

# Teoretická informatika

Obor C, 3. ročník

David Weber

SPŠE JEČNÁ

*Poslední aktualizace: 23. července 2023*

# Obsah

<b>Předmluva</b>	<b>2</b>
<b>1 Grafové algoritmy</b>	<b>3</b>
1.1 Grafy a jejich reprezentace . . . . .	3
1.2 Stromy . . . . .	3
1.3 Prohledávání do šířky . . . . .	3
1.4 Prohledávání do hloubky . . . . .	5
1.5 Dijkstrův algoritmus . . . . .	6
1.6 Algoritmus A* . . . . .	6
<b>2 Dynamické programování</b>	<b>7</b>

# Předmluva

# Kapitola 1

## Grafové algoritmy

### 1.1 Grafy a jejich reprezentace

**Definice 1.1.1** (Graf). Grafem  $G$  nazveme uspořádanou dvojici  $(V, E)$ , kde  $V$  je množina *vrcholů* (nebo také *uzlů*) a  $E$  množina *hran*, přičemž pokud

- $E \subseteq \{\{u, v\} \mid u, v \in V\}$ , pak  $G$  nazýváme *neorientovaným* grafem (tj. po hraně lze pohybovat v obou směrech).
- $E \subseteq \{(u, v) \mid u, v \in V\}$ , pak  $G$  nazýváme *orientovaným* grafem (tj. po hranách se lze pohybovat pouze v jednom směru).

### 1.2 Stromy

### 1.3 Prohledávání do šířky

Jednou ze základních úloh je procházení grafu z určitého vrcholu a zjištění dosažitelnosti ostatních vrcholů. Nejjednodušším algoritmem v tomto ohledu je tzv. *prohledávání do šířky* (angl. *breadth-first search*, zkráceně BFS). Jeho základní princip spočívá v postupném objevování následníků již nalezených vrcholů. Na počátku dostaneme graf  $G = (V, E)$  a nějaký počáteční vrchol  $v_0 \in V$ . Postupně objevíme všechny sousedy vrcholu  $v_0$ , poté všechny sousedy těchto nalezených sousedů, atd. Na BFS lze nahlížet tak, že do počátečního vrcholu nalijeme vodu a sledujeme, jak postupuje vzniklá vlna.

Pro každý vrchol si budeme uchovávat jeho *stav*.

- *Nenalezený* – vrchol jsme ještě během výpočtu neviděli.
- *Otevřený* – vrchol jsme viděli, ale ještě nejsme neprozkoumali všechny jeho sousedy.
- *Uzavřený* – vrchol jsme prozkoumali společně se všemi jeho sousedy a dál se jím již netřeba zabývat.

Na počátku začneme s jedním otevřeným vrcholem a to  $v_0$  (zde začínáme). Po prozkoumání všech sousedních vrcholů se jejich stav změní na uzavřený a počáteční vrchol  $v_0$  se uzavře. Obdobně pokračujeme pro nově otevřené vrcholy. Pokud by náhodou mezi dvojicí otevřených vrcholů existovala hrana, pak si sousedního vrcholu všimnat nebudeme, neboť byl již otevřen. Pro každý vrchol se ještě dodatečně můžeme uchovávat informaci, jak daleko se nachází od  $v_0$ , co do počtu hran ležících na cestě.

**Algoritmus 1.3.1** (BFS)

*Vstup:* Graf  $G = (V, E)$  a počáteční vrchol  $v_0 \in V$ .

