# Welcome To BigBlueButton

BigBlueButton is an open source web conferencing system designed for online learning

**CHAT**

Send public and private messages.

**WEBCAMS**

Hold visual meetings.

**AUDIO**

Communicate using high quality audio.

**EMOJIS**

Express yourself.

**BREAKOUT ROOMS**

Group users into breakout rooms for team collaboration.

**POLLING**

Poll your users anytime.

**SCREEN SHARING**

Share your screen.

**MULTI-USER WHITEBOARD**

Draw together.

For more information visit bigbluebutton.org →

# Algoritmizace

Marko Genyg-Berezovskyj, Daniel Průša

2010 - 2020

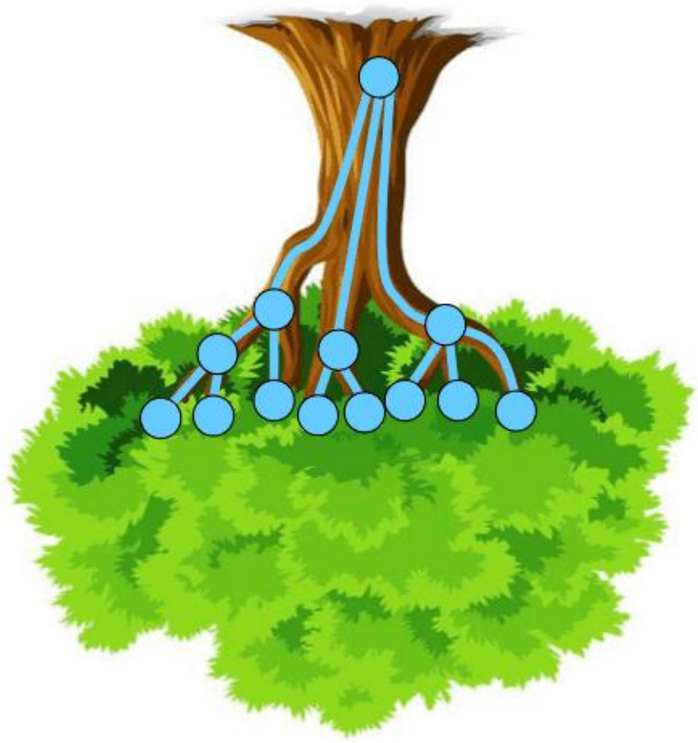
Začátek v 11:00

Přednáší: Daniel Průša

prusa@fel.cvut.cz

# Přehled

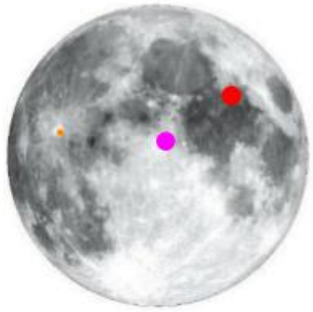
- Stromy
- Prohledávání s návratem (backtracking)
- Druhá domácí úloha

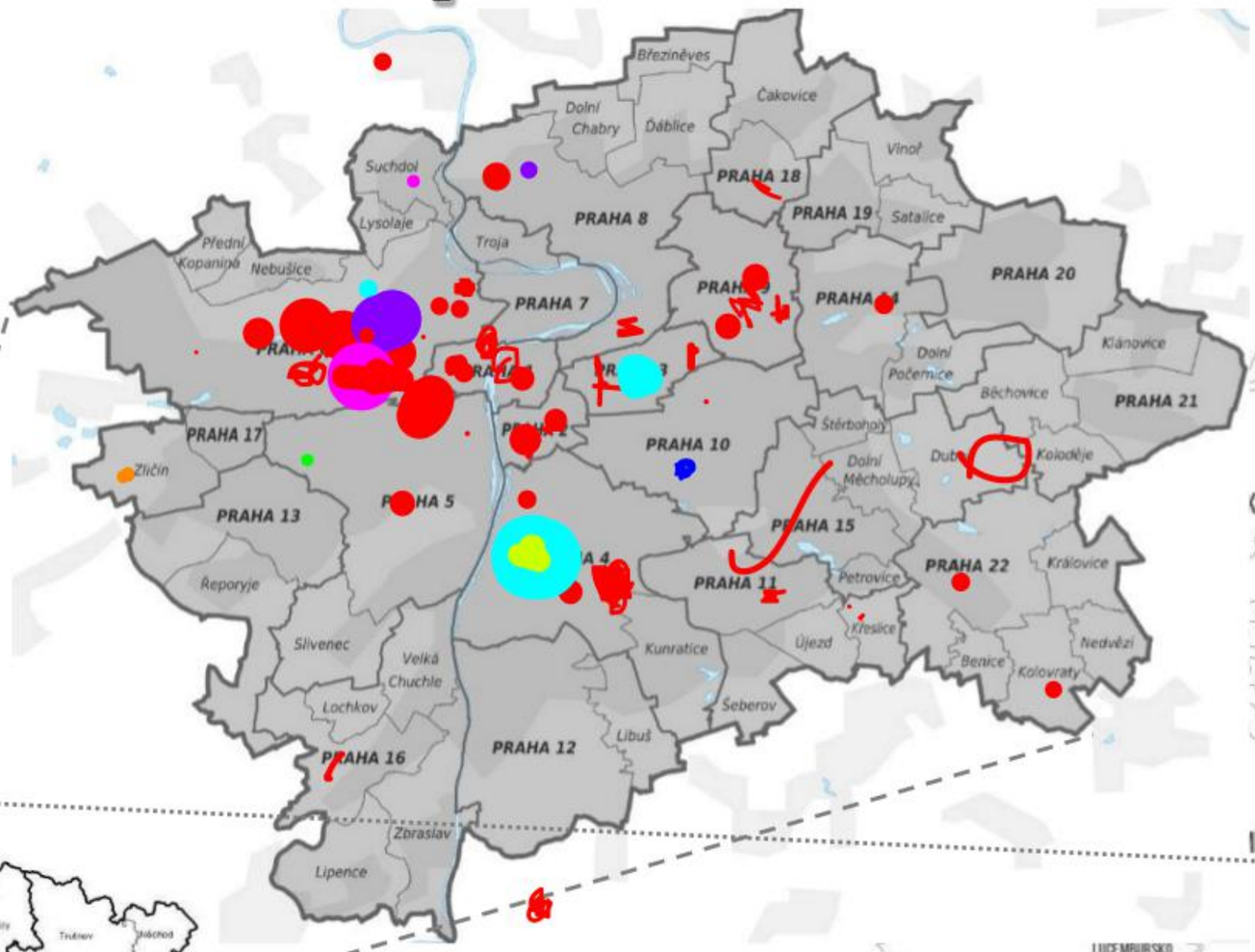


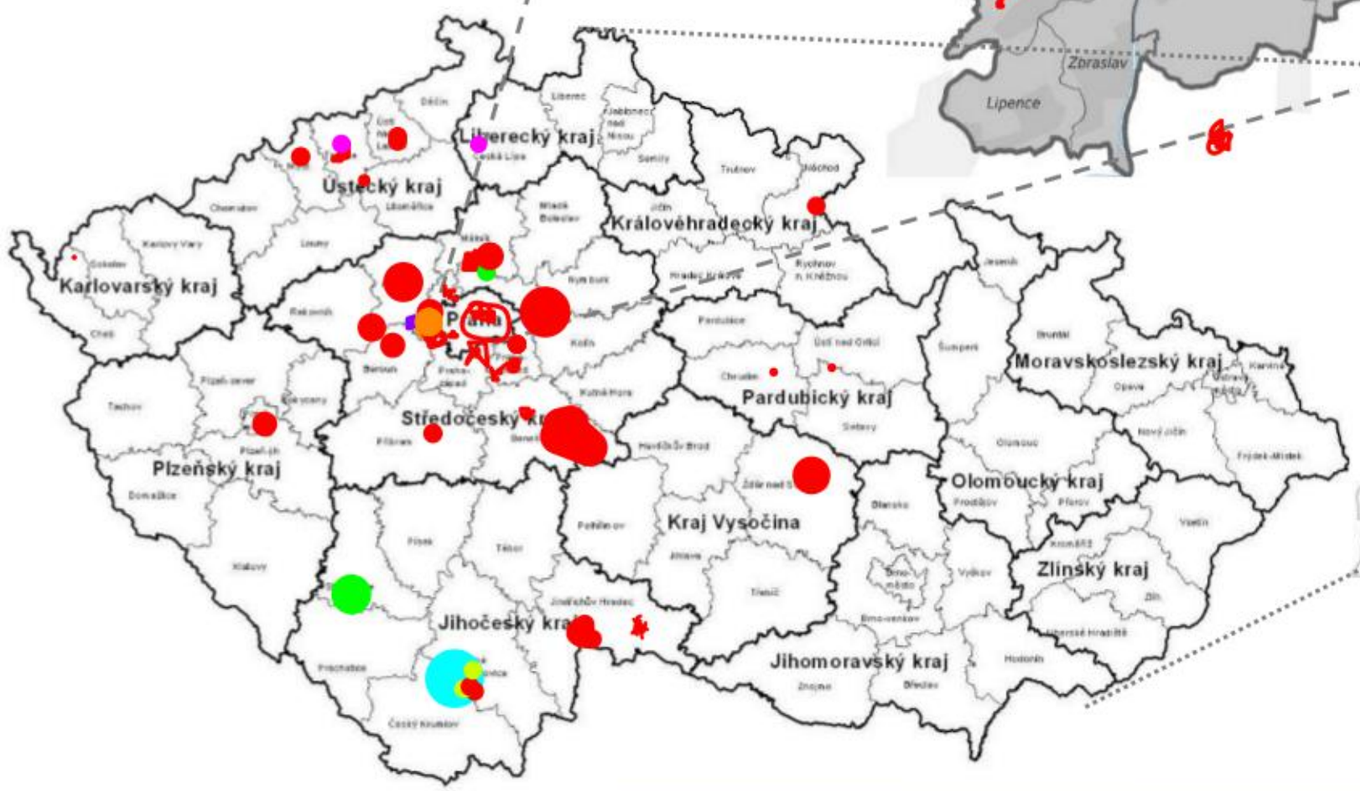
# Kde se právě teď nacházíte?



# Kde se právě teď nacházíte?

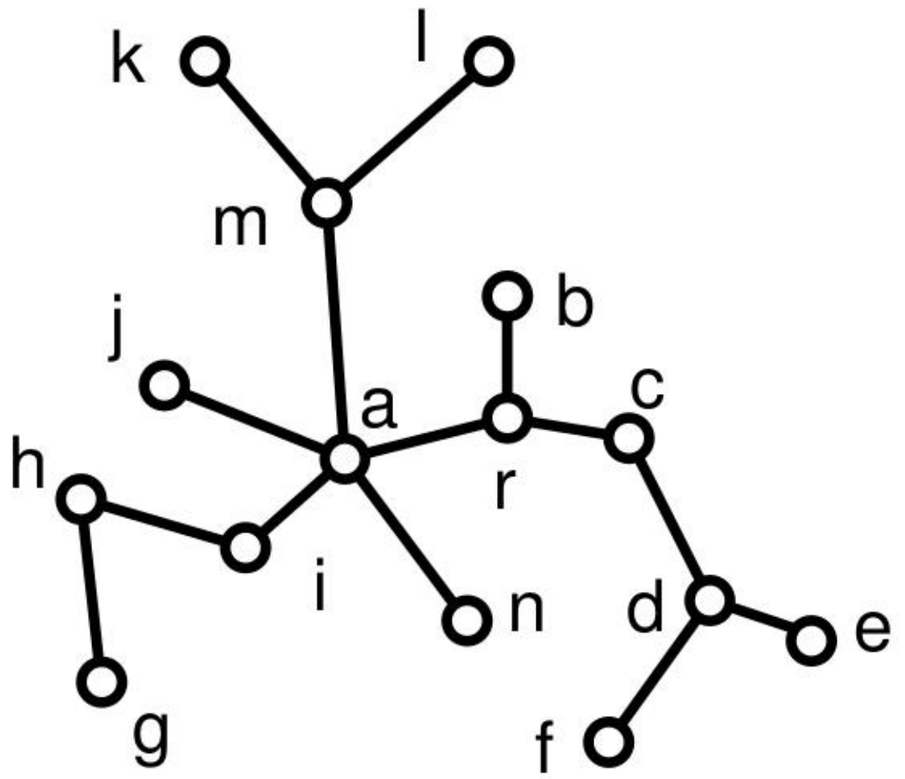




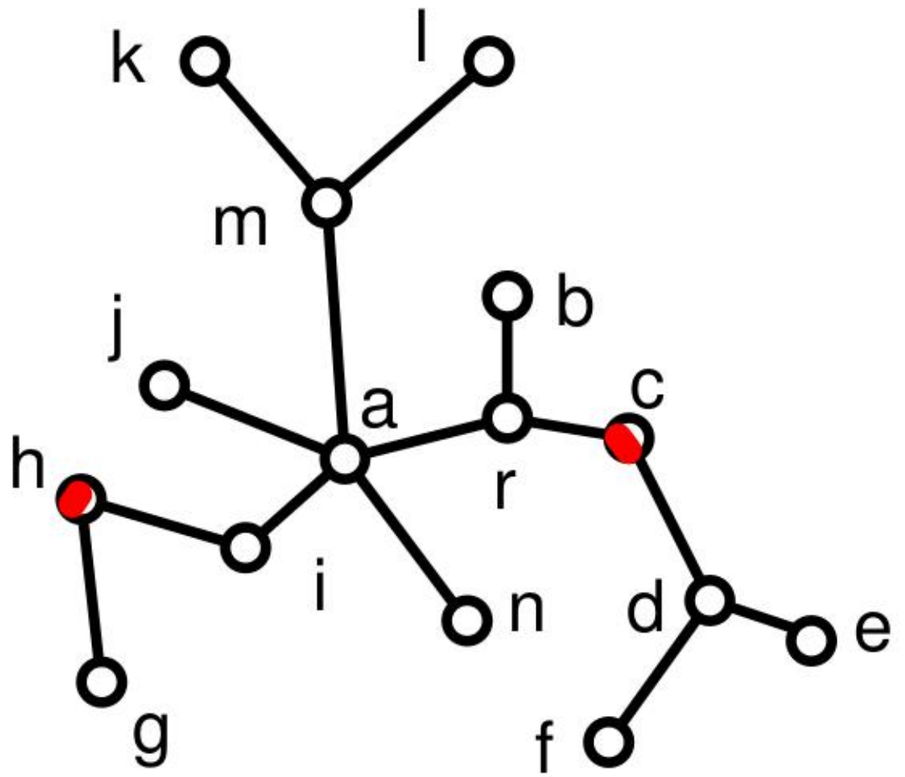

# Strom jako graf

- $G = (V, E)$
  - Souvislý graf bez cyklů.
  - Každé dva uzly spojuje právě jedna cesta.
  - $|E| = |V| - 1$.

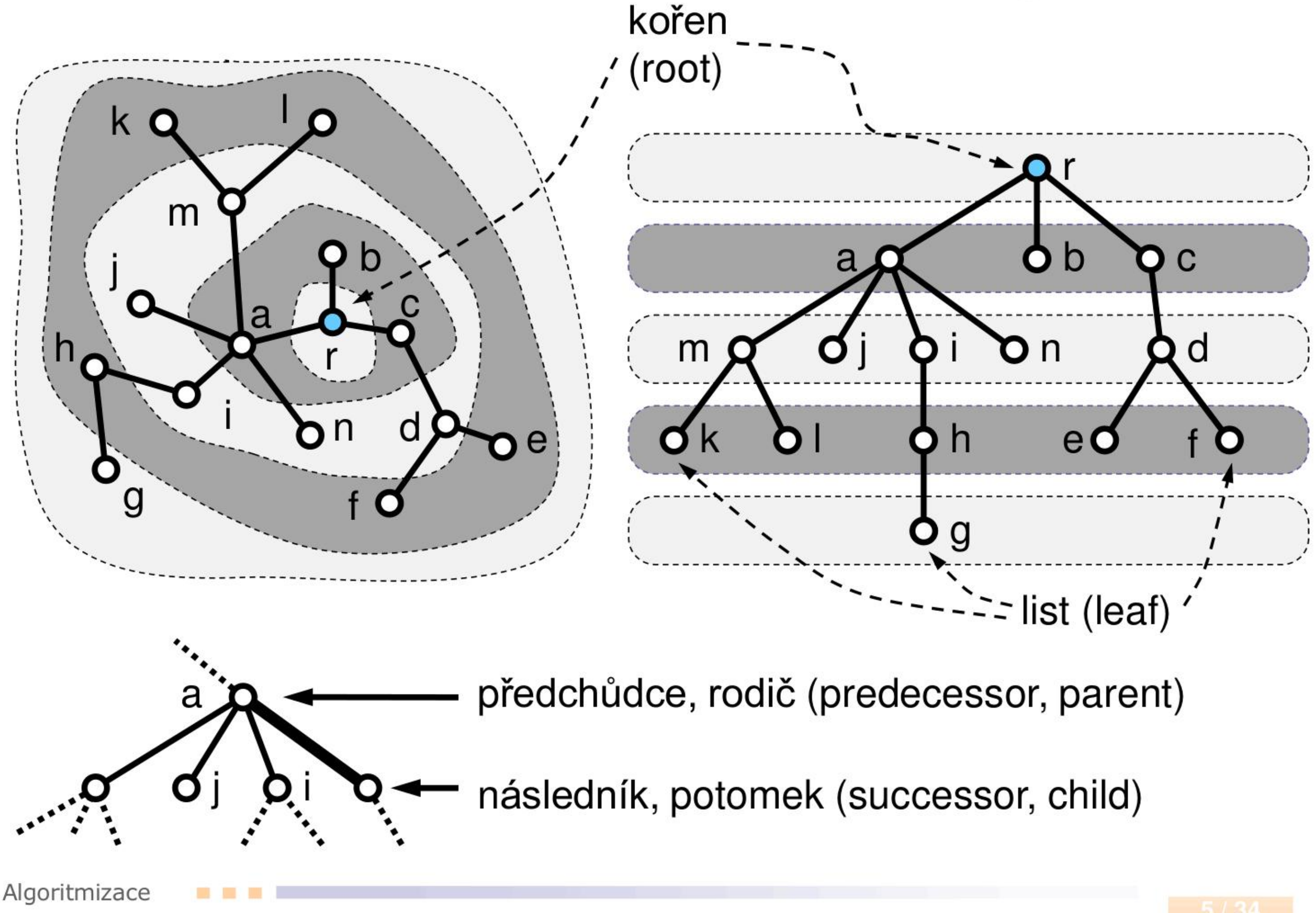


# Strom jako graf

- $G = (V, E)$
  - Souvislý graf bez cyklů.
  - Každé dva uzly spojuje právě jedna cesta.
  - $|E| = |V| - 1$.



# Kořenový strom



# Co lze stromem reprezentovat?

Aritmetický výraz

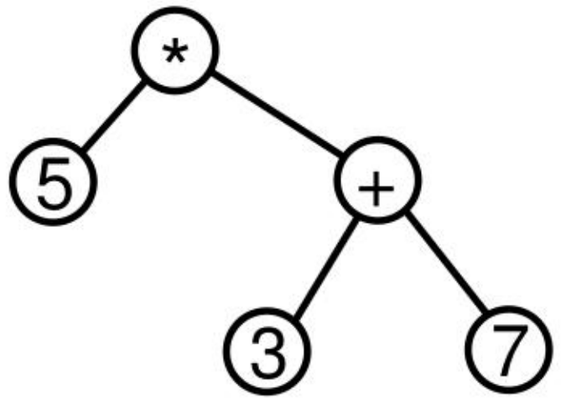
Datová struktura (BVS, halda)

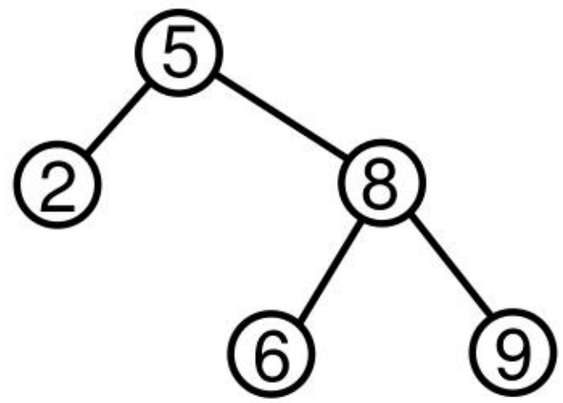
Rozhodovací strom (botanický klíč)

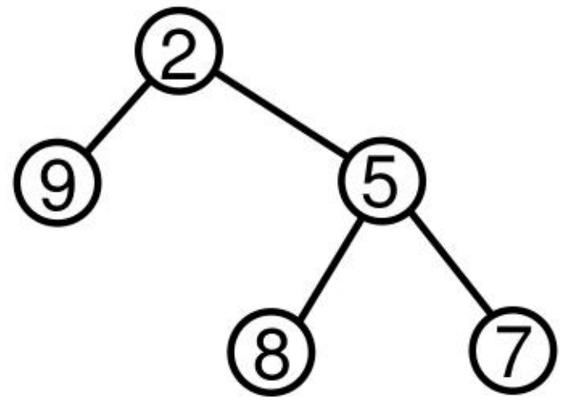
Hierarchie entit (zaměstnanci, rodokmen)

Rekurzivní volání

Stavový prostor



# Co lze stromem reprezentovat?

Aritmetický výraz

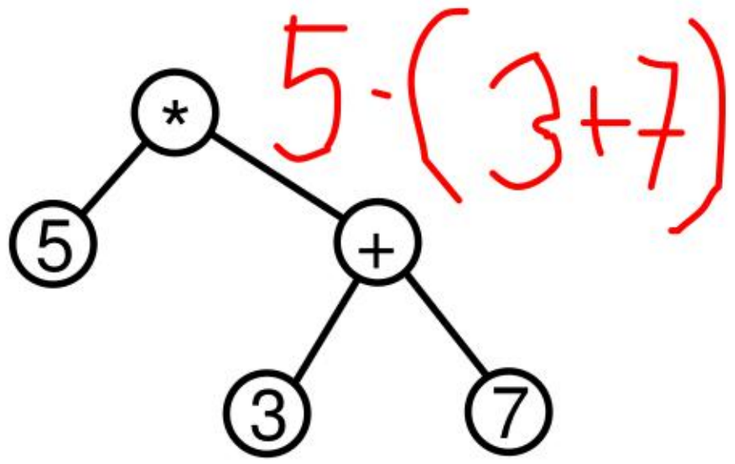
Datová struktura (BVS, halda)

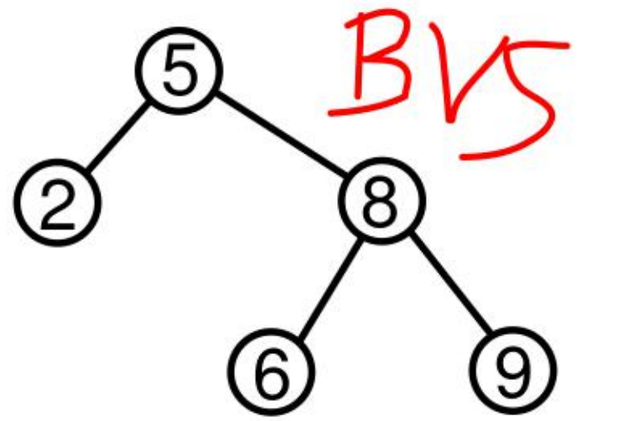
Rozhodovací strom (botanický klíč)

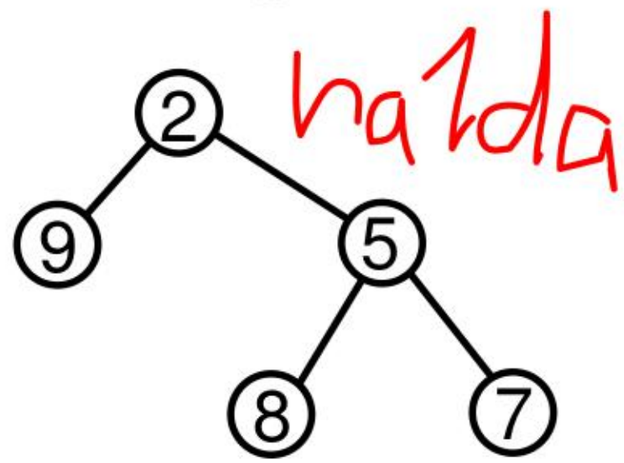
Hierarchie entit (zaměstnanci, rodokmen)

Rekurzivní volání

Stavový prostor



# Znáte tyto pojmy?

**Pravidelný** binární strom, **vyvážený** binární strom.

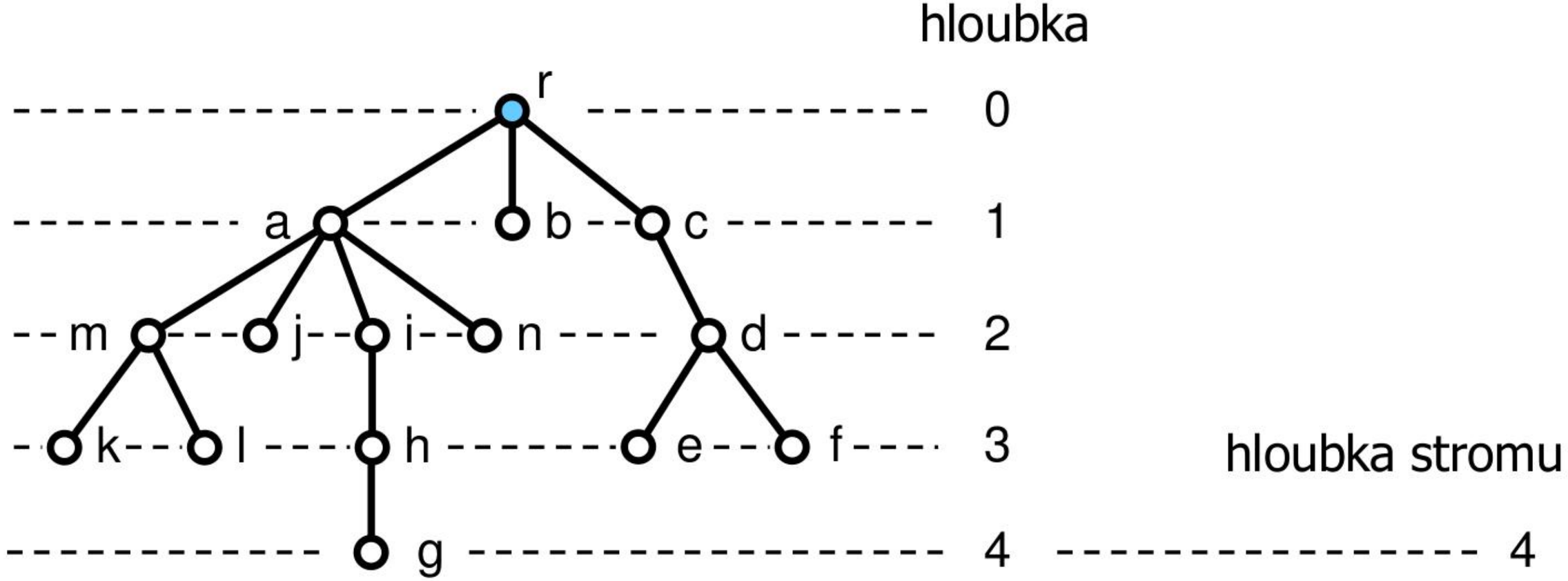
**Reprezentace** binárního stromu **v paměti**.

**Prohledávání** stromu **do hloubky** (rekurzivně, se zásobníkem).

Průchod stromem v pořadí **preorder**, **inorder**, **postorder**.

A. Znám vše uvedené jak své boty.
B. Znám víceméně vše, jen si nepamatuji některé detaily.
C. Znám pouze část.
D. Vůbec nic mi to neříká.

# Hloubka kořenového stromu

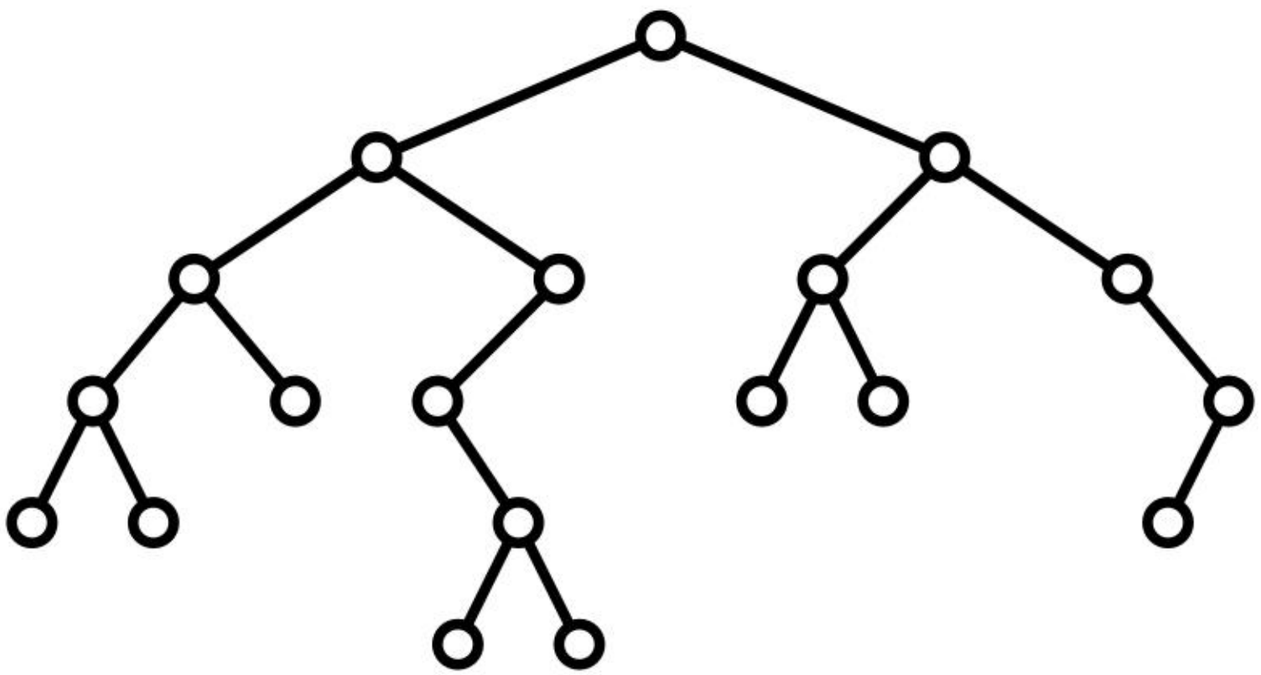



Hloubka uzlu $u$ je hranová vzdálenost $u$ od kořene.

Hloubka stromu $T$ je maximum z hloubek uzlů v $T$.

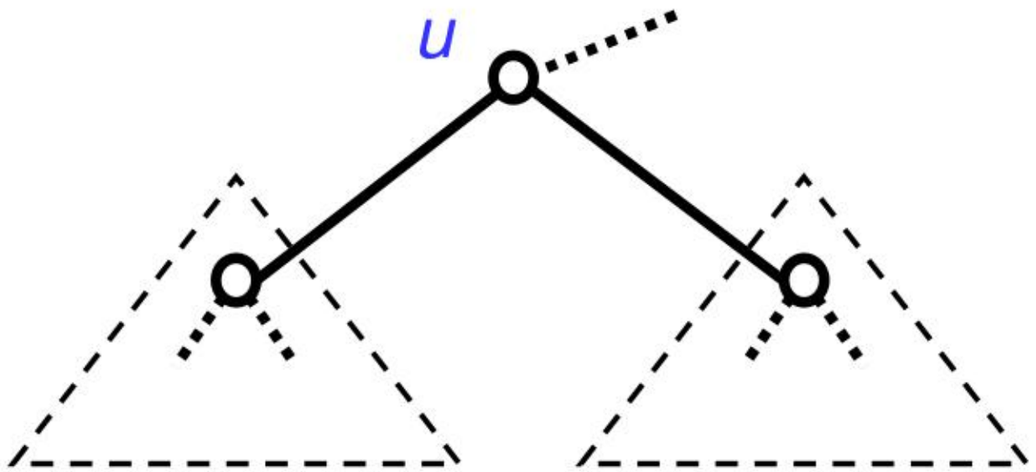
Hloubku prázdného stromu definujeme jako $-1$.

# Binární strom



Počet potomků každého uzlu je 0,1, nebo 2.

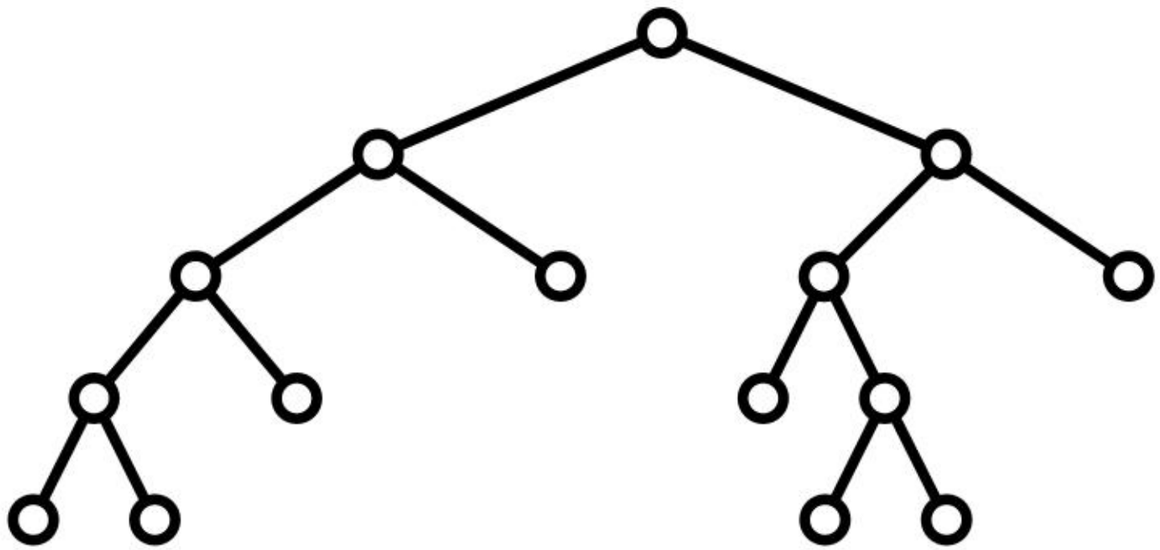
Levý a pravý podstrom (left and right subtree):




Podstrom uzlu *u* ..... levý ..... pravý

# Binární strom

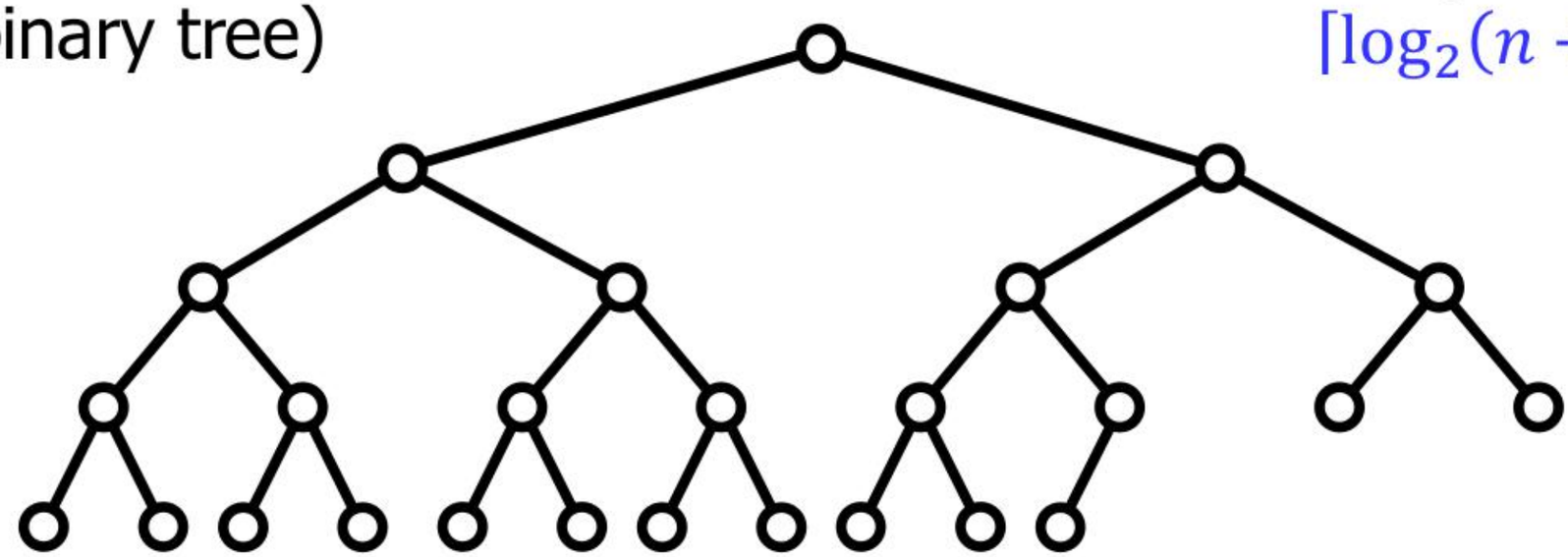
Pravidelný binární strom
(regular binary tree)



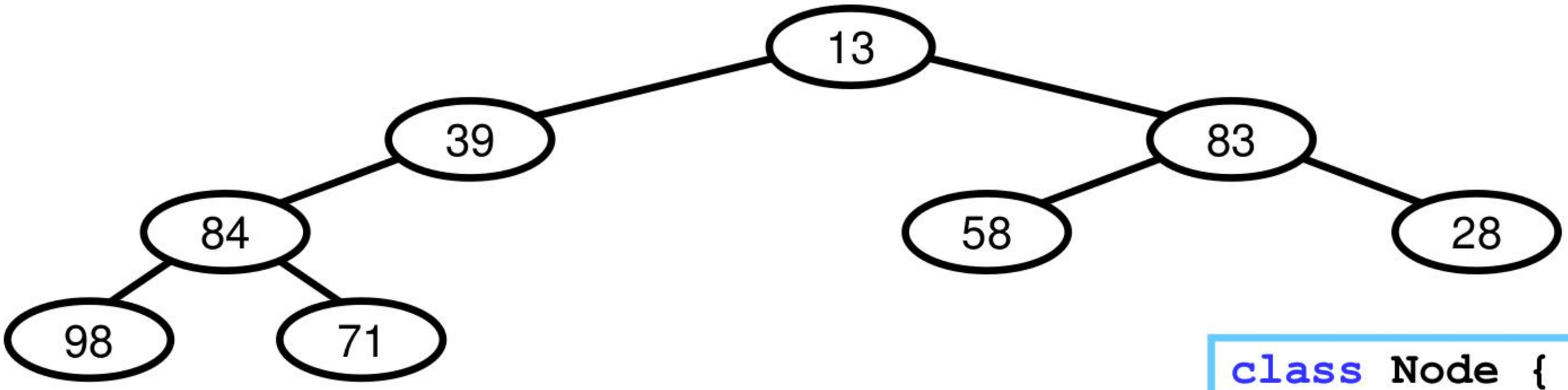
Počet potomků každého uzlu je jen 0 nebo 2.

