Prohledávání grafu do hloubky (DFS)

Metodický materiál pro výuku algoritmizace

Materiál pro učitelství informatiky

17. února 2026

Co je prohledávání do hloubky (DFS)?

- Grafový algoritmus, který se vydává podél jedné cesty co nejhlouběji to jde, než se začne vracet.
- **Základní myšlenka:** Jdi stále vpřed na dalšího nenavštíveného souseda. Pokud narazíš na slepou uličku (vrchol bez nenavštívených sousedů), vrať se o krok zpět (backtracking) a zkus jinou cestu.
- **Analogie pro žáky:** Průchod bludištěm (pravidlo pravé ruky), nebo prohledávání rodokmenu co nejhlouběji v jedné rodové linii před návratem ke strýcům a tetám.
- **Klíčová datová struktura:** Pro správné fungování DFS využíváme **Zásobník** (Stack) – nejčastěji implicitně pomocí rekurze.

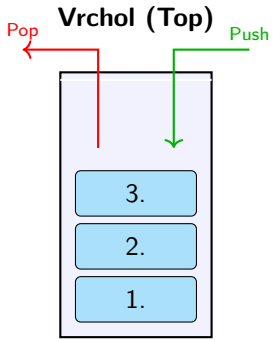
Nezbytná prerekvizita: Datová struktura Zásobník

Princip LIFO (Last In, First Out)

- „Poslední dovnitř, první ven.“
- Jako sloupec talířů v jídelně nebo tlačítko „Zpět“ (Undo) v textovém editoru nebo zásobník papírů v tiskárně.

Základní operace:

- `Push(x)`: Položí prvek na vrchol zásobníku.
- `Pop()`: Odebere a vrátí prvek z vrcholu.



Pseudokód algoritmu DFS (Rekurzivní verze)

Poznámka pro žáky: Rekurze automaticky využívá systémový zásobník volání (Call Stack).

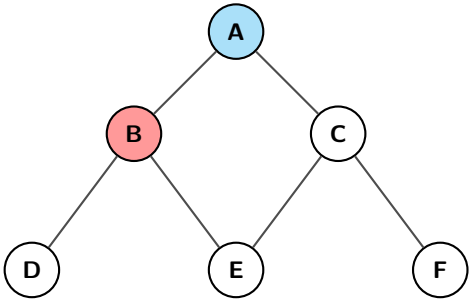
Algorithm 1 DFS(graf, start)

1: Označ všechny vrcholy jako **nenavštívené**
2: DFS_VISIT(*start*)

Algorithm 2 DFS_Visit(v)

1: Označ *v* jako **zpracovávaný** (vstoupili jsme do něj)
2: **for** každého souseda *u* vrcholu *v* **do**
3: **if** *u* **není** navštívený **then**
4: DFS_VISIT(*u*) ▷ Zanoříme se hlouběji (Push na zásobník)
5: **end if**
6: **end for**
7: Označ *v* jako **kompletně zpracovaný** ▷ Vracíme se (Pop ze zásobníku)

Krokování DFS - Ponor do hloubky



Legenda:

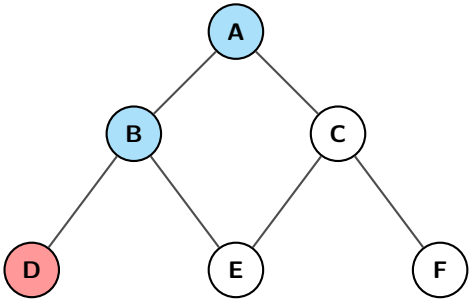
- Nenavštívený
- Zpracovávaný (v zásobníku)
- Právě volaný (vrchol zásobníku)
- Kompletně hotový

Zásobník volání: `[ A, B ]`

Postup (abecední pořadí sousedů):

1. Začneme v **A**, zavoláme souseda **B**.

Krokování DFS - Ponor do hloubky



Legenda:

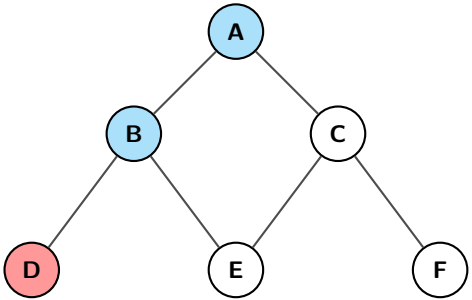
- Nenavštívený
- Zpracovávaný (v zásobníku)
- Právě volaný (vrchol zásobníku)
- Kompletně hotový

