



Algoritmizace

Marko Genyg-Berezovskyj, Daniel Průša

2010 – 2021

Dnešní témata

- Průchod stromem do šířky
- Průchod grafem do hloubky a do šířky
- Domácí úlohy
 - Rady k HW_02
 - Zadání HW_03

slido



**Join at slido.com
#065475**

① Start presenting to display the joining instructions on this slide.

slido



**Otázky, komentáře,
povzdechy...**

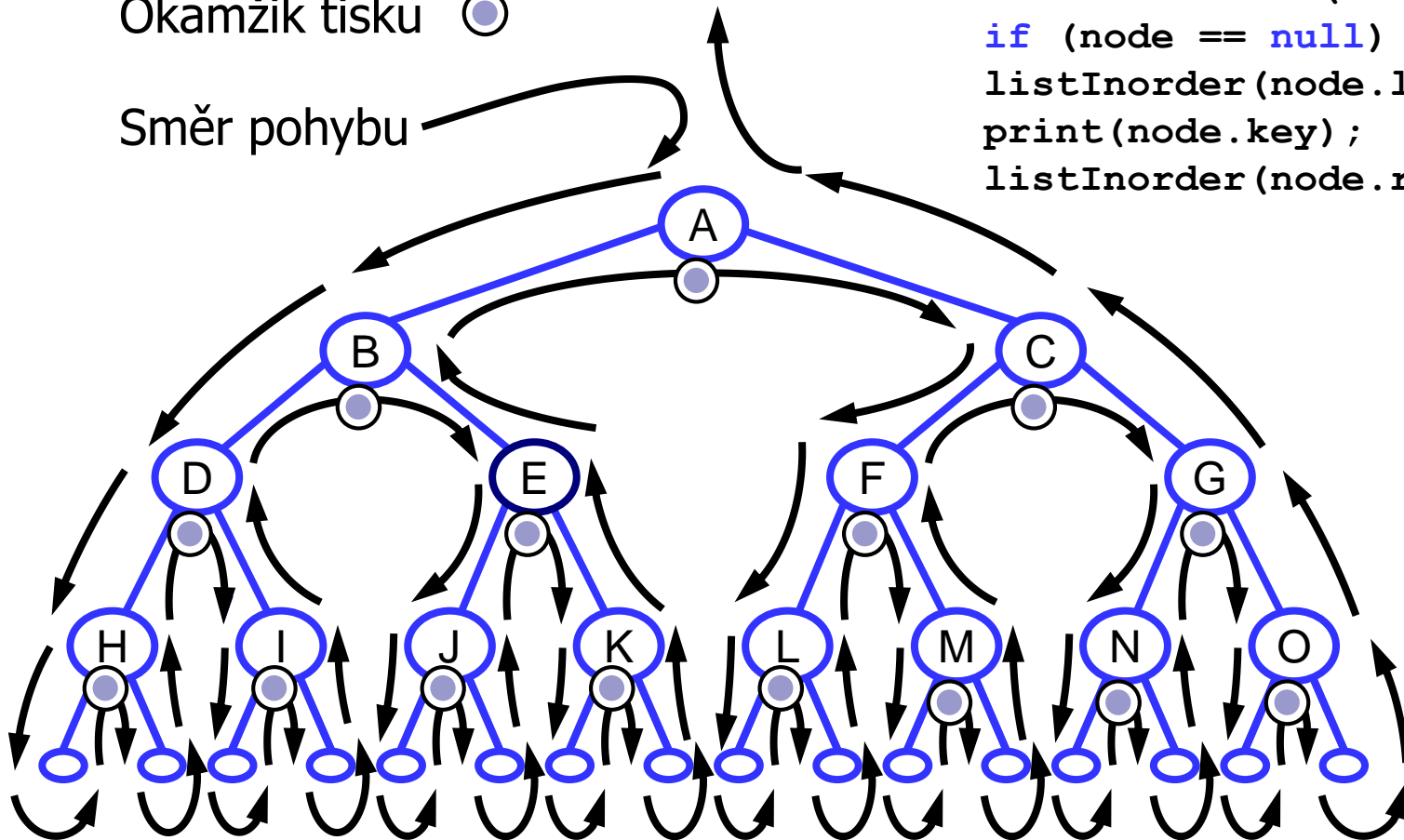
ⓘ Start presenting to display the poll results on this slide.

Z minula: průchod stromem do hloubky

Okamžik tisku ○

Směr pohybu

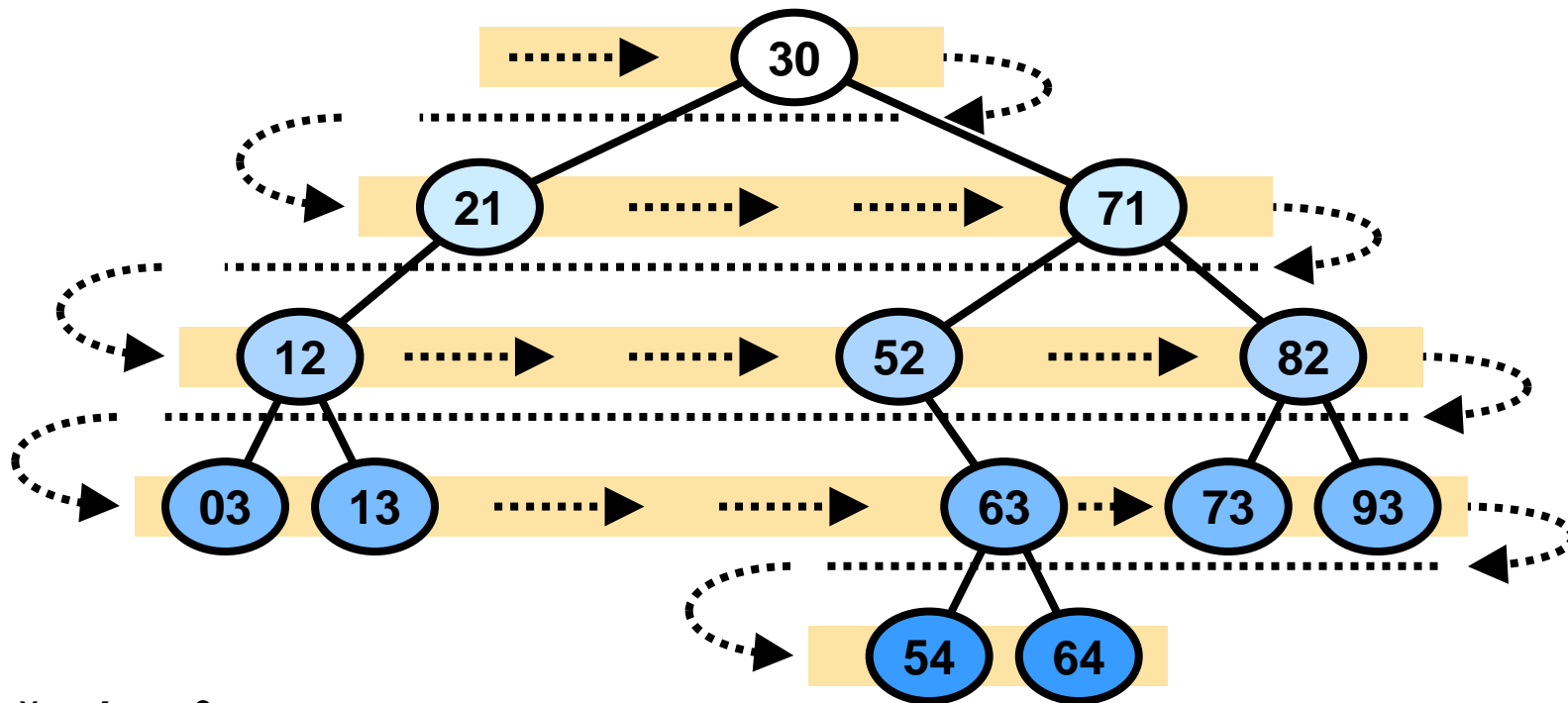
```
void listInorder(Node node) :  
    if (node == null) return;  
    listInorder(node.left);  
    print(node.key);  
    listInorder(node.right);
```



