

# ALG 04

**Zásobník**

**Fronta**

**Operace Enqueue, Dequeue, Front, Empty....**

**Cyklická implementace fronty**

**Průchod stromem do šířky**

**Grafy**

**průchod grafem do šířky**

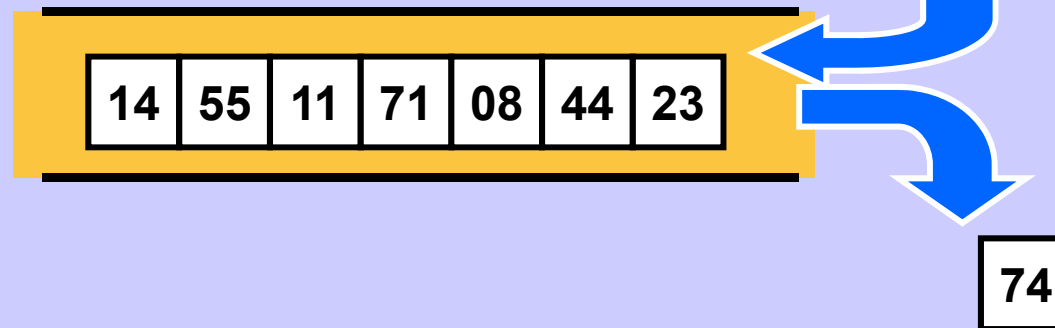
**průchod grafem do hloubky**

**Ořezávání a heuristiky**

## Zásobník / stack

Prvky se před zpracováním vkládají na vrchol zásobníku.

Vrchol / top

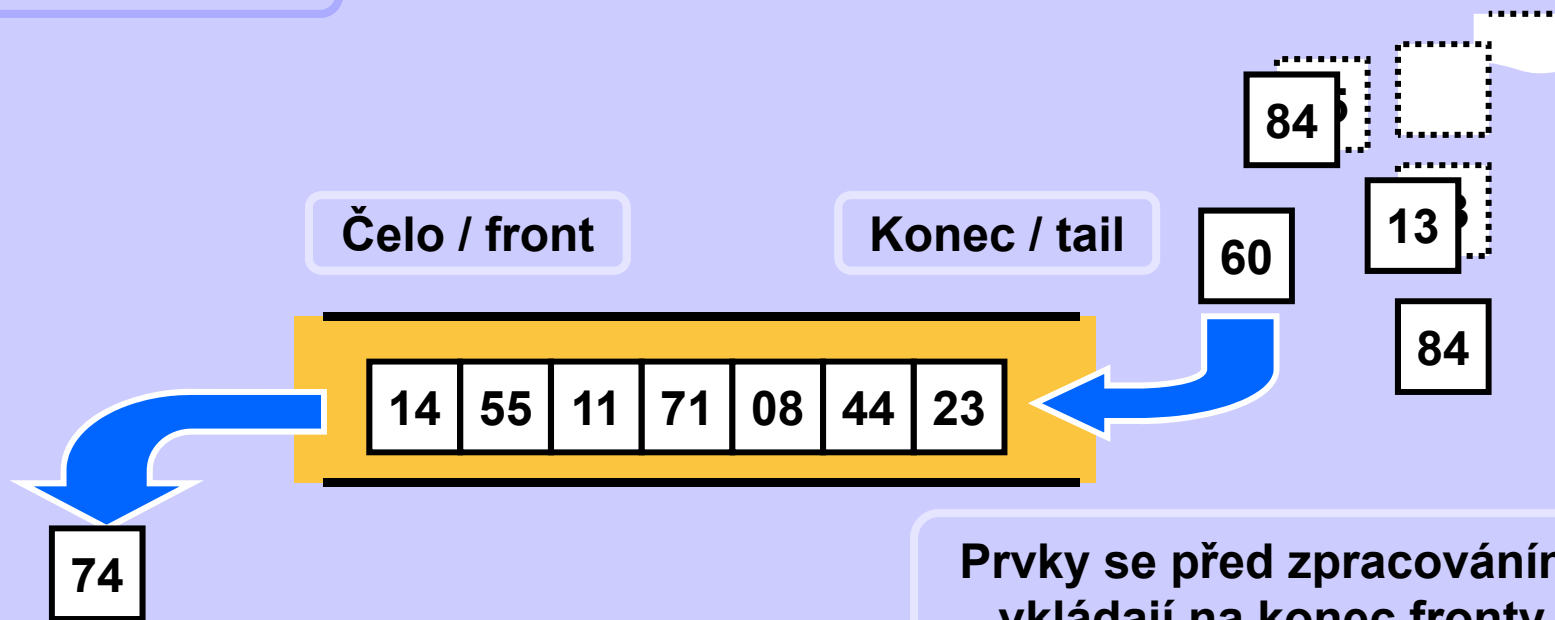


Prvky se odebírají z vrcholu zásobníku a pak se zpracovávají.

### Operace

Vlož na vrchol	Push
Odeber z vrcholu	Pop
Čti začátek	Top
Je prázdný?	Empty

## Fronta / queue



Prvky se odebírají z čela fronty a pak se zpracovávají.

### Operace

Vlož na konec

Enqueue / InsertLast / Push ...

Odeber ze začátku

Dequeue / delFront / Pop ...

Čti začátek

Front / Peek ...

Je prázdná?

Empty

# Fronta

Jednoduchý  
příklad života  
fronty

Čelo

Konec

Prázdná

Vlož(24)

Vlož(11)

Vlož(90)

Odeber()

Vlož(43)

Odeber()

Odeber()

Vlož(79)

24

24 11

24 11 90

11 90

11 90 43

90 43

43

43 79

## Cyklická implementace fronty polem

Prázdná fronta  
v poli pevné délky

Vlož 24, 11, 90, 43, 70.

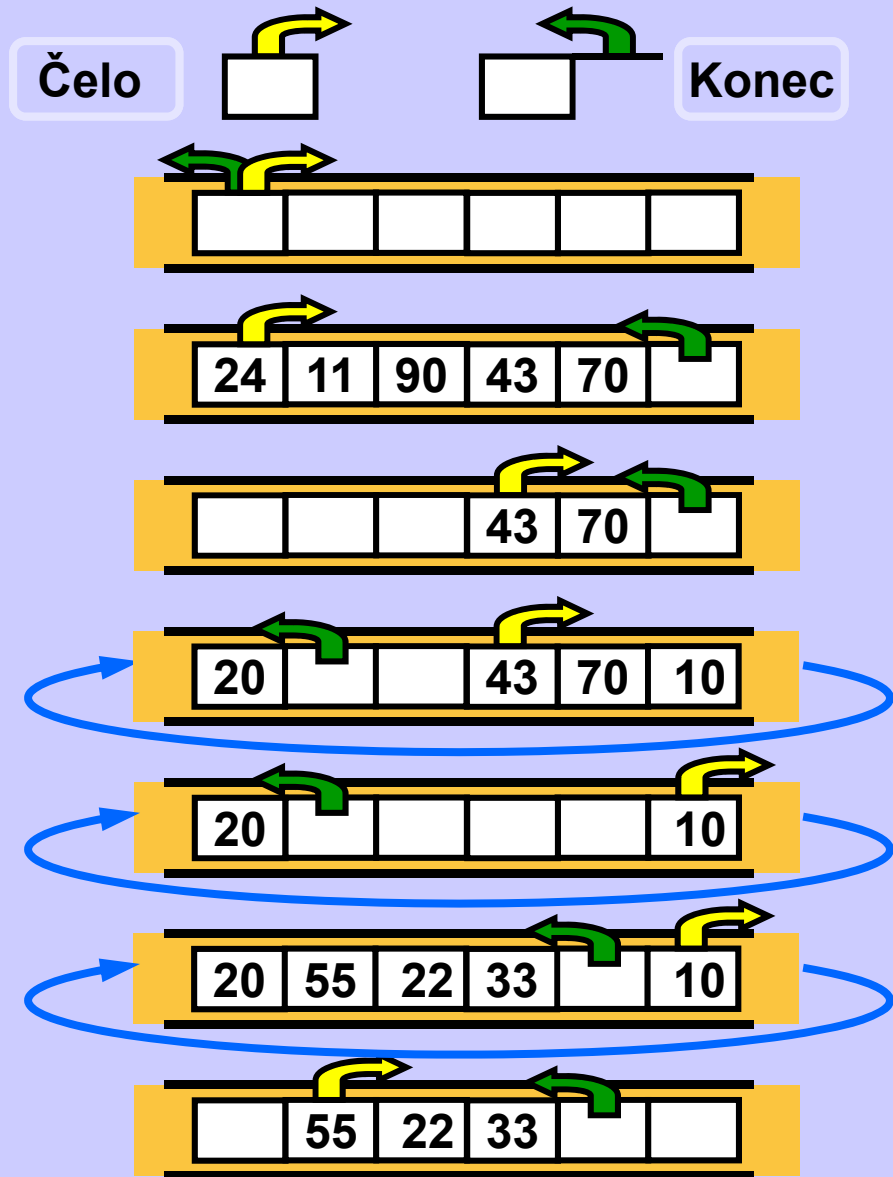
Odeber, odeber, odeber.