Zásobník volání: `[ A, B, D ]`

Postup (abecední pořadí sousedů):

1. Začneme v **A**, zavoláme souseda **B**.
2. Z vrcholu **B** voláme souseda **D**.

Krokování DFS - Ponor do hloubky



Legenda:

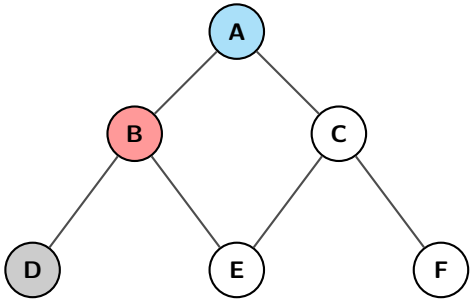
- Nenavštívený
- Zpracovávaný (v zásobníku)
- Právě volaný (vrchol zásobníku)
- Kompletně hotový

Zásobník volání: `[ A, B, D ]`

Postup (abecední pořadí sousedů):

1. Začneme v **A**, zavoláme souseda **B**.
2. Z vrcholu **B** voláme souseda **D**.
3. Jsme v **D**. D nemá žádné další nenavštívené sousedy (B už je v zásobníku).

Krokování DFS - Ponor do hloubky



Legenda:

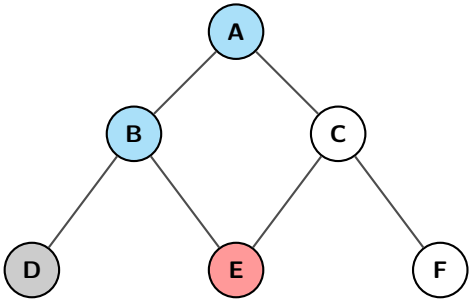
- Nenavštívený
- Zpracovávaný (v zásobníku)
- Právě volaný (vrchol zásobníku)
- Kompletně hotový

Zásobník volání: `[ A, B ]`

Postup (abecední pořadí sousedů):

1. Začneme v **A**, zavoláme souseda **B**.
2. Z vrcholu **B** voláme souseda **D**.
3. Jsme v **D**. D nemá žádné další nenavštívené sousedy (B už je v zásobníku).
4. Tím **D** končí a my se vracíme (backtracking).

Krokování DFS - Návrat a nová cesta

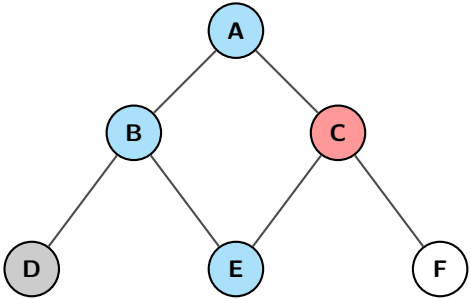


Zásobník volání: `[ A, B, E ]`

Postup:

1. Z D jsme se vrátili do **B**. B má dalšího souseda: **E**.

Krokování DFS - Návrat a nová cesta

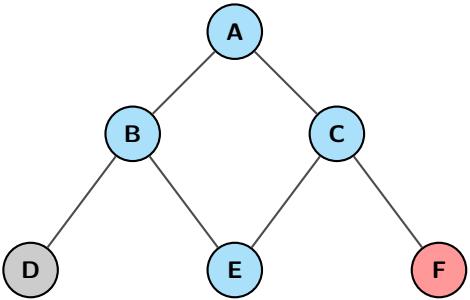


Zásobník volání: `[ A, B, E, C ]`

Postup:

1. Z D jsme se vrátili do **B**. B má dalšího souseda: **E**.
2. Z **E** jdeme do **C** (B je již navštívený).

Krokování DFS - Návrat a nová cesta

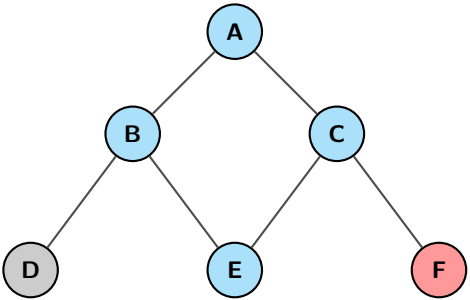


Zásobník volání: `[ A, B, E, C, F ]`

Postup:

1. Z D jsme se vrátili do **B**. B má dalšího souseda: **E**.
2. Z **E** jdeme do **C** (B je již navštívený).
3. Z **C** jdeme do **F** (A i E už v zásobníku máme).

