

Stromy, grafy

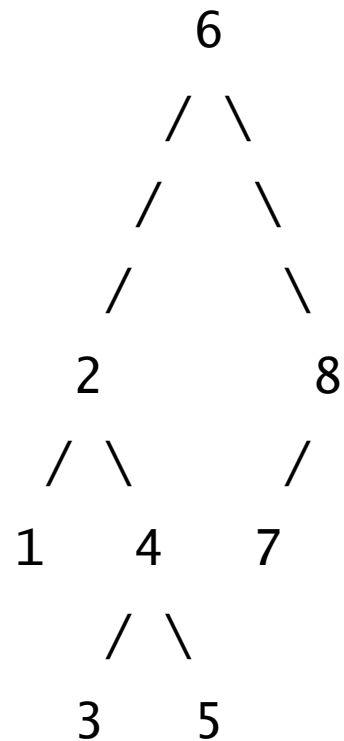
Stromy

Uzly stromu Tree jsou reprezentovány termy

● `tree(Left,Value,Right)`: Left a Right jsou opět stromy, Value je ohodnocení uzlu

● `leaf(Value)`: Value je ohodnocení uzlu

● Příklad:



`tree(tree(leaf(1), 2, tree(leaf(3),4,leaf(5))), 6, tree(leaf(7),8,[])))`

Stromy: hledání prvku $\text{in}(X,T)$

Predikát $\text{in}(X,T)$ uspěje, pokud se prvek X nachází ve stromu T .

Prvek X se nachází ve stromě T , jestliže

- X je listem stromu T , jinak leaf(X)
- X je kořen stromu T , jinak tree(Left, X ,Right)
- X je menší než kořen stromu T , pak se nachází v levém podstromu T , jinak
- X se nachází v pravém podstromu T

Stromy: hledání prvku $\text{in}(X,T)$

Predikát $\text{in}(X,T)$ uspěje, pokud se prvek X nachází ve stromu T .

Prvek X se nachází ve stromě T , jestliže

- X je listem stromu T , jinak leaf(X)
- X je kořen stromu T , jinak tree(Left,X,Right)
- X je menší než kořen stromu T , pak se nachází v levém podstromu T , jinak
- X se nachází v pravém podstromu T

```
in(X, leaf(X)) :- !.
```

```
in(X, tree(_,X,_)) :- !.
```

```
in(X, tree(Left, Root, Right) ) :-  
    X<Root, !,  
    in(X,Left).
```

```
in(X, tree(Left, Root, Right) ) :-  
    in(X,Right).
```

Stromy: přidávání `add(T, X, TWithX)`

Prvek X přidej do stromu T jednou z následujících možností:

- pokud $T = []$, pak je nový strom `leaf(X)`
- pokud $T = \text{leaf}(V)$ a $X > V$, pak vznikne nový strom s kořenem V , vpravo má `leaf(X)` (vlevo je `[]`)
pokud $T = \text{leaf}(V)$ a $X < V$, pak vznikne nový strom s kořenem V , vlevo má `leaf(X)` (vpravo je `[]`)
- pokud $T = \text{tree}(L, V, R)$ a $X > V$, pak v novém stromě L ponechej a X přidej doprava (rekurzivně)
pokud $T = \text{tree}(L, V, R)$ a $X < V$, pak v novém stromě R ponechej a X přidej doleva (rekurzivně)

Stromy: přidávání $\text{add}(T, X, T\text{with}X)$

Prvek X přidej do stromu T jednou z následujících možností:

- pokud $T = []$, pak je nový strom $\text{leaf}(X)$
- pokud $T = \text{leaf}(V)$ a $X > V$, pak vznikne nový strom s kořenem V , vpravo má $\text{leaf}(X)$ (vlevo je $[]$)
pokud $T = \text{leaf}(V)$ a $X < V$, pak vznikne nový strom s kořenem V , vlevo má $\text{leaf}(X)$ (vpravo je $[]$)
- pokud $T = \text{tree}(L, V, R)$ a $X > V$, pak v novém stromě L ponechej a X přidej doprava (rekurzivně)
pokud $T = \text{tree}(L, V, R)$ a $X < V$, pak v novém stromě R ponechej a X přidej doleva (rekurzivně)