Výstup

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

Průchod stromem do šířky

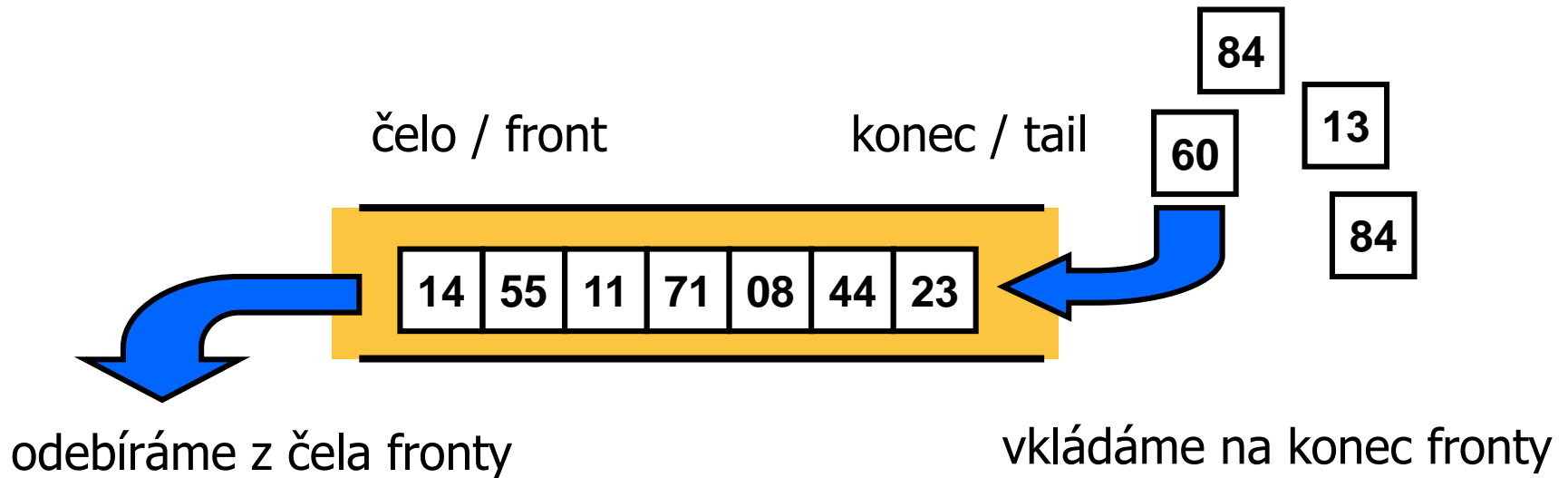


Pořadí uzlů

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

Struktura stromu ani rekurzivní přístup tento průchod nepodporují.

Fronta



`java.util.LinkedList`

```
addLast(element)
removeFirst()
getFirst()
isEmpty()
```

`queue` (C++ Standard Template Library)

```
queue::push(element)
queue::pop()
queue::front()
queue::empty()
```

Cyklická implementace fronty polem



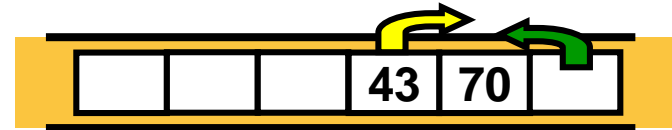
Prázdná fronta
v poli pevné délky



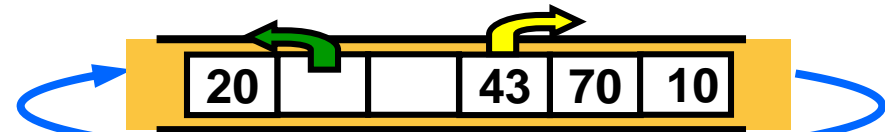
Vlož 24, 11, 90, 43, 70.



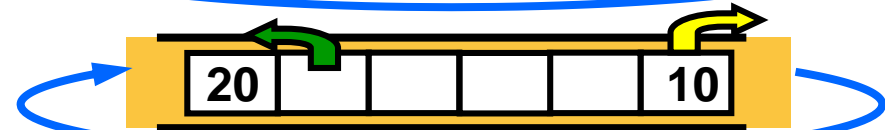
Odeber, odeber, odeber.



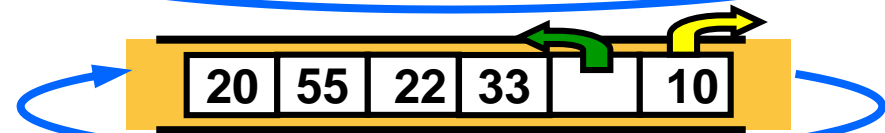
Vlož 10, 20.



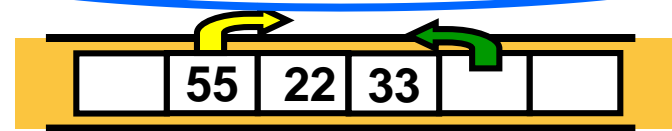
Odeber, odeber.



Vlož 55, 22, 33.



Odeber, odeber.



Průchod stromem do šířky

Inicializace:

Vytvoř prázdnou frontu

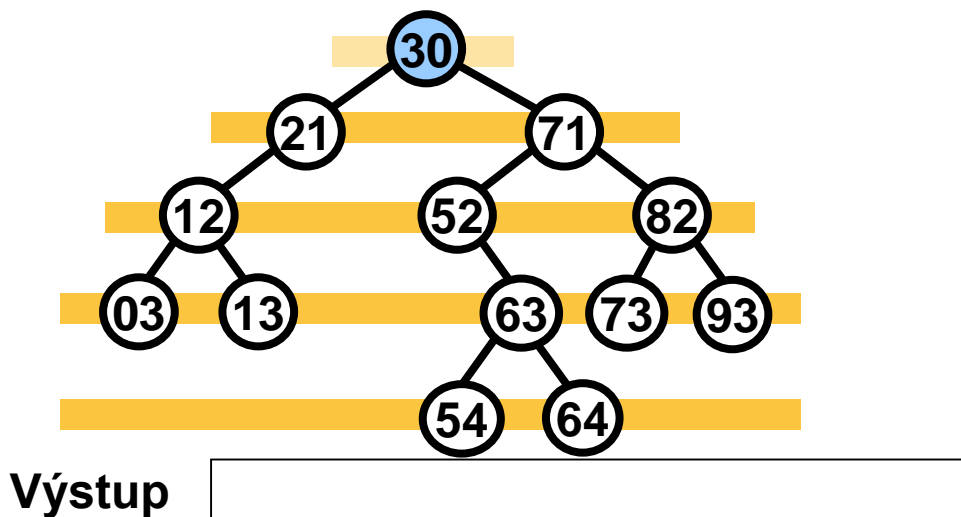


Do fronty vlož kořen stromu



Čelo

Konec

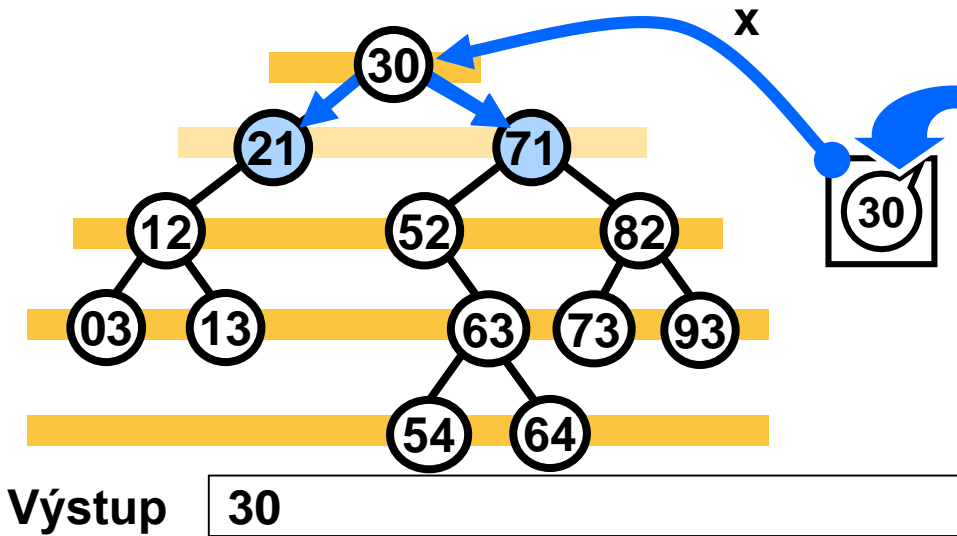


Hlavní cyklus

Dokud není fronta prázdná, opakuj:

1. Odeber první uzel z fronty a zpracuj ho.
2. Do fronty vlož jeho potomky, pokud existují.

Průchod stromem do šířky

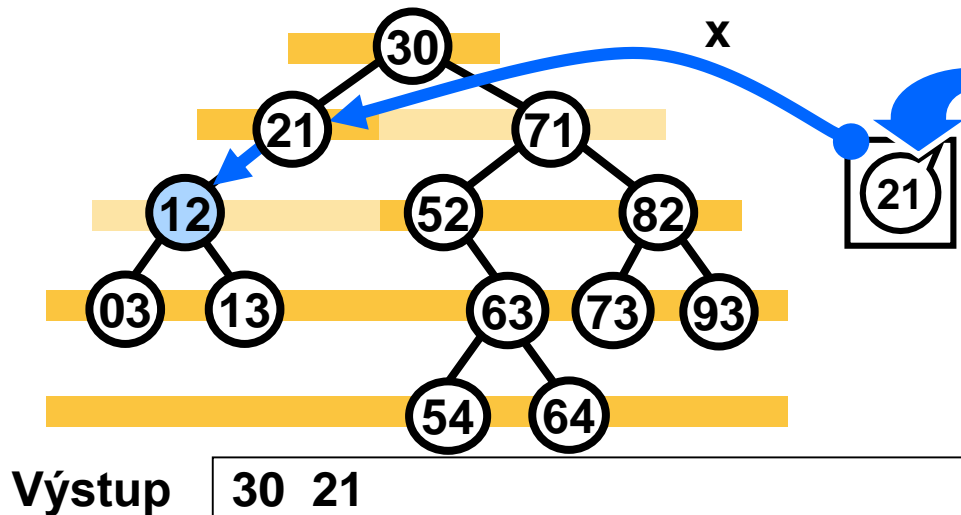


1. $x = \text{Odeber}()$, $\text{tisk}(x.\text{key})$.

