

▲ **Def:** Necht' V , H_V , H_E jsou množiny, E je nějaká množina dvouprvkových a jednoprvkových podmnožin množiny V , $h_V : V \rightarrow H_V$, $h_E : E \rightarrow H_E$. Čtveřici (V, E, h_V, h_E) nazýváme *uzlově a hranově ohodnocený neorientovaný graf*.

▲ Jaké znáte vhodné reprezentace grafu? Existují podmínky, kdy je která reprezentace vhodnější? Jaká je jejich paměťová náročnost?

▲ Najděte vhodnou prezentaci dat, včetně zdůvodnění.

1. dopravní síť
2. organizace souborů na disku
3. počítačová síť
4. seznam kamarádů, kolegů, spolužáků. . .
5. skládání Rubikovy kostky
6. evidence studentů pro přístup do poslucháren při písemce
7. rozložení jednoduchých matematických výrazů typu $a + b * c$ na elementy
8. evidence jmen počítačů a odpovídajících IP adres
9. postup řešení her dvou hráčů (šachy, dáma, . . .)

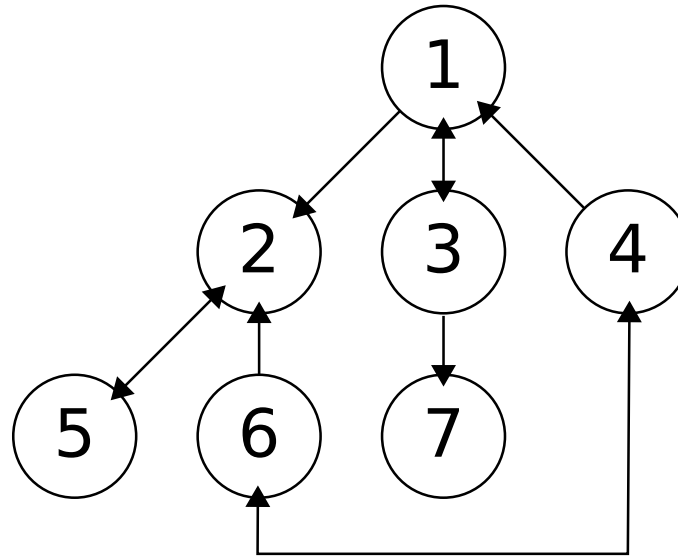
▲ Jaké způsoby řešení byste vybrali pro řešení následujících úloh?

1. stahování obsahu webu do určité hloubky
2. pokrytí brněnských sídlišť linkami MHD
3. hledání občerstvení v nejbližším okolí zadaného bodu
4. hledání cesty v bludišti
5. nalezení vlakového spojení z vesnice A do vesnice B

▲ Popište chování následujícího algoritmu. V kterém okamžiku dochází k ukončení rekurze?

```
for all (uzly v) do
  navstivene[v] := False;
dfs(v)
  zpracuj(v)
  navstivene[v] := True;
  for all (uzel i je nenavštíveným sousedem v)
    dfs(i)
```

- ▲ Na následujícím grafu ukažte procházení do hloubky počínaje uzlem č. 1.



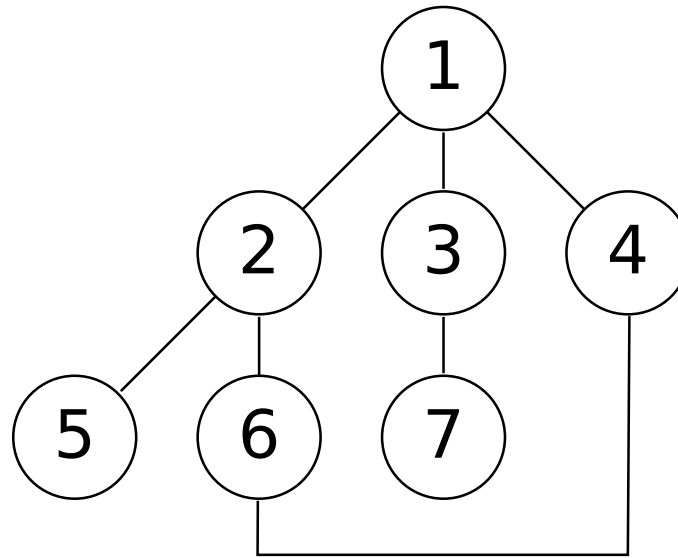
- ▲ Existuje vhodnější počáteční uzel pro projítí daného grafu?

▲ Jakým způsobem probíhá prohledávání grafu pomocí následujícího algoritmu?

```
procedure BFS(G: Graph, s: Node);
begin
  for each  $u \in G.V$  do begin {V = seznam vrcholu grafu G}
    navstivene[u] := False; d[u] := MaxInt; p[u] := nil;
  end;
  navstivene[s] := True; d[s] := 0; Enqueue(Fronta, s);
  while not(isEmpty(Fronta)) do begin
    u := HeadQ(Fronta);
    navstivene[u] := True;
    Fronta := Dequeue(Fronta); {zkrátí frontu o uzel u}
    for each {v je soused u} do
      if not(navstivene[v]) then begin
        navstivene[v] := True; d[v] := d[u] + 1; p[v] := u;
        Enqueue(Fronta, v);
      end;
    end;
  end;
end;
```

▲ Co je obsahem polí p[] a d[] před a po skončení hlavního cyklu programu?