Vlož 10, 20.

Odeber, odeber.

Vlož 55, 22, 33.

Odeber, odeber.



## Cyklická implementace fronty polem

Index/ukazatel konce fronty ukazuje na první volnou pozici za posledním prvkem fronty. Index/ukazatel čela fronty ukazuje na první obsazenou pozici. Pokud oba ukazují tamtéž, fronta je prázdná.

```
class Queue {
    Node q [];
    int size;
    int front;
    int tail;

    Queue(int qsize) {
        size = qsize;
        q = new Node[size];
        front = 0;
        tail = 0;
    }

    boolean Empty() {
        return (tail==front);
    }

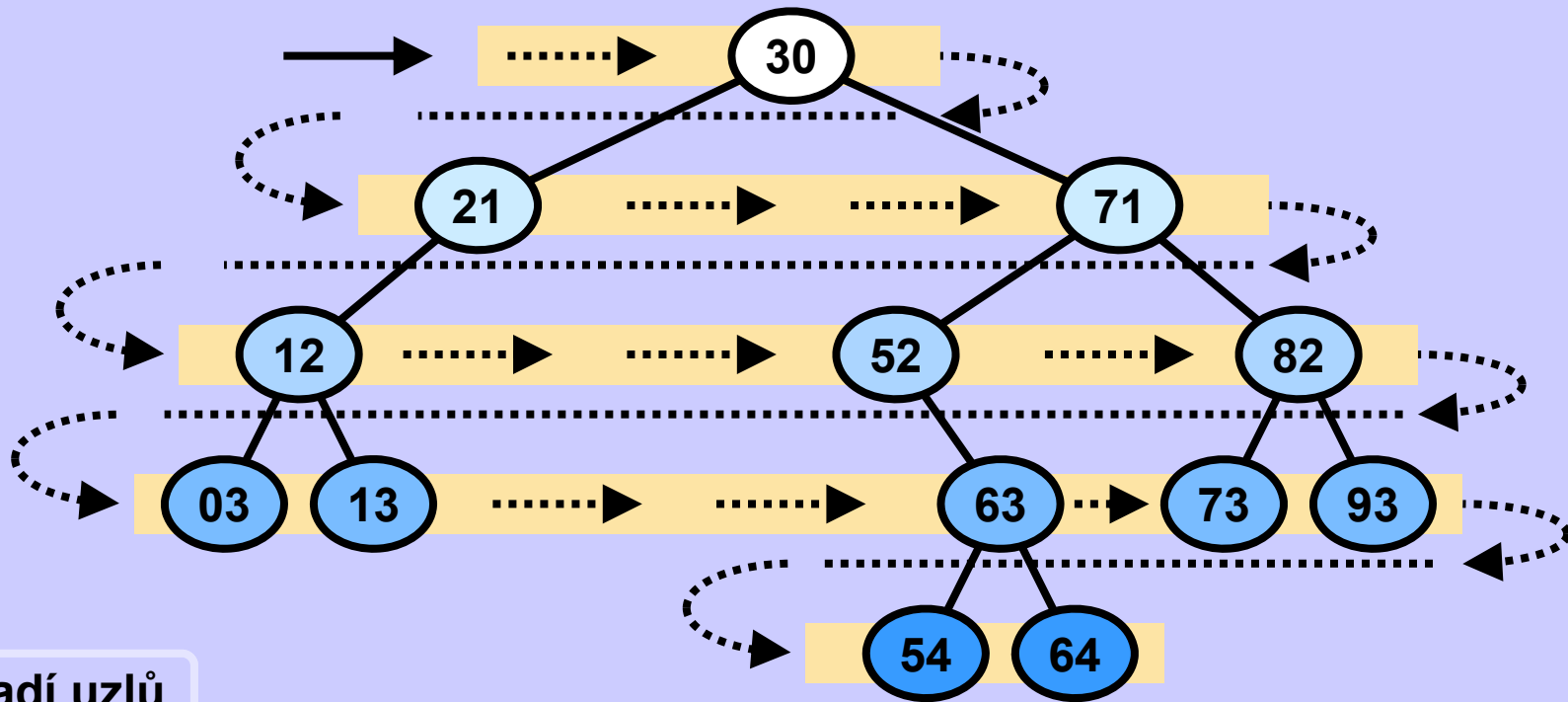
    void Enqueue(Node node) {
        if ((tail+1 == front) ||
            (tail-front == size-1))
            ... // queue full, fix it

        q[t++] = node;
        if (tail==size) tail=0;
    }

    Node Dequeue() {
        Node n = q[front++];
        if (front==size) front=0;
        return n;
    }
} // end of Queue
```

## Průchod stromem do šířky

Strom s naznačeným směrem průchodu



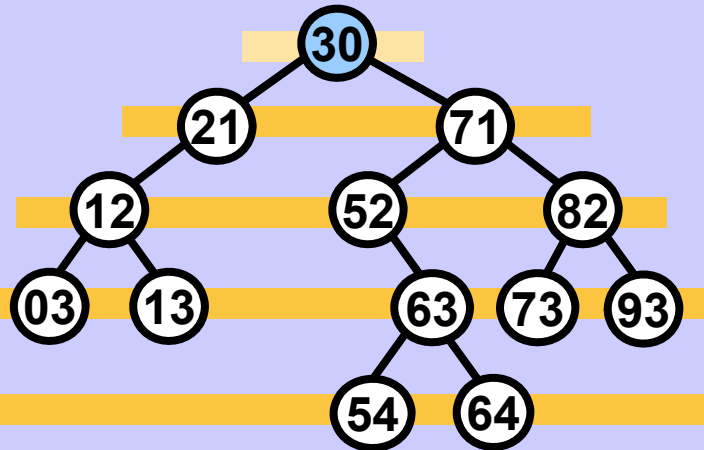
Pořadí uzlů

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

### Inicializace



Výstup

2.