2. $\text{Vlož}(x.\text{left})$, $\text{vlož}(x.\text{right})$. *)



*) pokud existuje

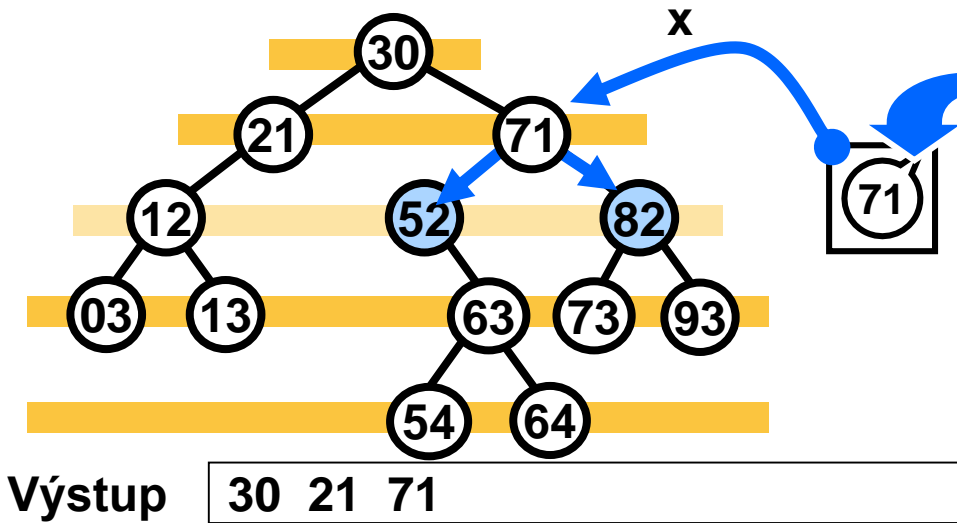


1. $x = \text{Odeber}()$, $\text{tisk}(x.\text{key})$.

2. $\text{Vlož}(x.\text{left})$, $\text{vlož}(x.\text{right})$. *)

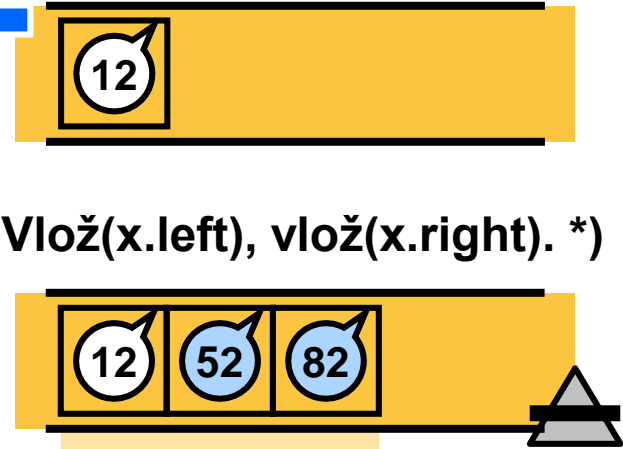


Průchod stromem do šířky



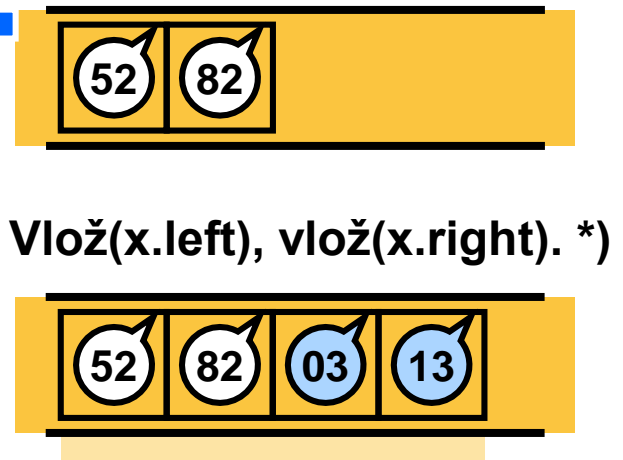
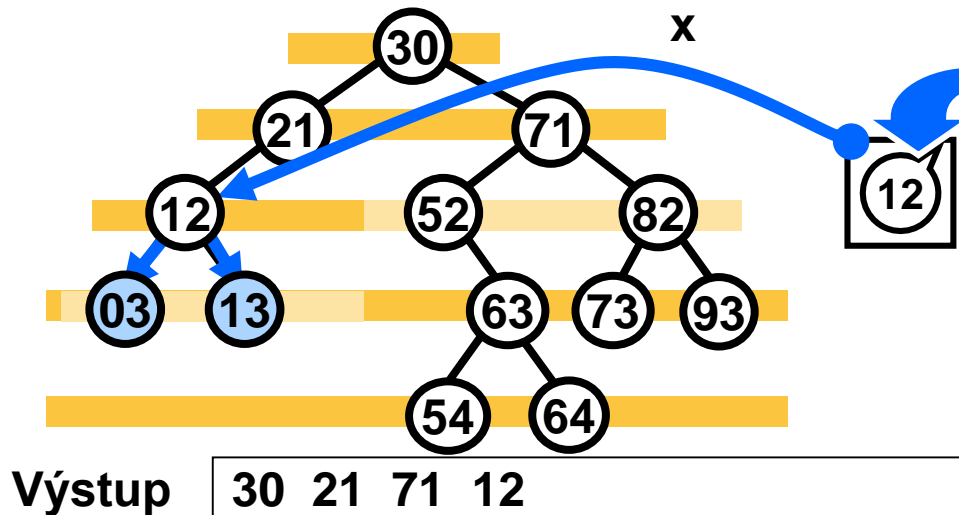
1. $x = \text{Odeber}()$, $\text{tisk}(x.\text{key})$.

2. Vlož($x.\text{left}$), vlož($x.\text{right}$). *)



1. $x = \text{Odeber}()$, $\text{tisk}(x.\text{key})$.

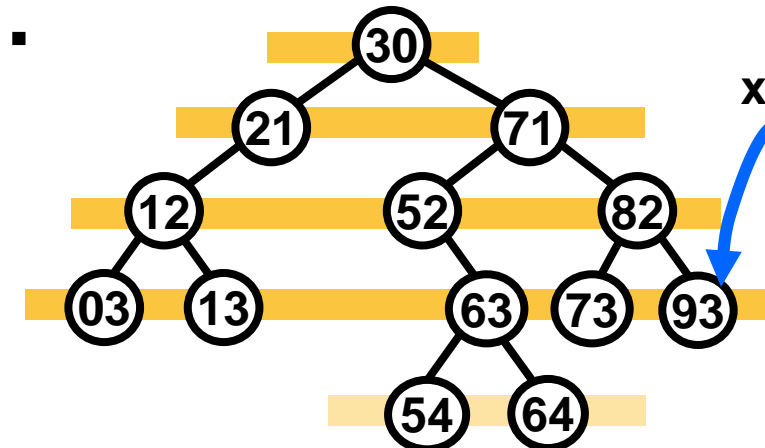
2. Vlož($x.\text{left}$), vlož($x.\text{right}$). *)



*) pokud existuje

Průchod stromem do šířky

...



1. $x = \text{Odeber}()$, $\text{tisk}(x.\text{key})$.

