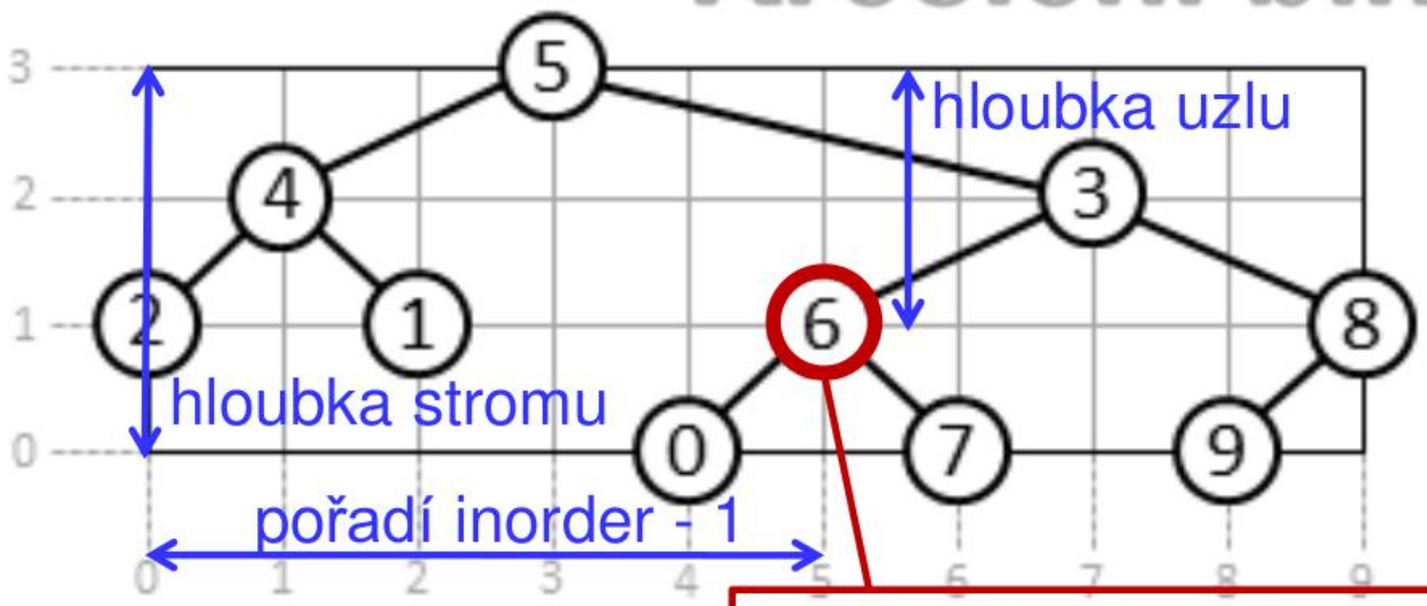
Kreslení binárního stromu



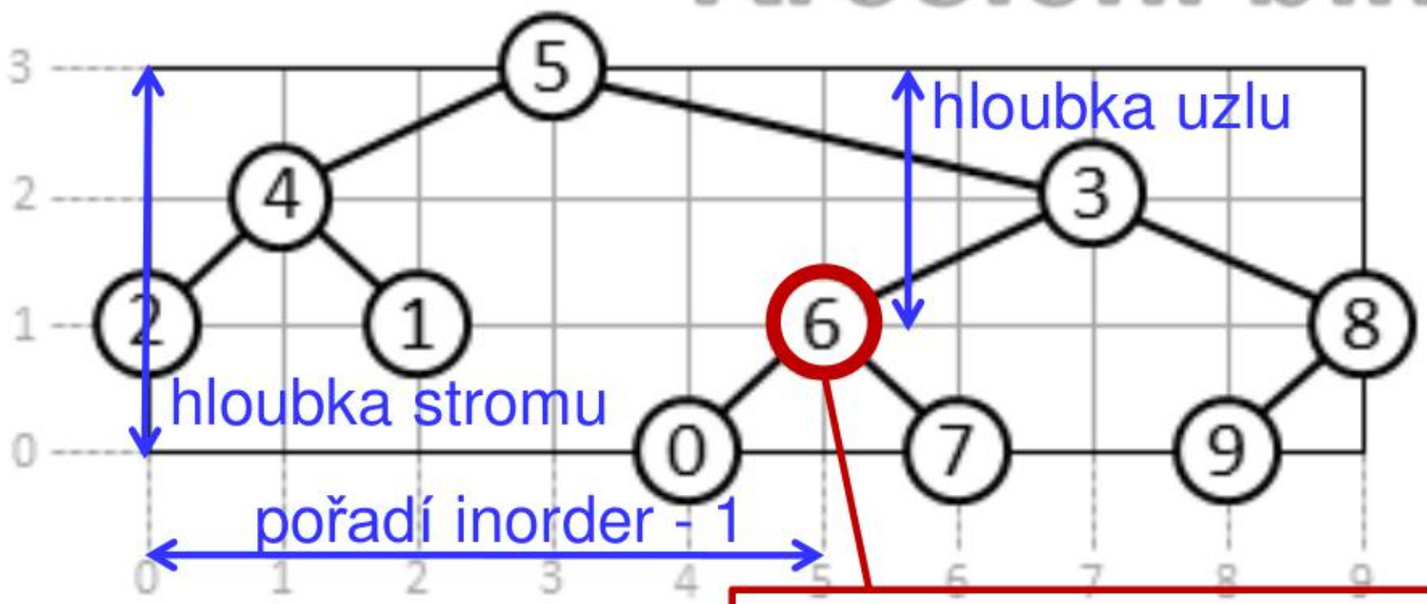

```
static int treeDepth = 0;
static int counter = 0;

class Node {
    Node left, right;
    int key;
    int depth, inorder; }
```

```
void process(Node node, int depth) {
    if (node == null) return;
    node.depth = depth;
    if (depth > treeDepth) treeDepth = depth;
    process(node.left, depth + 1);
    node.inorder = ++counter;
    process(node.right, depth + 1);  }
```

```
Call: process(root, 0);
```

Kreslení binárního stromu




```
static int treeDepth = 0;
static int counter = 0;

class Node {
    Node left, right;
    int key;
    int depth, inorder; }
```

```
void process(Node node, int depth) {
    if (node == null) return;
    node.depth = depth;
    if (depth > treeDepth) treeDepth = depth;
    process(node.left, depth + 1);
    node.inorder = ++counter;
    process(node.right, depth + 1);  }
```