Vytvoř prázdnou frontu

Do fronty vlož kořen stromu



Čelo

Konec

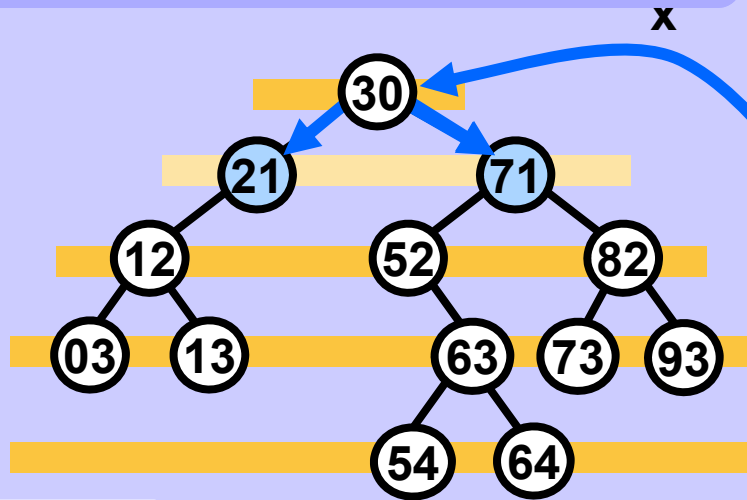
### Hlavní cyklus

Dokud není fronta prázdná, opakuj:

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



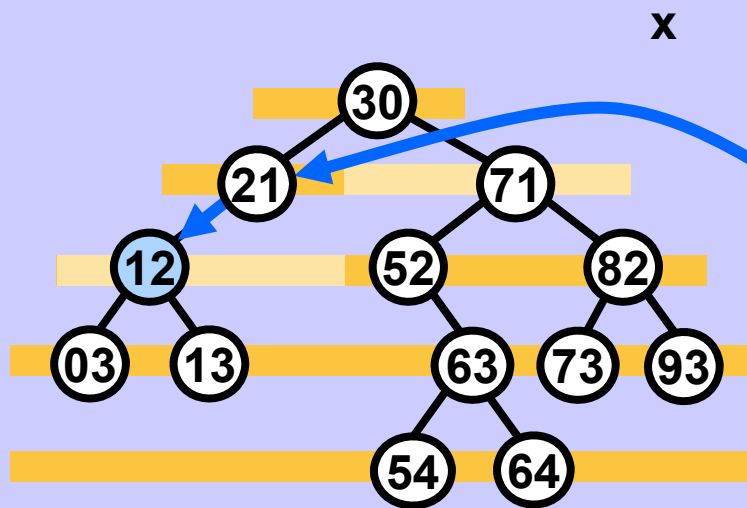
# Průchod stromem do šířky



Výstup 30

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

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



Výstup 30 21

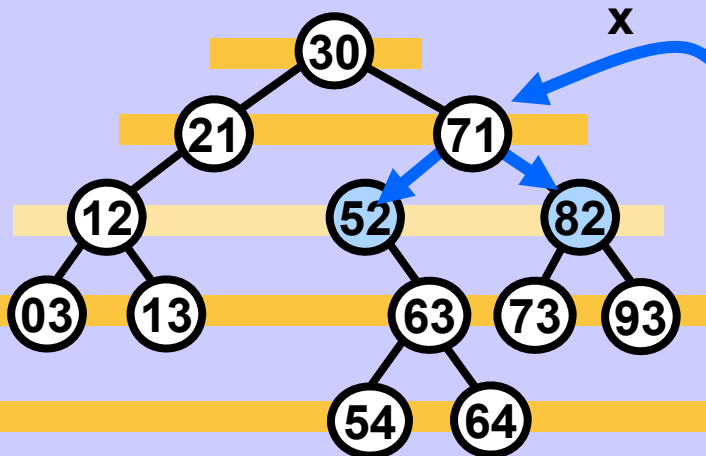
1.  $x = \text{Odeber}(), \text{ tisk}(x.\text{key}).$

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



\*) pokud existuje

# Průchod stromem do šířky

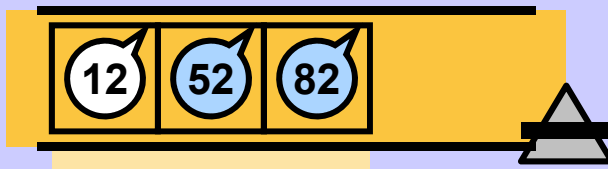


Výstup 30 21 71

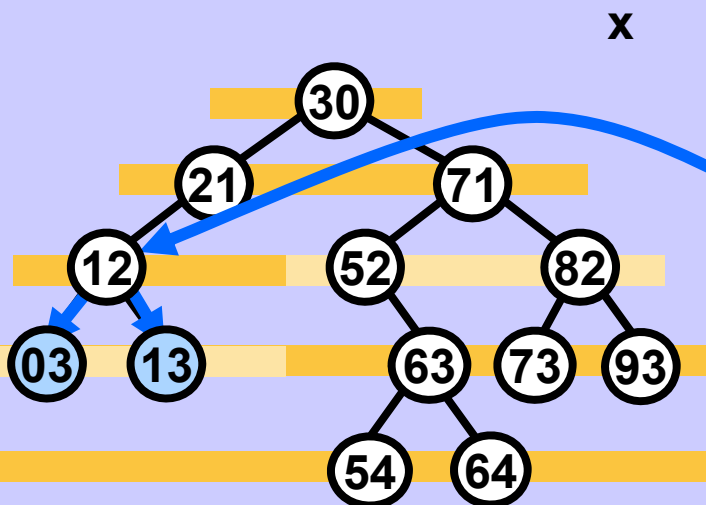
1.  $x = \text{Odeber}(), \text{ tisk}(x.\text{key}).$



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



\*) pokud existuje



Výstup 30 21 71 12

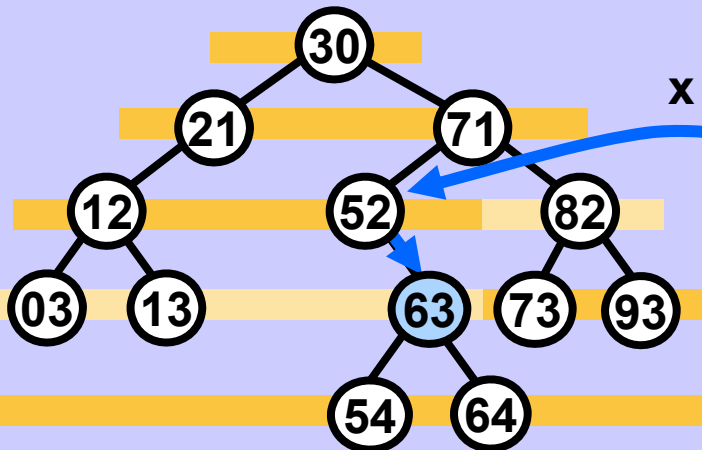
1.  $x = \text{Odeber}(), \text{ tisk}(x.\text{key}).$



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

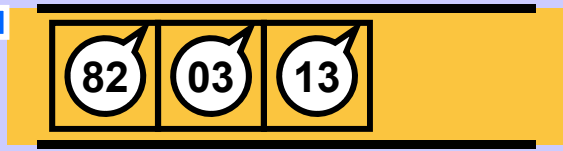


# Průchod stromem do šířky



Výstup 30 21 71 12 52

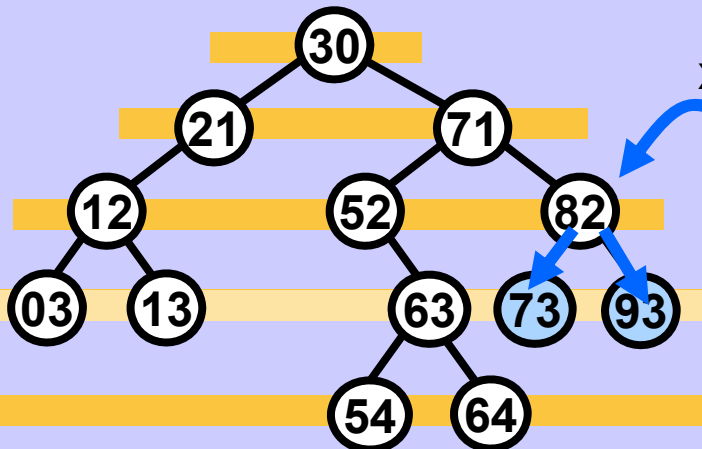
1.  $x = \text{Odeber}()$ , tisk ( $x.\text{key}$ ).



