

Stromy

KIV/ADT – 8. přednáška

Miloslav Konopík, Libor Váša

12. dubna 2024

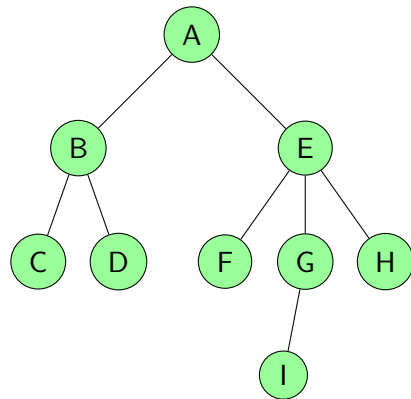
- 1 Stromy
- 2 ADT Strom
- 3 Procházení stromů
- 4 Nejbližší společný předek

Stromy

Definice

Strom je datová struktura, která se skládá z uzlů a hran.

- podchycuje **vztahy** mezi prvky
- struktura s hierarchií
- větvení pouze jedním směrem



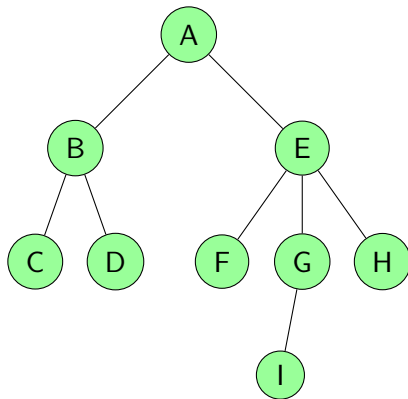
Definice

Strom je datová struktura, která se skládá z uzlů a hran.

- podchycuje **vztahy** mezi prvky
- struktura s hierarchií
- větvení pouze jedním směrem

Příklady:

- rodokmen (osoba a potomci, popř. osoba a předci),
- taxonomie (např. v biologii),
- struktura vedení ve firmě (nadřízený/podřízený),
- analýza věty,
- aritmetický výraz,
- adresářová struktura,
- ...



Vrchol (Node)


- prvek struktury,
- může obsahovat další data,
- má žádného nebo jednoho předka,
- má žádného, jednoho nebo více potomků.

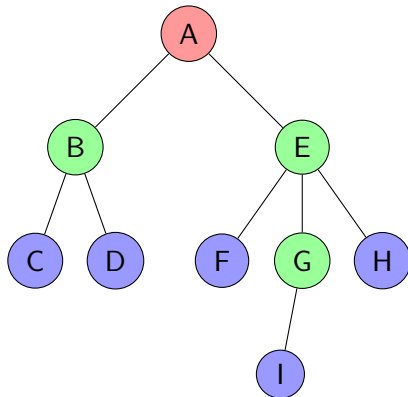
/ Hrana (Edge)

- vztah (spojení) dvou vrcholů (předek/potomek, nikdy mezi „sourozenci“)

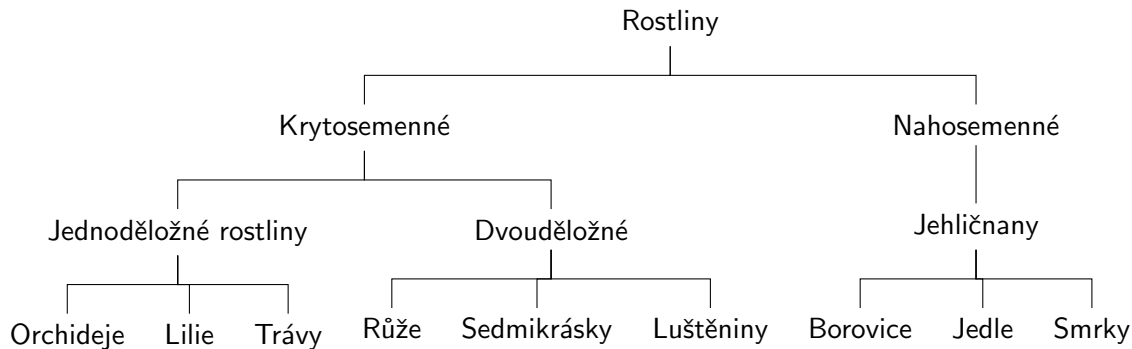
 List: žádní potomci.