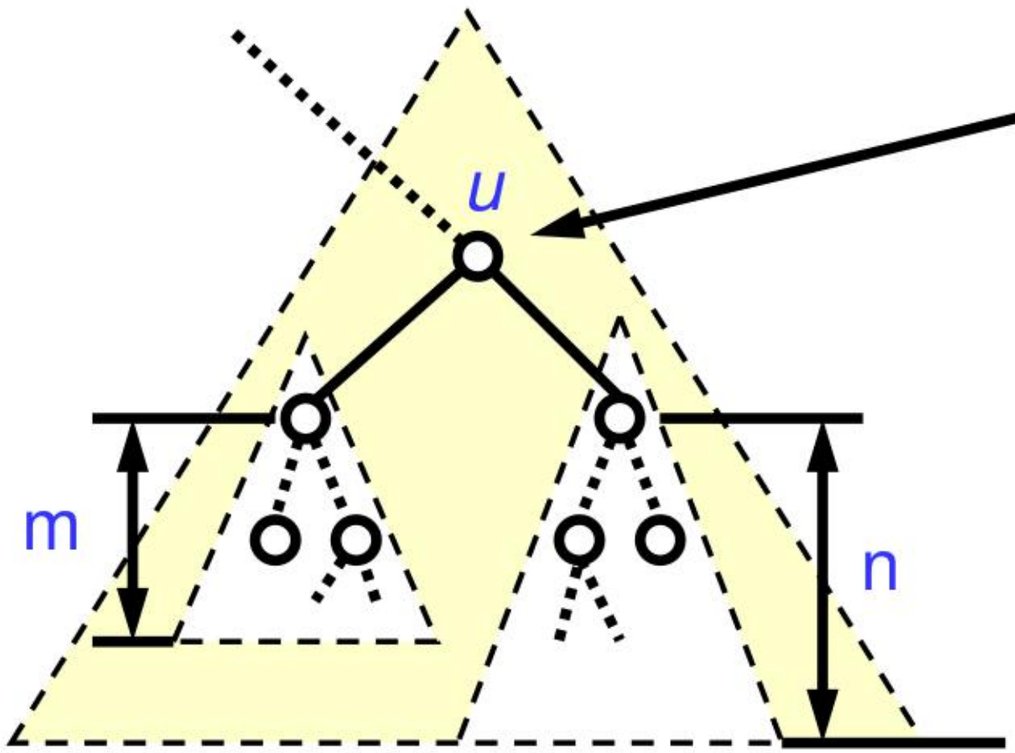
```
Call: process(root, 0);
```

Yes 61 98%

No 1 1%

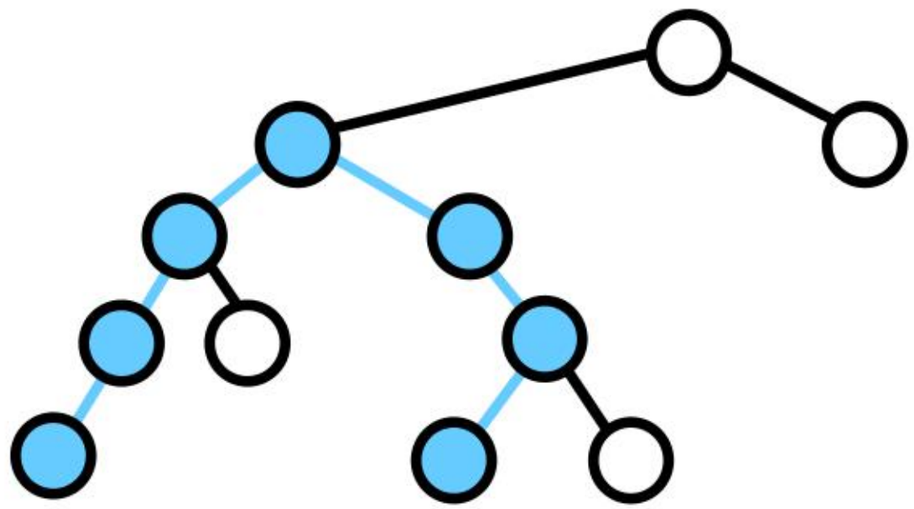
Počítání nejdelší cesty

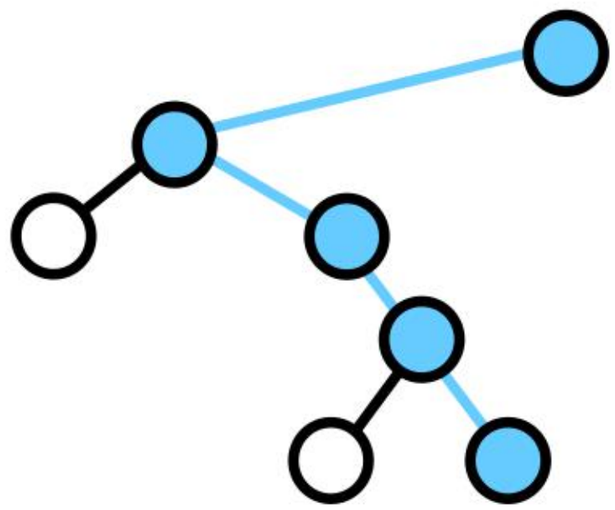
Strom nebo podstrom



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

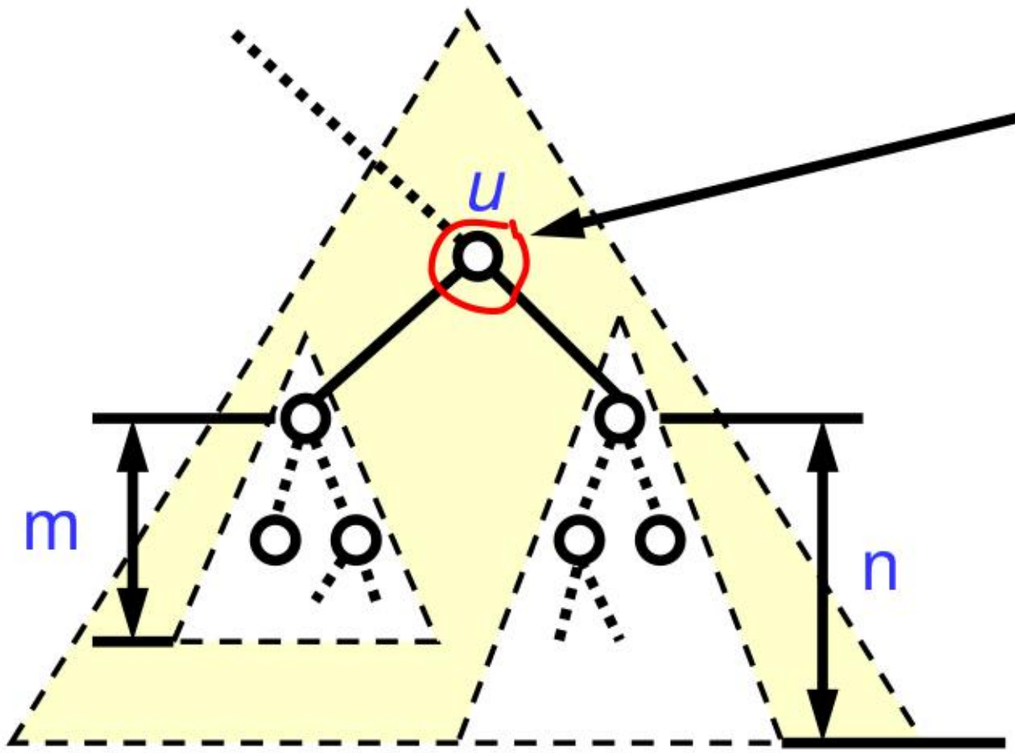
Příklady





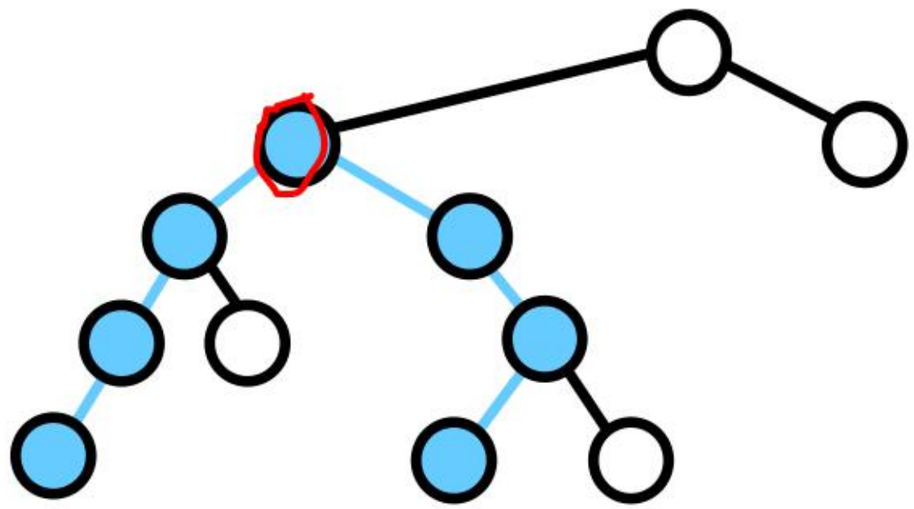
Počítání nejdelší cesty

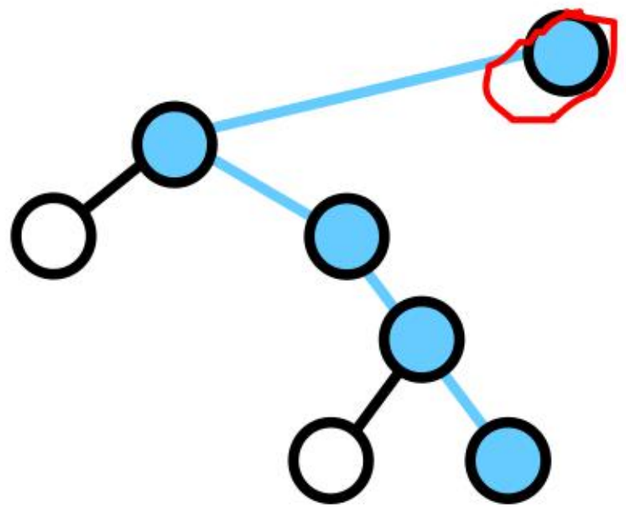
Strom nebo podstrom



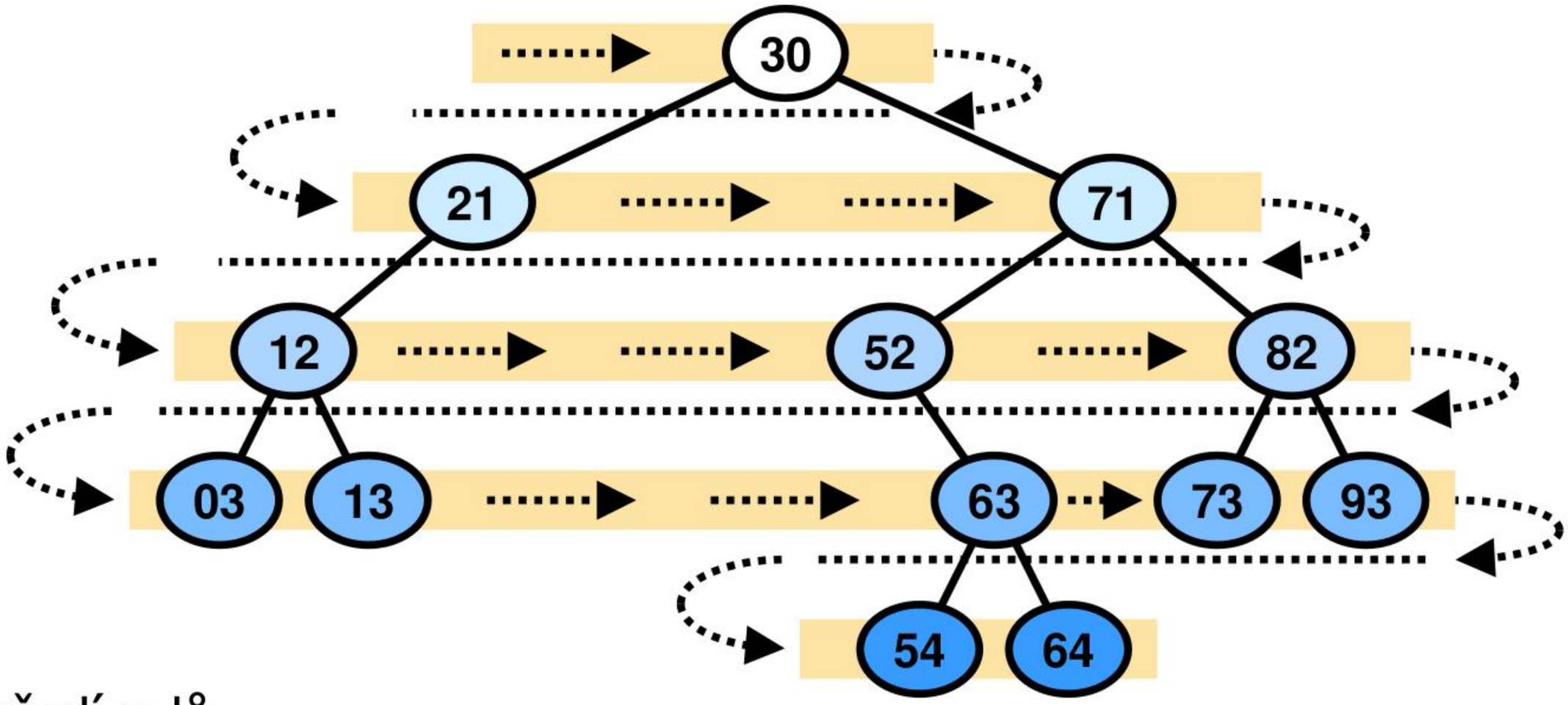
m+n+2 … délka nejdelší cesty s nejvyšším bodem v uzlu *u*

Příklady





Průchod stromem do šířky

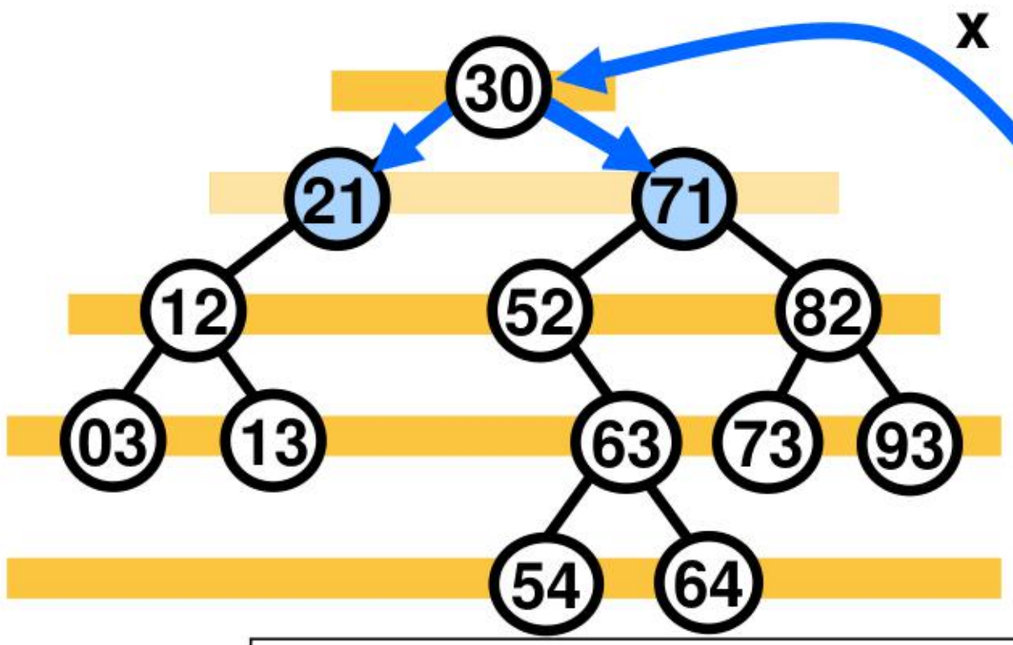


Pořadí uzlů

30	21	71	12	52	82	03	13	63	73	93	54	64

Struktura stromu ani rekurzivní přístup tento průchod nepodporují.

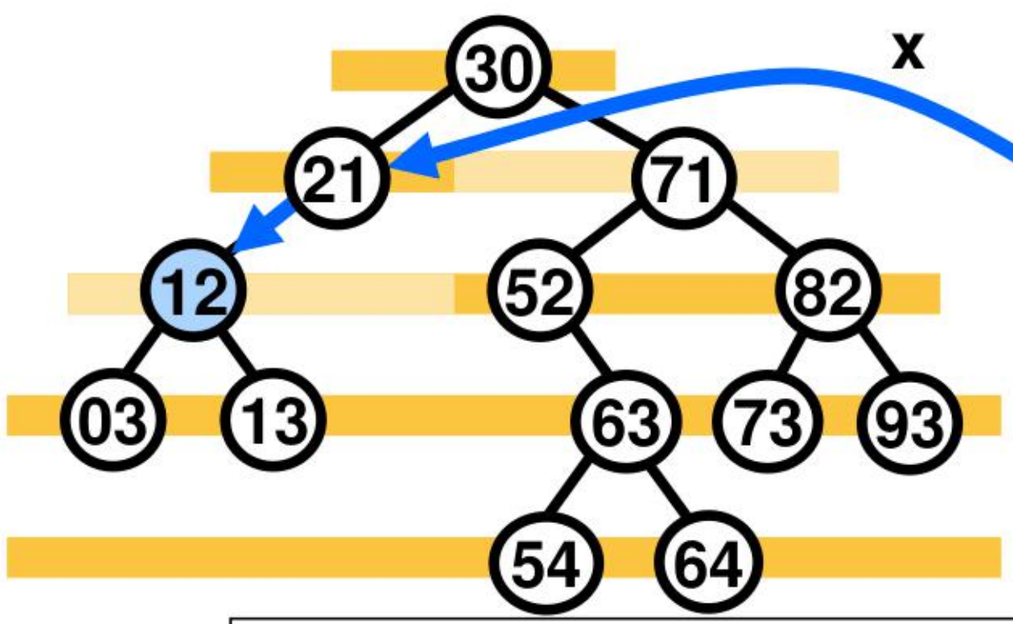
Průchod stromem do šířky



1. x = Odeber(), tisk(x.key).

2. Vlož(x.left), vlož(x.right). *)

Výstup 30



1. x = Odeber(), tisk(x.key).

2. Vlož(x.left), vlož(x.right). *)

Výstup 30 21

*) pokud existuje

Porovnání časové složitosti

Který z průchodů stromem má asymptoticky menší časovou složitost?

A. průchod do hloubky
B. průchod do šířky
C. vyjde to nastejno

Porovnání časové složitosti

Který z průchodů stromem má asymptoticky menší časovou složitost?

A. průchod do hloubky

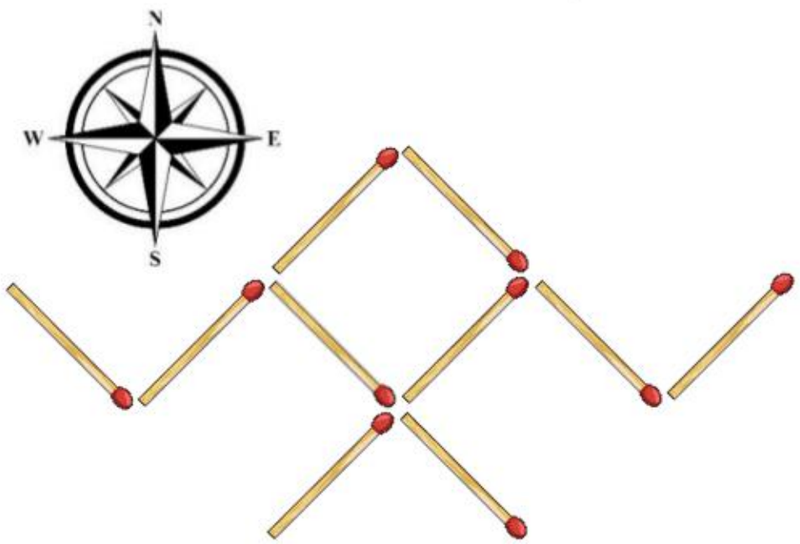
B. průchod do šířky

C. vyjde to nastejno

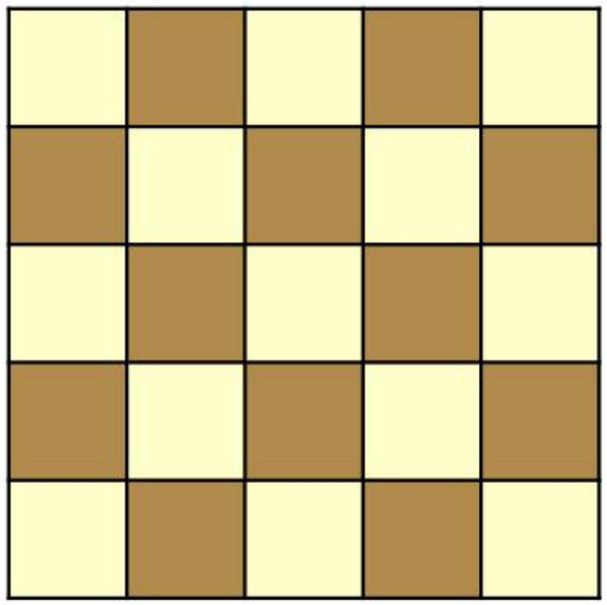
$\Theta(n)$

Kvízy

Přesuňte 3 sirky tak, aby vlaštovka letěla na jih.