Vyvážený binární strom
(balanced binary tree)

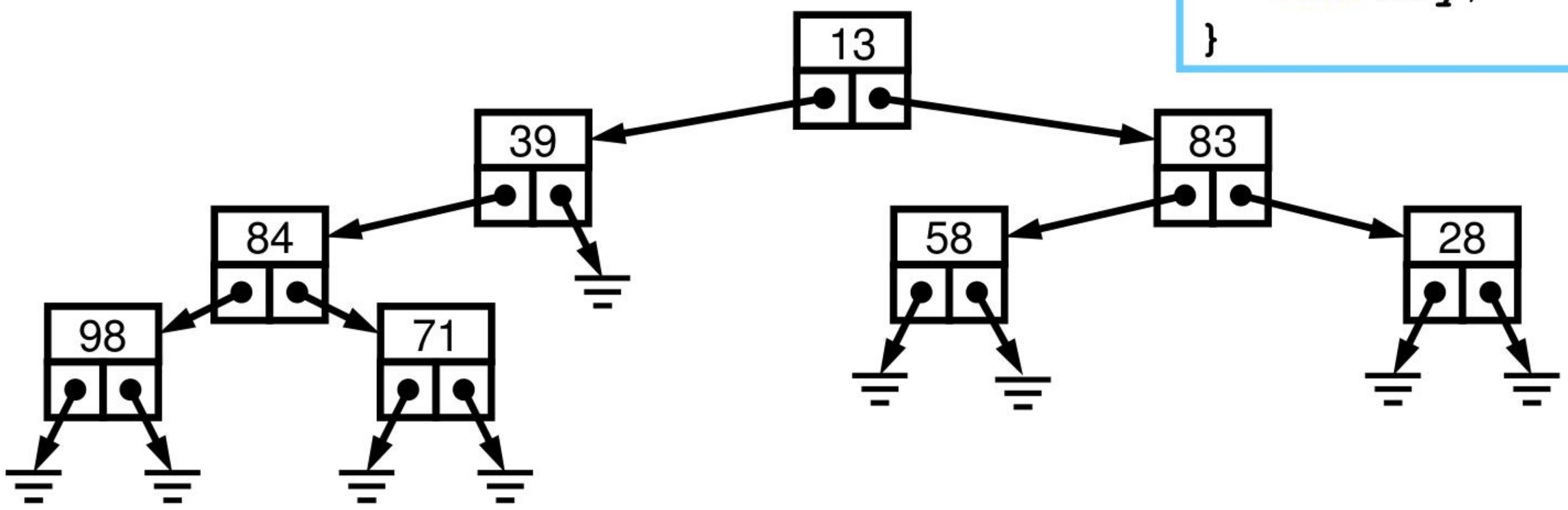
hloubka pro $n$ uzlů je
$\lceil \log_2(n+1) \rceil - 1$



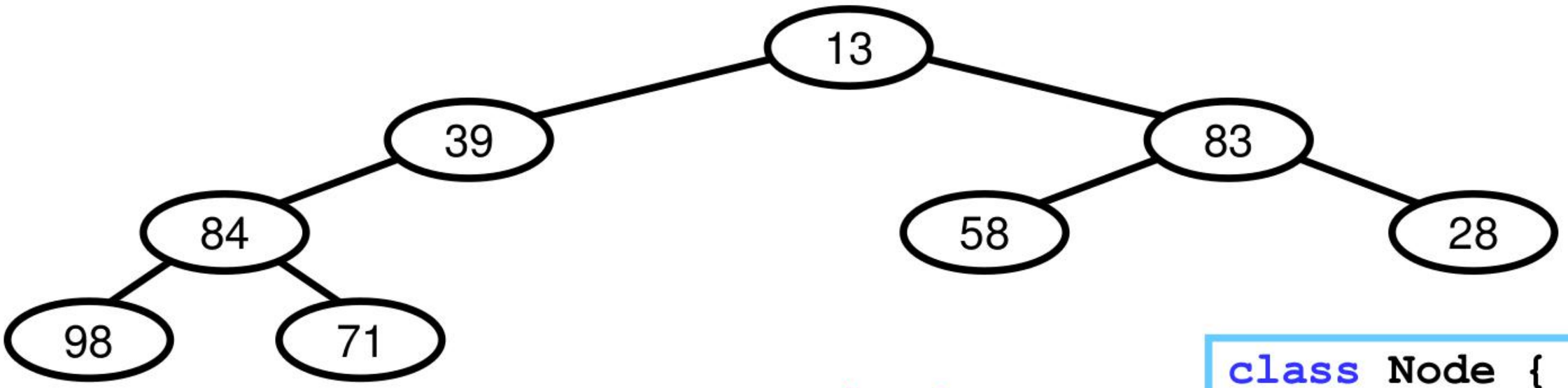
# Reprezentace binárního stromu




```
class Node {
    Node left;
    Node right;
    int key;
}
```

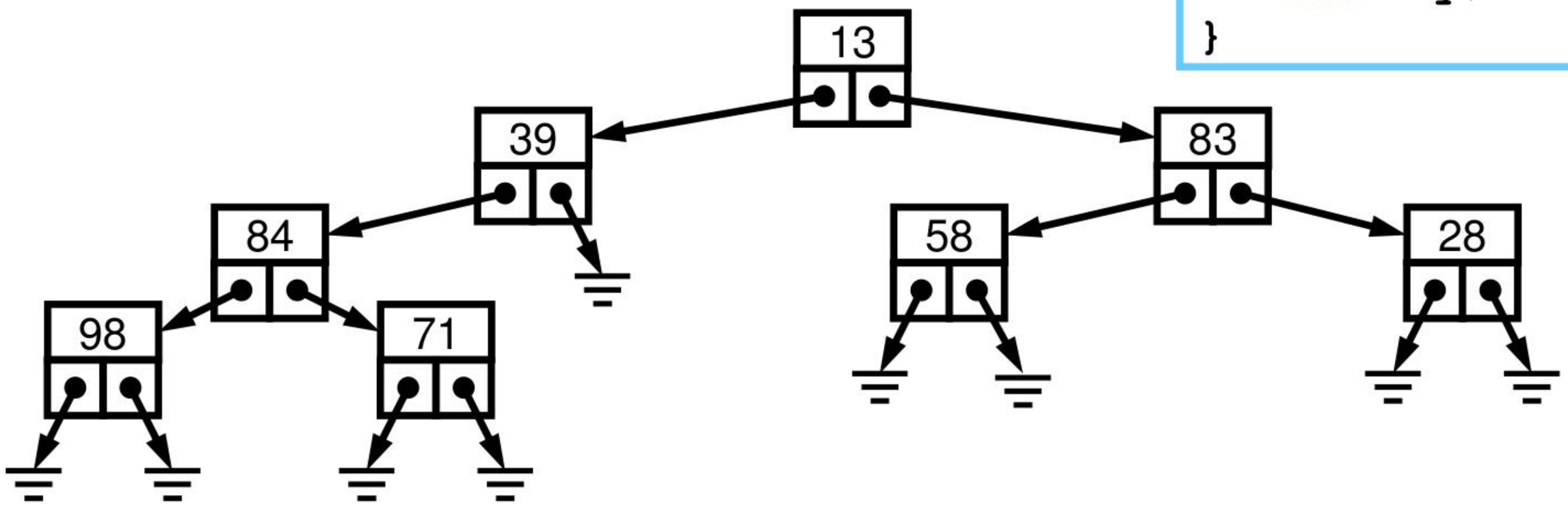


# Reprezentace binárního stromu




```
class Node {
    Node left;
    Node right;
    int key;
}
```

Node parent;

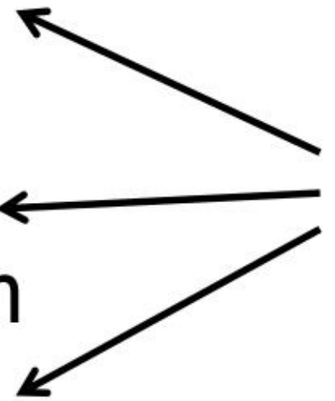


# Průchod stromem do hloubky

Začni v kořeni stromu a opakuj:

Pro aktuální uzel *u*

- projdi rekurzivně levý podstrom
- projdi rekurzivně pravý podstrom



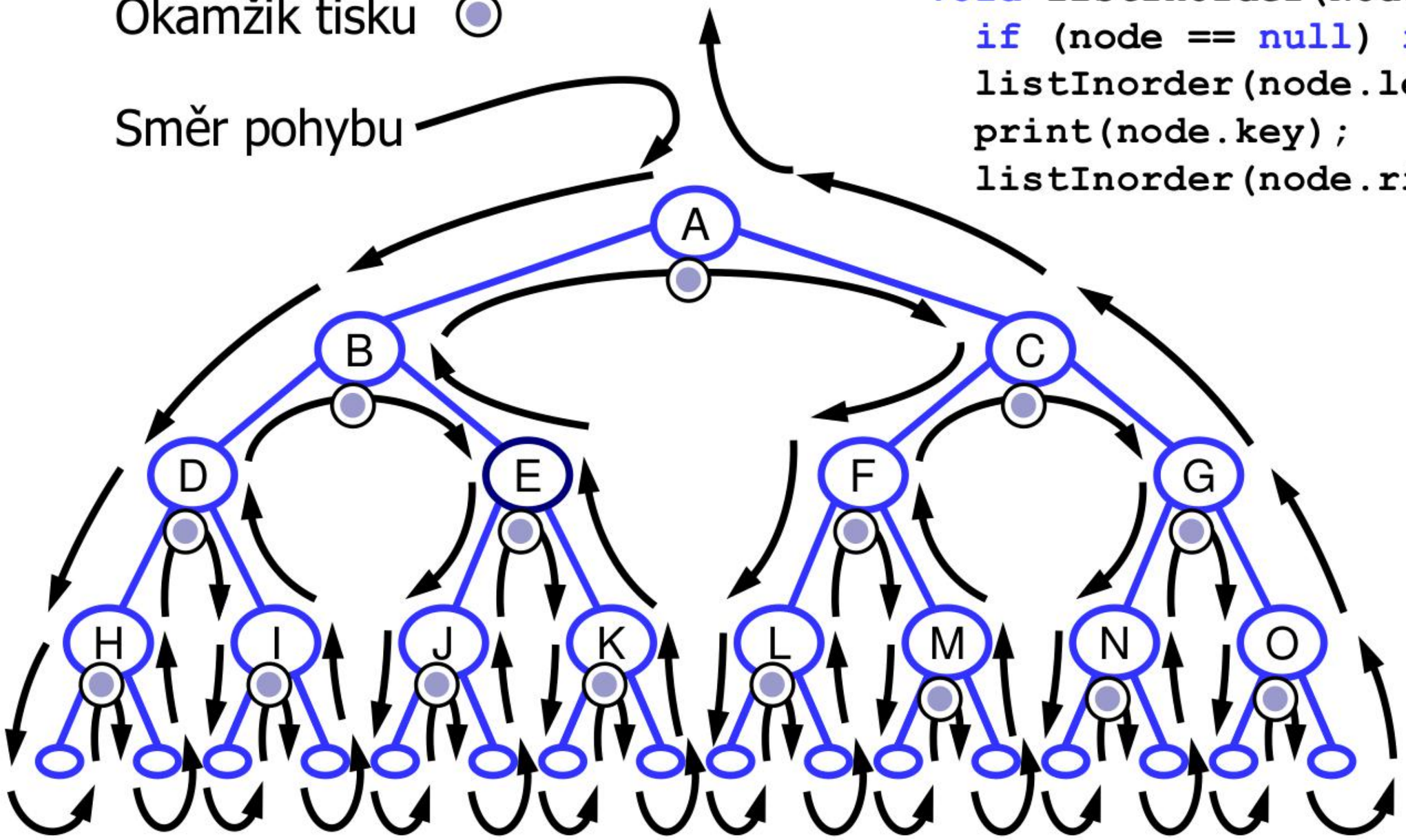
zpracuj uzel *u*
(je více možností, kdy to provést)

# Průchod v pořadí Inorder

Okamžik tisku

Směr pohybu

```
void listInorder(Node node):
  if (node == null) return;
  listInorder(node.left);
  print(node.key);
  listInorder(node.right);
```



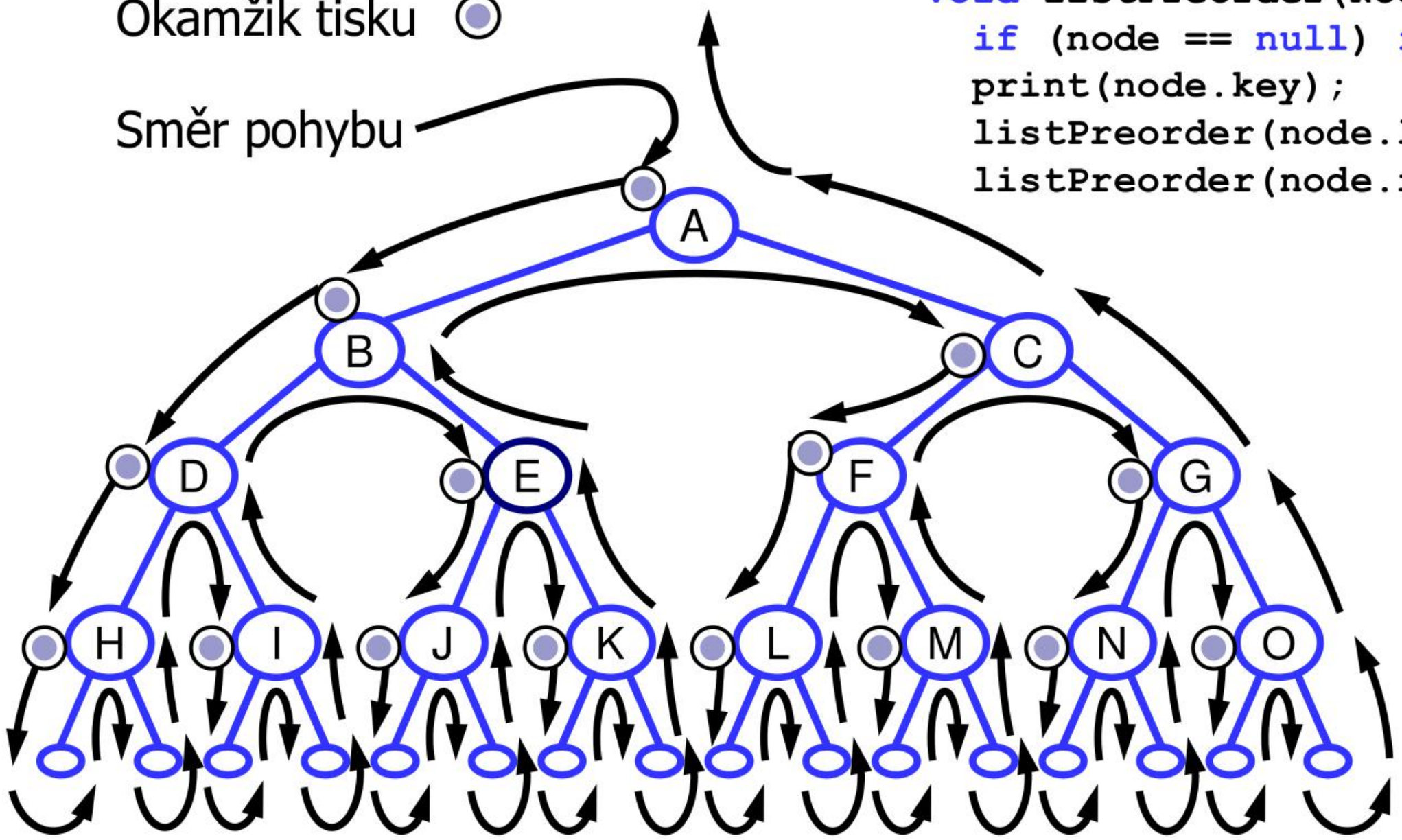

Výstup

H D I B J E K A L F M C N G O

# Průchod v pořadí Preorder

Okamžik tisku

Směr pohybu




```
void listPreorder(Node node):
  if (node == null) return;
  print(node.key);
  listPreorder(node.left);
  listPreorder(node.right);
```

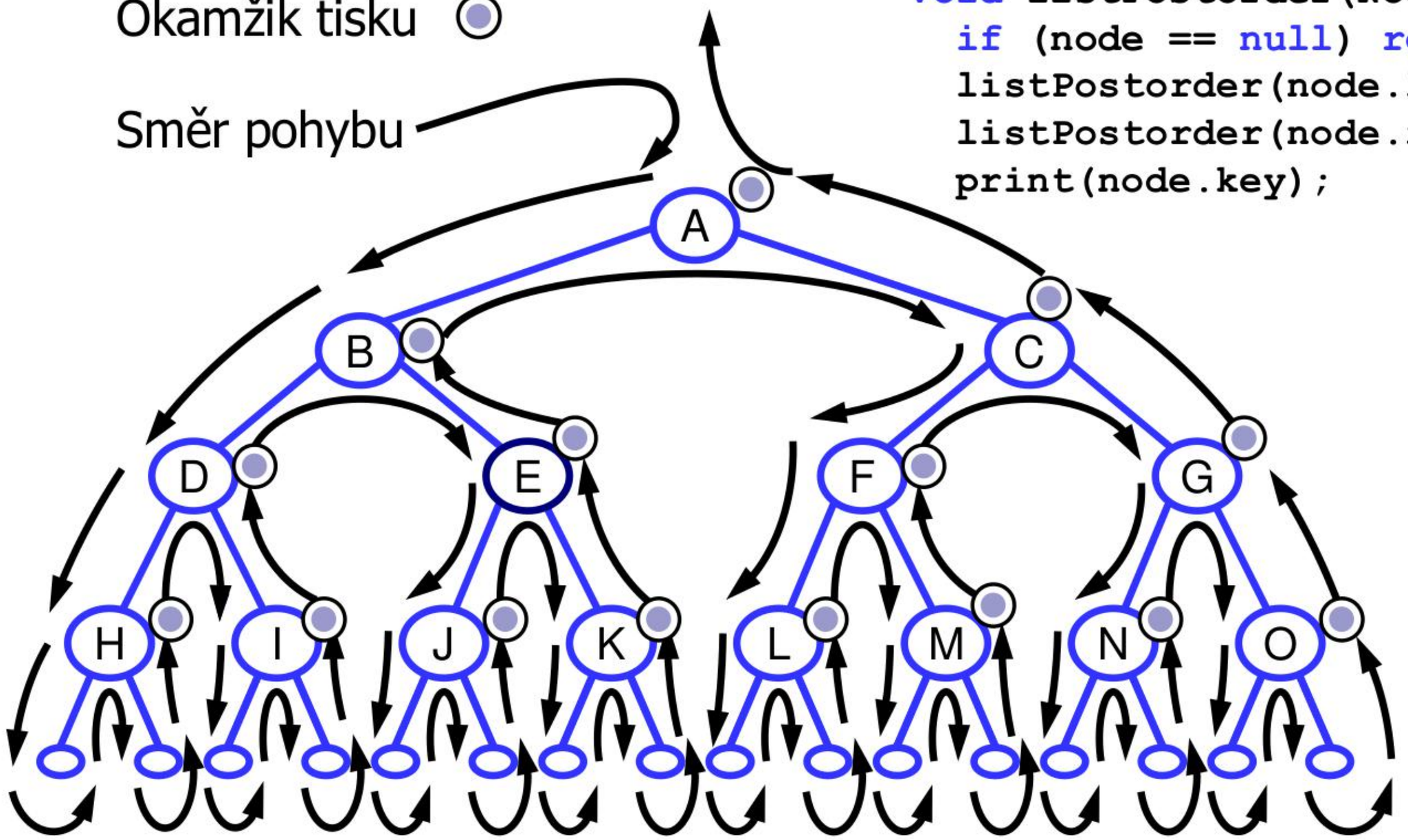
Výstup

A B D H I E J K C F L M G N O

# Průchod v pořadí Postorder

Okamžik tisku

Směr pohybu




```
void listPostorder(Node node):
  if (node == null) return;
  listPostorder(node.left);
  listPostorder(node.right);
  print(node.key);
```

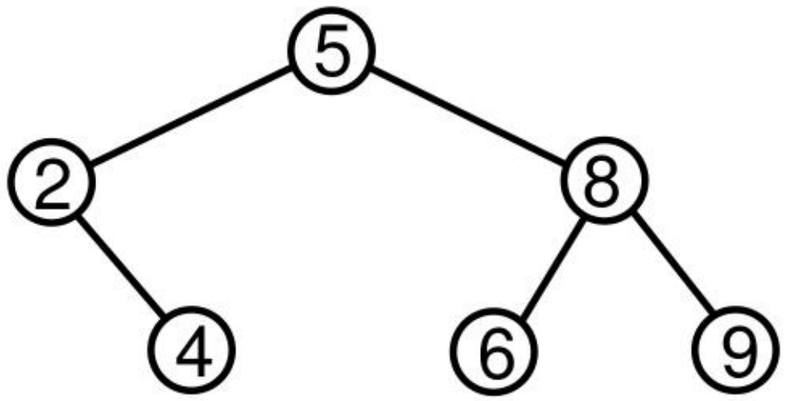
Výstup

H I D J K E B L M F N O G C A

# Zásobník implementuje rekurzi

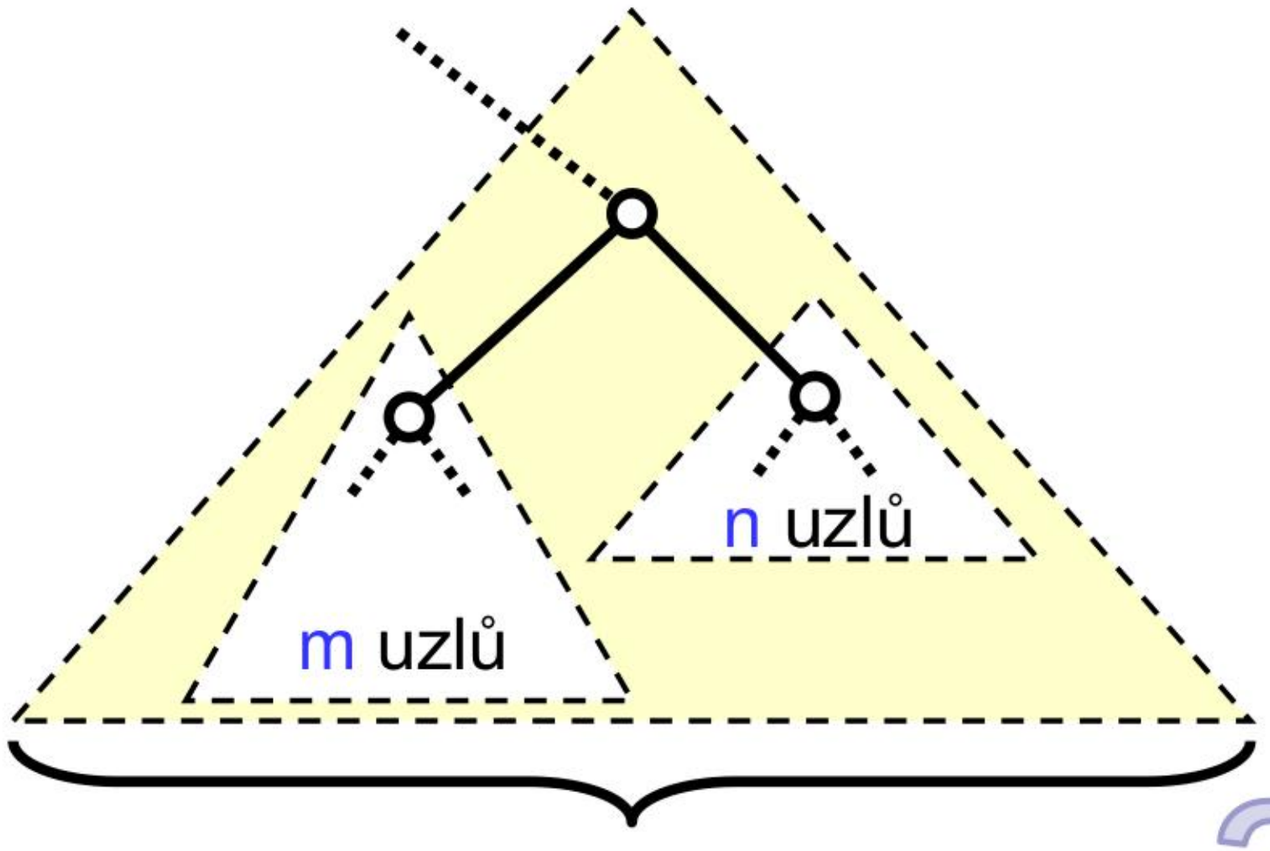
```
void inorderIterative(Node root) {
    Stack<Node> stack = new Stack();
    Node curr = root;
    while (!stack.empty() || curr != null) {
       if (curr != null) {
           stack.push(curr);
           curr = curr.left;
        } else {
           curr = stack.pop();
           System.out.print(curr.key + " ");
           curr = curr.right;
       }
    }
}
```

Uzel uložíme na zásobník v okamžiku jeho objevení a odebereme jej po zpracování jeho levého podstromu.



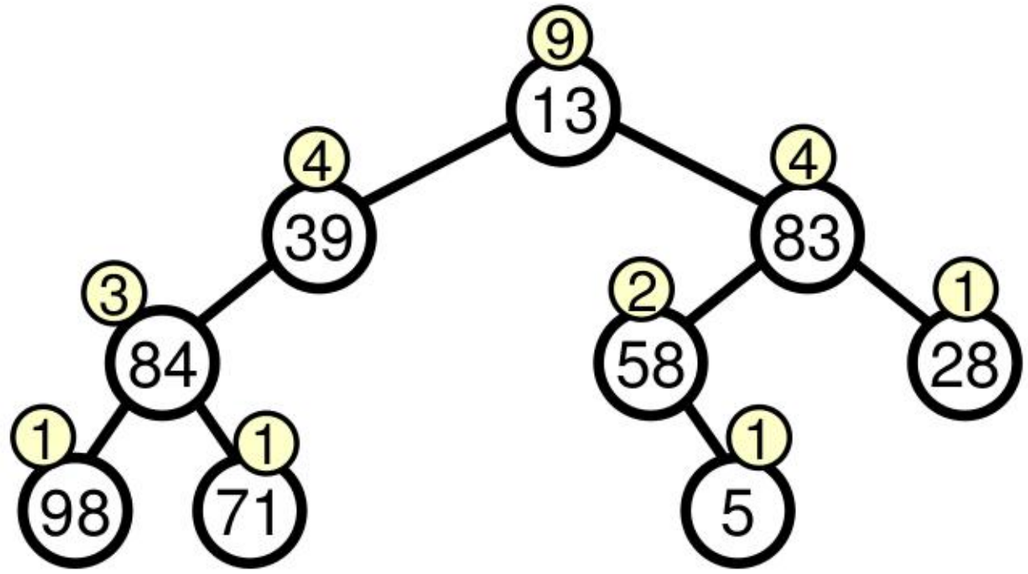
# Počítání uzlů

Strom nebo podstrom




celkem ... m+n+1 uzlů

Příklad

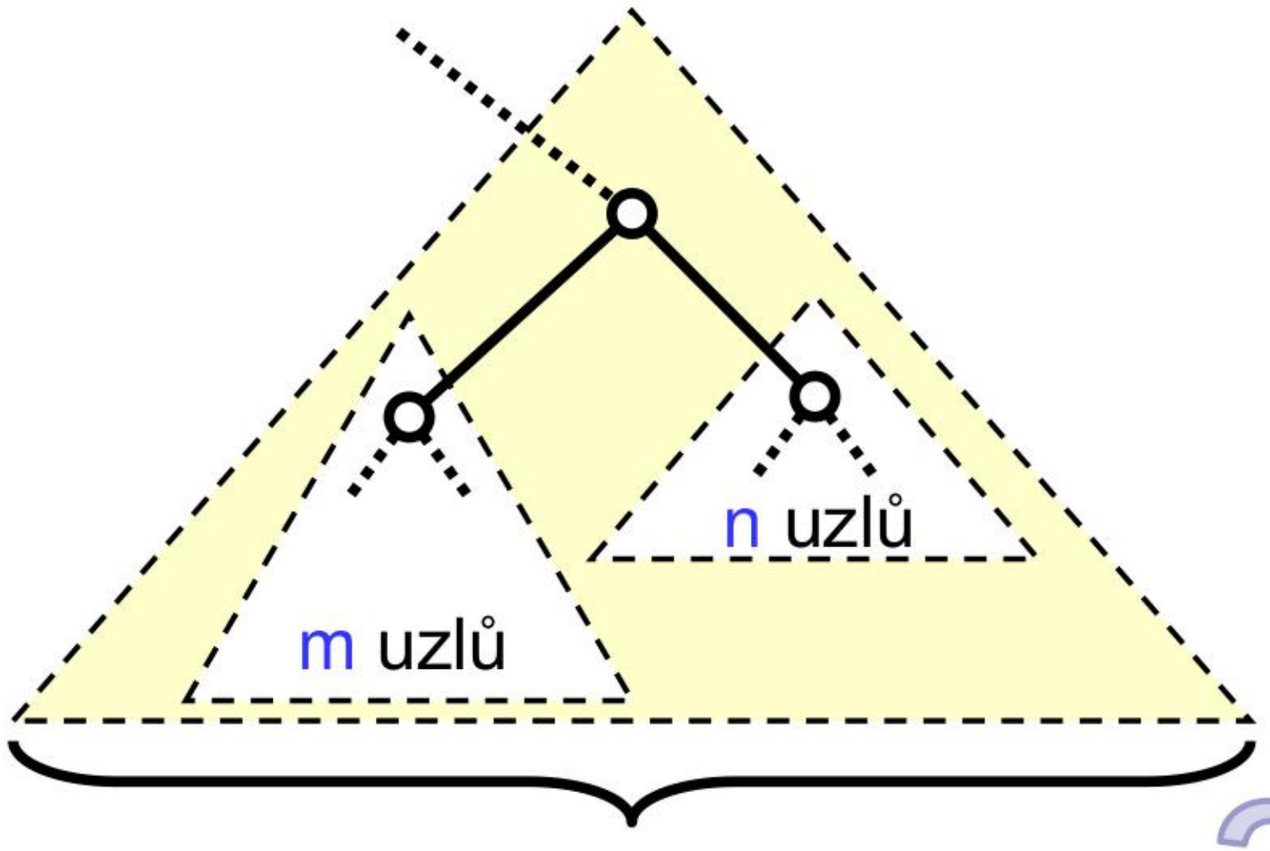



Je to průchod v pořadí preorder, inorder nebo postorder?

```
int count(Node node) {
  if (node == null) return 0;
  return (count(node.left) + count(node.right) + 1);
}
```

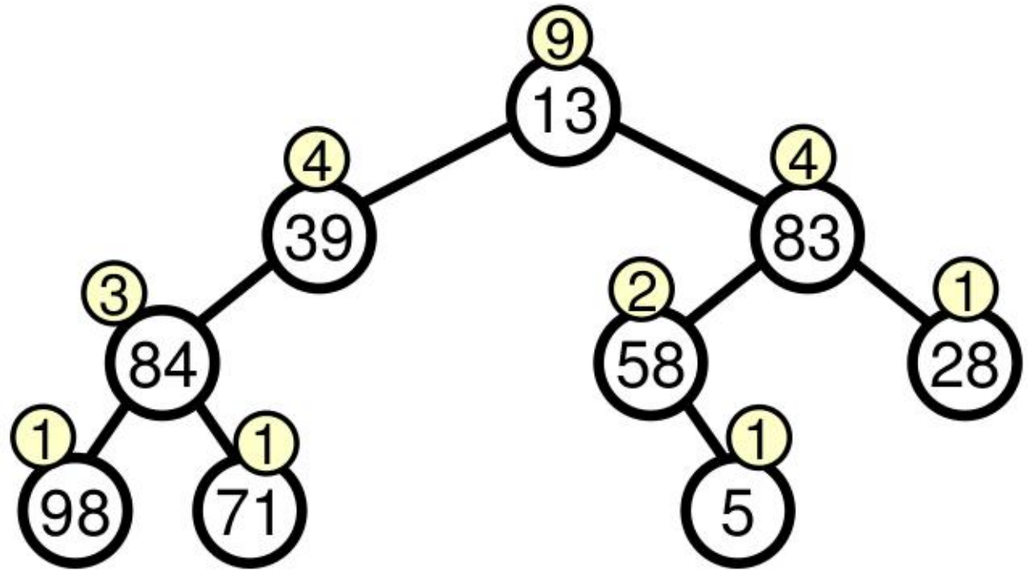
# Počítání uzlů

Strom nebo podstrom




celkem ... m+n+1 uzlů

Příklad

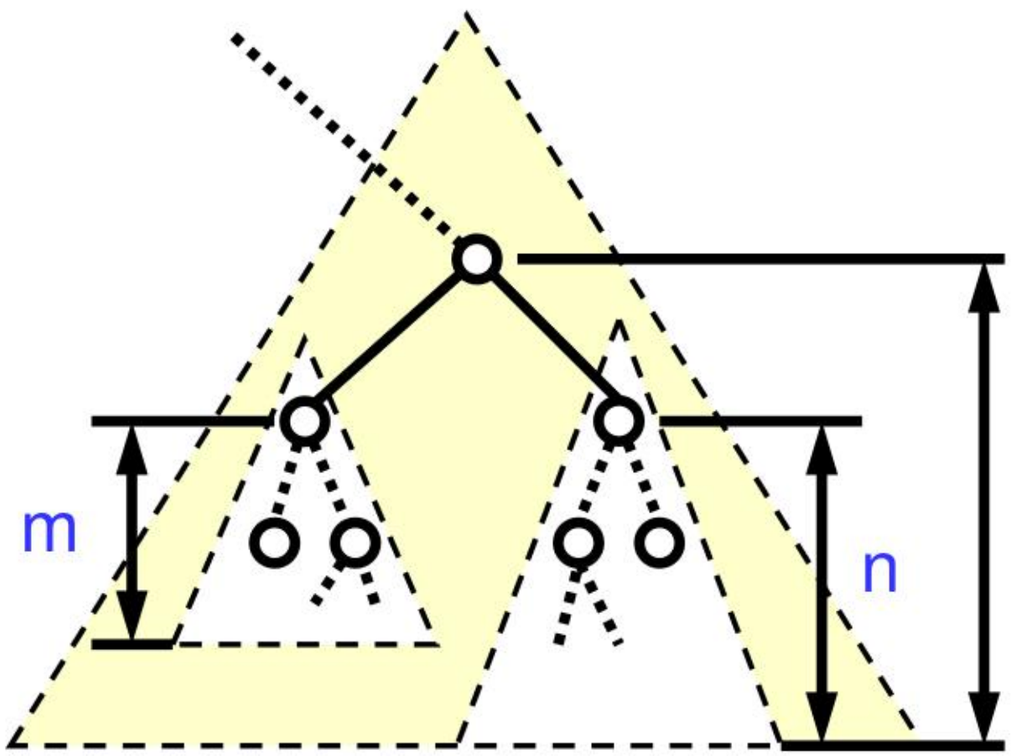


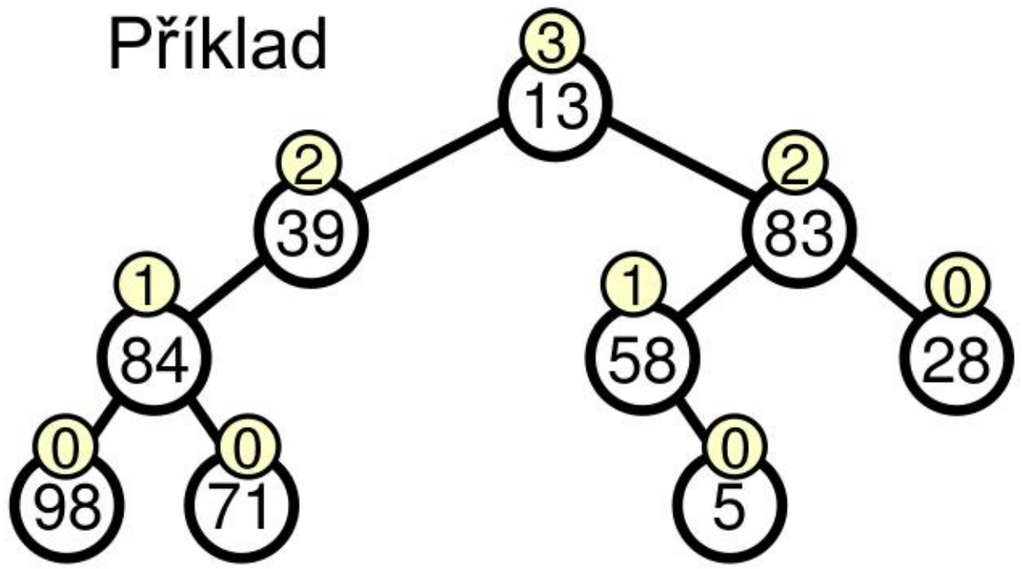

Je to průchod v pořadí preorder, inorder nebo postorder?

```
int count(Node node) {
  if (node == null) return 0;
  return (count(node.left) + count(node.right) + 1);
}
```

1+

# Počítání hloubky

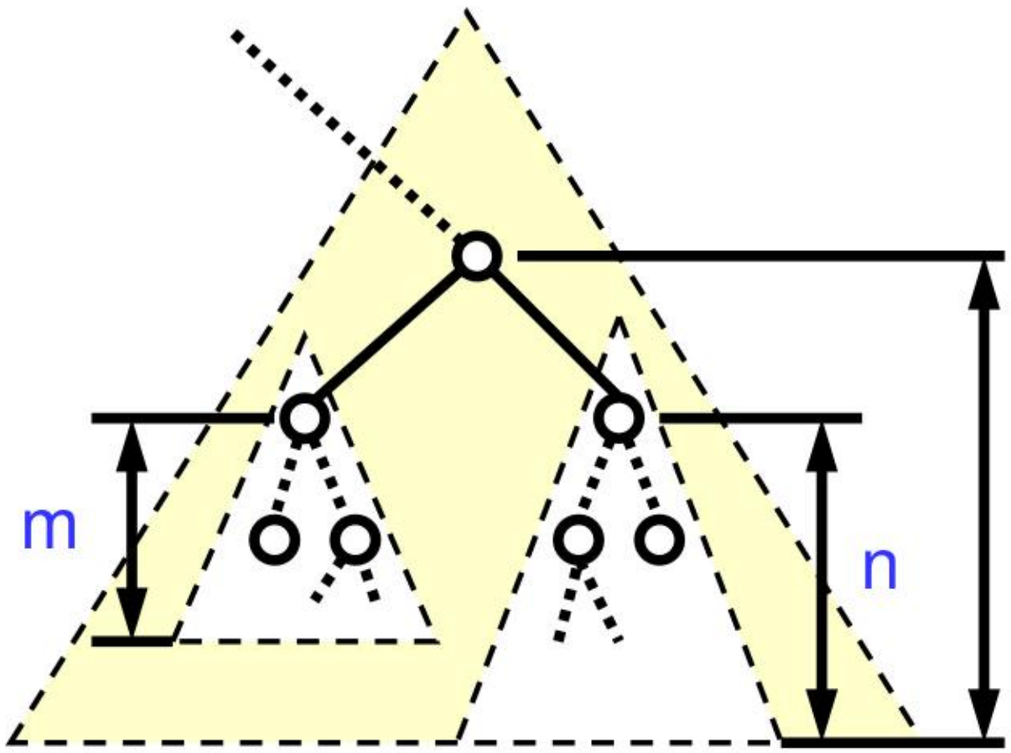


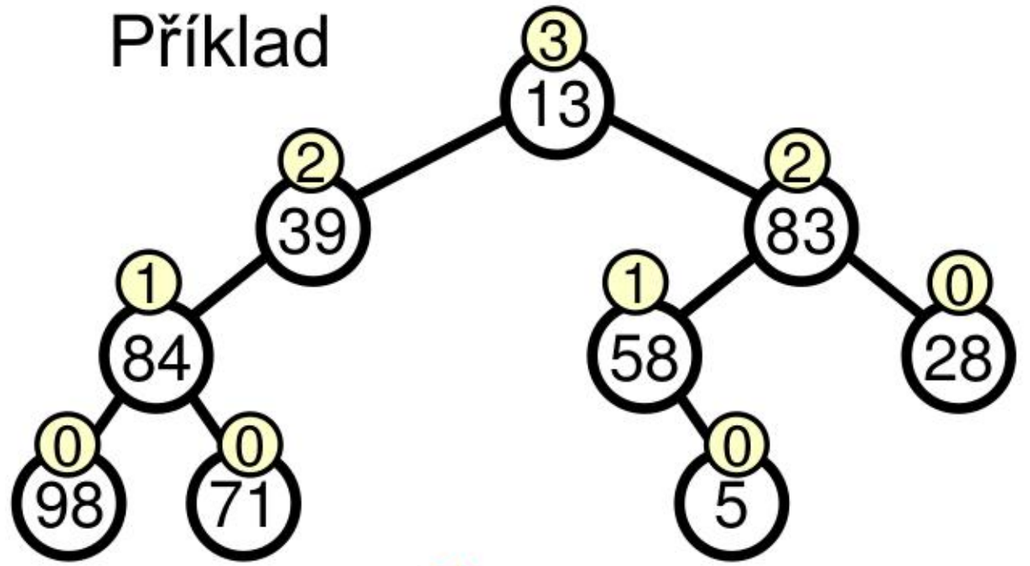

```
int depth(Node node) {
  if (node == null) return -1;
  return max(depth(node.left), depth(node.right))+1);
}
```

