

Stromy

Karel Richta a kol.

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

© Karel Richta a kol., 2018

Datové struktury a algoritmy, B6B36DSA
01/2018, Lekce 9

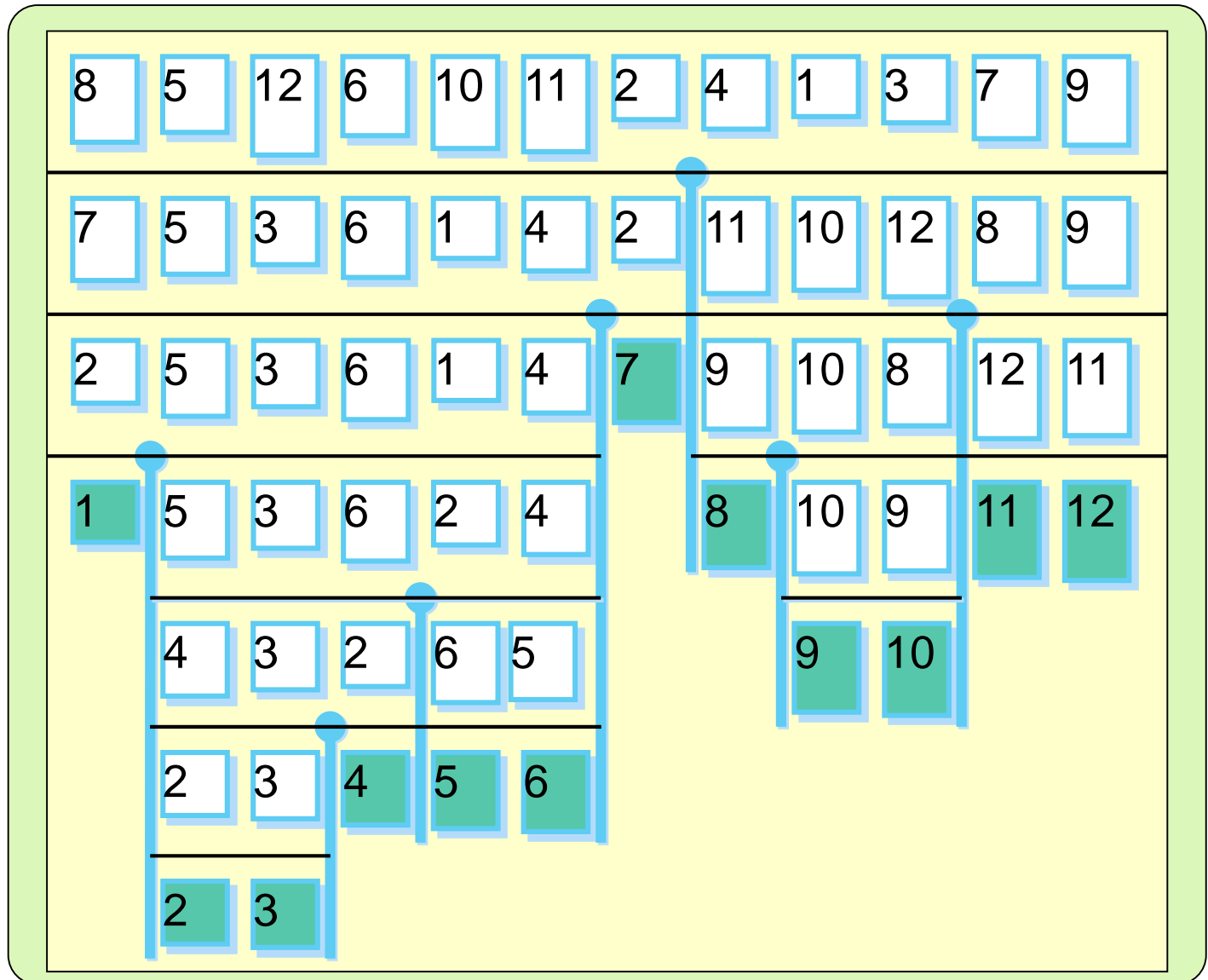
<https://cw.fel.cvut.cz/wiki/courses/b6b36dsa/start>



Připomenutí: QUICKSORT

Ukázka průběhu

pivot =
= první
v úseku



Motivace: QUICKSORT

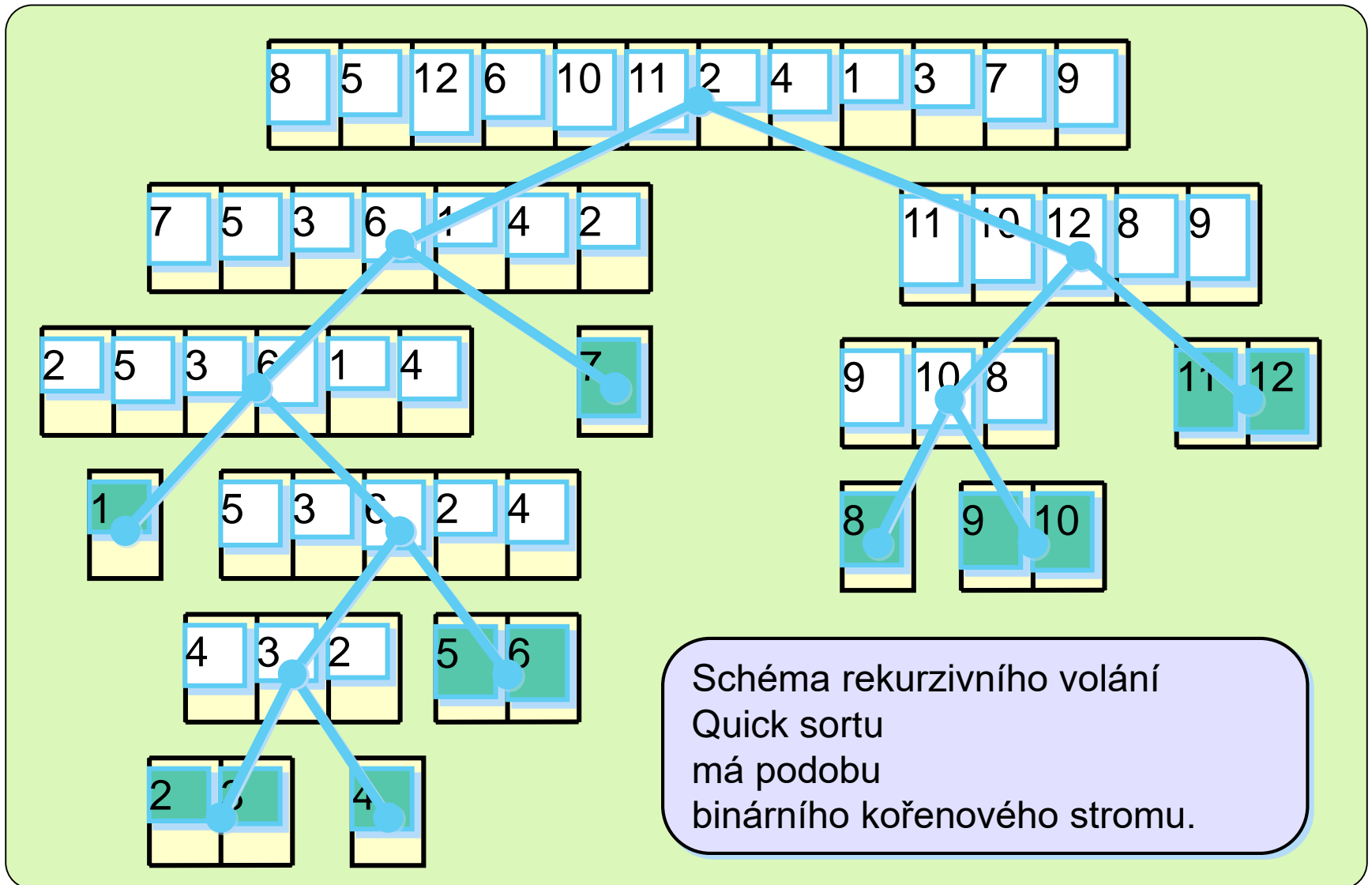
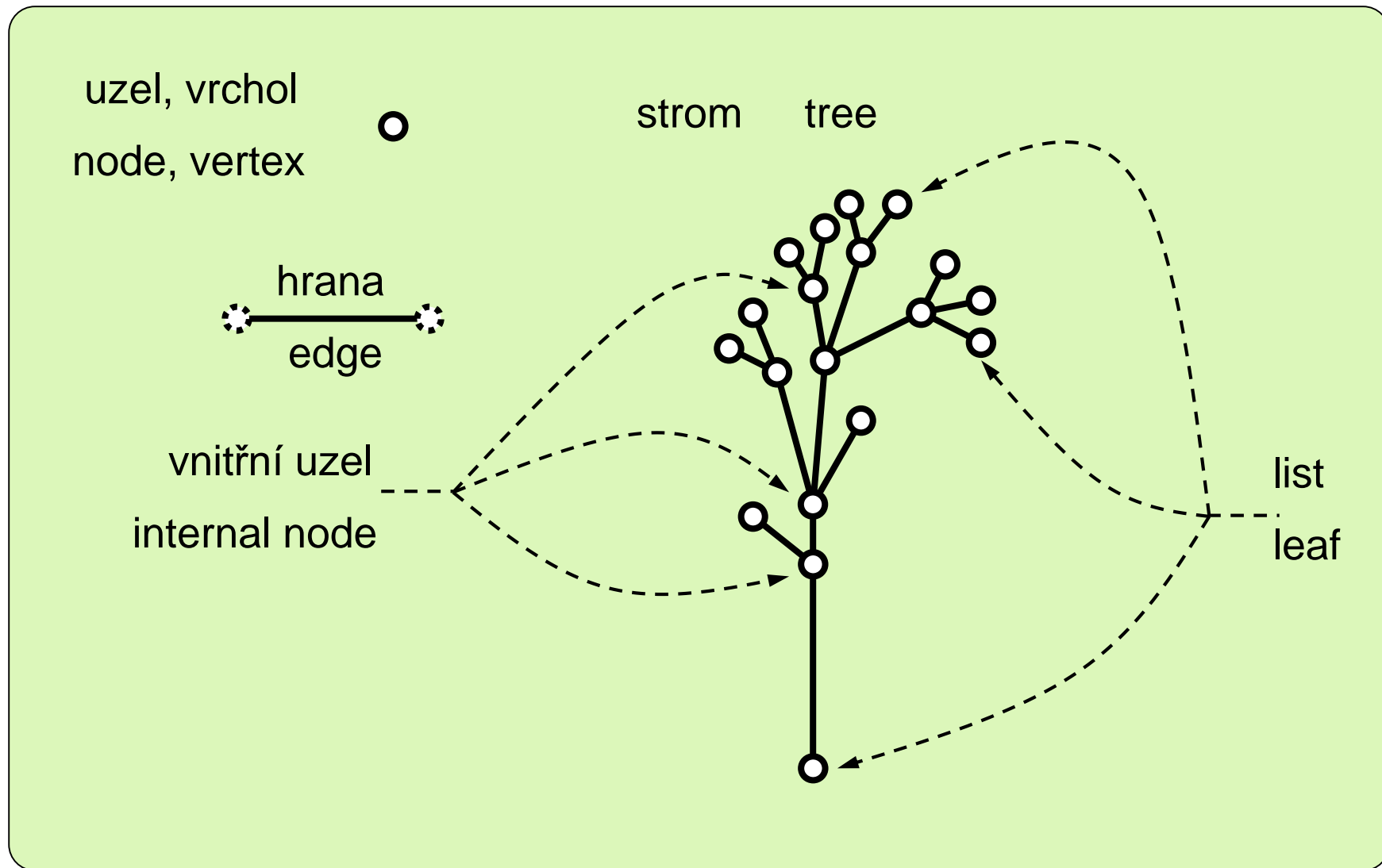


Schéma rekurzivního volání Quick sortu má podobu binárního kořenového stromu.

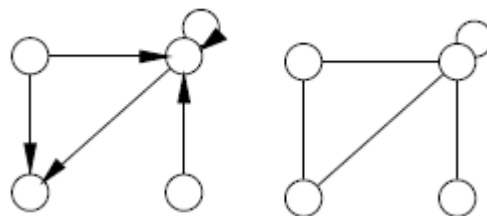
Strom



Orientované a neorientované grafy

Definice

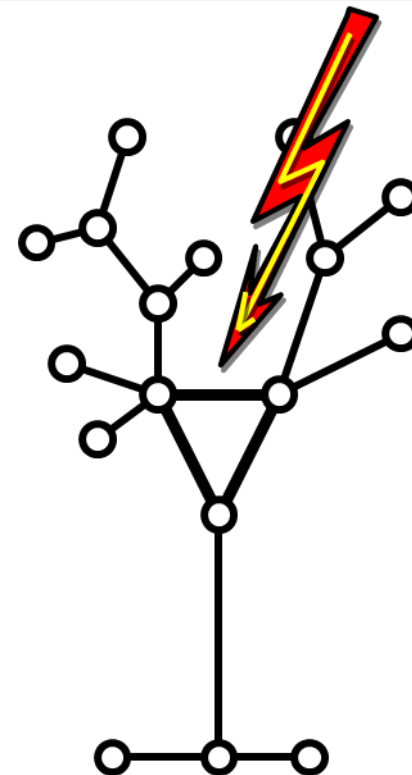
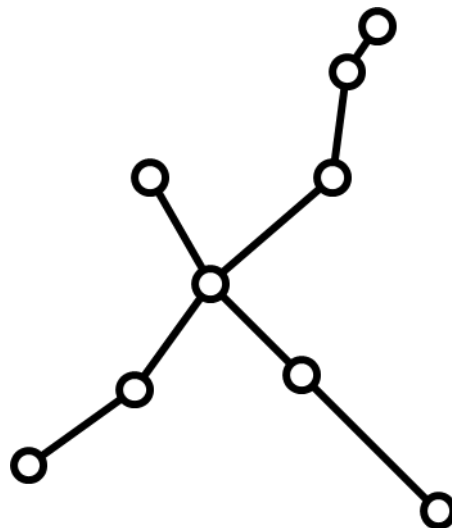
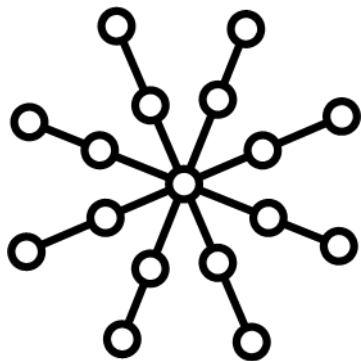
- **Orientovaný graf** o n uzlech je dvojice $G = (V(G), E(G))$, kde
 - ▶ $V(G)$ je množina uzlů, $n = |V(G)|$, a
 - ▶ $E(G) = \{\langle u, v \rangle; u, v \in V(G)\} \subset V(G) \times V(G) =$ množina orientovaných hran, čili uspořádaných dvojic uzlů (graficky šipek).
 - ▶ Orientovaná hrana $\langle u, u \rangle$ se nazývá smyčka.
- **(Neorientovaný obyčejný) graf** o n uzlech je dvojice $G = (V(G), E(G))$, kde $V(G)$ je množina uzlů a
 - ▶ $E(G) = \{\{u, v\}; u, v \in V(G), u \neq v\}$
= množina neorientovaných hran, čili neuspořádaných dvojic uzlů.
 - ▶ Smyčky se někdy povolují, někdy ne..



Definice stromu (volného stromu)

Definice

- **Volný strom** je každý souvislý acyklický a neorientovaný graf.
- **Les** je každý acyklický a neorientovaný graf.



Vlastnosti volných stromů

Věta

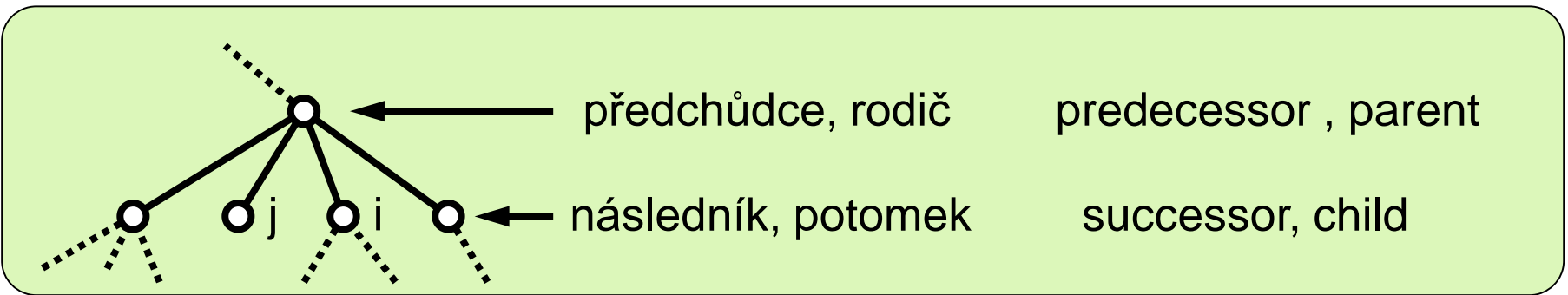
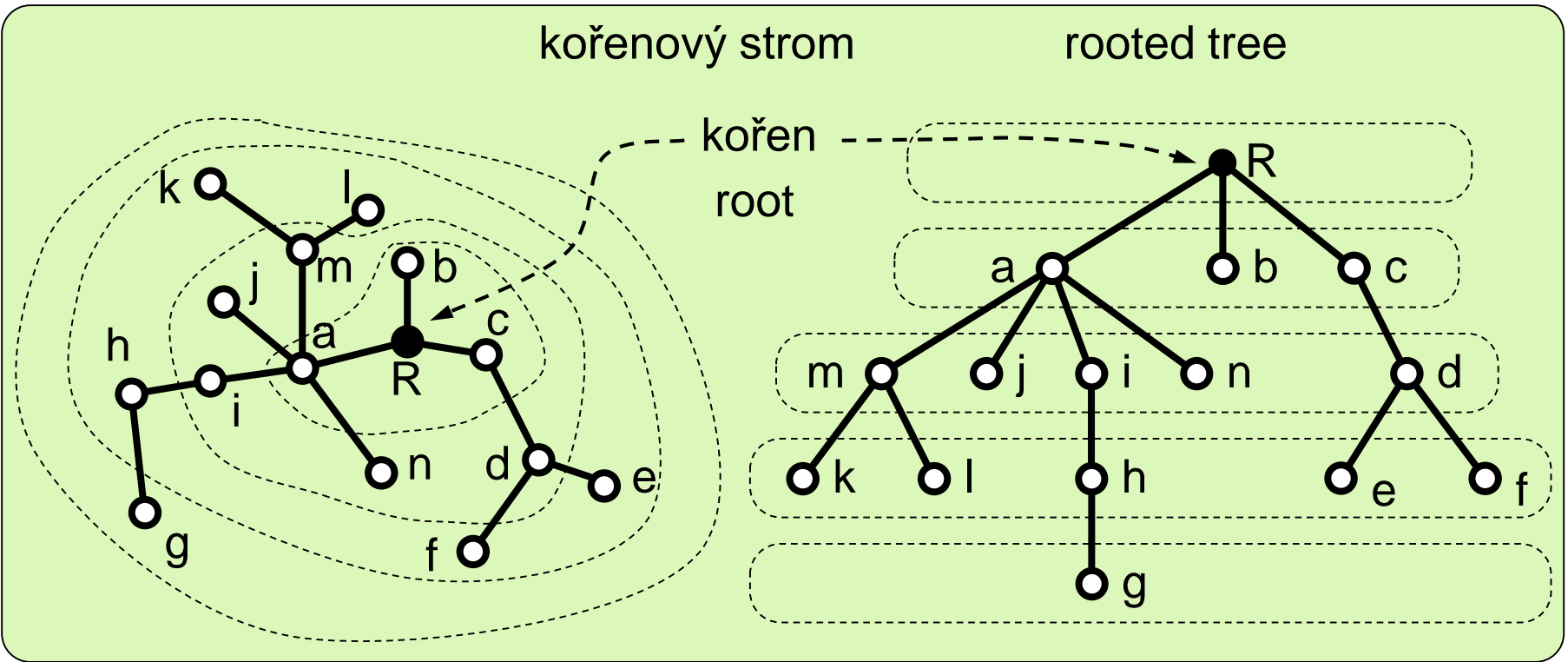
Nechť $G = (V(G), E(G))$ je neorientovaný graf. Pak následující tvrzení jsou ekvivalentní.

