Vysoké učení technické v Brně
Fakulta informačních technologií

ZADÁNÍ NÁHRADNÍHO PROJEKTU Z PŘEDMĚTŮ IAL

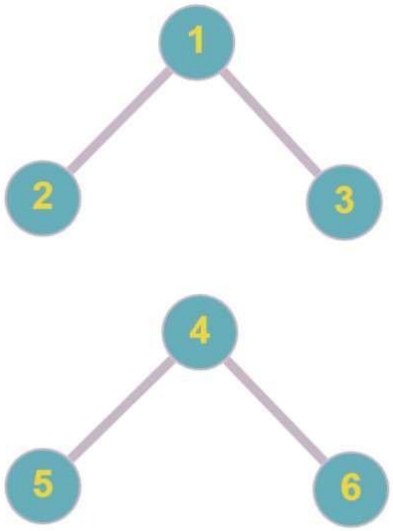
Zadání č. 3 - Vlastnosti neorientovaných grafů

Marek Gergel xgergel01
Lukáš Kysela xkysel16
Daniel Jacobs xjacob00
Jakub Pekárek xpekar19

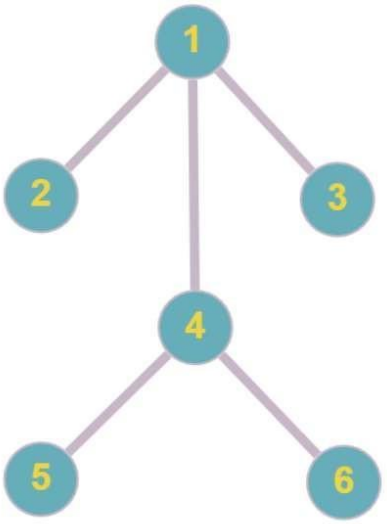
Obsah

1 Popis problematiky

Naším zadáním projektu byla konzolová aplikace, realizovaná pomocí programovacího jazyka C, která na základě vstupních dat uložených v příslušném textovém souboru vyhodnocuje, zda se jedná o data neorientovaného grafu, a následně vyhodnocuje jeho základní vlastnosti, mezi něž patří počet hran grafu, počet vrcholů, kružnice (existuje v grafu cesta, která se dostane zpět do výchozího vrcholu a hrany se na cestě neopakují), maximální stupeň vrcholu (počet hran končících ve vrcholu), souvislost (existuje cesta mezi každým vrcholem), úplnost (obsahuje všechny možné hrany) a zda je graf lesem nebo stromem. V případě, že se v grafu nenacházejí žádné kružnice, jedná se o les, pokud je navíc souvislý, jedná se o strom. Výsledky analýzy program vypisuje přehledně na výstup do konzole.



Obrázek 1 Neorientovaný graf - les



Obrázek 2 Neorientovaný graf -strom

2 Implementace

Výsledné řešení je reprezentováno několika hlavičkovými soubory ve složce include a několika C soubory ve složce src; jejich funkcionalitu rozebereme v následující kapitole.

2.1 Soubor main.c

Obsahuje základní funkci main(), která po spuštění programu volá funkci pro parsování dat ze souboru parser.c a poté funkci pro analýzu vlastností grafu. Obsahuje základní funkci main(), která po spuštění programu volá funkci pro parsování dat ze souboru parser.c a poté funkci pro analýzu vlastností grafu.

2.2 Soubor graph.c

Zajišťuje inicializaci grafové struktury, správnou alokaci paměti a po dokončení práce i správné uvolnění paměťových zdrojů. Kontroluje také, zda v zadaných datech nejsou chyby, jako jsou hrany s neexistujícími uzly nebo vícekrát definovaná jedna hrana. V případě nenalezení některého z těchto problémů je definována chybová hláška.

2.3 Soubor parser.c

Obstarává parsování vstupních dat, z kterých je tvořen zadaný graf. Nejprve parsuje data pro zadané uzly a poté pro zadané hrany. Pokud je v průběhu analýzy nalezena nějaká

syntaktická chyba, dochází k vygenerování a vypsání příslušného chybového hlášení s uvedením pozice chyby na výstup do konzole.

2.4 Soubor graph properties.c

V tomto souboru se nacházejí funkce, které zpracovávají nebo vypisují parametry grafu získané jeho analýzou. Mezi tyto parametry patří množství hran, počet uzlů, stupeň jednotlivých uzlů a další, které analyzujeme podle zadání projektu.

2.5 Soubor error.c

Obsahuje chybové funkce, které zajišťují výpis chybových hlášení na výstup, ukončení provádění programu a volání příslušných funkcí pro uklizení paměti.

3 Teoretické složitosti

V této kapitole nalezneme časové složitosti jednotlivých funkcí, pro zjištění jednotlivých vlastností grafu.