# Počítání hloubky

Strom nebo podstrom




Příklad




max(-1,-1)+1=0

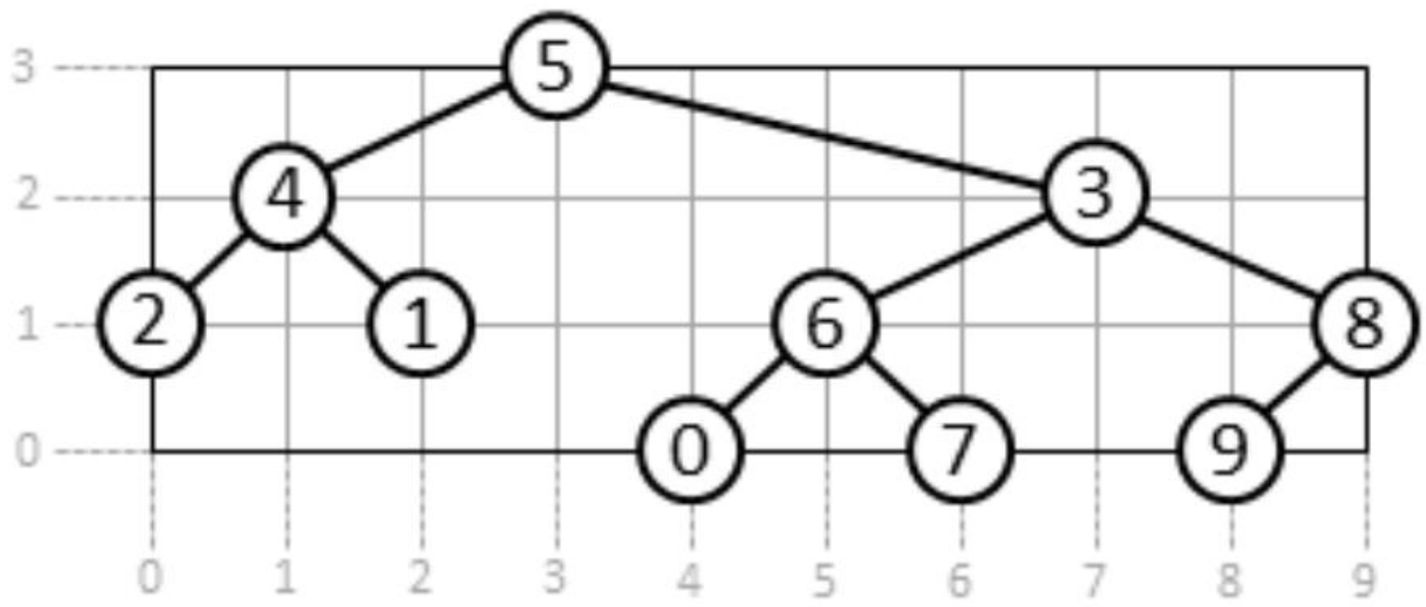
```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

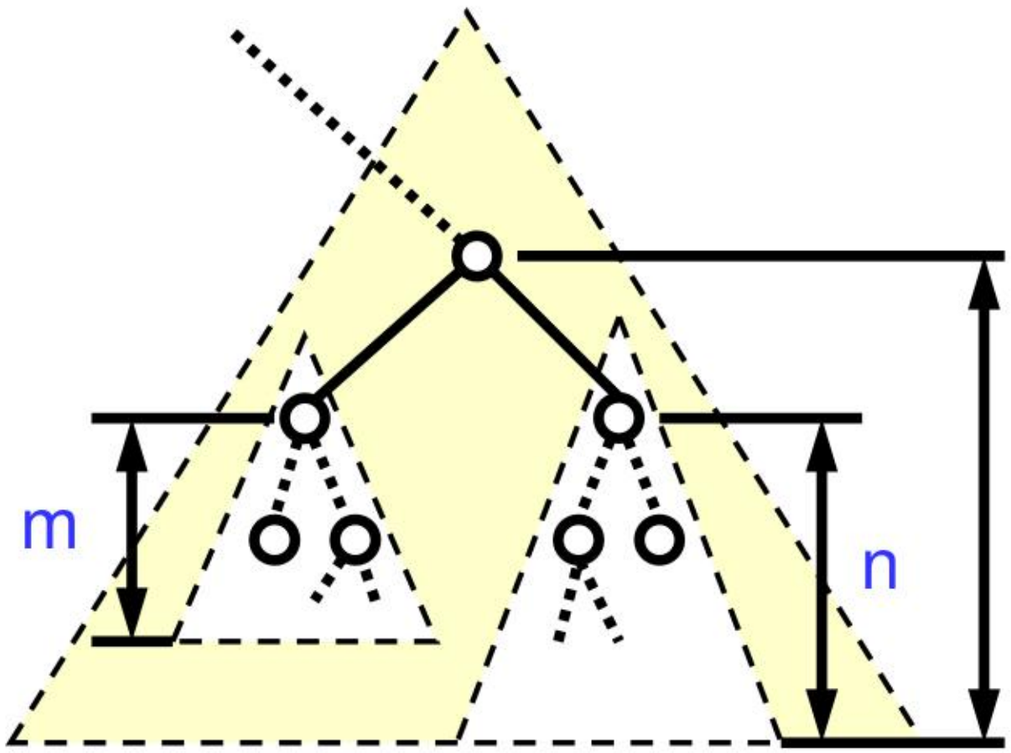
Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?

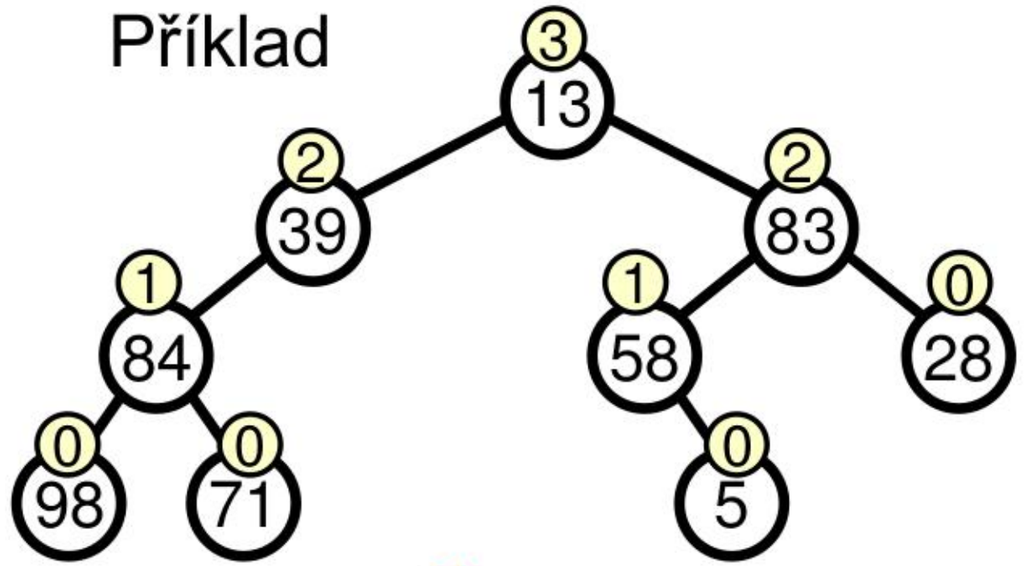


# Počítání hloubky

Strom nebo podstrom




Příklad

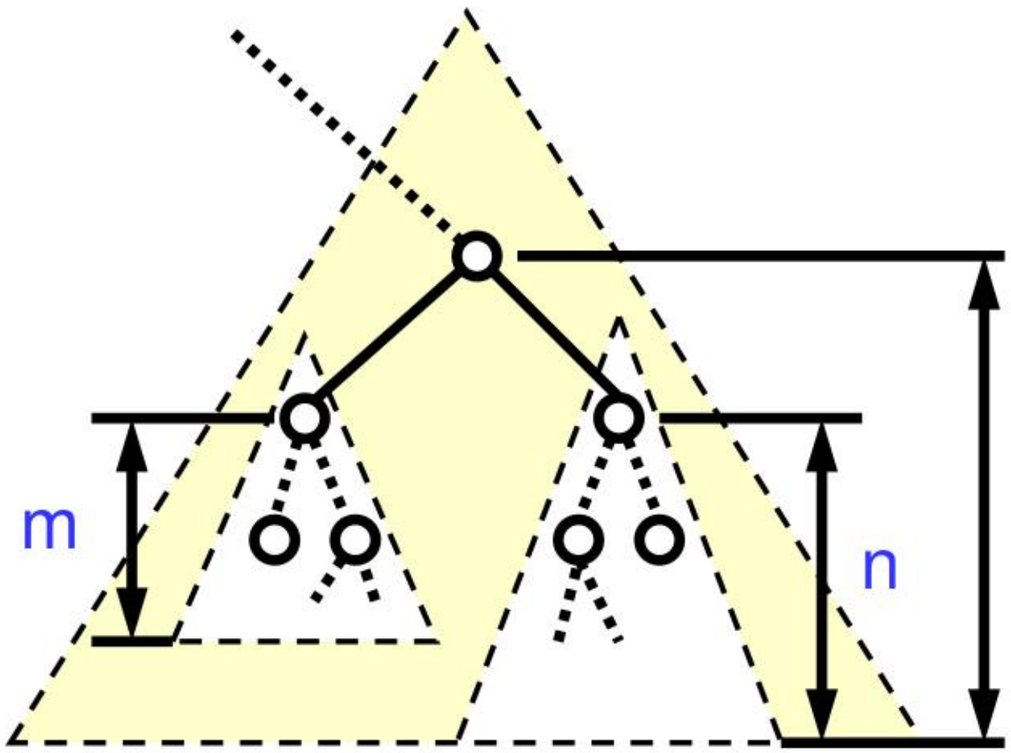



max(-1,-1)+1=0

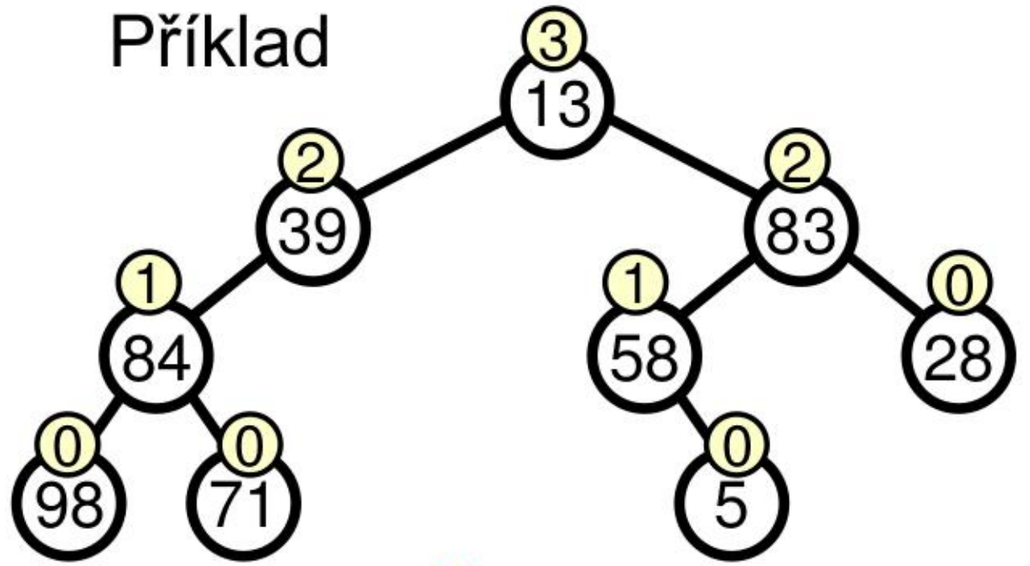
```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

# Počítání hloubky

Strom nebo podstrom




Příklad




max(-1,-1)+1=0

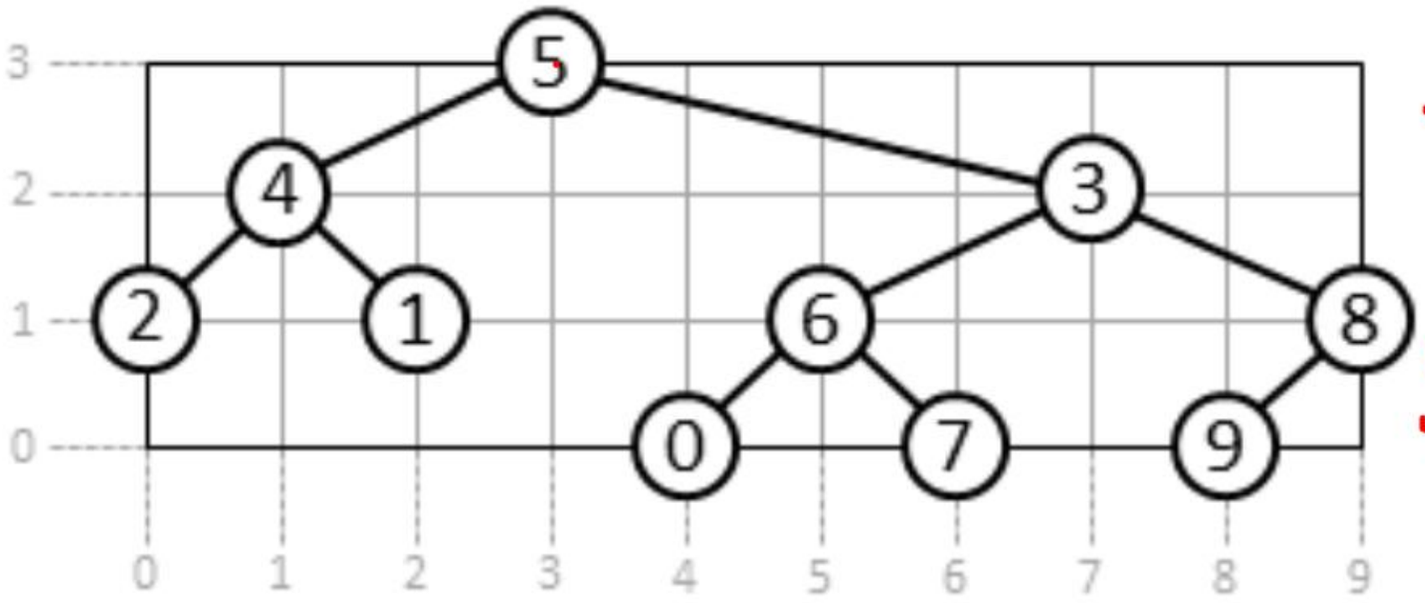
```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

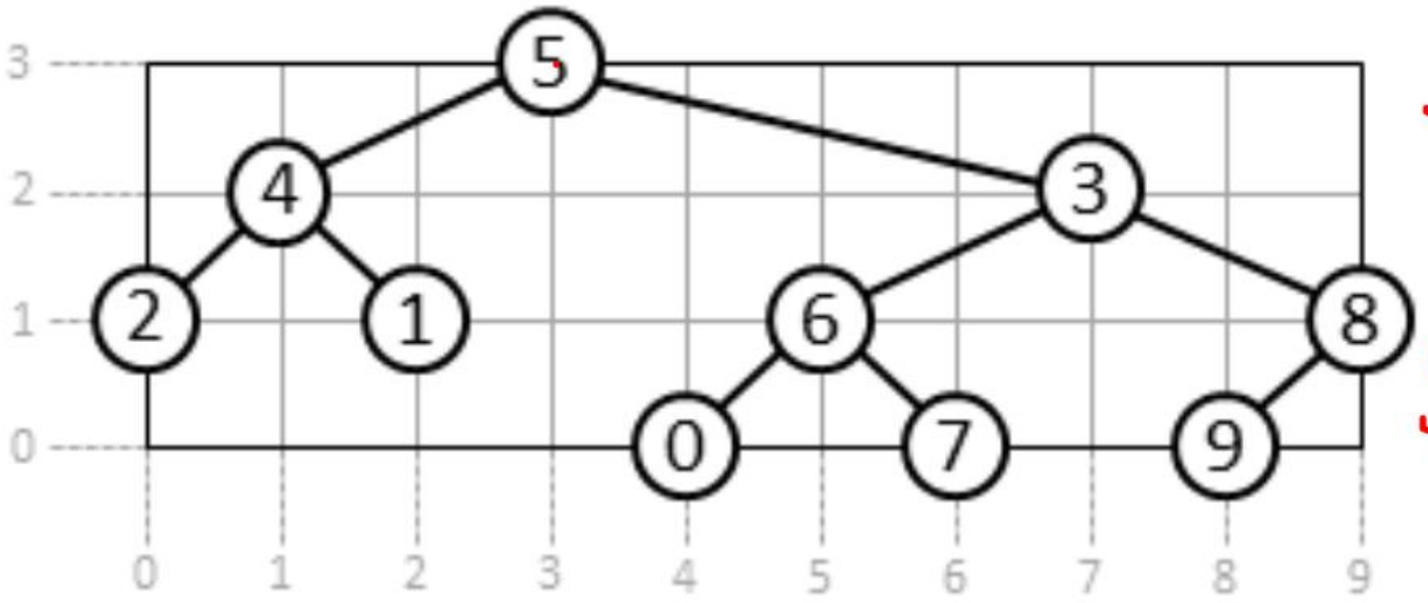
y: h(T) - h(u)

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?



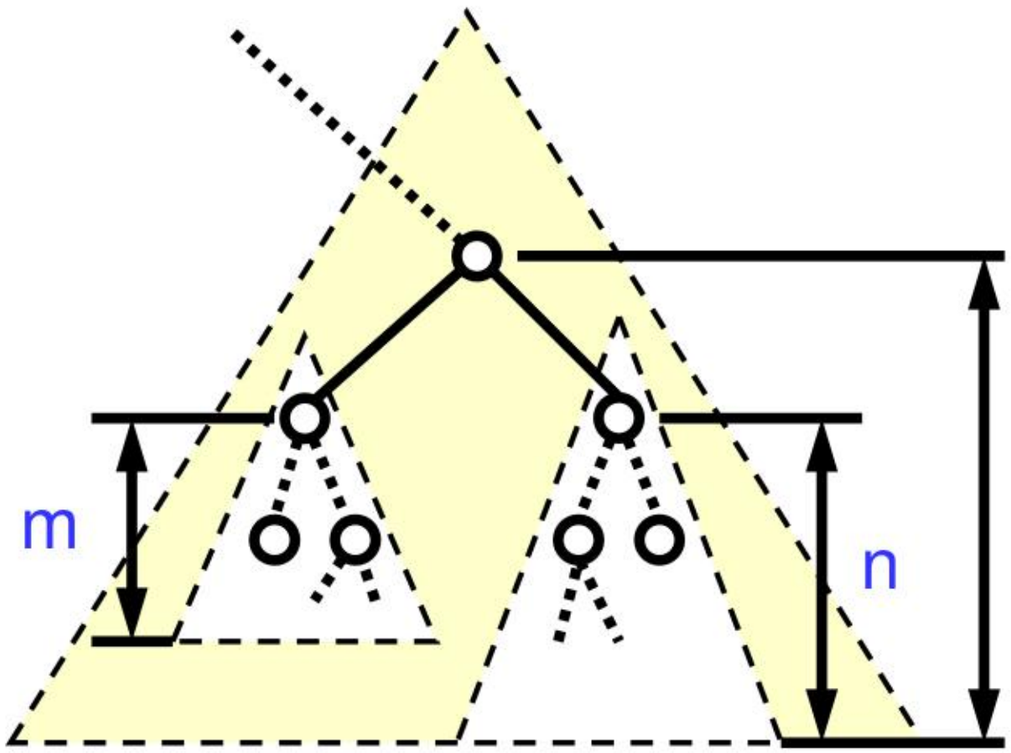

x: inorder(u)

y: h(T) − h(u)

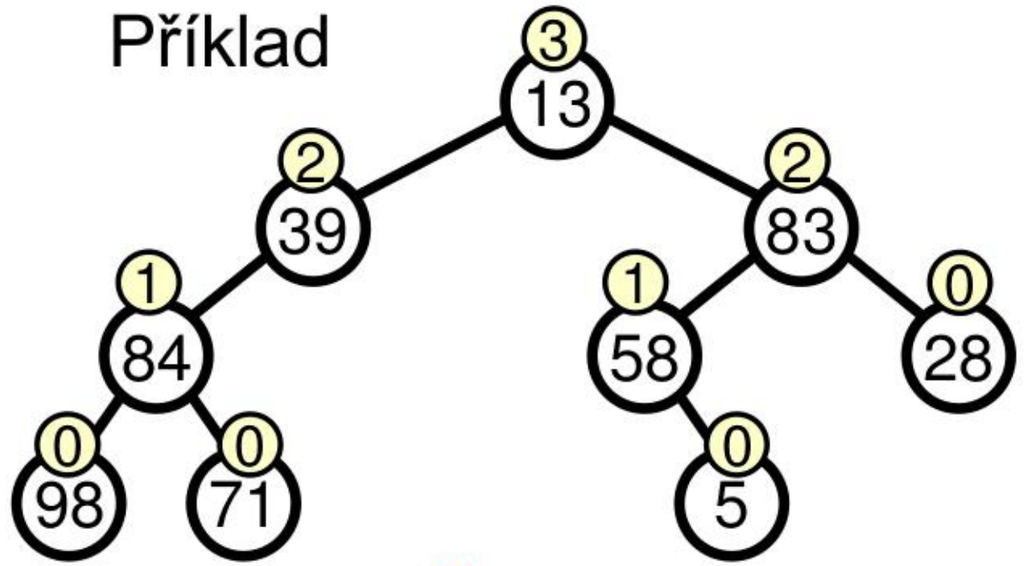
3 1

# Počítání hloubky

Strom nebo podstrom




Příklad



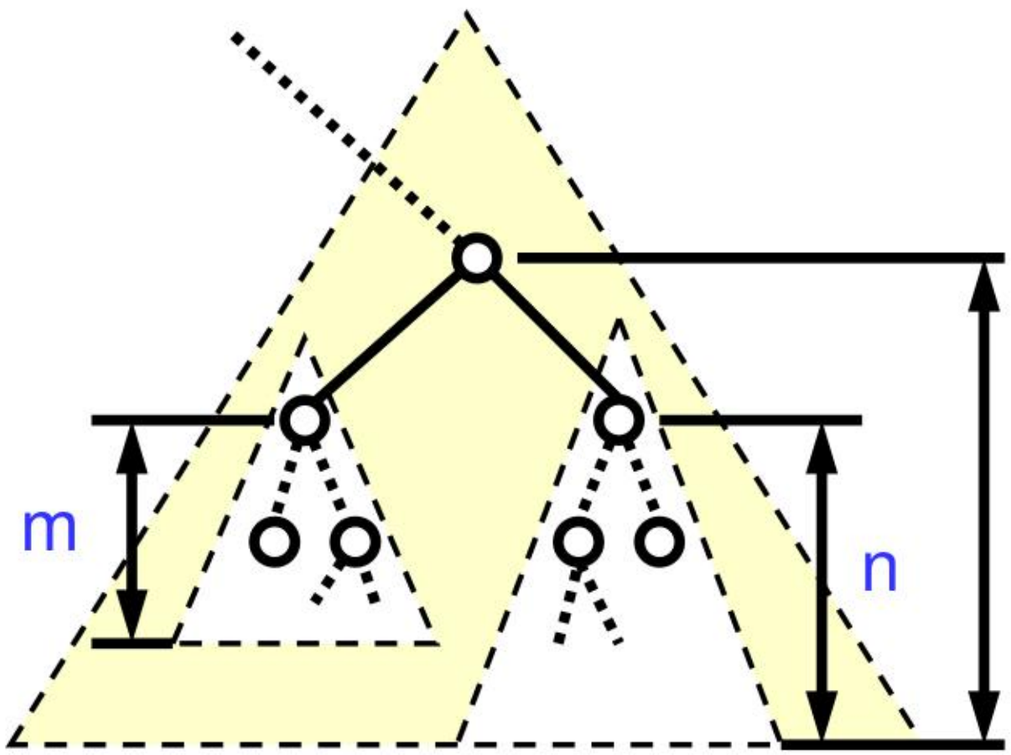

max(-1,-1)+1=0

```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

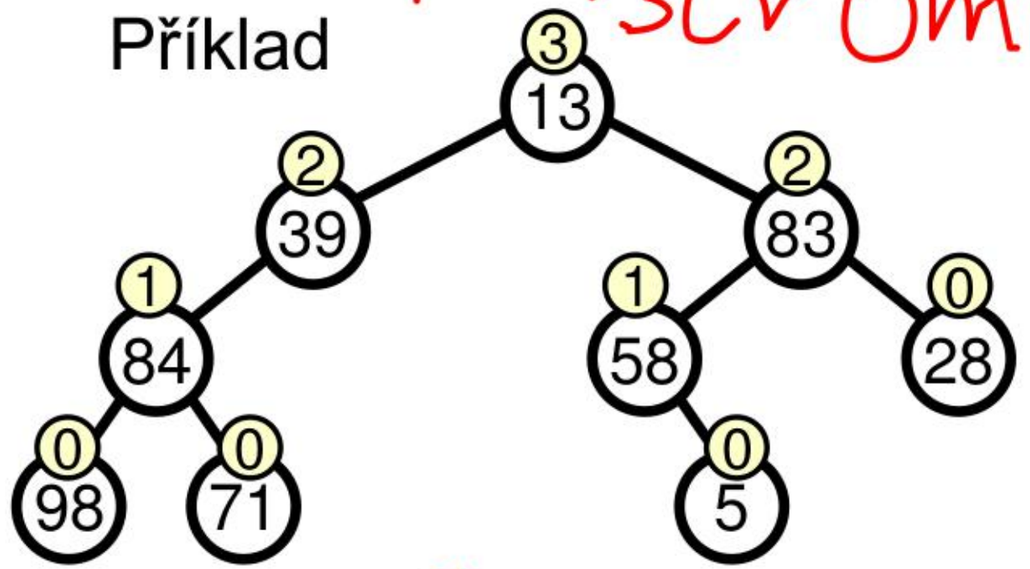
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad




0

max(-1,-1)+1=0

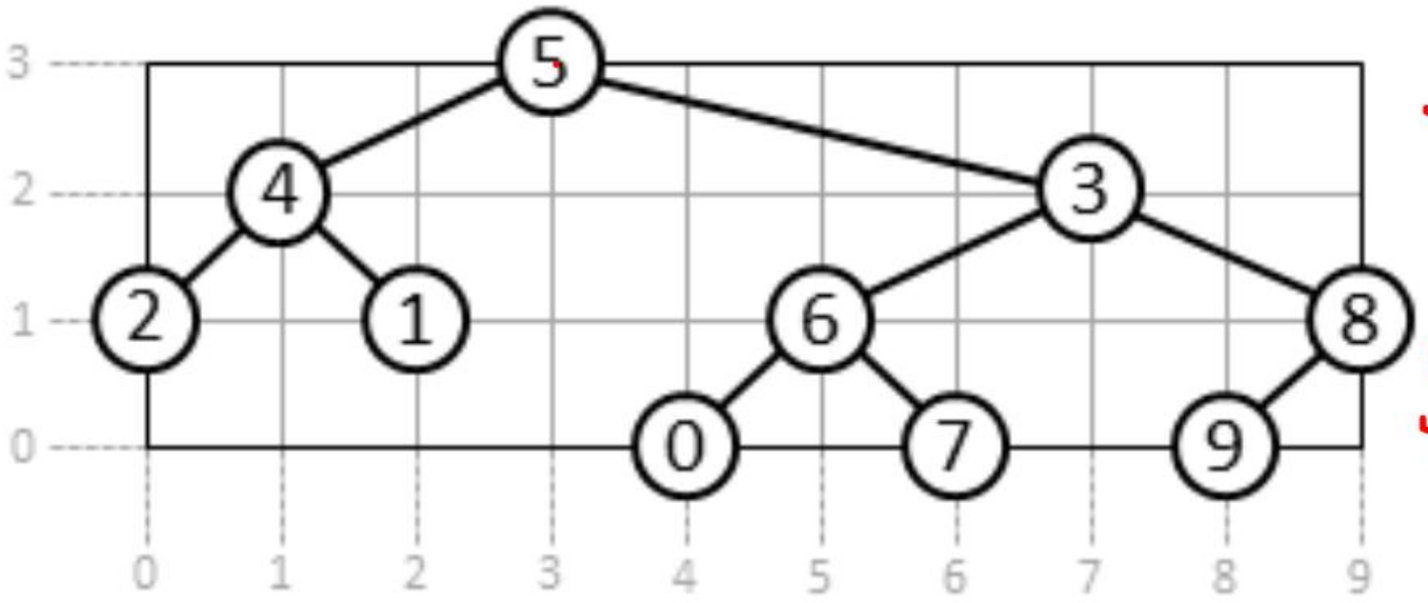
```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

y: h(T) − h(u)

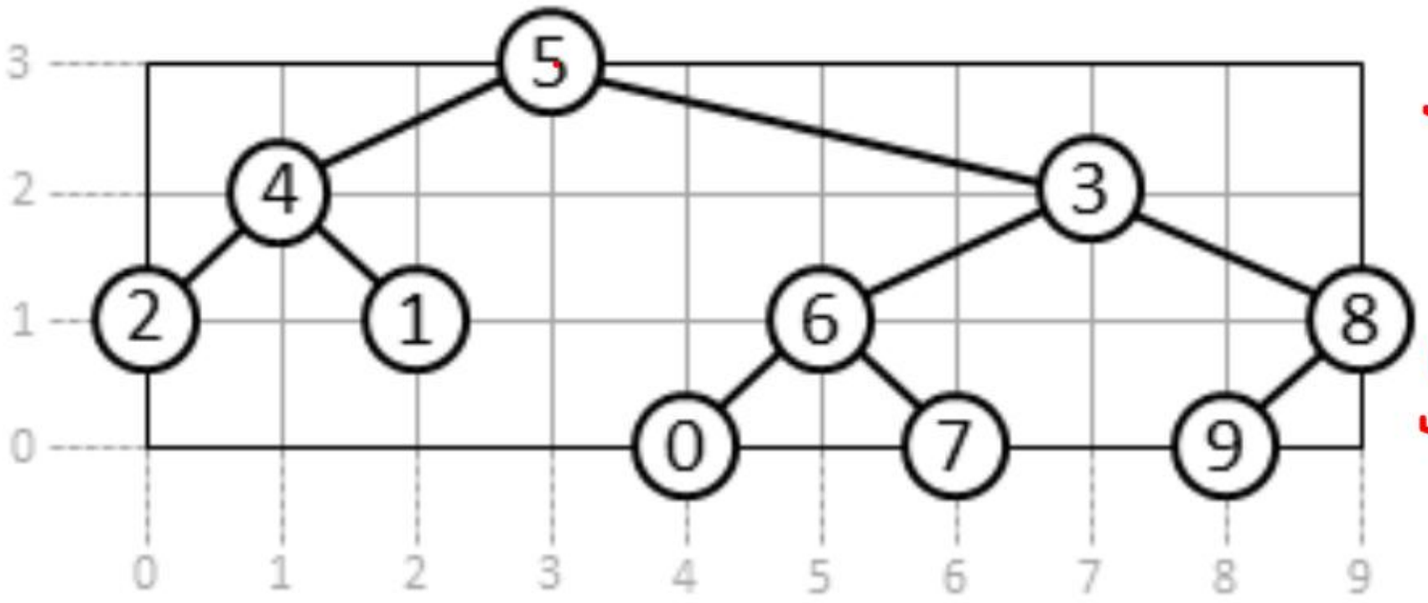
3 1

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

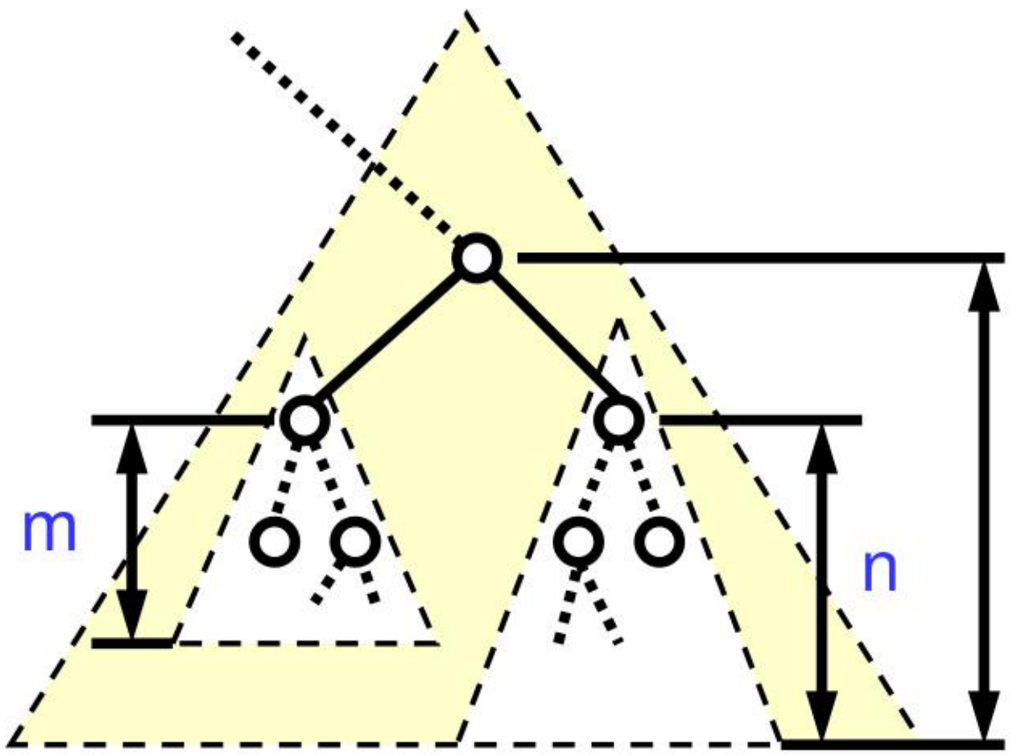
y: h(T) − h(u)

3 1

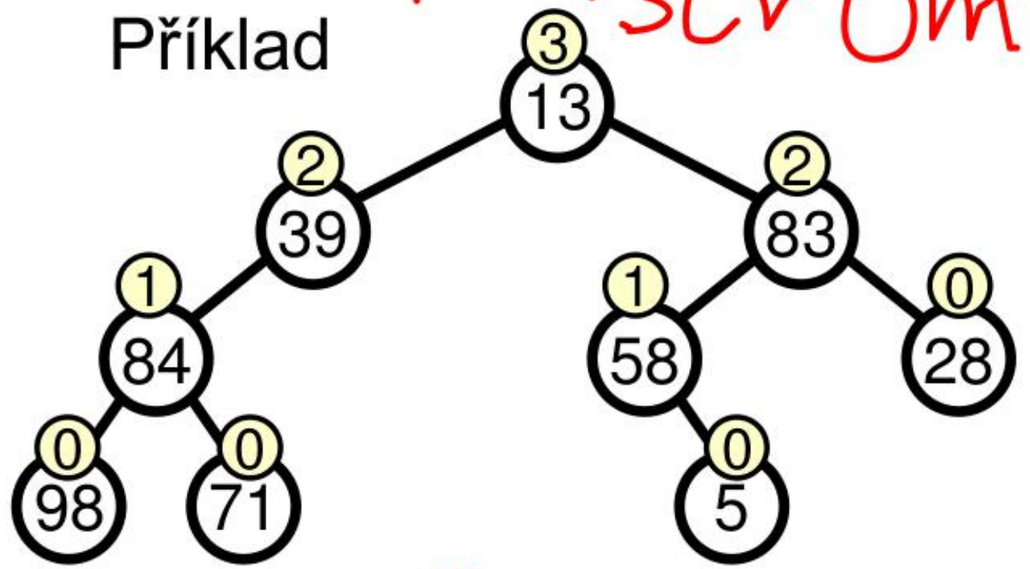
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad




0

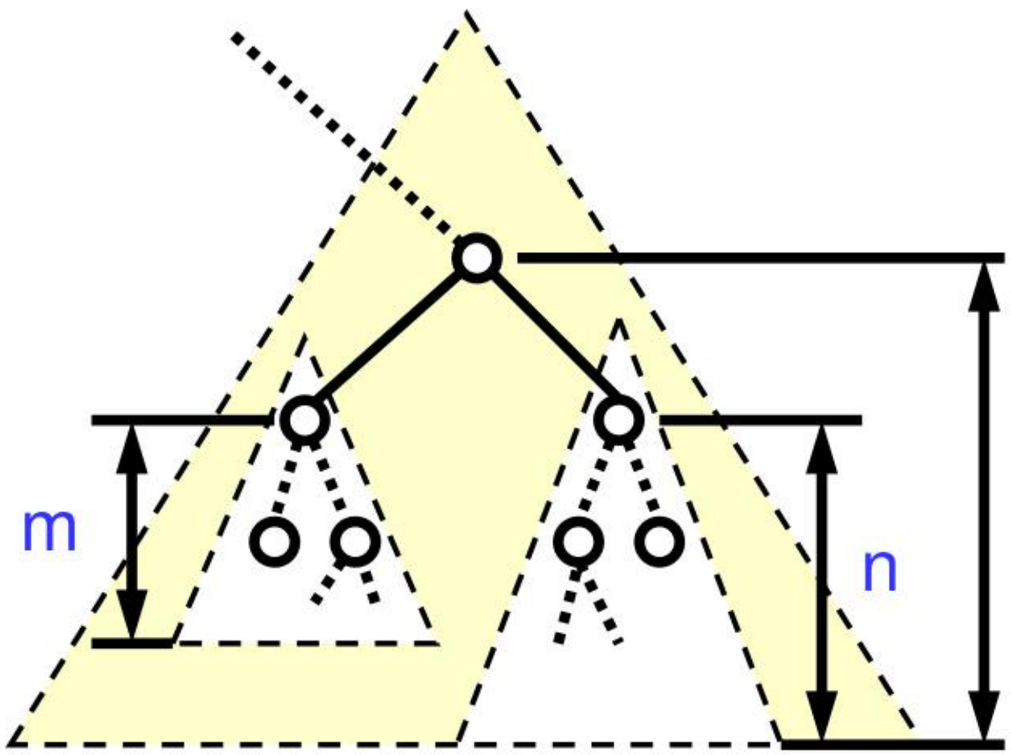
max(-1,-1)+1=0

```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

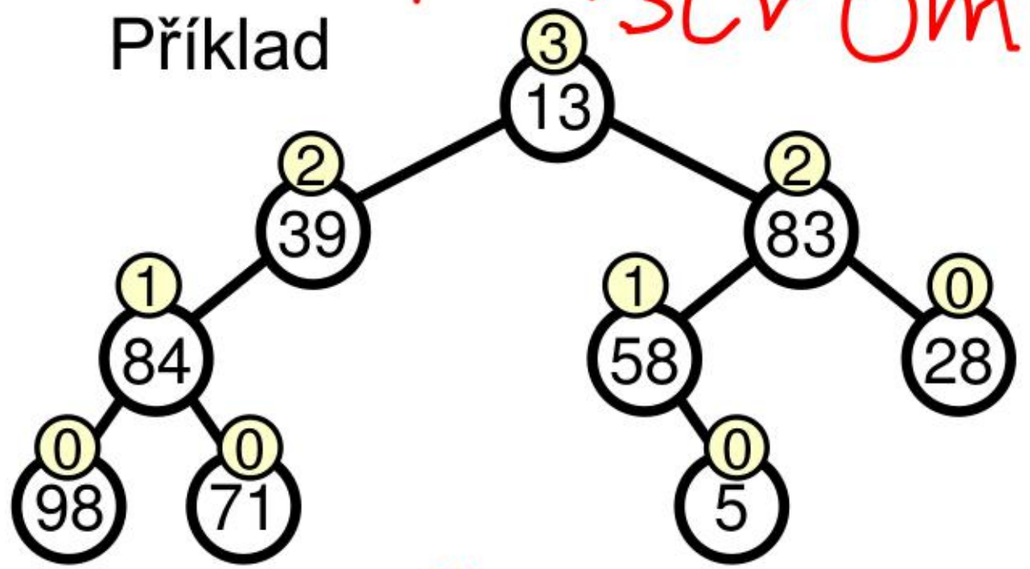
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad