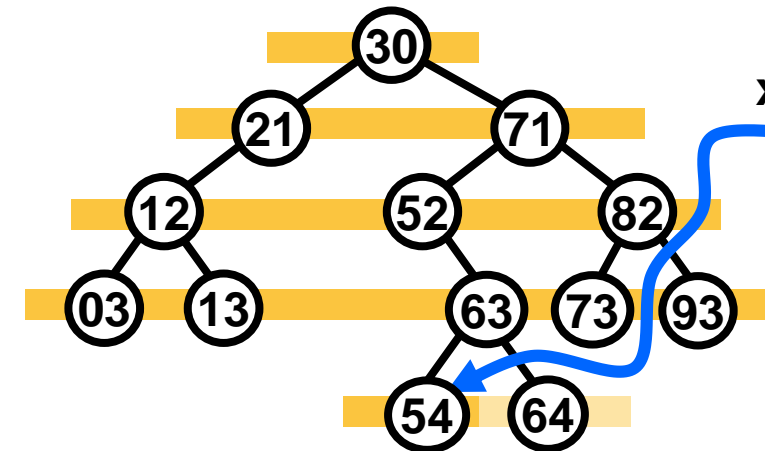


2. Vlož($x.\text{left}$), vlož($x.\text{right}$). *)



Výstup 30 21 71 12 52 82 03 13 63 73 93

*) pokud existuje



1. $x = \text{Odeber}()$, $\text{tisk}(x.\text{key})$.

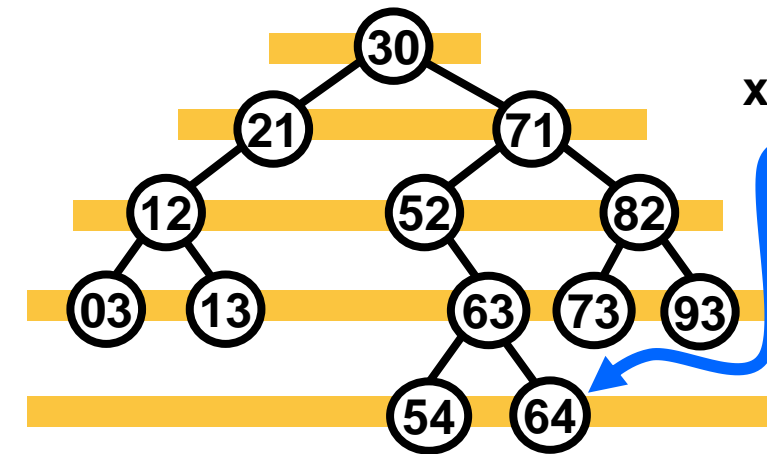


2. Vlož($x.\text{left}$), vlož($x.\text{right}$). *)



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

Průchod stromem do šířky



1. $x = \text{Odeber}(), \text{tisk}(x.\text{key}).$

2. $\text{Vlož}(x.\text{left}), \text{vlož}(x.\text{right}).$ *)

*) pokud existuje

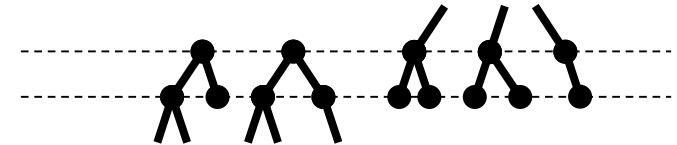
Výstup 30 21 71 12 52 82 03 13 63 73 93 54 64

Fronta je prázdná,
průchod stromem končí.

V neprázdné frontě jsou vždy právě

-- některé (třeba všechny) uzly jednoho patra

-- a všichni potomci těch uzlů tohoto patra, které už nejsou ve frontě.



Někdy jsou ve frontě přesně všechny uzly jednoho patra. Viz výše.



Průchod stromem do šířky

```
void listBreadth (Node node) {
    if (node == null) return;
    Queue q = new Queue();           // create queue
    q.push(node);                     // init queue
    while (!q.empty()) {
        node = q.pop();
        print(node.key);              // process node
        if (node.left != null) q.push(node.left);
        if (node.right != null) q.push(node.right);
    }
}
```



Jakou má průchod stromem do šířky časovou složitost?



Otázky, komentáře, povzdechy...

ⓘ Start presenting to display the poll results on this slide.

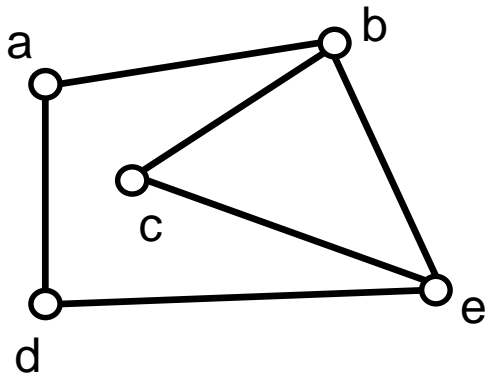
Průchod grafem

Průchody stromem zobecníme pro orientované i neorientované grafy.

$$G_1 = (V, E_1)$$

$$V = \{a, b, c, d, e\}$$

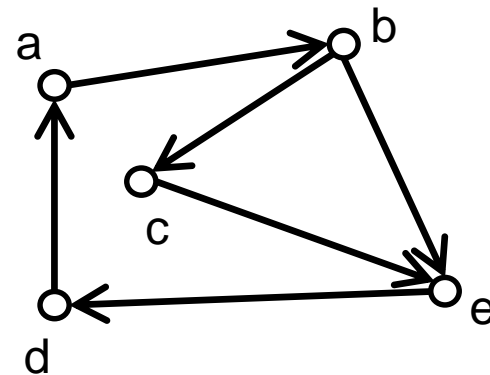
$$E_1 = \left\{ \begin{array}{l} \{a, b\}, \{b, c\}, \{b, e\}, \\ \{e, c\}, \{d, e\}, \{a, d\} \end{array} \right\}$$



$$G_2 = (V, E_2)$$

$$V = \{a, b, c, d, e\}$$

$$E_2 = \left\{ \begin{array}{l} (a, b), (b, c), (b, e), \\ (c, e), (e, d), (d, a) \end{array} \right\}$$



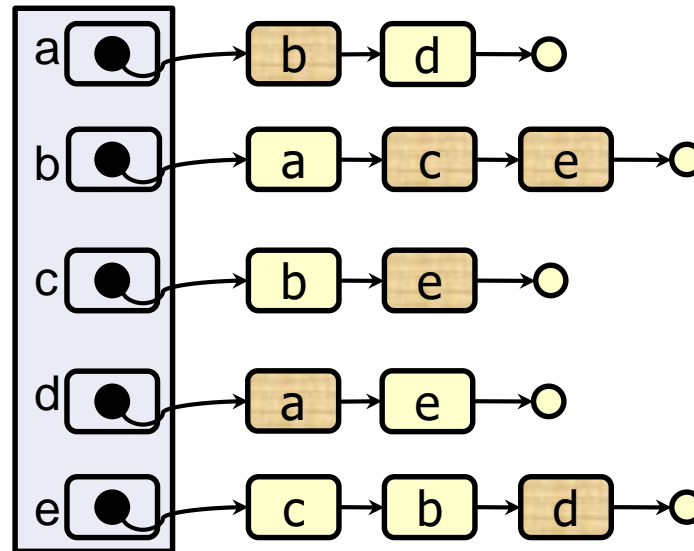
Použití: hledání komponent souvislosti, cyklů, hranově nejkratší cesty, ...

Reprezentace grafu v paměti

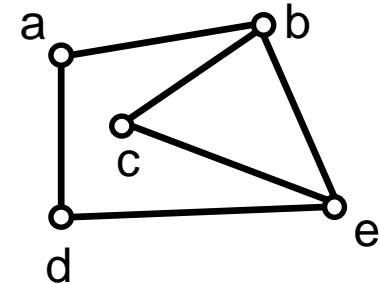
Matice sousednosti:

	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

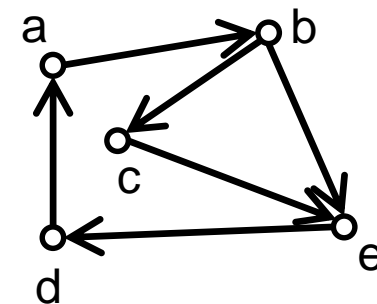
Seznam sousedů:



$G_1 = (V, E_1)$



$G_2 = (V, E_2)$




Prostorová složitost

$$\Theta(|V|^2)$$

test existence hrany
v konstantním čase

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

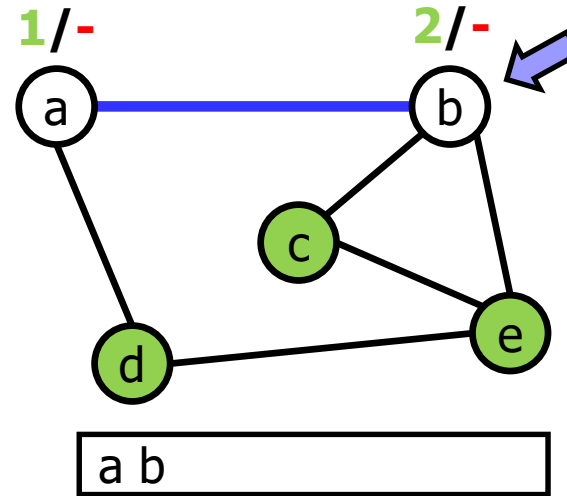
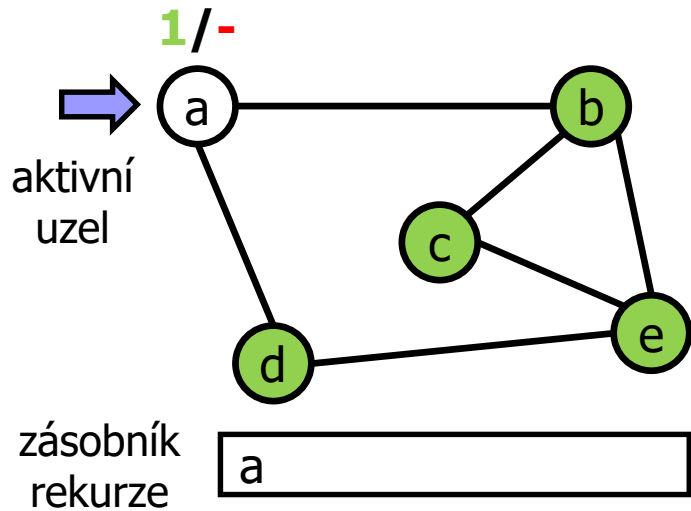
výhodnější pro řídké grafy