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

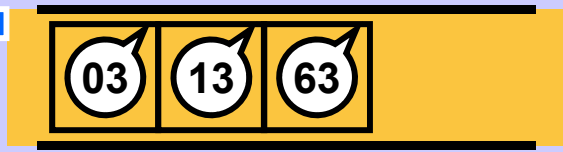


\*) pokud existuje

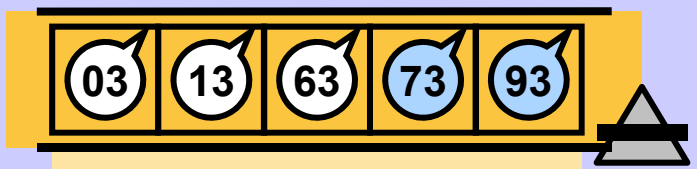


Výstup 30 21 71 12 52 82

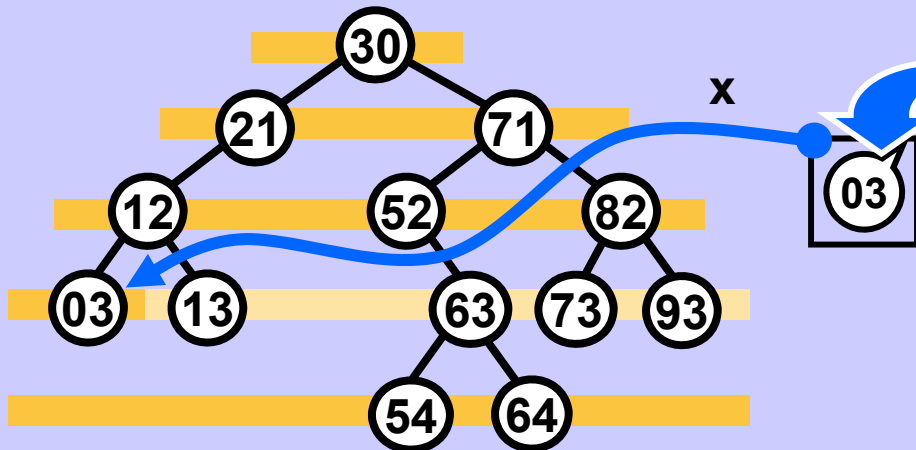
1.  $x = \text{Odeber}()$ , tisk( $x.\text{key}$ ).



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

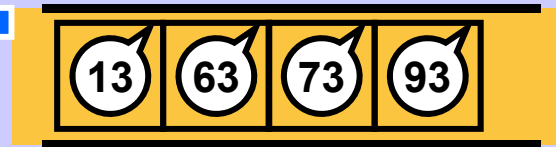


# Průchod stromem do šířky

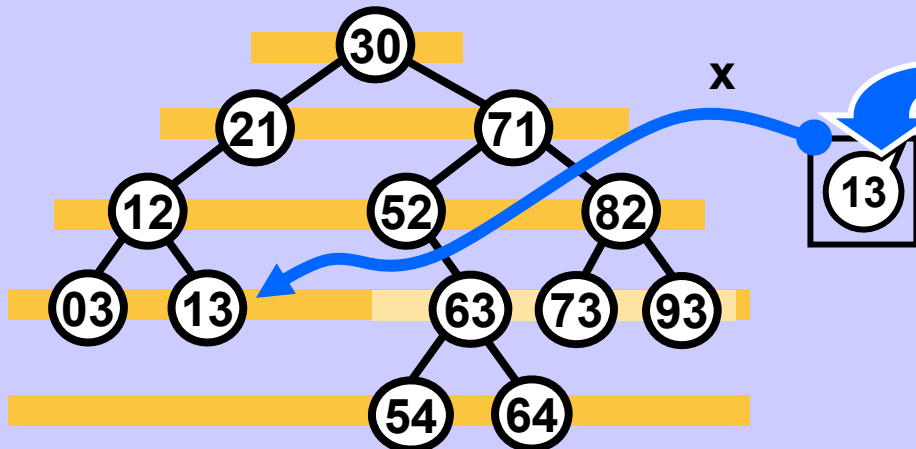
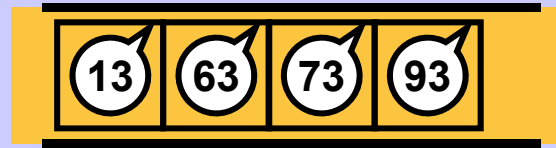