- 1 G je volný strom.
- 2 Jakékoli 2 uzly v G jsou spojeny jedinečnou jednoduchou cestou.
- 3 G je souvislý, ale pokud vyjmeme jakoukoli hranu z $E(G)$, výsledný graf bude nesouvislý.
- 4 G je souvislý a $|E(G)| = |V(G)| - 1$.
- 5 G je acyklický a $|E(G)| = |V(G)| - 1$.
- 6 G je acyklický, ale pokud přidáme jakoukoli hranu do $E(G)$, výsledný graf bude obsahovat aspoň jeden cyklus.

Kořenové stromy

kořenový strom

rooted tree

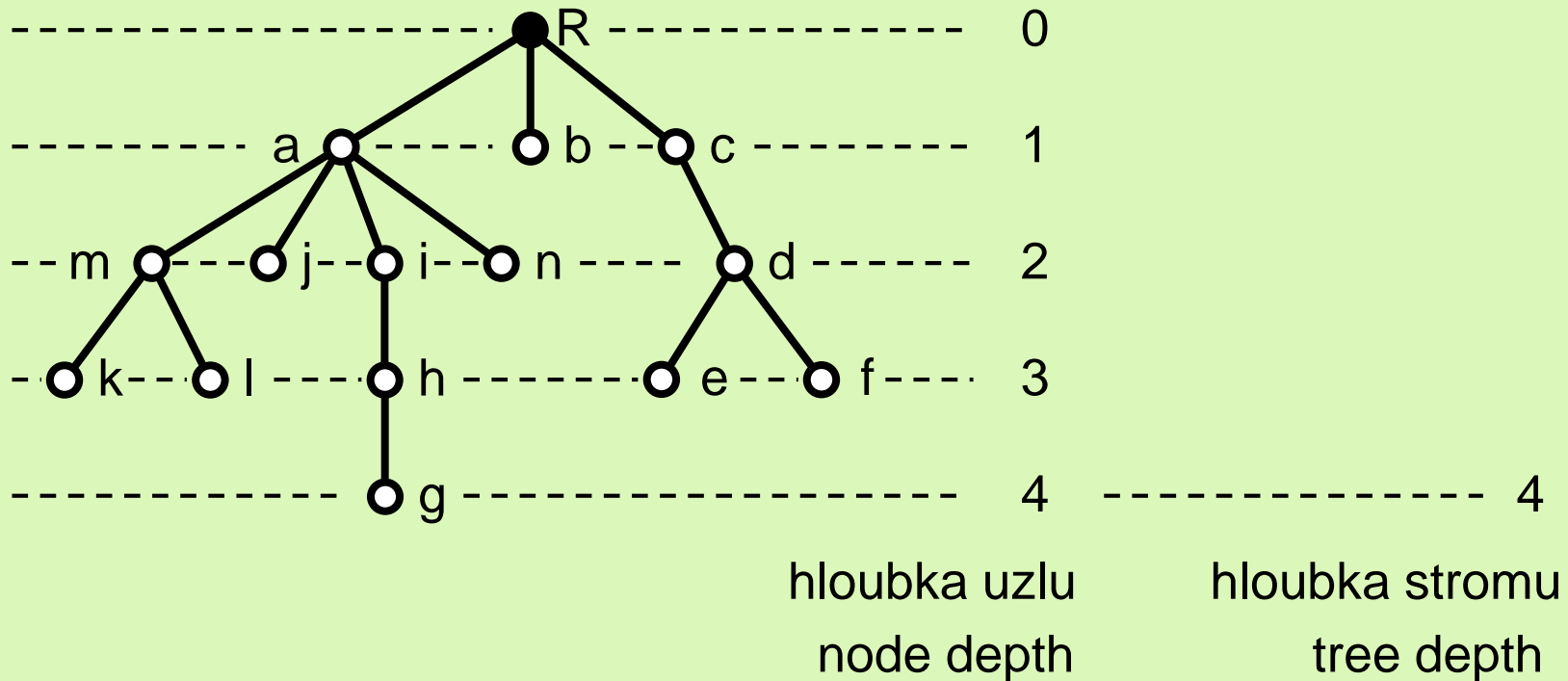


Definice kořenového stromu

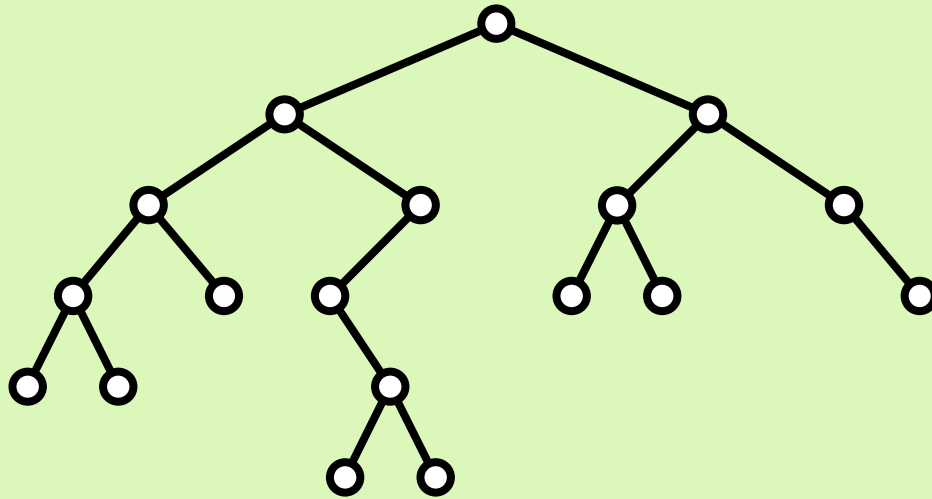
Definice

- **Kořenový strom** je volný strom, ve kterém jeden z uzlů je odlišen od ostatních jako **kořen**, r .
- Uzly u ve vzdálenosti d od kořenu r tvoří hladinu uzlů v **hloubce** $h(u) = d$. Kořen má hloubku $h(r) = 0$.
- Necht' uzel u leží na (jedinečné) cestě z kořene r do uzlu v . Pak u je **předek** v a v je **potomek** u .
- Nejbližší předek uzlu je jeho **rodič** a nejbližší potomek je jeho **syn**.
⇒ Každý uzel kromě kořenu má jedinečného rodiče.
- Uzly se stejným rodičem jsou **sourozenci**.
- Uzel, který nemá potomky, se nazývá **list**. Ostatní uzly jsou **vnitřní**.
- **Stupeň** vnitřního uzlu je počet jeho synů (rodič se nepočítá)!!!!!!
- **k -ární strom**: každý vnitřní uzel má stupeň **nejvýše** k .

Hloubka kořenového stromu

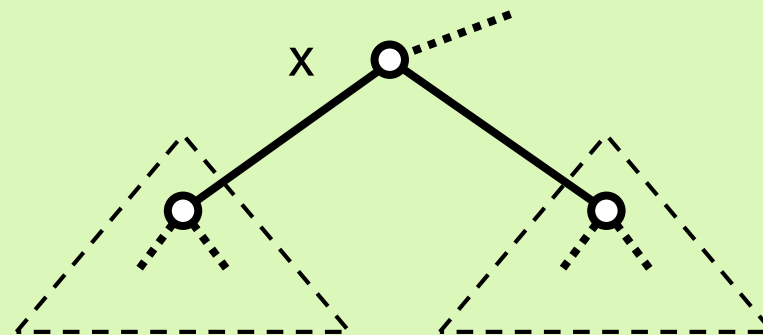


Binární kořenové stromy



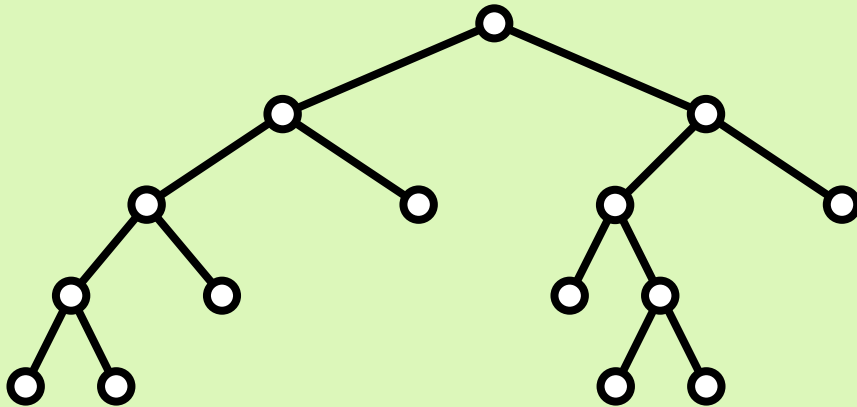
binární (kořenový!!) strom
0, 1 nebo 2 následníci

binary (rooted!!) tree
0, 1 or 2 successors



podstrom uzlu x levý pravý
subtree of node x left right

Pravidelné a vyvážené stromy

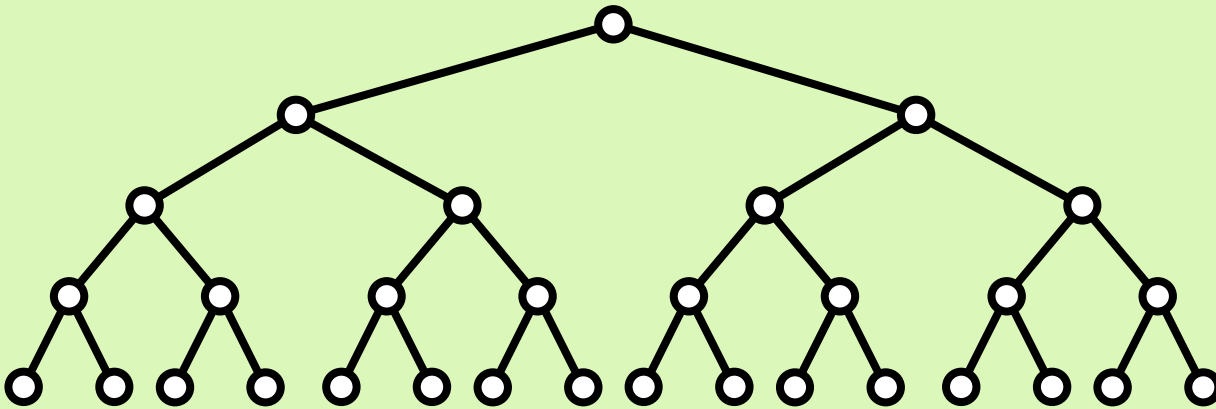


pravidelný binární strom

0 nebo 2 následníci

regular binary tree

0 or 2 successors



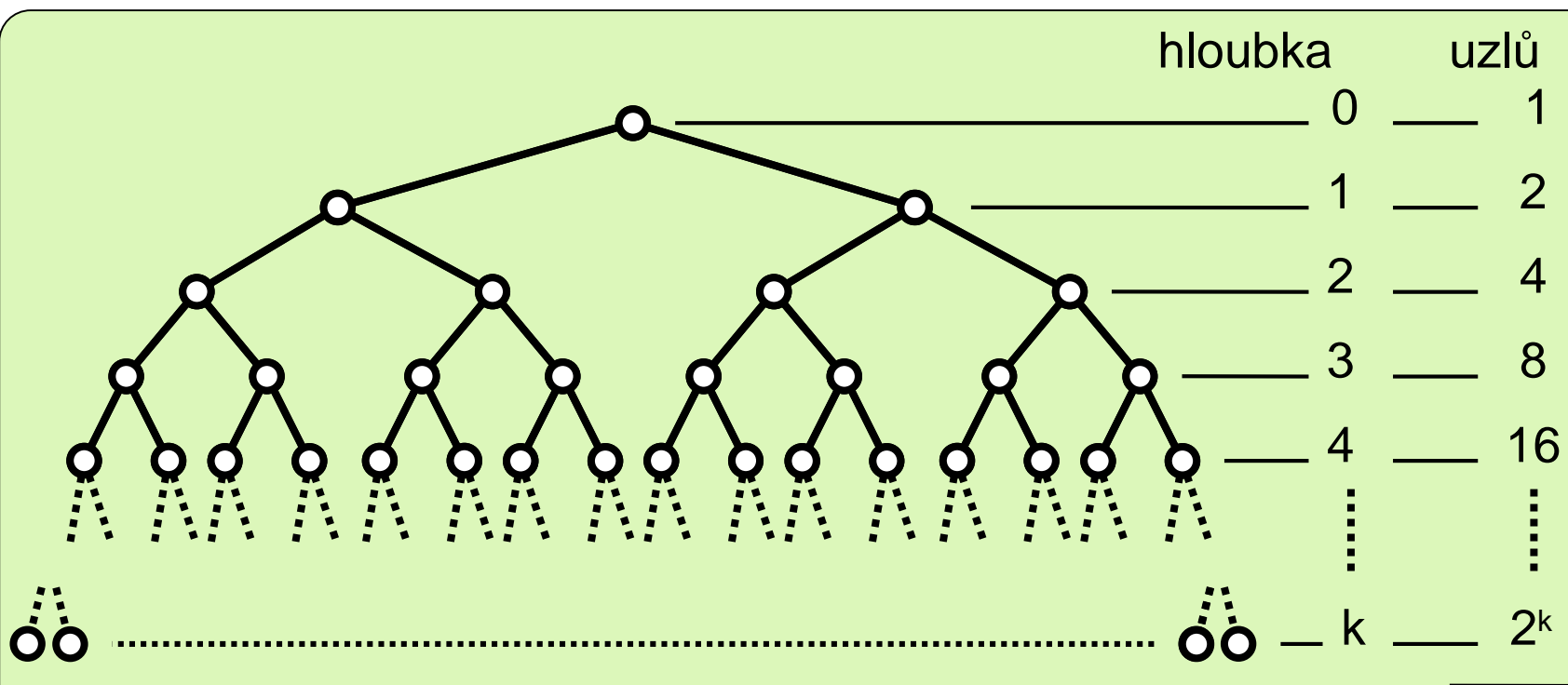
vyvážený strom

balanced tree

Hloubky všech listů jsou (víceméně) stejné.

All leaf depths are (more or less) equal.

Hloubka vyváženého stromu

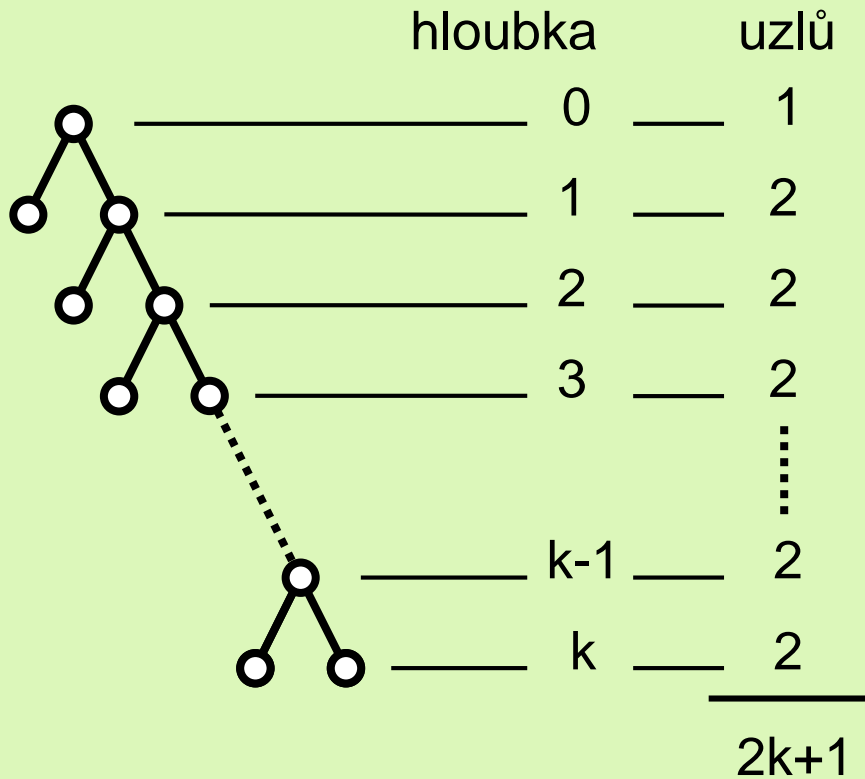


$$(2^{(\text{hloubka stromu})+1} - 1) \sim \text{uzlů}$$

$$\text{hloubka stromu} \sim \log_2(\text{uzlů}+1) - 1$$

vyvážený strom: hloubka $\sim \log_2(\text{uzlů})$

Hloubka nevyváženého stromu

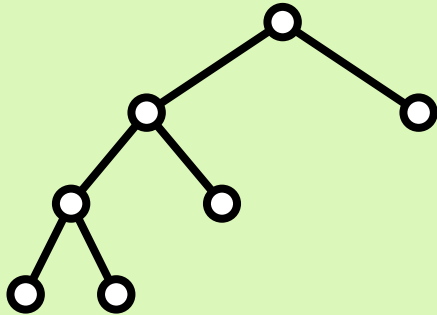


$$2(\text{hloubka})+1 \sim \text{uzlů}$$

$$\text{hloubka} \sim (\text{uzlů}-1)/2$$

extrémně nevyvážený pravidelný strom: $\text{hloubka} \sim (\text{uzlů}-1)/2$

Souhrn velikosti



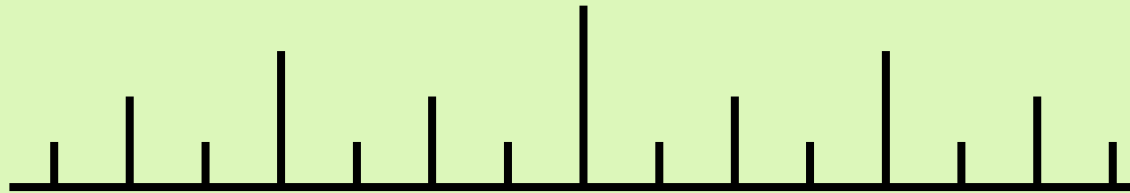
hloubka pravidelného stromu = $O(\text{počet uzlů})$

obvykle

hloubka pravidelného stromu = $\Theta(\log_2(\text{počet uzlů}))$

Jednoduchý příklad rekurze

pravítko



1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

rysky
pravítka

délky
rysek

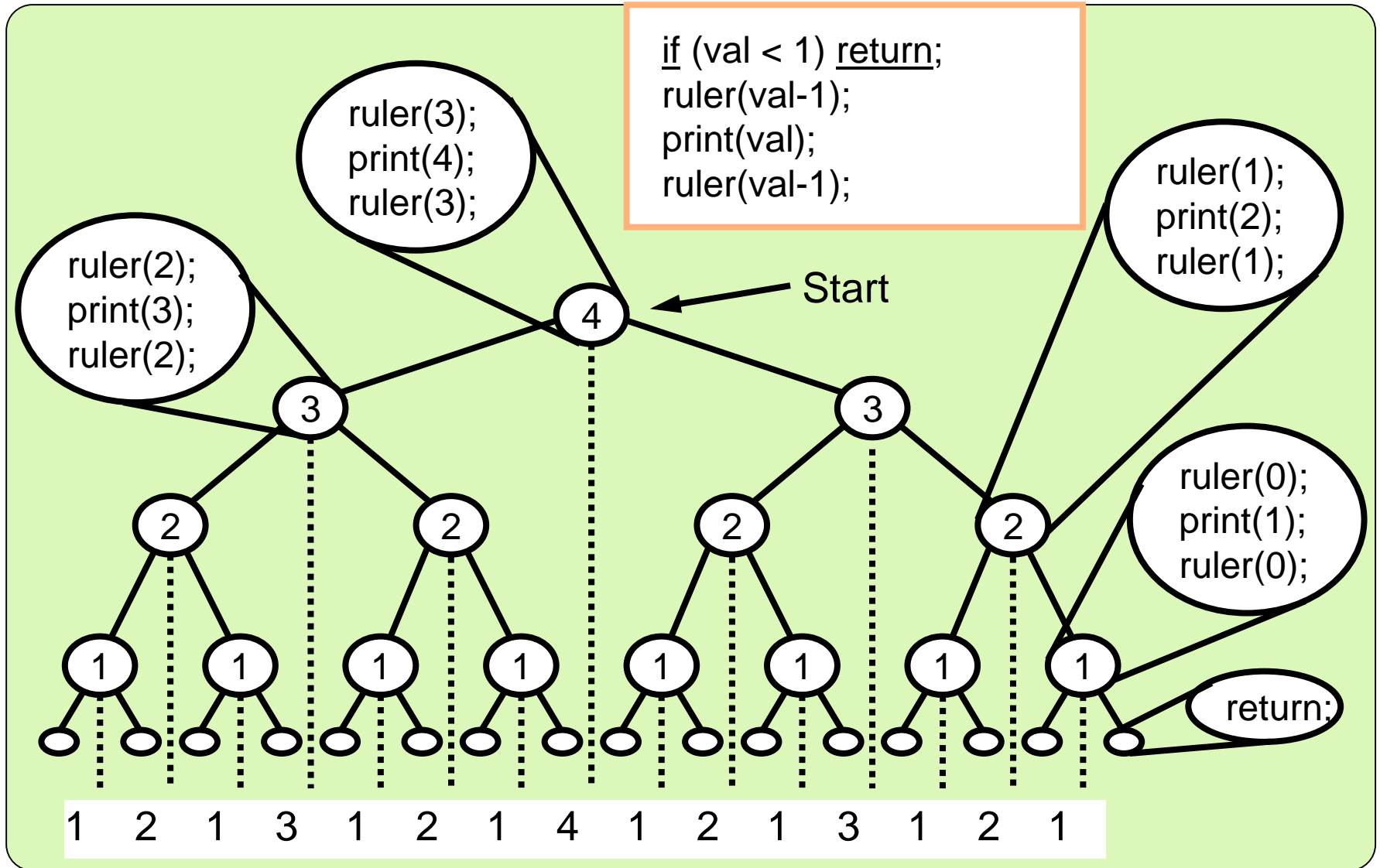
```
void ruler(int val) {
    if (val < 1) return;

    ruler(val-1);
    print(val);
    ruler(val-1);
}
```