*// Inicializace*

**Pro** každý vrchol  $v \in V$  **opakuj:**

$stav(v) \leftarrow \text{nenalezený}$

$D(v) \leftarrow \infty$

$stav(v_0) \leftarrow \text{otevřený}$

$D(v_0) \leftarrow 0$

Založ frontu  $Q$  a přidej do ní vrchol  $v_0$

**Dokud** je fronta  $Q$  neprázdná, **opakuj:**

$v \leftarrow$  první vrchol ve frontě  $Q$ , který z ní odebereme

*// Prozkoumáváme všechny dosud neobjevené sousedy*

**Pro** každý sousední vrchol  $w$  vrcholu  $v$  **opakuj:**

**Pokud**  $stav(w) = \text{nenalezený}$ , **proved:**

$stav(w) \leftarrow \text{otevřený}$

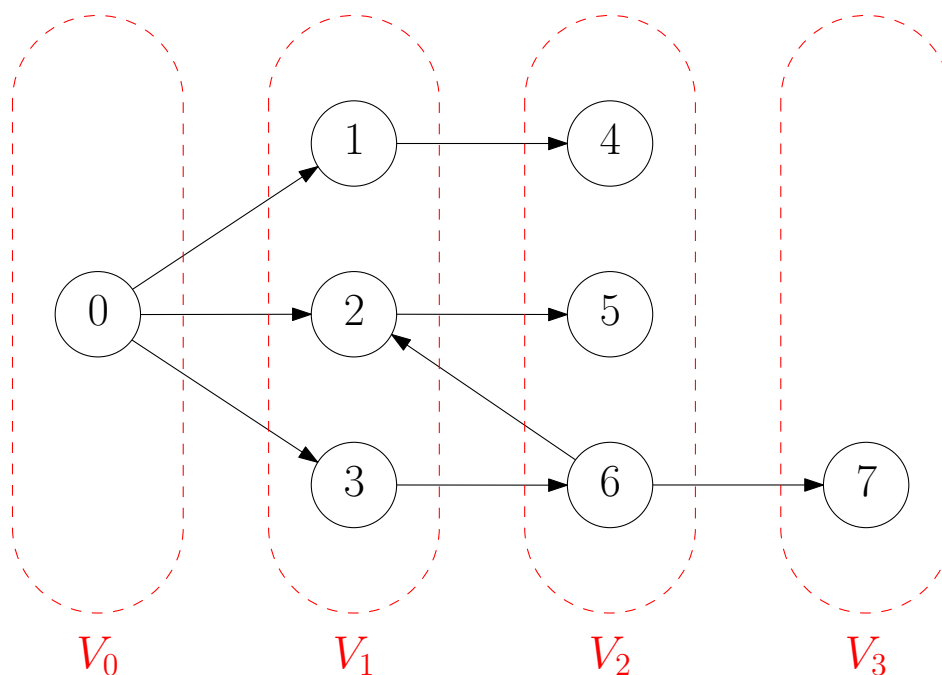
$D(w) \leftarrow D(v) + 1$

Přidej  $w$  do fronty  $Q$

$stav(v) \leftarrow \text{uzavřený}$

*Výstup:* Seznam vzdáleností  $D$ .

BFS rozděluje vrcholy do vrstev podle toho, v jaké vzdálenosti od počátečního vrcholu se nachází (viz obrázek 1.1). Nejdříve jsou prozkoumány vrcholy ve vzdálenosti 0 (tj. pouze  $v_0$ ), poté ve vzdálenostech 1, 2, ... Z toho vyplývá, že kdykoliv při otevírání libovolného vrcholu  $v$  nastavujeme hodnotu  $D(v)$ , bude tato hodnota vždy odpovídat délce nejkratší cesty z  $v_0$  do  $v$ . Tato hodnota bude však nastavena pouze u těch vrcholů, které jsou z  $v_0$  dosažitelné (ostatní vrcholy zůstanou ve stavu nenalezený).



Obrázek 1.1: Vrcholy rozdělené do vrstev podle průběhu BFS.

Zbývá prozkoumat časovou a paměťovou složitost BFS. Označme si počet vrcholů grafu  $G$  na vstupu  $n$  a počet jeho hran  $m$ .

**Věta 1.3.2** (Složitost BFS). *Algoritmus BFS doběhne v čase  $\mathcal{O}(n + m)$  a spotřebuje paměť  $\mathcal{O}(n + m)$ .*

*Důkaz.* Inicializace potrvá  $\mathcal{O}(n)$ , neboť cyklus iteruje přes všechny vrcholy. Vnější cyklus provede maximálně  $n$  iterací, protože každý z vrcholů uzavřeme nejvýše jednou, tj.  $\mathcal{O}(n)$ .

S vnitřním cyklem je to trochu složitější, protože jeho počet iterací závisí na tom, který z vrcholů otevíráme (resp. na počtu jeho sousedů). To znamená, že pokud si označíme  $d_i$  počet sousedů vrcholu  $i$ , pak celkový počet iterací vnitřního cyklu přes všechny vrcholy bude  $\sum_i d_i$ . Lze si ovšem všimnout jedné užitečné věci. Pokaždé, když prozkoumáváme sousední vrchol  $w$  nějakého vrcholu  $v$ , mohou nastat dva případy podle toho, jestli je  $G$  orientovaný graf, nebo neorientovaný.

- (i) Graf  $G$  je neorientovaný. Pak hranu, která spojuje  $v$  a  $w$  prozkoumáme právě *dvakrát* (jednou z vrcholu  $v$  a podruhé z vrcholu  $w$ ). Každá hrana se tak započítá dvakrát, tzn. vnitřní cyklus se celkově provede max  $2m$ -krát (po všech iteracích vnějšího cyklu).
- (ii) Graf  $G$  je orientovaný. Pak se hrana započítá pouze jednou, a to z vrcholu, z něhož vede. Celkově se vnitřní cyklus provede  $m$ -krát.