- |V| = počet vrcholů
- |E| = počet hran

  - ➔ Počet vrcholů: Θ(1)
  - ➔ Počet hran: Θ(|V|+|E|)
  - ➔ Počet cyklů: Θ(|V|+|E|)
  - ➔ Maximální stupeň vrcholu: Θ(|V|)
  - ➔ Graf je souvislý: Θ(|V|+|E|)
  - ➔ Graf je úplný: Θ(|V|)
  - ➔ Graf je strom: Θ(|V|+|E|)
  - ➔ Graf je les: Θ(|V|+|E|)

4 Spuštění programu a formát vstupních dat

Postup zkompilování projektu a následného spuštění testovacích dat uložených v podsložce projektu s názvem **testData**.

Postup:

1. Stažený zip file, ve kterém je řešení projektu, rozbalíme na našem zařízení
2. Otevřeme terminál ve složce s rozbaleným projektem.
3. Příkazem 'make' v terminálu projekt zkompilujeme.
4. Spustíme analýzu vstupních dat ze souboru
5. Formát vstupu: *./graph_properties < cesta_k_souboru_s_daty*
   - ➔ Například: *./graph_properties < testData/test1*
6. Příkazem 'make clean' smažeme zkompilovaný projekt.

Formát vstupních dat:

Máme matici o velikosti s×s, kde s je počet vrcholů grafu. Tato matice je reprezentace grafu, kde každý řádek a sloupec odpovídá jednomu vrcholu. Pokud je na pozici v 7. řádku a 3. sloupci číslo 1, znamená to, že mezi vrcholem 7 a vrcholem 3 existuje hrana. Nula v této matici znamená, že mezi odpovídajícími vrcholy hrana neexistuje.

Příklad:

```
0 1 1 1
1 0 1 1
1 1 0 1
1 1 1 0
```

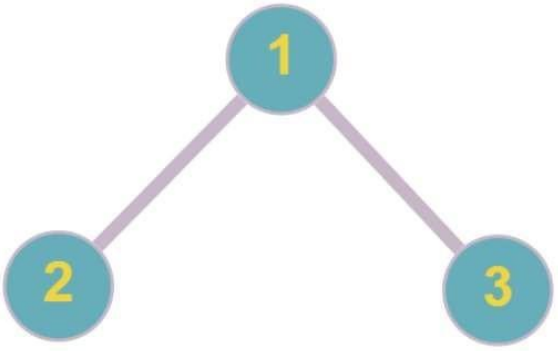
5 Testování

V této kapitole demonstrujeme funkčnost našeho řešení pro různé složité scénáře a porovnáváme, zda hodnoty vypsány naším programem odpovídají očekávaným výsledkům. Testovací data jsou uložena v podsložce 'testData' projektu a zahrnují i další testy na ověření jednotlivých vlastností, které nejsou uvedeny v této kapitole. Pro testování rozsáhlejších dat jsme si vytvořili soubory v jazyce Python, které nám generují matice podle potřeb. Tyto soubory jsou uloženy ve složce s názvem testScripts.

5.1 1. Experiment

Demonstrujeme funkčnost na jednom z nejjednodušších grafů, popisujících všechny testované vlastnosti.

5.1.1 Grafické zobrazení grafu:



Obrázek 3 - 1. Experiment

5.1.2 Příkaz pro spuštění 1. Experimentu

Příkaz do konzole: *./graph_properties < testData/test1*

5.1.3 Výstup vygenerovaný naším programem:

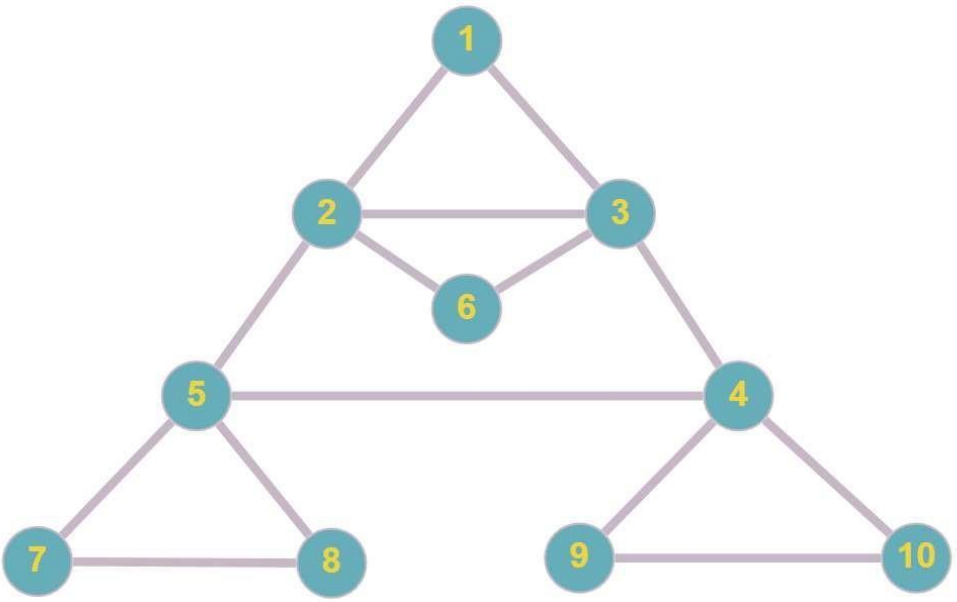
```
==================================
Node count:                      3
Edge count:                      2
Cycle count:                     0
Maximum degree:                  2
Graph is connected:              yes
Graph is complete:               no
Graph is tree:                   yes
Graph is forest                  no
==================================
```

