
# IMPLEMENTACE ALGORITMU PRO NALEZENÍ NEJMENŠÍ KOSTRY GRAFU

TÝM XKULIN01

# Obsah

# 1 Úvod

Jméno souboru: IAL_kostra_grafu
Identifikace projektu: IAL, Náhradní projekt - 07. Minimální kostra grafu
Členové týmu:
Kulinkovich Andrei (xkulin01)
Marochkina Elena (xmaroc00)
Tréšek Roman (xtrese00)
Hierarchie souborů:

src - hlavní adresář

- implementace algoritmu rozdělena do několika jednotlivých souborů, pro jednotlivé části grafu

- graphs - adresář s testovými soubory

Datum vytvoření: 12.11.2023, datum poslední změny: 27.11.2023

# 2 Zadání

07 . Minimální kostra grafu - Vytvořte program pro hledání kostry grafu s minimálním ohodnocením pro ohodnocené neorientované grafy.

# 3 Návrh a implementace

K řešení jsme využili Primmova algoritmu, jeho teoretická složitost je $O(E + V * \log_2 E)$, kde E je počet hran a V je počet vrcholů v grafu.

## 3.1 Postup algoritmu

Ze začátku se náhodně vybere jeden uzel jako počáteční, zbylé uzly se zapíšou do pole nezpracovaných uzlů. Všechny hrany, které vedou z počátečního uzlu se zapíšou na seznam dostupných hran. Algoritmus projde seznam dostupných hran a vybere tu s nejnižším ohodnocením, která ale musí vést do ještě nezpracovaného uzlu. Hrana je poté odstraněna ze seznamu dostupných hran a uzel se odstraní z pole nezpracovaných uzlů. Zpracovávaná hrana se přidává do fronty hran reprezentujících postupně rostoucí kostru grafu. Uzel do kterého zpracovávaná hrana vedla se odstraní z pole nezpracovaných uzlů. Všechny hrany dostupné z nového uzlu jsou zapsány na seznam dostupných hran. Algoritmus probíhá tak dlouho, dokud nezpracuje všechny uzly.

## 3.2 Složitost implementovaného algoritmu

Složitost námi implementovaného algoritmu je $O(V^2 + E)$, protože jsme využili implementaci pomocí seznamu sousedů oproti rychlejší variantě s využitím haldy.

Implementace pomocí seznamu sousedů je pomalejší, protože prochází seznam uzlů a hledá jeho minimum.

# 4 Datové struktury

## 4.1 Seznam

Implementovali jsme jednosměrně vázaný seznam, pro podporu následující datové struktury.

### 4.1.1 Hrana

Máme 2 typy hran, zpracované hrany a dostupné hrany. Pro obojí jsme využili našeho jednoduchého seznamu. S tím, že dostupné hrany zároveň v seznamu vytváří frontu.

## 4.2 Pole

Vytvořili jsme jednoduché pole pro podporu implementace uzlů. Při vytváření tohoto pole mu předáváme následující parametry: celkovou velikost (počet uzlů v grafu), aktuální pole (na začátku prázdné).

### 4.2.1 Uzel

Po prvním průchodu grafem, uložíme všechny uzly do pole. Poté jsou uzly z pole postupně oddělávány při jejich zpracování.

# 5 Členění implementačního řešení

| | |
|---|---|
| Algoritmus | `algorithm. {c, h}` |
| Hrana | `edge. {c, h}` |
| Seznam | `list. {c, h}` |
| Uzel | `node. {c, h}` |
| Čtečka | `reader. {c, h}` |
| Pole | `set. {c, h}` |
| Funkce main | `main. c` |

# 6 Práce v týmu

| Člen týmu | Přidělená práce |
|---|---|
| Andrei Kulinkovich | Implementace algoritmu, vytvoření struktur, dokumentace |
| Elena Marochkina | Implementace algoritmu, testování |
| Roman Trébšek | Implementace algoritmu, dokumentace, prezentace |

## 7 Pole

```
typedef struct {
    char *elements;
    size_t size;
    size_t capacity;
} CharSet;
```

## 8 Seznam

```
typedef struct {
    listElementPtr firstElement;
    listElementPtr activeElement;
    listElementPtr lastElement;
} list;
```