Výstup 30 21 71 12 52 82 03

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

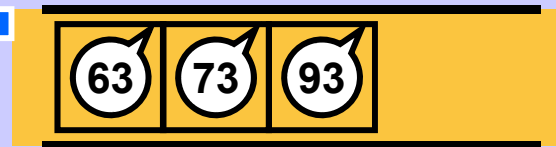


2. Vlož( $x.\text{left}$ ), vlož( $x.\text{right}$ ). \*)

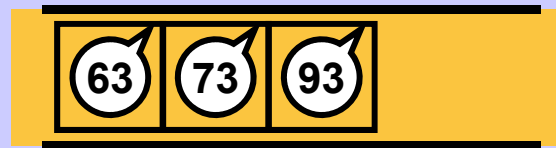


Výstup 30 21 71 12 52 82 03 13

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

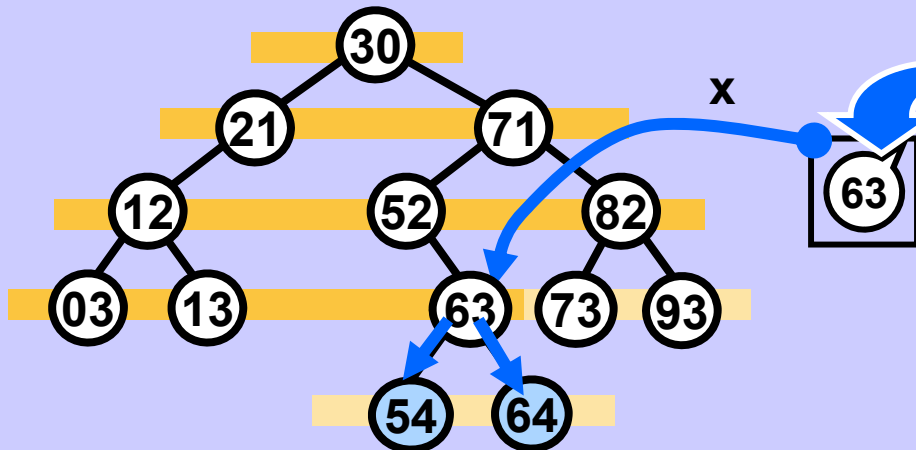


2. Vlož( $x.\text{left}$ ), vlož( $x.\text{right}$ ). \*)



\*) pokud existuje

# Průchod stromem do šířky

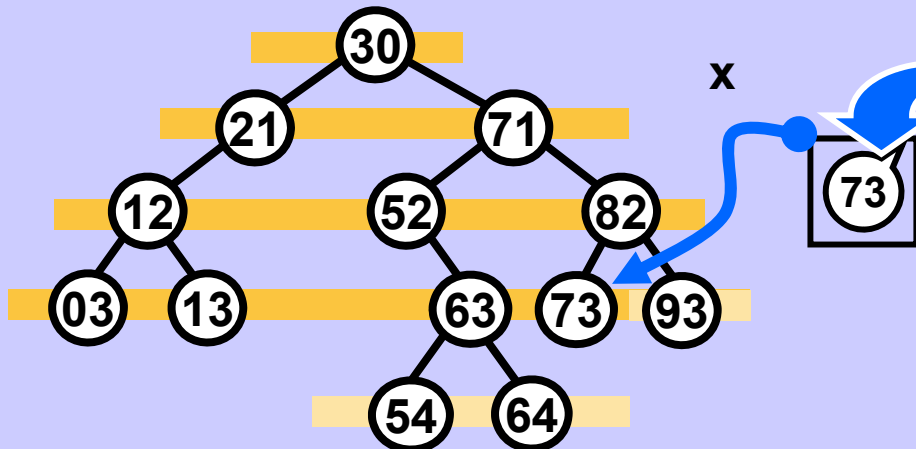
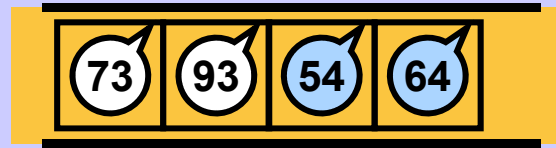


Výstup 30 21 71 12 52 82 03 13 63

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

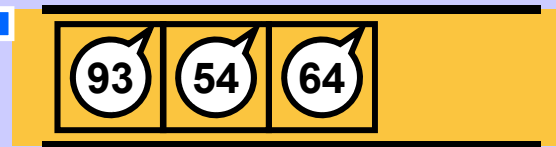


2. Vlož( $x.\text{left}$ ), vlož( $x.\text{right}$ ). \*)

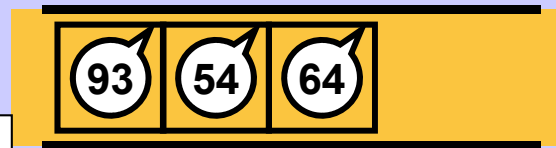


Výstup 30 21 71 12 52 82 03 13 63 73

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

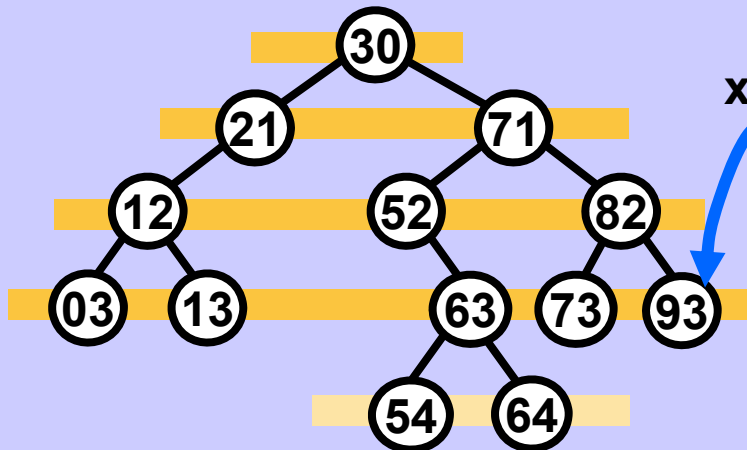


2. Vlož( $x.\text{left}$ ), vlož( $x.\text{right}$ ). \*)



\*) pokud existuje

# Průchod stromem do šířky



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

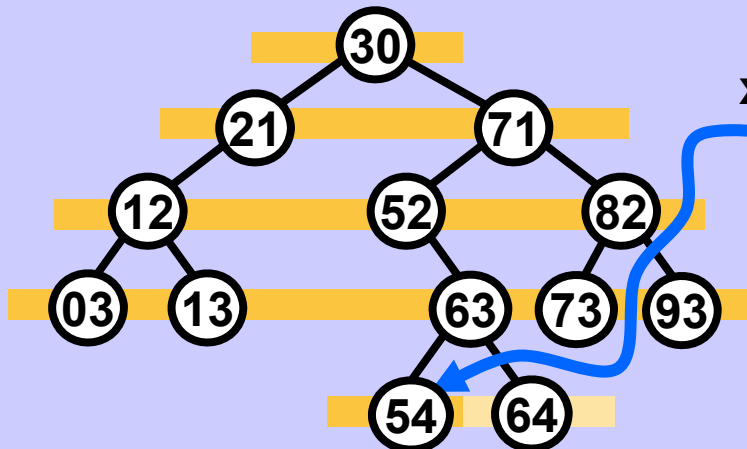


2. Vlož( $x.\text{left}$ ), vlož( $x.\text{right}$ ). \*)



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

\*) pokud existuje



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

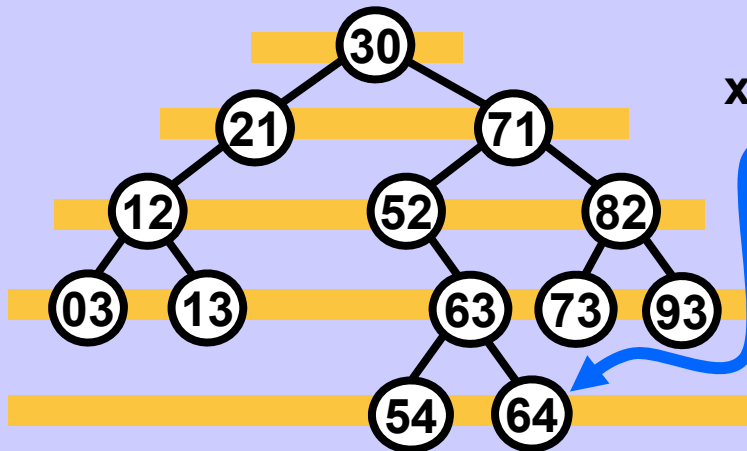


2. Vlož( $x.\text{left}$ ), vlož( $x.\text{right}$ ). \*)



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

## Průchod stromem do šířky



Výstup

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

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

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

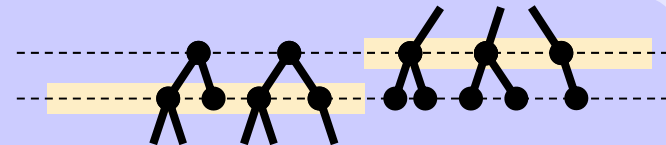
\*) pokud existuje

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. 