▲ Na následujícím grafu ukažte procházení do šířky. Začněte v uzlu č. 1.



- ▲ Jaká je složitost předchozího algoritmu, je-li graf zadán pomocí seznamu sousedů?
- ▲ Jaké změny je třeba udělat, aby předchozí algoritmus fungoval i na orientovaných grafech?

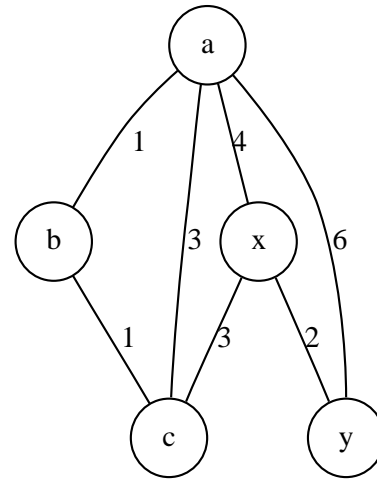
- ▲ Prohlédněte si ještě jednou a pozorně kód pro BFS. Dokážete navrhnout a odůvodnit co nejkratší změnu programu tak, aby místo BFS fungoval jako DFS?
- ▲ Změní se nějak složitost algoritmu? A pokud ano, tak jak?

▲ Dijkstraův algoritmus pro hledání nejkratších cest ze zadaného uzlu do všech ostatních uzlů

```
function Dijkstra(G, w, s)
  for each uzel v ∈ V[G] {Inicializace}
    d[v] := infinity
    p[v] := undefined
  d[s] := 0                                {vzdálenost z s do s}
  S := empty set
  Q := V[G]                                {množina všech nenavštívených uzlů}
  while not(IsEmptySet(Q)) do
    u := ExtractMin(Q)                      {vyber nejvhodnější hranu}
    S := S sjednoceno u                    {přidej do množiny navštívených}
    for each {hrana (u,v) je odchozí z u}
      if d[u] + w(u,v) < d[v]
        d[v] := d[u] + w(u,v)
        p[v] := u
```

▲ Navrhněte postup, jakým z cílové datové struktury získat nejkratší cestu.

▲ V následujícím grafu nalezněte nejkratší cestu z uzlu *a* do všech ostatních uzlů pomocí Dijkstraho algoritmu



▲ Jaká je podstata následujícího algoritmu?

```
type GraphMatrix = array [1..Max, 1..Max] of Integer;
procedure xy (var A:GraphMatrix; n:Integer);
begin
  for i := 1 to n do A[i,i] := 0;
  for k := 1 to n do
    for i := 1 to n do
      for j := 1 to n do
        A[i,j] := min (A[i,j], A[i,k]+A[k,j])
      end
    end
  end
end
```

▲ Navrhněte algoritmus, který k daným délkám nejkratších cest najde i cesty samotné.

▲ Pro graf G (zadaný maticí sousednosti) rozhodněte, jsou-li následující možnosti kostrou:

$$G = \begin{bmatrix} 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix}$$

1. $\{(4,5), (1,2), (3,1), (5,3)\}$
2. $\{(3,1), (3,2), (3,4), (3,5)\}$
3. $\{(5,3), (3,4), (4,5), (1,3)\}$
4. $\{(1,2), (1,3), (3,4), (4,5)\}$

▲ Algoritmus pro hledání kostry grafu – Kruskalův algoritmus:

```
type SpT = Set of Edges;
```

```
Component = Set of Nodes; {„vhodně reprezentovaná“ množina}
```

```
procedure kruskal (G:Graph; var A:SpT);
```

```
var W: Set of Component;
```

```
begin A :=  $\emptyset$ ;
```

```
W :=  $\emptyset$ ;
```

```
for  $v \in V$  do W :=  $W \cup \{v\}$ ;
```

```
seřadíme množinu E podle ohodnocení;
```

```
for  $(u, v) \in E$  {v pořadí od nejnižšího ohodnocení} do
```

```
if  $W_u \neq W_v$ 
```

```
{tj. u, v leží v různých komponentách}
```

```
then begin A :=  $A \cup \{(u, v)\}$ ;
```

```
sjednotíme obě tyto komponenty
```

```
end
```

```
end
```

▲ Popište, jak algoritmus funguje a v kterém okamžiku skončí.

▲ Další algoritmus pro hledání kostry v grafu – Primův algoritmus:

```
PRIM( $G, w, r$ )  
  for each  $u \in V[G]$   
    do  $d[u] := \infty$   
        $p[u] := \text{NIL}$   
 $d[r] := 0$   
 $Q := V[G]$   
while  $Q \neq \emptyset$   
  do  $u := \text{EXTRACT-MIN}(Q)$   
     for each  $v \in \text{Adj}[u]$   
       do if  $v \in Q$  and  $w(u, v) < d[v]$   
          then  $p[v] := u$   
               $d[v] := w(u, v)$ 
```

Jak tento algoritmus funguje?

▲ V následujícím grafu nalezněte minimální kostru nejdříve pomocí Kruskalova a poté pomocí Primova algoritmu.