0

max(-1,-1)+1=0

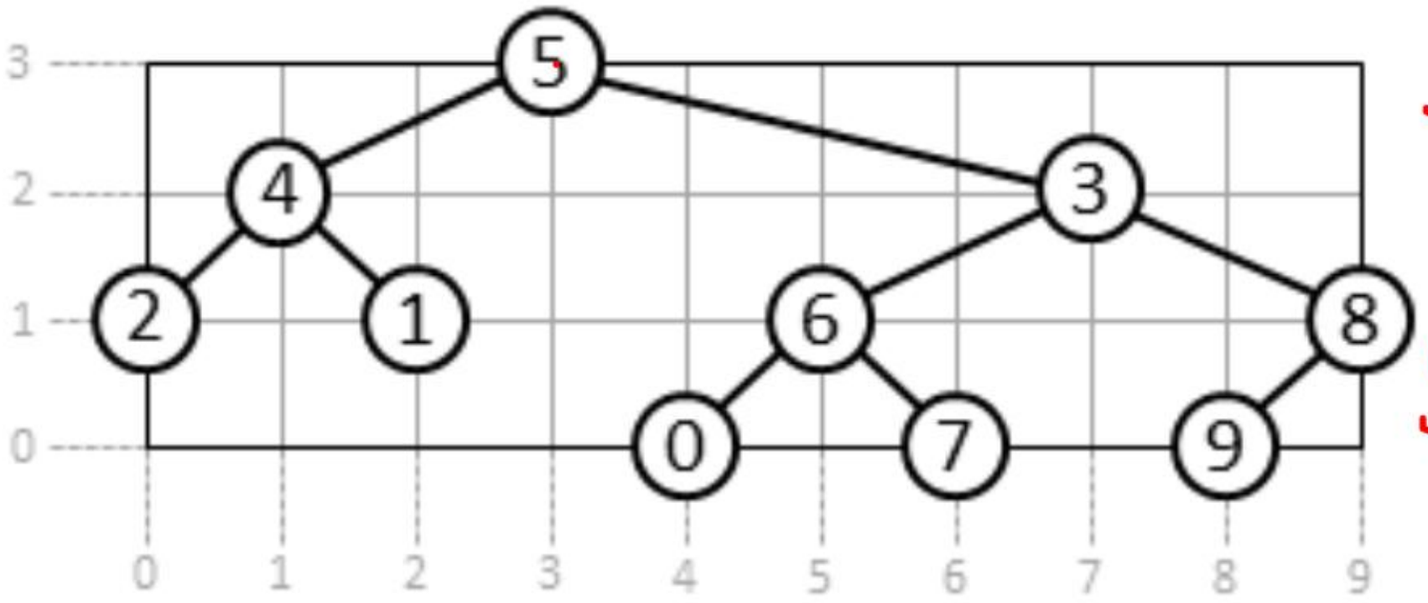
```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

y: h(T) − h(u)

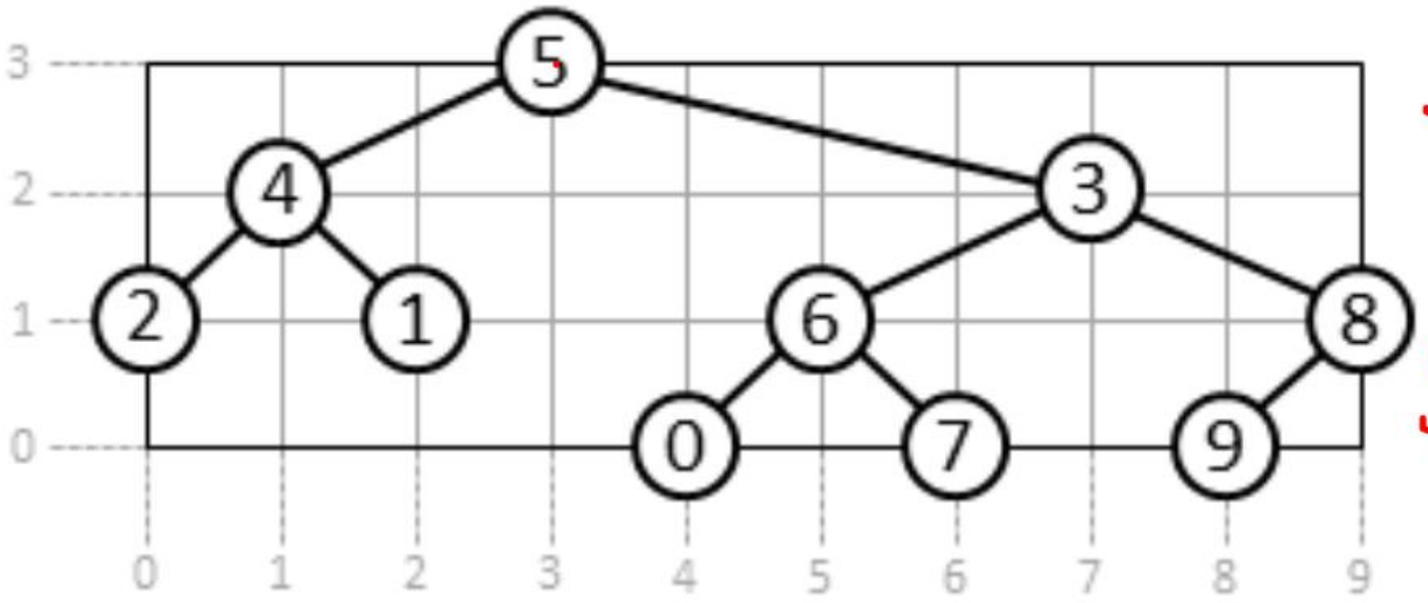
3 1

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

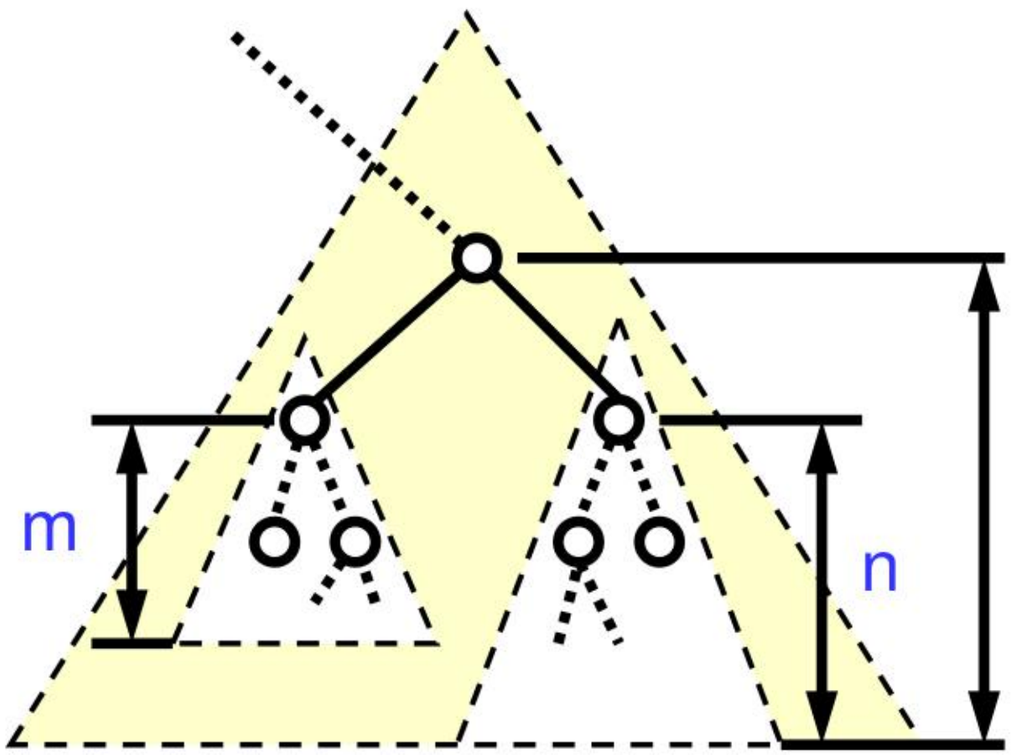
y: h(T) − h(u)

3 1

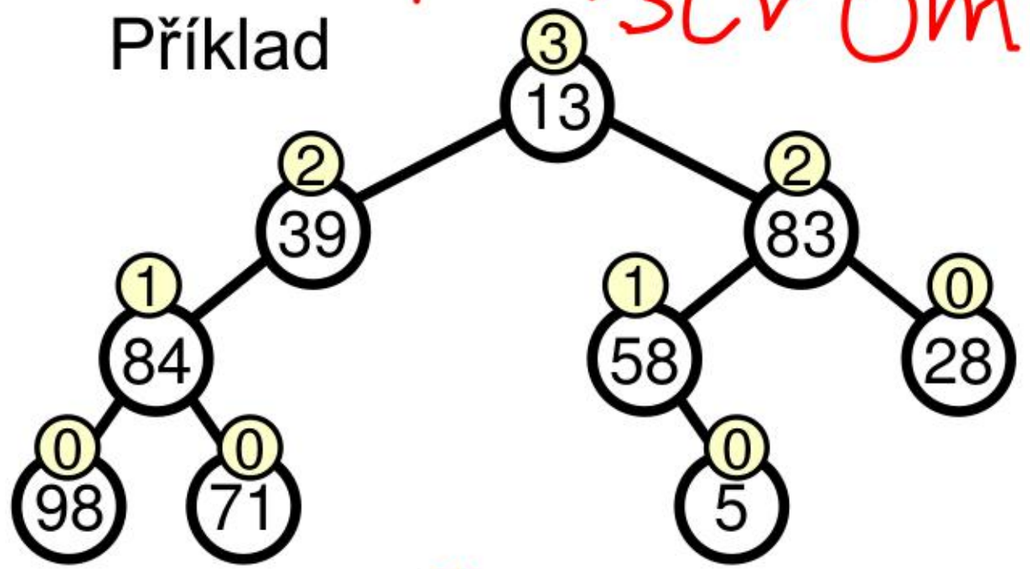
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad




0

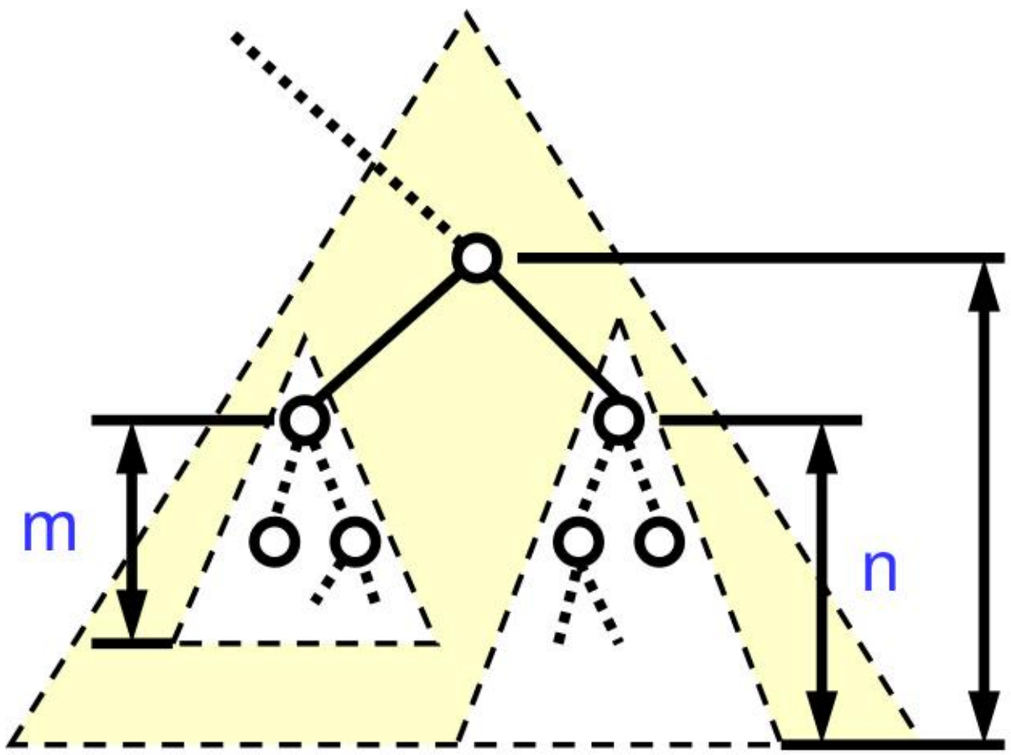
max(-1,-1)+1=0

```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

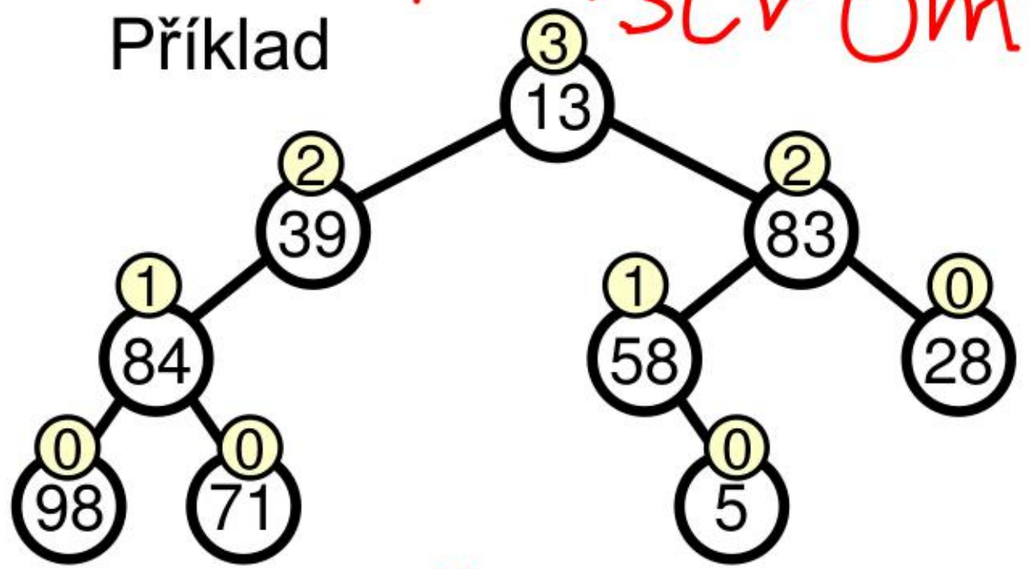
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad



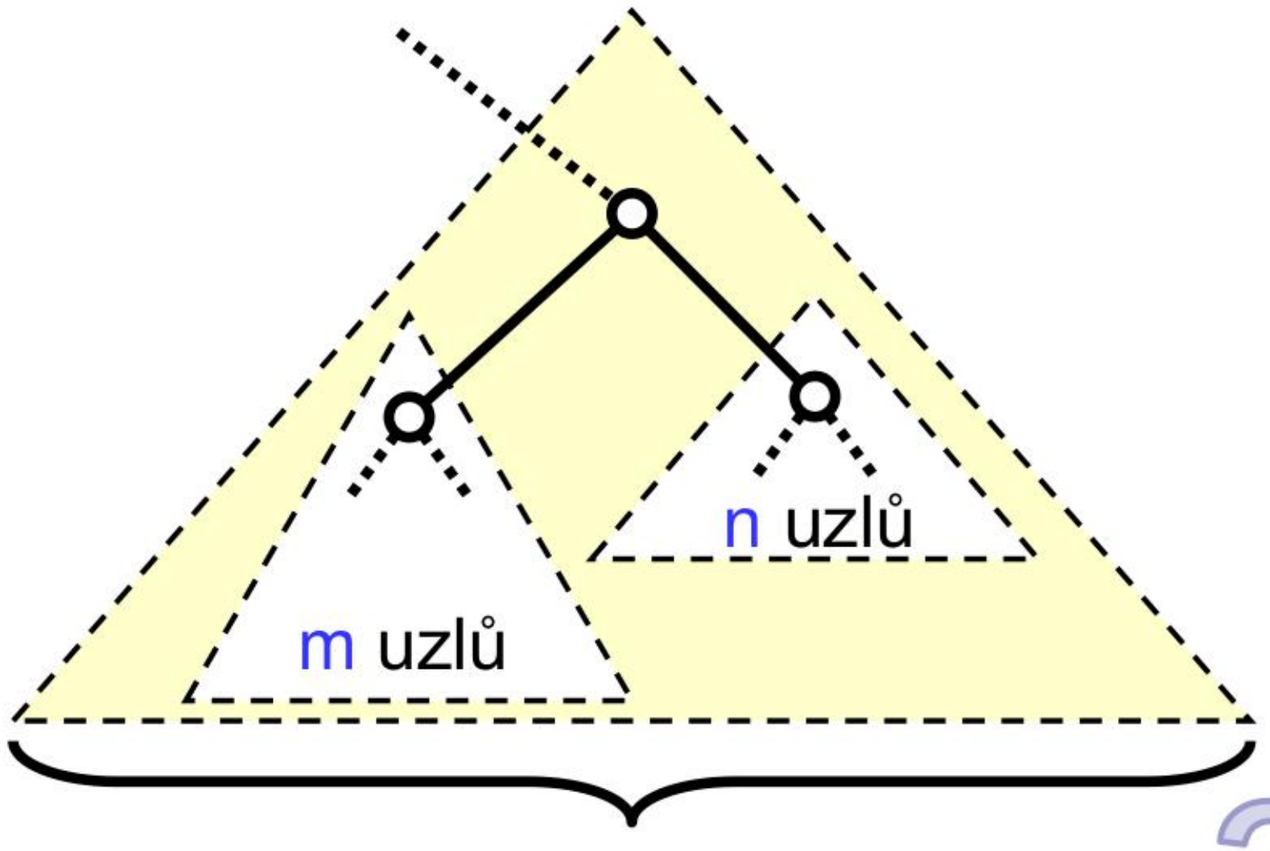

0

max(-1,-1)+1=0

```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

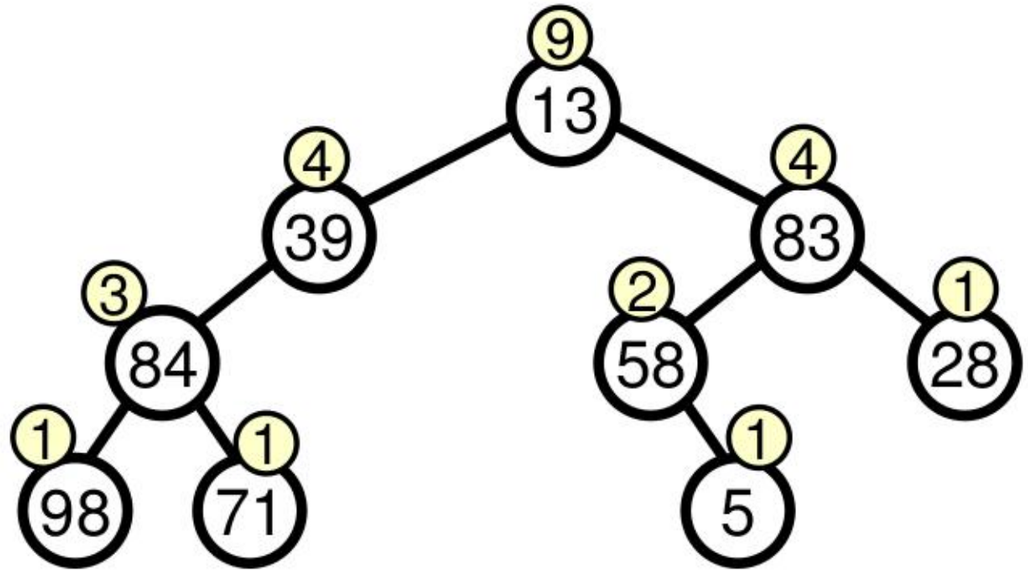
# Počítání uzlů

Strom nebo podstrom




celkem ... m+n+1 uzlů

Příklad



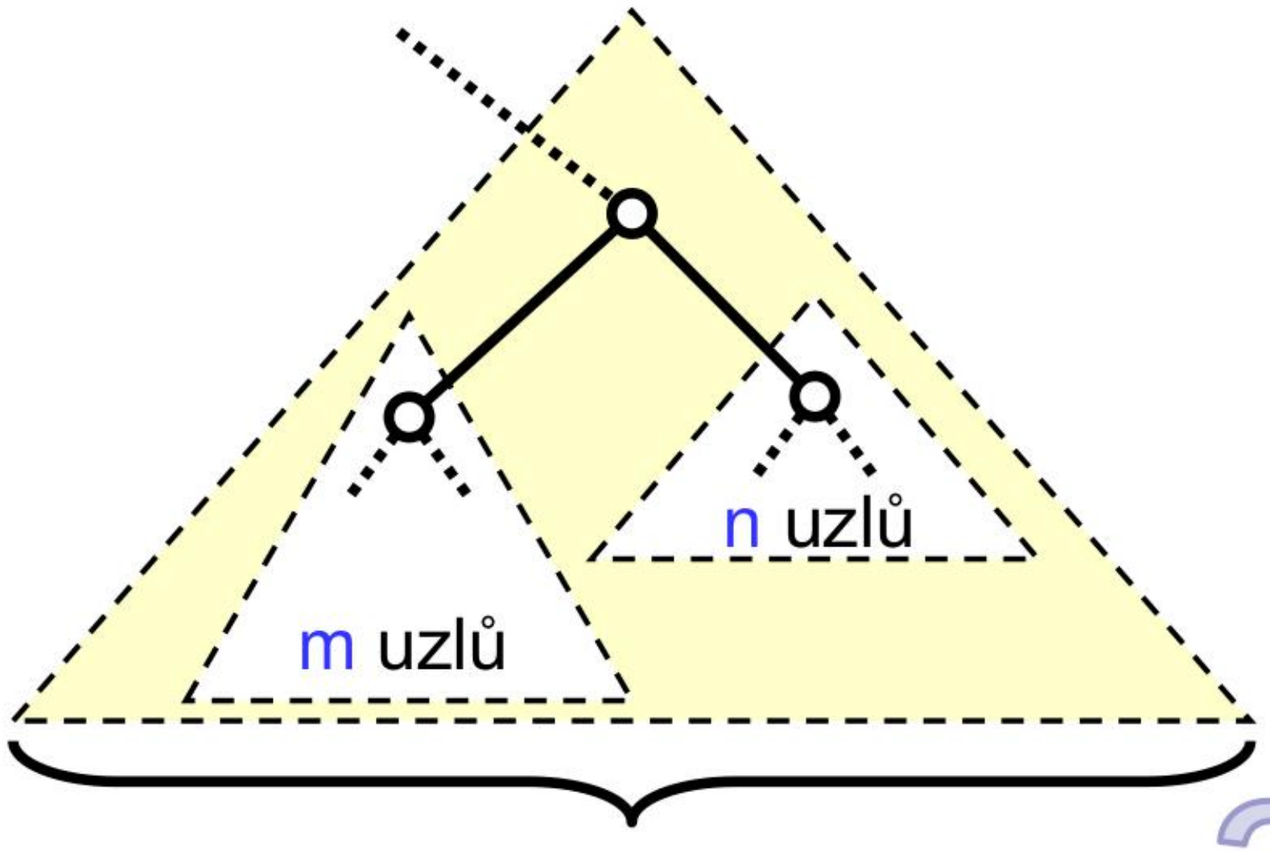

Je to průchod v pořadí preorder, inorder nebo postorder?

```
int count(Node node) {
  if (node == null) return 0;
  return (count(node.left) + count(node.right) + 1);
}
```

1+

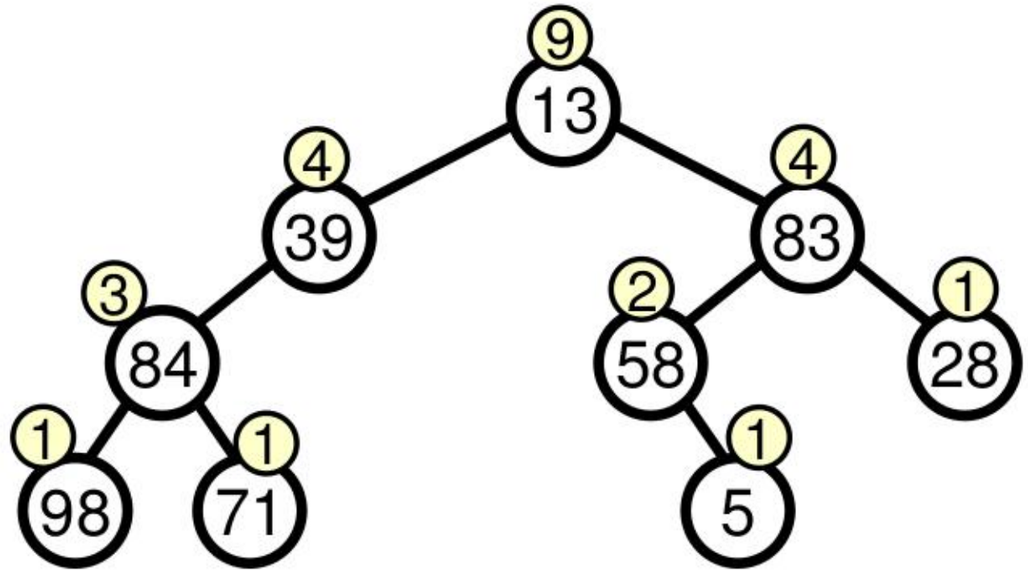
# Počítání uzlů

Strom nebo podstrom




celkem ... m+n+1 uzlů

Příklad




Je to průchod v pořadí preorder, inorder nebo postorder?

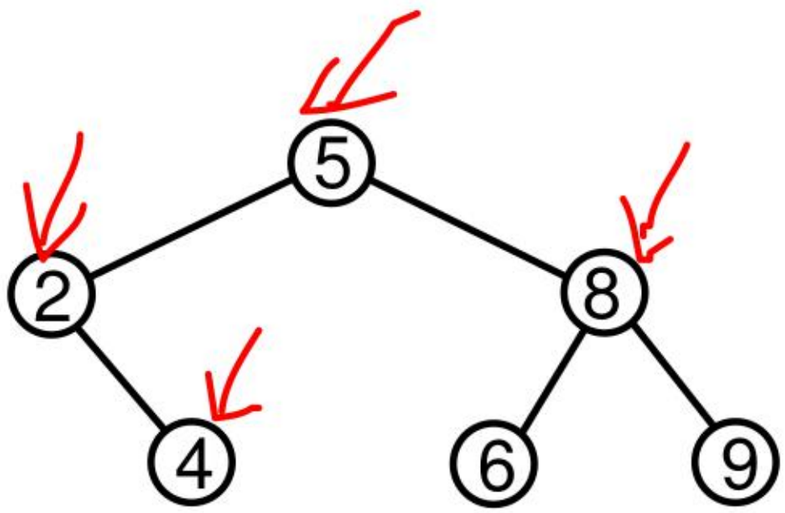
```
int count(Node node) {
  if (node == null) return 0;
  return (count(node.left) + count(node.right) + 1);
}
```

1+

# Zásobník implementuje rekurzi

```
void inorderIterative(Node root) {
    Stack<Node> stack = new Stack();
    Node curr = root;
    while (!stack.empty() || curr != null) {
       if (curr != null) {
           stack.push(curr);
           curr = curr.left;
        } else {
           curr = stack.pop();
           System.out.print(curr.key + " ");
           curr = curr.right;
       }
    }
}
```

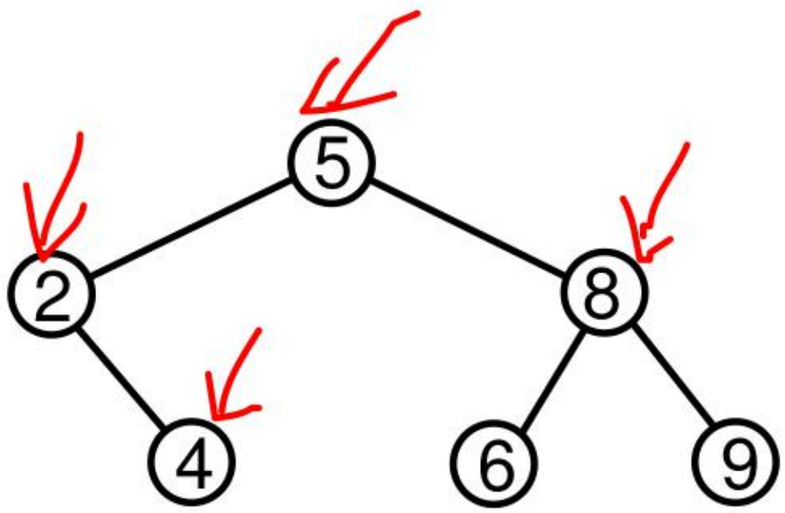
Uzel uložíme na zásobník v okamžiku jeho objevení a odebereme jej po zpracování jeho levého podstromu.




# Zásobník implementuje rekurzi

```
void inorderIterative(Node root) {
    Stack<Node> stack = new Stack();
    Node curr = root;
    while (!stack.empty() || curr != null) {
       if (curr != null) {
           stack.push(curr);
           curr = curr.left;
        } else {
           curr = stack.pop();
           System.out.print(curr.key + " ");
           curr = curr.right;
       }
    }
}
```

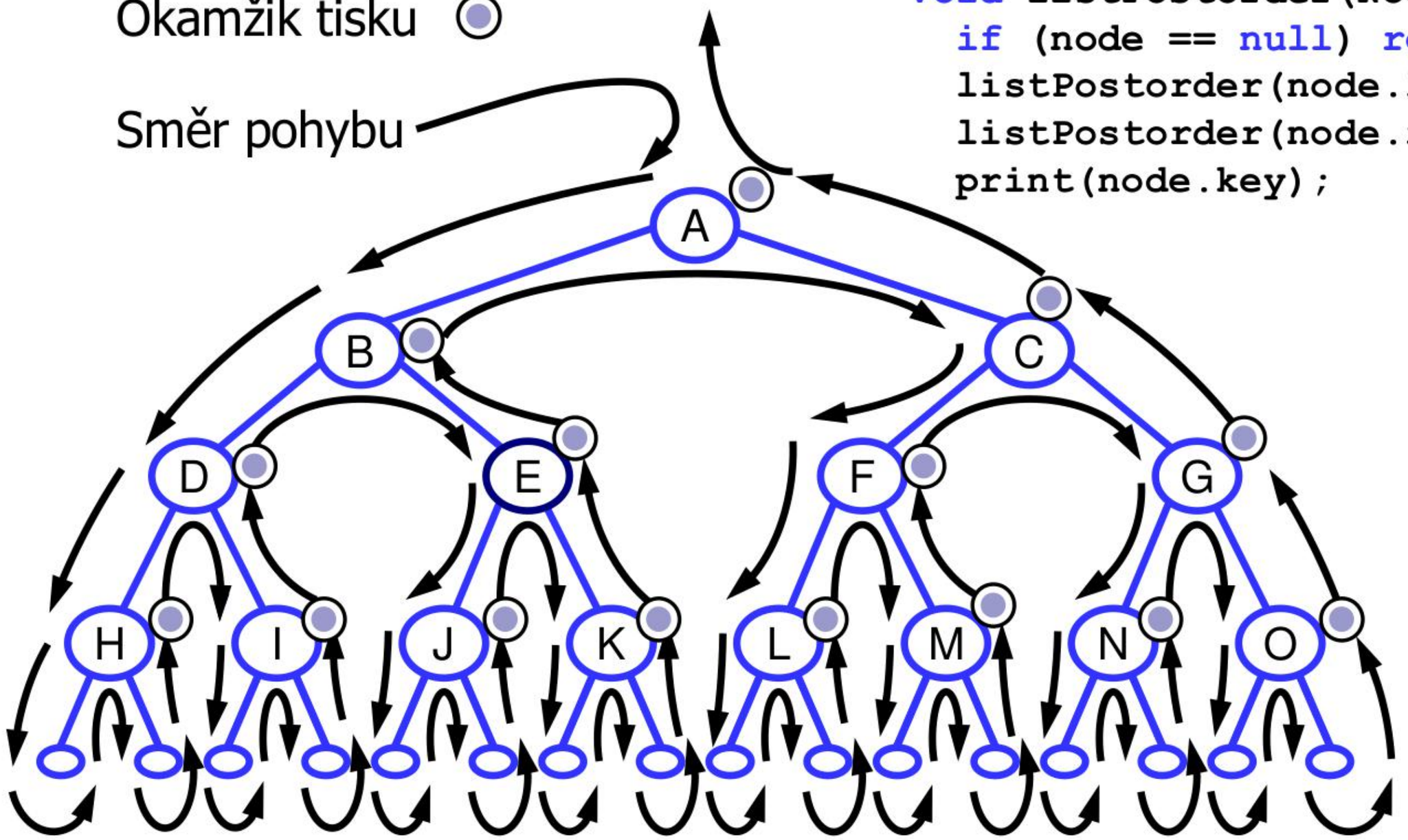
Uzel uložíme na zásobník v okamžiku jeho objevení a odebereme jej po zpracování jeho levého podstromu.




# Průchod v pořadí Postorder

Okamžik tisku

Směr pohybu




```
void listPostorder(Node node):
  if (node == null) return;
  listPostorder(node.left);
  listPostorder(node.right);
  print(node.key);
```

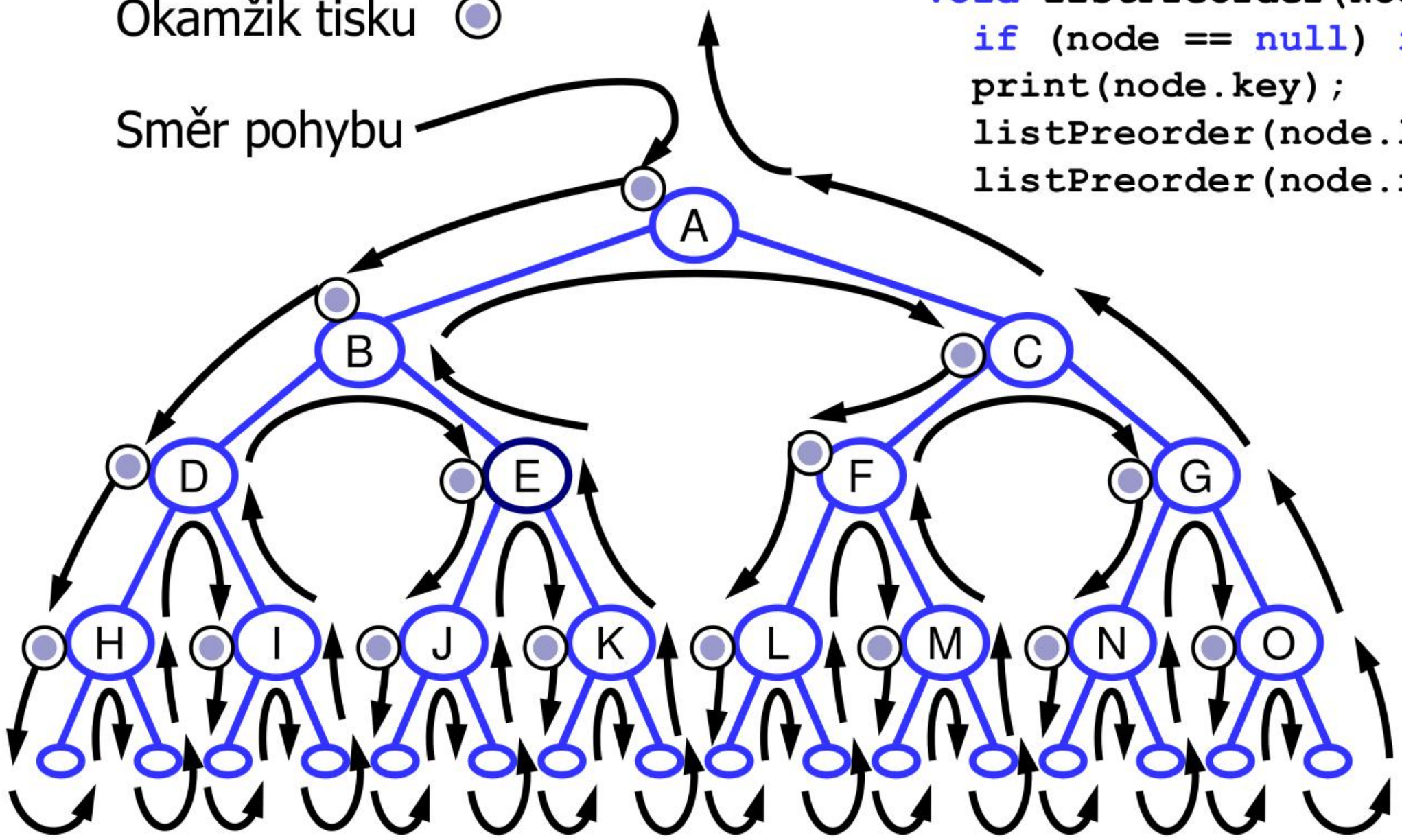
Výstup

H I D J K E B L M F N O G C A

# Průchod v pořadí Preorder

Okamžik tisku

Směr pohybu




```
void listPreorder(Node node):
  if (node == null) return;
  print(node.key);
  listPreorder(node.left);
  listPreorder(node.right);
```

Výstup

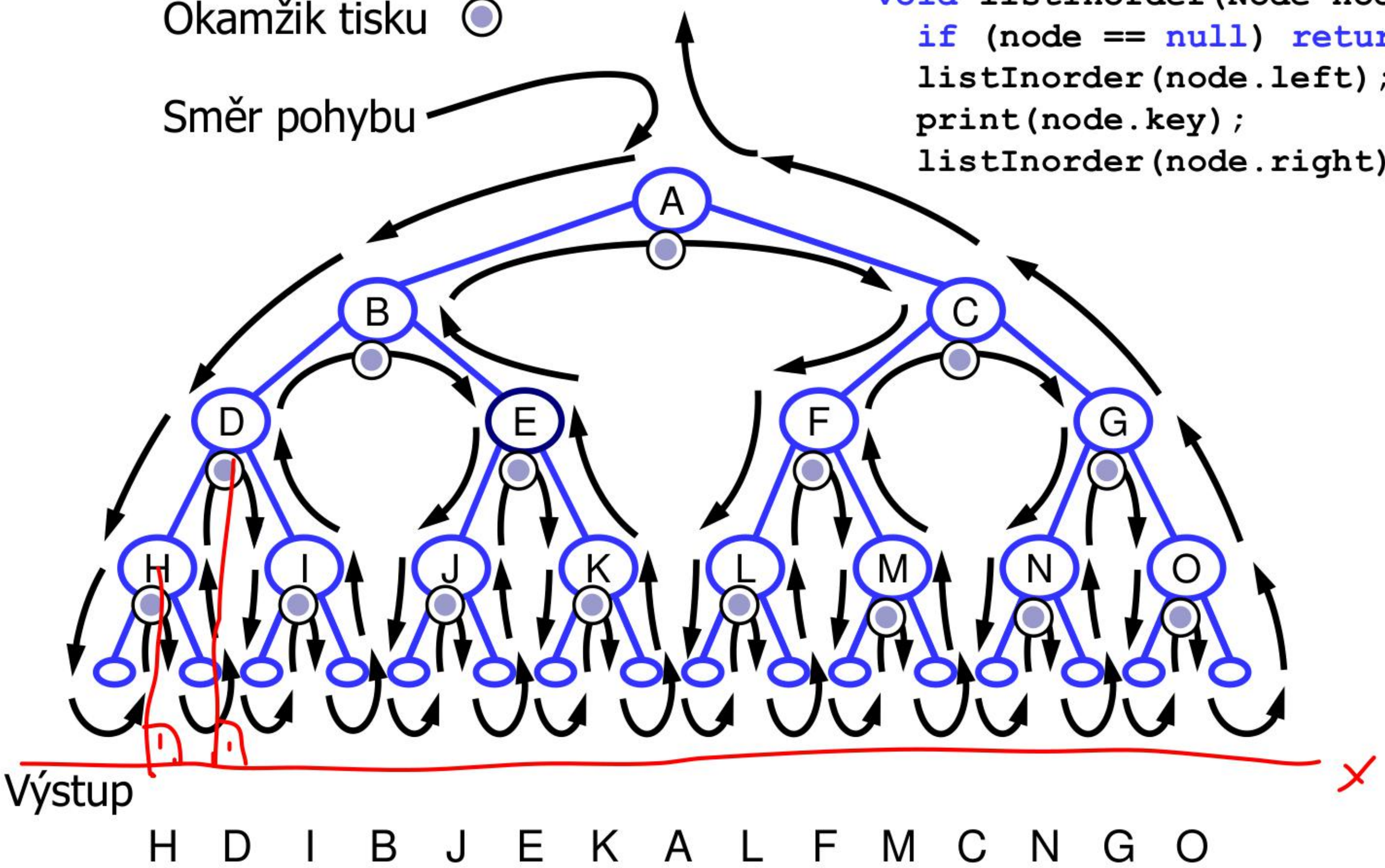
A B D H I E J K C F L M G N O

# Průchod v pořadí Inorder

Okamžik tisku

Směr pohybu