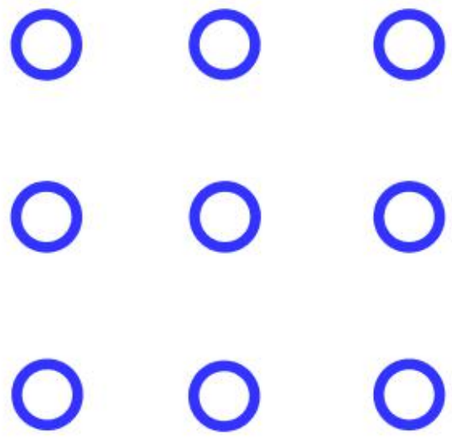
Rozestavte na šachovnici 5 dam tak, aby se neohrožovaly.



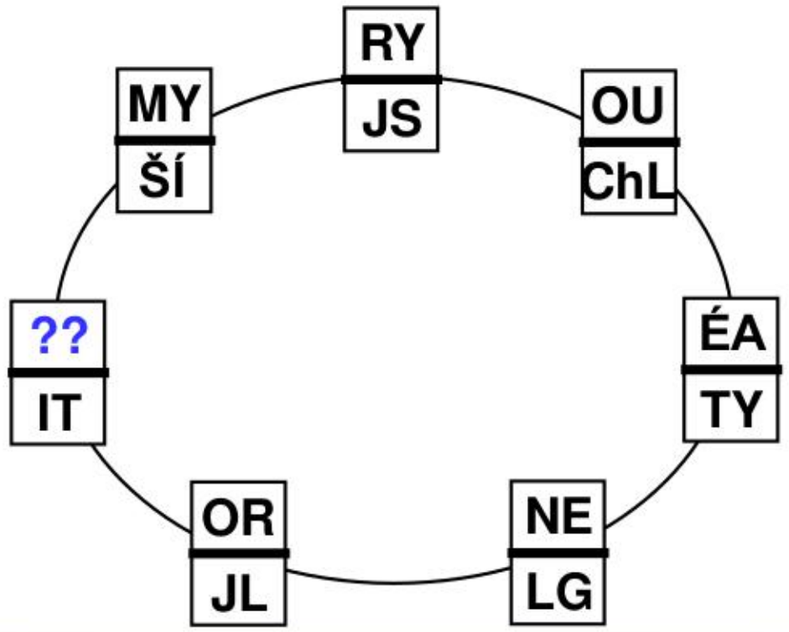
Přesuňte právě jednu z pěti modrých číslic tak, aby rovnost platila.

$$62 - 63 = 1$$

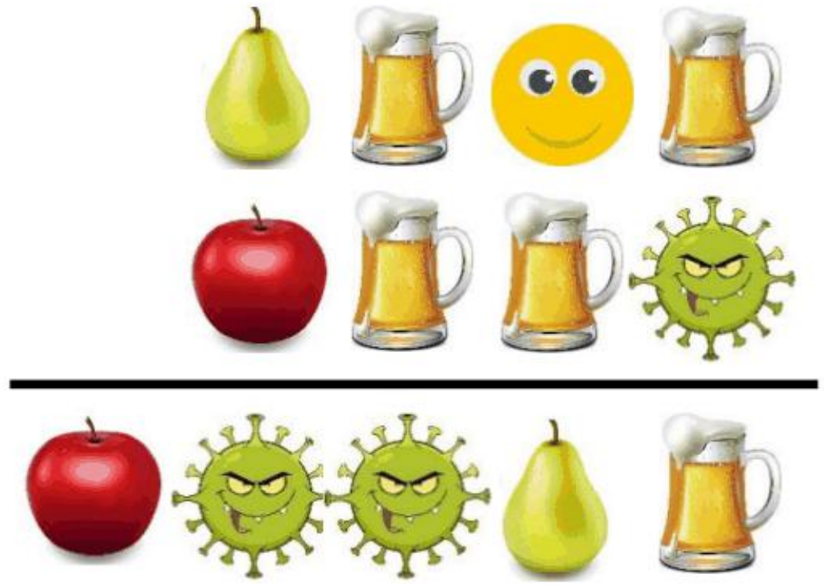
Nakreslete lomenou čáru sestávající ze 4 na sebe navazujících úseček, která protne všech 9 kružnic.



Jaká dvojice písmen logicky patří na místo otazníků?

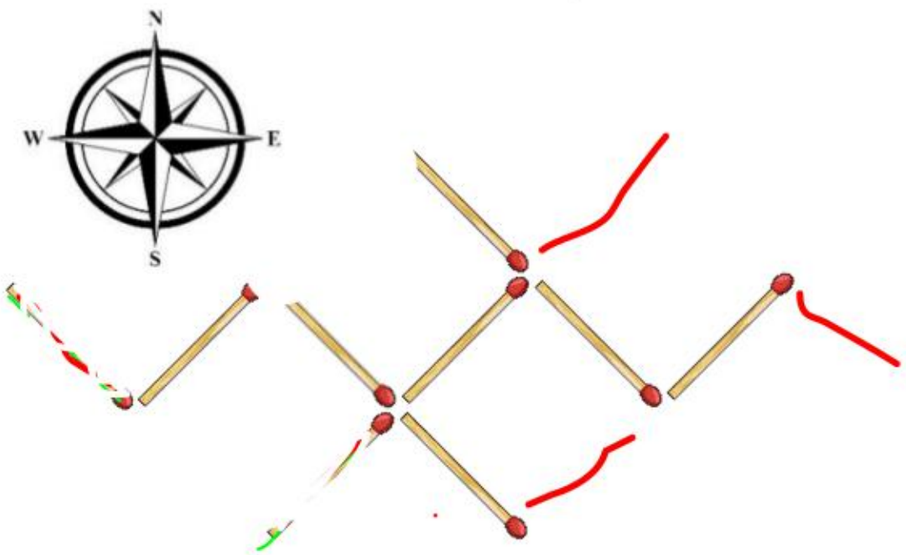


Vyřešte algebrogram.

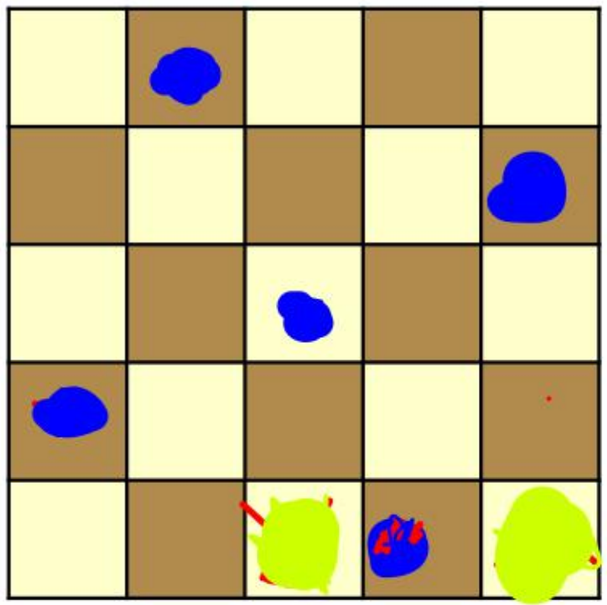


Kvízy

Přesuňte 3 sirky tak, aby vlaštovka letěla na jih.



Rozestavte na šachovnici 5 dam tak, aby se neohrožovaly.



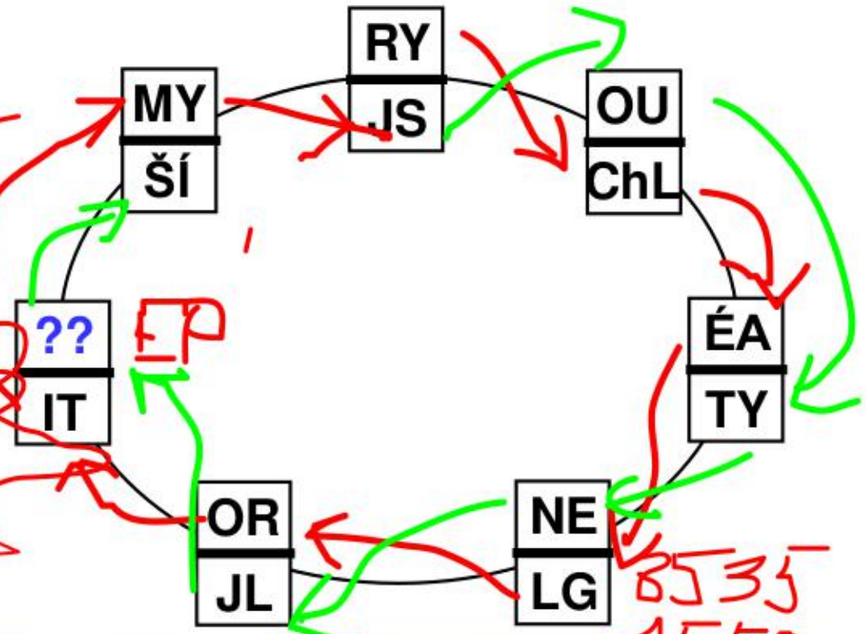
Přesuňte právě jednu z pěti modrých číslic tak, aby rovnost platila.

$$62 - 63 = 1$$

$64 = 2^6 - 63 = 1$ respekt

Nakreslete lomenou čáru sestávající ze 4 na sebe navazujících úseček, která protne všech 9 kružnic.

Jaká dvojice písmen logicky patří na místo otazníků?



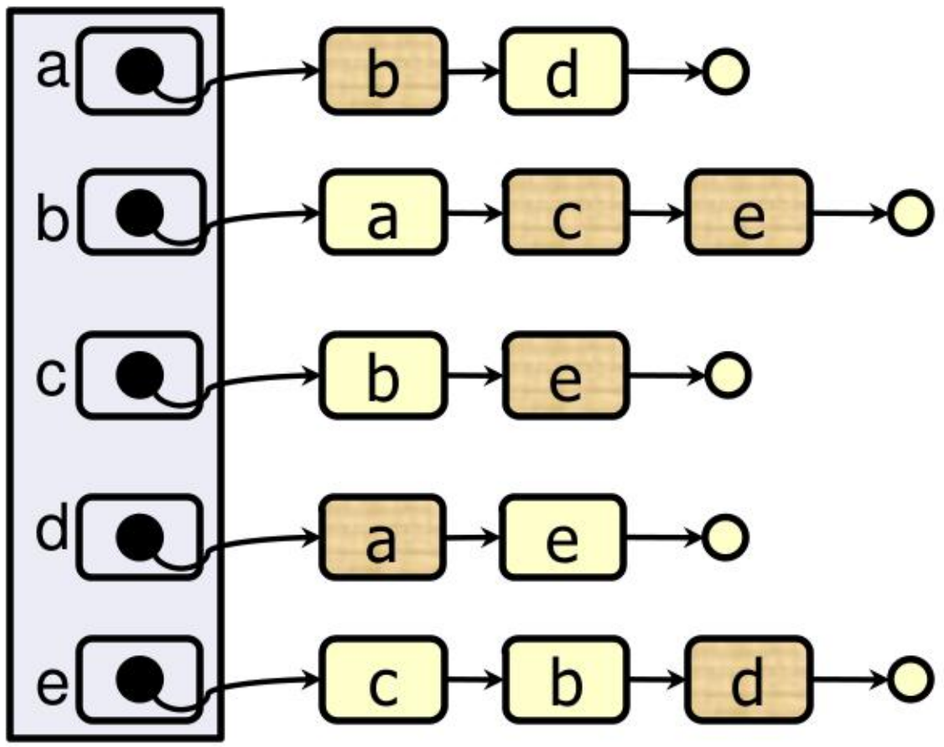
Vyřešte algebrogram.