$\text{add}([], X, \text{leaf}(X)) \text{ :- !.}$

$\text{add}(\text{leaf}(V), X, \text{tree}([], V, \text{leaf}(X))) \text{ :- } X > V, \text{ !.}$

$\text{add}(\text{leaf}(V), X, \text{tree}(\text{leaf}(X), V, [])) \text{ :- } !. X < V,$

Stromy: přidávání `add(T, X, TwithX)`

Prvek X přidej do stromu T jednou z následujících možností:

- pokud $T = []$, pak je nový strom `leaf(X)`
- pokud $T = \text{leaf}(V)$ a $X > V$, pak vznikne nový strom s kořenem V , vpravo má `leaf(X)` (vlevo je `[]`)
pokud $T = \text{leaf}(V)$ a $X < V$, pak vznikne nový strom s kořenem V , vlevo má `leaf(X)` (vpravo je `[]`)
- pokud $T = \text{tree}(L, V, R)$ a $X > V$, pak v novém stromě L ponechej a X přidej doprava (rekurzivně)
pokud $T = \text{tree}(L, V, R)$ a $X < V$, pak v novém stromě R ponechej a X přidej doleva (rekurzivně)

```
add([], X, leaf(X)) :- !.
```

```
add(leaf(V), X, tree([], V, leaf(X)) ) :- X > V, !.
```

```
add(leaf(V), X, tree(leaf(X), V, []) ) :- !, X < V,
```

```
add(tree(L, V, R), X, tree(L, V, R1)) :- X > V, !, add(R, X, R1).
```

```
add(tree(L, V, R), X, tree(L1, V, R)) :- X < V, add(L, X, L1).
```

Procházení stromů

Napište predikát `traverse(Tree, List)`, který projde traversálně strom `Tree`. Seznam `List` pak obsahuje všechny prvky tohoto stromu.

Pořadí preorder: nejprve uzel, pak levý podstrom, nakonec pravý podstrom

```
?- traverse(tree(tree(leaf(1),2,tree(leaf(3),4,leaf(5))),6,
              tree(leaf(7),8,leaf(9))), [6,2,1,4,3,5,8,7,9]).
```

 (preorder)

```
traverse(T,Pre):- t_pre(T,Pre,[]).
```

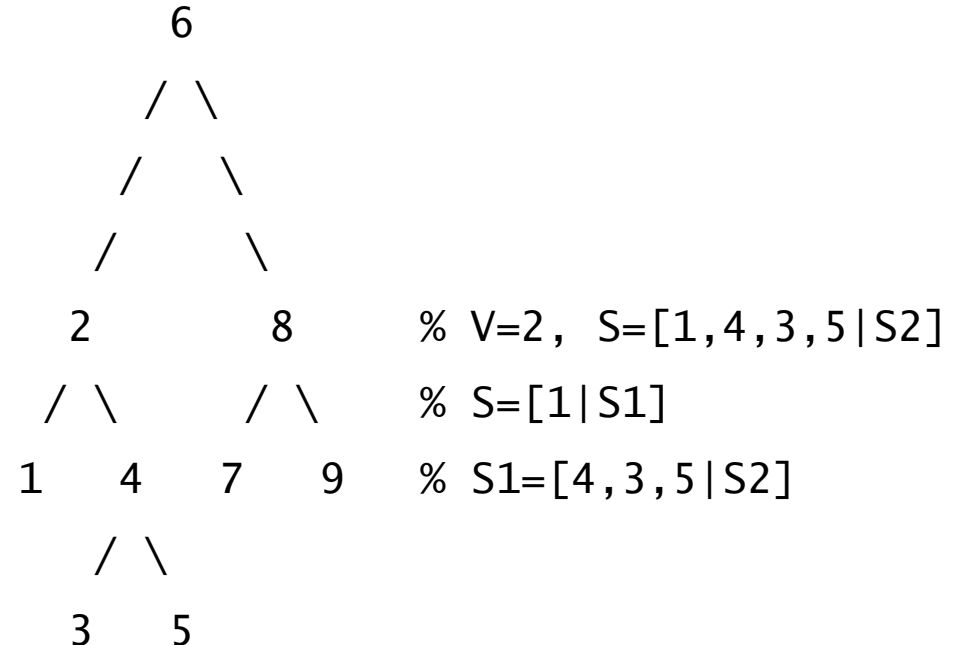
```
t_pre([],S,S).
```

```
t_pre(leaf(V),[V|S],S).
```

```
t_pre(tree(L,V,R),[V|S],S2):-
```

```
    t_pre(L,S,S1),
```

```
    t_pre(R,S1,S2).
```