```
void listInorder(Node node):
  if (node == null) return;
  listInorder(node.left);
  print(node.key);
  listInorder(node.right);
```

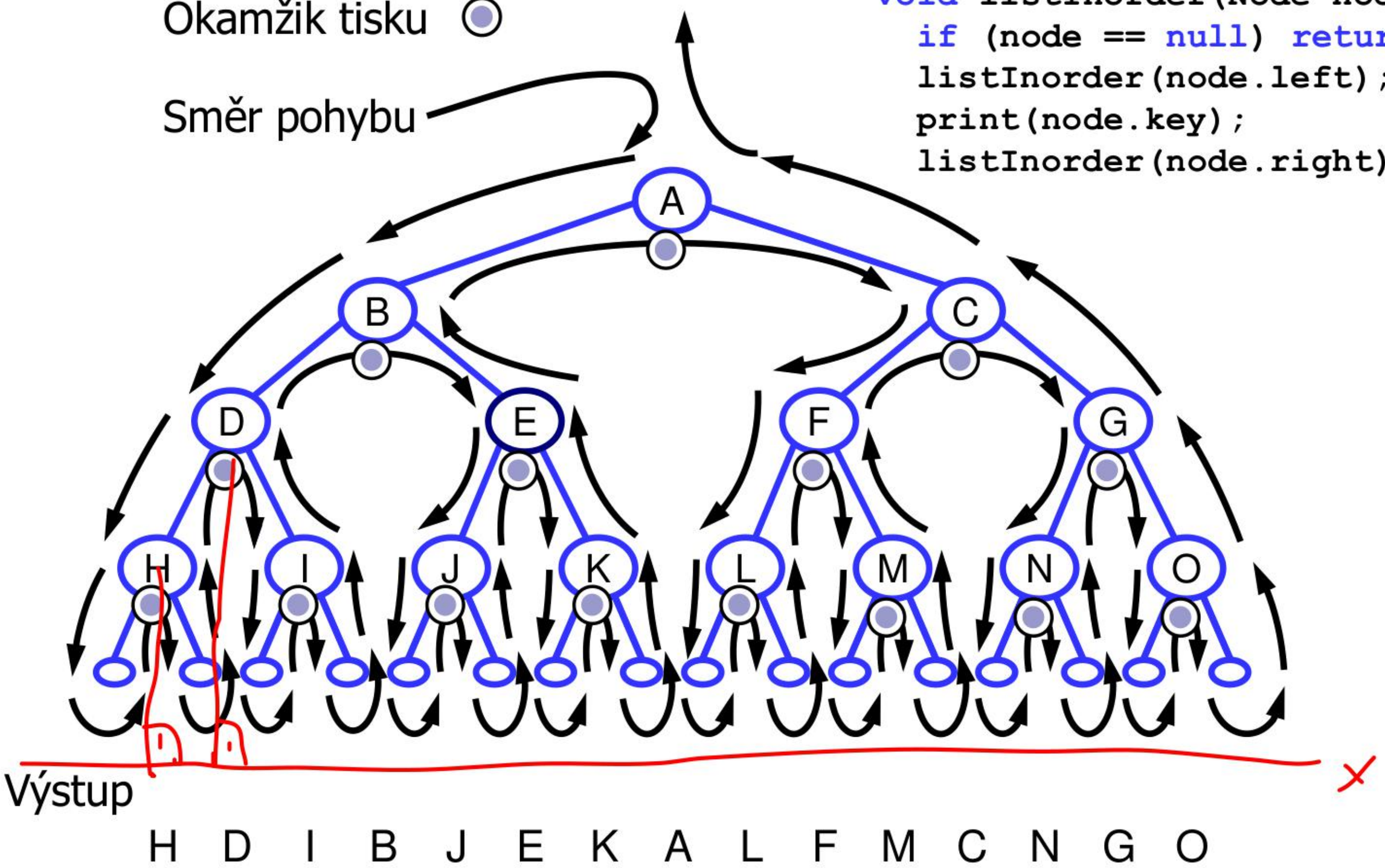


# Průchod v pořadí Inorder

Okamžik tisku

Směr pohybu

```
void listInorder(Node node):
  if (node == null) return;
  listInorder(node.left);
  print(node.key);
  listInorder(node.right);
```

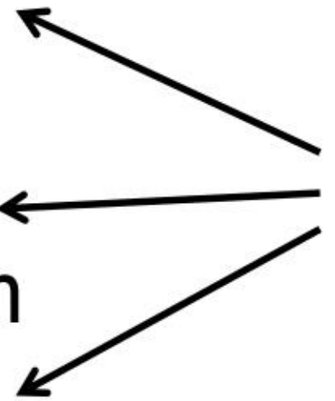


# Průchod stromem do hloubky

Začni v kořeni stromu a opakuj:

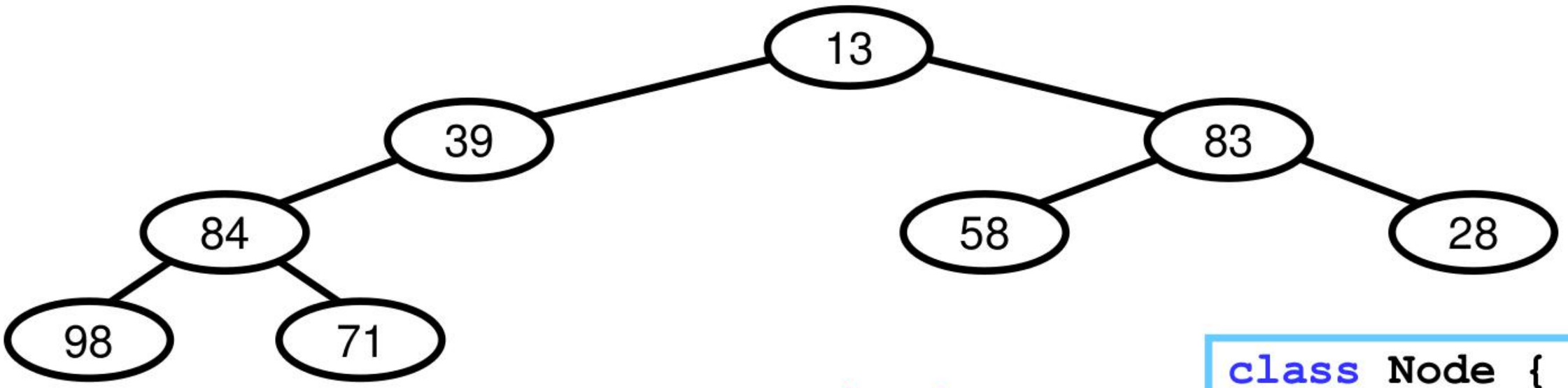
Pro aktuální uzel *u*

- projdi rekurzivně levý podstrom
- projdi rekurzivně pravý podstrom



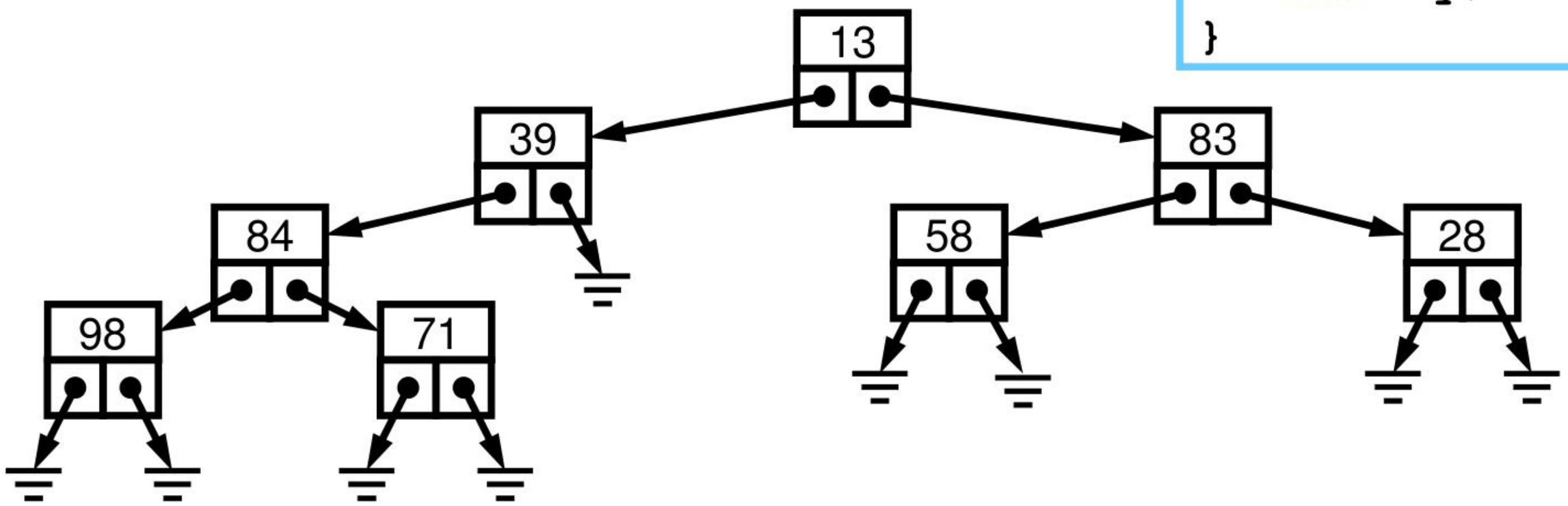
zpracuj uzel *u*
(je více možností, kdy to provést)

# Reprezentace binárního stromu

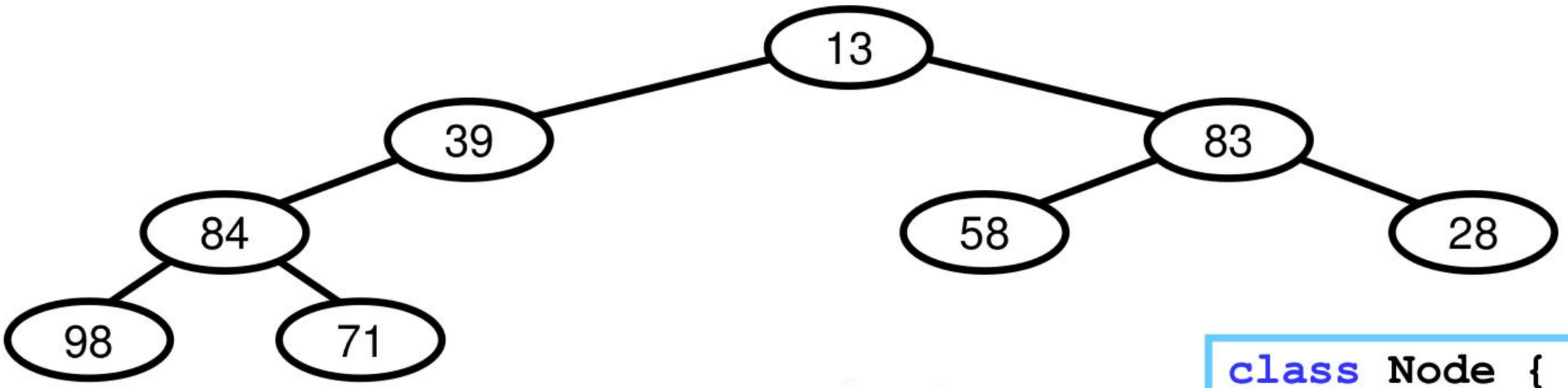



```
class Node {
    Node left;
    Node right;
    int key;
}
```

Node parent;

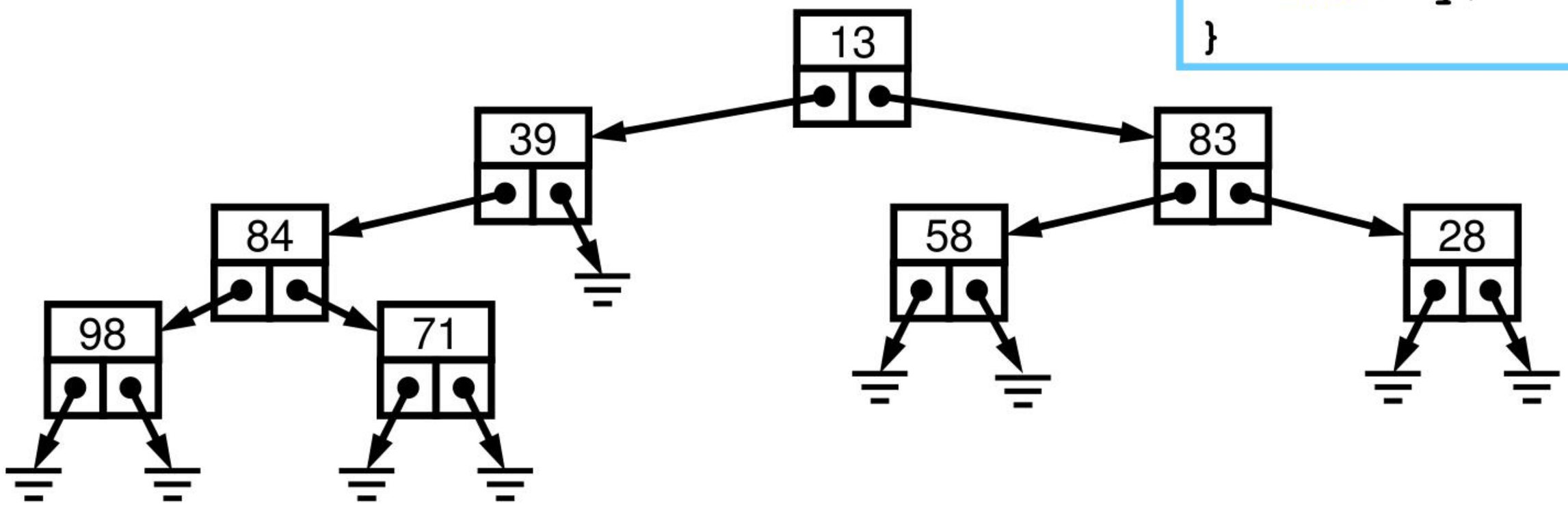


# Reprezentace binárního stromu



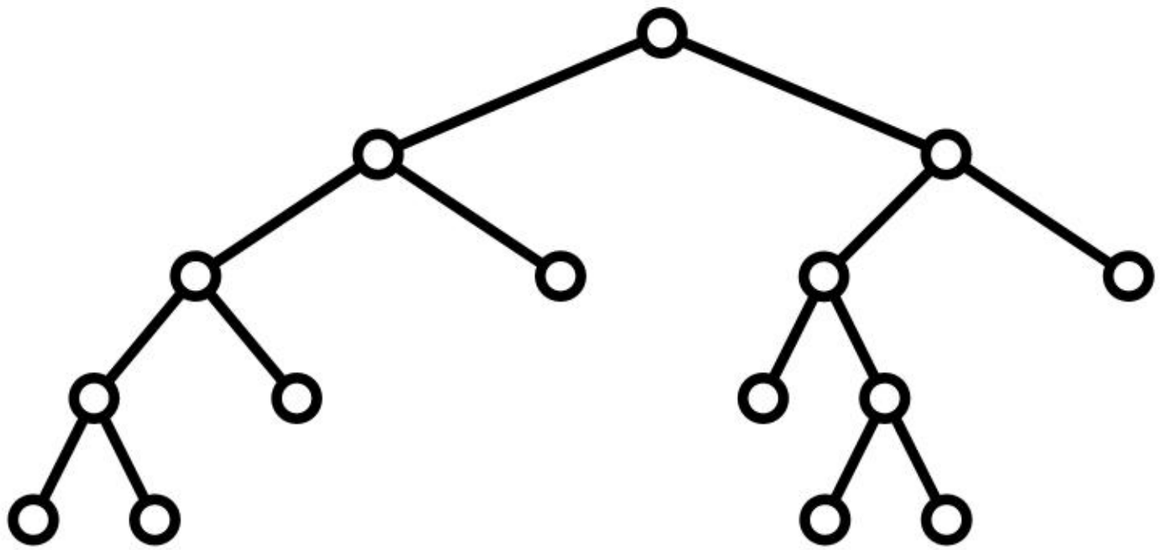

```
class Node {
    Node left;
    Node right;
    int key;
}
```

Node parent;



# Binární strom

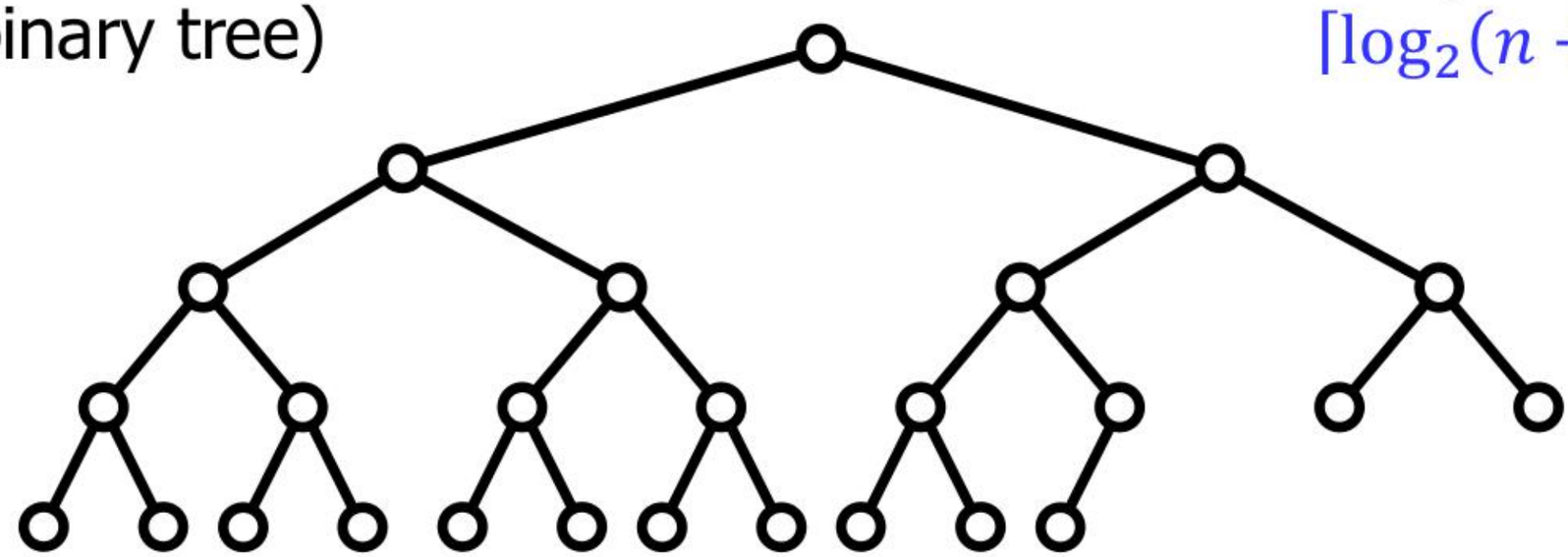
Pravidelný binární strom
(regular binary tree)



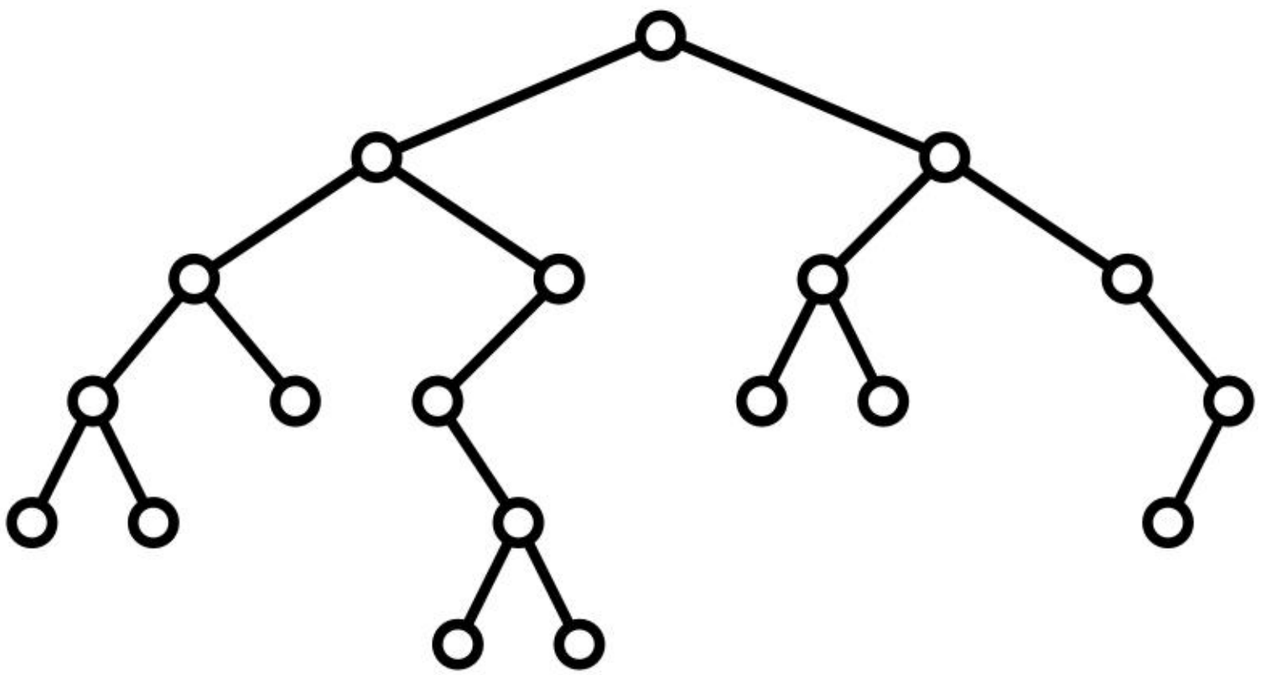
Počet potomků každého uzlu je jen 0 nebo 2.

Vyvážený binární strom
(balanced binary tree)

hloubka pro $n$ uzlů je
$\lceil \log_2(n+1) \rceil - 1$

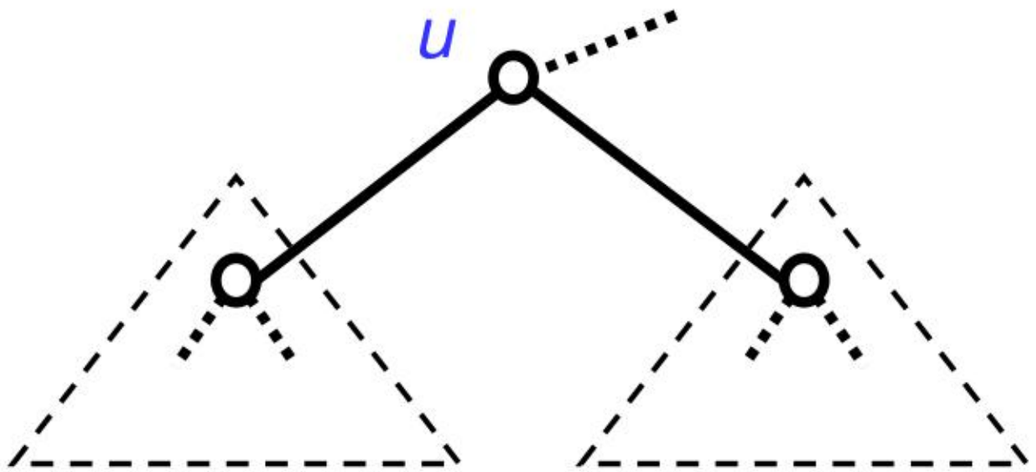


# Binární strom



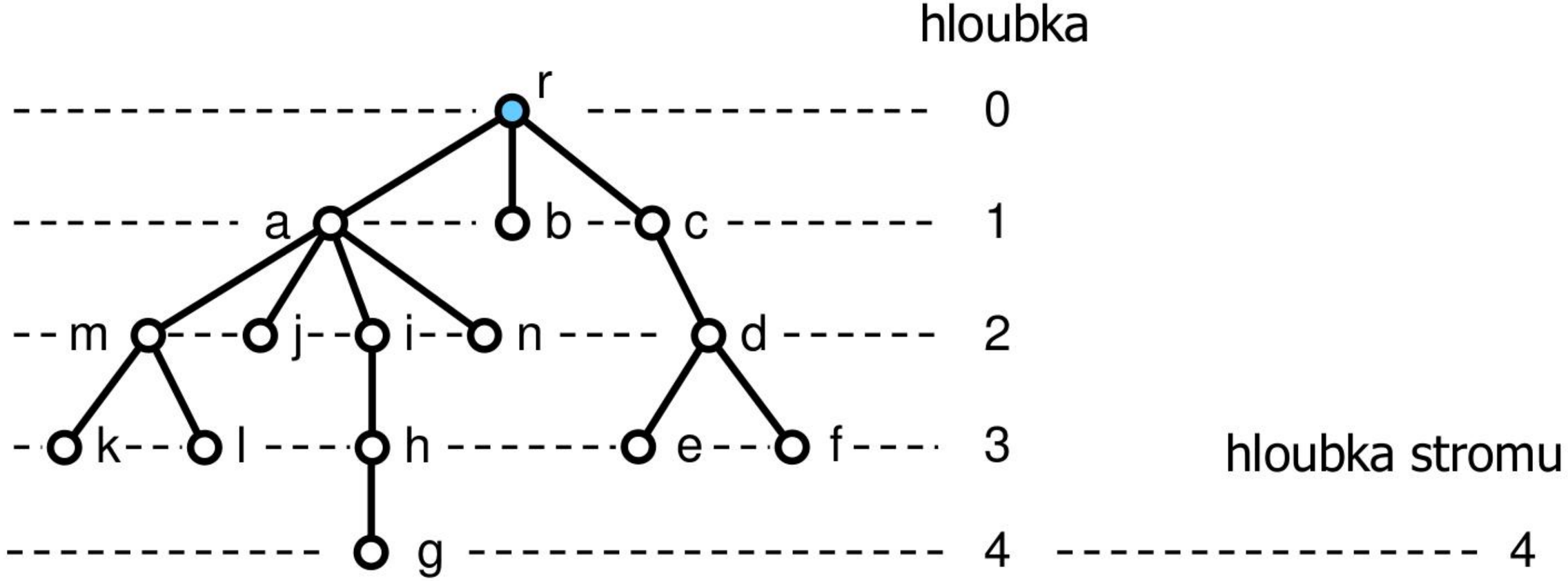
Počet potomků každého uzlu je 0,1, nebo 2.

Levý a pravý podstrom (left and right subtree):




Podstrom uzlu *u* ..... levý ..... pravý

# Hloubka kořenového stromu

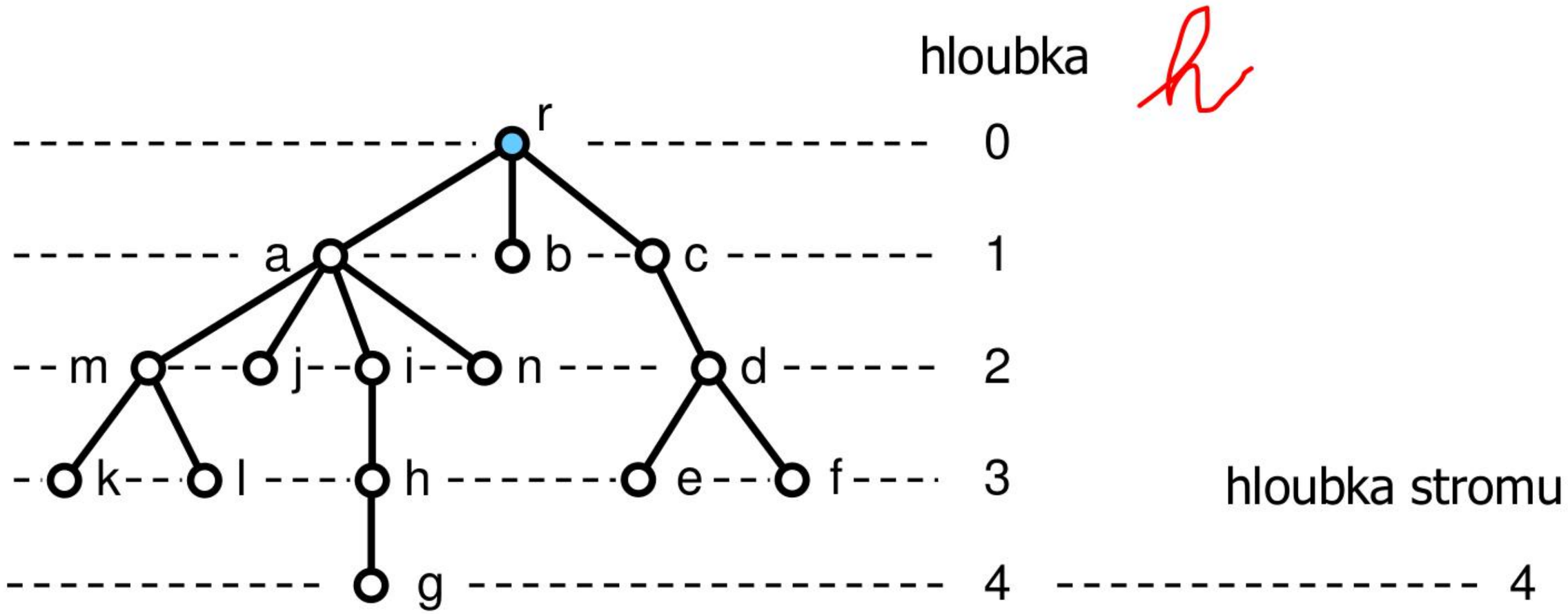



Hloubka uzlu $u$ je hranová vzdálenost $u$ od kořene.

Hloubka stromu $T$ je maximum z hloubek uzlů v $T$.

Hloubku prázdného stromu definujeme jako $-1$.

# Hloubka kořenového stromu

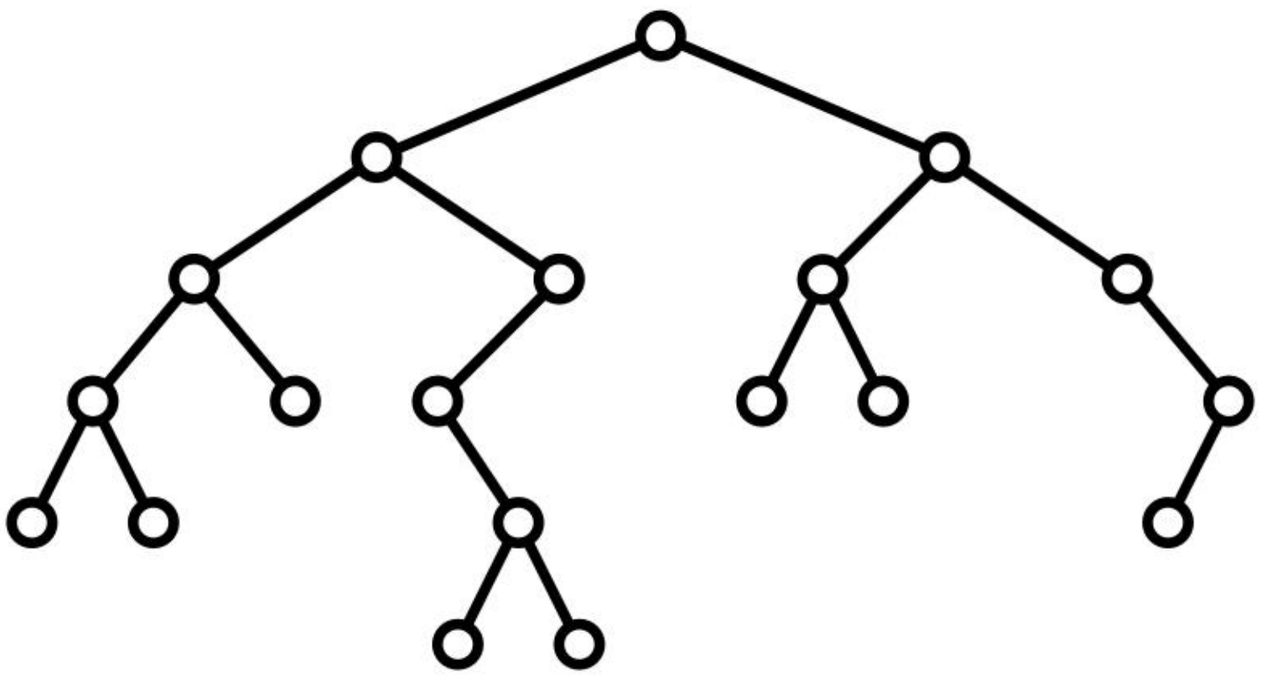



Hloubka uzlu $u$ je hranová vzdálenost $u$ od kořene.

Hloubka stromu $T$ je maximum z hloubek uzlů v $T$.

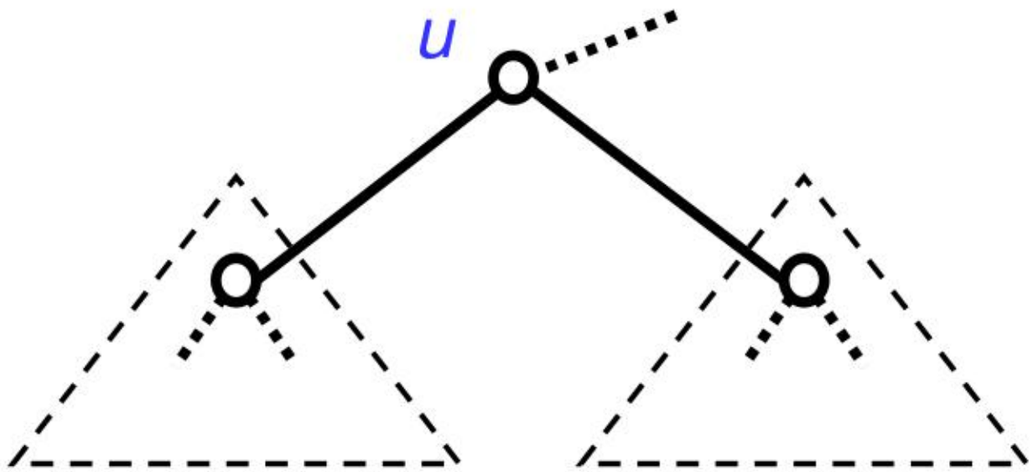
Hloubku prázdného stromu definujeme jako $-1$.

# Binární strom



Počet potomků každého uzlu je 0,1, nebo 2.

Levý a pravý podstrom (left and right subtree):

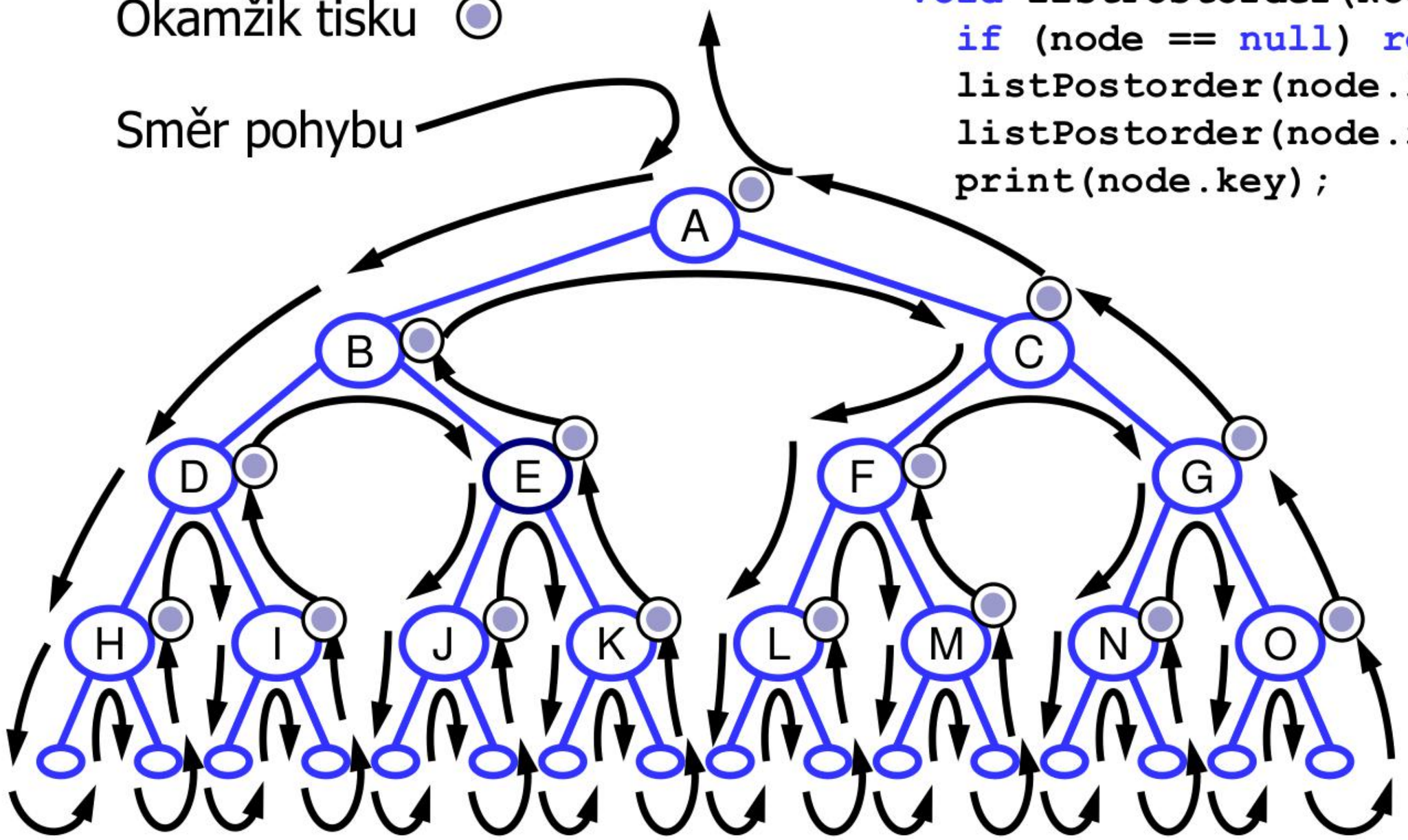



Podstrom uzlu *u* ..... levý ..... pravý

# Průchod v pořadí Postorder

Okamžik tisku

Směr pohybu




```
void listPostorder(Node node):
  if (node == null) return;
  listPostorder(node.left);
  listPostorder(node.right);
  print(node.key);
```

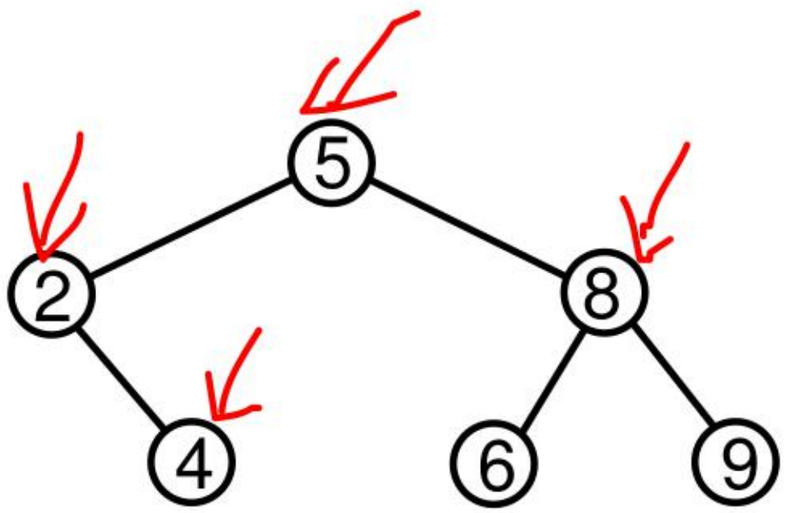
Výstup

H I D J K E B L M F N O G C A

# Zásobník implementuje rekurzi