Call: ruler(4);

kód
vypisující
délky
rysek
pravítka

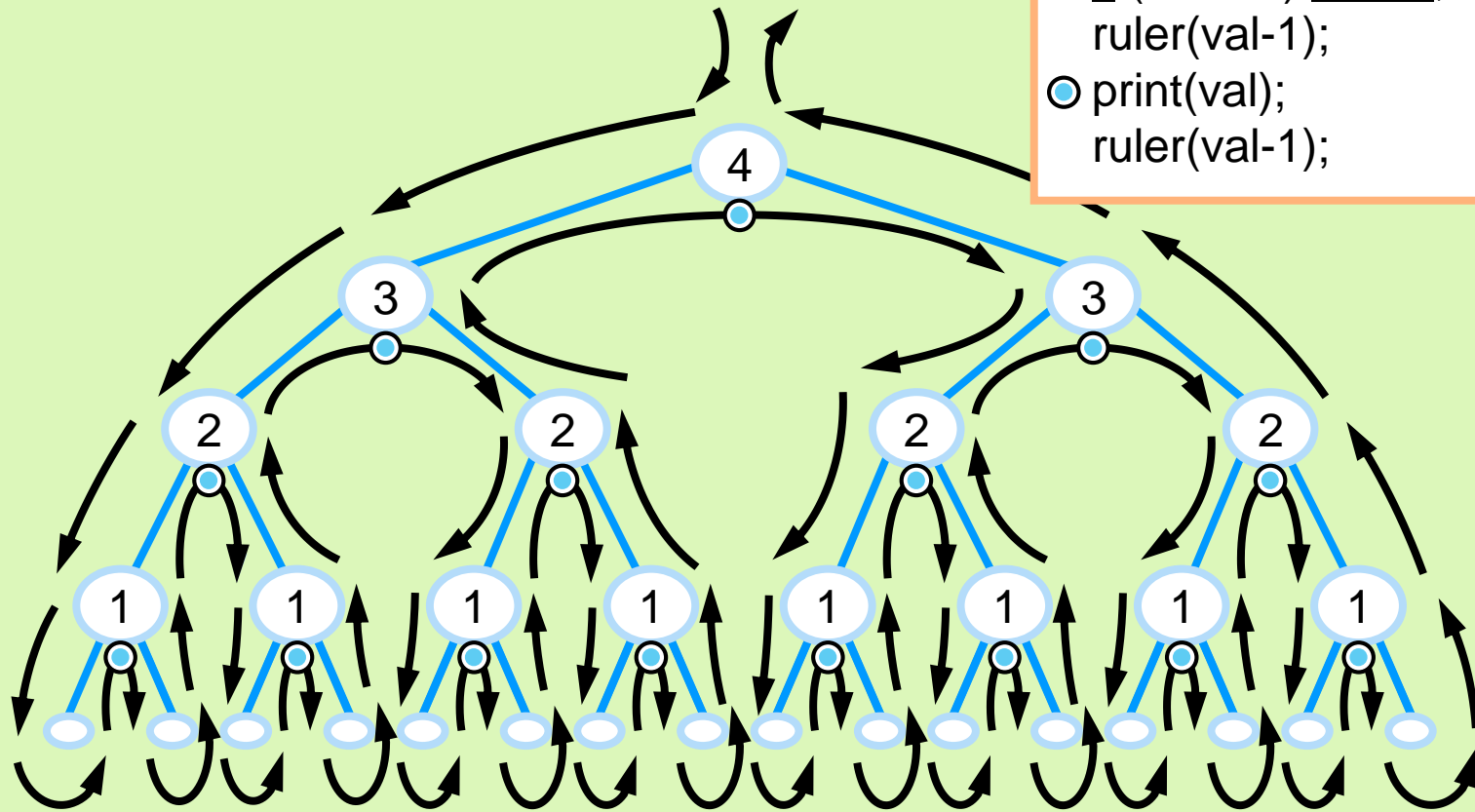
Jednoduchý příklad rekurze (pokr.)



Jednoduchý příklad rekurze (pokr.)

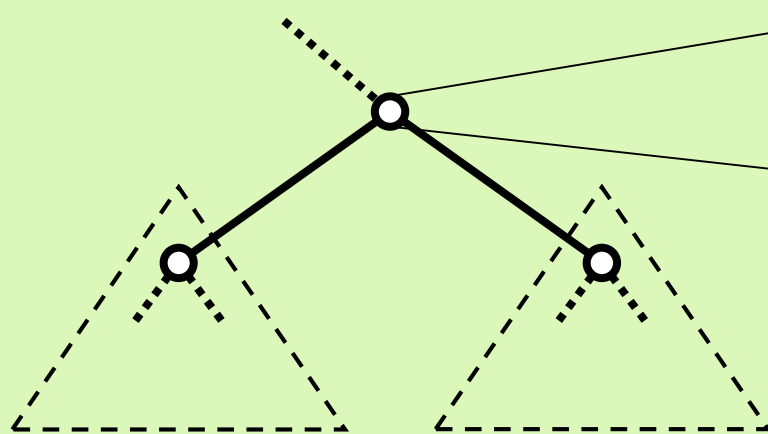
```

if (val < 1) return;
ruler(val-1);
○ print(val);
ruler(val-1);
    
```

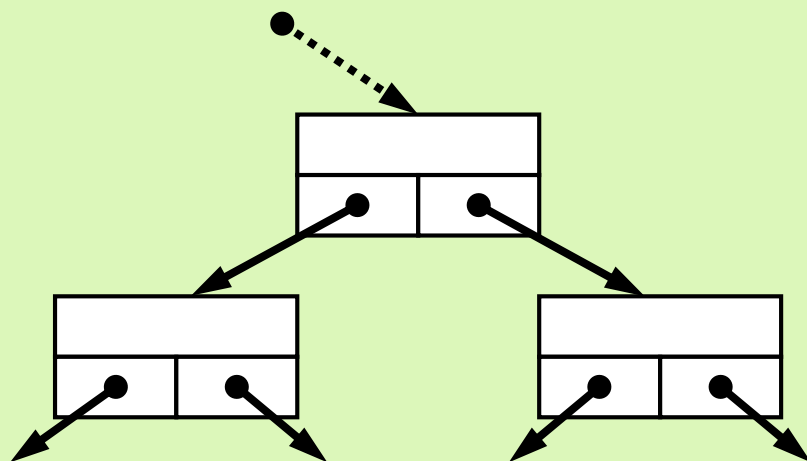


1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

Implementace binárního stromu – C

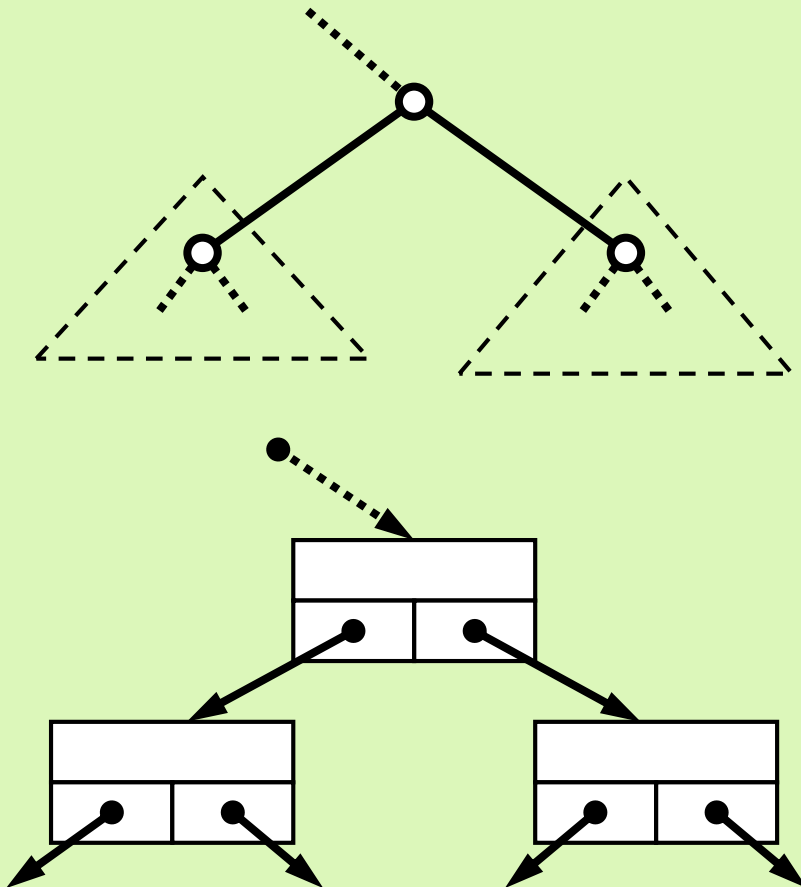


key	
left	right



```
typedef struct node {
    int key;
    struct node *left;
    struct node *right;
} NODE;
```

Implementace binárního stromu – Java



```

public class Node {
    public Node left;
    public Node right;
    public int key;
    public Node(int k) {
        key = k;
        left = null;
        right = null;
    }
}

public class Tree {
    public Node root;
    public Tree() {
        root = null;
    }
}

```

Vybudování náhodného binárního stromu v C

```
NODE *randTree(int depth) {
    NODE *pnode;
    if ((depth <= 0) || (random(10) > 7))
        return (NULL); //stop recursion
    pnode = (NODE *) malloc(sizeof(NODE)); // create node
    if (pnode == NULL) {
        printf("%s", "No memory.");
        return NULL;
    }
    pnode->left = randTree(depth-1); // make left subtree
    pnode->key = random(100); // some value
    pnode->right = randTree(depth-1); // make right subtree
    return pnode; // all done
}
```

```
NODE *root;
root = randTree(4);
```

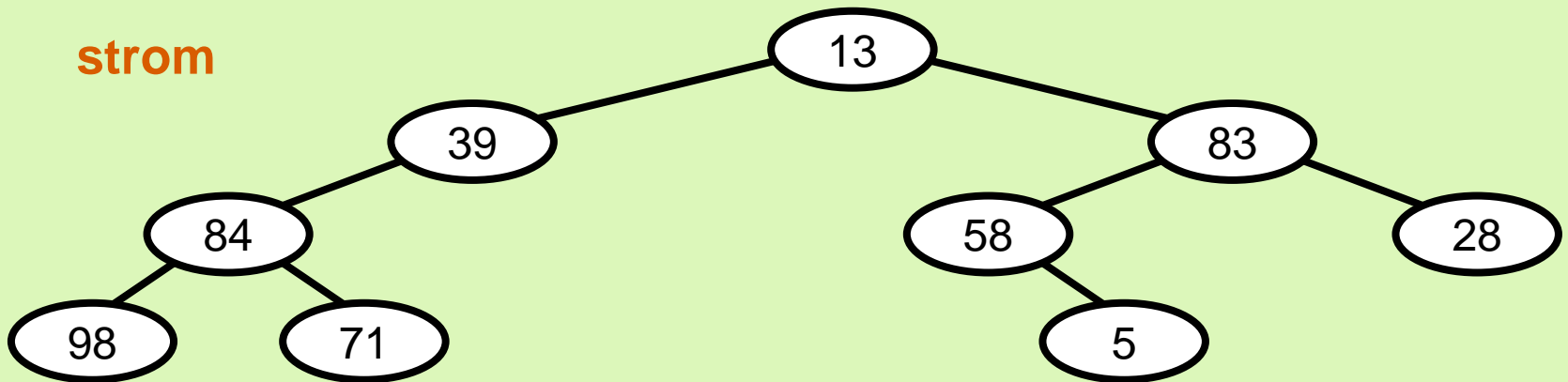
Vybudování náhodného binárního stromu v Javě

```
public Node randTree(int depth) {  
    Node node;  
    if ((depth <= 0) || ((int) Math.random()*10 > 7)  
        return null;  
                                        // create node with a key value  
    node = new Node((int) (Math.random()*100));  
  
    node.left = randTree(depth-1); // make left subtree  
    node.right = randTree(depth-1); // make right subtree  
    return node; // all done  
}
```

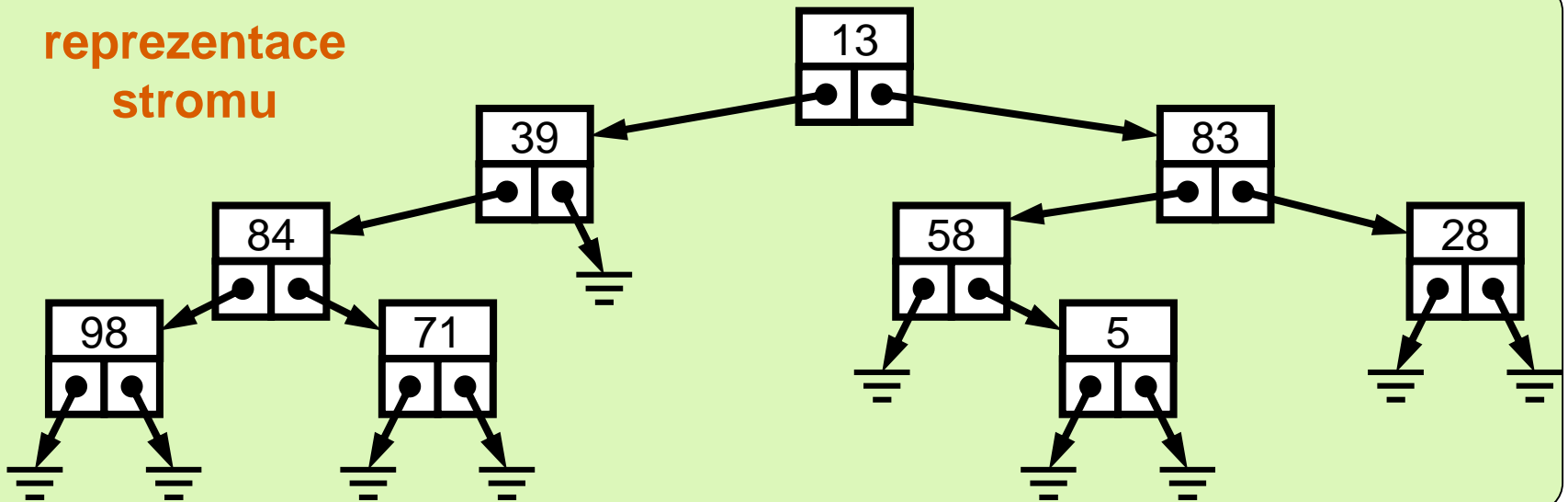
```
Node root;  
root = randTree(4);
```

Náhodný binární strom

strom

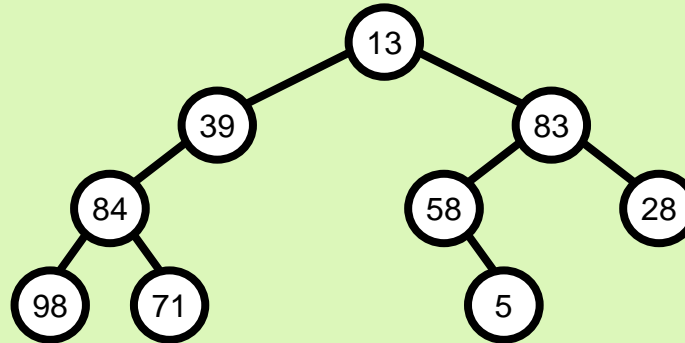


reprezentace
stromu



Průchod stromem v pořadí Inorder

Strom



Průchod
stromem
v pořadí
INORDER

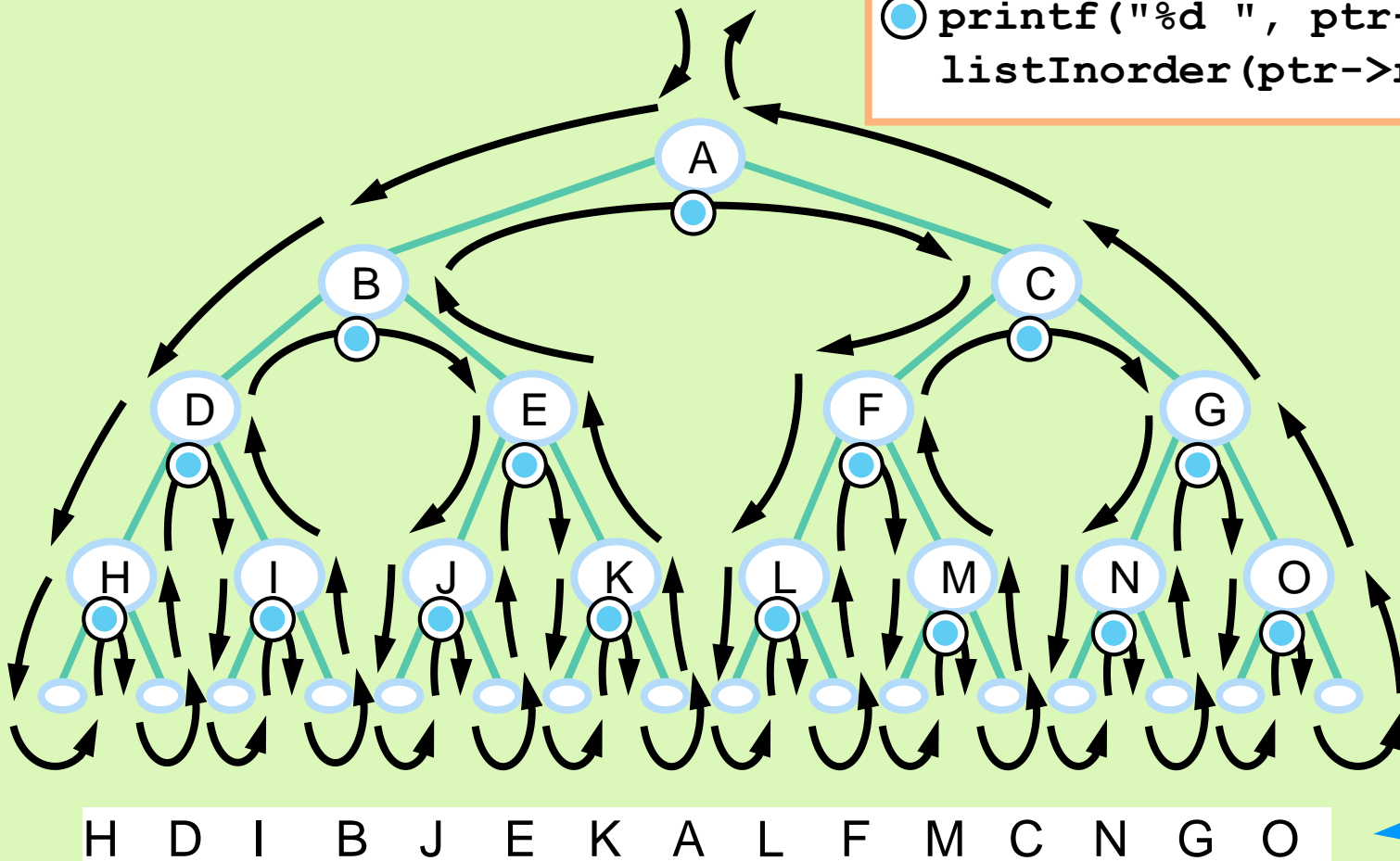
```
void listInorder( NODE *ptr) {  
    if (ptr == NULL) return;  
    listInorder(ptr->left);  
    printf("%d ", ptr->key);  
    listInorder(ptr->right);  
}
```