Procházení stromů

Napište predikát `traverse(Tree, List)`, který projde traversálně strom `Tree`. Seznam `List` pak obsahuje všechny prvky tohoto stromu.

Pořadí preorder: nejprve uzel, pak levý podstrom, nakonec pravý podstrom

?- `traverse(tree(tree(leaf(1),2),tree(leaf(3),4,leaf(5))),6,
tree(leaf(7),8,leaf(9))), [6,2,1,4,3,5,8,7,9]).` (preorder)

```
traverse(T,Pre):- t_pre(T,Pre, []).
```

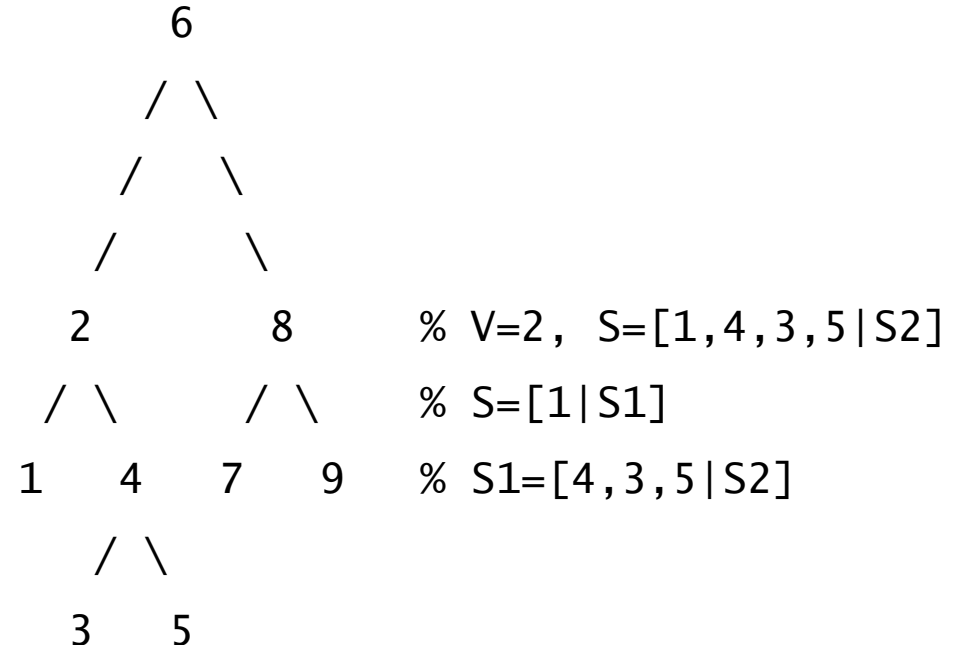
```
t_pre([],S,S).
```

```
t_pre(leaf(V),[V|S],S).
```

```
t_pre(tree(L,V,R),[V|S],S2):-
```

```
    t_pre(L,S,S1),
```

```
    t_pre(R,S1,S2).
```