Reprezentace grafu v paměti

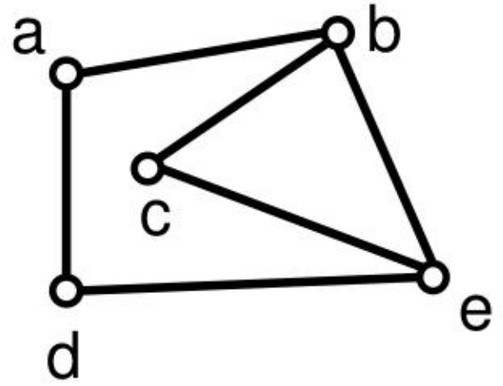
Matice sousednosti:

	a	b	c	d	e
a	0	**1**	0	**1**	0
b	**1**	0	**1**	0	**1**
c	0	**1**	0	0	**1**
d	**1**	0	0	0	**1**
e	0	**1**	**1**	**1**	0

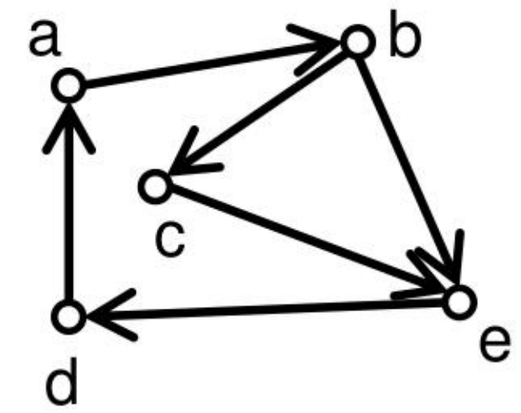
Seznam sousedů:



$G_1 = (V, E_1)$



$G_2 = (V, E_2)$



Reprezentace grafu G_2 obsahuje pouze prvky s pozadím 

Prostorová složitost

$\Theta(|V|^2)$

test existence hrany v konstantním čase

$\Theta(|V| + |E|)$

výhodnější pro řídké grafy

DFS se zásobníkem

```
DFS_iterative(Node node):
    S = new Stack();
    S.push(node);
    while !S.isEmpty() do
        node = S.pop();
        if !node.visited then
            node.visited = true;
            foreach n in node.neighbors do
                S.push(n);
```

- jiné pořadí uzlů než při rekurzi
- uzel může být na zásobník vložen opakovaně

simuluje rekurzi

```
DFS_iterative2(Node node):
    S = new Stack();
    S.push(new Iterator(node.neighbors));
    while !S.isEmpty() do
        if S.peek().hasNext() then
            n = S.peek().next();
            if !n.visited then
                n.visited = true;
                S.push(new Iterator(node.n));
        else
            node = S.pop();
```

DFS se zásobníkem

```
DFS_iterative(Node node):
    S = new Stack();
    S.push(node);
    while !S.isEmpty() do
        node = S.pop();
        if !node.visited then
            node.visited = true;
            foreach n in node.neighbors do
                S.push(n);
```

- jiné pořadí uzlů než při rekurzi
- uzel může být na zásobník vložen opakovaně

simuluje rekurzi

```
DFS_iterative2(Node node):
    S = new Stack();
    S.push(new Iterator(node.neighbors));
    while !S.isEmpty() do
        if S.peek().hasNext() then
            n = S.peek().next();
            if !n.visited then
                n.visited = true;
                S.push(new Iterator(node.n));
        else
            node = S.pop();
```

Časová složitost DFS

Která z následujících možností nejlépe popisuje časovou složitost DFS pro graf $G = (V, E)$?

A. $O(|E|)$

B. $O(|V| + |E|)$

C. $O(|V|^2)$

D. $O(|V| \cdot |E|)$

Časová složitost DFS

```
DFS(Node node):
    node.visited = true;
    foreach n in node.neighbors do
        if !n.visited then
            DFS(n);
        end;
    end;
```

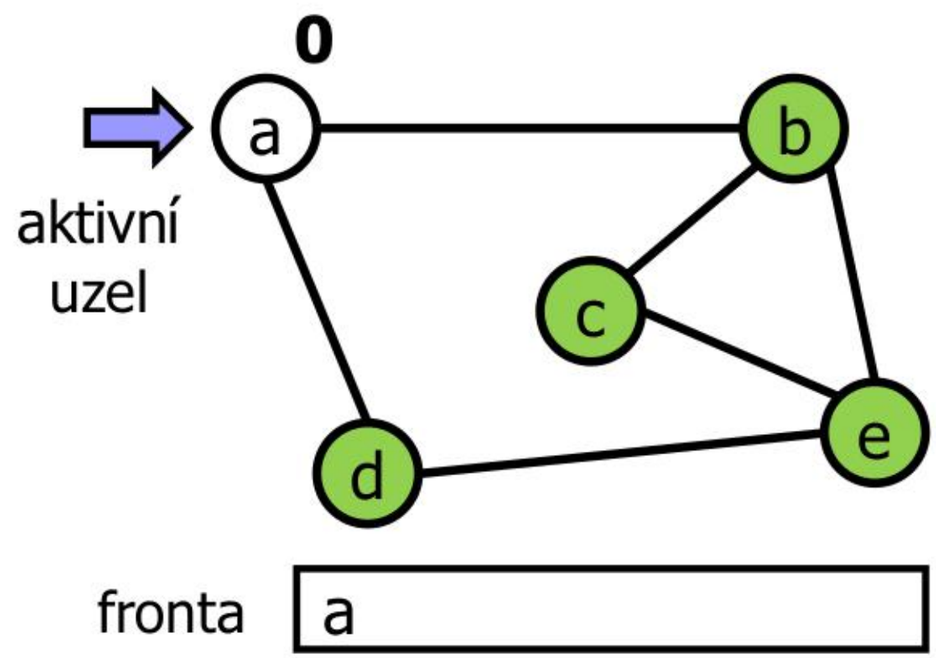
```
DFS(Node[] nodes):
    foreach node in nodes do
        if !node.visited then
            DFS(node);
        end;
    end;
```

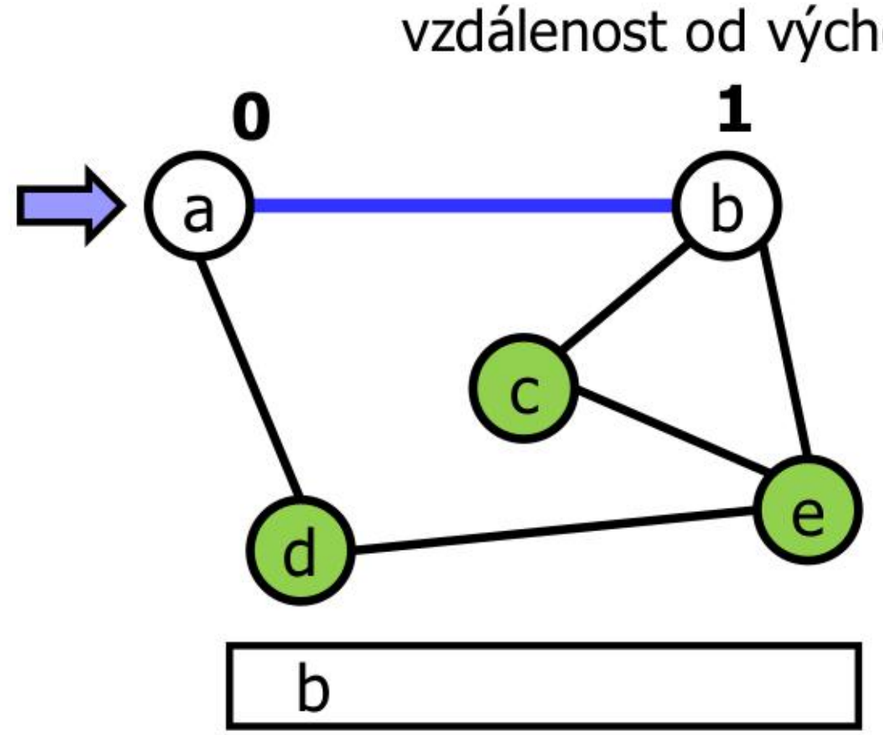
$G = (V, E)$ reprezentovaný jako seznam sousedů

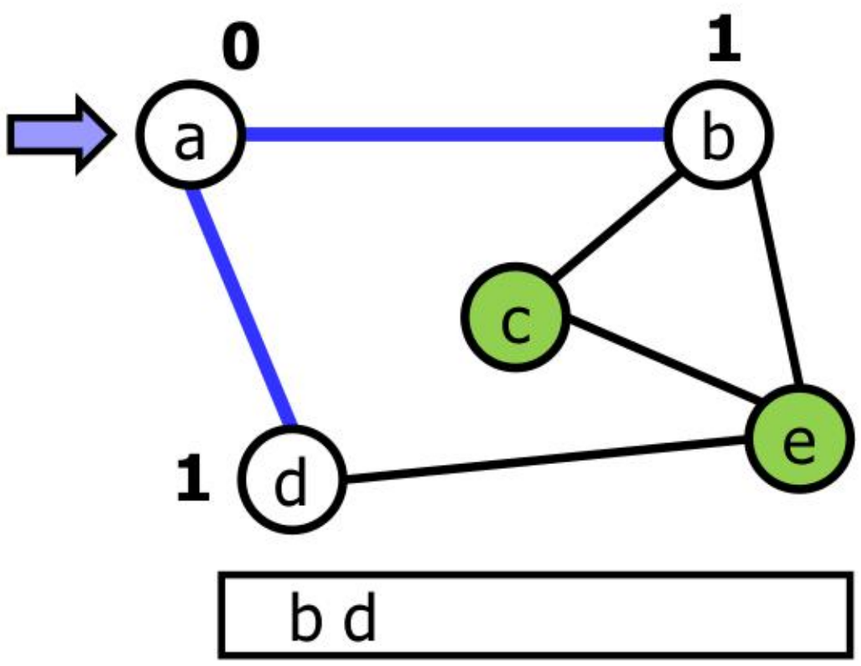
$$T(|V|, |E|) = O\left(|V| + \sum_{v \in V} \mathrm{d}_v\right) = O(|V| + 2|E|) = O(|V| + |E|)$$

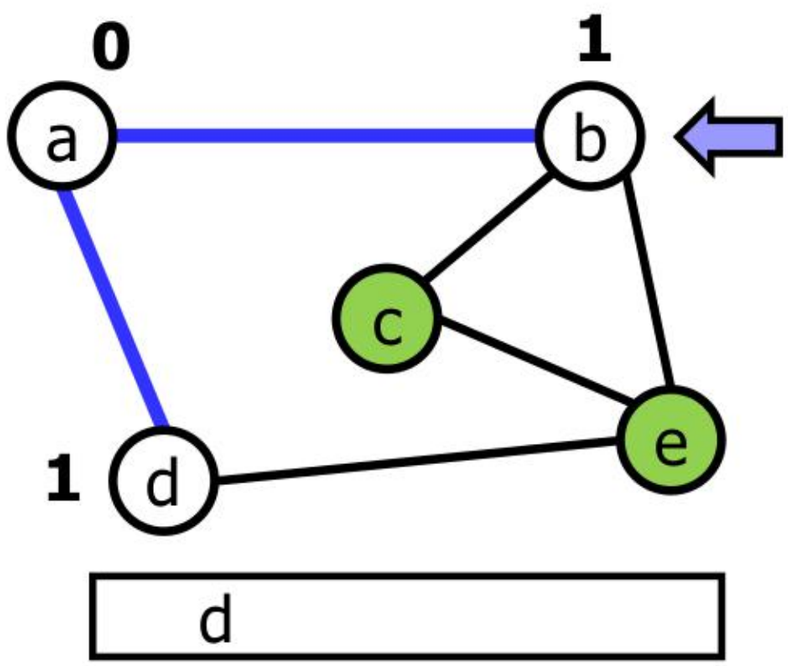
Průchod grafem do šířky (BFS)