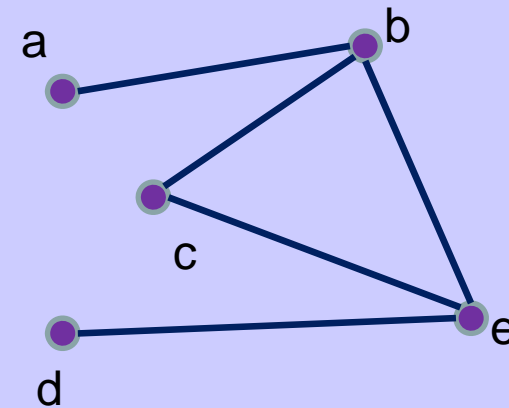
## Průchod stromem do šířky

```
void listBreadth (Node node) {  
    if (node == null) return;  
    Queue q = new Queue();           // init  
    q.Enqueue(node);                  // root into queue  
    while (!q.Empty()) {  
        node = q.Dequeue();  
        print(node.key);              // process node  
        if (node.left != null) q.Enqueue(node.left);  
        if (node.right != null) q.Enqueue(node.right);  
    }  
}
```



## Grafy

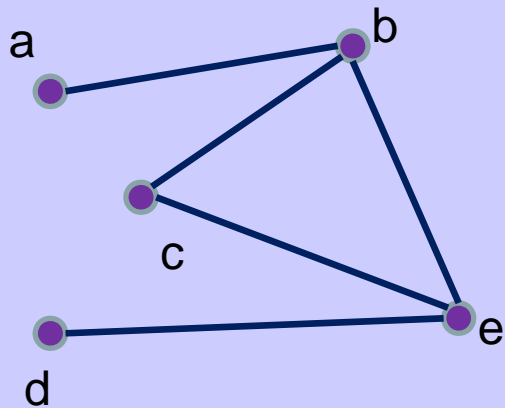
- graf je uspořádaná dvojice
  - množiny vrcholů  $\mathcal{V}$  a
  - množiny hran  $\mathcal{E}$
- $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
- příklad:
  - $\mathcal{V} = \{a, b, c, d, e\}$
  - $\mathcal{E} = \{\{a,b\}, \{b,e\}, \{b,c\}, \{c,e\}, \{e,d\}\}$



## Grafy - orientovanost

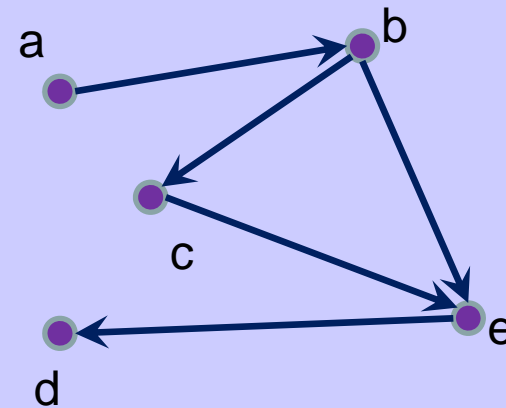
### • neorientovaný graf

- hrana je neuspořádaná dvojice vrcholů
- $E = \{\{a,b\},\{b,e\},\{b,c\},\{c,e\},\{e,d\}\}$



### • orientovaný graf

- hrana je uspořádaná dvojice vrcholů
- $E = \{\{a,b\},\{b,e\},\{b,c\},\{c,e\},\{e,d\}\}$



## Grafy – matice sousednosti

- Necht'  $G = (\mathcal{V}, \mathcal{E})$  je graf s  $n$  vrcholy
- Označme vrcholy  $v_1, \dots, v_n$  (v nějakém libovolném pořadí)
- Matice sousednosti grafu  $G$  je čtvercová matice

$$A_G = (a_{i,j})_{i,j=1}^n$$

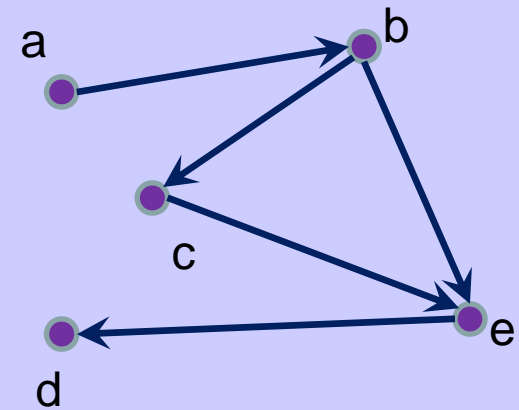
definovaná předpisem

$$a_{i,j} = \begin{cases} 1 & \text{pro } \{v_i, v_j\} \in E \\ 0 & \text{jinak} \end{cases}$$

## Grafy – matice sousednosti

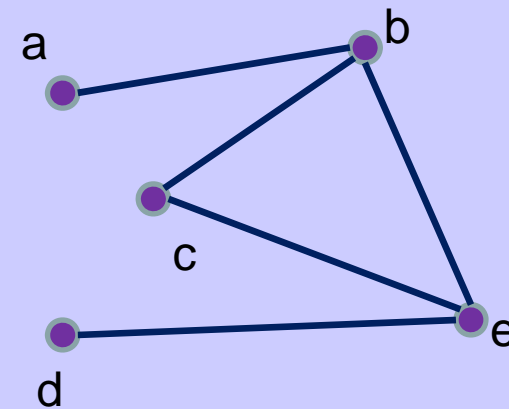
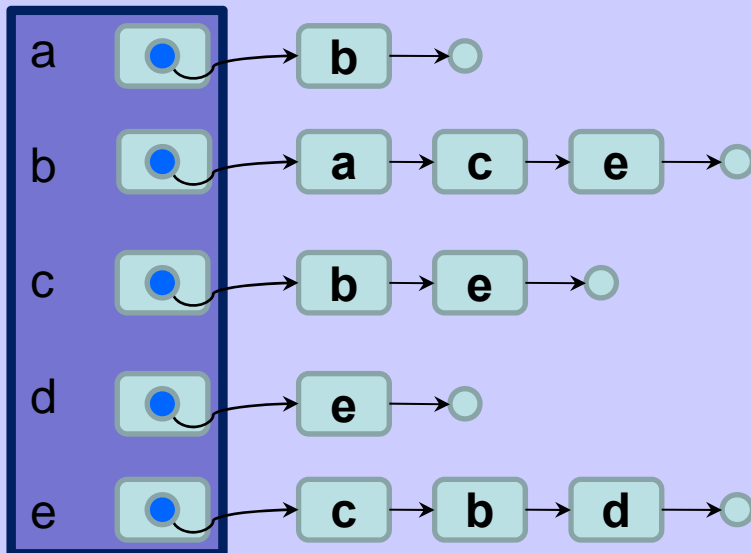
- pro orientovaný graf

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



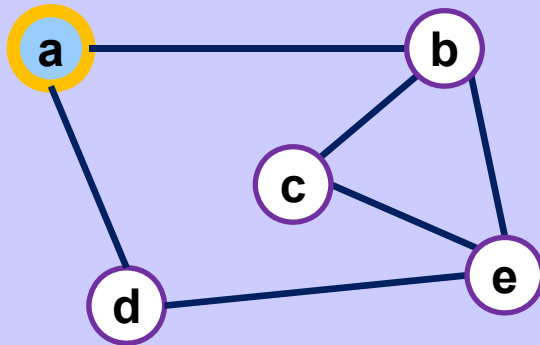
## Grafy – seznam sousedů

- Necht'  $G = (\mathcal{V}, E)$  je (ne)orientovaný graf s  $n$  vrcholy
- Označme vrcholy  $v_1, \dots, v_n$  (v nějakém libovolném pořadí)
- Seznam sousedů grafu  $G$  je pole  $\mathcal{P}$  ukazatelů velikosti  $n$ 
  - kde  $\mathcal{P}[i]$  ukazuje na spojový seznam vrcholů, se kterými je vrchol  $v_i$  spojen hranou



## Průchod grafem do hloubky

### Inicializace



Vytvoř prázdný zásobník



Do zásobníku vlož počáteční uzel



Výstup

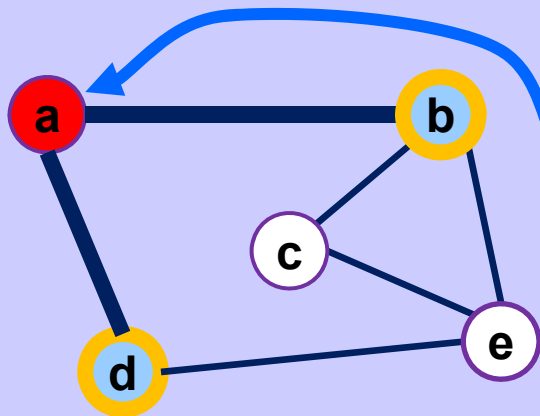
Vrchol

### Hlavní cyklus

Dokud není zásobník prázdný, opakuj:

1. Odeber první uzel ze zásobníku a zpracuj ho.
2. Do zásobníku vlož jeho *nenavštívené* sousedy, pokud existují.

# Průchod grafem do hloubky



Výstup

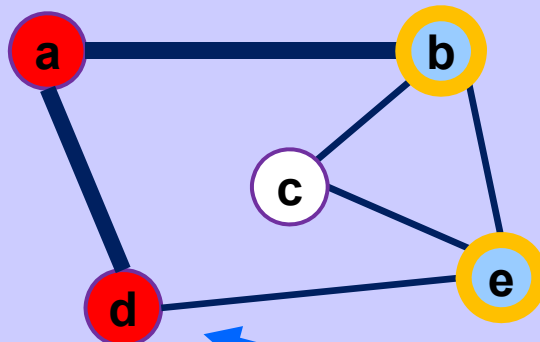
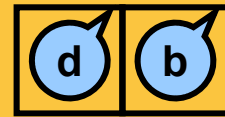
a

1. node = Pop(), print (node.key)

node



2. Push(node.Neighbors())



Výstup

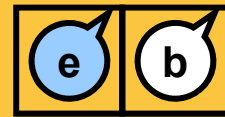
a d

1. node = Pop(), print(node.key)

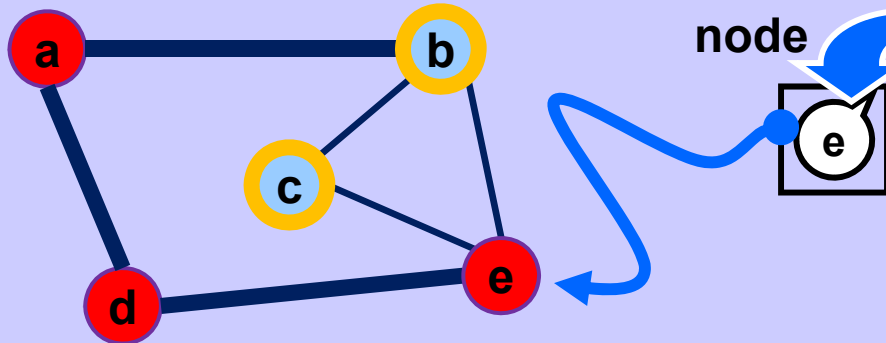
node



2. Push(node.Neighbors())



# Průchod grafem do hloubky



1. `node = Pop(), print(node.key)`

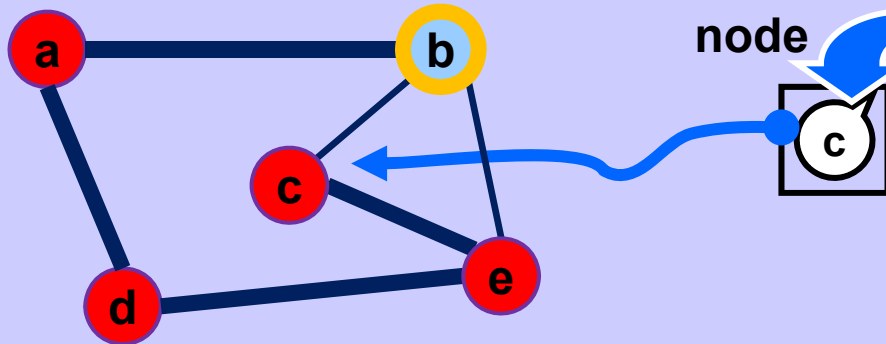


2. `Push(node.Neighbors())`



Výstup

a d e



1. `node = Pop().`

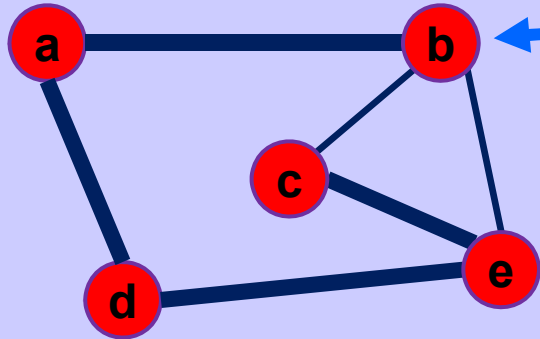


Výstup

a d e c



## Průchod grafem do hloubky



1. `node = Pop(), print(node.key)`

node



KONEC

Výstup

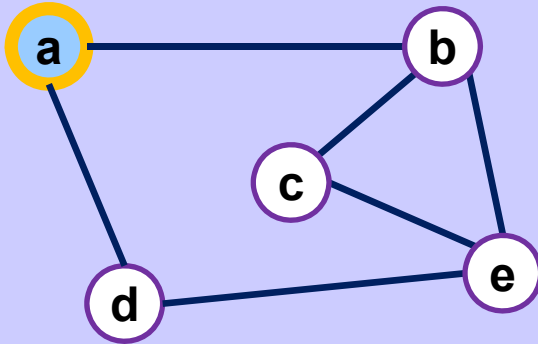
a d e c b

## Průchod grafem do hloubky

```
void graphDepth( Node startNode ){
    Set visited = new Set();
    Stack q = new Stack ();           // init
    q.Push( startNode );              // startNode into queue
    visited.add( startNode );
    while( !q.Empty() ){
        node = q.Pop();
        print( node.key );             // process node
        forall x in node.Neighbors()
            if( x not in visited ){
                visited.add( x );
                q.Push( x );
            }
    }
}
```

## Průchod grafem do šířky

### Inicializace



Vytvoř prázdnou frontu



Do fronty vlož počáteční uzel



Výstup

Čelo

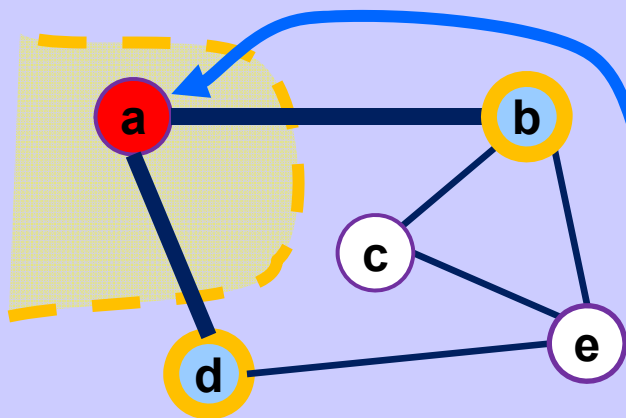
Konec

### Hlavní cyklus

Dokud není fronta prázdná, opakuj:

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

# Průchod grafem do šířky



1. `node = Dequeue(), print (node.key)`

node

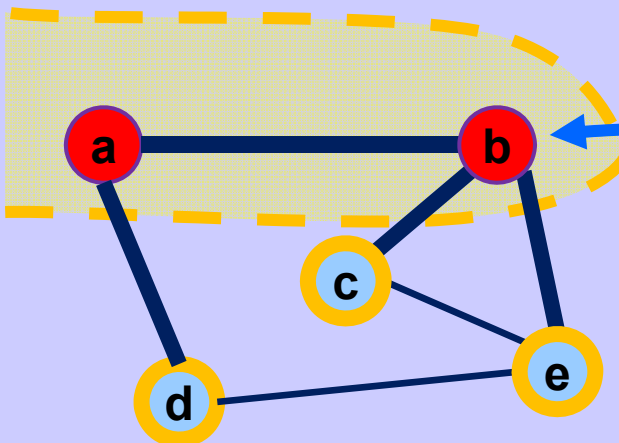


2. `Enqueue(node.Neighbors())`



Výstup

a

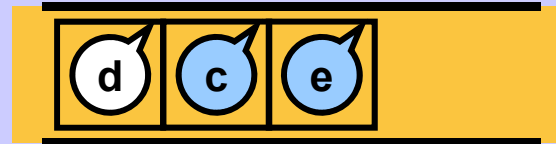


1. `node = Dequeue(), print (node.key)`

node



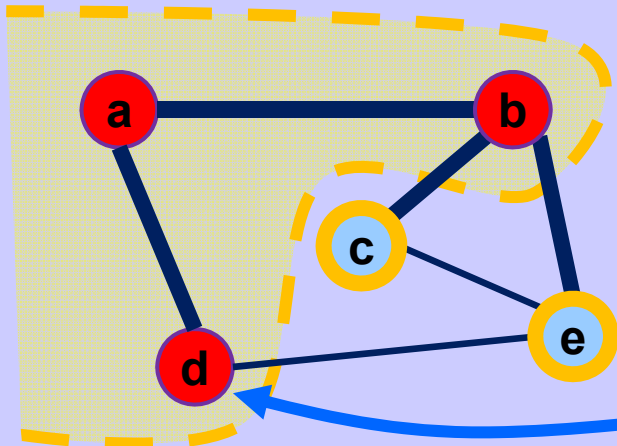
2. `Enqueue(node.Neighbors())`



Výstup

a b

# Průchod grafem do šířky



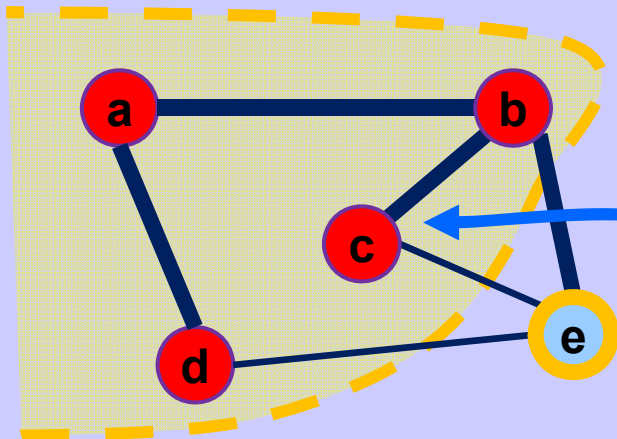
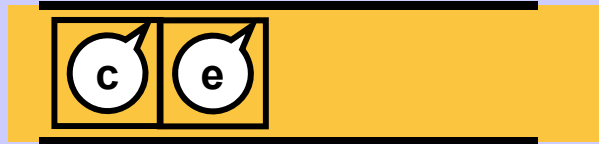
Výstup a b d

1. node = Dequeue(), print (node.key)

node



2. Enqueue(node.Neighbors())



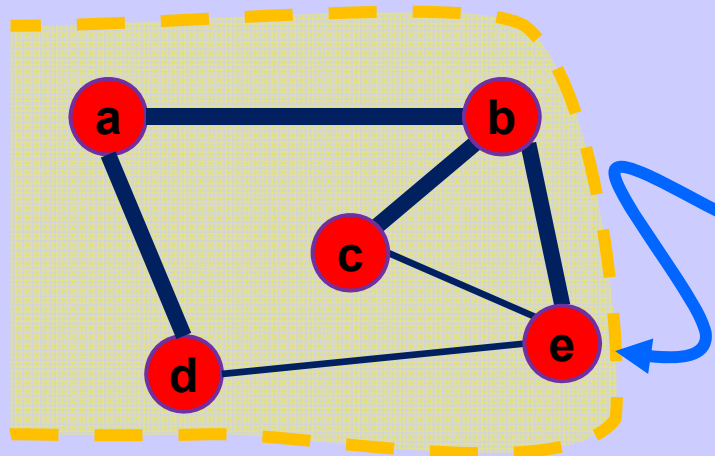
Výstup a b d c

1. node = Dequeue()

node



## Průchod grafem do šířky



1. `node = Dequeue(), print (node.key).`

node



KONEC

Výstup

a b d c e

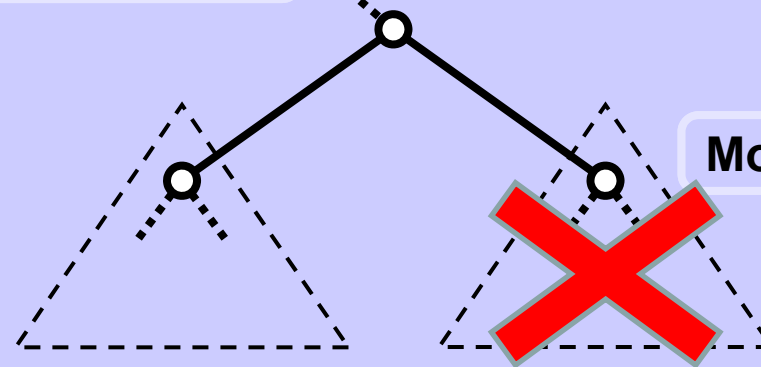
## Průchod grafem do šířky

```
void graphBreadthFirstSearch( Node startNode ) {  
    Set visited = new Set();  
    Queue q = new Queue();           // init  
    q.Enqueue( startNode );          // startNode into queue  
    visited.add( startNode );  
    while( !q.Empty() ) {  
        node = q.Dequeue();  
        print( node.key );           // process node  
        forall x in node.Neighbors()  
        if( x not in visited ){  
            visited.add( x );  
            q.Enqueue( x );  
        }  
    }  
}
```

## Ořezávání

- Urychlení prohledávání
- Ořezávání neperspektivních větví
- Pokud jsme schopni na základě vyhodnocení momentálního stavu zjistit,
  - že je to stav neperspektivní a
  - že rozhodně nepovede k řešení úlohy
- „odřízneme“ ze stromu celý podstrom momentálního stavu

Strom prohledávání





## Příklad ořezávání – magický čtverec

- Magický čtverec řádu  $\mathcal{N}$ 
  - čtvercové schéma čísel velikost  $\mathcal{N} \times \mathcal{N}$
  - obsahuje právě jednou každé celé číslo od  $1$  do  $\mathcal{N}^2$
  - součet čísel ve všech řádcích a ve všech sloupcích stejný

- Příklad

2	9	4
7	5	3
6	1	8

- Triviální řešení: generování všech možných rozmístění čísel od  $1$  do  $\mathcal{N}^2$
- Ořezávání: kdykoliv je součet na řádku neperspektivní
  - součet všech čísel čtverce je  $\frac{1}{2} \mathcal{N}^2 (\mathcal{N}^2 + 1)$
  - součet čísel na řádku je  $\frac{1}{2} \mathcal{N} (\mathcal{N}^2 + 1)$

## Heuristiky

- **Heuristika** je návod, který nám říká, jaký postup řešení úlohy vede obvykle k rychlému dosažení výsledku.
- Nezaručuje vždy zrychlení výpočtu.
- Heuristika se používá pro stanovení pořadí,
  - v jakém se zkoumají možné průchody stromem/grafem

- **Příklad:** úloha projít šachovým koněm celou šachovnicí  $\mathcal{N} \times \mathcal{N}$
- účinná heuristika: nejprve se navštíví ta dosud nenavštívená pole, z nichž bude nejméně možností dalšího bezprostředního pokračování cesty koně.
- urychlení na šachovnici  $\delta \times \delta$  až **stotisíckrát**.