Výstup

98 84 71 39 13 58 5 83 28

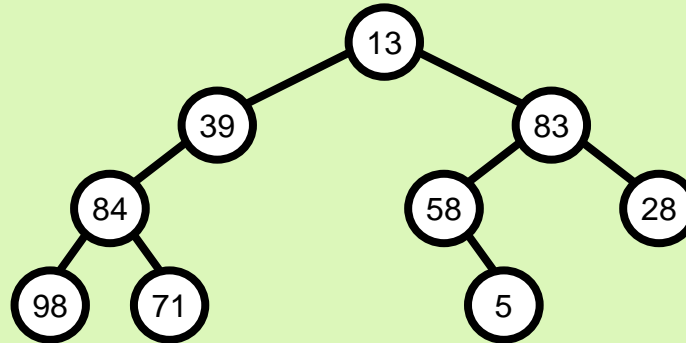
Rekurzivní pohyb v průchodu Inorder

```
listInorder(ptr->left);
printf("%d ", ptr->key);
listInorder(ptr->right);
```



Průchod stromem v pořadí Preorder

Strom



Průchod
stromem
v pořadí
PREORDER

```
void listPreorder( NODE *ptr) {  
    if (ptr == NULL) return;  
    printf("%d ", ptr->key);  
    listPreorder(ptr->left);  
    listPreorder(ptr->right);  
}
```

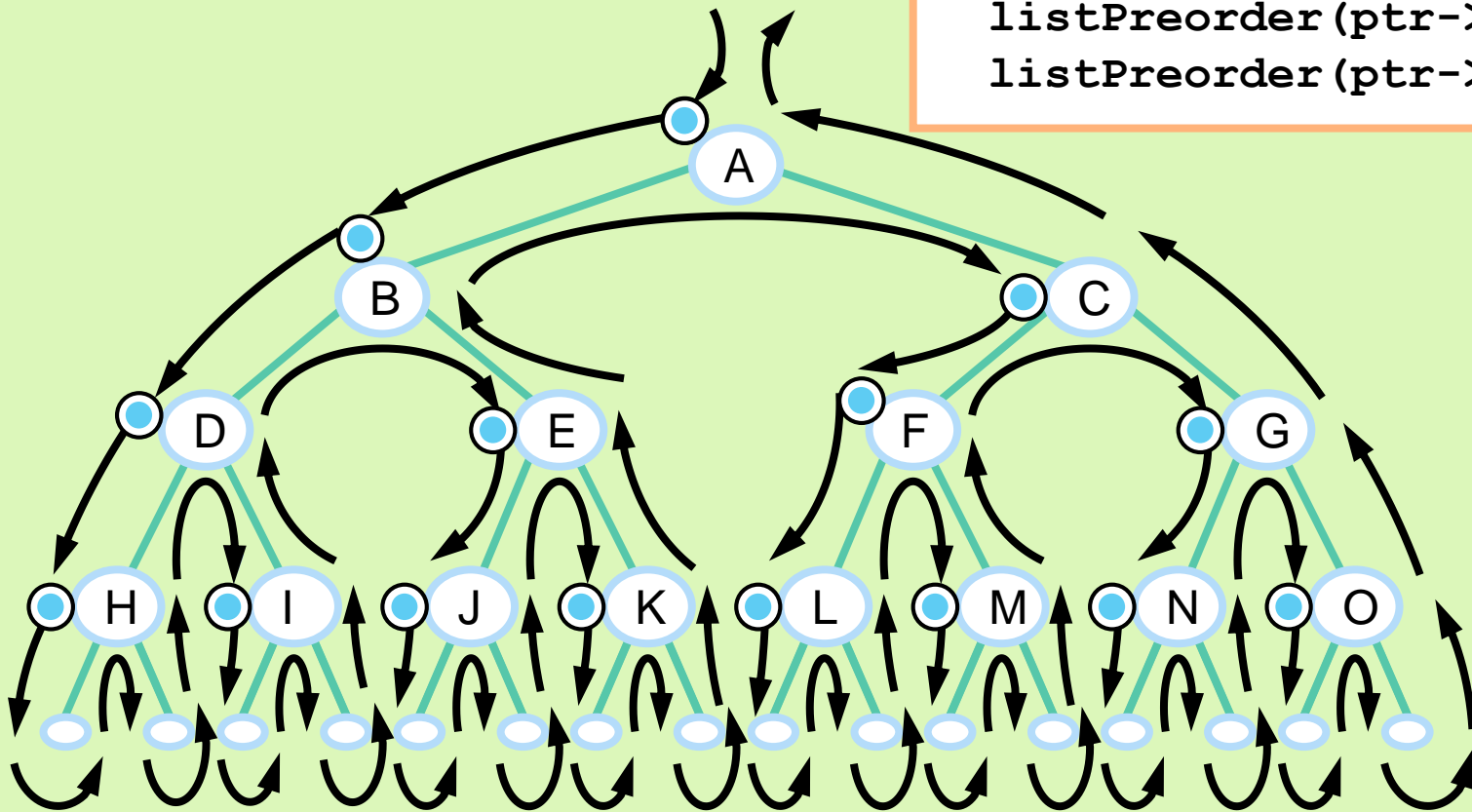
Výstup

13 39 84 98 71 83 58 5 28

Rekurzivní pohyb v průchodu Preorder

```

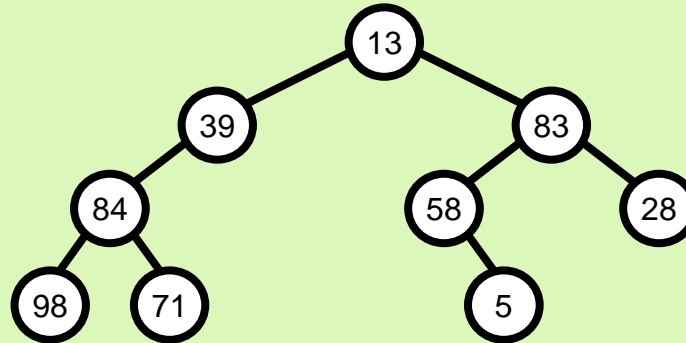
    ● printf("%d ", ptr->key);
      listPreorder(ptr->left);
      listPreorder(ptr->right);
  
```



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

Průchod stromem v pořadí Postorder

Strom



Průchod
stromem
v pořadí
POSTORDER

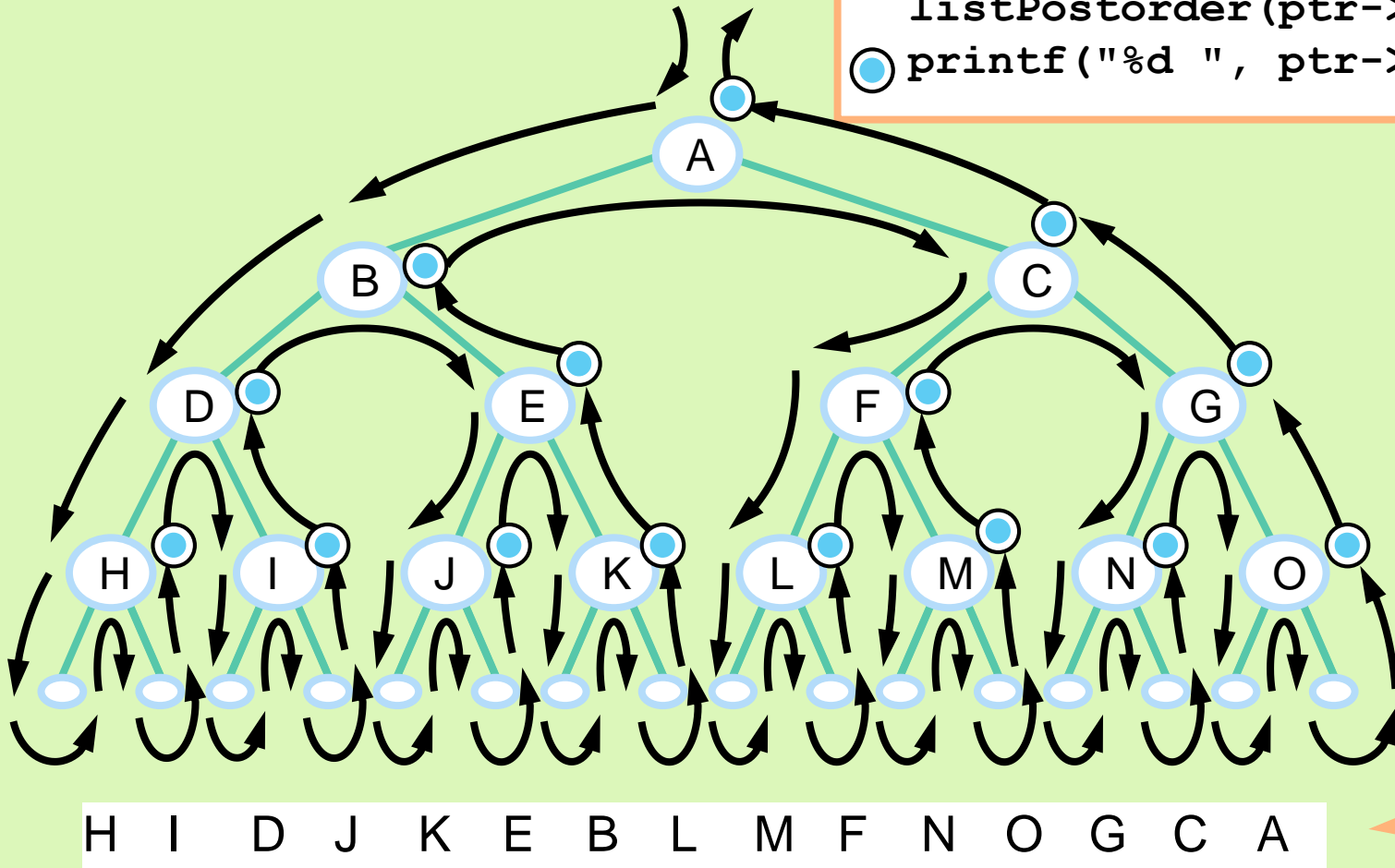
```
void listPostorder( NODE *ptr) {  
    if (ptr == NULL) return;  
    listPostorder(ptr->left);  
    listPostorder(ptr->right);  
    printf("%d ", ptr->key);  
}
```

Výstup

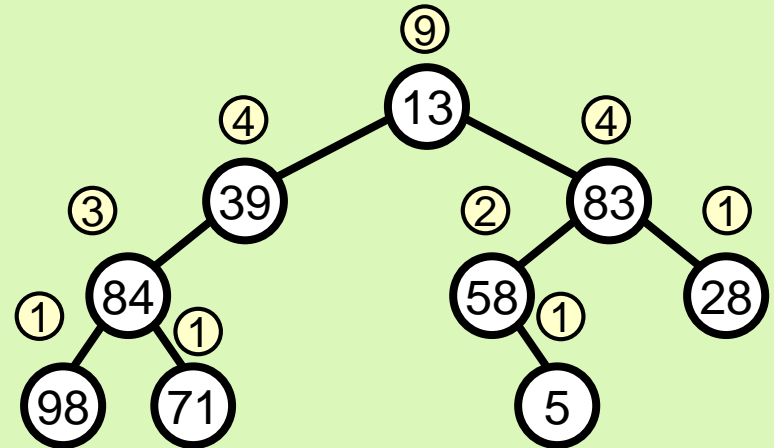
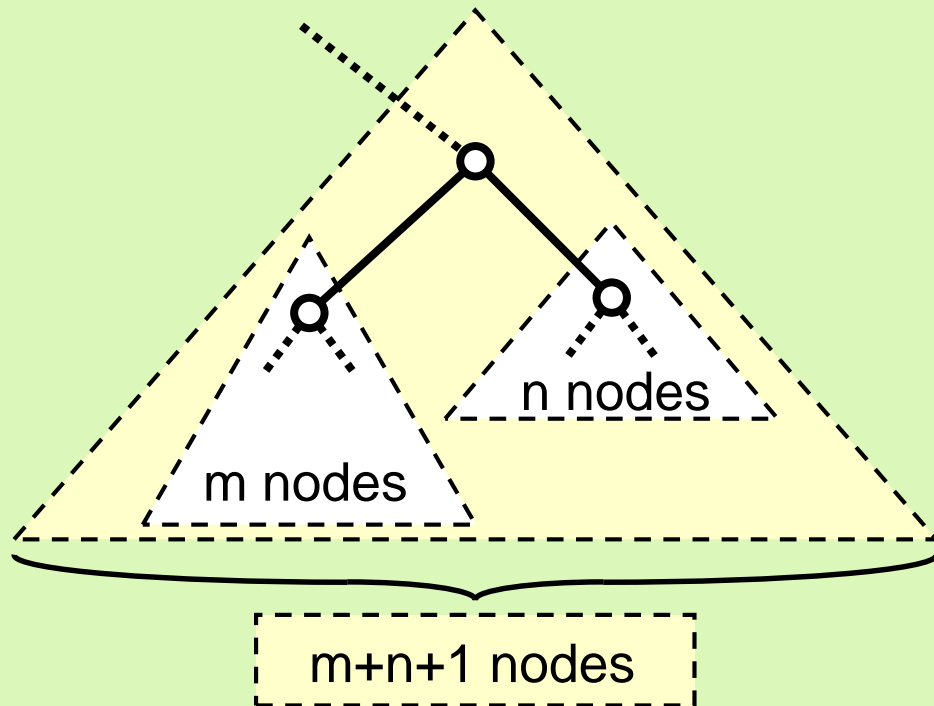
98 71 84 39 5 58 28 83 13

Rekurzivní pohyb v průchodu Postorder

```
listPostorder(ptr->left);
listPostorder(ptr->right);
printf("%d ", ptr->key);
```



Velikost stromu rekurzivně

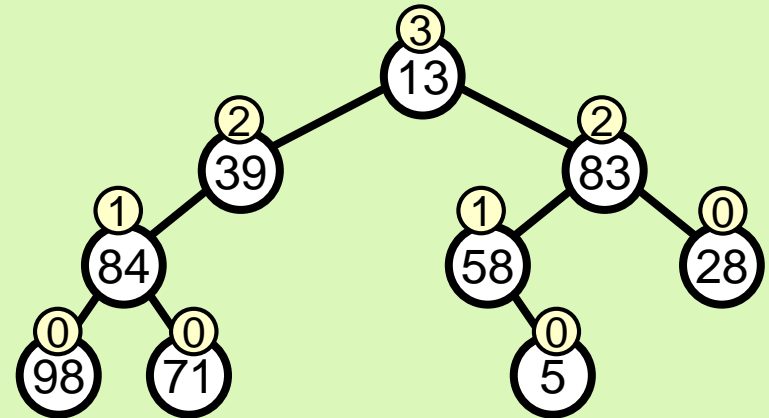
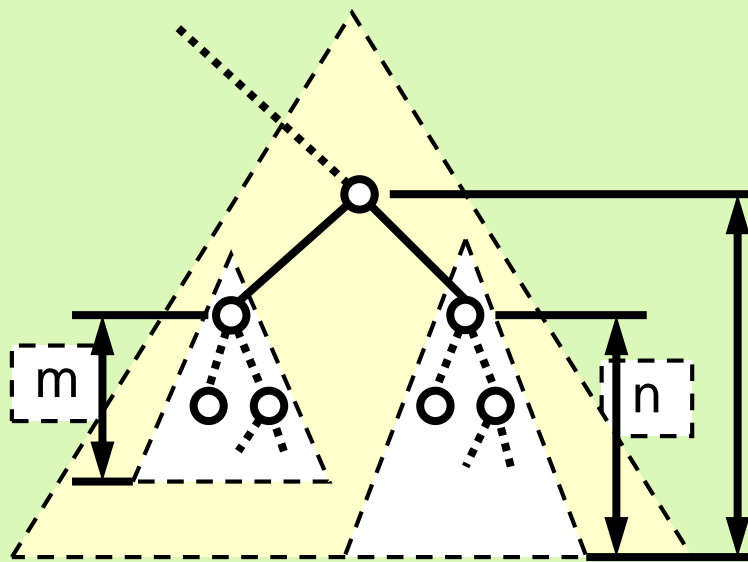


```

int count(NODE *ptr) {
    if (ptr == NULL) return (0);
    return (count(ptr->left) + count(ptr->right)+1);
}