 Vnitřní vrchol: alespoň jeden potomek.

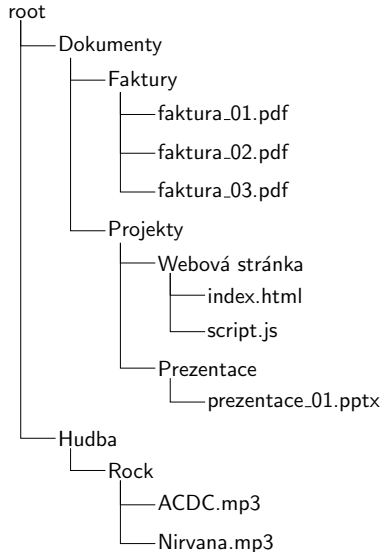
 Kořen: předek všech vrcholů, nemá žádného přímého předka.



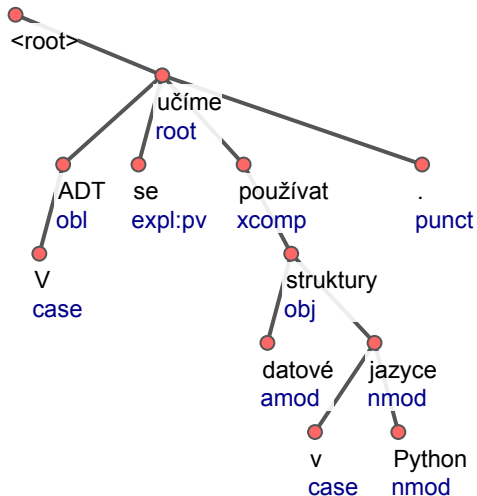
Příklad stromu – taxonomie



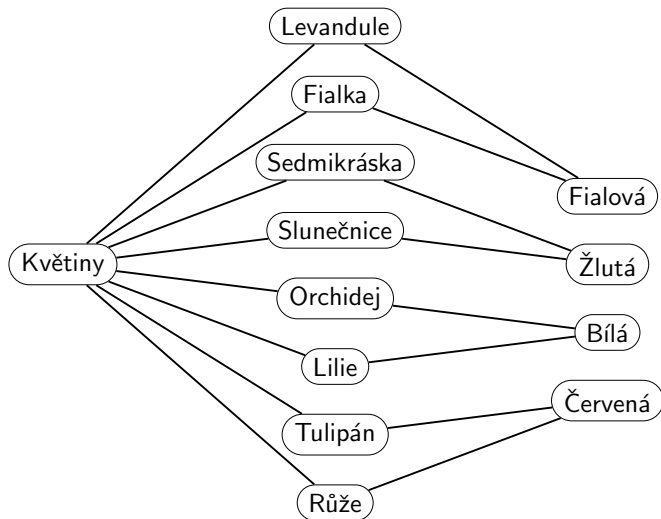
Příklad stromu – adresářová struktura



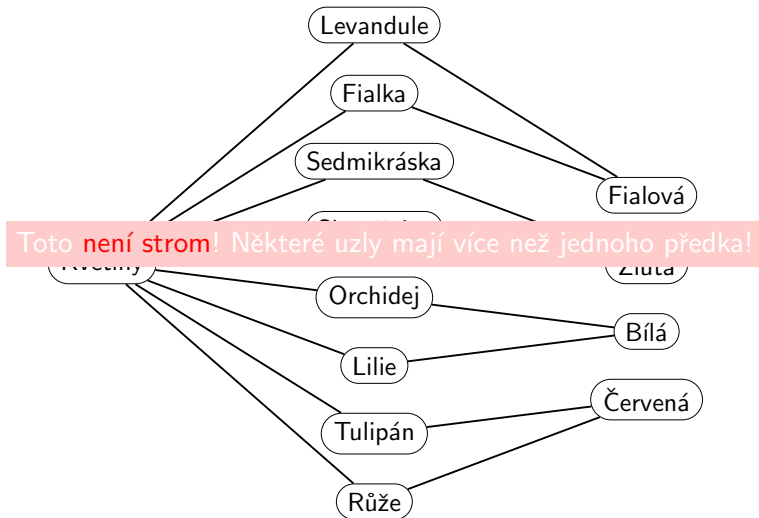
Příklad stromu – Analýza věty



Příklad – vlastnosti květin



Příklad – vlastnosti květin



Pravidla:

- 1 Prázdná množina vrcholů je strom. Jeho kořen je prázdná množina.

