená paměť a program skončí.

4 Uživatelská příručka

4.1 Spuštění programu

Pro spuštění aplikace je potřeba ji nejprve přeložit nástrojem make. K tomu slouží soubory makefile (unix) a makefile.win (windows). Nástroj make zavoláme příkazem `make` v konzoli, v kořenovém adresáři programu. Tímto je aplikace přeložena a vznikne nový soubor s názvem `freq.exe`. Ten je možné spustit v unixovém operačním systému příkazem `./freq.exe <filename>`. Symbol `<filename>` zastupuje název textového souboru se vstupními daty. Pro operační systém `Windows` je spouštěcí příkaz téměř totožný, pouze bez počátečních symbolů `./` před názvem programu. Pokud bude aplikace spuštěna s chybným počtem či zcela bez argumentů, vypíše nápovědu pro své správné spuštění.

4.2 Průběh programu

Po spuštění programu s validním argumentem (názvem souboru obsahující vstupní data), program načte data ze souboru do příslušných dynamických polí (vektorů), následně vytvoří graf kolizí, graf obarví a výsledek algoritmu vypíše na konzoli v zestupném pořadí jako dvojice `id-vysilace␣frekvence`.

4.3 Ukázka běhu programu

```
root@LAPTOP-2QSD50MR:/mnt/c/users/danie/work/pc_final# make
gcc -c -Wall -pedantic -ansi collision_graph.c -o collision_graph.o
gcc -c -Wall -pedantic -ansi stack.c -o stack.o
gcc -c -Wall -pedantic -ansi vector.c -o vector.o
gcc -c -Wall -pedantic -ansi main.c -o main.o
gcc collision_graph.o stack.o vector.o main.o -o freq.exe -lm
```

Obrázek 4.1: Přeložení programu příkazem make

```
root@LAPTOP-2QSD50MR:/mnt/c/users/danie/work/pc_final# ./freq.exe vysilace-25.txt
0 93400
1 93400
2 93400
3 93400
4 93400
5 93400
6 93400
7 139700
8 93400
9 93400
10 104600
11 104600
12 93400
13 139700
14 104600
15 93400
16 104600
17 93400
18 139700
19 104600
20 104600
21 154600
22 139700
23 104600
24 104600
```

Obrázek 4.2: Spuštění a výstup programu

5 Závěr

Výsledný program je schopný vyřešit problém kolizí frekvencí vysílačů dle požadavků uvedených v zadání.

Při analýze a výběru datových struktur byl kladen důraz zejména na rychlost programu a to i za cenu zvýšené paměťové náročnosti. Program byl otestován na vzorových datech pro 25 a 1000 vysílačů a v obou případech vrací výsledné řešení v uspokojivých časech.

V aktuálním stavu představuje program pevný základ, který by bylo možné obohatit o podrobnější výpisy, či například o grafickou vizualizaci grafu kolizí a práce obarvovacího algoritmu.