```

Hloubka stromu rekurzivně



$$n+1 = \max(m, n) + 1$$

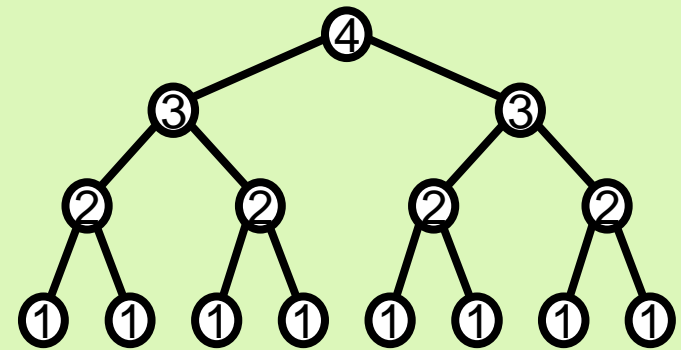
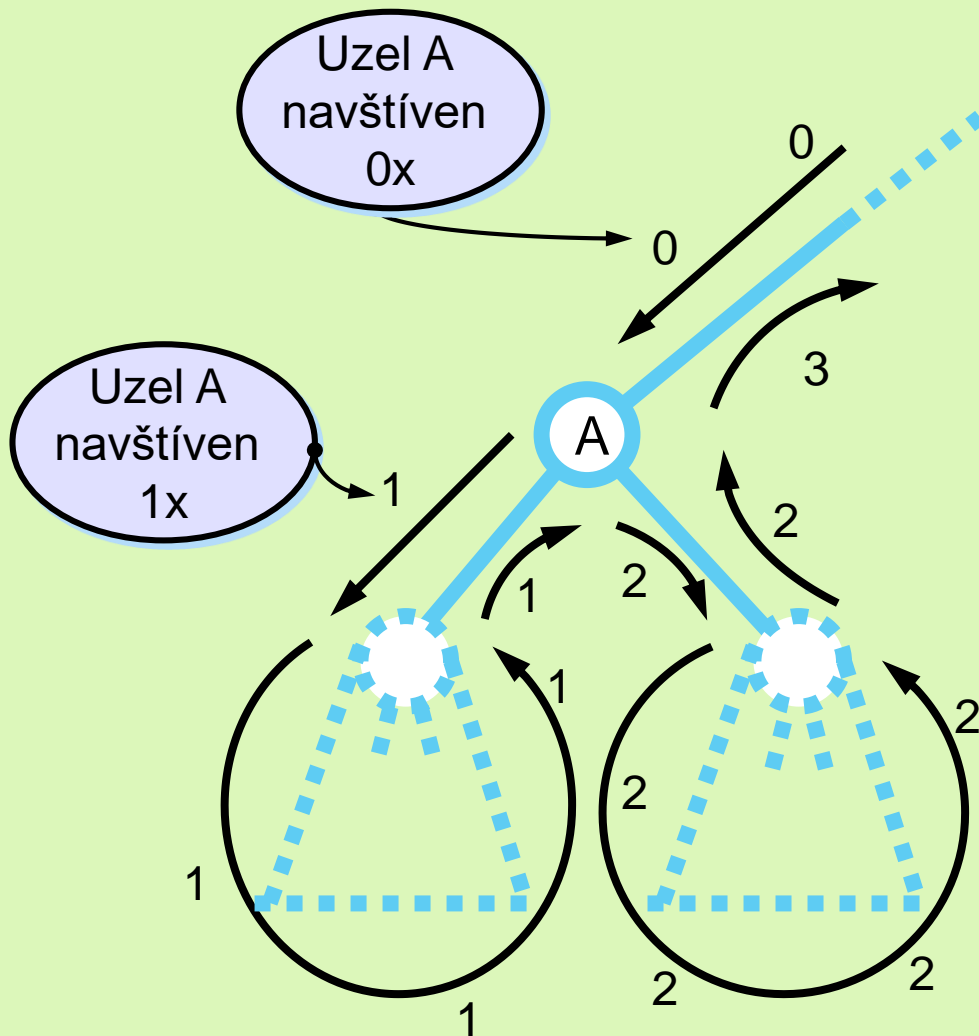
```

int depth(NODE *ptr) {
    if (ptr == NULL) return (-1);
    return ( max(depth(ptr->left), depth(ptr->right) ) + 1 );
}

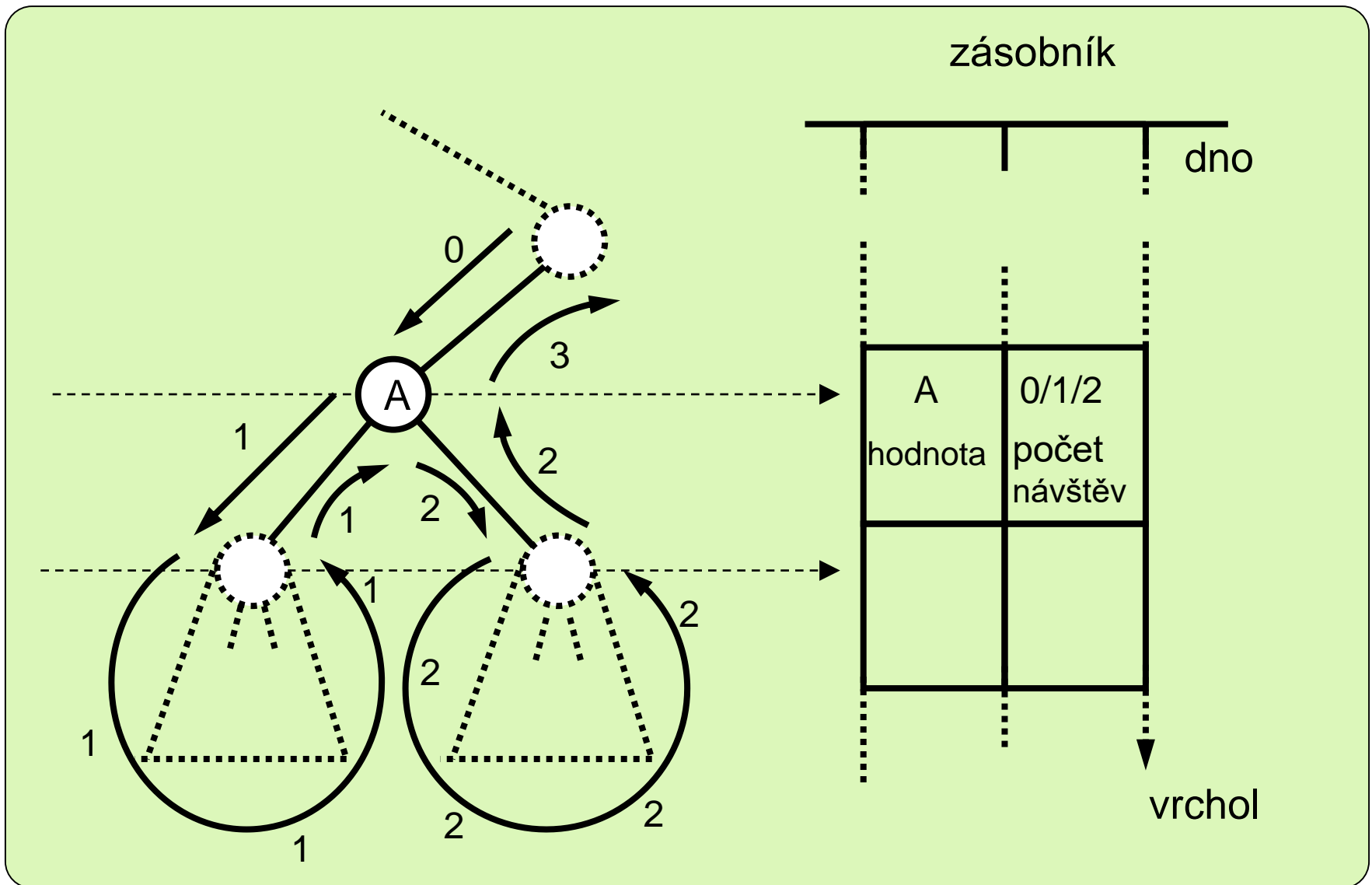
```

Zásobník implementuje rekurzi

Pravítko bez rekurze



Zásobník implementuje rekurzi



Zásobník implementuje rekurzi

Standardní strategie

Při používání zásobníku:

Je-li to možné, zpracovávej jen data ze zásobníku.

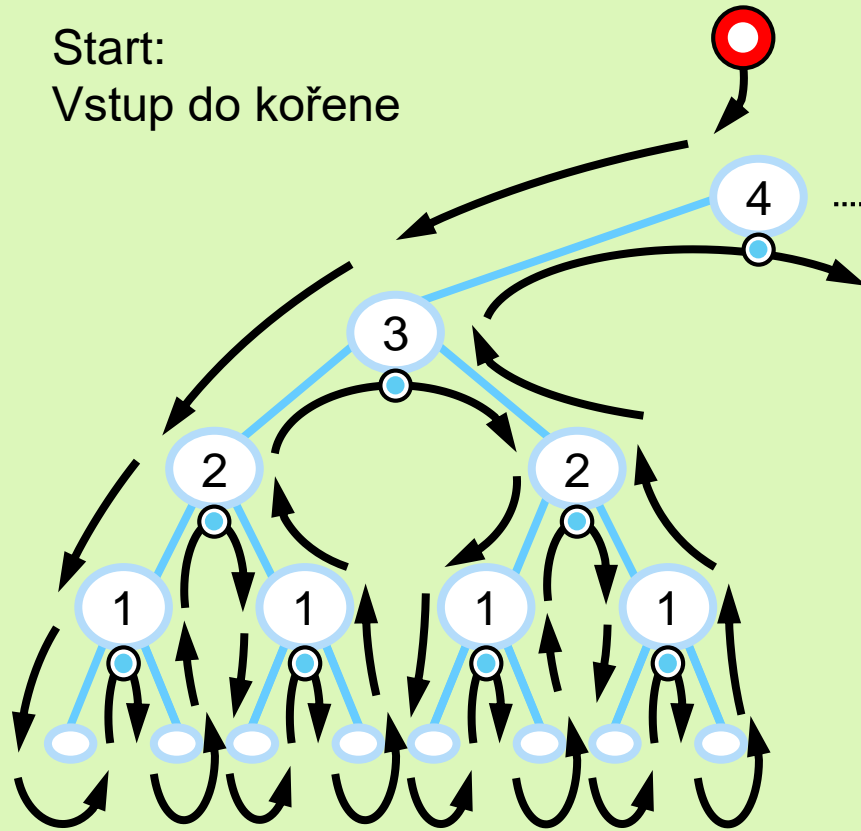
Standardní postup

Ulož první uzel (první zpracovávaný prvek) do zásobníku.
Každý další uzel (zpracovávaný prvek) ulož také na zásobník.
Zpracovávej vždy pouze uzel na vrcholu zásobníku.
Když jsi s uzlem (prvkem) hotov, ze zásobníku ho odstraň.
Skonči, když je zásobník prázdný.

Zásobník implementuje rekurzi

Každý záběr v následující sekvenci předvádí situaci PŘED zpracováním uzlu.

Start:
Vstup do kořene

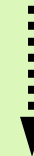


zásobník

hodnota návštěv

push(4,0)

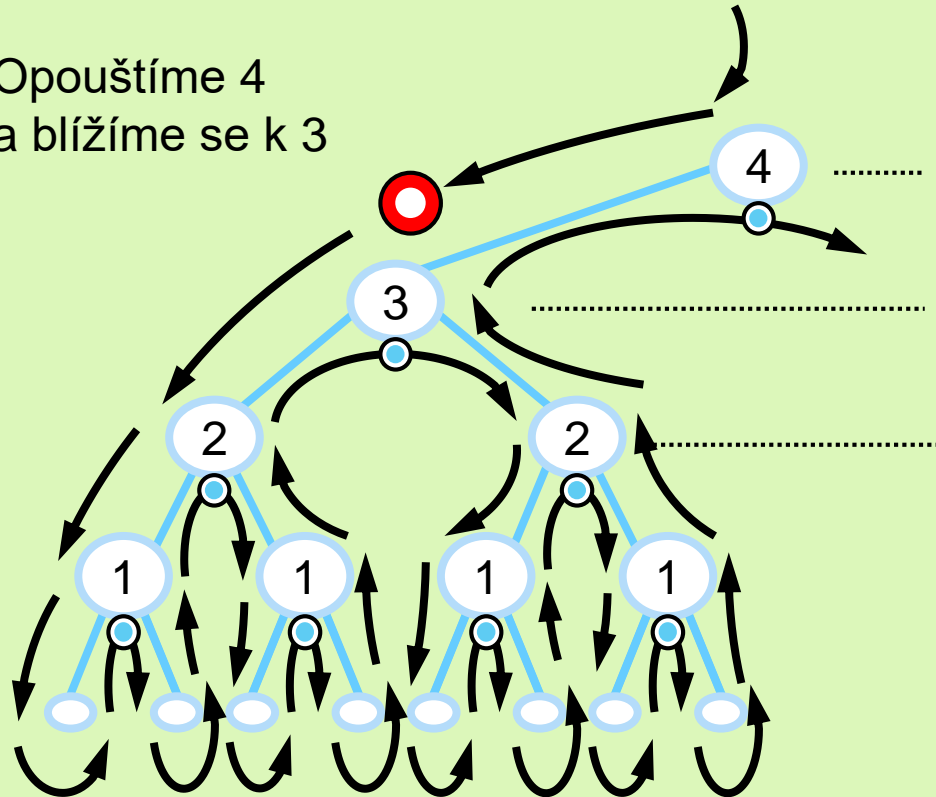
4	0
---	---



Výstup

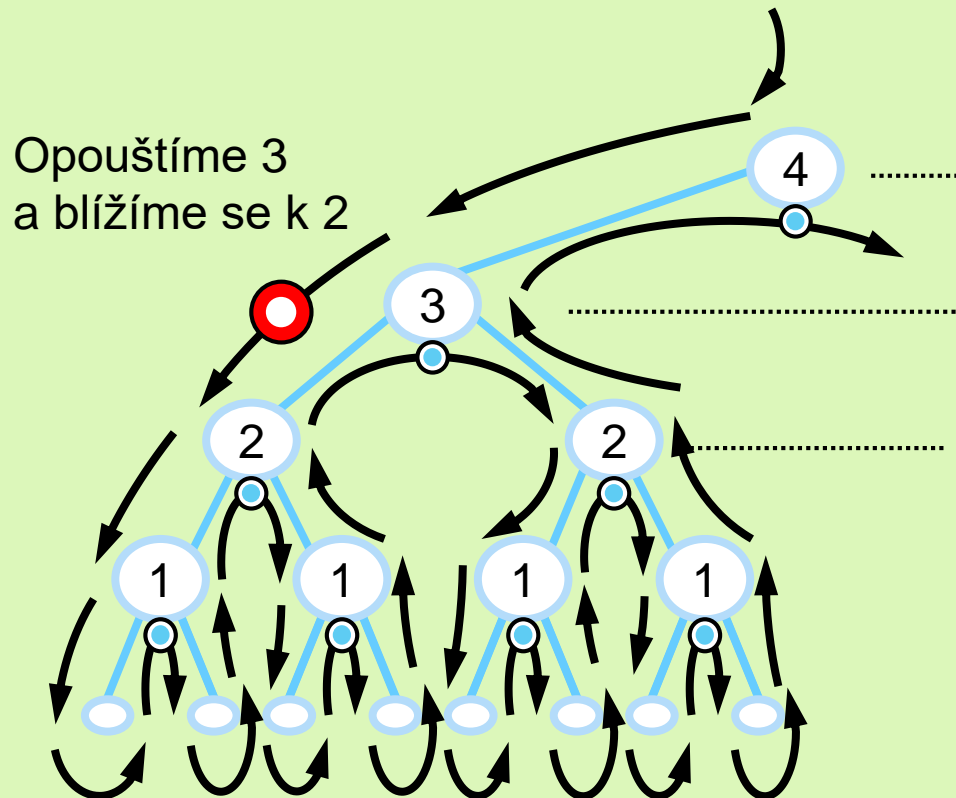
Zásobník implementuje rekurzi

Opouštíme 4
a blížíme se k 3



Výstup

Zásobník implementuje rekurzi



zásobník

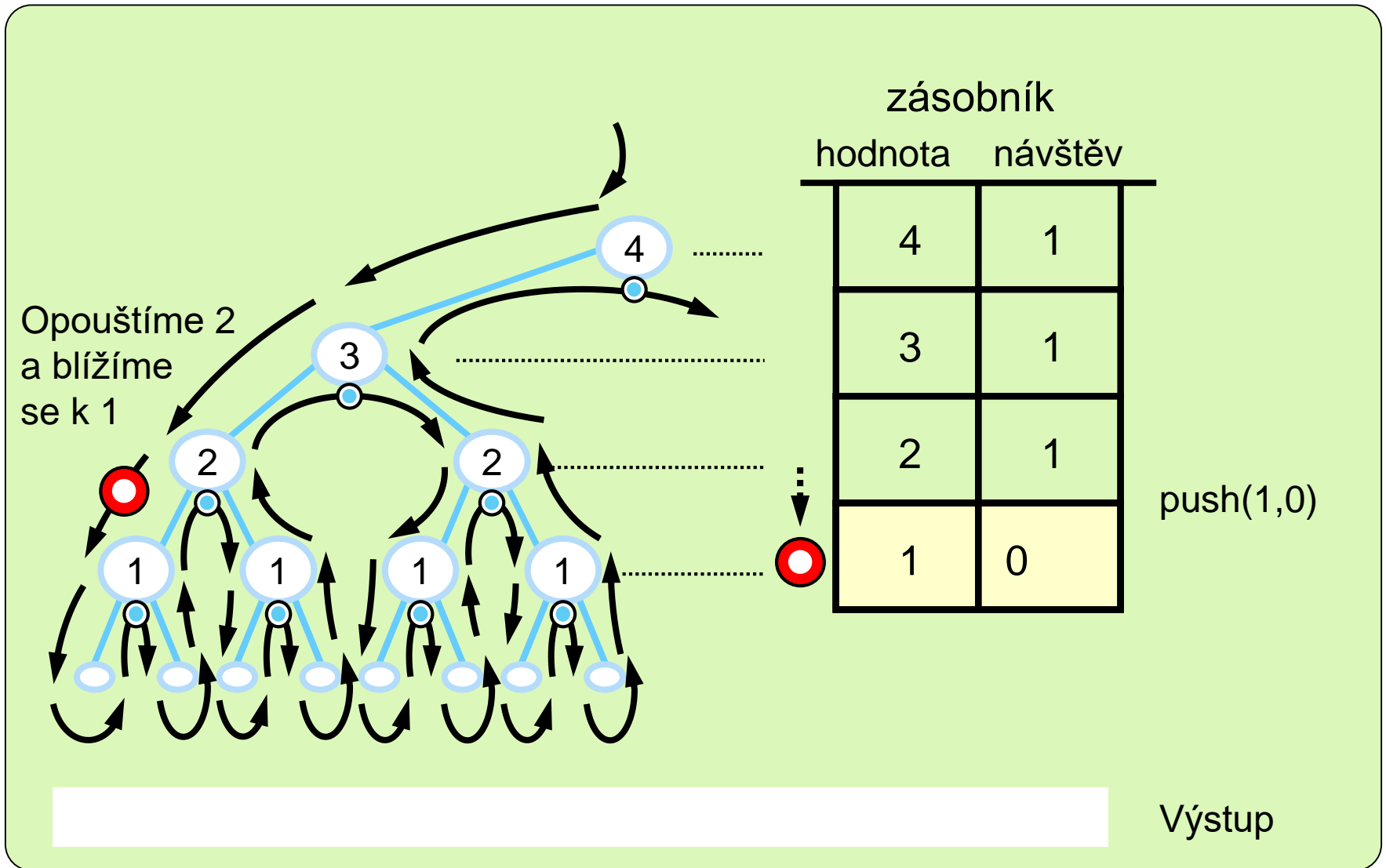
hodnota	návštěv
4	1
3	1
2	0

hodnota	návštěv
4	1
3	1
2	0

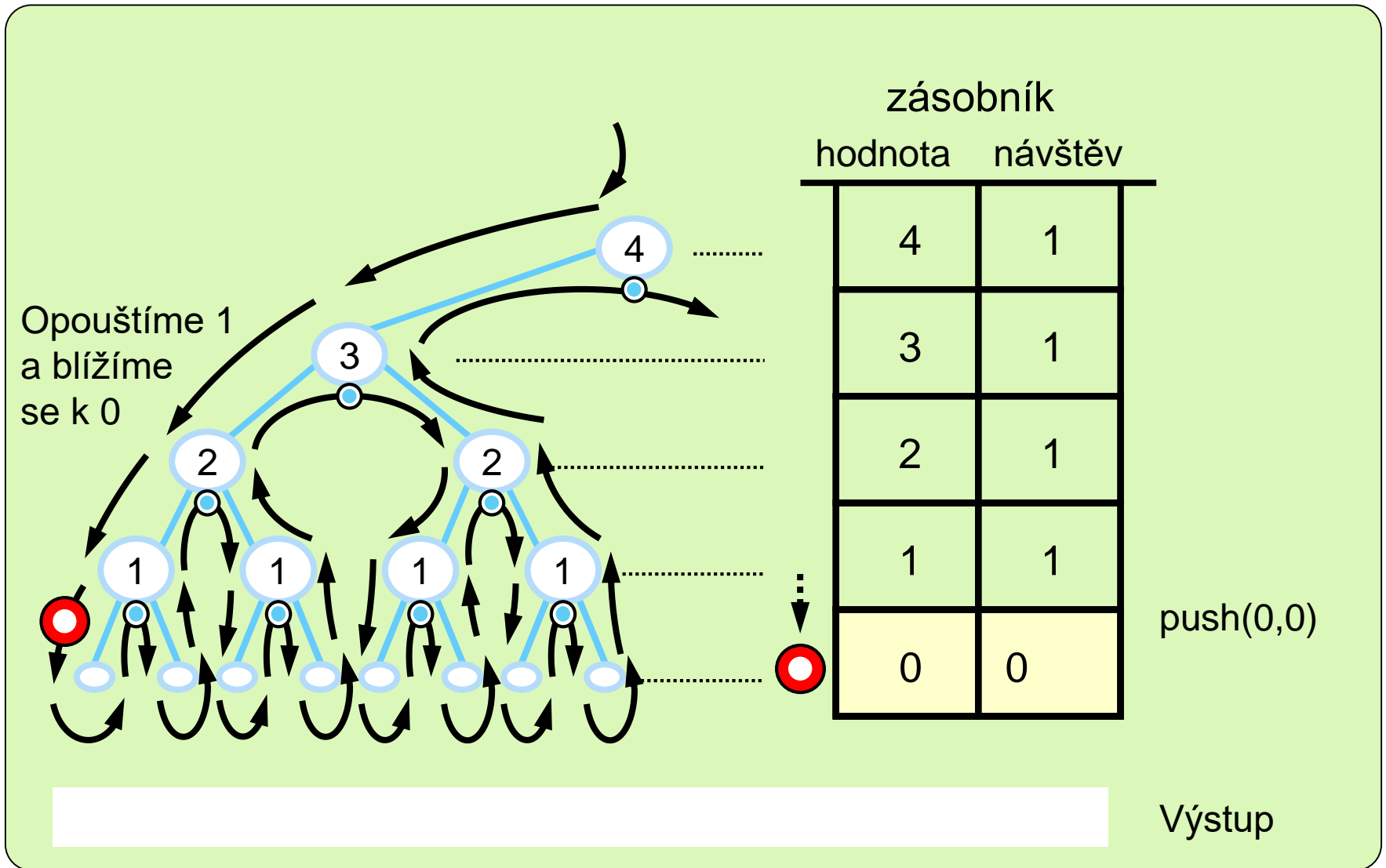
push(2,0)