```
void inorderIterative(Node root) {
    Stack<Node> stack = new Stack();
    Node curr = root;
    while (!stack.empty() || curr != null) {
       if (curr != null) {
           stack.push(curr);
           curr = curr.left;
        } else {
           curr = stack.pop();
           System.out.print(curr.key + " ");
           curr = curr.right;
       }
    }
}
```

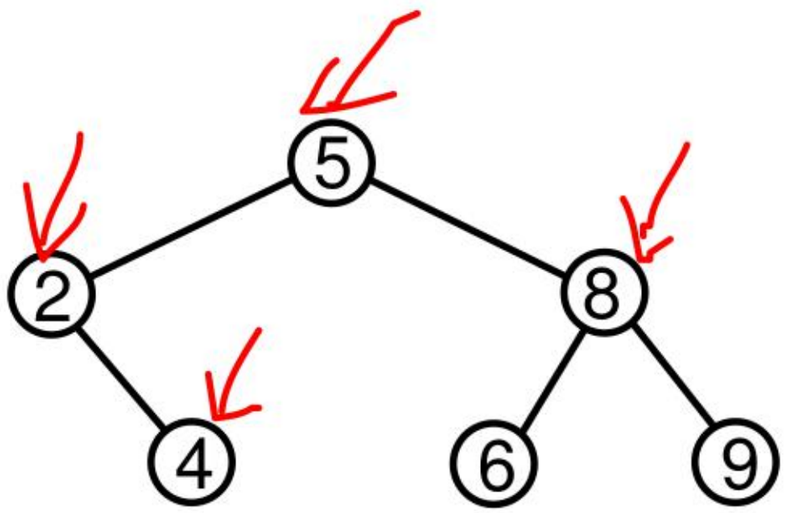
Uzel uložíme na zásobník v okamžiku jeho objevení a odebereme jej po zpracování jeho levého podstromu.




# Zásobník implementuje rekurzi

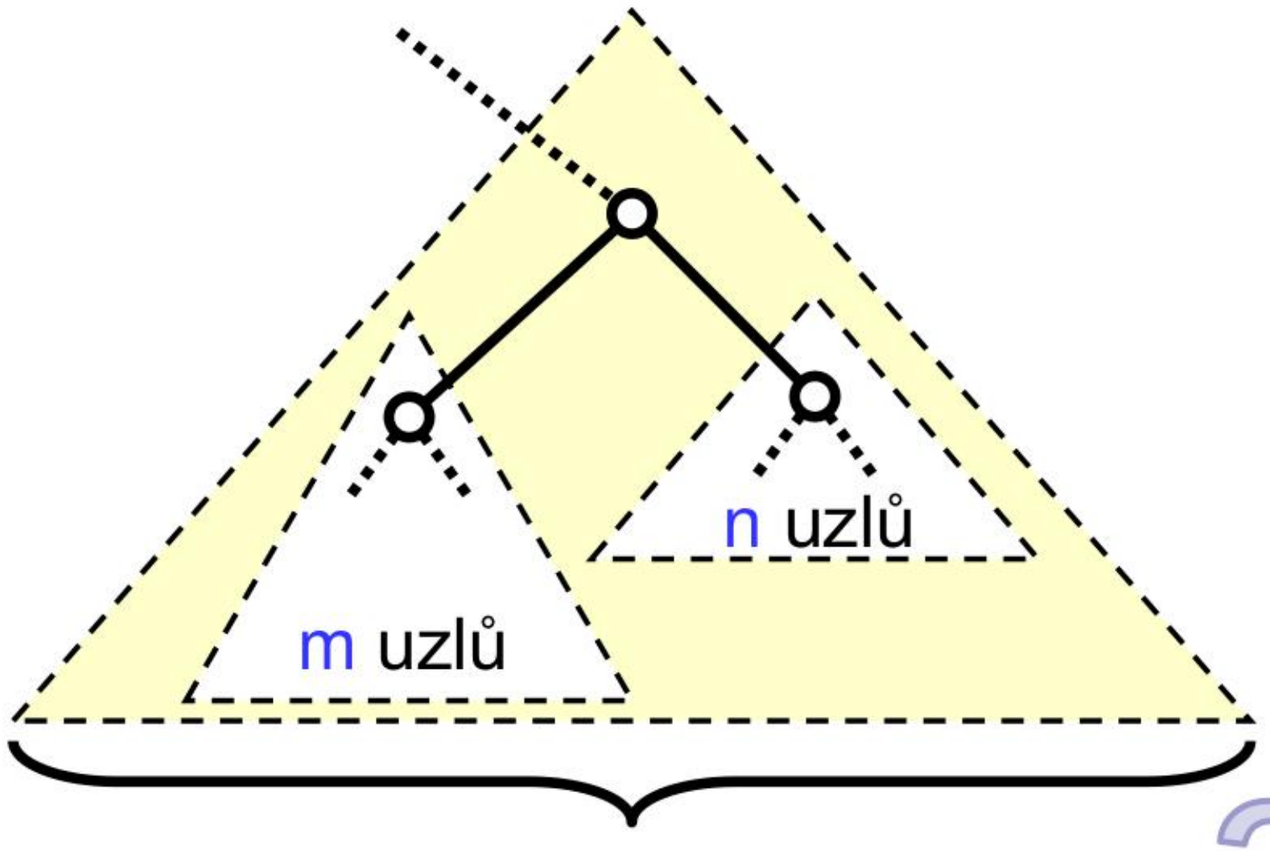
```
void inorderIterative(Node root) {
    Stack<Node> stack = new Stack();
    Node curr = root;
    while (!stack.empty() || curr != null) {
       if (curr != null) {
           stack.push(curr);
           curr = curr.left;
        } else {
           curr = stack.pop();
           System.out.print(curr.key + " ");
           curr = curr.right;
       }
    }
}
```

Uzel uložíme na zásobník v okamžiku jeho objevení a odebereme jej po zpracování jeho levého podstromu.



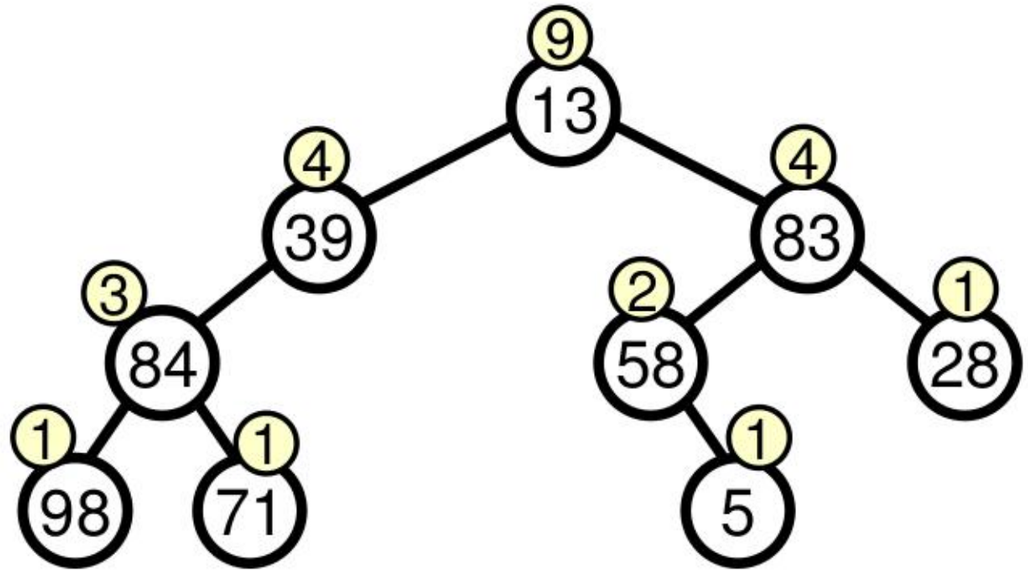

# Počítání uzlů

Strom nebo podstrom




celkem ... m+n+1 uzlů

Příklad




Je to průchod v pořadí preorder, inorder nebo postorder?

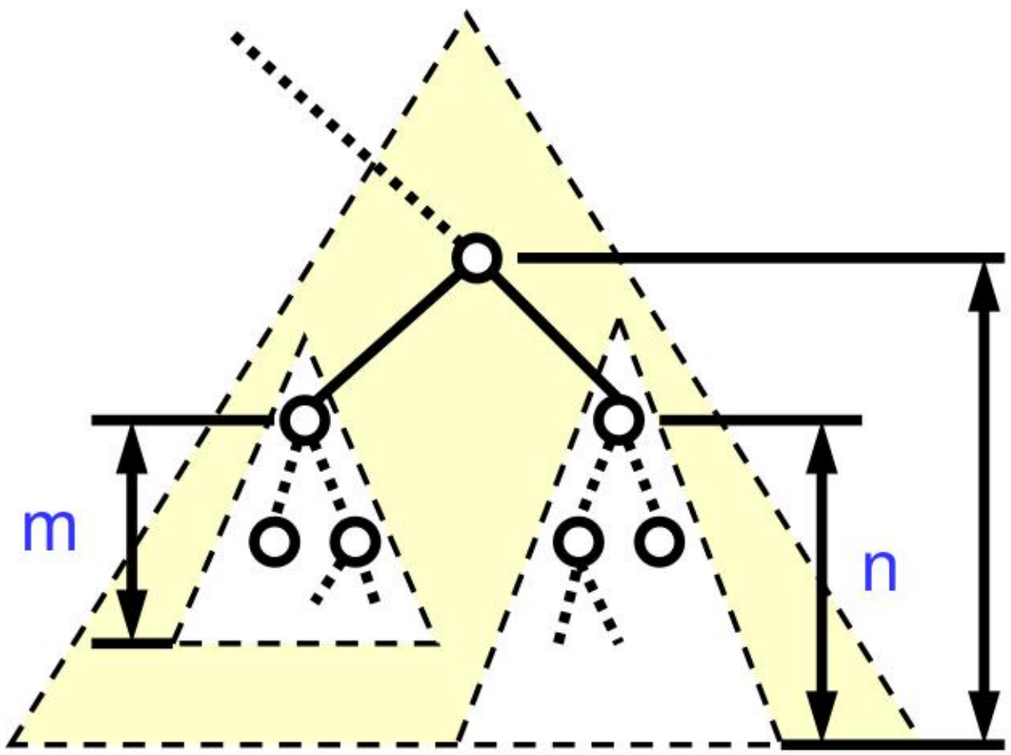
```
int count(Node node) {
  if (node == null) return 0;
  return (count(node.left) + count(node.right) + 1);
}
```

1+

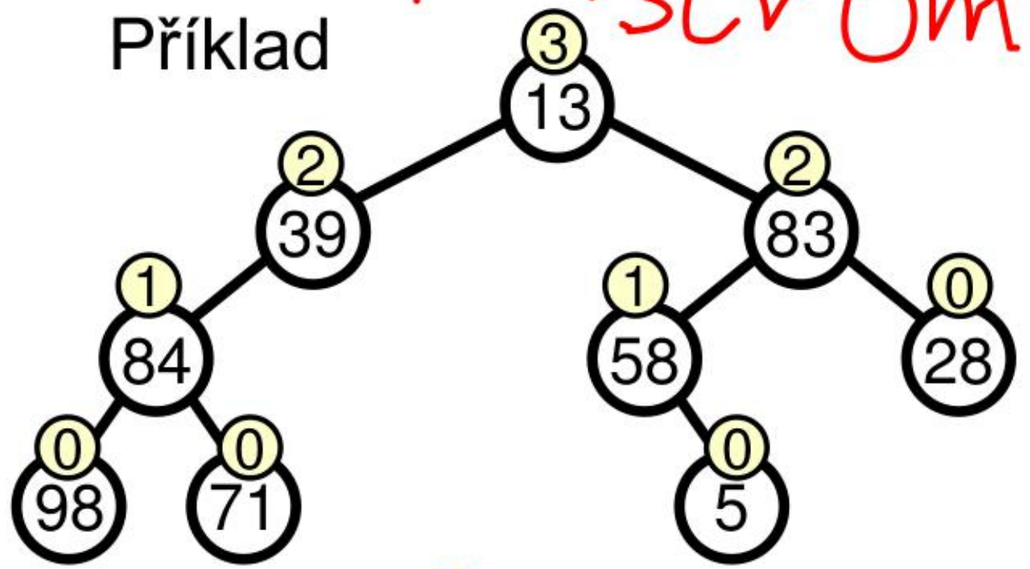
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad




0

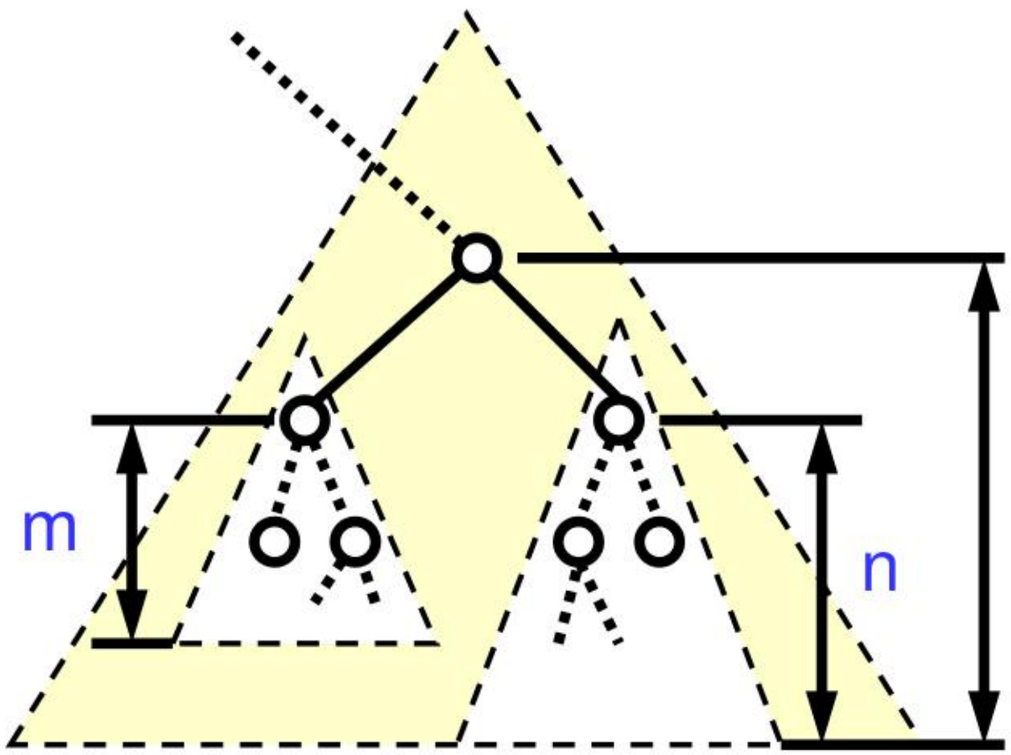
max(-1,-1)+1=0

```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

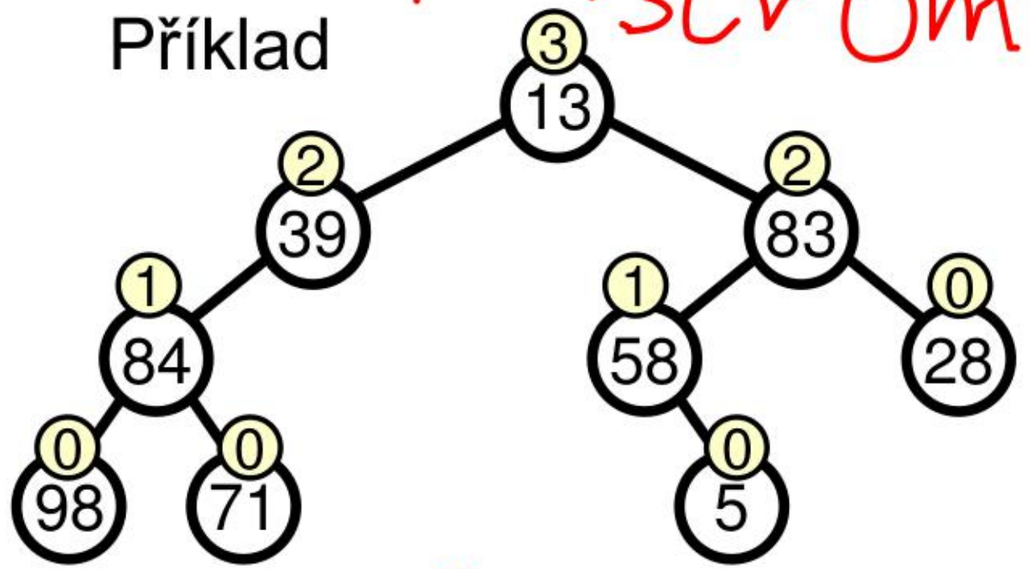
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad




0

max(-1,-1)+1=0

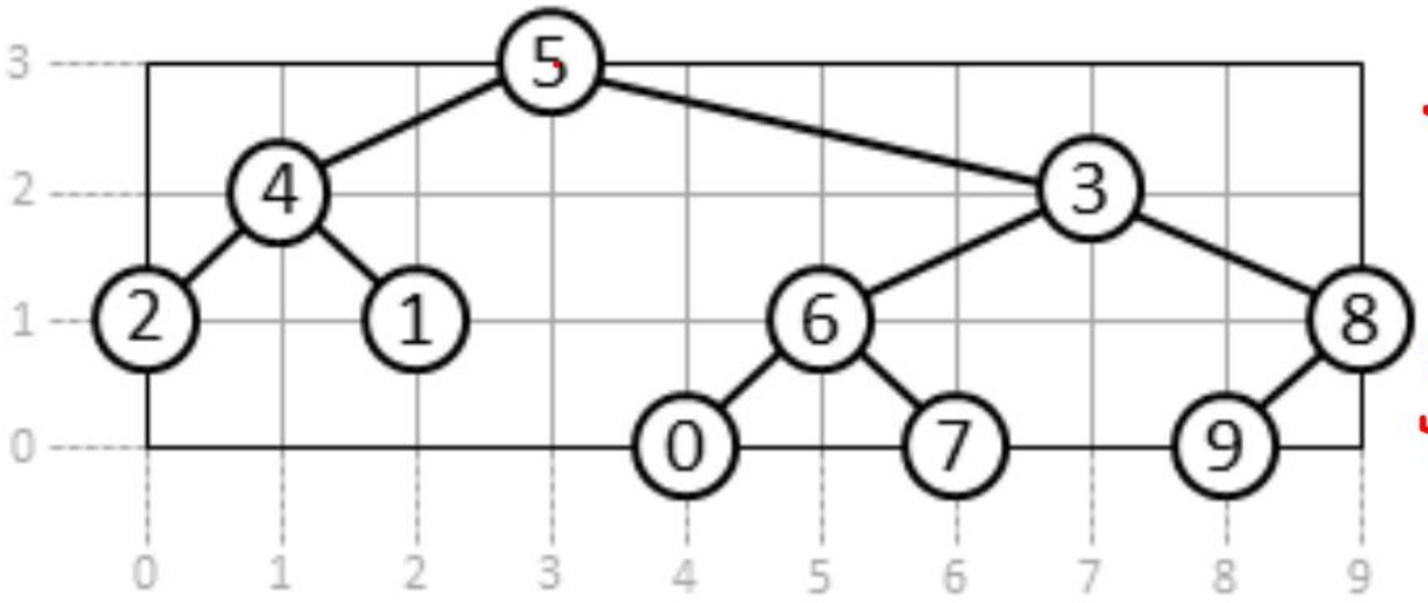
```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

y: h(T) − h(u)

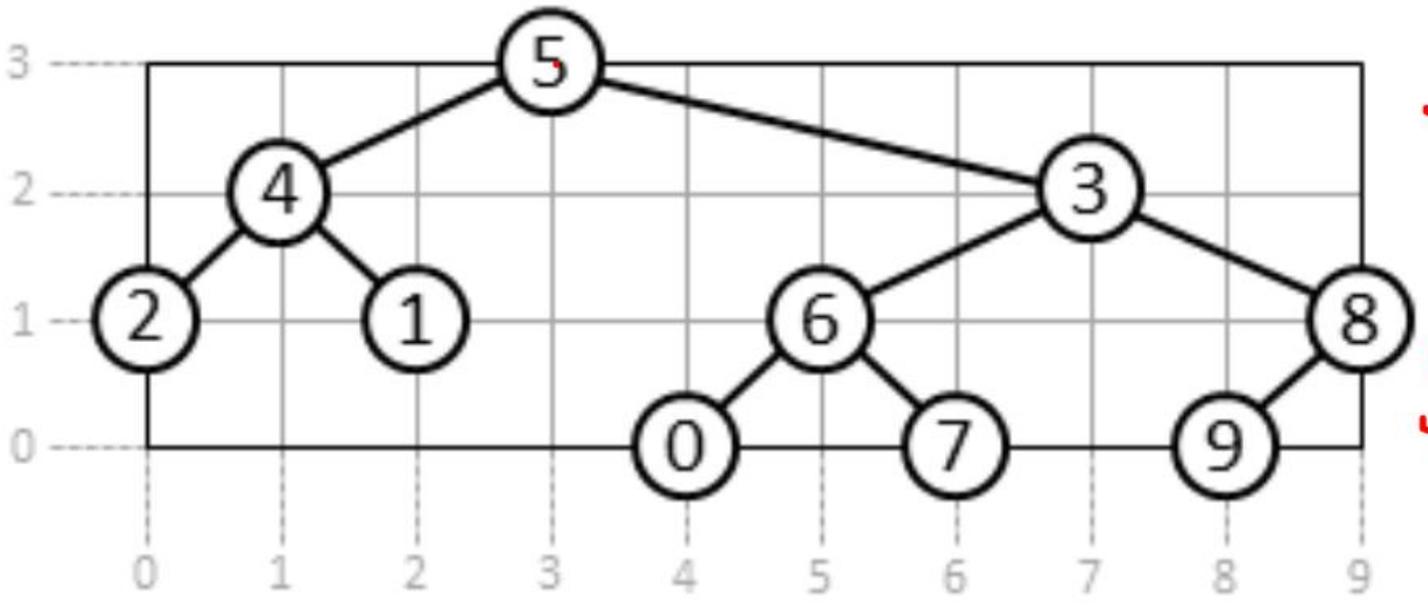
3 1

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

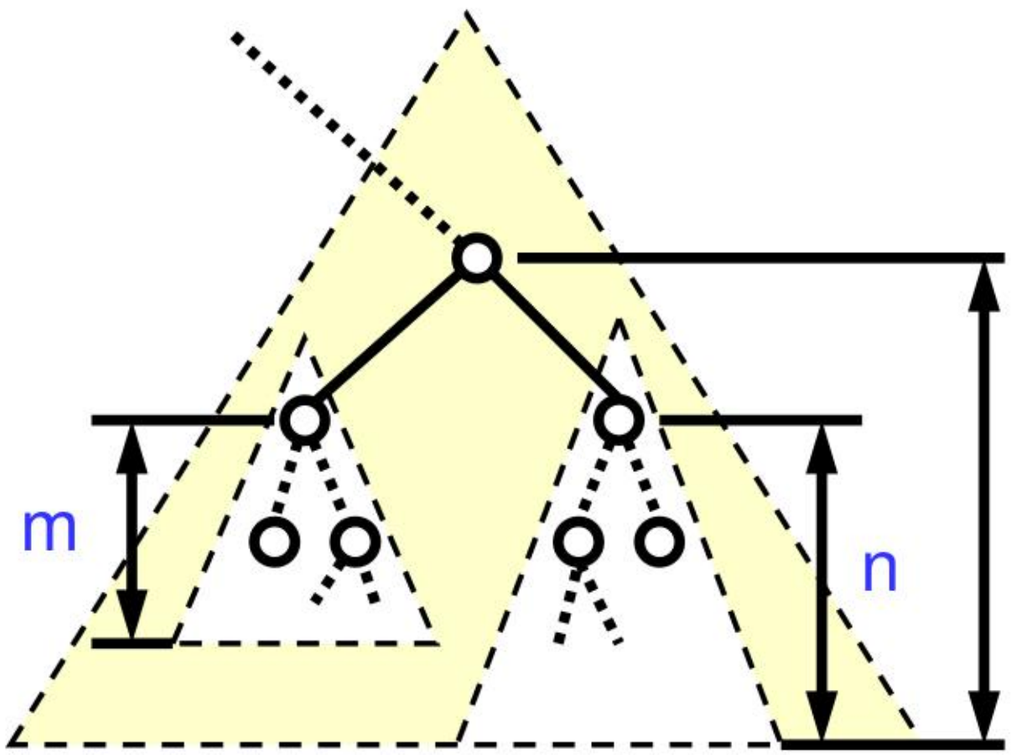
y: h(T) − h(u)

3 1

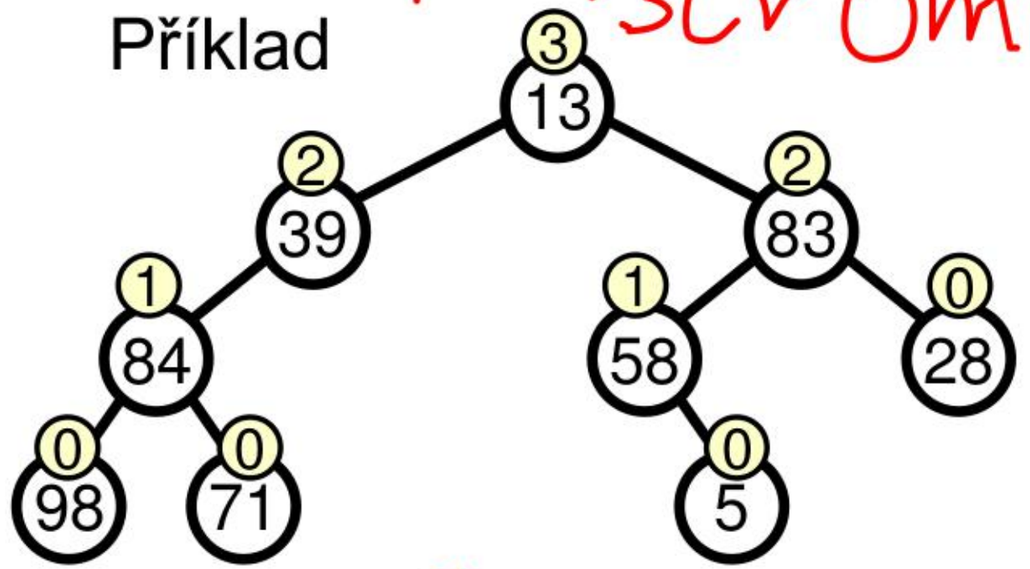
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad




0

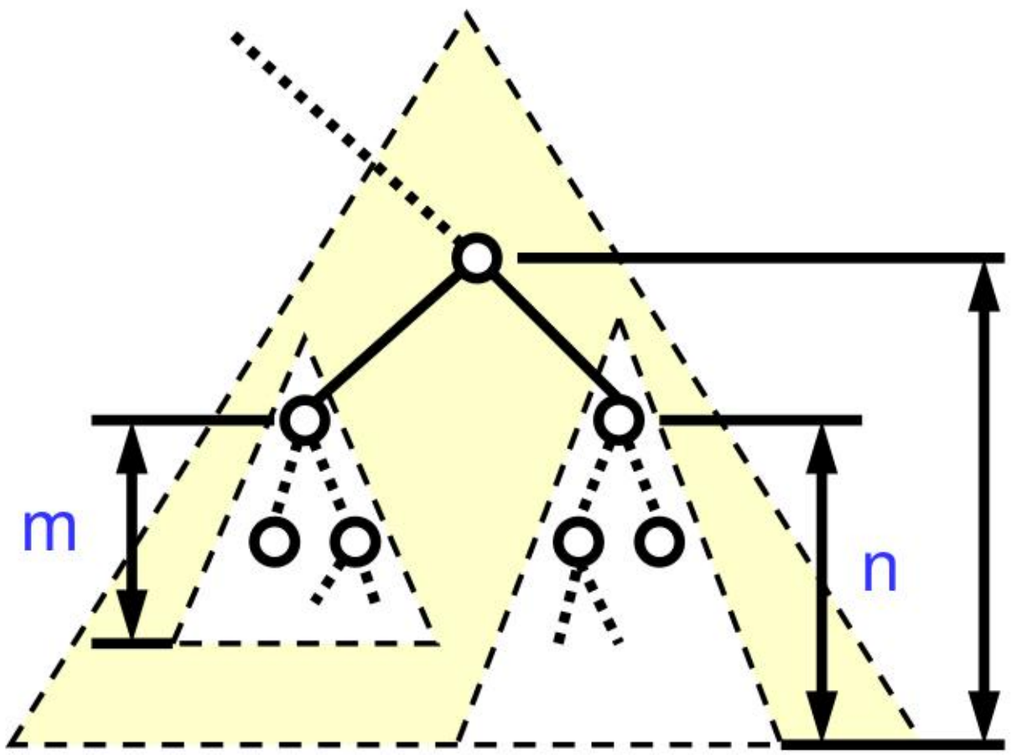
max(-1,-1)+1=0

```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

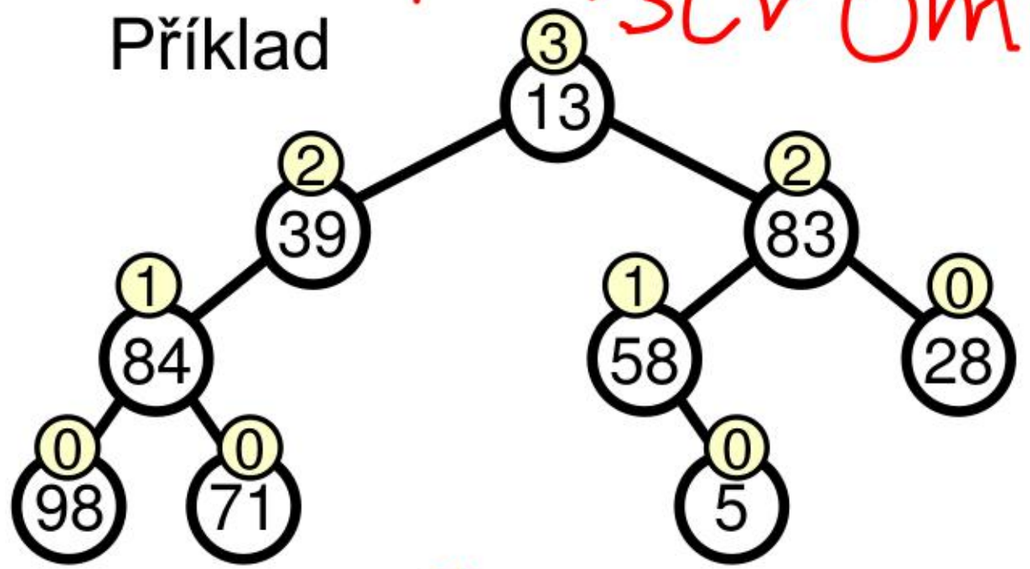
# Počítání hloubky

podstromu

Strom nebo podstrom




Příklad




0

max(-1,-1)+1=0

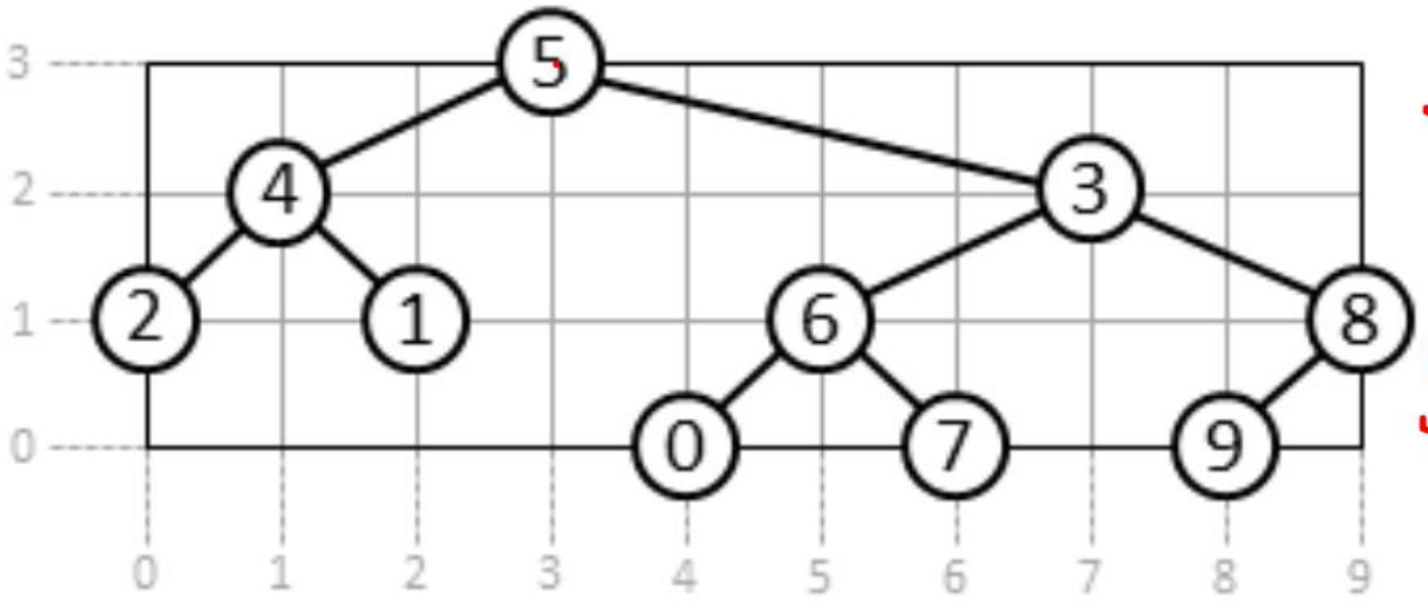
```
int depth(Node node) {
   if (node == null) return -1;
   return max(depth(node.left), depth(node.right))+1;
}
```

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

y: h(T) − h(u)

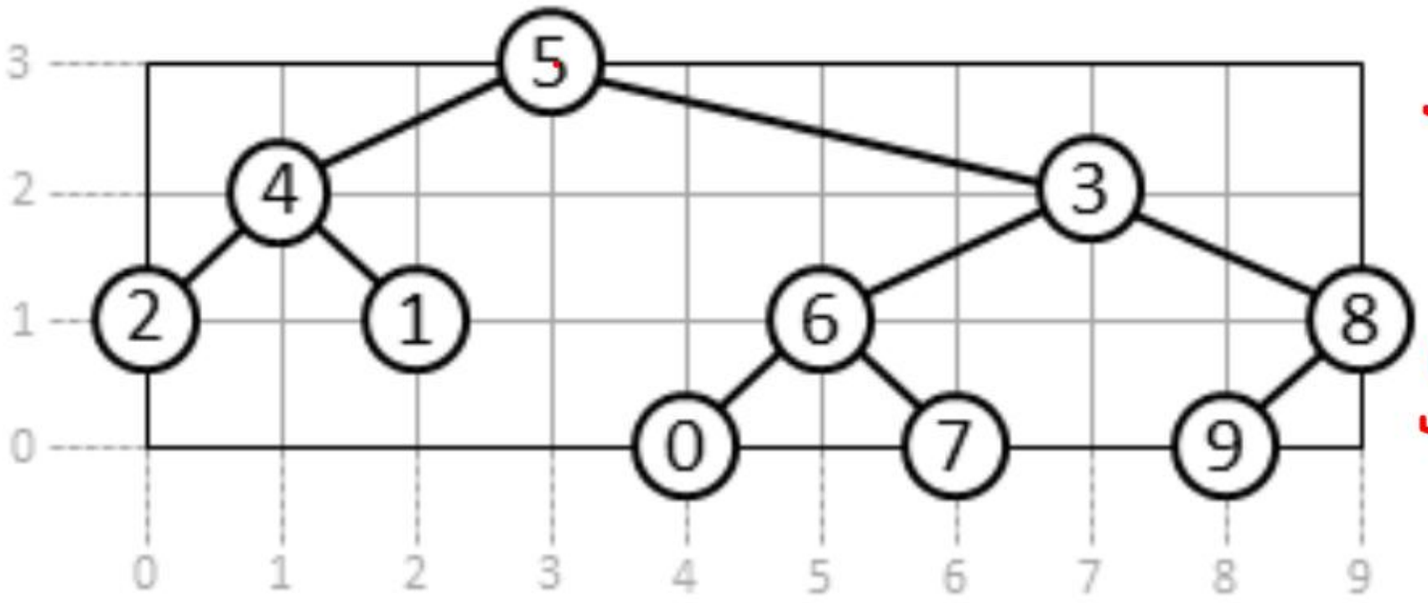
3 1

# Kreslení binárního stromu

Vstup: binární strom $T$ s $n$ uzly.

Jak určíme souřadnice uzlů (středů kružnic), aby

- každý uzel měl celočíselnou $y$-ovou souřadnici o $1$ větší než jeho potomci,
- všechny uzly měly navzájem různé celočíselné $x$-ové souřadnice od $0$ do $n-1$,
- každý levý (pravý) potomek měl $x$-ovou souřadnici menší (větší) než jeho rodič?




x: inorder(u)

y: h(T) − h(u)

3 1

# Nejdelší cesta ve stromě

Jak dokončíme tvrzení, aby bylo správné?

Pro každý binární kořenový strom $T$, který má alespoň $2$ listy, platí, že nejdelší cesta v $T$

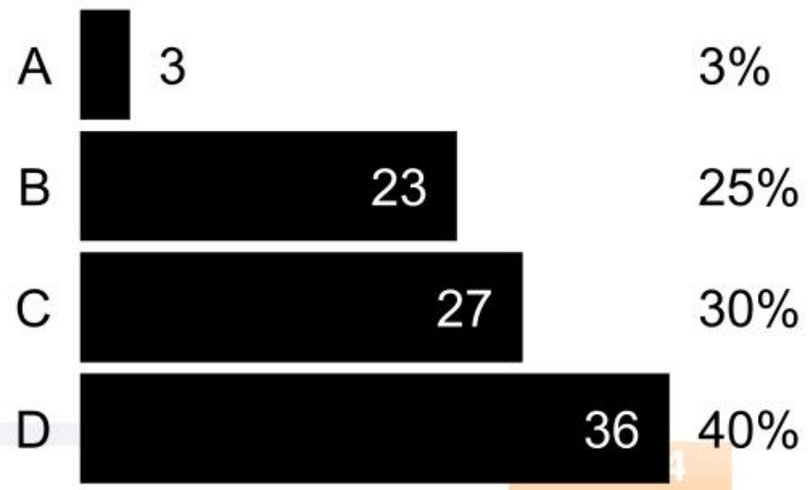
A. vždy spojuje kořen s listem.
B. vždy spojuje dva listy.
C. vždy spojuje list buď s jiným listem a nebo s kořenem.
D. vždy obsahuje kořen.

# Nejdelší cesta ve stromě

Jak dokončíme tvrzení, aby bylo správné?

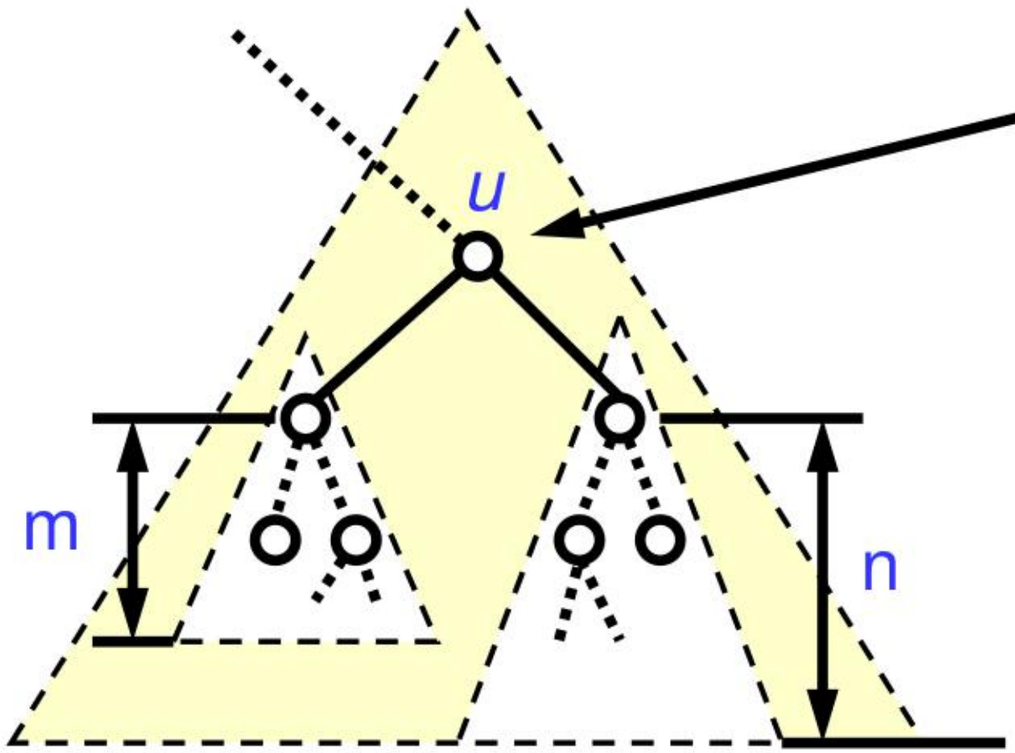
Pro každý binární kořenový strom $T$, který má alespoň $2$ listy, platí, že nejdelší cesta v $T$

A. vždy spojuje kořen s listem.
B. vždy spojuje dva listy.
C. vždy spojuje list buď s jiným listem a nebo s kořenem.
D. vždy obsahuje kořen.