Pravidla:

- 1 Prázdná množina vrcholů je strom. Jeho kořen je prázdná množina.
- 2 Jediný vrchol x je strom. Jeho kořen je x .

Pravidla:

- 1 Prázdná množina vrcholů je strom. Jeho kořen je prázdná množina.
- 2 Jediný vrchol x je strom. Jeho kořen je x .
- 3 Je-li x vrchol a T_1, T_2, \dots, T_n stromy s kořeny r_1, r_2, \dots, r_n , pak x propojený hranou s každým kořenem r_1, \dots, r_n je strom a jeho kořen je x .

Pravidla:

- 1 Prázdná množina vrcholů je strom. Jeho kořen je prázdná množina.
- 2 Jediný vrchol x je strom. Jeho kořen je x .
- 3 Je-li x vrchol a T_1, T_2, \dots, T_n stromy s kořeny r_1, r_2, \dots, r_n , pak x propojený hranou s každým kořenem r_1, \dots, r_n je strom a jeho kořen je x .

Poznámky:

- ve stromě **neexistují cykly** (uzavřené posloupnosti hran)
- když neexistují cykly, pak lze strukturu popsat jako strom („zavěsit“ za nějaký kořen)

Cesta

- posloupnost vrcholů, ve které jsou každé dva po sobě následující vrcholy spojeny hranou,
- ke každému vrcholu vede z kořene právě jedna cesta.

Cesta

- posloupnost vrcholů, ve které jsou každé dva po sobě následující vrcholy spojeny hranou,
- ke každému vrcholu vede z kořene právě jedna cesta.

Délka cesty

- počet hran cesty.

Cesta

- posloupnost vrcholů, ve které jsou každé dva po sobě následující vrcholy spojeny hranou,
- ke každému vrcholu vede z kořene právě jedna cesta.

Délka cesty

- počet hran cesty.

Hloubka vrcholu

- délka cesty z kořene do vrcholu,
- hloubka kořene = 0.

Cesta

- posloupnost vrcholů, ve které jsou každé dva po sobě následující vrcholy spojeny hranou,
- ke každému vrcholu vede z kořene právě jedna cesta.

Délka cesty

- počet hran cesty.

Hloubka vrcholu

- délka cesty z kořene do vrcholu,
- hloubka kořene = 0.

Výška (hloubka) stromu

- maximální hloubka vrcholu ve stromě.

Uspořádaný strom

Potomci vrcholu mají jednoznačně definované, neměnné pořadí.

- Např. děti v rodokmenu seřazené podle věku,
- slova ve větě,
- rodiče: matka první, otec druhý.

Uspořádaný strom

Potomci vrcholu mají jednoznačně definované, neměnné pořadí.

- Např. děti v rodokmenu seřazené podle věku,
- slova ve větě,
- rodiče: matka první, otec druhý.

Neuspořádaný strom

Potomci tvoří množinu v matematickém smyslu.

- Např. podřízení nadřízeného v hierarchii firmy

Vliv na význam nebo složitost algoritmů (zejména vkládání a vyhledávání).



ADT Strom

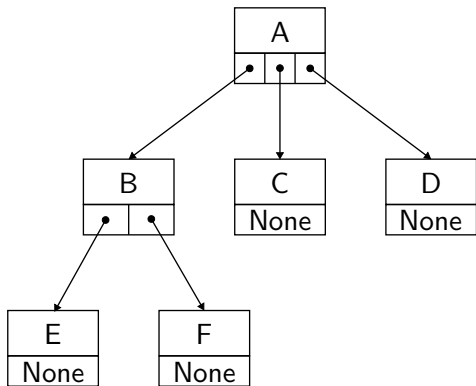
Operace:

- vybrání kořene stromu
- přidání potomka danému vrcholu
- vybrání předka daného vrcholu
- vybrání potomků daného vrcholu
- odebrání daného vrcholu ze stromu (spolu se všemi potomky vrcholu)

Operace se můžou lišit podle příchutě stromu (binární, uspořádaný, ...)