Výstup

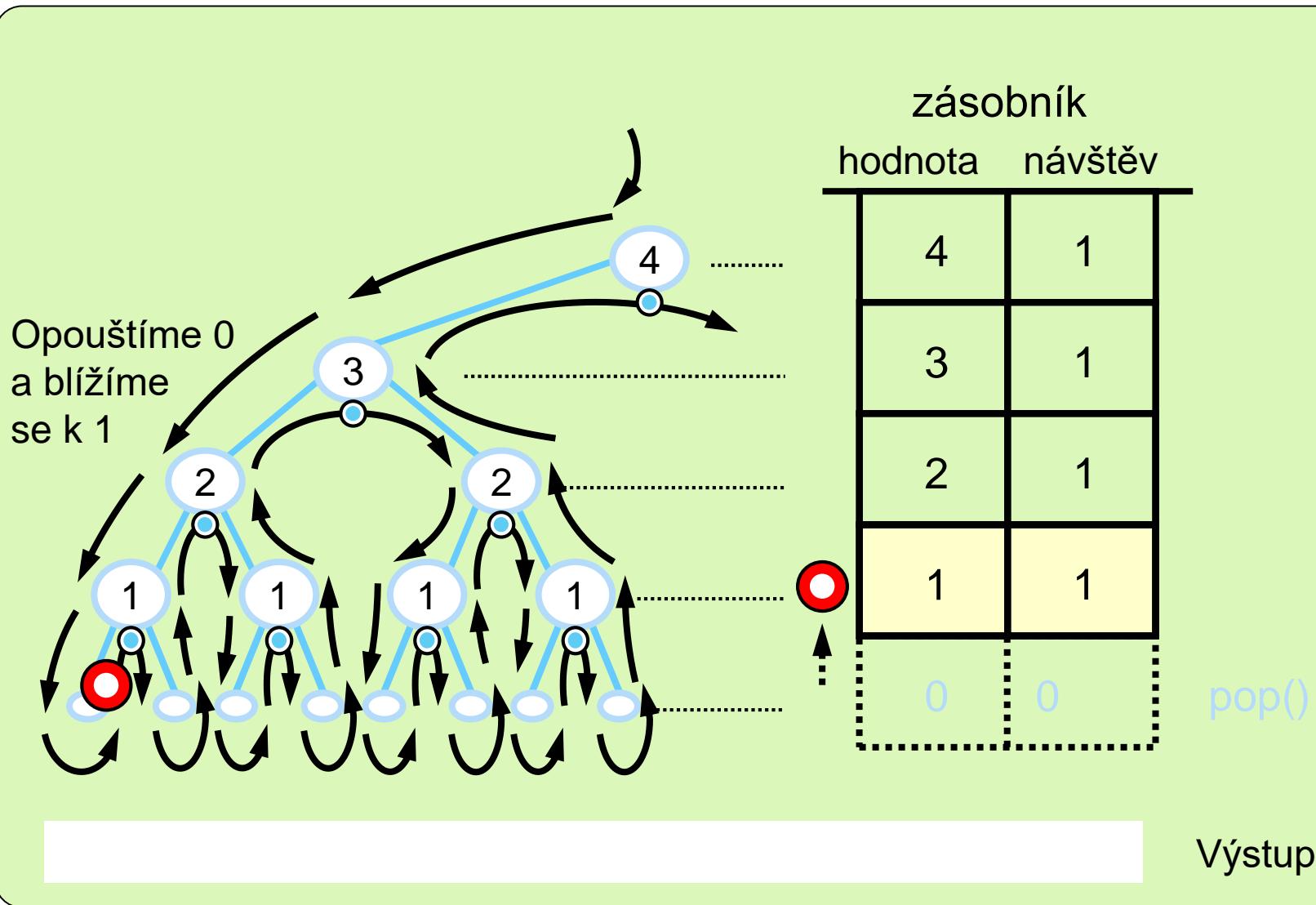
Zásobník implementuje rekurzi



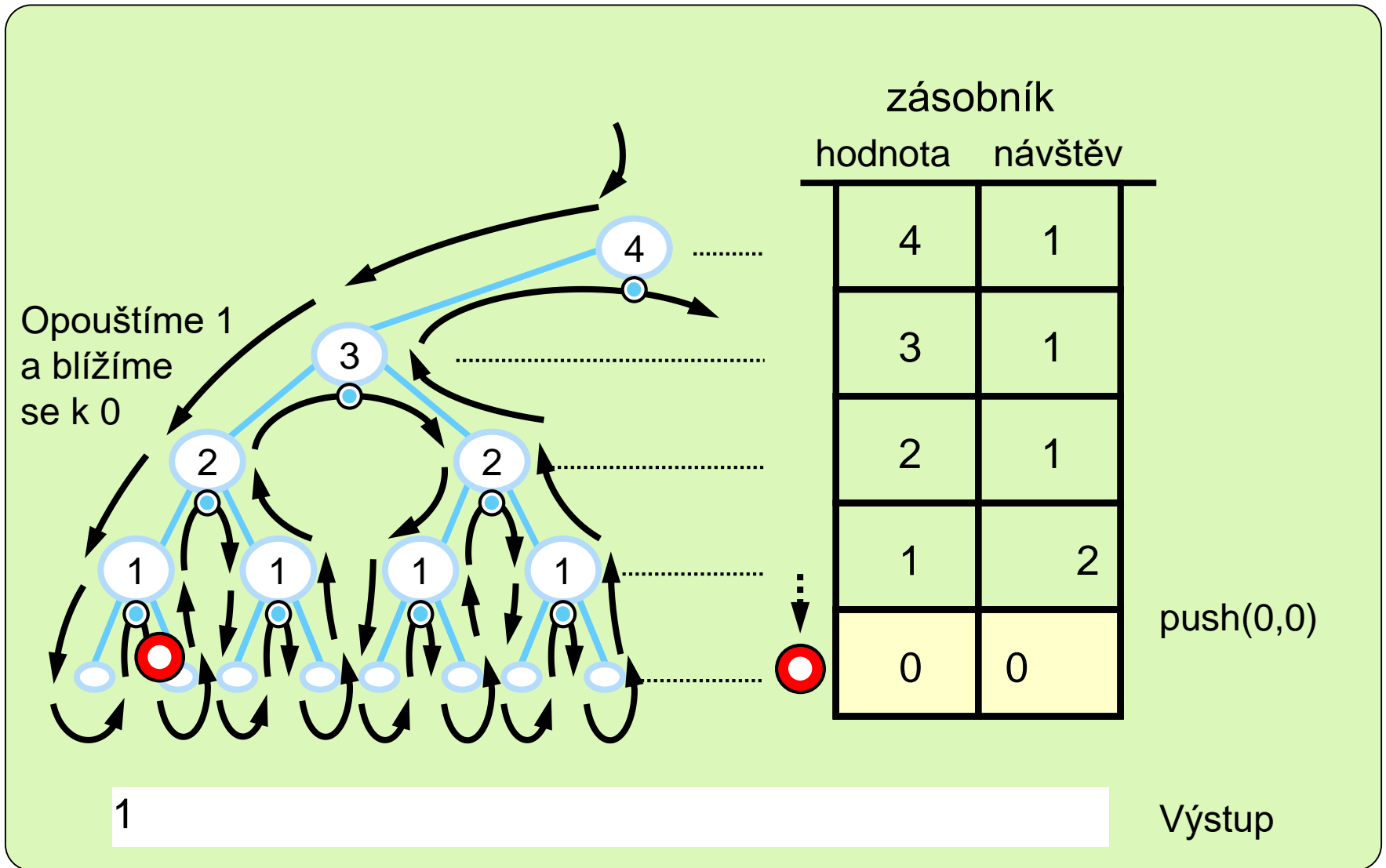
Zásobník implementuje rekurzi



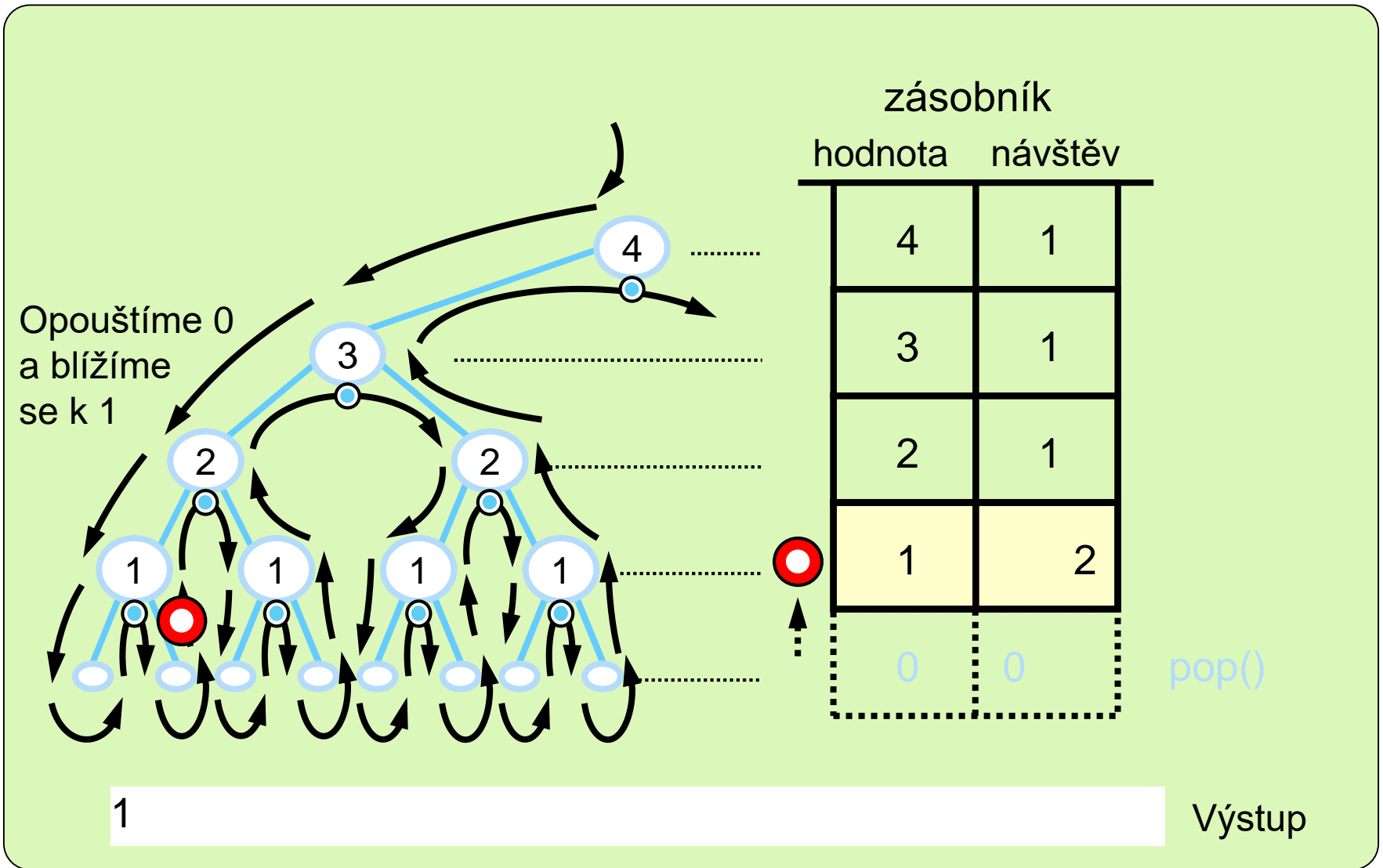
Zásobník implementuje rekurzi



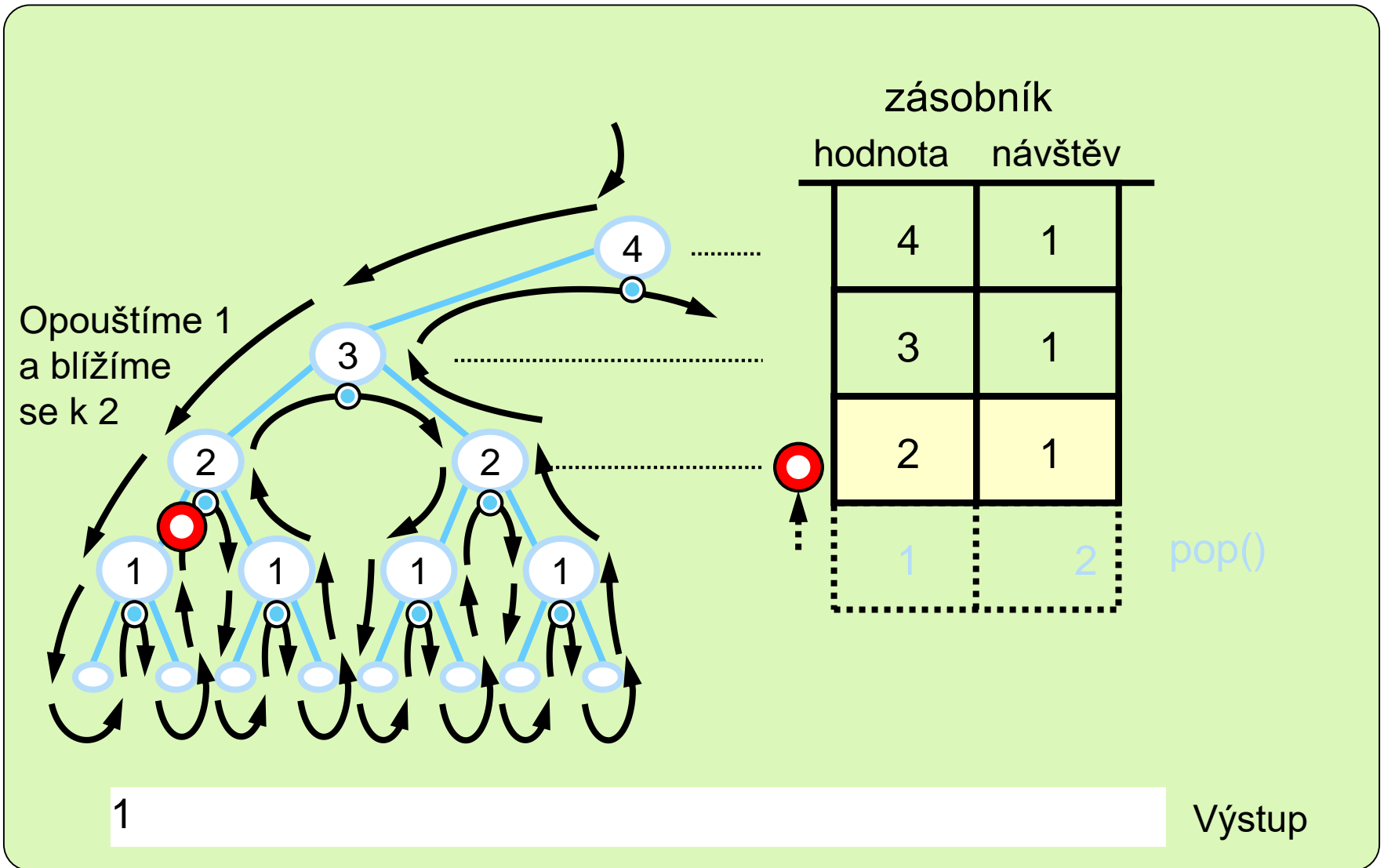
Zásobník implementuje rekurzi



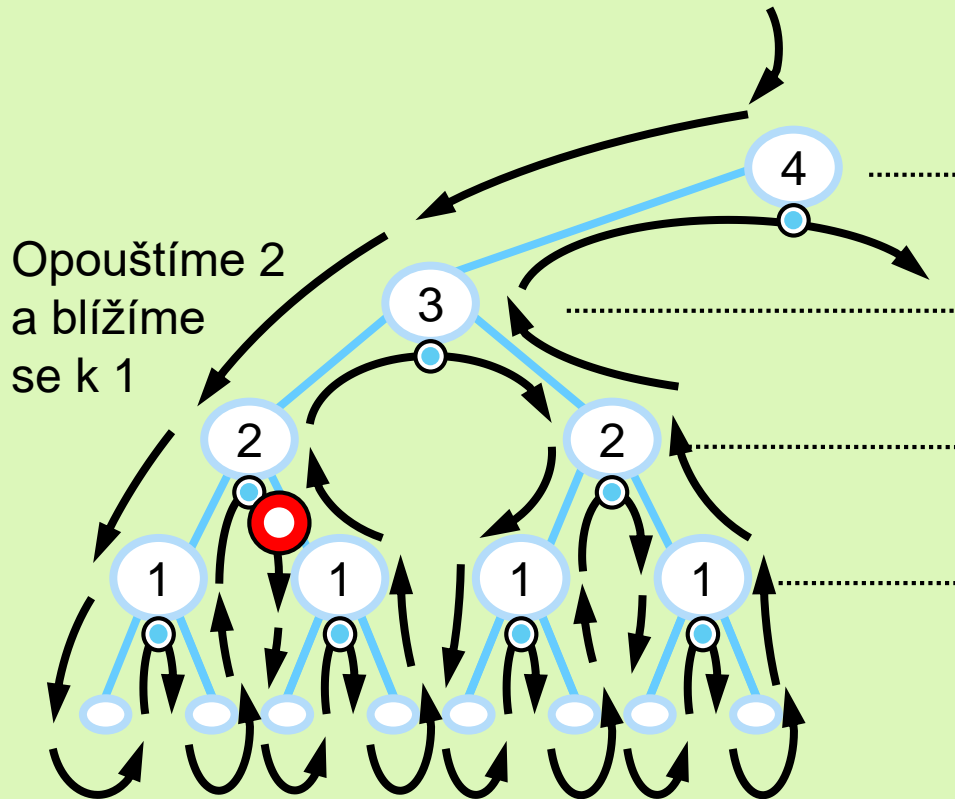
Zásobník implementuje rekurzi



Zásobník implementuje rekurzi



Zásobník implementuje rekurzi



zásobník

hodnota	návštěv
4	1
3	1
2	2
1	0

push(1,0)

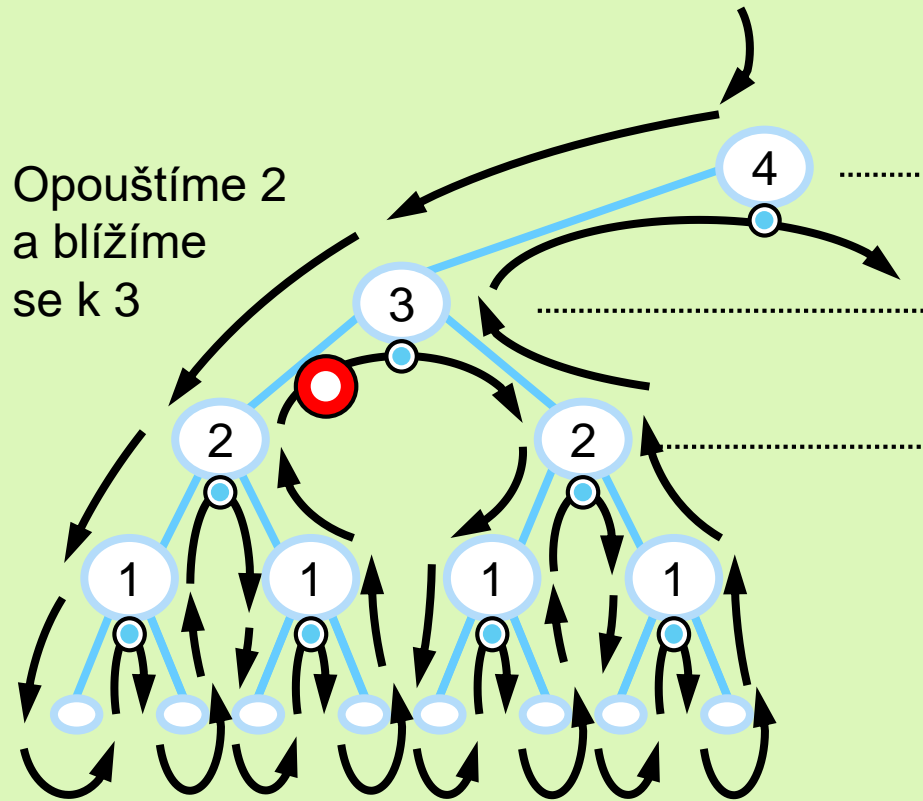
atd...