Obrázek 4 - 1. Experiment - program output

5.2 2. Experiment

V tomto experimentu demonstrujeme funkčnost na výrazně složitějším grafu. Složitost ve srovnání s předchozím experimentem vychází z hojnosti základních smyček a z rostoucího počtu složených smyček, které musíme identifikovat.

5.2.1 Grafické zobrazení grafu:



Obrázek 5 - 2. Experiment

5.2.2 Příkaz pro spuštění 2. Experimentu

Příkaz do konzole: *./graph_properties < testData/test2*

5.2.3 Výstup vygenerovaný naším programem:

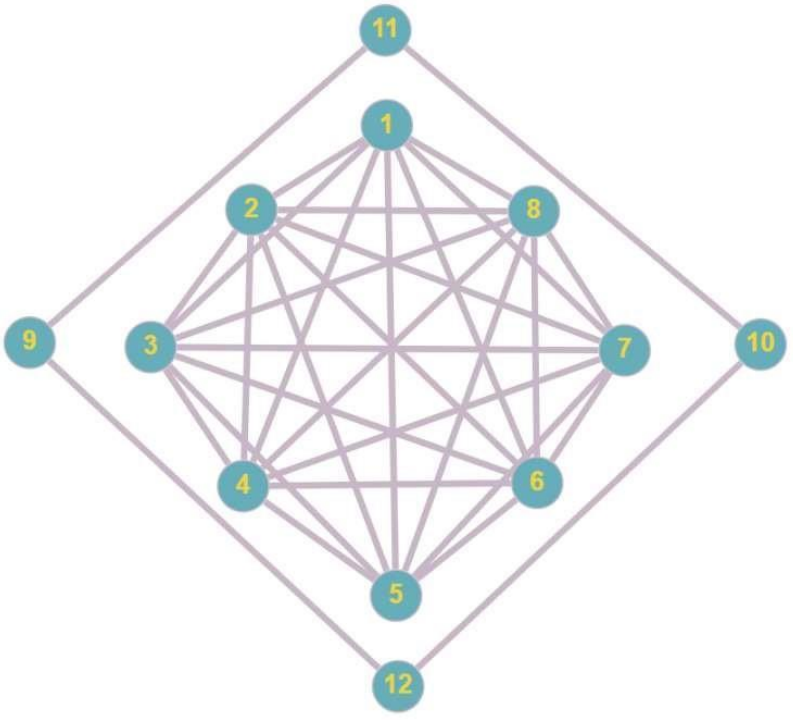
```
==================================
Node count:                      10
Edge count:                      14
Cycle count:                     8
Maximum degree:                  4
Graph is connected:              yes
Graph is complete:               no
Graph is tree:                   no
Graph is forest                  no
==================================
```

Obrázek 6 - 2. Experiment - program output

5.3 3. Experiment

V posledním experimentu se zaměříme na velmi komplexní řešení, které je zároveň nejkomplexnější ze všech zde uvedených experimentů.

5.3.1 Grafické zobrazení grafu:



Obrázek 7 - 3. Experiment

5.3.2 Příkaz pro spuštění 3. Experimentu

Příkaz do konzole: *./graph_properties < testData/test3*

5.3.3 Výstup vygenerovaný naším programem:

```
==================================
Node count:                      12
Edge count:                      32
Cycle count:                     220
Maximum degree:                  7
Graph is connected:              no
Graph is complete:               no
Graph is tree:                   no
Graph is forest                  no
==================================
```

Obrázek 8 - 3. Experiment - program output

5.4 Výsledky testování

Výsledky námi provedených testů se shodovaly s očekávanými výsledky, čímž jsme ověřili validitu našeho řešení.

6 Průběh společné spolupráce

Práce na projektu jsme zahájili ihned po získání zadání a rozdělení úkolů mezi jednotlivé členy týmu, kteří se ihned pustili do práce. Během práce na projektu byla komunikace a řešení problémů mezi týmovými členy efektivní. Jako hlavní komunikační kanál jsme zvolili Discord. Pro sdílení a verzování našeho postupu na řešení projektu jsme zvolili platformu GitHub s verzovacím nástrojem Git.

Seznam obrázků