Reprezentace grafu G_2
obsahuje pouze prvky
s pozadím 

Průchod grafem do hloubky (DFS)

Depth-first search

čas otevření / uzavření uzlu



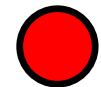
stavy uzlu:



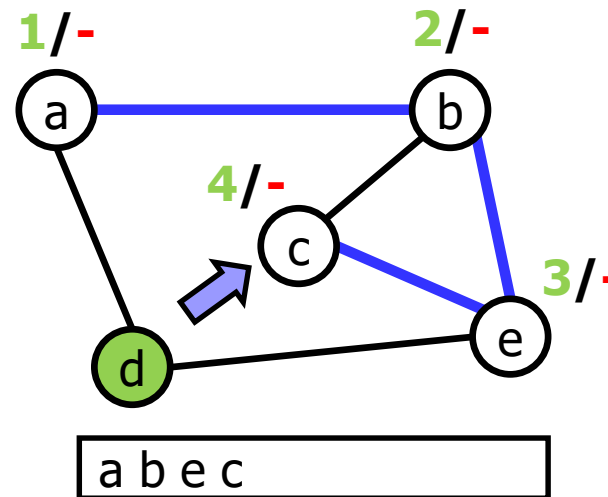
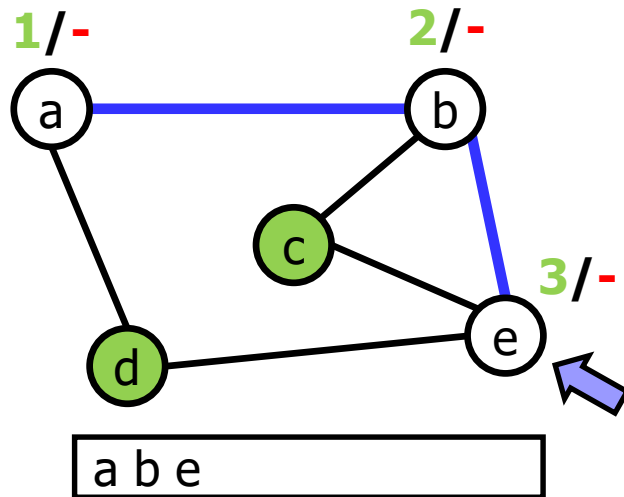
FRESH



OPEN

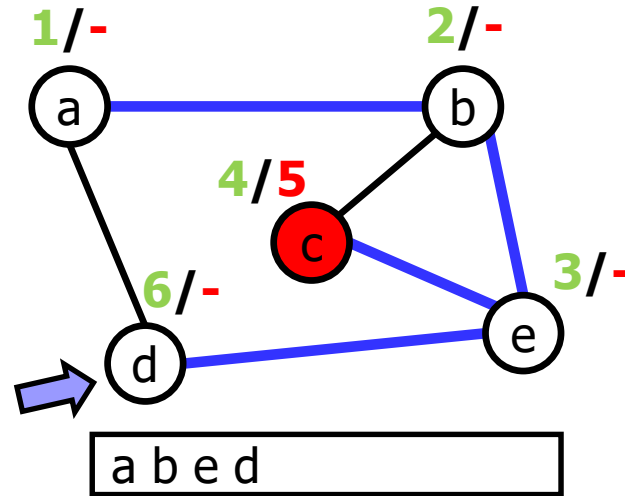
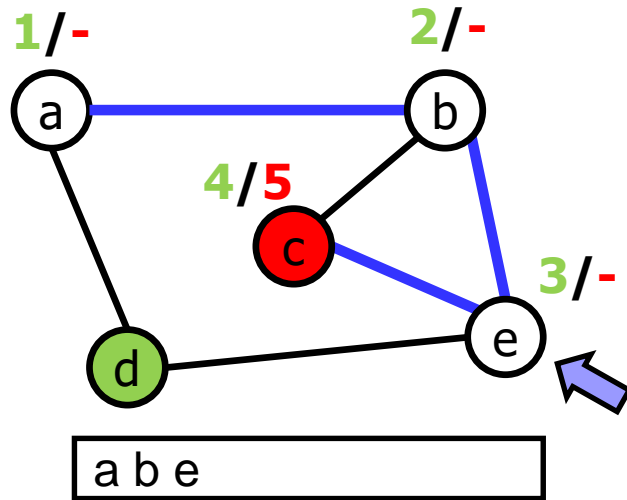


CLOSED



Průchod grafem do hloubky (DFS)

Depth-first search



stavy uzlu:



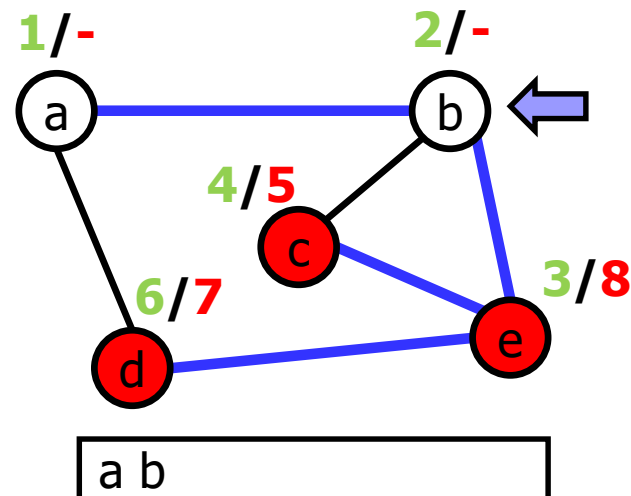
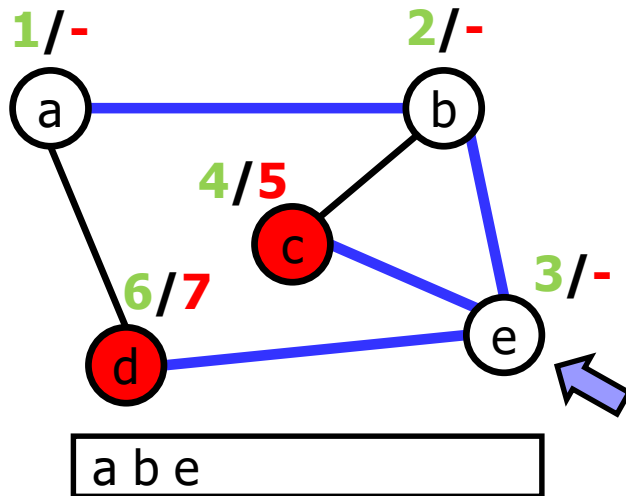
FRESH



OPEN

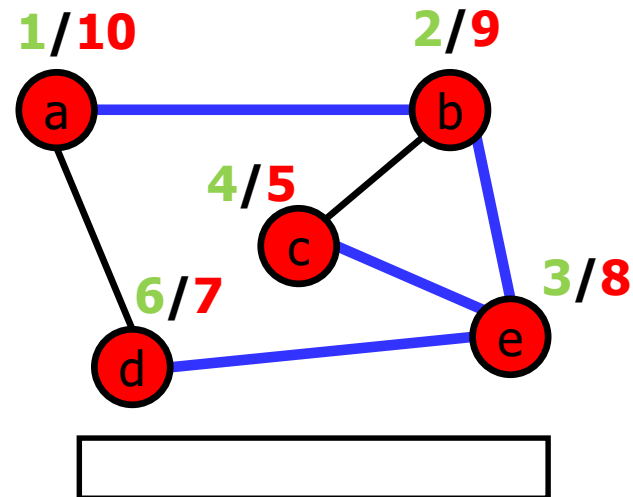
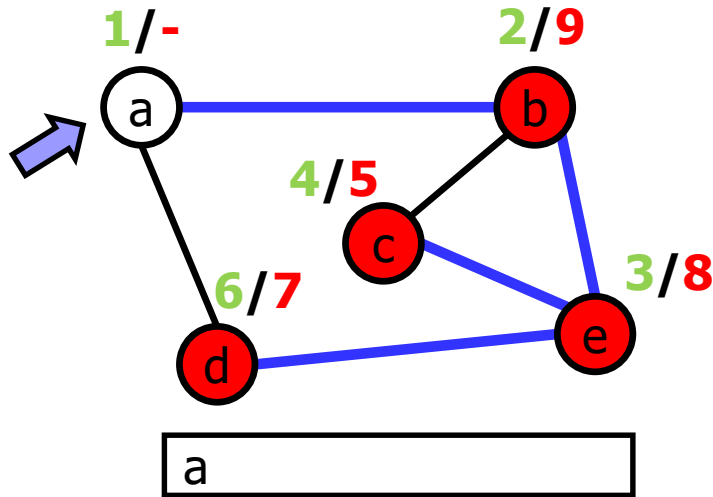