Použit princip rozdílových seznamů

Procházení stromů

```
traverse(T,Pre):- t_pre(T,Pre, []).
```

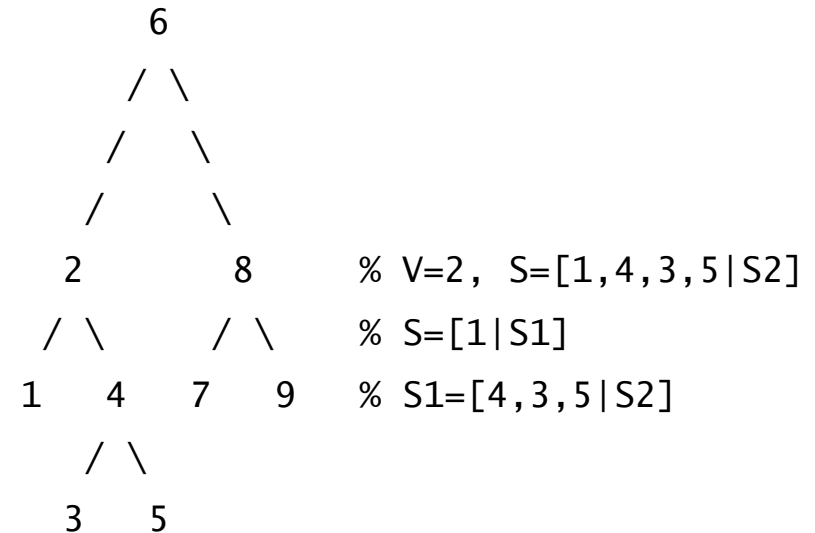
```
t_pre([],S,S).
```

```
t_pre(leaf(V),[V|S],S).
```

```
t_pre(tree(L,V,R),[V|S],S2):-
```

```
    t_pre(L,S,S1),
```

```
    t_pre(R,S1,S2).
```



Modifikuje algoritmus tak, aby byly uzly vypsány v pořadí inorder (nejprve levý podstrom, pak uzel a nakonec pravý podstrom), tj. [1,2,3,4,5,6,7,8,9]

Procházení stromů

```
traverse(T,Pre):- t_pre(T,Pre, []).
```

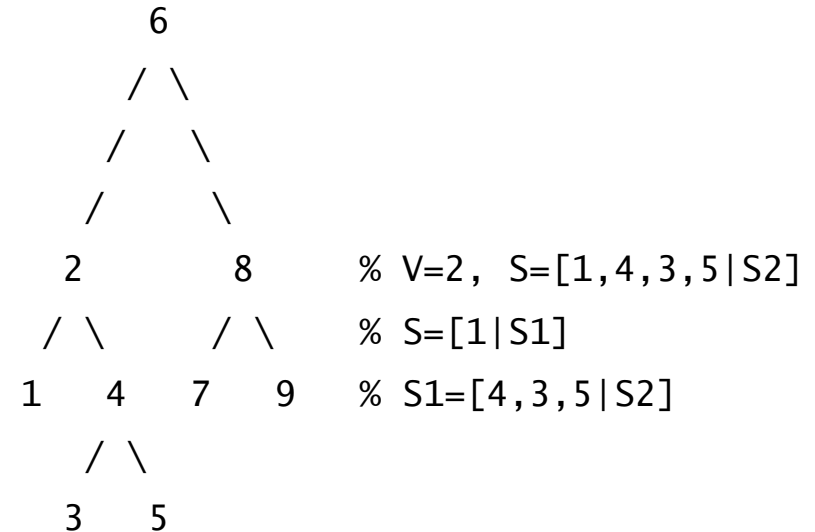
```
t_pre([],S,S).
```

```
t_pre(leaf(V),[V|S],S).
```

```
t_pre(tree(L,V,R),[V|S],S2):-
```

```
    t_pre(L,S,S1),
```

```
    t_pre(R,S1,S2).
```



Modifikuje algoritmus tak, aby byly uzly vypsány v pořadí inorder (nejprve levý podstrom, pak uzel a nakonec pravý podstrom), tj. [1,2,3,4,5,6,7,8,9]

```
traverse(T,In):- t_in(T,In, []).
```

```
t_in([],S,S).
```

```
t_in(leaf(V),[V|S],S).
```

```
t_in(tree(L,V,R),S,S2):-
```

```
    t_in(L,S,[V|S1]),
```

```
    t_in(R,S1,S2).
```