V prvním případě tak pro časovou složitost bude platit

$$\mathcal{O}\left(n + \sum_i d_i\right) = \mathcal{O}(n + 2m) = \mathcal{O}(n + m).$$

V druhém případě dojdeme ke stejnému výsledku, akorát zmíněná suma bude, kvůli orientaci hran, rovna přesně počtu hran, tj.

$$\mathcal{O}\left(n + \sum_i d_i\right) = \mathcal{O}(n + m).$$

Nyní k prostorové složitosti algoritmu. Na reprezentaci grafu (např. pomocí seznamu sousedů) spotřebujeme paměť  $\mathcal{O}(n + m)$  ( $n$  vrcholů,  $m$  hran)<sup>1</sup>. Dále si uchováváme vrcholy ve frontě, v níž může být v jednu chvíli maximálně  $n$  vrcholů, tj.  $\mathcal{O}(n)$ , a k tomu máme seznam *stav*, kde je uloženo  $\mathcal{O}(n)$  vrcholů. Celkově spotřebujeme paměti

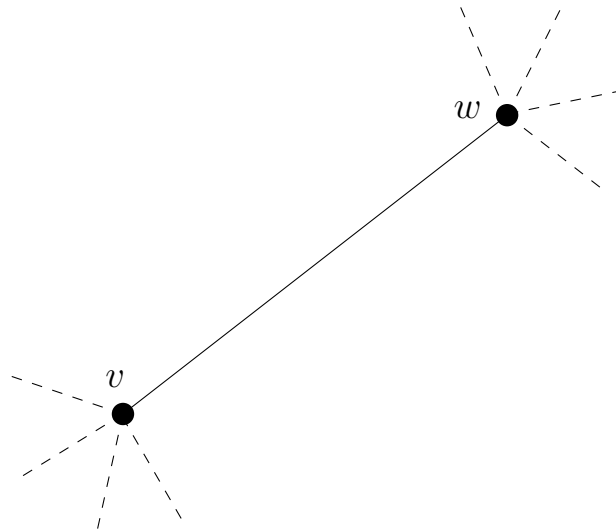
$$\mathcal{O}(n + m) + \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n + m).$$

□

## 1.4 Prohledávání do hloubky

Na podobném přístupu, jako BFS, je založeno tzv. *prohledávání do hloubky* (anglicky *depth-first search*, zkráceně DFS). Vrcholy však tentokrát budeme zpracovávat rekurzivně. Pokaždé, když budeme otevírat nový vrchol, se rekurzivně zavoláme všechny jeho sousední vrcholy, u nichž opakujeme stejnou proceduru. Po prozkoumání všech sousedů daný vrchol uzavřeme. Stejně jako BFS si tak budeme uchovávat pole *stavů* pro jednotlivé vrcholy.

<sup>1</sup>Resp. každá hrana je v neorientovaných grafech započítána dvakrát, protože každý vrchol obsahuje v seznamu svých sousedů vždy „protějščí“ vrchol (stejný argument, jako u odvozování časové složitosti BFS).



Obrázek 1.2: Znárodnění situace v důkazu části (i) věty 1.3.2.

**Algoritmus 1.4.1** (DFS)

*Vstup:* Graf  $G = (V, E)$  a počáteční vrchol  $v_0 \in V$ .

// Inicializace

**Pro** každý vrchol  $v \in V$  **opakuj:**

$stav(v) \leftarrow nenalezený$

Zavolej DFS2( $v_0$ )

// Rekursivní volání na vrcholy

**Funkce** DFS2 (vrchol  $v$ )

$stav(v) \leftarrow otevřený$

**Pro** každý sousední vrchol  $w$  vrcholu  $v$  **opakuj:**

**Pokud**  $stav(w) = nenalezený$ , **proved:**

Zavolej DFS2( $w$ )

$stav(v) \leftarrow uzavřený$

**Věta 1.4.2** (Složitost DFS). *Algoritmus DFS doběhne v čase  $\mathcal{O}(n + m)$  a spotřebuje paměť  $\mathcal{O}(n + m)$ .*

*Důkaz.* Algoritmus DFS se oproti BFS liší pouze v pořadí, v jakém pořadí dosažitelné vrcholy zpracovává, to však nemá na časovou složitost žádný vliv. Argument pro její odvození je tak stejný jako u BFS, viz věta 1.3.2 v minulé sekci.

V paměti si musíme uchovávat reprezentaci grafu, to zabere  $\mathcal{O}(n + m)$  paměti (opět např. pomocí seznamu sousedů) a máme seznam  $stav$  s  $n$  prvky (vrcholy). Zároveň si při volání DFS2 musíme na zásobník rekurze ukládat jednotlivé *aktivační záznamy*. Protože vrcholů je v grafu  $n$ , pak na zásobníku rekurze bude v jednu chvíli maximálně  $n$  záznamů, tj.  $\mathcal{O}(n)$ . Celkově spotřebujeme  $\mathcal{O}(n + m) + \mathcal{O}(n) + \mathcal{O}(n) = \mathcal{O}(n + m)$  paměti.  $\square$

## 1.5 Dijkstrův algoritmus

## 1.6 Algoritmus A\*

## Kapitola 2

# Dynamické programování