

**Všechna řešení,  
stromy, grafy**

# Všechna řešení

```
% z(Jmeno,Prijmeni,Sex,Vek,Prace,Firma)
z(petr,novak,m,30,skladnik,skoda).  z(pavel,novy,m,40,mechanik,skoda).
z(rostislav,lucensky,m,50,technik,skoda).  z(alena,vesela,z,25,sekretarka,skoda).
z(jana,dankova,z,35,asistentka,skoda).  z(lenka,merinska,z,35,ucetni,skoda).
z(roman,maly,m,35,manazer,cs).  z(alena,novotna,z,40,ucitelka,zs_stara).
z(david,novy,z,30,ucitel,zs_stara).  z(petra,spickova,z,45,reditelka,zs_stara).
```

● Najděte jméno a příjmení všech lidí.

```
?- findall(Jmeno-Prijmeni, z(Jmeno,Prijmeni,_,_,_,_),L).
```

```
?- bagof( Jmeno-Prijmeni, [S,V,Pr,F] ^ z(Jmeno,Prijmeni,S,V,Pr,F) , L ).
```

```
?- bagof( Jmeno-Prijmeni, [V,Pr,F] ^ z(Jmeno,Prijmeni,S,V,Pr,F) , L ).
```

● Najděte jméno a příjmení všech zaměstnanců firmy skoda a cs

```
?- findall( c(J,P,Firma), ( z(J,P,_,_,_,Firma), ( Firma=skoda ; Firma=cs ) ), L
```

```
?- bagof( J-P, [P,S,V,Pr]^z(J,P,S,V,Pr,F),( F=skoda ; F=cs ) ) , L ).
```

```
?- setof( P-J, [P,S,V,Pr]^z(J,P,S,V,Pr,F),( F=skoda ; F=cs ) ) , L ).
```

# Všechna řešení: příklady

1. Jaká jsou příjmení všech žen?
2. Kteří lidé mají více než 30 roků? Nalezněte jejich jméno a příjmení.
3. Nalezněte abecedně seřazený seznam všech lidí.
4. Nalezněte příjmení učitelů ze zs\_stara.
5. Jsou v databázi dva bratři (mají stejné příjmení a různá jména)?
6. Které firmy v databázi mají více než jednoho zaměstnance?

# Všechna řešení: příklady

1. Jaká jsou příjmení všech žen?
2. Kteří lidé mají více než 30 roků? Nalezněte jejich jméno a příjmení.
3. Nalezněte abecedně seřazený seznam všech lidí.
4. Nalezněte příjmení učitelů ze zs\_stara.
5. Jsou v databázi dva bratři (mají stejné příjmení a různá jména)?
6. Které firmy v databázi mají více než jednoho zaměstnance?

1. `findAll(Prijmeni, z(_, Prijmeni, z(_, _, _), L)).`
2. `findAll(Jmeno-Prijmeni, (z(Jmeno, Prijmeni, _, Vek, _, _), Vek > 30), L).`
3. `setof(P-J, [S, V, Pr, F]^z(J, P, S, V, Pr, F), L).`
4. `findAll(Prijmeni, (z(_, Prijmeni, _, _, P, zs_stara), (P=ucitel; P=ucitelka)), L).`
5. `findAll(b(J1-P, J2-P), ( z(J1, P, _, _, _, _), z(J2, P, _, _, _, _), J1@<J2, J1\=J2 ), L).`
6. `bagof(P, [J, S, V, Pr]^z(J, P, S, V, Pr, F), L), length(L, Pocet), Pocet > 1).`

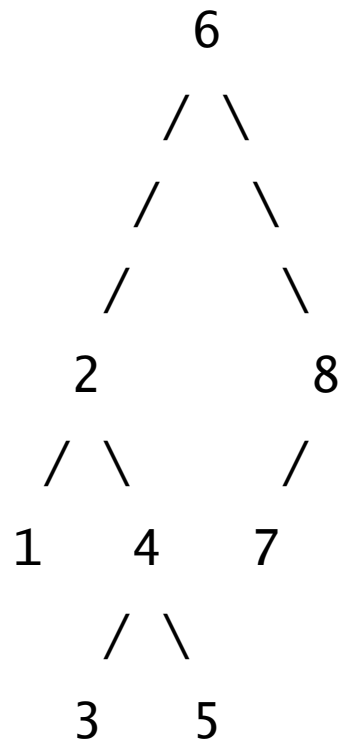
# Stromy

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 $\text{in}(X, \text{Tree})$

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 $\text{in}(X, \text{Tree})$

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(Tree, X, TreeWithX)`

Prvek  $X$  přidej do stromu  $T$  jednou z

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



# Stromy: přidávání `add(Tree, X, TreeWithX)`

Prvek X přidej do stromu T jednou z

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

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

```
add(leaf(V), X, T1) :-
```

```
    ( X > V, !, T1 = tree([], V, leaf(X))
```

```
    ; X < V, T1 = tree(leaf(X), V, [])
```

```
    ).
```

# Stromy: přidávání `add(Tree, X, TreeWithX)`

Prvek X přidej do stromu T jednou z

- `T = []`, pak je nový strom `leaf(X)`
- `T=leaf(V)` a `X>V`, pak má nový strom kořen V a `leaf(X)` vpravo (vlevo je `[]`)  
`T=leaf(V)` a `X<V`, pak má nový strom kořen V a `leaf(X)` vlevo (vpravo je `[]`)
- `T=tree(L,_,_)` a `X>V`, pak v novém stromě L ponechej a X přidej doprava  
`T=tree(_,_,R)` a `X<V`, pak v novém stromě R ponechej a X přidej doleva

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

```
add(leaf(V), X, T1) :-
```

```
    ( X>V, !, T1=tree([],V,leaf(X))
```

```
    ; X<V, T1=tree(leaf(X),V,[])
```

```
    ).
```

```
add(tree(L,V,R), X, tree(L1,V,R1)) :-
```

```
    ( X>V, !, L1=L, add(R,X,R1)
```

```
    ; X<V, R1=R, add(L,X,L1)
```

```
    ).
```

# Procházení stromů

?- 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, []).