Krokování DFS - Návrat a nová cesta

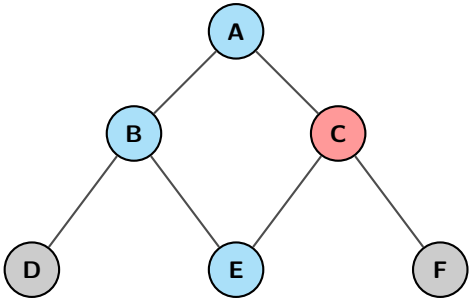


Zásobník volání: `[ A, B, E, C, F ]`

Postup:

1. Z D jsme se vrátili do **B**. B má dalšího souseda: **E**.
2. Z **E** jdeme do **C** (B je již navštívený).
3. Z **C** jdeme do **F** (A i E už v zásobníku máme).
4. Jsme ve **F**. Soused C už je v zásobníku → slepá ulička. Začíná dlouhý návrat.

Krokování DFS - Backtracking (Návrat)

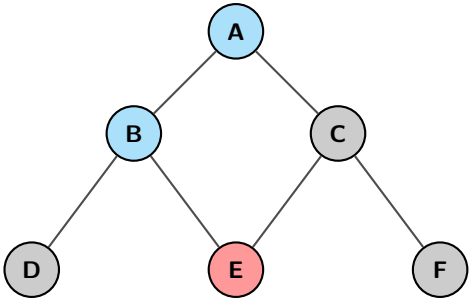


Zásobník volání: `[ A, B, E, C ]`

Postup:

1. **F** skončilo → návrat do C.

Krokování DFS - Backtracking (Návrat)

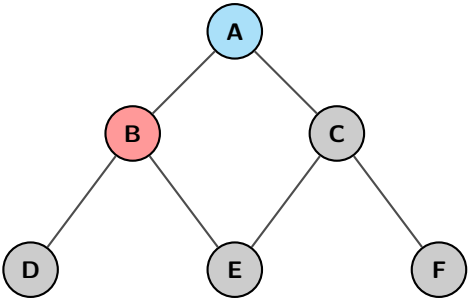


Zásobník volání: `[ A, B, E ]`

Postup:

1. **F** skončilo → návrat do C.
2. **C** už nemá nenavštívené sousedy → návrat do E.

Krokování DFS - Backtracking (Návrat)

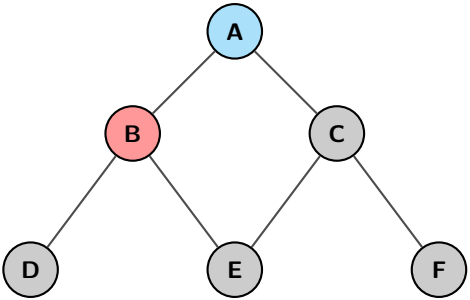


Zásobník volání: `[ A, B ]`

Postup:

1. **F** skončilo → návrat do C.
2. **C** už nemá nenavštívené sousedy → návrat do E.
3. **E** už nemá nenavštívené sousedy → návrat do **B**.

Krokování DFS - Backtracking (Návrat)

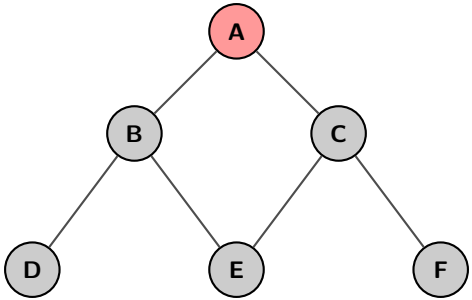


Zásobník volání: `[ A, B ]`

Postup:

1. **F** skončilo → návrat do C.
2. **C** už nemá nenavštívené sousedy → návrat do E.
3. **E** už nemá nenavštívené sousedy → návrat do **B**.
4. V zásobníku tak zůstávají jen A a B, čekající na dokončení.