CLOSED



Průchod grafem do hloubky (DFS)

Depth-first search



stavy uzlu:



FRESH

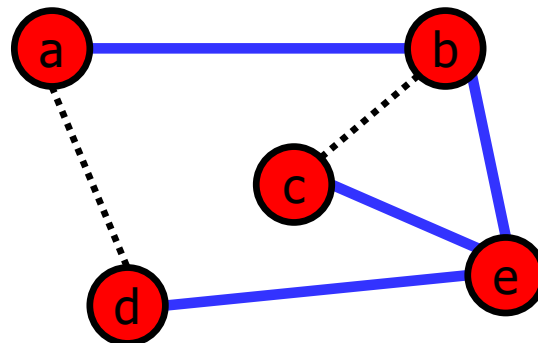


OPEN



CLOSED

DFS-strom:



DFS rekurzivně

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

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

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);
```

simuluje rekurzi

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

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

slido



Jakou má DFS časovou složitost?

ⓘ Start presenting to display the poll results on this slide.

Časová složitost DFS

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

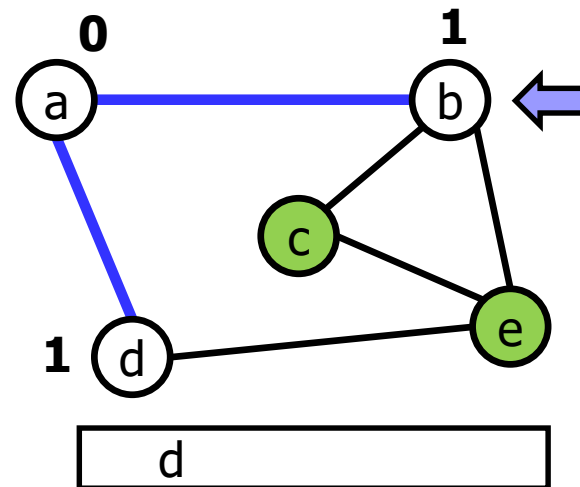
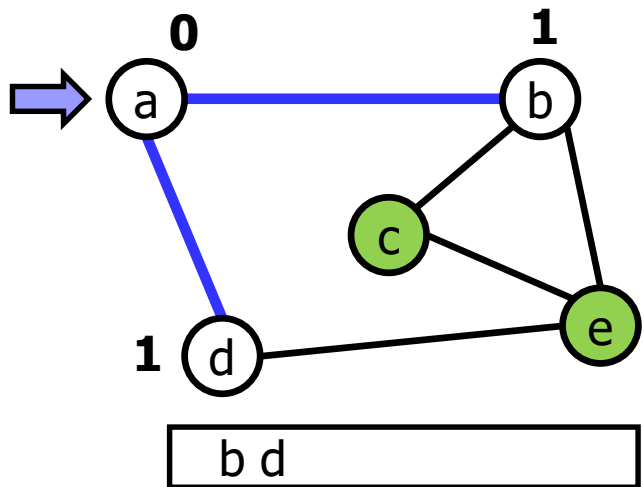
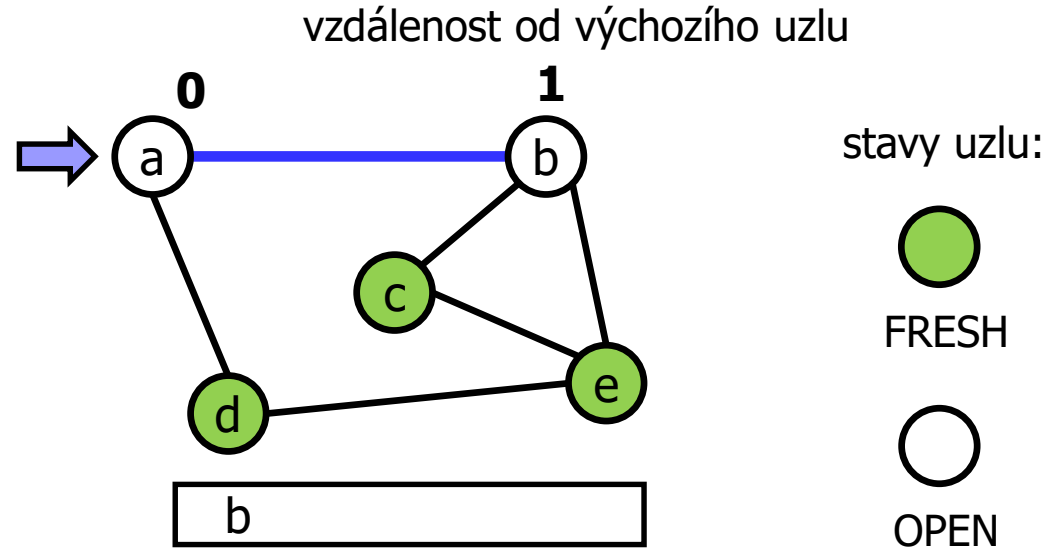
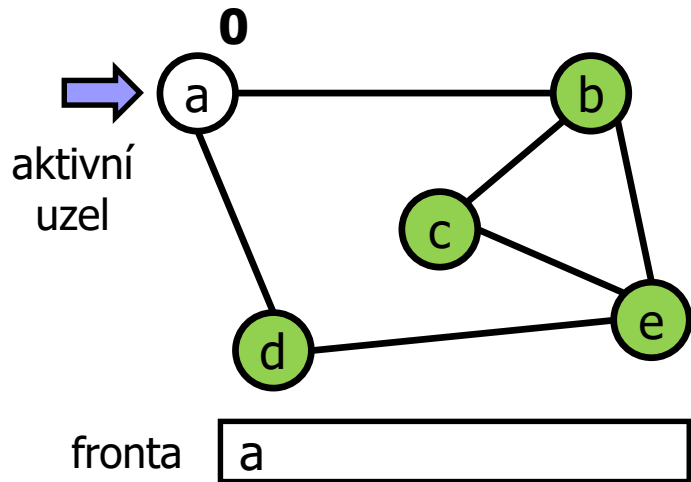
```
DFS(Node[] nodes) :  
    foreach node in nodes do  
        if !node.visited then  
            DFS(node);  
        end;  
    end;
```

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

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

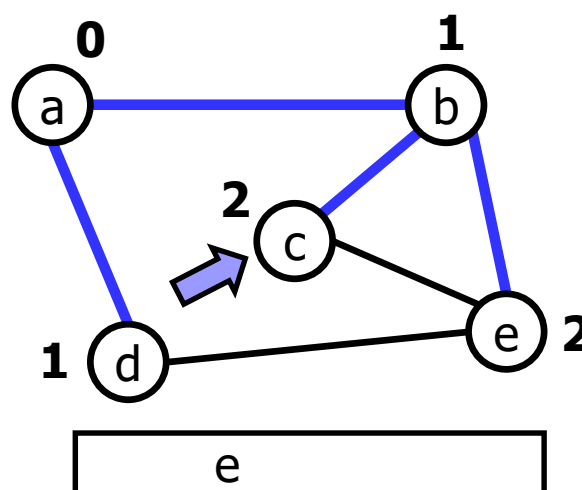
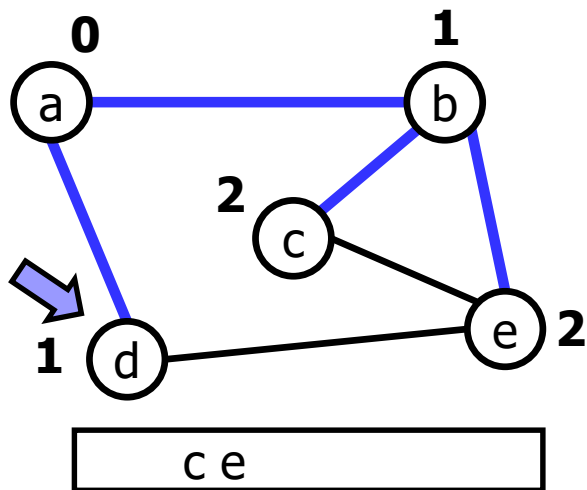
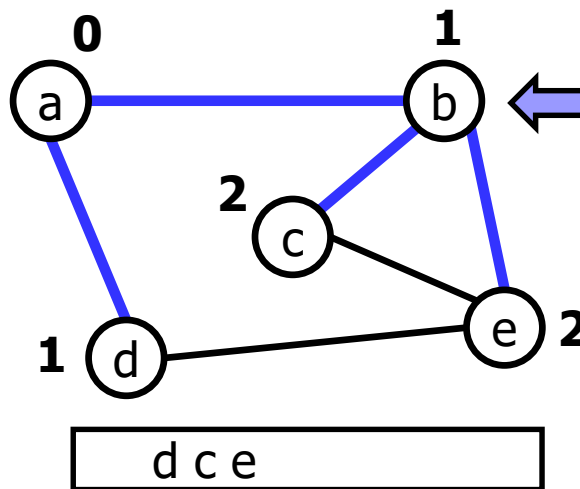
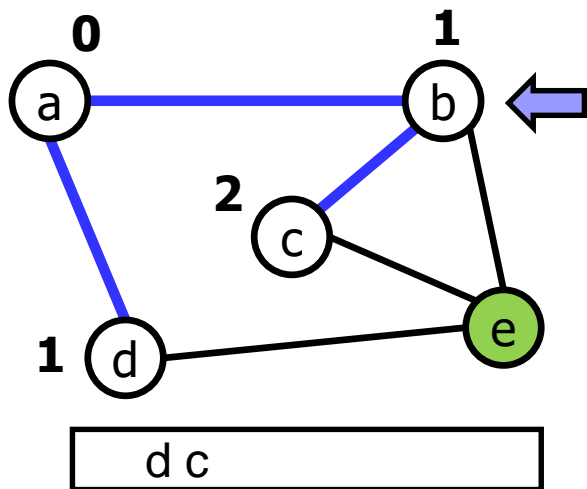
Průchod grafem do šířky (BFS)

Breadth-first search



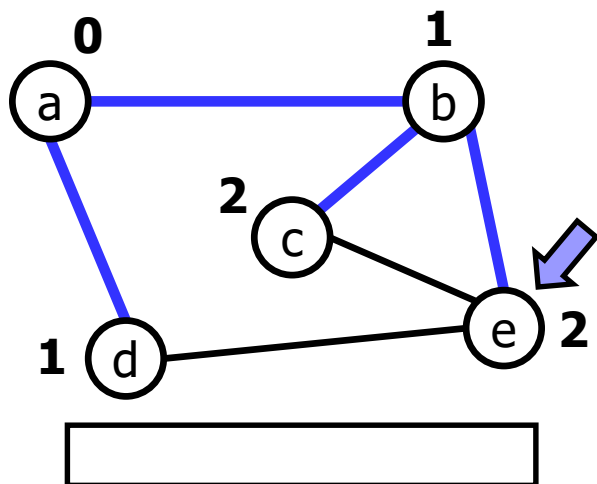
Průchod grafem do šířky (BFS)

■ Breadth-first search

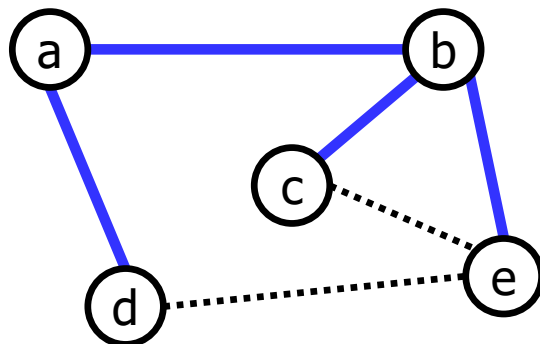


Průchod grafem do šířky (BFS)

■ Breadth-first search



BFS-strom:



BFS s frontou