ADT Strom v Pythonu

```
1 class TreeNode:
2     """Třída reprezentující vrchol ve stromu."""
3     def __init__(self, data) -> None:
4         """Inicializuje vrchol s daty a prázdným seznamem potomků."""
5         self.data = data
6         self.children: list["TreeNode"] = []
7
8     def add_child(self, child: "TreeNode") -> None:
9         """Přidá potomka k vrcholu."""
10        self.children.append(child)
11
12    def remove_child(self, child: "TreeNode") -> None:
13        """Odebere potomka z vrcholu."""
14        self.children.remove(child)
15
16    def __str__(self) -> str:
17        """Vrátí řetězcovou reprezentaci vrcholu."""
18        return f"TreeNode({self.data})"
```



Procházení stromů

Procházení do hloubky (DFS)

- postupně procházíme strom do hloubky,
- každý vrchol ihned zpracujeme.

Procházení do hloubky (DFS)

- postupně procházíme strom do hloubky,
- každý vrchol ihned zpracujeme.

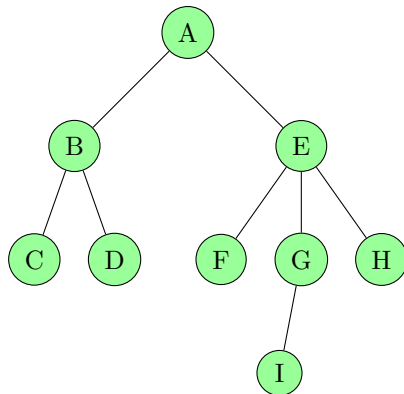
Procházení do šířky (BFS)

- postupně procházíme strom do šířky,
- každý vrchol vložíme nakonec fronty a zpracujeme první, co je na řadě.

```
1 class TreeNode:
2     ...
3     def print(self, ident=0):
4         """Vytiskne vrchol a jeho potomky rekurzivně."""
5         print("  " * ident + str(self))
6         for child in self.children:
7             child.print(ident + 1)
```

Procházení do hloubky – ukázka výstupu

A
B
C
D
E
F
G
H
I

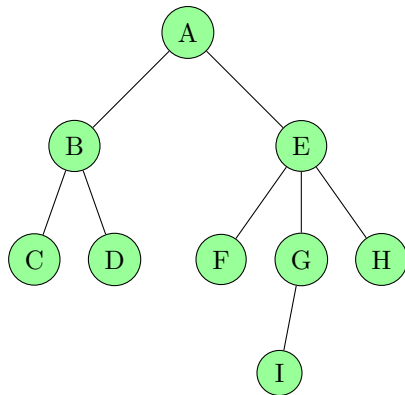


Procházení do šířky – algoritmus

```
1  from collections import deque
2  class TreeNode:
3      ...
4      def print_levels(self):
5          """Vytiskne vrcholy stromu na jednotlivých úrovních."""
6          queue = deque([self])
7          while queue:
8              level = []
9              for _ in range(len(queue)):
10                 node = queue.popleft()
11                 level.append(node.data)
12                 queue.extend(node.children)
13             print(level)
```


Procházení do šířky – ukázka výstupu

```
['A']  
['B', 'E']  
['C', 'D', 'F', 'G', 'H']  
['I']
```



Nejbližší společný předek

Nejbližší společný předek

- nalezení nejbližšího společného předka dvou vrcholů ve stromu,
- obecně pro libovolný strom.

Nejbližší společný předek

- nalezení nejbližšího společného předka dvou vrcholů ve stromu,
- obecně pro libovolný strom.

Algoritmus

- 1 Nalezení cesty z kořene do prvního vrcholu.
- 2 Nalezení cesty z kořene do druhého vrcholu.
- 3 Porovnání cest a nalezení nejdelšího společného prefixu.