Krokování DFS - Konec algoritmu

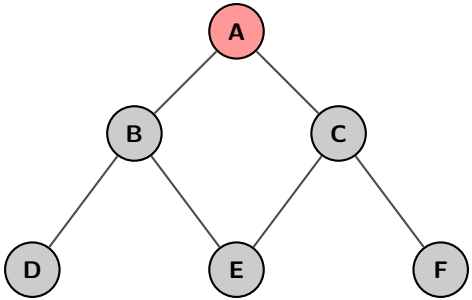


Zásobník volání: [A]

Postup:

1. Z vrcholu B se vrátíme do **A**.

Krokování DFS - Konec algoritmu

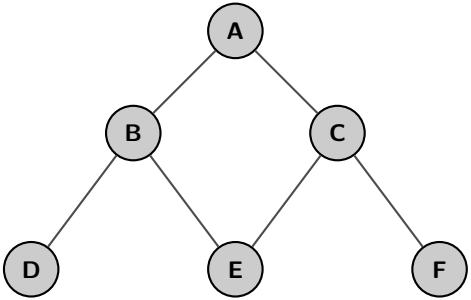


Zásobník volání: [A]

Postup:

1. Z vrcholu B se vrátíme do **A**.
2. Zkontrolujeme sousedy A (B, C) - oba jsou již kompletně zpracováni.

Krokování DFS - Konec algoritmu

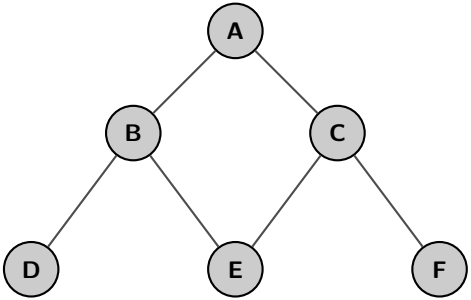


Zásobník volání: [ ] (Prázdný)

Postup:

1. Z vrcholu B se vrátíme do **A**.
2. Zkontrolujeme sousedy A (B, C) - oba jsou již kompletně zpracováni.
3. Vrchol **A** je prohlášen za kompletní.

Krokování DFS - Konec algoritmu



Zásobník volání: [ ] (Prázdný)

Postup:

1. Z vrcholu B se vrátíme do **A**.
2. Zkontrolujeme sousedy A (B, C) - oba jsou již kompletně zpracováni.
3. Vrchol **A** je prohlášen za kompletní.
4. Zásobník se vyprázdnil, algoritmus končí.

Časová a prostorová složitost

Časová složitost: $\mathcal{O}(|V| + |E|)$

- Kde $|V|$ je počet vrcholů a $|E|$ je počet hran.
- Každý vrchol je do zásobníku vložen maximálně jednou (pouze pokud nebyl dříve navštíven), což odpovídá dílčí složitosti $\mathcal{O}(|V|)$.
- Každá hrana je v průběhu algoritmu prozkoumána právě dvakrát (detaily najde čtenář v prezentaci o časové složitosti a vnořených cyklech), což přispívá složitostí $\mathcal{O}(|E|)$.
- Při určování asymptotické složitosti multiplikativní konstanty (např. faktor 2 u hran) zanedbáváme. Zásadní je pro nás pouze to, jak celkový počet operací roste v závislosti na zvětšujícím se vstupu.
- **Srovnání s BFS:** Časová složitost je naprosto **stejná** jako u BFS. Rozdíl je pouze v *pořadí*, v jakém vrcholy objevujeme.

Prostorová složitost: $\mathcal{O}(|V|)$

- Je dána hloubkou rekurze (velikostí zásobníku). V nejhorším případě (např. graf ve tvaru dlouhé přímky) se do zásobníku uloží všechny vrcholy za sebou.

Typické příklady použití DFS

V jakých situacích by měli žáci naopak upřednostnit DFS před BFS?

1. **Hledání cyklů v grafu:**
   - Narazí-li DFS během prohledávání na vrchol, který už je v zásobníku (na tzv. „zpětnou hranu“), našli jsme cyklus.
2. **Topologické třídění:**
   - Řazení úkolů, které na sobě závisí (např. prerekvizity předmětů na univerzitě). K tomu využijeme pořadí, v jakém vrcholy „kompletně opouštíme“ (šedé zbarvení).
3. **Generování a řešení bludišť / Sudoku:**
   - Ideální pro tzv. *Backtracking* algoritmy. Uděláme tah, jdeme do hloubky, a pokud to nevyjde, krok vrátíme.