```
BFS(Node node): // průchod jednou komponentou
```

```
  Q = new Queue();
```

```
  node.discovered = true;
```

```
  Q.push(node);
```

```
  while !Q.isEmpty() do
```

```
    node = Q.pop();
```

```
    foreach n in node.neighbors do
```

```
      if !n.discovered then
```

```
        n.discovered = true;
```

```
        Q.push(n);
```

```
      end;
```

```
    end;
```

```
  end;
```

```
BFS(Node[] nodes): // průchod celým grafem
```

```
  foreach node in nodes do
```

```
    if !node.discovered then
```

```
      BFS(node);
```

```
    end;
```

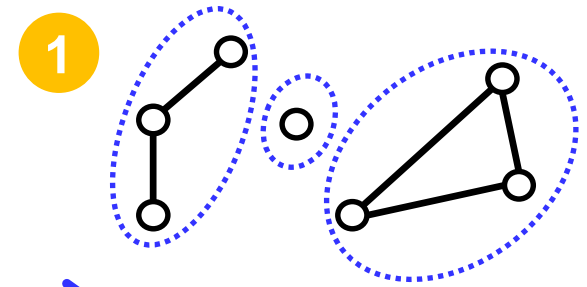
```
  end;
```

časová složitost je

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

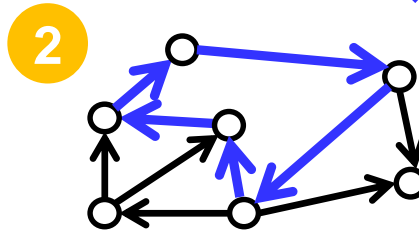
Aplikace průchodu grafem

1. Detekce komponent souvislosti

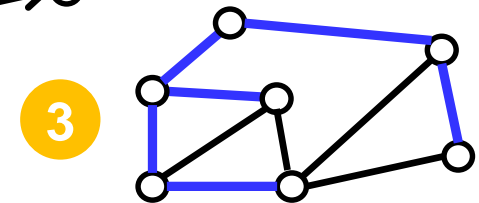


2. Detekce cyklu

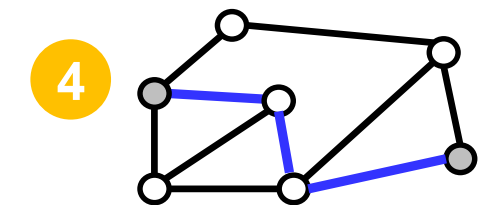
(pokud při DFS objevíme uzel ve stavu OPEN => cyklus)



3. Nalezení kostry

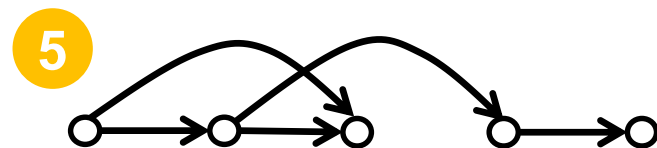


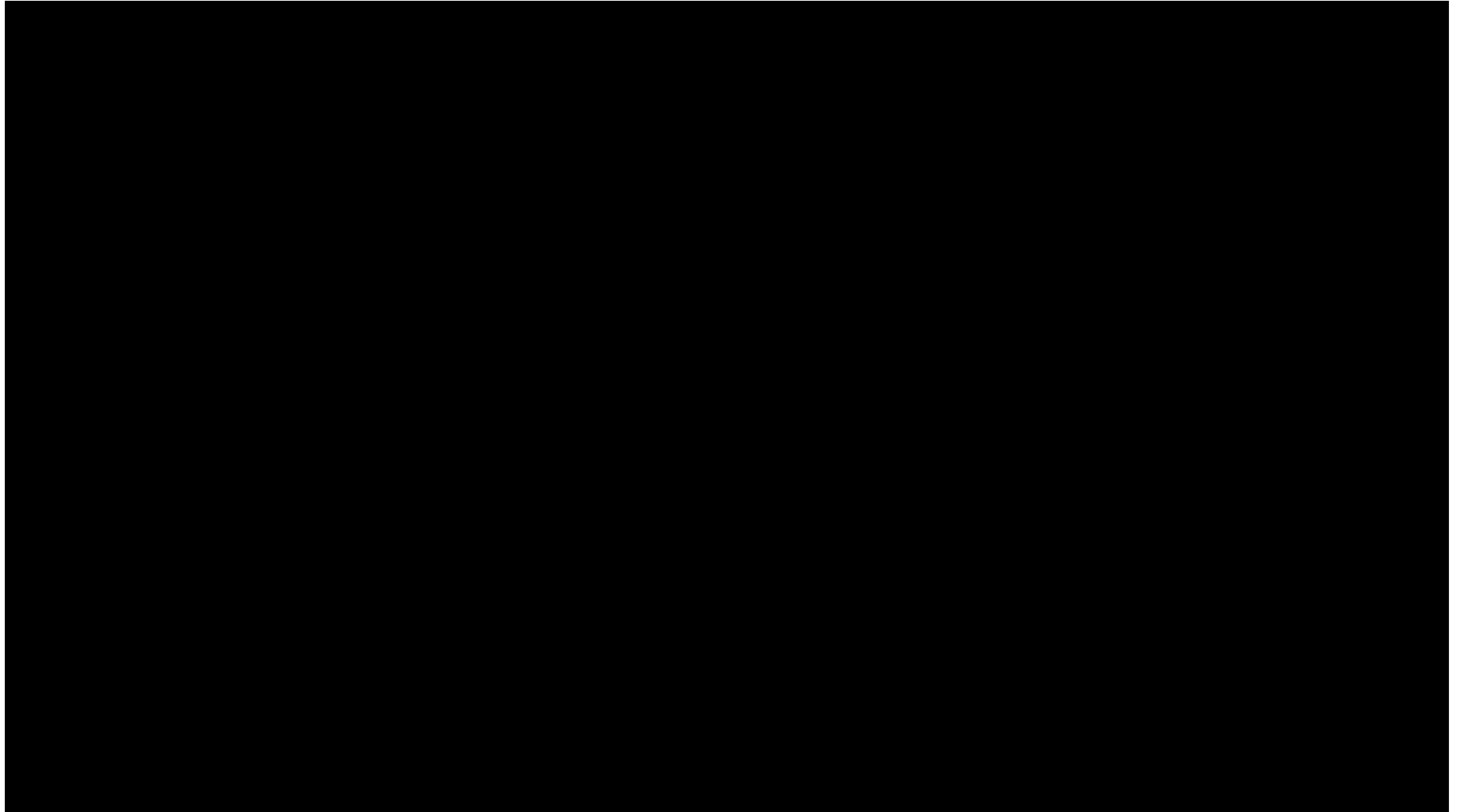
4. Hranově nejkratší cesta (jen BFS)



5. Topologické uspořádání (jen DFS)

(uzly uspořádáme sestupně podle časů jejich uzavření)