Nejbližší společný předek – algoritmus

```
1 class TreeNode:
2     ...
3     def find_lca(self, data1, data2, search_alg:Callable[[Any,Any],
4         ↪ list["TreeNode"]]) -> "TreeNode":
5         """Najde nejnižšího společného předka dvou vrcholů."""
6         # Najde cesty od kořene k oběma vrcholům
7         path_to_node1 = search_alg(data1, [])
8         path_to_node2 = search_alg(data2, [])
9
10        # Najde nejnižšího společného předka zjištěním společného začátku cest
11        lca = self
12        for a, b in zip(path_to_node1, path_to_node2):
13            if a == b:
14                lca = a
15            else:
16                break
17        return lca
```

Hledání cesty od kořene – prohlédávání do hloubky

```
1 class TreeNode:
2     ...
3     def find_path_dfs(self, target_data, path=[]) -> Optional[list["TreeNode"]]:
4         """Finds path from root to a node with the given data using DFS."""
5         path.append(self)           # Přidá aktuální vrchol do cesty
6         if self.data == target_data:
7             return path             # Pokud jsou data shodná s těmi v tomto
8                                     ↪ vrcholu, vrátí cestu
9         for child in self.children: # Jinak prohlédá všechny potomky rekurzivně
10            result = child.find_path_dfs(target_data, path)
11            # Pokud je nalezená cesta nenulová, vrátí ji
12            if result is not None:
13                return result
14
15         path.pop()
16         return None                 # Pokud není nalezena žádná cesta, vrátí None
```

Hledání cesty od kořene – prohlédávání do šířky

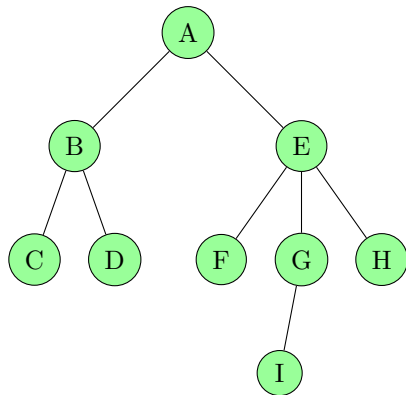
```
1 class TreeNode:
2     ...
3     def find_path_bfs(self, target_data, path=[]) ->
4         ↪ Optional[list["TreeNode"]]:
5             """Najde cestu od kořene k vrcholu s danými daty pomocí BFS."""
6             queue = deque([(self, [self])]) # fronta (vrchol, cesta k
7                 ↪ vrcholu)
8             while queue:
9                 current, path = queue.popleft()
10                if current.data == target_data:
11                    return path
12                for child in current.children:
13                    queue.append((child, path + [child]))
14            return None
```



Hledání cesty mezi libovolnými uzly

Nalezení cesty z vrcholu A do vrcholu B.

- 1 Nalezení vrcholu A a zaznamenání cesty.
- 2 Nalezení vrcholu B a zaznamenání cesty.
- 3 Porovnání cest a nalezení nejbližšího společného předka.
- 4 Určení cesty od vrcholu A k nejbližšímu společnému předkovi a k vrcholu B.



Hledání cesty – algoritmus hledání vrcholů

```
1 class TreeNode:
2     ...
3     def _dfs(self, node: "TreeNode", end: "TreeNode", path:
4         ↪ List["TreeNode"]) -> bool:
5         """Provede prohledávání do hloubky pro nalezení koncového
6         ↪ vrcholu."""
7         if node is end:
8             return True
9         for child in node.children:
10             if self._dfs(child, end, path):
11                 path.append(child)
12                 return True
13         return False
```

Hledání cesty – algoritmus, zaznamenávání cesty

```
12 class TreeNode:
13     ...
14     def find_path(self, end: "TreeNode") -> List["TreeNode"]:
15         """Najde cestu mezi dvěma vrcholy ve stromu. """
16         path: List[TreeNode] = []
17         if self._dfs(self, end, path):
18             path.append(self)
19             path.reverse()
20             return path
21         else:
22             return []
```