1 2

Výstup

Zásobník implementuje rekurzi

... po chvíli ...



Opouštíme 2
a blížíme
se k 3

zásobník

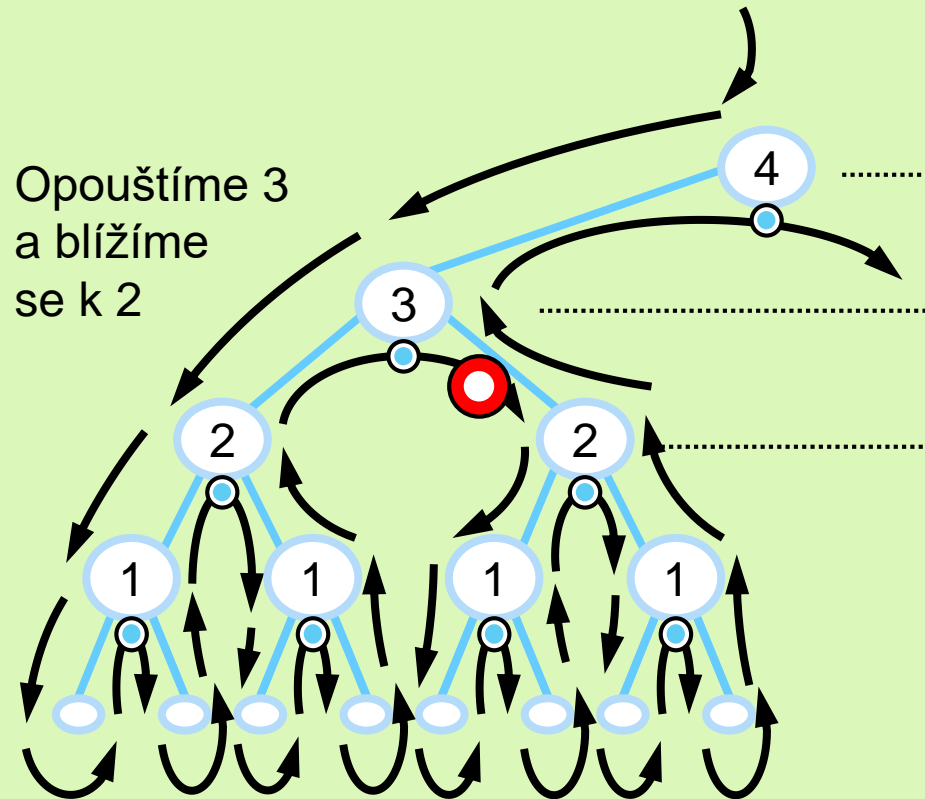
hodnota	návštěv
4	1
3	1
2	2

pop()

1 2 1

Výstup

Zásobník implementuje rekurzi



Opouštíme 3
a blížíme
se k 2

zásobník

hodnota	návštěv
4	1
3	2
2	0

push(2,0)

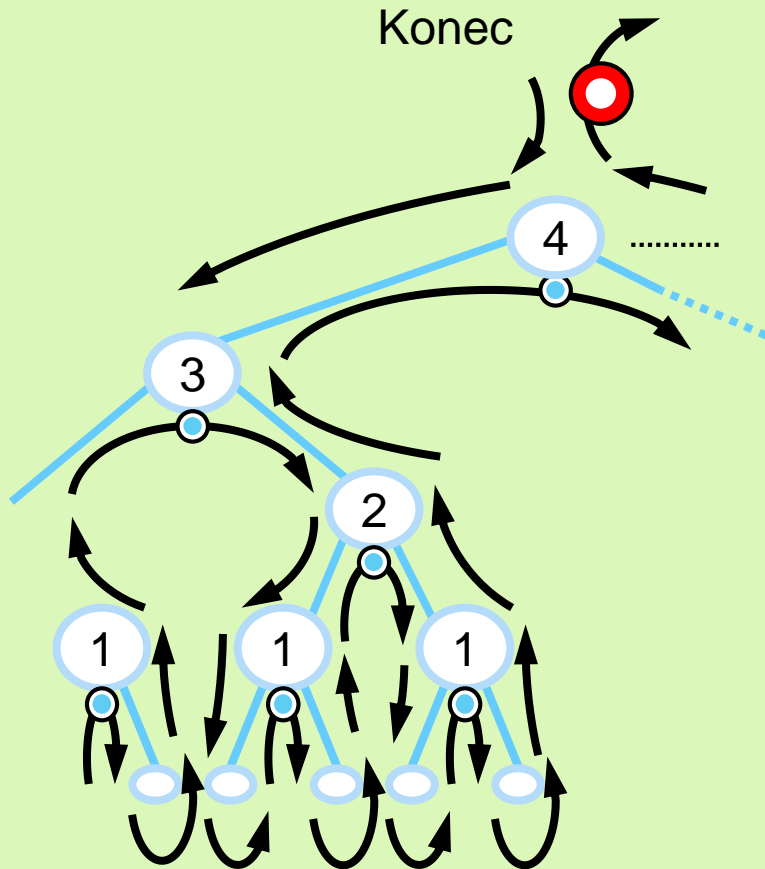
1 2 1 3

Výstup

atd...

Zásobník implementuje rekurzi

... a je hotovo



1 2 1 3 1 2 1 4 1 2 1 3 1 2 1

Výstup

Zásobník implementuje rekurzi

```
// Rekurzivní pravítko bez rekurzivníhop volání
// Pseudokód (skoro kód :-))

while (stack.empty() == false) {
    if (stack.top.hodnota == 0) stack.pop();
    if (stack.top.navstev == 0) {
        stack.top.navstev++;
        stack.push(stack.top.hodnota-1, 0);
    }
    if (stack.top.navstev == 1) {
        print(stack.top.hodnota);
        stack.top.navstev++;
        stack.push(stack.top.hodnota-1, 0);
    }
    if (stack.top.navstev==2) stack.pop();
}
```

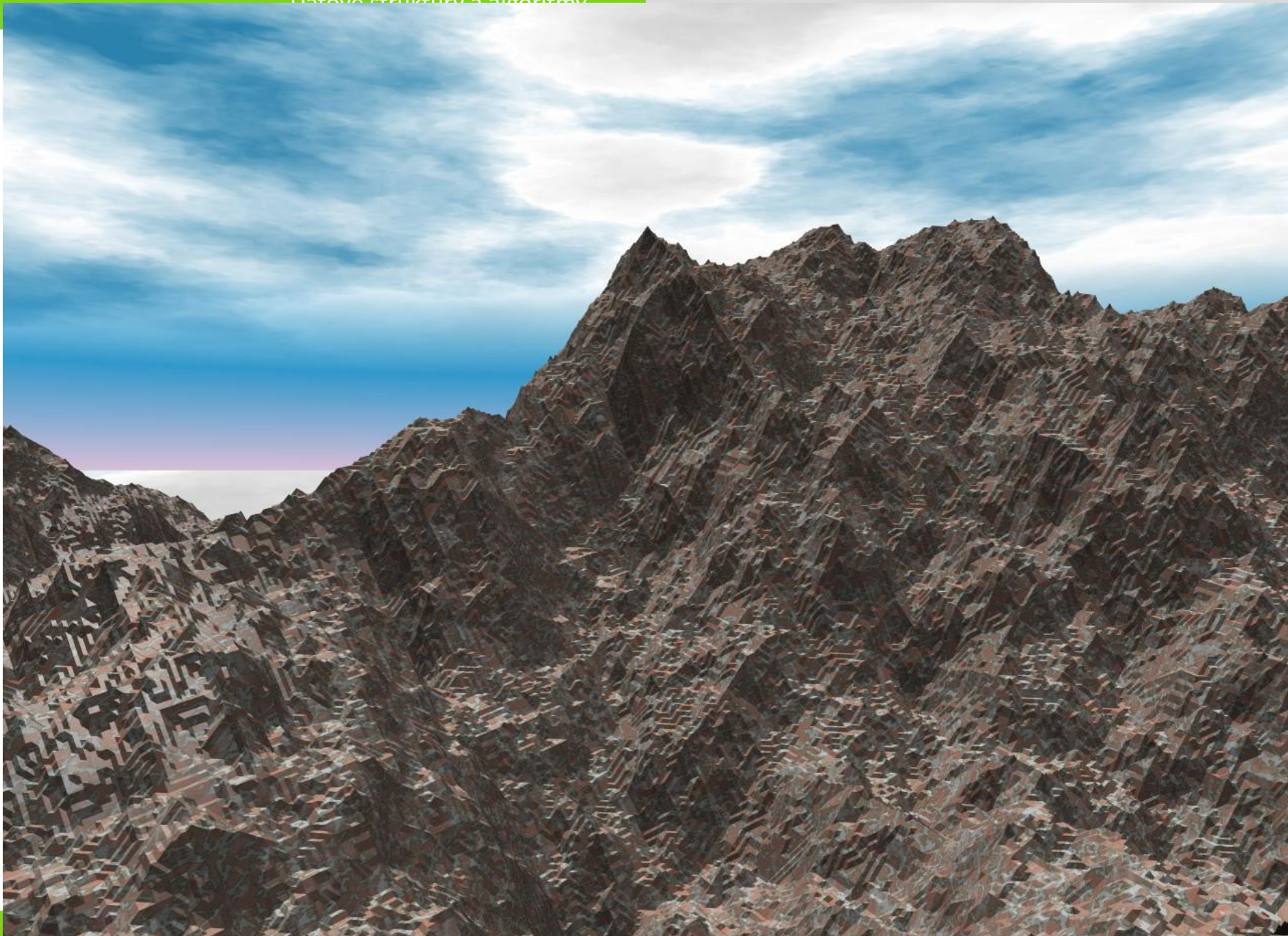

Zásobník implementuje rekurzi

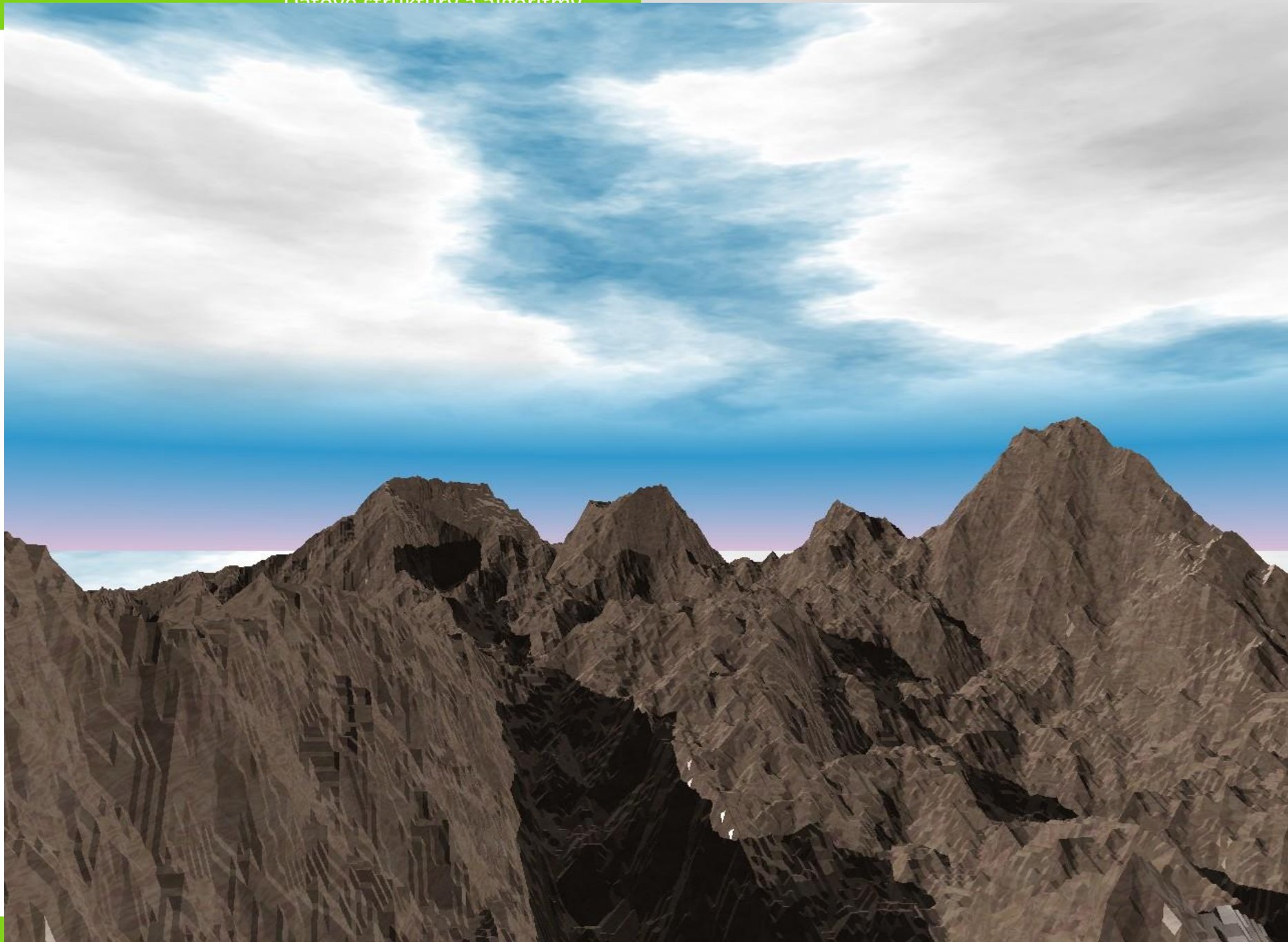
```

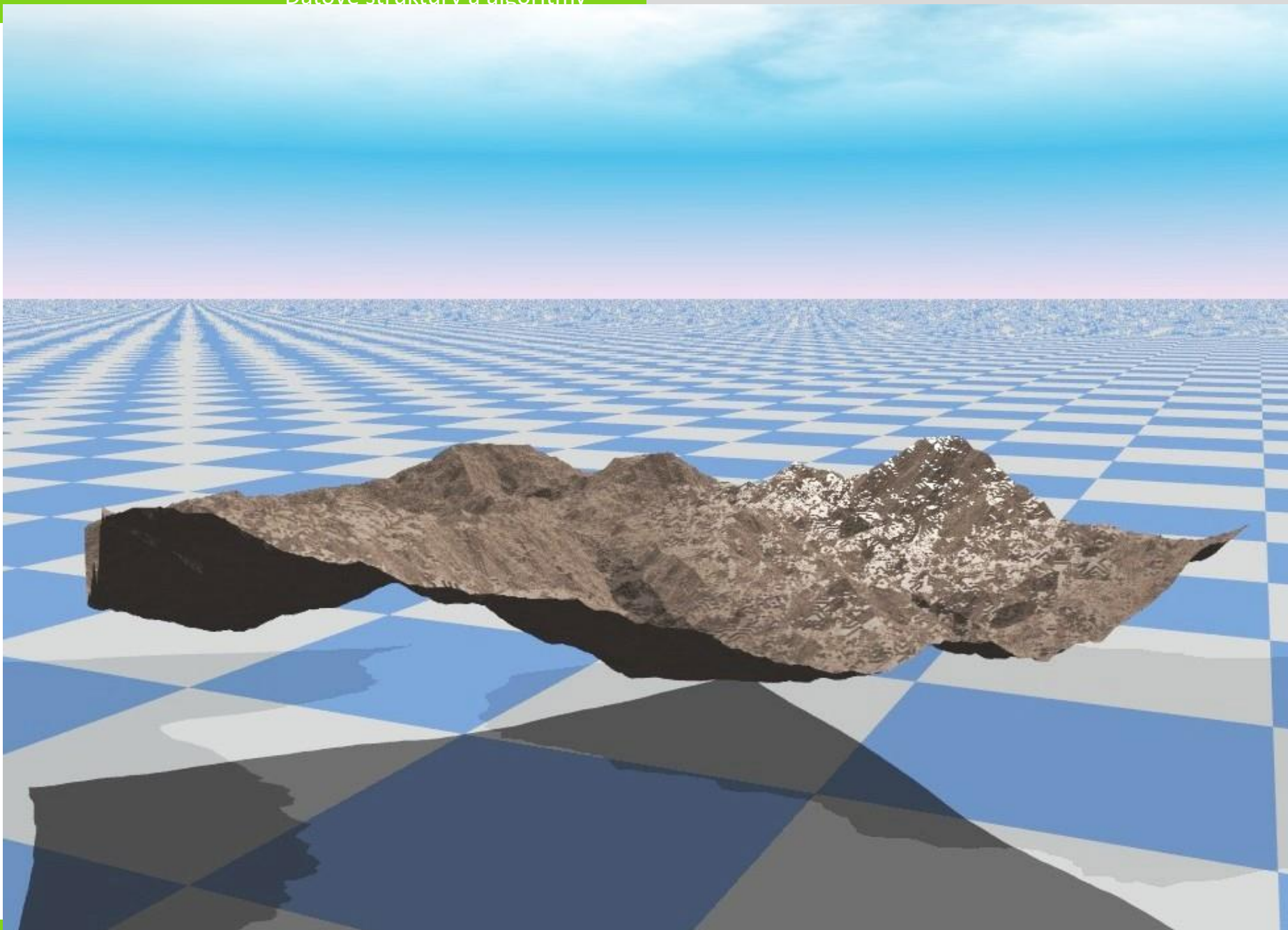
int zasHodn[10];  int zasNavst[10];  int SP;
// SP = Stack Pointer

void ruler2() {
    while (SP >= 0) {
        if (zasHodn[SP] == 0) SP--;           // pop
        if (zasNavst[SP] == 0) {             // první návštěva
            zasNavst[SP]++; SP++;
            zasHodn[SP] = zasHodn[SP-1]-1;  // jdi doleva
            zasNavst[SP] = 0;
        }
        if (zasNavst[SP] == 1) {           // druhá návštěva
            printf("%d%s", zasHodn[SP], " "); // zpracuj uzel
            zasNavst[SP]++; SP++;
            zasHodn[SP] = zasHodn[SP-1]-1;  // jdi dopreva
            zasNavst[SP] = 0;
        }
        if (zasNavst[SP] == 2) SP --;      // pop: uzel hotov
    } }

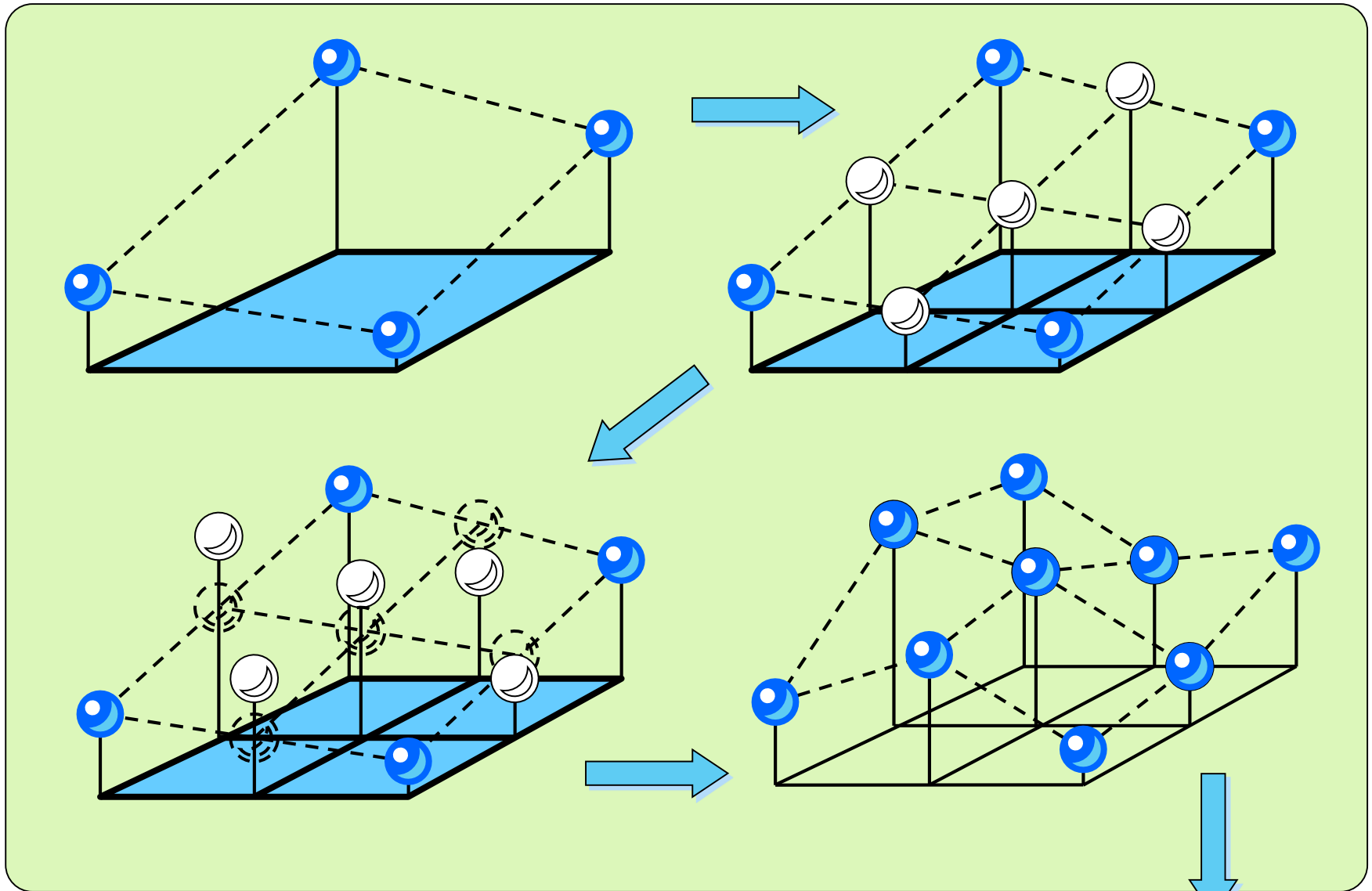
```