- Breadth-first search



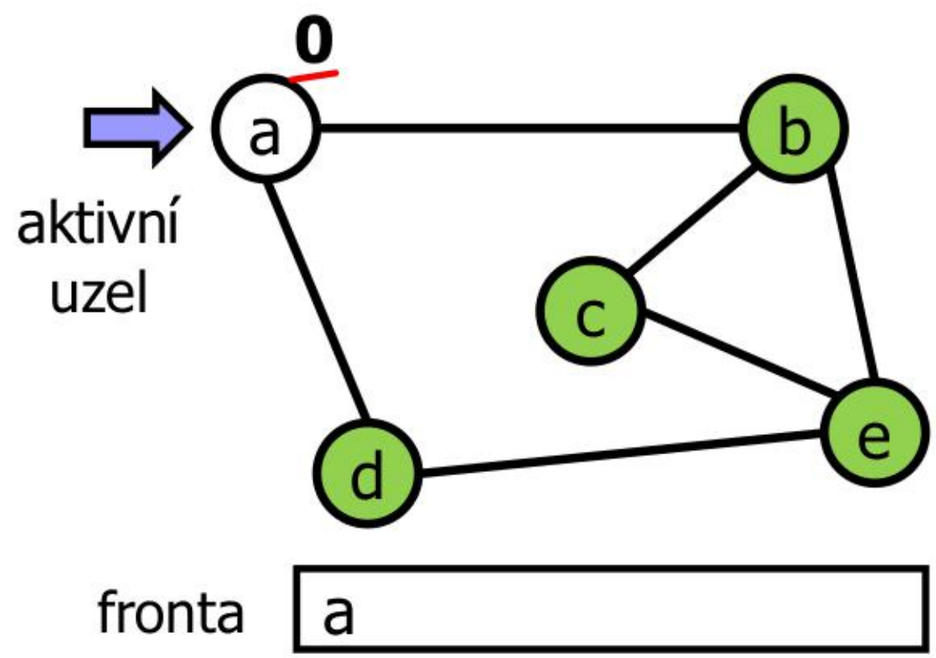


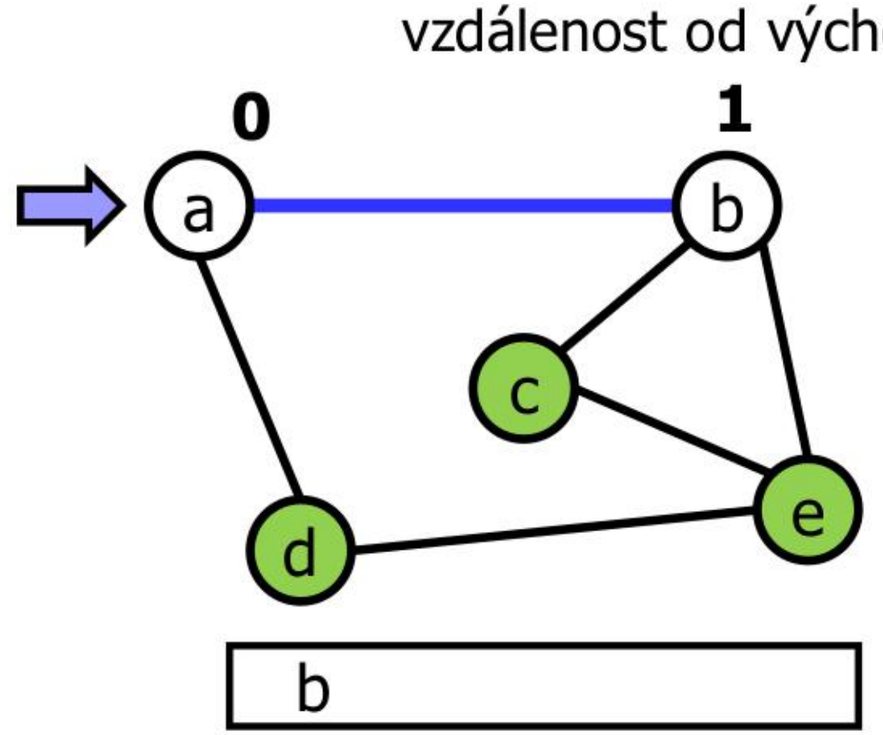


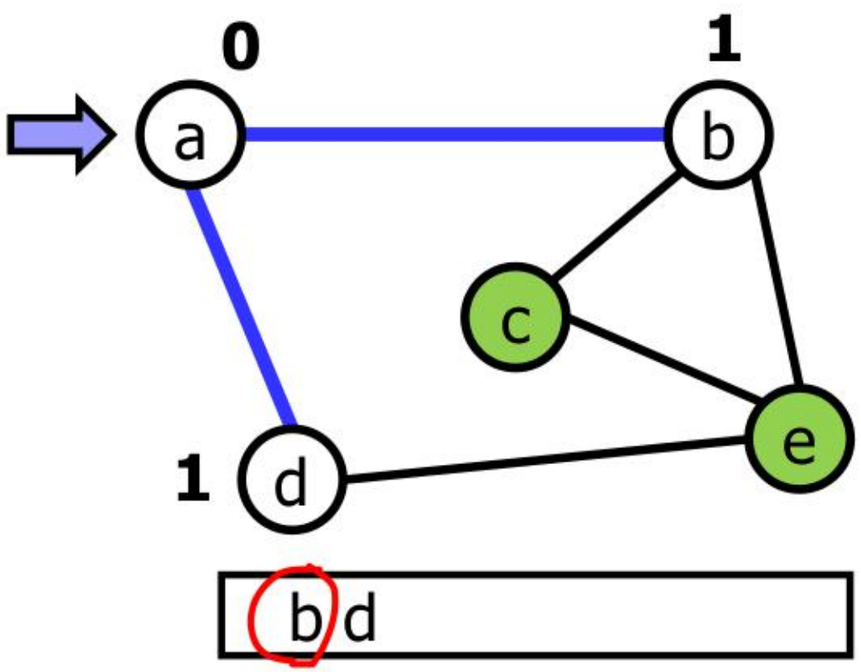


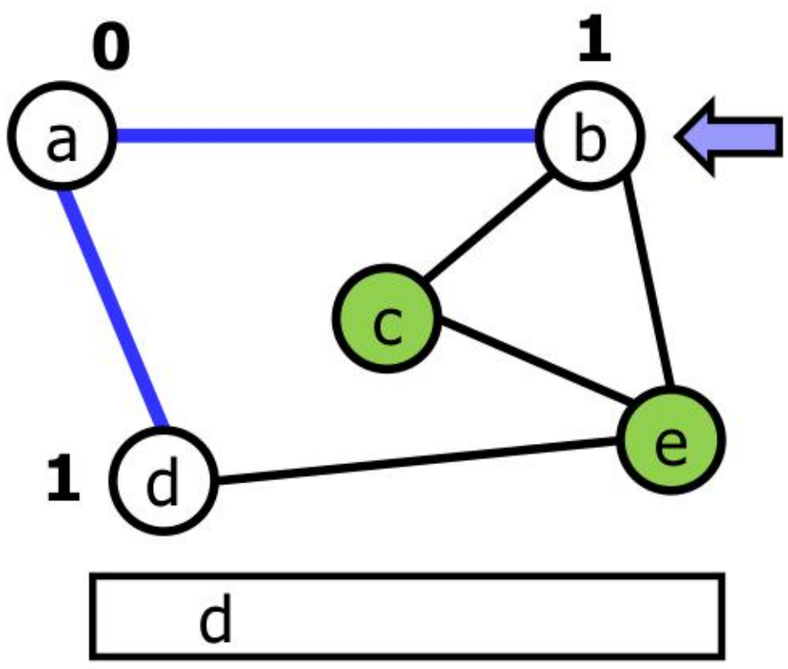
Průchod grafem do šířky (BFS)

- Breadth-first search



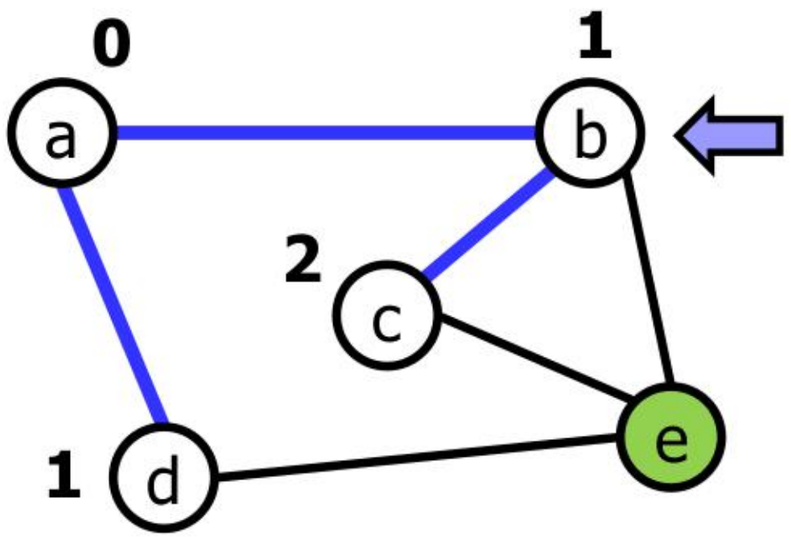


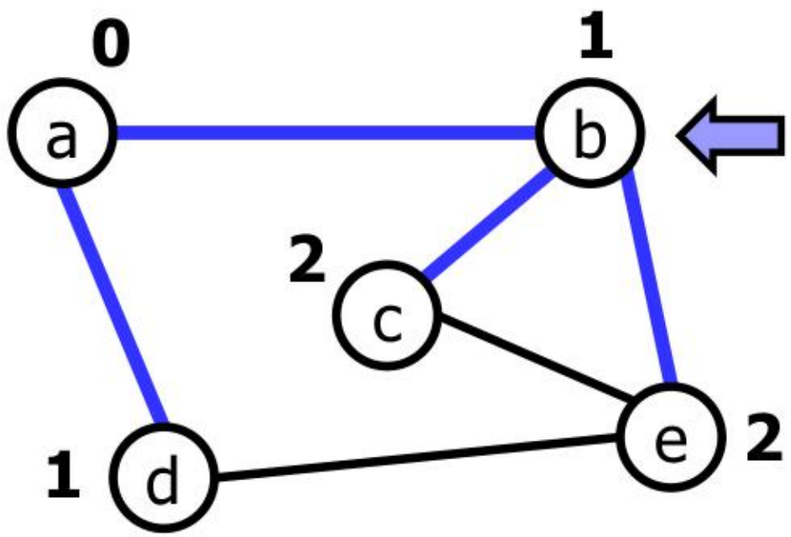




Průchod grafem do šířky (BFS)

- Breadth-first search

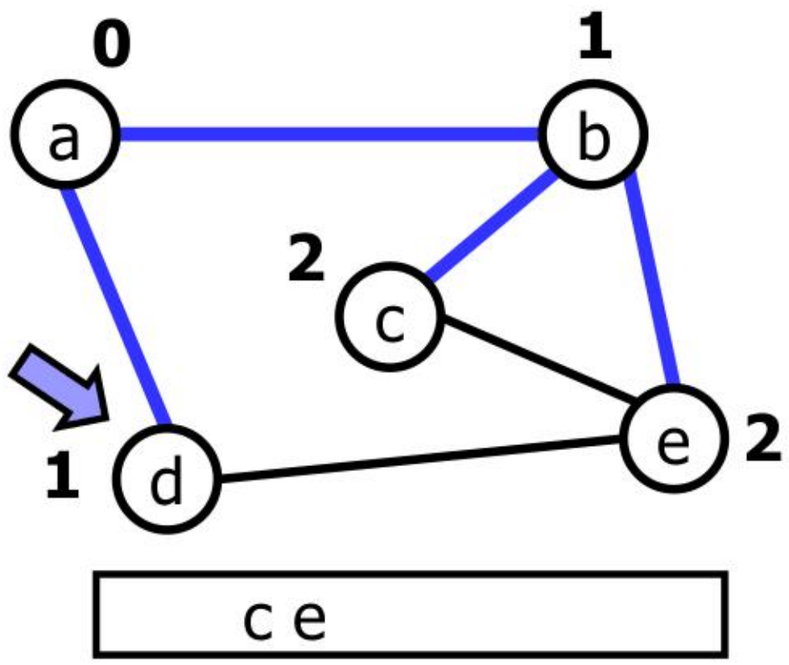


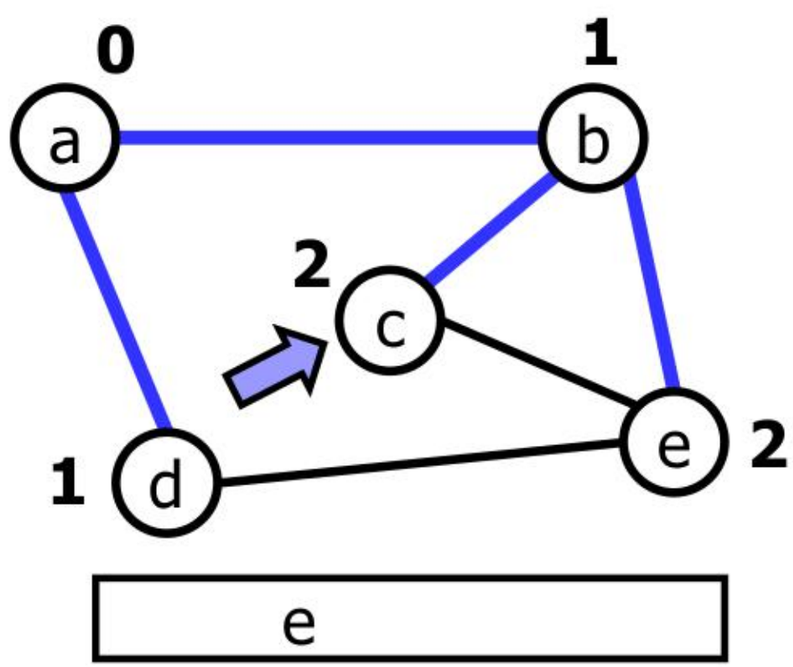


stavy uzlu:

FRESH

OPEN

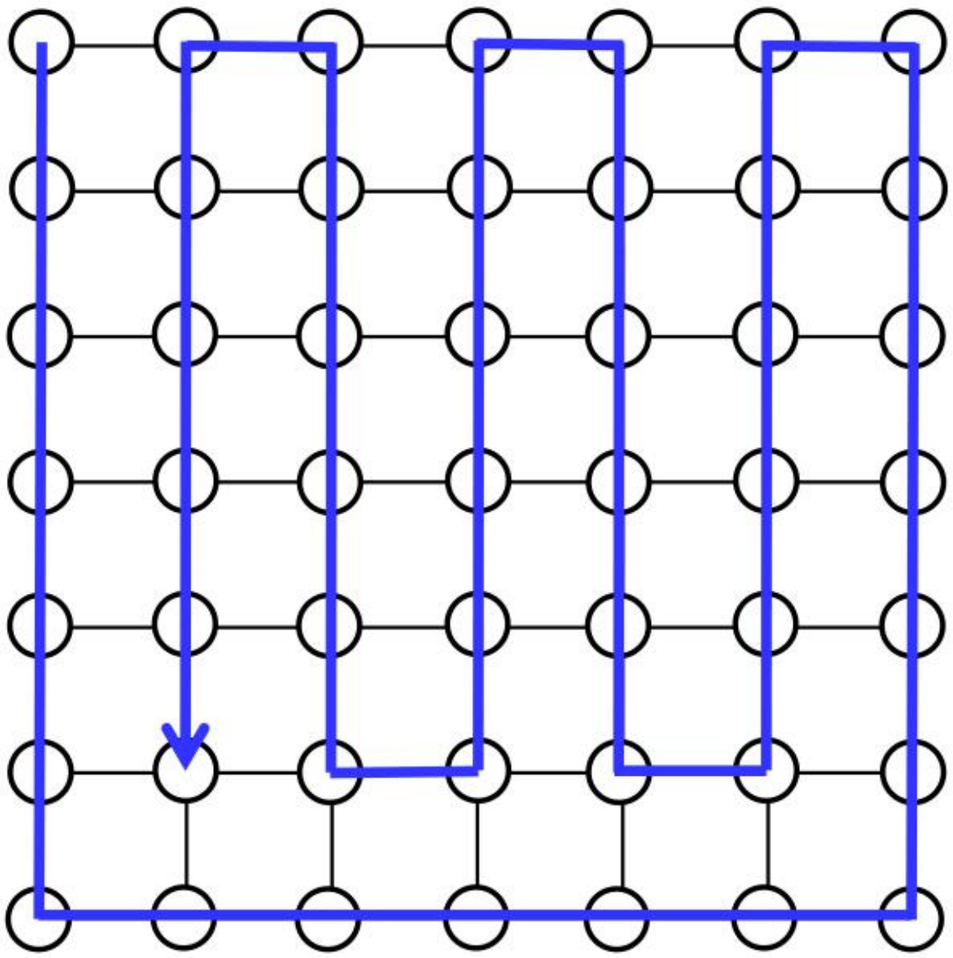




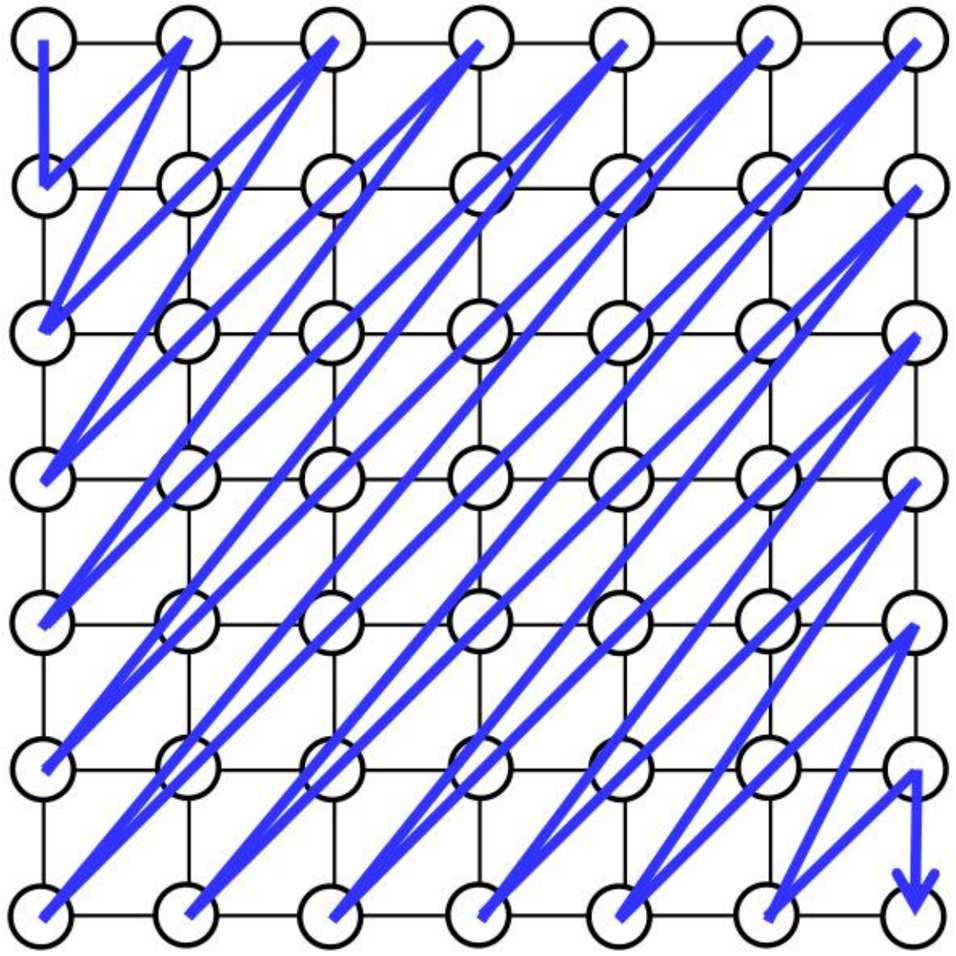
Porovnání DFS a BFS

Mřížka $N \times N$, uspořádání sousedů: ↓ → ↑ ←

DFS (rekurzivně)



BFS



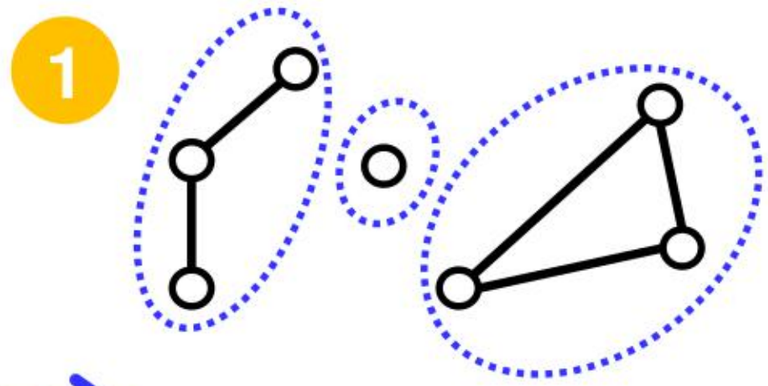
Potřebná velikost zásobníku / fronty:

$\Theta(N^2)$

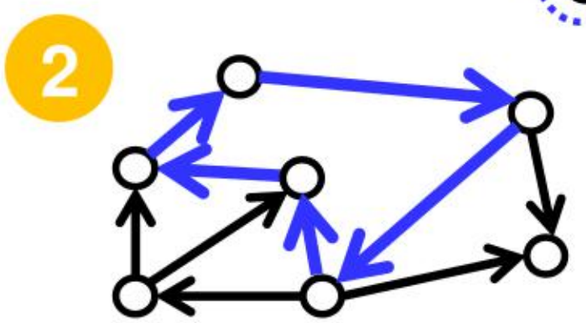
$\Theta(N)$

Aplikace průchodu grafem

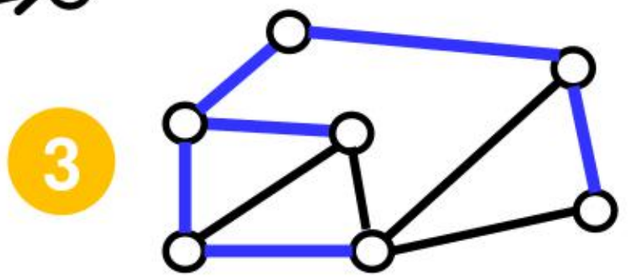
1. Detekce komponent souvislosti



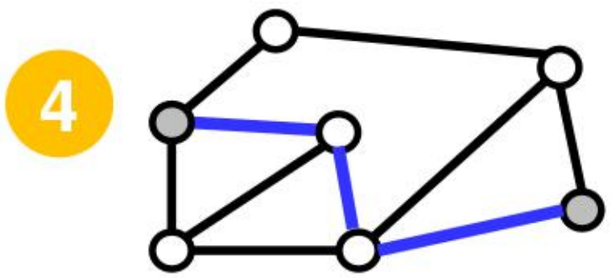
2. Detekce cyklu
   (pokud při DFS objevíme uzel ve stavu OPEN => cyklus)



3. Nalezení kostry

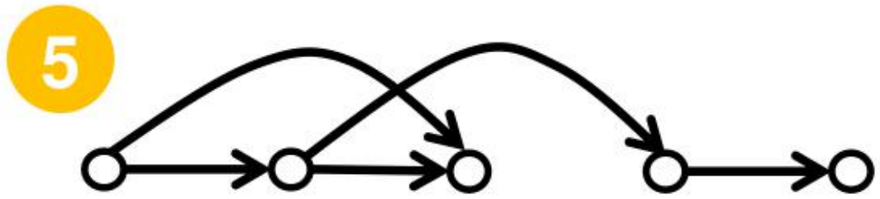


4. Hranově nejkratší cesta (jen BFS)



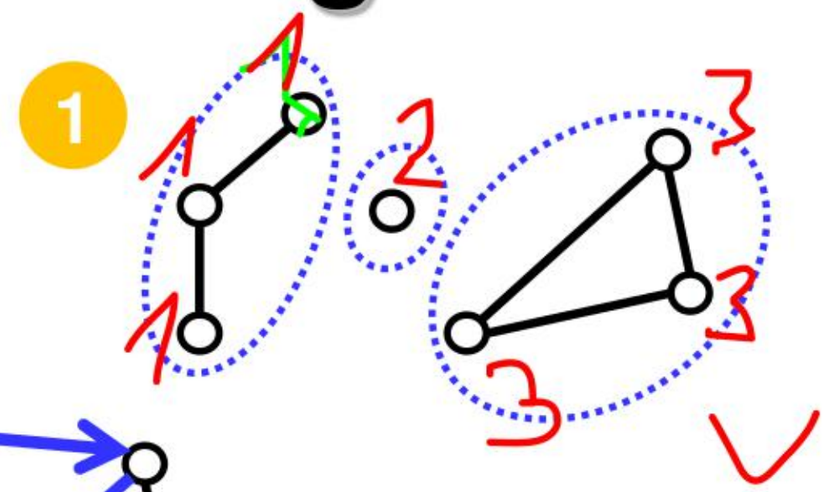
5. Topologické uspořádání (jen DFS)

   (uzly uspořádáme sestupně podle časů jejich uzavření)

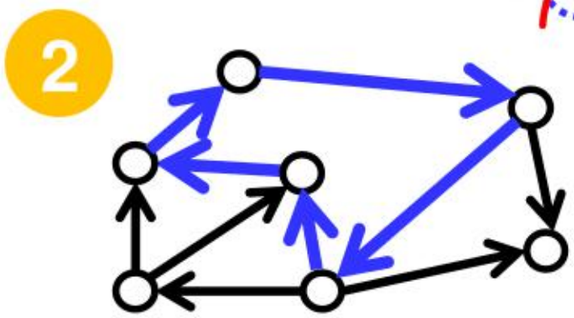


Aplikace průchodu grafem

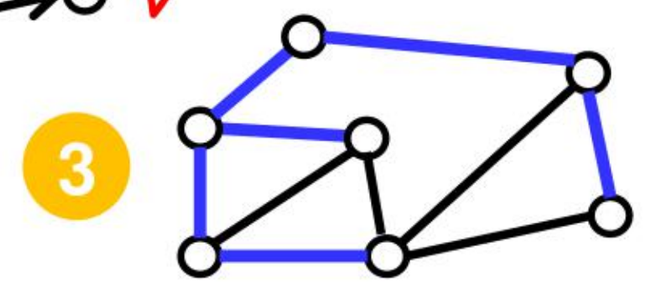
1. Detekce komponent souvislosti



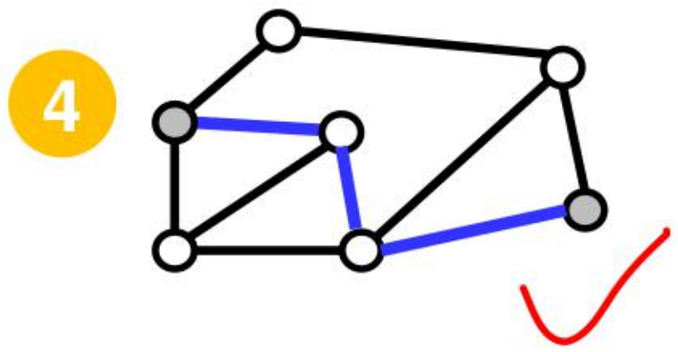
2. Detekce cyklu
   (pokud při DFS objevíme uzel ve stavu OPEN => cyklus)



3. Nalezení kostry

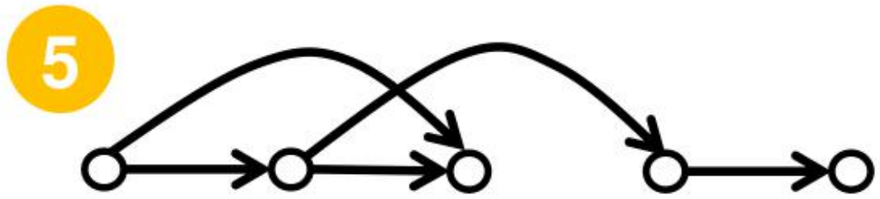


4. Hranově nejkratší cesta (jen BFS)



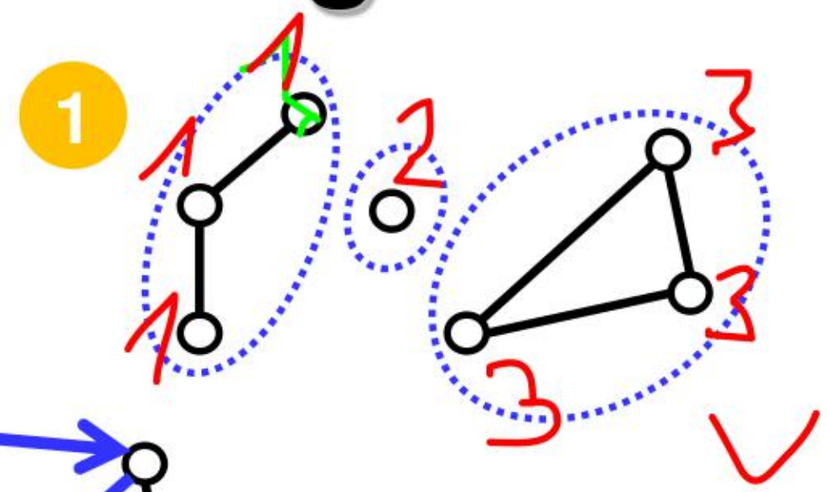
5. Topologické uspořádání (jen DFS)