DÚ: Procházení stromu postorder

Modifikuje algoritmus tak, aby byly uzly vypsány v pořadí postorder (nejprve levý podstrom, pak pravý podstrom a nakonec uzel), tj. [1,3,5,4,2,7,9,8,6]

DÚ: Procházení stromu postorder

Modifikuje algoritmus tak, aby byly uzly vypsány v pořadí postorder (nejprve levý podstrom, pak pravý podstrom a nakonec uzel), tj. [1,3,5,4,2,7,9,8,6]

```
traverse_post(T,Post):-
```

```
    t_post(T,Post, []).
```

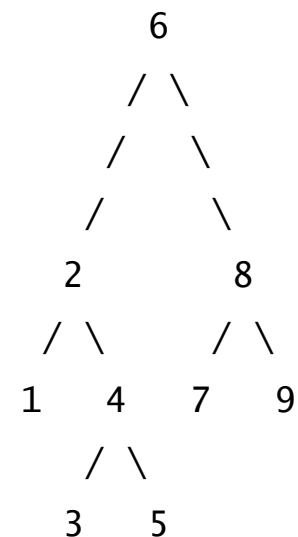
```
t_pre([],S,S).
```

```
t_post(leaf(V),[V|S],S).
```

```
t_post(tree(L,V,R),S,S2):-
```

```
    t_post(L,S,S1),
```

```
    t_post(R,S1,[V|S2]).
```



Reprezentace grafu

- Reprezentace grafu: pole následníků uzlů
- Grafy nebudeme modifikovat, tj. pro reprezentaci pole lze využít term
- (Orientovany) neohodnocený graf

`graf([2, 3], [1, 3], [1, 2]).`

```
1--2
 \ |
  \|
   3
```

`graf([2, 4, 6], [1, 3], [2], [1, 5], [4, 6], [1, 5]).`

```
5--4
 |  |
6--1--2--3
```

?- functor(Graf, graf, PocetUzlu).

?- arg(Uzel, Graf, Sousedi).

- (Orientovany) ohodnocený graf [Soused-Ohodnoceni|Sousedi]

`graf([2-1, 3-2], [1-1, 3-2], [1-2, 2-2]).`

`graf([2-1, 4-3, 6-1], [1-1, 3-2], [2-2], [1-3, 5-1], [4-1, 6-2], [1-1, 5-2]).`

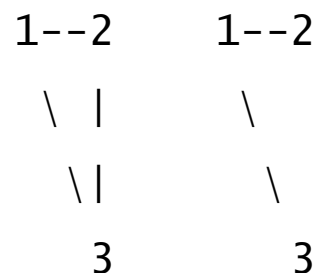
Procházení grafu do hloubky

Napište predikát `dfs(Uzel,Graf,Parents)` pro procházení grafu Graf do hloubky z uzlu Uzel. Výsledkem je datová struktura Parents, která reprezentuje strom vzniklý při prohledávání do hloubky (pro každý uzel stromu známe jeho rodiče).

Datová struktura pro rodiče uzlů:

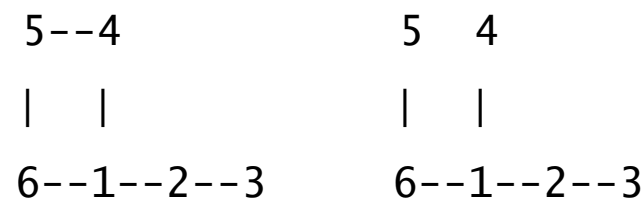
- při reprezentaci rodičů lze využít term s aritou odpovídající počtu uzlů
- iniciálně jsou argumentu termu volné proměnné
- na závěr je v N-tém argumentu uložen rodič N-tého uzlu (iniciální uzel označíme empty)