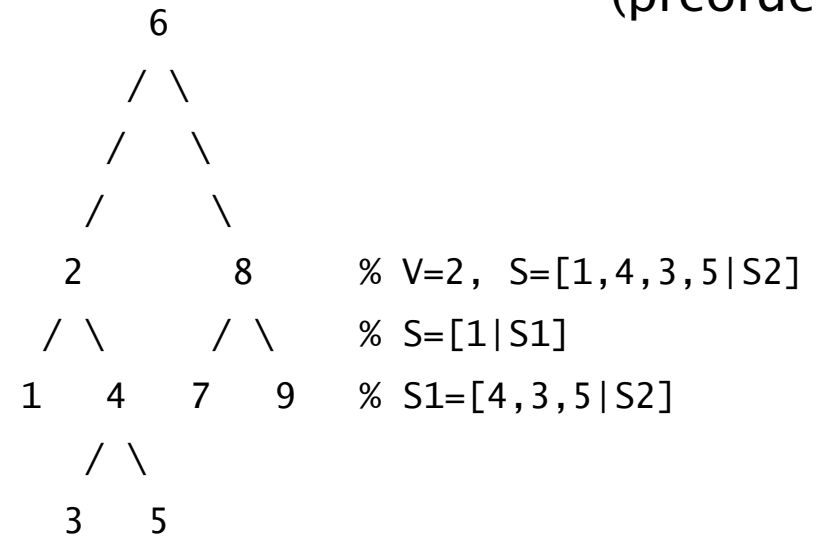
t\_pre(leaf(V), [V|S], S).

t\_pre(tree(L,V,R), [V|S], S2):-

    t\_pre(L,S,S1),

    t\_pre(R,S1,S2).

Použit princip rozdílových seznamů



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(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, []).
```

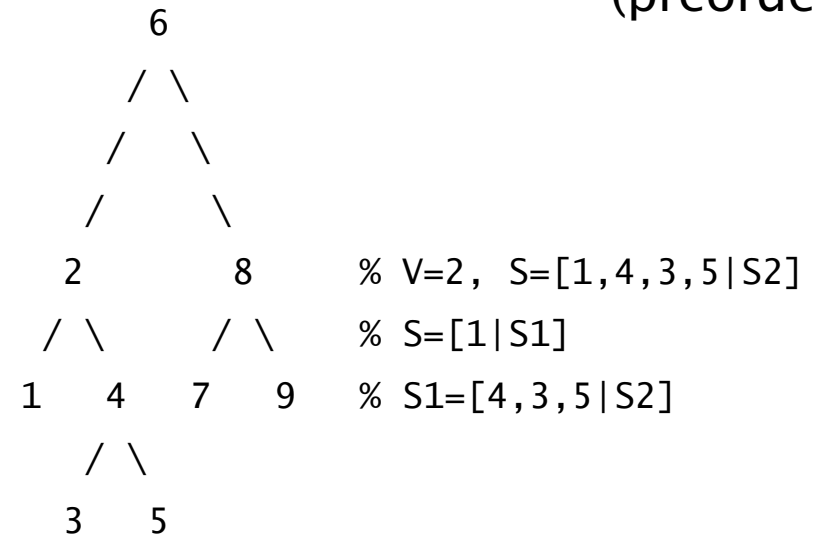
```
t_pre(leaf(V), [V|S], S).
```

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

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

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

Použit princip rozdílových seznamů



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

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

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

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

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

?- `functor(Graf, graf, PocetUzlu).`

?- `arg(Uzel, Graf, Sousedi).`

- (Orientovany) ohodnocený graf

`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, 6-1], [4-1], [1-1, 4-1]).`

# Procházení grafu do hloubky

## ● Rodiče uzlů:

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

## ● Procházení grafu z uzlu U

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

## ● Procházení sousedů S uzlu U

- Uzel V je první soused
- Nastavíme rodiče uzlu V na uzel U
- Pokud jsme V ještě neprošli (nemá rodiče), tak rekurzivně procházej všechny jeho sousedy
- Procházej zbývající sousedy uzlu U

# DFS: algoritmus

```
dfs(U,G,P) :-  
    functor(G,graf,Pocet),  
    functor(P,rodice,Pocet),  
    arg(U,G,Sousedi),  
    arg(U,P,empty),  
    prochazej_sousedy(Sousedi,U,G,P).  
  
prochazej_sousedy([],_,-,-).  
prochazej_sousedy([V|T],U,G,P) :-  
    arg(V,P,Rodic),  
    ( nonvar(Rodic), !  
    ;  
    Rodic = U,  
    arg(V,G,SousediV),  
    prochazej_sousedy(SousediV,V,G,P)  
    ),  
    prochazej_sousedy(T,U,G,P).
```