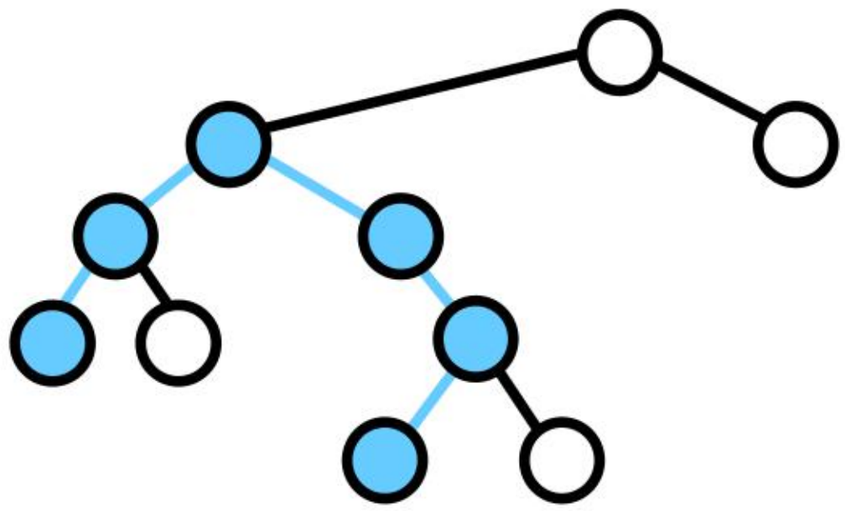
# Počítání nejdelší cesty

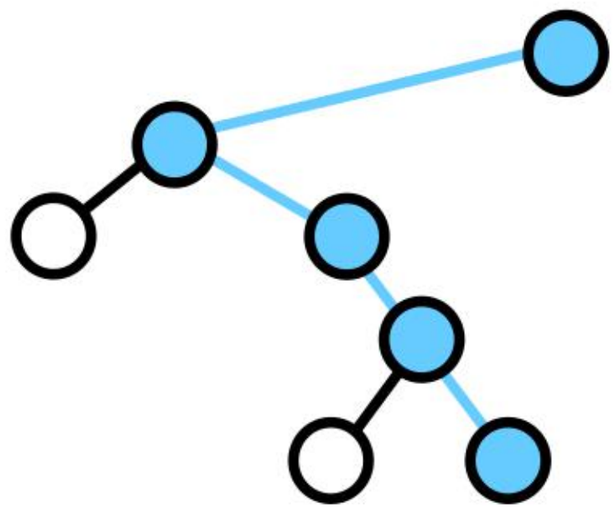
Strom nebo podstrom




m+n+2 … délka nejdelší cesty s nejvyšším bodem v uzlu *u*

Příklady



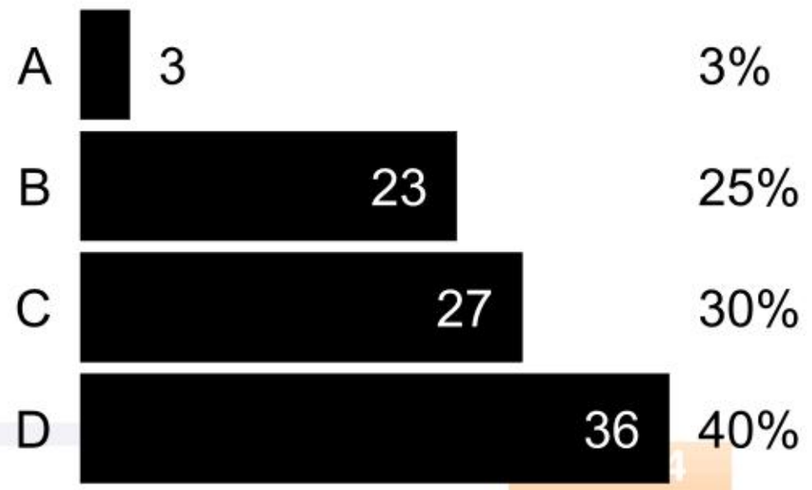


# Nejdelší cesta ve stromě

Jak dokončíme tvrzení, aby bylo správné?

Pro každý binární kořenový strom $T$, který má alespoň $2$ listy, platí, že nejdelší cesta v $T$

A. vždy spojuje kořen s listem.
B. vždy spojuje dva listy.
C. vždy spojuje list buď s jiným listem a nebo s kořenem.
D. vždy obsahuje kořen.

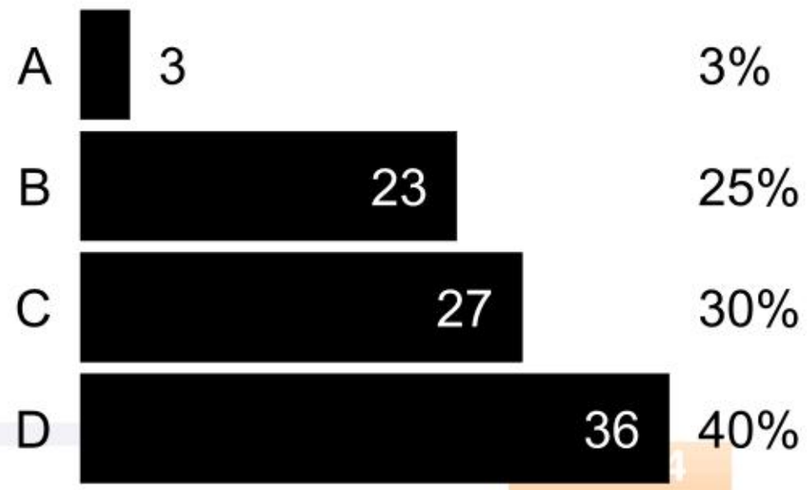


# Nejdelší cesta ve stromě

Jak dokončíme tvrzení, aby bylo správné?

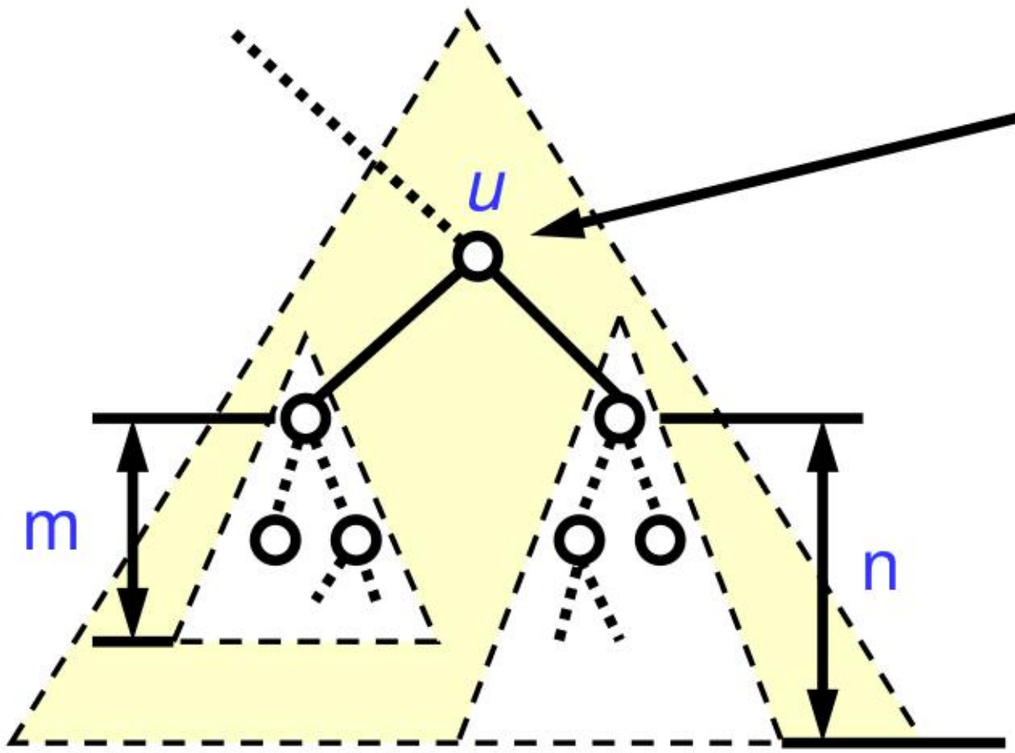
Pro každý binární kořenový strom $T$, který má alespoň $2$ listy, platí, že nejdelší cesta v $T$

A. vždy spojuje kořen s listem.
B. vždy spojuje dva listy.
C. vždy spojuje list buď s jiným listem a nebo s kořenem.
D. vždy obsahuje kořen.



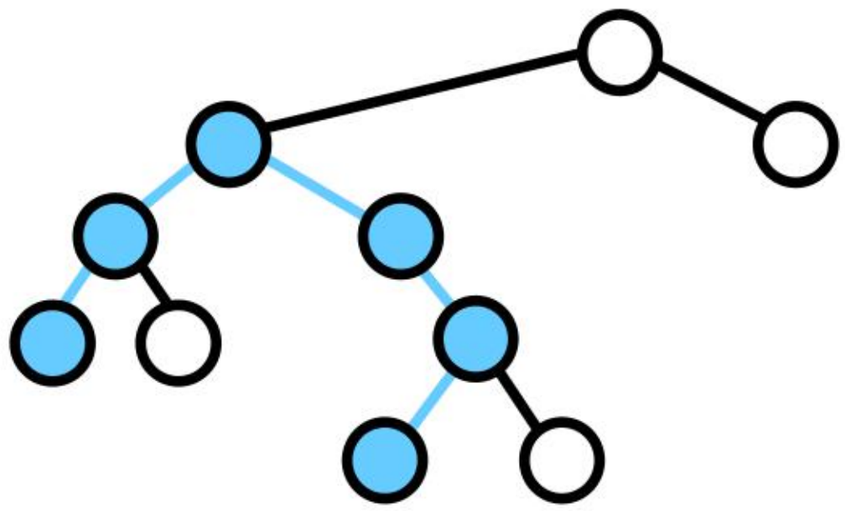
# Počítání nejdelší cesty

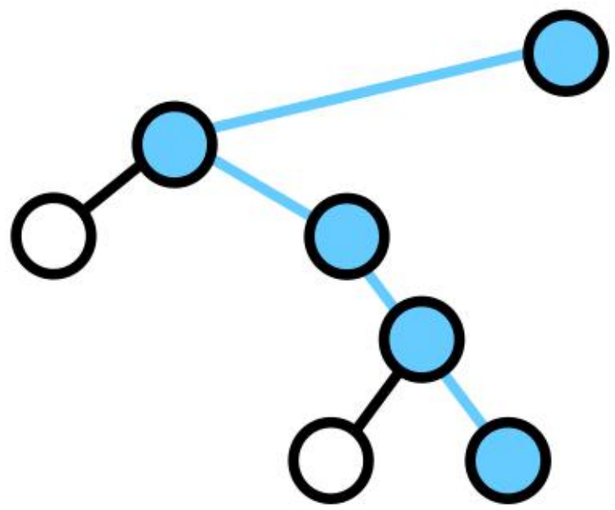
Strom nebo podstrom




m+n+2 … délka nejdelší cesty s nejvyšším bodem v uzlu *u*

Příklady

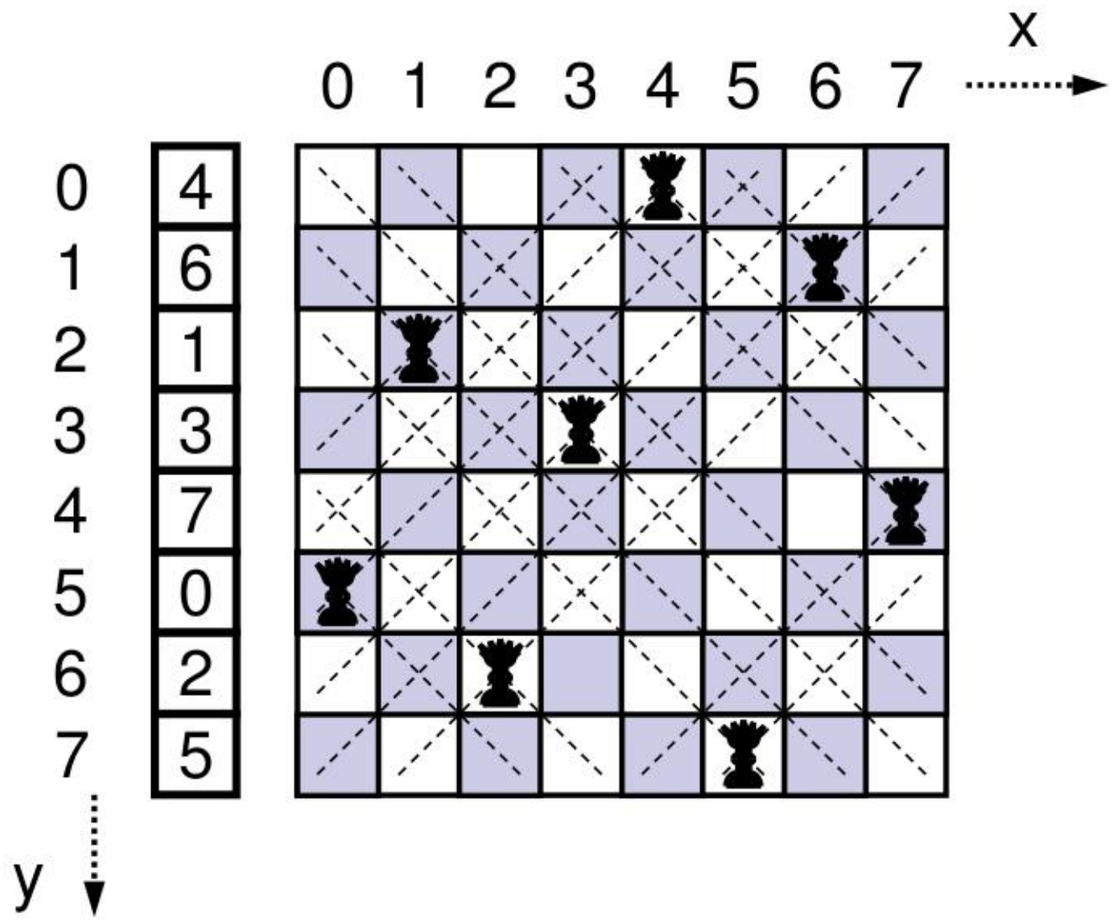


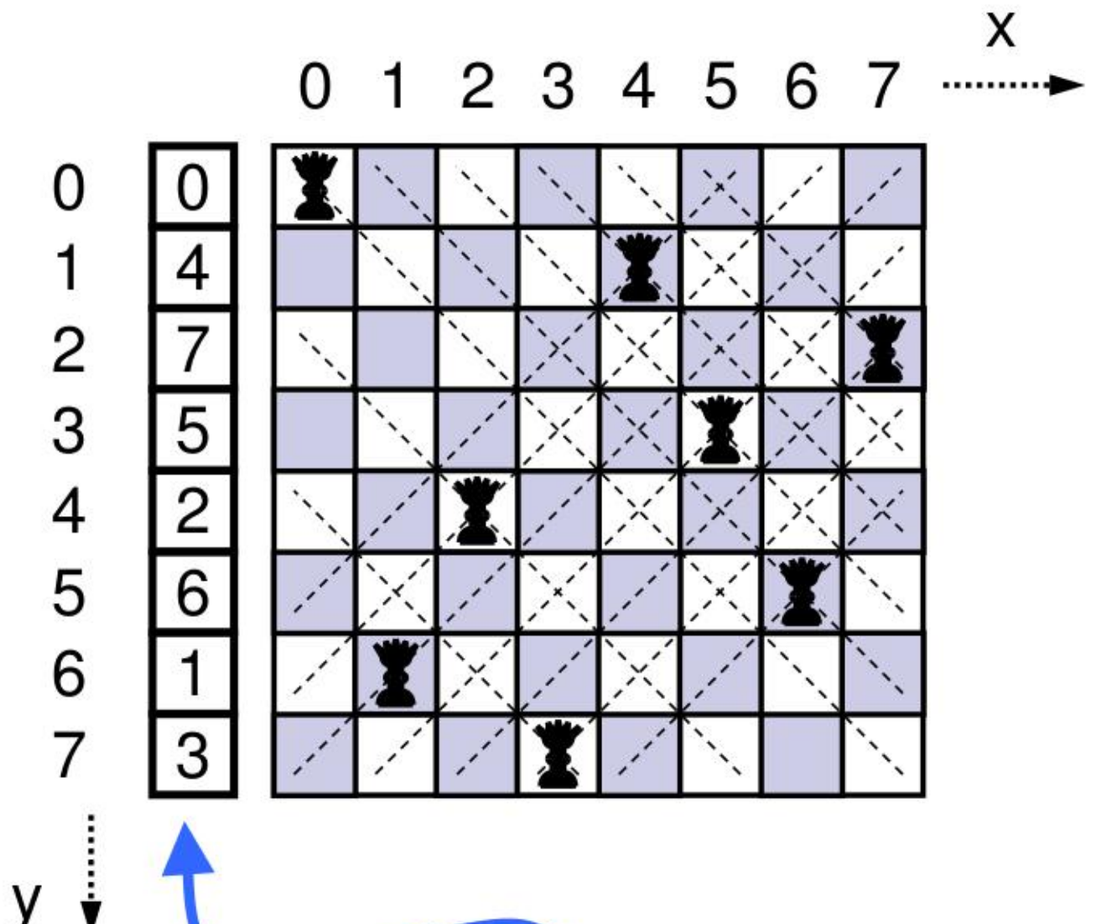


# Prohledávání s návratem

Problém osmi dam na šachovnici

Některá řešení:



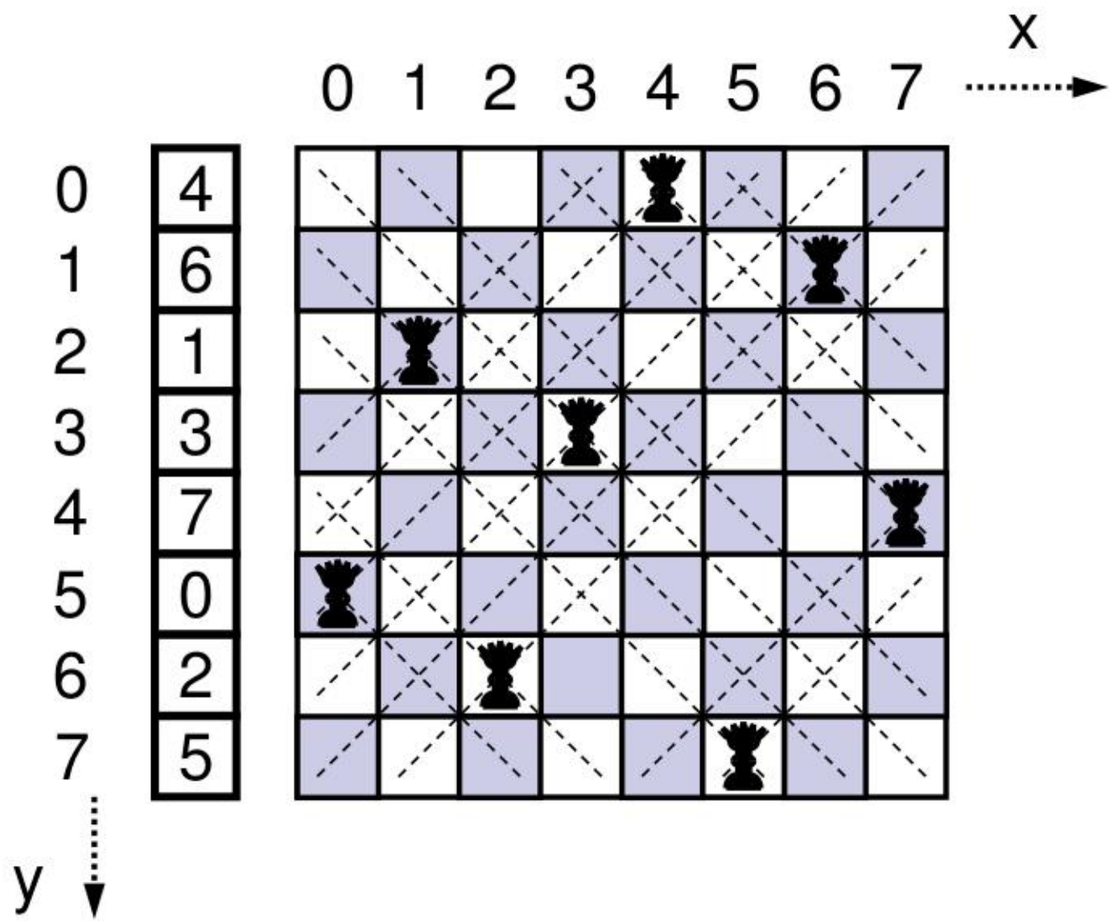



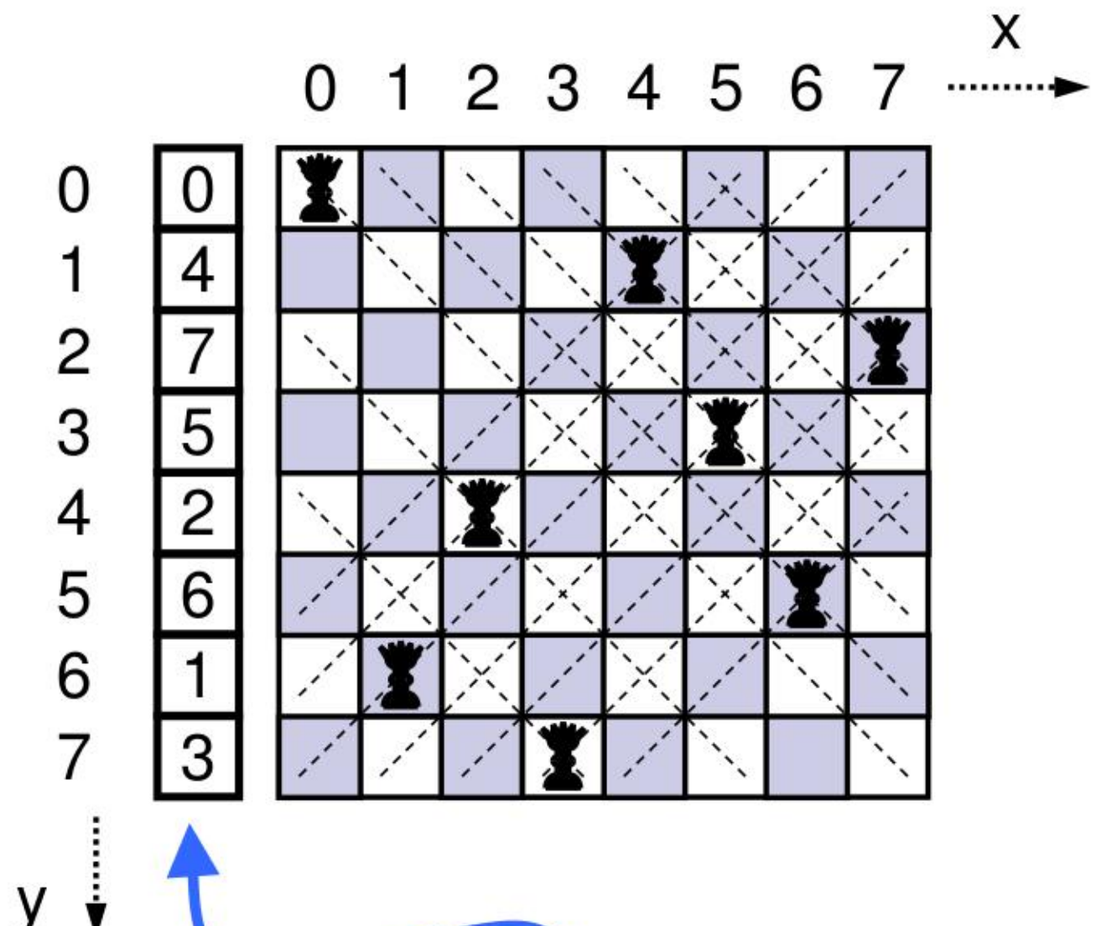

Jediná datová struktura xPosArray[ ] (viz kód)

# Prohledávání s návratem

Problém osmi dam na šachovnici

Některá řešení:

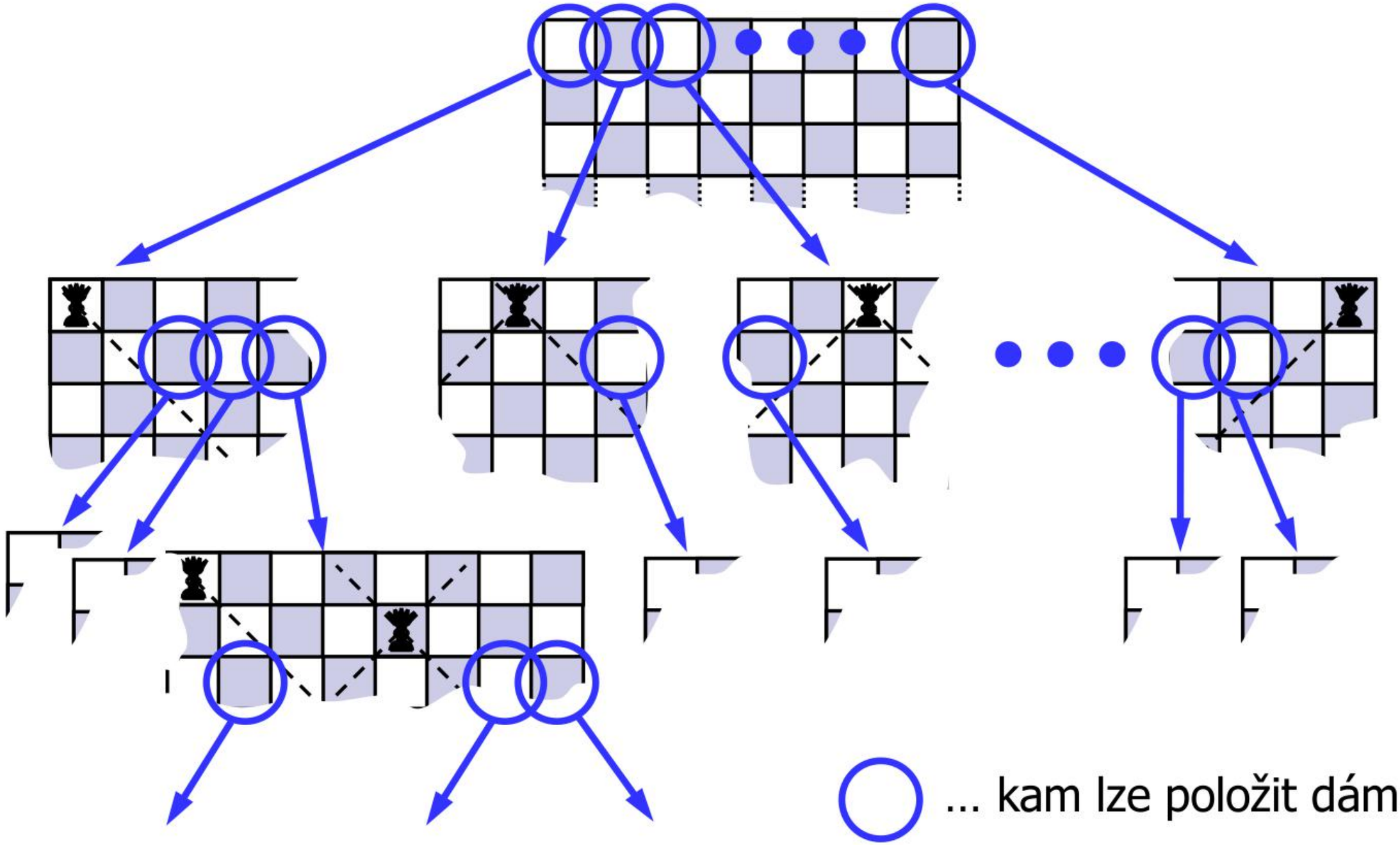



Jediná datová struktura xPosArray[ ] (viz kód)

# Prohledávání s návratem

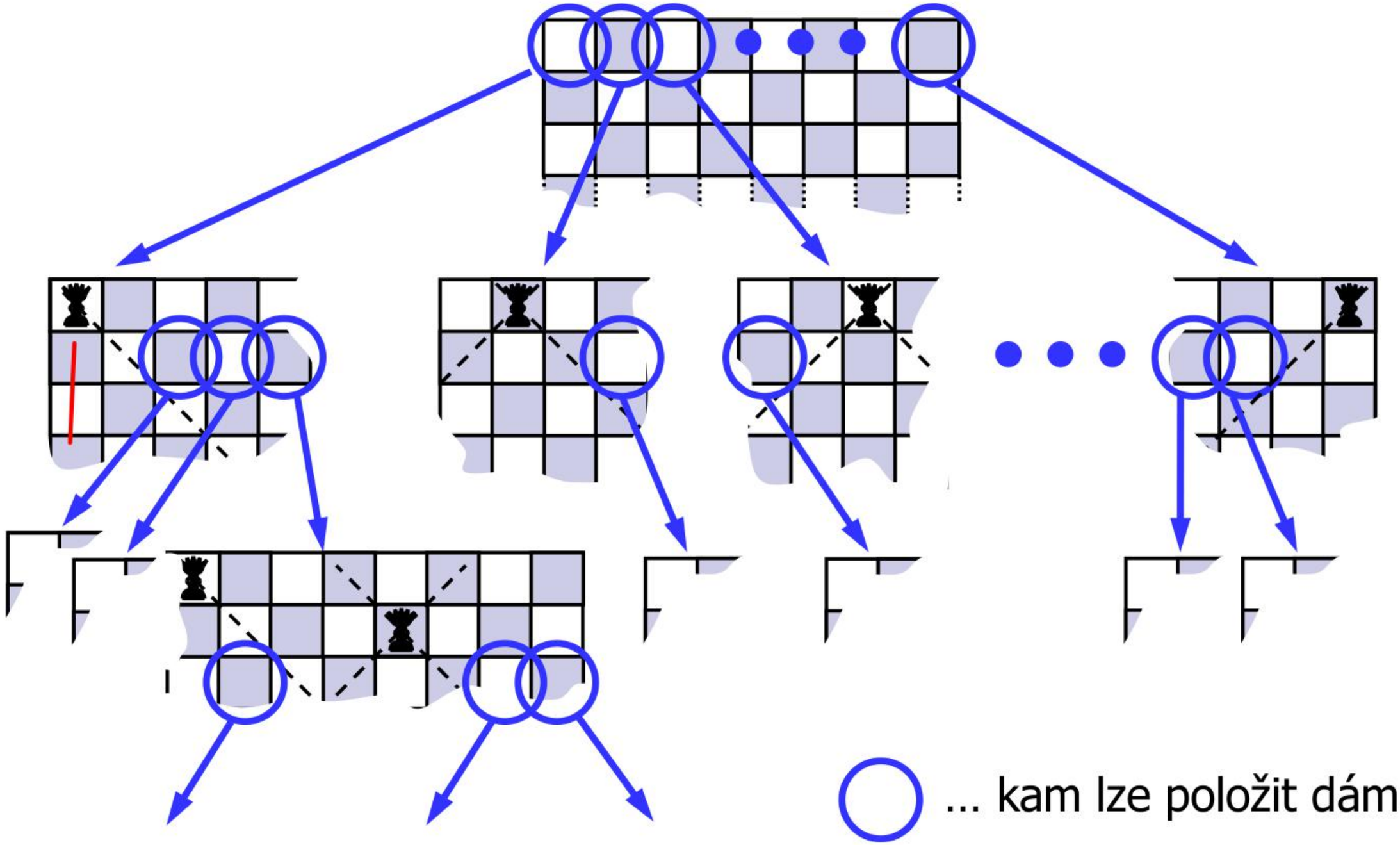
Strom testovaných pozic  (kořen a několik potomků)



... kam lze položit dámu

# Prohledávání s návratem

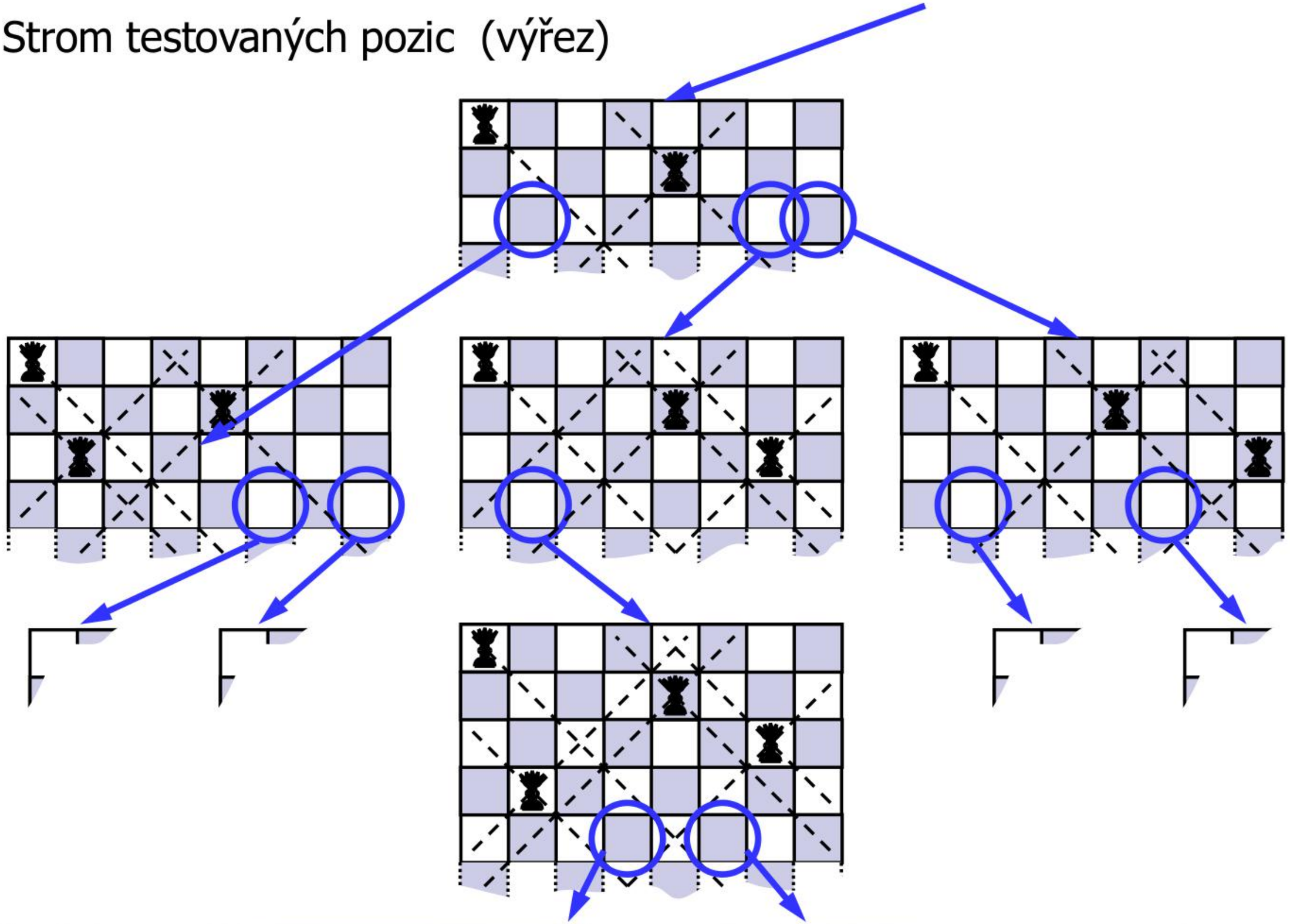
Strom testovaných pozic  (kořen a několik potomků)



... kam lze položit dámu

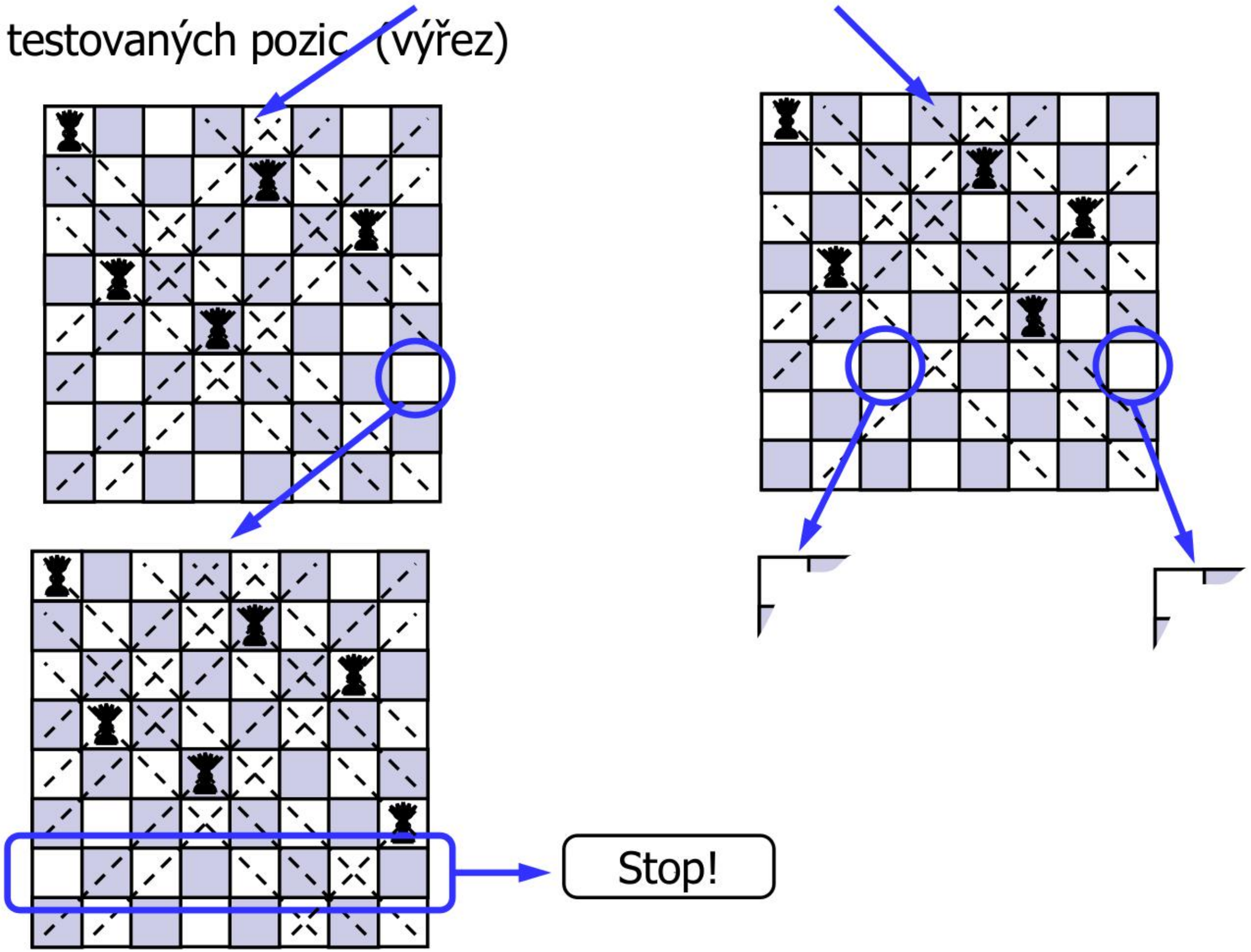
# Prohledávání s návratem

Strom testovaných pozic (výřez)



# Prohledávání s návratem

Strom testovaných pozic (výřez)



# Prohledávání s návratem

```
boolean posOK(int x, int y) {
    int i;
    for (i = 0; i < y; i++)
      if ((xPosArr[i] == x) ||                    // stejná řada
          (abs(y-i) == abs(xPosArr[i]-x)))        // nebo diagonála
        return false;
    return true;
}
```

```
void tryPutColumn(int y) {
    int x;
    if (y >= N) print_xPosArr();                  // řešení
    else
      for (x = 0; x < N; x++)                     // testuj sloupce
          if (posOK(x, y)) {                      // když je volno,
            xPosArr[y] = x;                       // dej tam dámu
            tryPutColumn(y + 1);                  // a volej rekurzi
 }        }
```

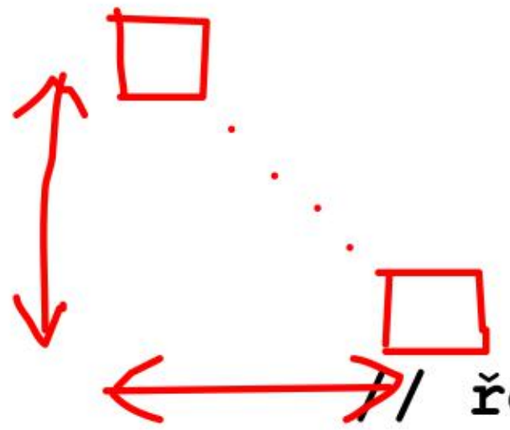
---

```
Call: tryPutColumn(0);
```

# Prohledávání s návratem

```
boolean posOK(int x, int y) {
    int i;
    for (i = 0; i < y; i++)
      if ((xPosArr[i] == x) ||                    // stejná řada
          (abs(y-i) == abs(xPosArr[i]-x)))        // nebo diagonála
        return false;
    return true;
}

void tryPutColumn(int y) {
    int x;
    if (y >= N) print_xPosArr();                  // řešení
    else
      for (x = 0; x < N; x++)                     // testuj sloupce
          if (posOK(x, y)) {                      // když je volno,
            xPosArr[y] = x;                       // dej tam dámu
            tryPutColumn(y + 1);                  // a volej rekurzi
 }        }
```



---

```
Call: tryPutColumn(0);
```

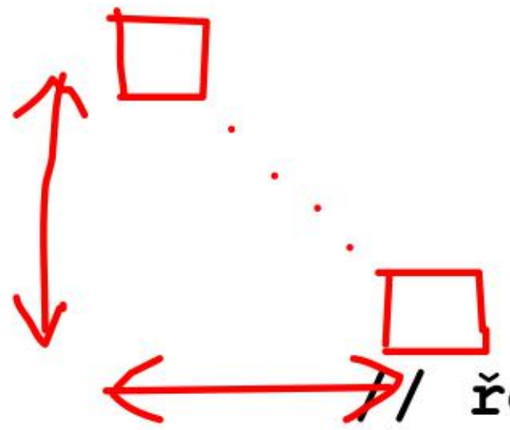
# Prohledávání s návratem

| N<br>poč. dam | Počet<br>řešení | Počet testovaných pozic dámy | | Zrychlení |
|---|---|---|---|---|
| | | Hrubá síla ($N^N$) | Backtrack | |
| **4** | **2** | **256** | **240** | **1.07** |
| **5** | **10** | **3 125** | **1 100** | **2.84** |
| **6** | **4** | **46 656** | **5 364** | **8.70** |
| **7** | **40** | **823 543** | **25 088** | **32.83** |
| **8** | **92** | **16 777 216** | **125 760** | **133.41** |
| **9** | **352** | **387 420 489** | **651 402** | **594.75** |
| **10** | **724** | **10 000 000 000** | **3 481 500** | **2 872.33** |
| **11** | **2 680** | **285 311 670 611** | **19 873 766** | **14 356.20** |
| **12** | **14 200** | **8 916 100 448 256** | **121 246 416** | **73 537.00** |