`graf([2,3],[1,3],[1,2]).`



`Uzel=2: rodic(2,empty,1)`

`graf([2,4,6],[1,3],[2],[1,5],[4,6],[1,5]).`



`Uzel=4: rodic(4, 1, 2, empty, 6, 1)`

Procházení grafu do hloubky: algoritmus I

Procházení grafu z uzlu Uzel

- Vytvoříme term pro rodiče (všichni rodiči jsou zatím volné proměnné)
- Uzel Uzel má prázdného rodiče a má sousedy Sousedí
- Procházíme (rekurzivně) všechny sousedy v Sousedí

`dfs(Uze1, Graf, Parents) :-`

`functor(Graf, graf, Pocet),`

`functor(Parents, rodice, Pocet),`

`arg(Uze1, Parents, empty),`

`arg(Uze1, Graf, Sousedí),`

`prochazej_sousedy(Sousedí, Uze1, Graf, Parents).`

Procházení grafu do hloubky: algoritmus II

Procházení sousedů uzlu Uzel (pokud Uzel nemá sousedy, tj. Sousedí=[], končíme)

1. Uzel V je první soused
2. Zjistíme rodiče uzlu V ... pomocí `arg(V,Parents,Rodic)`
3. Pokud jsme V ještě neprošli (tedy nemá rodiče a platí `var(Rodic)`), tak
 - (a) nastavíme rodiče uzlu V na Uzel ... pomocí `arg/3`
 - (b) rekurzivně procházej všechny sousedy uzlu V

pokud jsme V prošli, dále tímto uzlem nepokračujeme, tj. celkem (`var(Rodic) -> ...; true`)
4. Procházej zbývající sousedy uzlu Uzel

Procházení grafu do hloubky: algoritmus II

Procházení sousedů uzlu Uzel (pokud Uzel nemá sousedy, tj. Sousedí=[], končíme)

1. Uzel V je první soused
2. Zjistíme rodiče uzlu V ... pomocí `arg(V,Parents,Rodic)`
3. Pokud jsme V ještě neprošli (tedy nemá rodiče a platí `var(Rodic)`), tak
 - (a) nastavíme rodiče uzlu V na Uzel ... pomocí `arg/3`
 - (b) rekurzivně procházej všechny sousedy uzlu V

pokud jsme V prošli, dále tímto uzlem nepokračujeme, tj. celkem (`var(Rodic) -> ...; true`)

4. Procházej zbývající sousedy uzlu Uzel

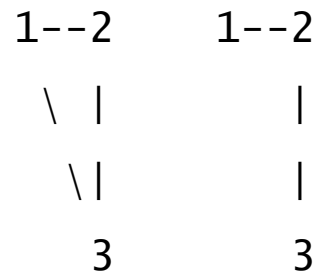
`prochazej_sousedy([],_,_,_)`.

```
prochazej_sousedy([V|T],Uzel,Graf,Parents) :- arg(V,Parents,Rodic),  
    ( nonvar(Rodic) -> true  
    ; Rodic = Uzel,  
      arg(V,Graf,SousediV),  
      prochazej_sousedy(SousediV,V,Graf,Parents)  
    ),  
    prochazej_sousedy(T,Uzel,Graf,Parents).
```

DÚ: Procházení grafu do šířky

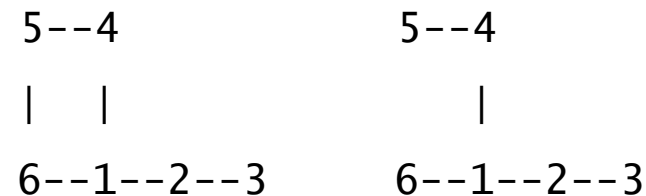
Napište predikát $\text{bfs}(U,G,P)$ pro procházení grafu G do šířky z uzlu U . Výsledkem procházení je datová struktura P , která reprezentuje strom vzniklý při prohledávání grafu G do šířky (pro každý uzel stromu známe jeho rodiče).

$\text{graf}([2,3], [1,3], [1,2])$.



$U=2: \text{rodic}(2, \text{empty}, 2)$

$\text{graf}([2,4,6], [1,3], [2], [1,5], [4,6], [1,5])$.



$U=4: \text{rodic}(4, 1, 2, \text{empty}, 4, 1)$