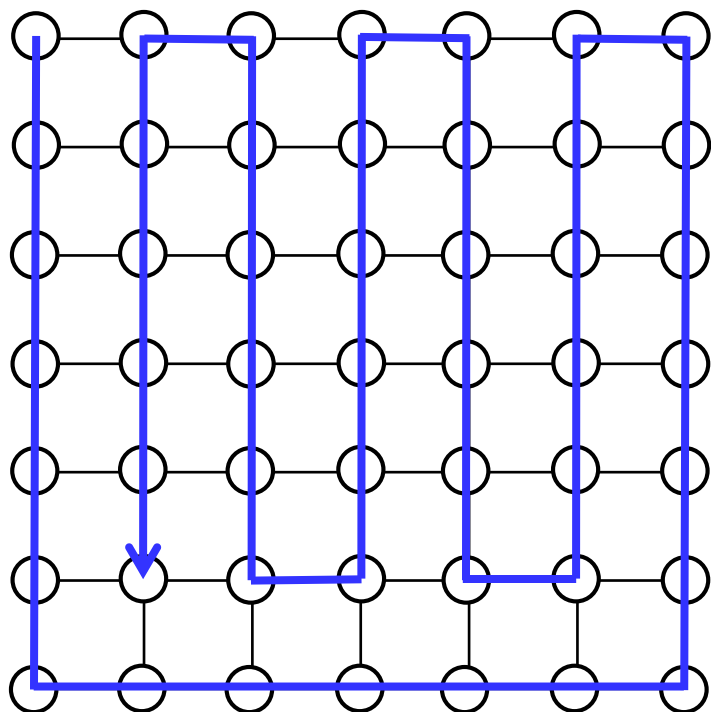


<https://youtu.be/NUgMa5coCoE>

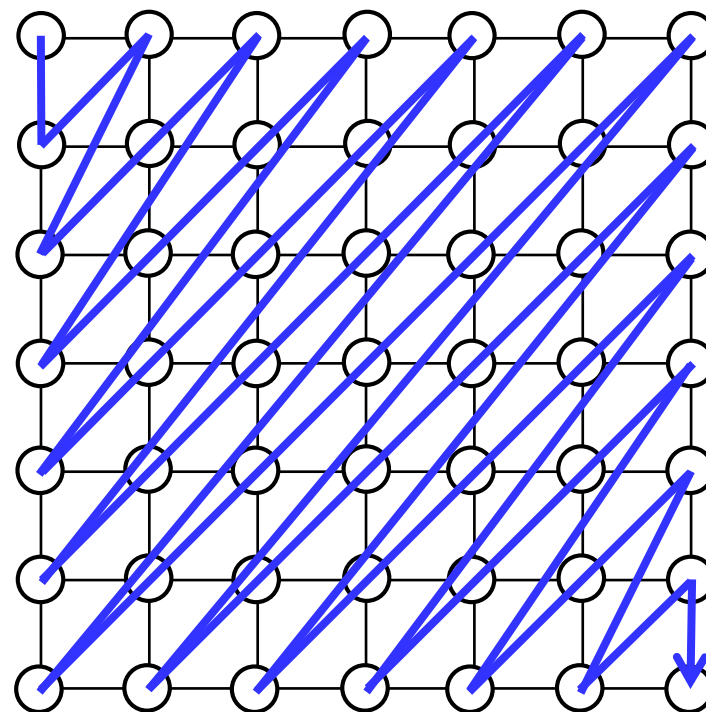
Porovnání DFS a BFS

Mřížka $N \times N$, uspořádání susedů: $\downarrow \rightarrow \uparrow \leftarrow$

DFS (rekurzivně)



BFS



Potřebná velikost zásobníku / fronty:

$$\Theta(N^2)$$

$$\Theta(N)$$



Otázky, komentáře, povzdechy...

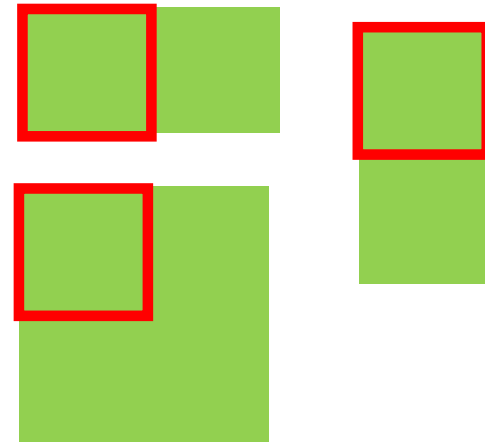
ⓘ Start presenting to display the poll results on this slide.

Druhá domácí úloha

Iterovat přes políčka mřížky, zkusit postupně umístit jednotlivé pozemky a nebo nechat políčko neobsazené.



9	9	9	8	6	4
9	8	8	8	5	3
8	7	6	6	4	2
6	5	5	5	3	1



Políčka můžeme procházet v uspořádání od největších hodnot po nejmenší.



Druhá domácí úloha

Horní odhad určíme na základě hodnot dosud neprošlých políček (pouze těch, které nesousedí s umístěnými pozemky).



9	9	9	8	6	4
9	8	8	8	5	3
8	7	6	6	4	2
6	5	5	5	3	1

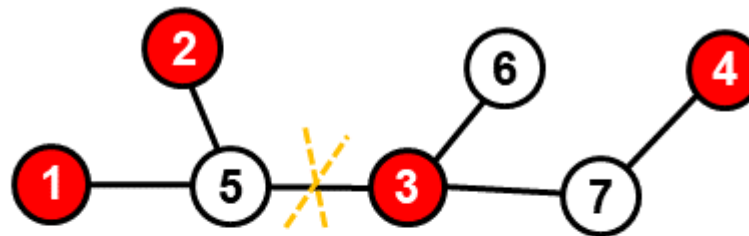
Zadání a řešení úlohy o nanorobotech:

<https://cw.fel.cvut.cz/wiki/courses/b4b33alg/prednasky>

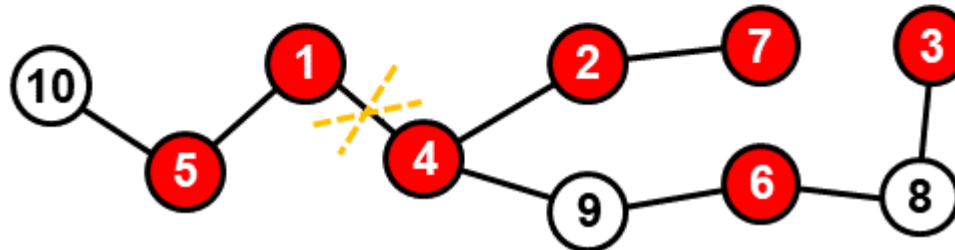


Třetí domácí úloha

a)



b)





Otázky, komentáře, povzdechy...

ⓘ Start presenting to display the poll results on this slide.