# Prohledávání s návratem

```
boolean posOK(int x, int y) {
    int i;
    for (i = 0; i < y; i++)
      if ((xPosArr[i] == x) ||                    // stejná řada
          (abs(y-i) == abs(xPosArr[i]-x)))        // nebo diagonála
        return false;
    return true;
}

void tryPutColumn(int y) {
    int x;
    if (y >= N) print_xPosArr();                  // řešení
    else
      for (x = 0; x < N; x++)                     // testuj sloupce
          if (posOK(x, y)) {                      // když je volno,
            xPosArr[y] = x;                       // dej tam dámu
            tryPutColumn(y + 1);                  // a volej rekurzi
 }        }
```



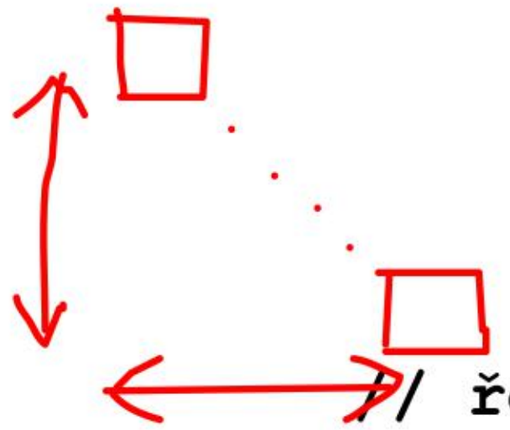
---

```
Call: tryPutColumn(0);
```

# Prohledávání s návratem

```
boolean posOK(int x, int y) {
    int i;
    for (i = 0; i < y; i++)
      if ((xPosArr[i] == x) ||                    // stejná řada
          (abs(y-i) == abs(xPosArr[i]-x)))        // nebo diagonála
        return false;
    return true;
}

void tryPutColumn(int y) {
    int x;
    if (y >= N) print_xPosArr();                  // řešení
    else
      for (x = 0; x < N; x++)                     // testuj sloupce
          if (posOK(x, y)) {                      // když je volno,
            xPosArr[y] = x;                       // dej tam dámu
            tryPutColumn(y + 1);                  // a volej rekurzi
 }        }
```



---

```
Call: tryPutColumn(0);
```

# Prohledávání s návratem

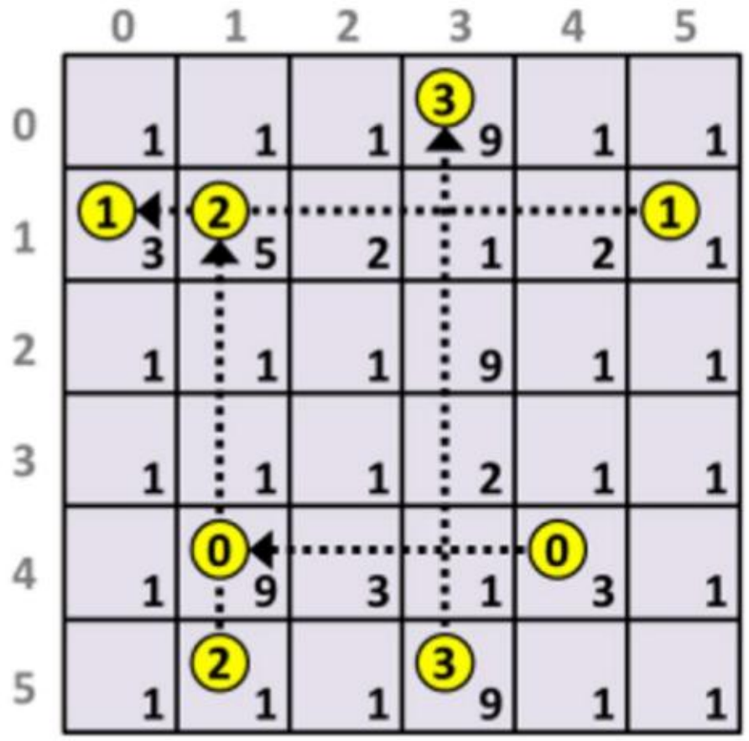
| N<br>poč. dam | Počet<br>řešení | Počet testovaných pozic dámy | | Zrychlení |
|---|---|---|---|---|
| | | Hrubá síla ($N^N$) | Backtrack | |
| **4** | **2** | **256** | **240** | **1.07** |
| **5** | **10** | **3 125** | **1 100** | **2.84** |
| **6** | **4** | **46 656** | **5 364** | **8.70** |
| **7** | **40** | **823 543** | **25 088** | **32.83** |
| **8** | **92** | **16 777 216** | **125 760** | **133.41** |
| **9** | **352** | **387 420 489** | **651 402** | **594.75** |
| **10** | **724** | **10 000 000 000** | **3 481 500** | **2 872.33** |
| **11** | **2 680** | **285 311 670 611** | **19 873 766** | **14 356.20** |
| **12** | **14 200** | **8 916 100 448 256** | **121 246 416** | **73 537.00** |

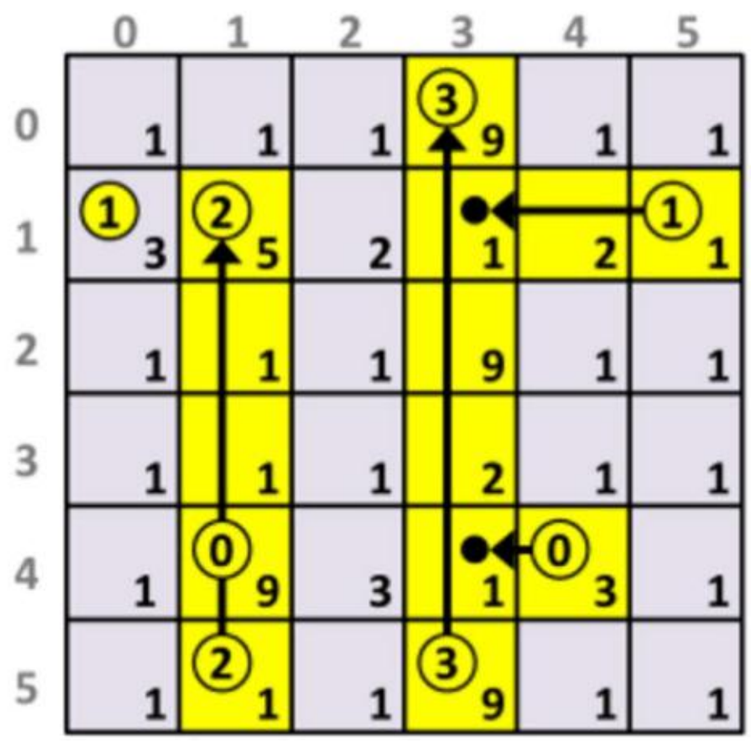
# Prohledávání s návratem

| N<br>poč. dam | Počet<br>řešení | Počet testovaných pozic dámy | | Zrychlení |
|---|---|---|---|---|
| | | Hrubá síla ($N^N$) | Backtrack | |
| **4** | **2** | **256** | **240** | **1.07** |
| **5** | **10** | **3 125** | **1 100** | **2.84** |
| **6** | **4** | **46 656** | **5 364** | **8.70** |
| **7** | **40** | **823 543** | **25 088** | **32.83** |
| **8** | **92** | **16 777 216** | **125 760** | **133.41** |
| **9** | **352** | **387 420 489** | **651 402** | **594.75** |
| **10** | **724** | **10 000 000 000** | **3 481 500** | **2 872.33** |
| **11** | **2 680** | **285 311 670 611** | **19 873 766** | **14 356.20** |
| **12** | **14 200** | **8 916 100 448 256** | **121 246 416** | **73 537.00** |

# Prohledávání s návratem

- Nanoroboti shromažďují vzorky v poli sektorů.
- Vzorky v sektorech mají různé hodnoty.
- Každý nanorobot má určen startovní a koncový sektor.
- Činnost nanorobota kontaminuje procházené sektory, jsou poté neprůchozí pro ostatní nanoroboty.
- V jakém pořadí nonoroboty sekvenčně aktivovat, aby hodnota sesbíraných vzorků byla maximální?




Zvolené pořadí:

3, 2, 0, 1

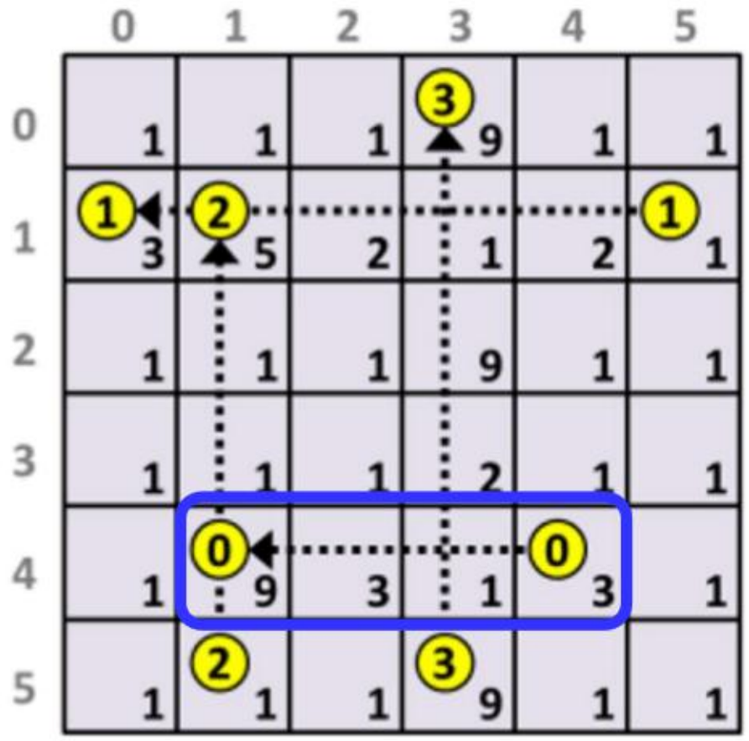
Hodnota vzorků:

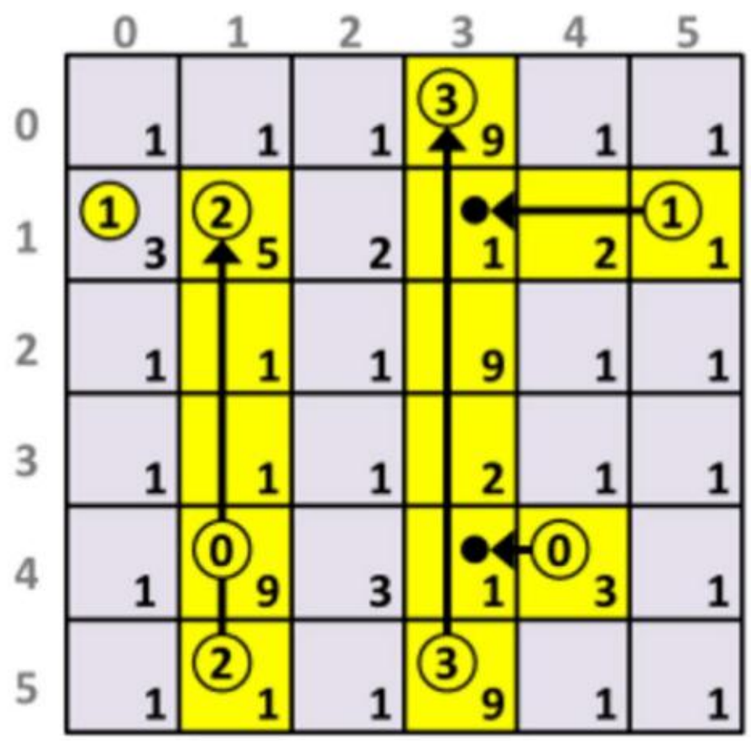
31+17+3+3=54

# Prohledávání s návratem

- Ořezávání stavového prostoru
  - Pamatujeme si dosud nejlepší nalezené řešení.
  - Pokud aktuálně prohledávané možnosti nemohou toto řešení vylepšit, provedeme návrat (k tomuto účelu stanovíme vhodný horní (dolní) odhad ceny řešení dosažitelného z aktuálního stavu).

Příklad: Je-li nanorobot 0 aktivován jako první, nebude hodnota nasbíraných vzorků větší než 16+(1+9+14)=40.





Zvolené pořadí:

3, 2, 0, 1

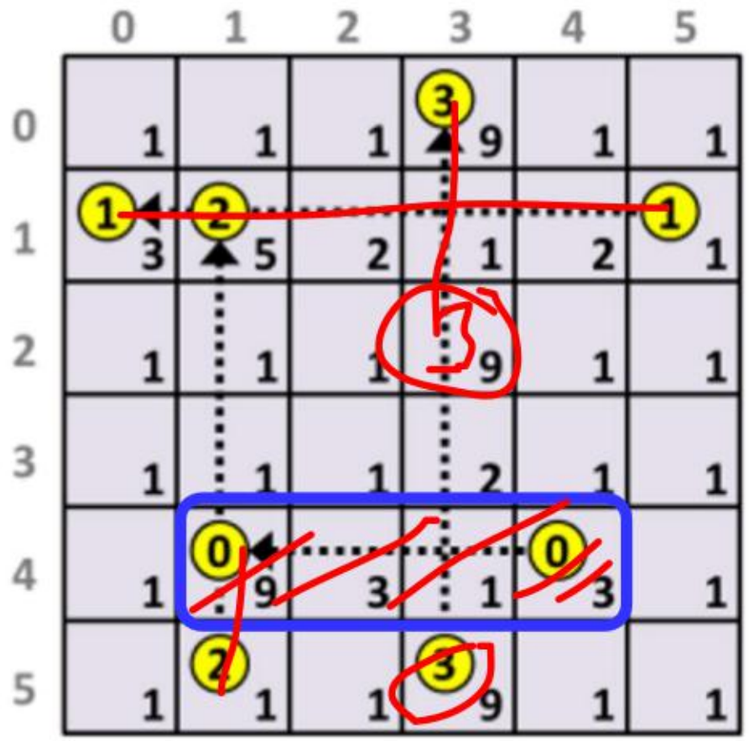
Hodnota vzorků:

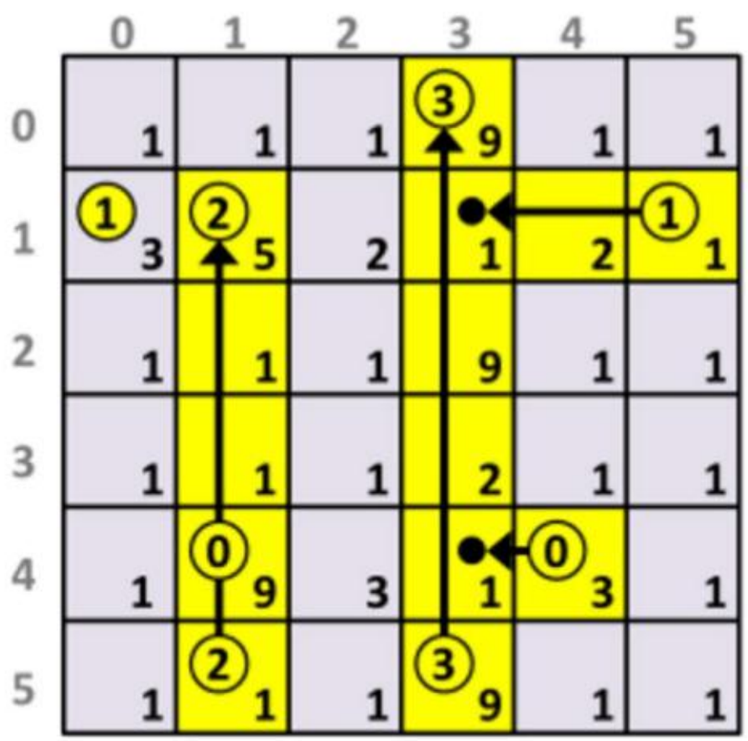
31+17+3+3=54

# Prohledávání s návratem

- Ořezávání stavového prostoru
  - Pamatujeme si dosud nejlepší nalezené řešení.
  - Pokud aktuálně prohledávané možnosti nemohou toto řešení vylepšit, provedeme návrat (k tomuto účelu stanovíme vhodný horní (dolní) odhad ceny řešení dosažitelného z aktuálního stavu).

Příklad: Je-li nanorobot 0 aktivován jako první, nebude hodnota nasbíraných vzorků větší než 16+(1+9+14)=40.





Zvolené pořadí:

3, 2, 0, 1

Hodnota vzorků:

31+17+3+3=54

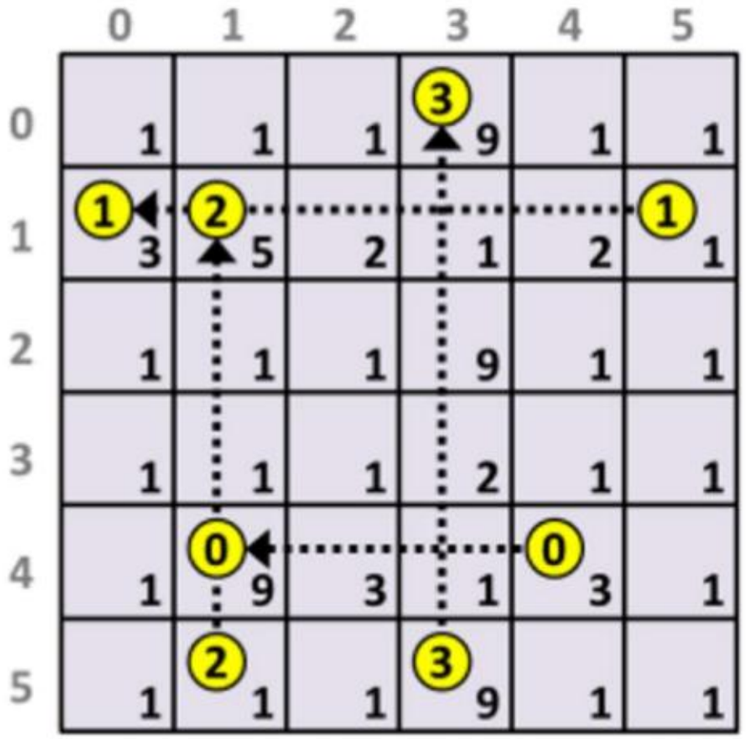
# Prohledávání s návratem

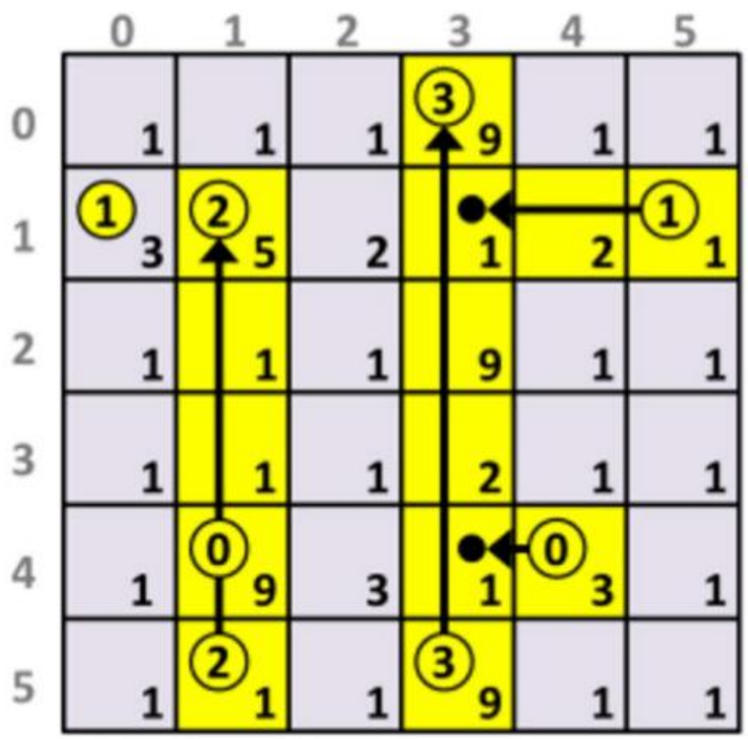
- Heuristiky
  - „Dobré“ řešení chceme naleznout co nejdříve, aby ořezávání bylo co nejefektivnější.

Příklad heuristiky: Zjistíme hodnotu vzorků, kterou jednotlivé nanoroboty sesbírají, pokud navštíví všechny sektory na jejich naplánované cestě.

Podle těchto hodnot nanoroboty uspořádáme sestupně. Stavový prostor procházíme podle získaného pořadí.

3 (31), 2 (17), 0 (16), 1 (14)





Zvolené pořadí:

3, 2, 0, 1

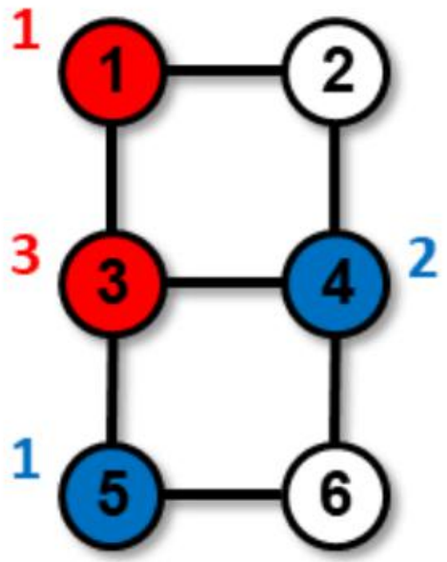
Hodnota vzorků:

31+17+3+3=54

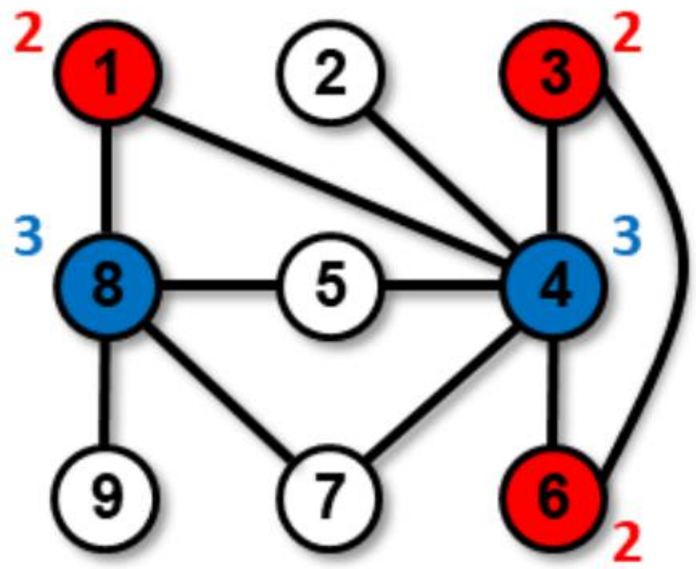
# Druhá domácí úloha

- Pro agenta typu $T_1$ je jeho skóre rovno počtu sousedních vrcholů obsazených agentem jakéhokoliv typu.
- Pro agenta typu $T_2$ je jeho skóre rovno počtu sousedních vrcholů neobsazených agentem.
- Pro daný počet agentů typu $T_1$ a $T_2$ je cílem maximalizovat skóre sítě definované jako součet skóre všech agentů.

a)




b)



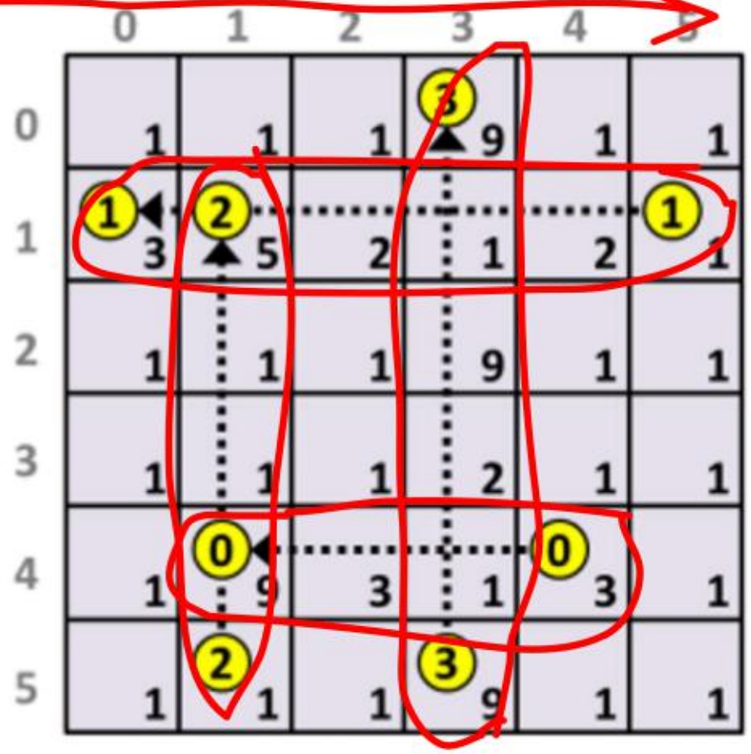
# Prohledávání s návratem

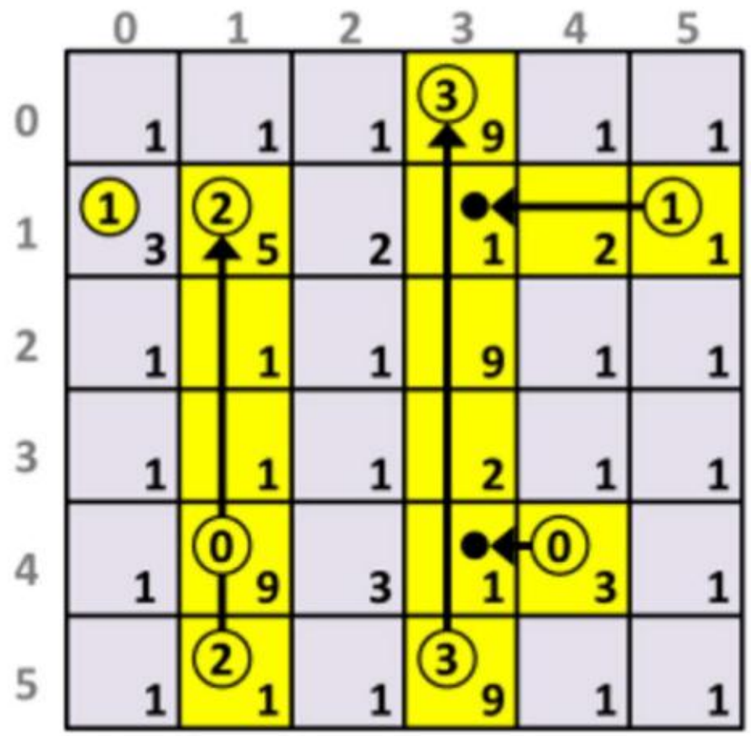
- Heuristiky
  - „Dobré“ řešení chceme naleznout co nejdříve, aby ořezávání bylo co nejefektivnější.

Příklad heuristiky: Zjistíme hodnotu vzorků, kterou jednotlivé nanoroboty sesbírají, pokud navštíví všechny sektory na jejich naplánované cestě.

Podle těchto hodnot nanoroboty uspořádáme sestupně. Stavový prostor procházíme podle získaného pořadí.

3 (31), 2 (17), 0 (16), 1 (14)





Zvolené pořadí:

3, 2, 0, 1

Hodnota vzorků:

31+17+3+3=54

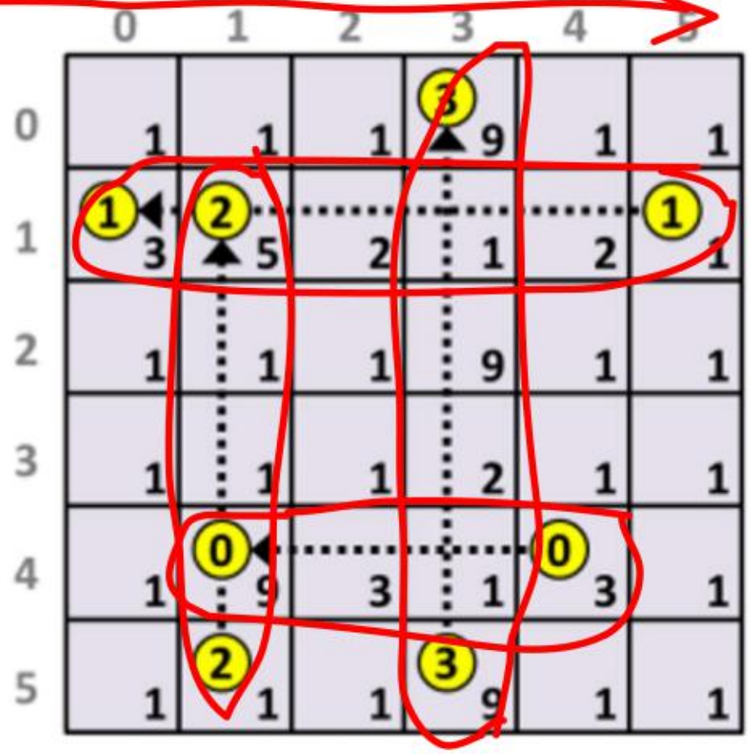
# Prohledávání s návratem

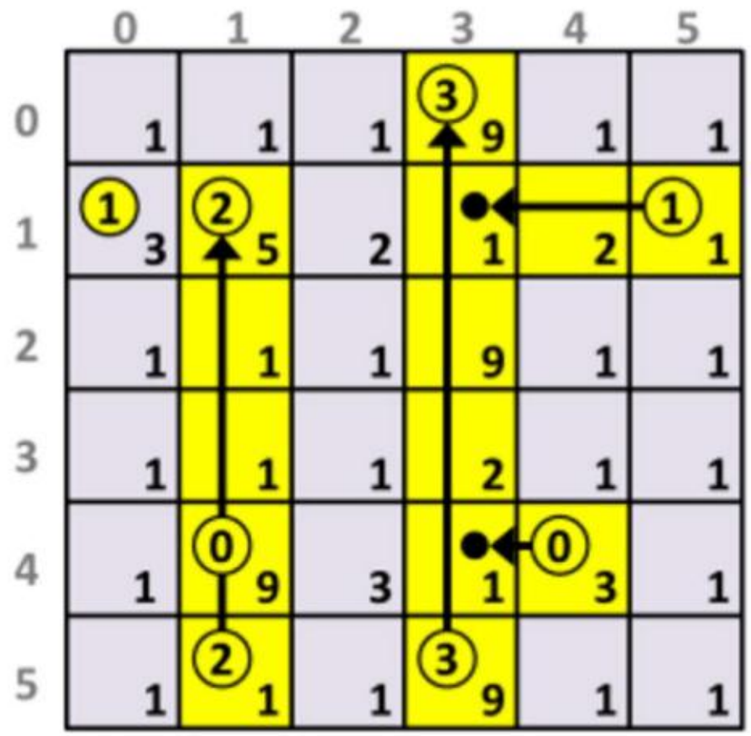
- Heuristiky
  - „Dobré“ řešení chceme naleznout co nejdříve, aby ořezávání bylo co nejefektivnější.

Příklad heuristiky: Zjistíme hodnotu vzorků, kterou jednotlivé nanoroboty sesbírají, pokud navštíví všechny sektory na jejich naplánované cestě.

Podle těchto hodnot nanoroboty uspořádáme sestupně. Stavový prostor procházíme podle získaného pořadí.

3 (31), 2 (17), 0 (16), 1 (14)





Zvolené pořadí:

3, 2, 0, 1

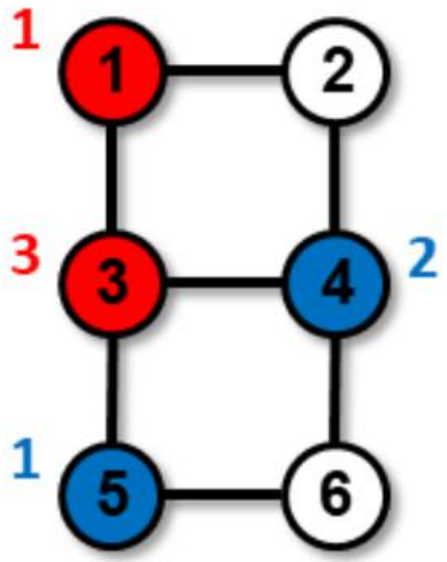
Hodnota vzorků:

31+17+3+3=54

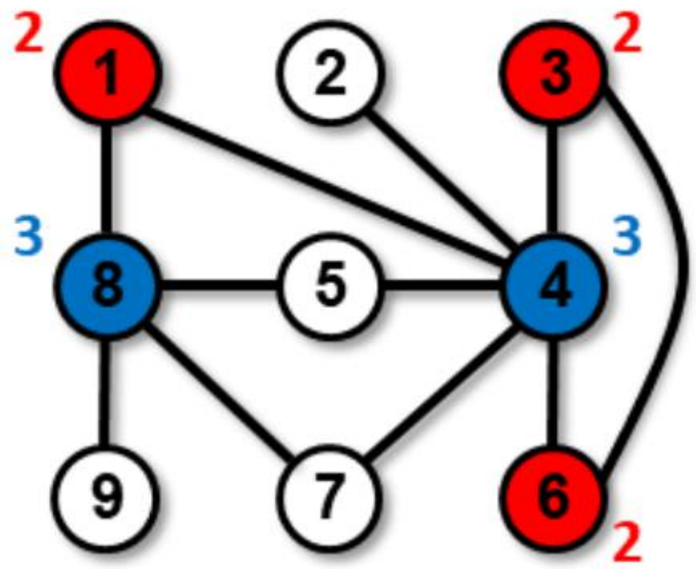
# Druhá domácí úloha

- Pro agenta typu $T_1$ je jeho skóre rovno počtu sousedních vrcholů obsazených agentem jakéhokoliv typu.
- Pro agenta typu $T_2$ je jeho skóre rovno počtu sousedních vrcholů neobsazených agentem.
- Pro daný počet agentů typu $T_1$ a $T_2$ je cílem maximalizovat skóre sítě definované jako součet skóre všech agentů.

a)



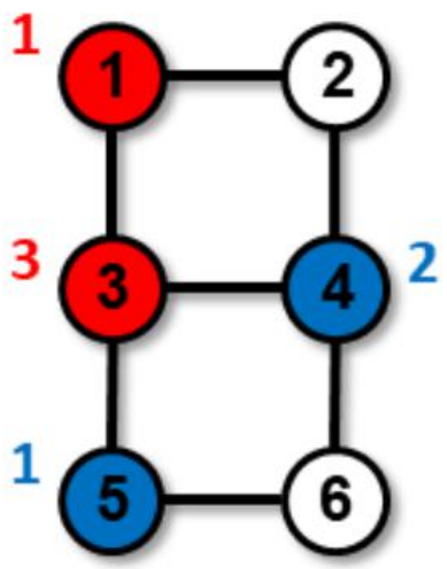

b)



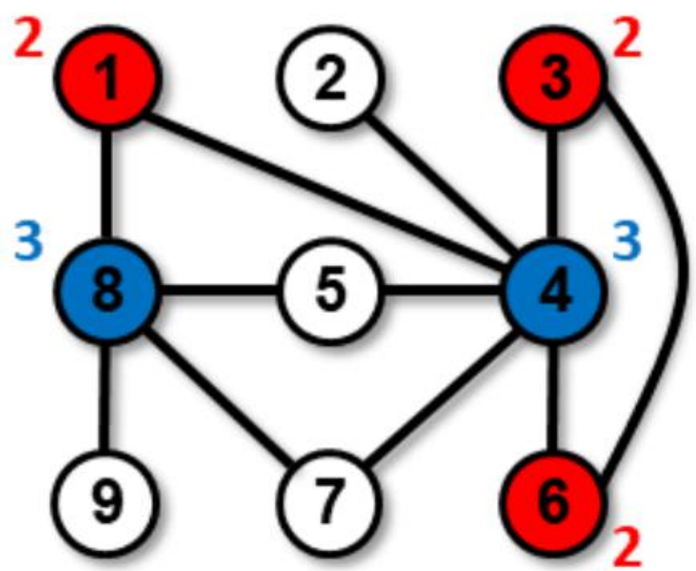
# Druhá domácí úloha

- Pro agenta typu $T_1$ je jeho skóre rovno počtu sousedních vrcholů obsazených agentem jakéhokoliv typu.
- Pro agenta typu $T_2$ je jeho skóre rovno počtu sousedních vrcholů neobsazených agentem.
- Pro daný počet agentů typu $T_1$ a $T_2$ je cílem maximalizovat skóre sítě definované jako součet skóre všech agentů.

a)




b)



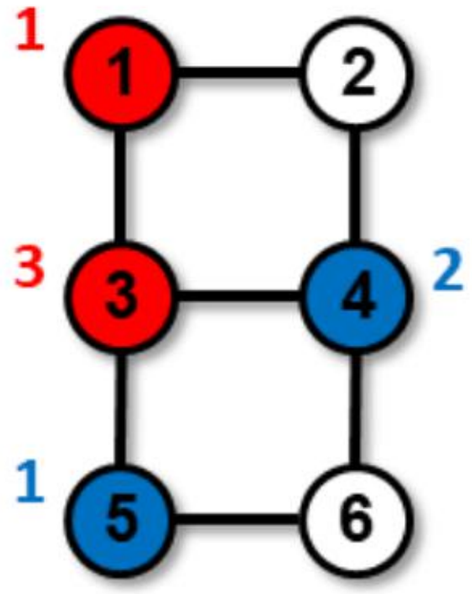
# Druhá domácí úloha

Vstup:

```
6 7 2 2
1 2
3 4
5 6
3 1
3 5
4 2
4 6
```

Reprezentace grafu:

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



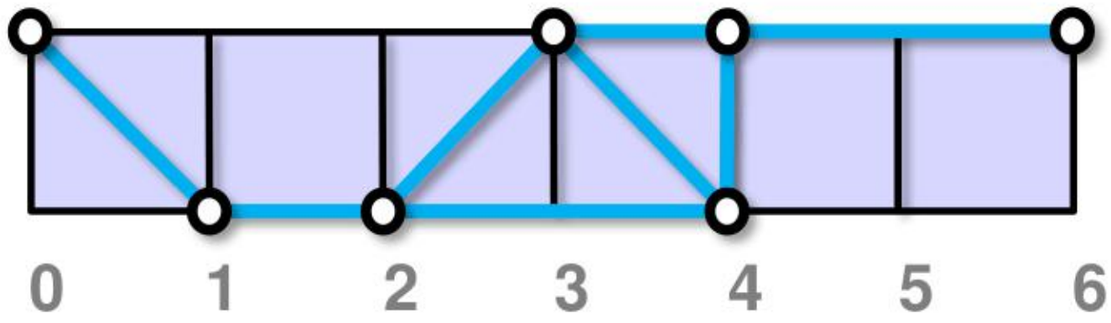

Vstupní graf má maximálně 30 uzlů.

# Motorové čluny

10 instancí: 44 řešitelů

9 instancí: 8 řešitelů

8 instancí: 22 řešitelů




Postup pro jeden břeh, D=8

souřadnice: 1 4 5 6 8 9 10 11 13 16

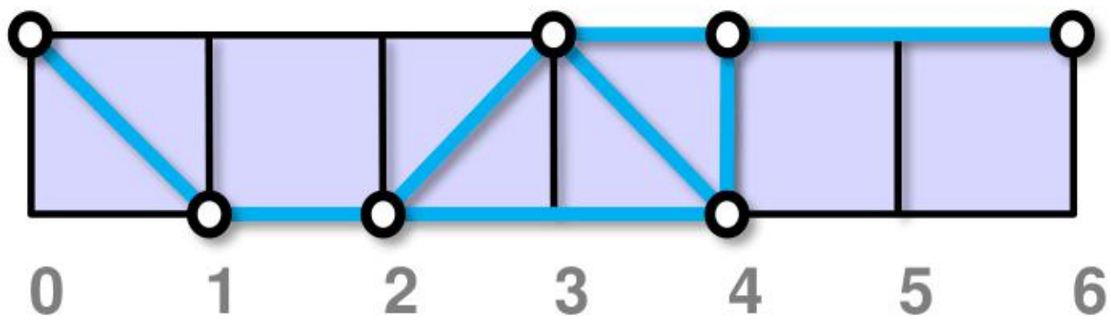
indexy: 0 1 5 7

# Motorové čluny

10 instancí: 44 řešitelů

9 instancí: 8 řešitelů

8 instancí: 22 řešitelů




Postup pro jeden břeh, D=8

souřadnice: 1 4 5 6 8 9 10 11 13 16

indexy: 0 1 5 7