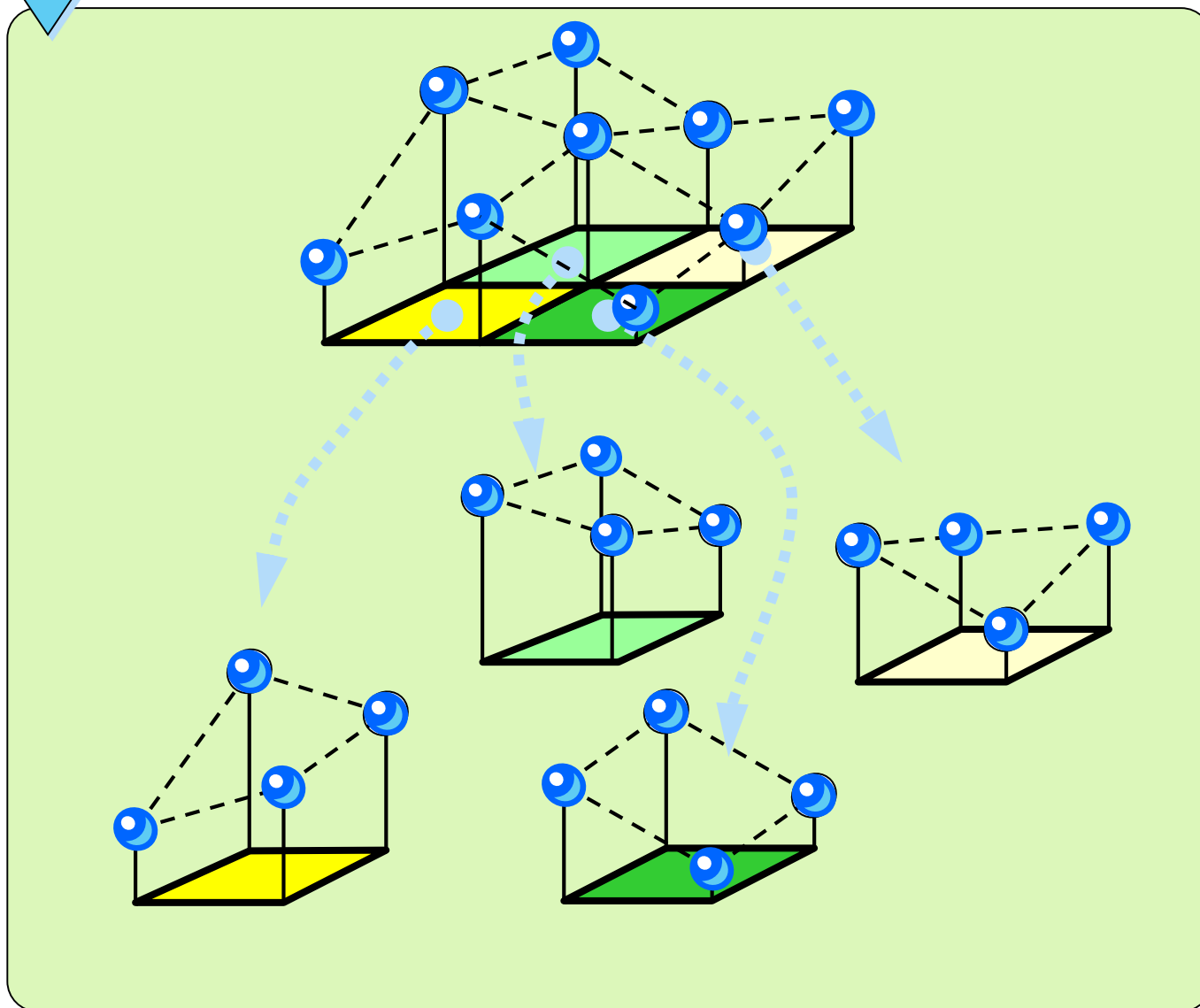




Rekurzivně definovaný terén

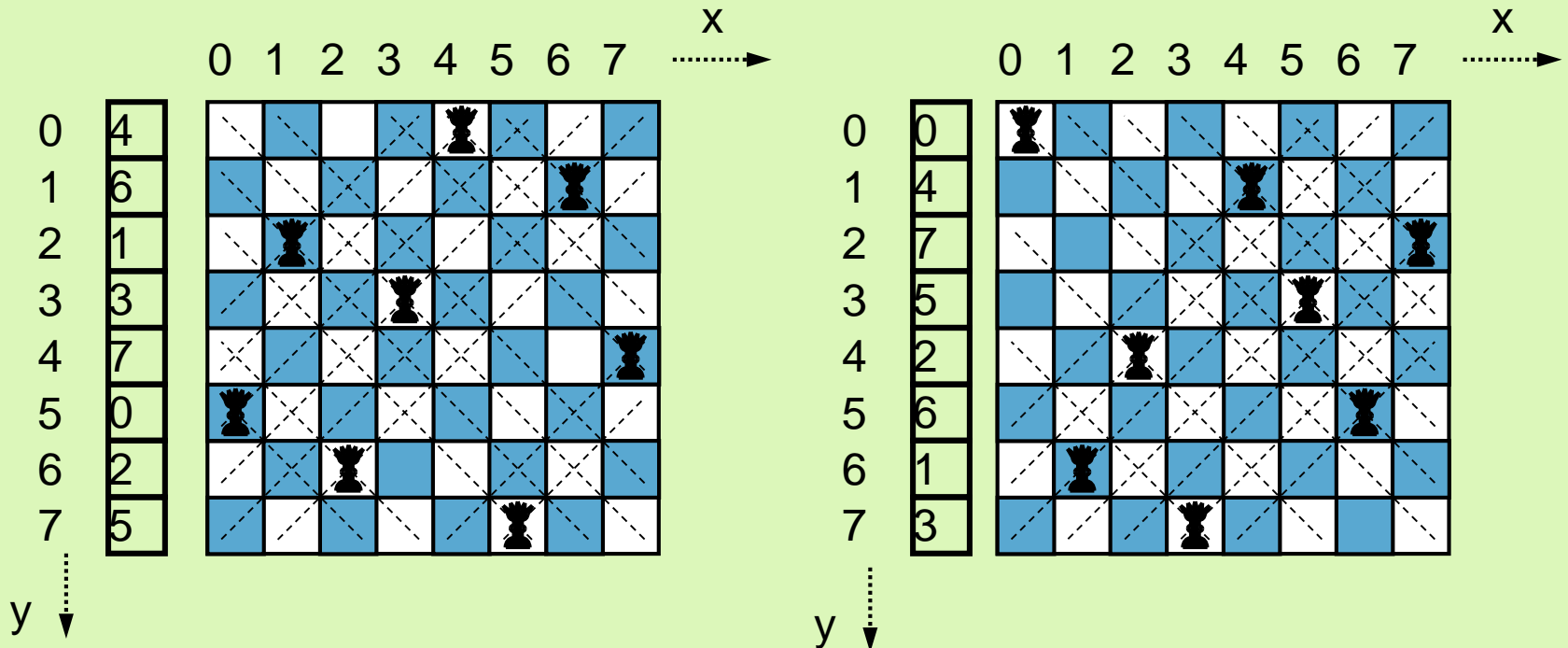


Rekurzivně definovaný terén

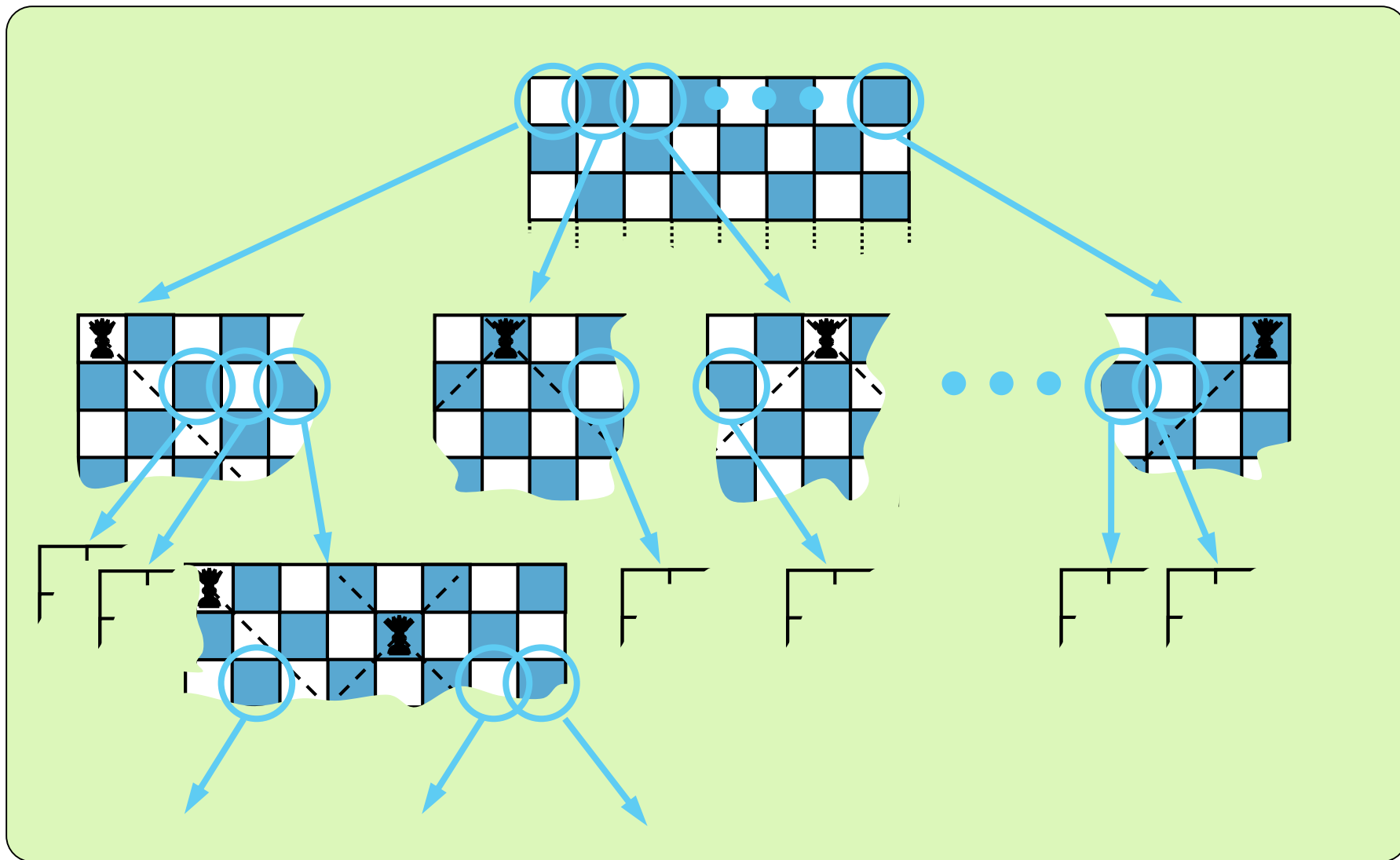


Snadné prohledávání s návratem (backtrack)

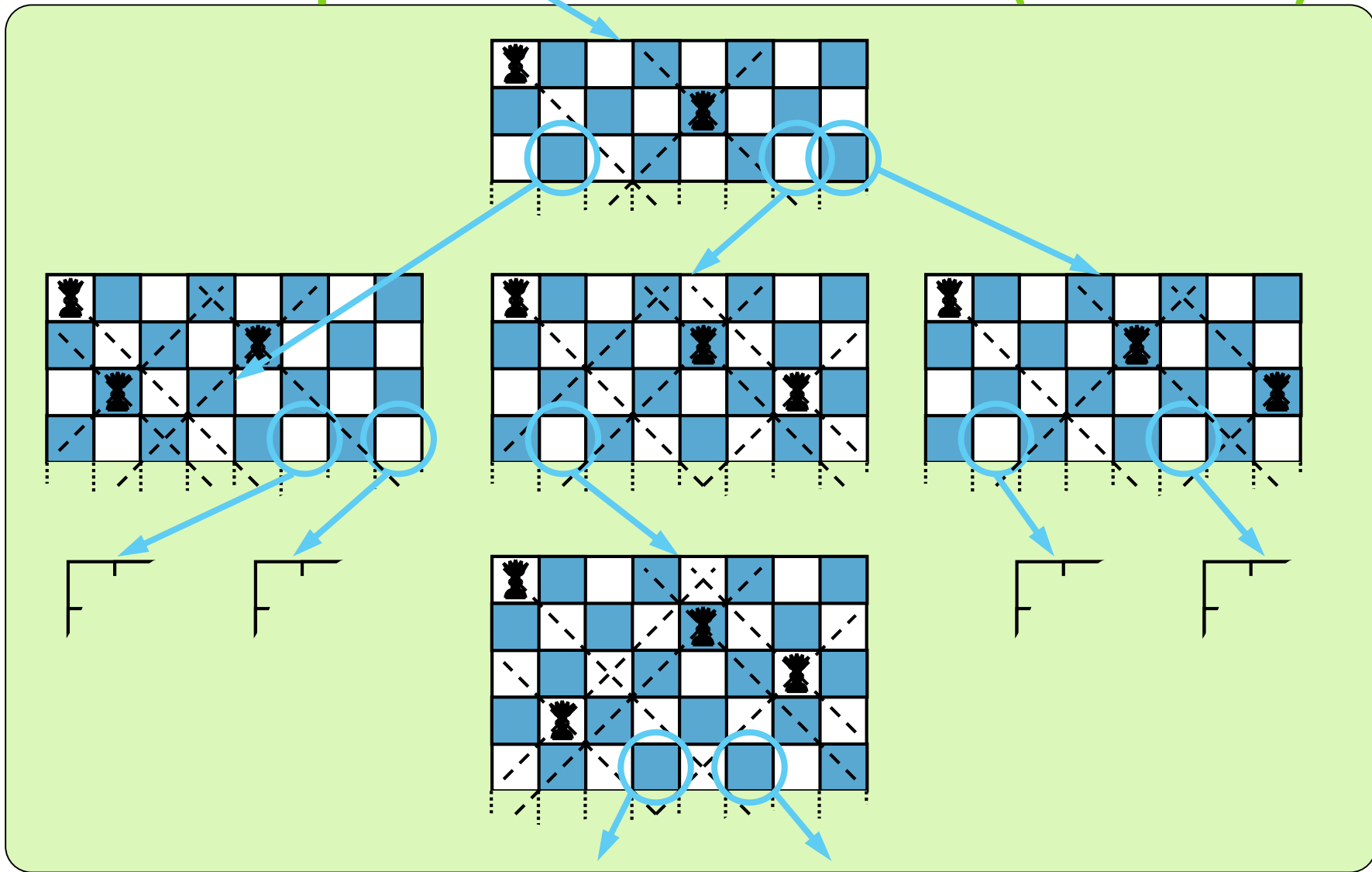
Problém osmi dam na šachovnici



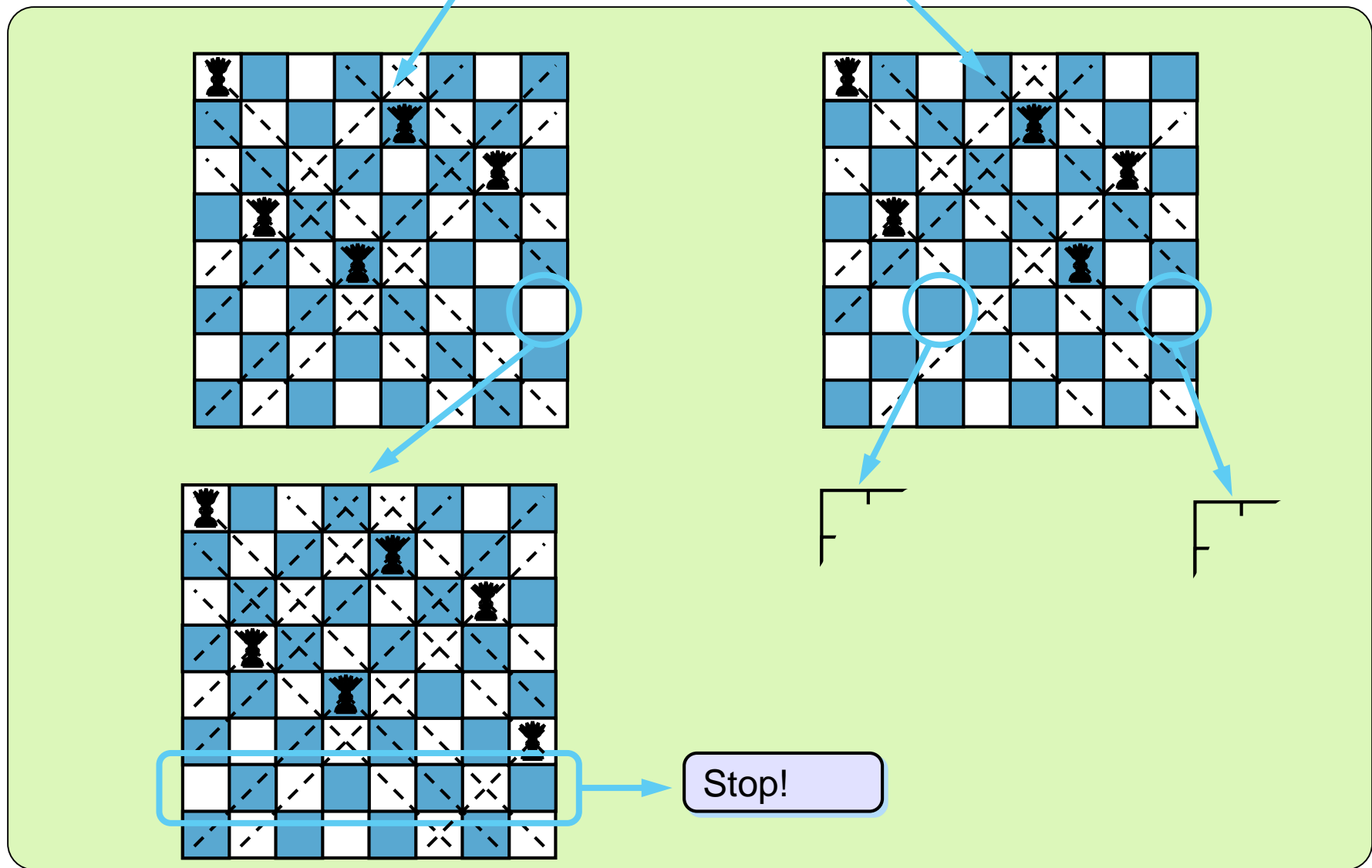
Snadné prohledávání s návratem (backtrack)



Snadné prohledávání s návratem (backtrack)



Snadné prohledávání s návratem (backtrack)



Snadné prohledávání s návratem (backtrack)

N	Počet 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

Tab.: Rychlosti řešení problému osmi dam

Snadné prohledávání s návratem (backtrack)

```

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

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

```

Call: tryPutColumn(0);



Salvador Dalí Tvář války (1940-1941)

The End