   (uzly uspořádáme sestupně podle časů jejich uzavření)

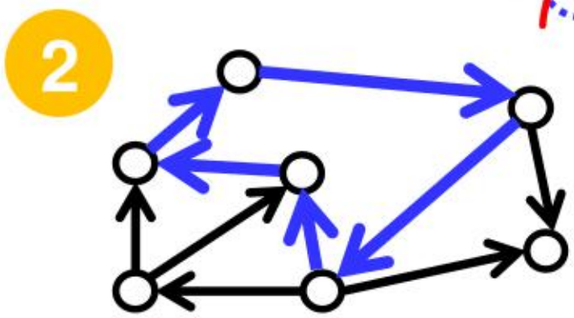


Aplikace průchodu grafem

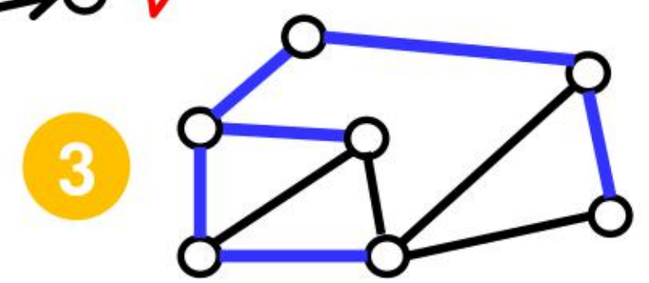
1. Detekce komponent souvislosti



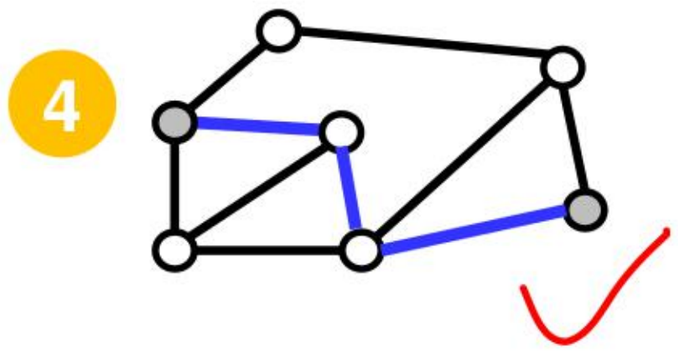
2. Detekce cyklu
   (pokud při DFS objevíme uzel ve stavu OPEN => cyklus)



3. Nalezení kostry

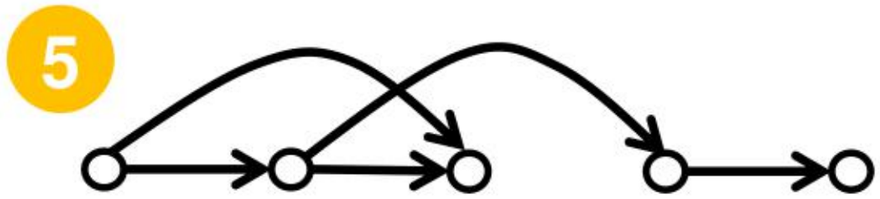


4. Hranově nejkratší cesta (jen BFS)



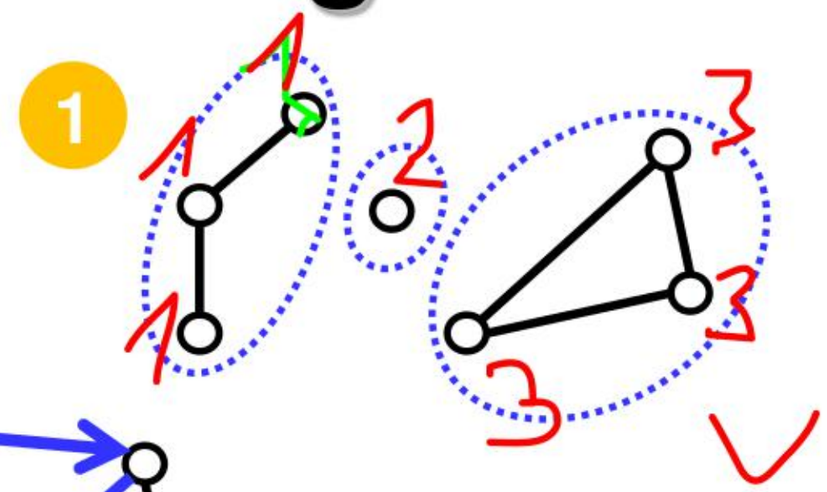
5. Topologické uspořádání (jen DFS)

   (uzly uspořádáme sestupně podle časů jejich uzavření)

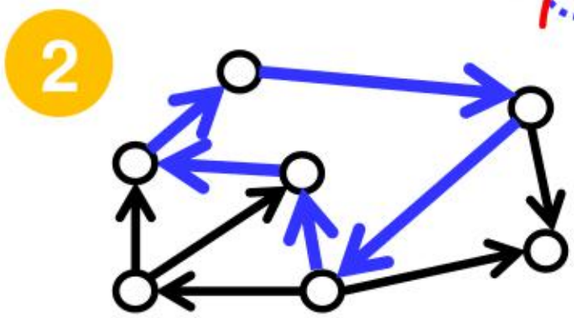


Aplikace průchodu grafem

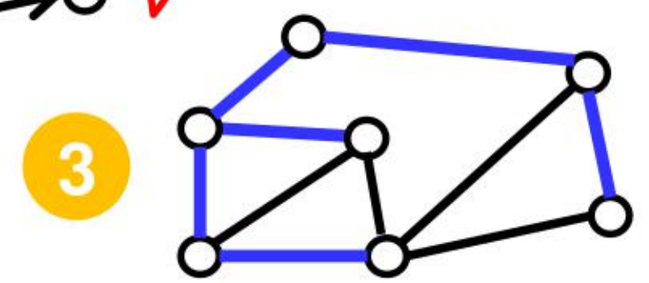
1. Detekce komponent souvislosti



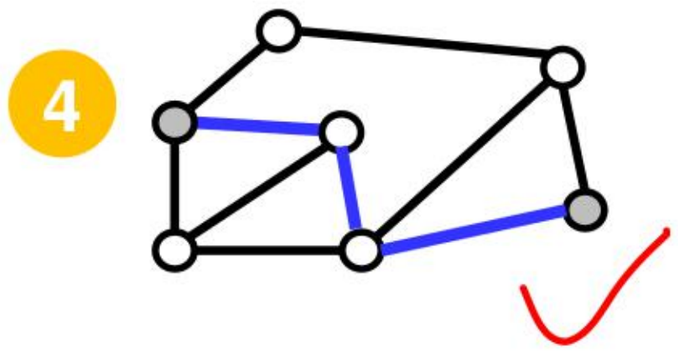
2. Detekce cyklu
   (pokud při DFS objevíme uzel ve stavu OPEN => cyklus)



3. Nalezení kostry

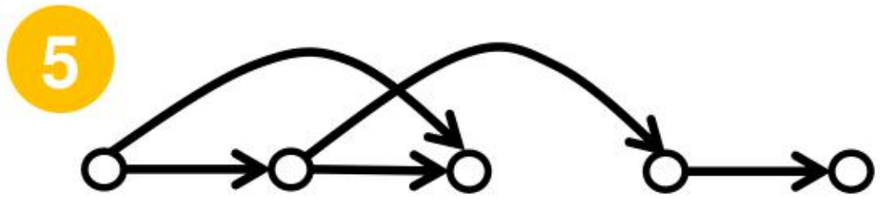


4. Hranově nejkratší cesta (jen BFS)



5. Topologické uspořádání (jen DFS)

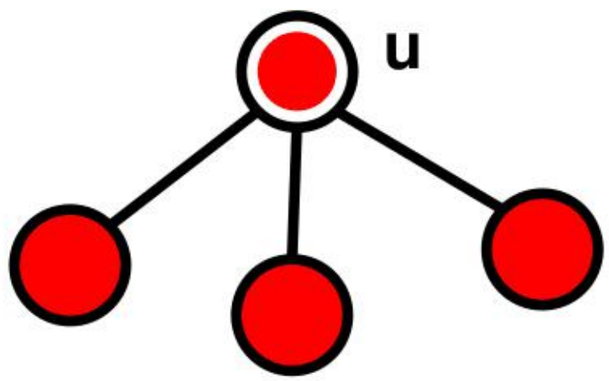
   (uzly uspořádáme sestupně podle časů jejich uzavření)



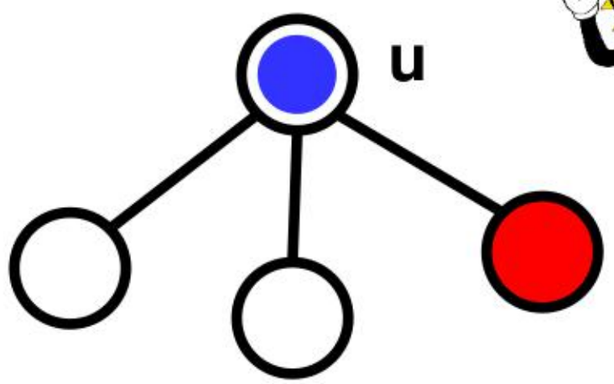
Druhá domácí úloha

Ořezávání: Jak stanovit horní odhad dosažitelného skóre?

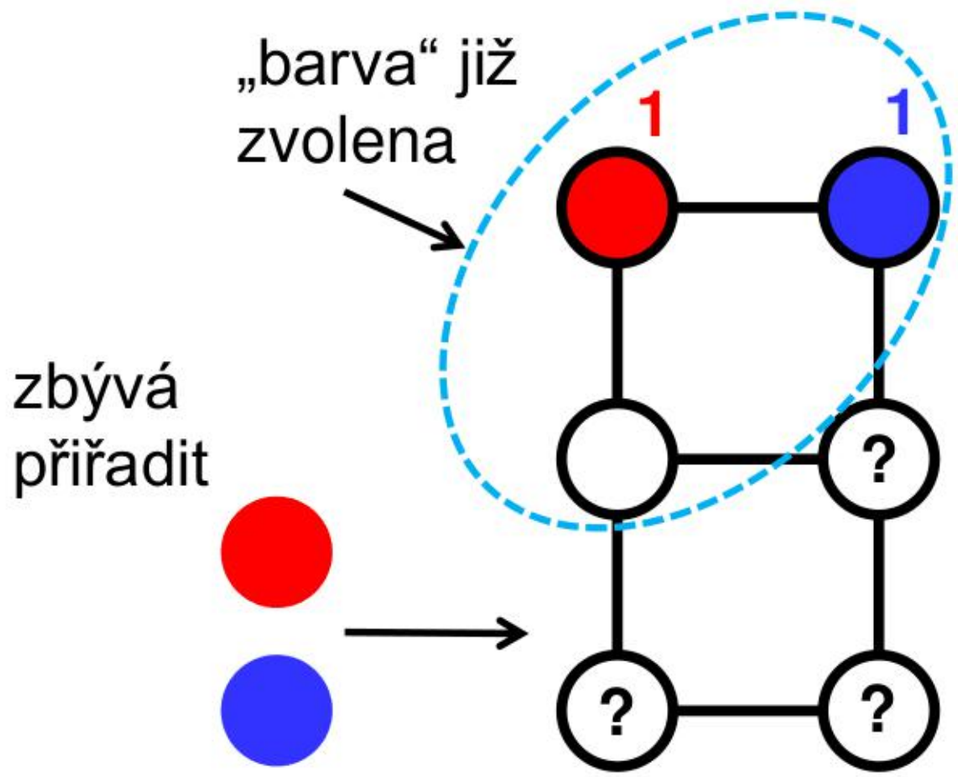
max. změna skóre po obarvení uzlu u na červeno je $2 \cdot \mathrm{d}_u$



max. změna skóre po obarvení uzlu u na modro je d_u



d_u .. stupeň uzlu u

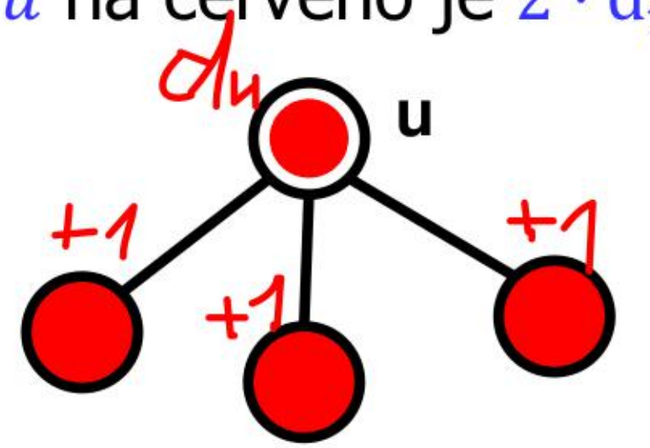


Heuristika: Stupně uzlů lze využít i pro určení vhodného pořadí pro backtracking.

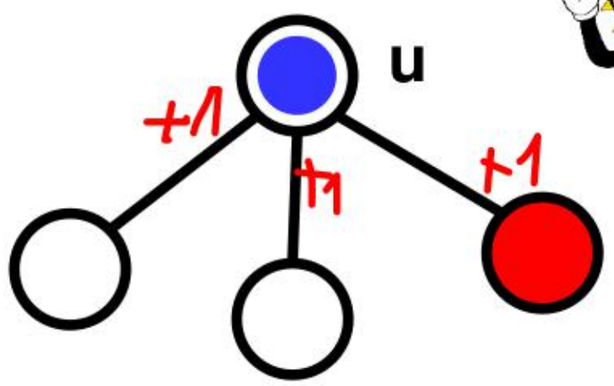
Druhá domácí úloha

Ořezávání: Jak stanovit horní odhad dosažitelného skóre?

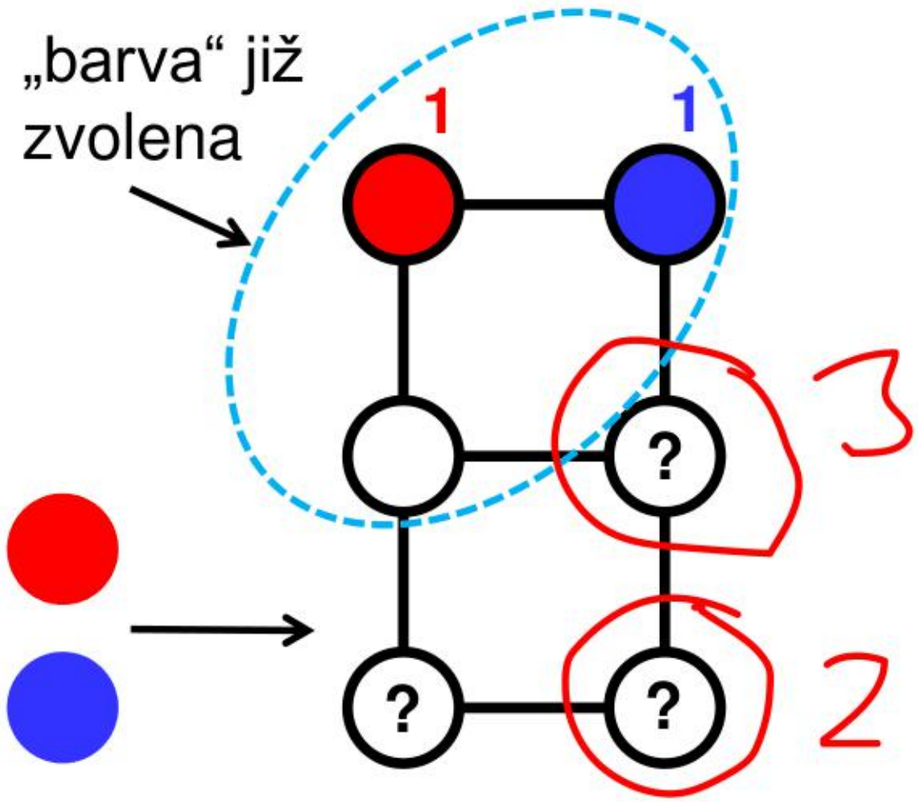
max. změna skóre po obarvení uzlu u na červeno je $2 \cdot \mathrm{d}_u$



max. změna skóre po obarvení uzlu u na modro je d_u



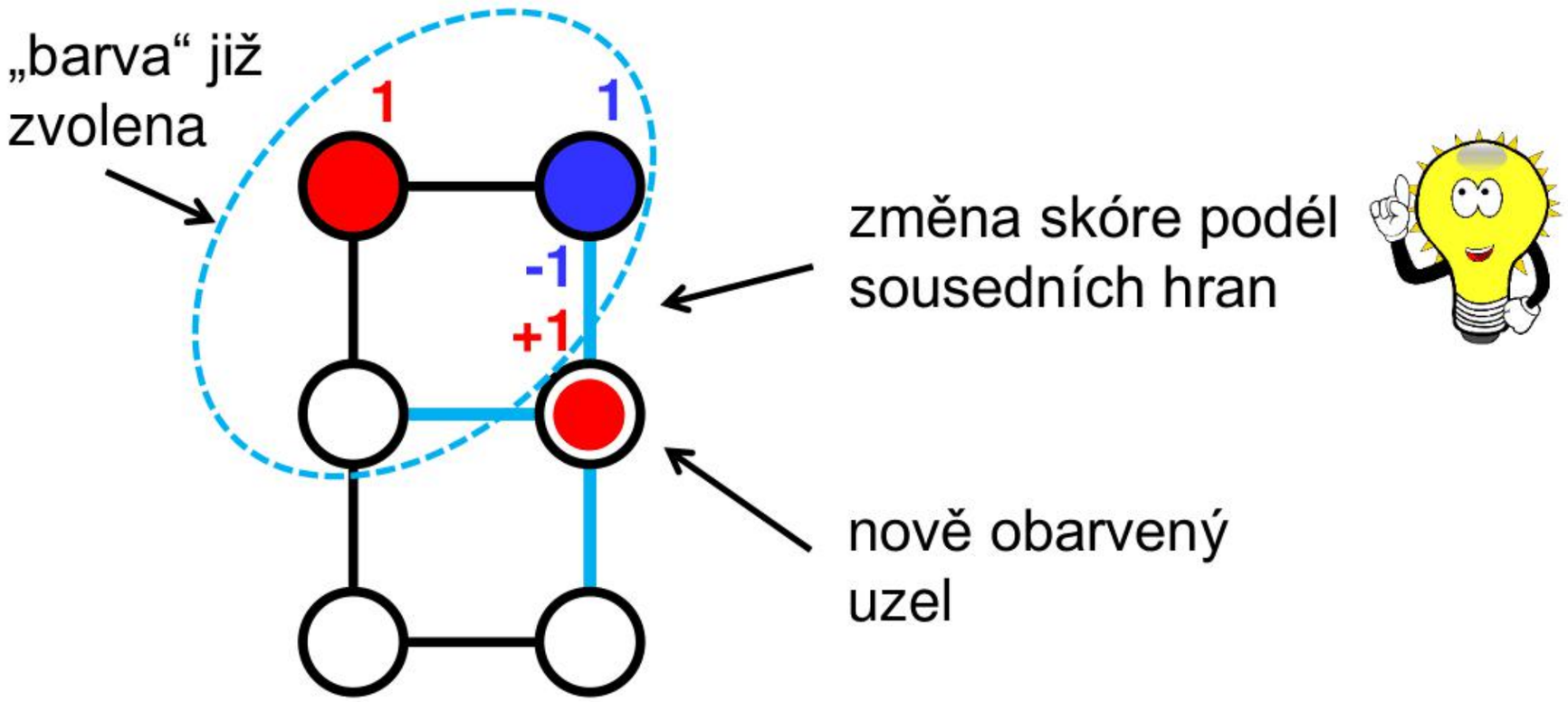
d_u .. stupeň uzlu u



Heuristika: Stupně uzlů lze využít i pro určení vhodného pořadí pro backtracking.

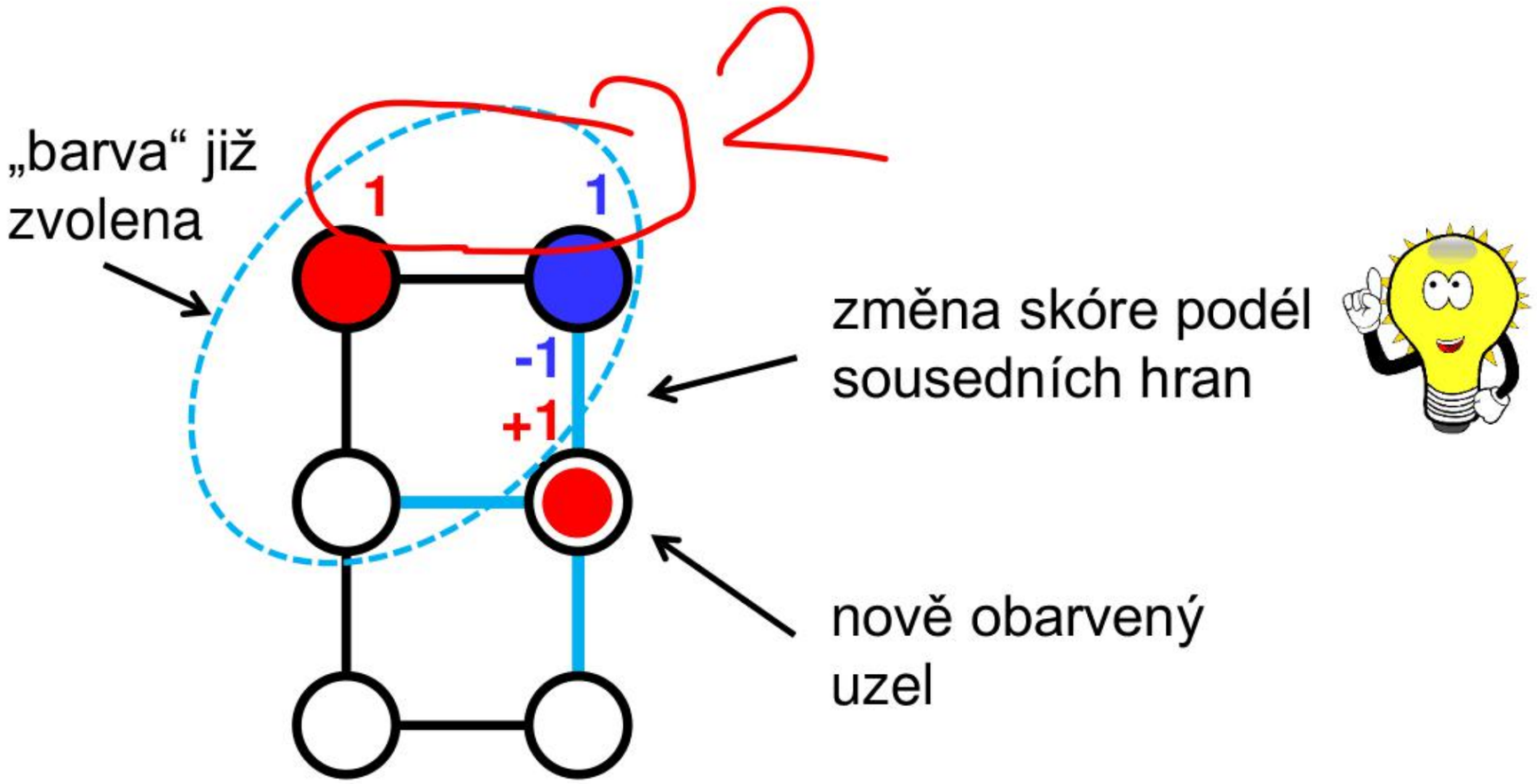
Druhá domácí úloha

Výpočet skóre inkrementálně (ne pokaždé znovu).



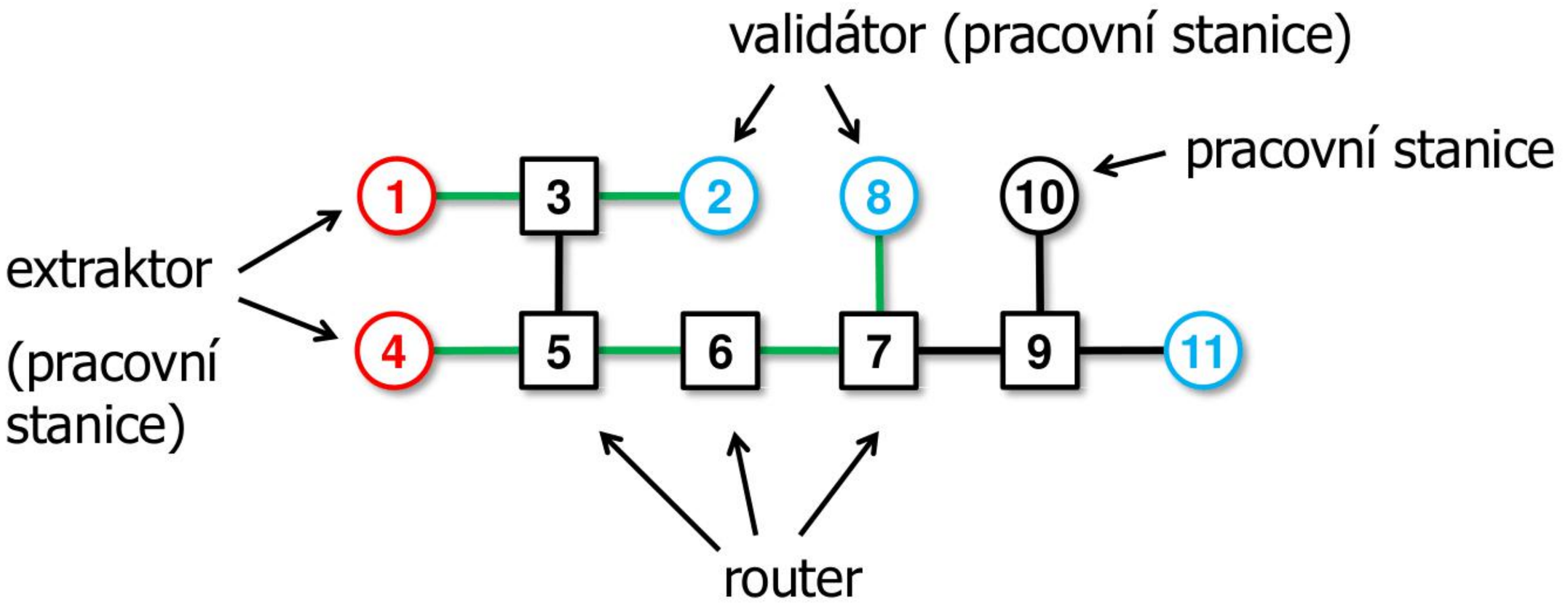
Druhá domácí úloha

Výpočet skóre inkrementálně (ne pokaždé znovu).



Třetí domácí úloha

Univerzitní počítačová síť, stromová topologie



Cíl: Propojit maximální možný počet dvojic extraktor-validátor, s tím, že odlišné dvojice nesmí sdílet žádný router.