▲ **Def:** Nechť V, H_V, H_E jsou množiny, E je nějaká množina dvouprvkových a jednoprvkových podmnožin množiny V, $h_V : V \to H_V$, $h_E : E \to H_E$. 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. počítačová síť
3. organizace souborů na disku
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?

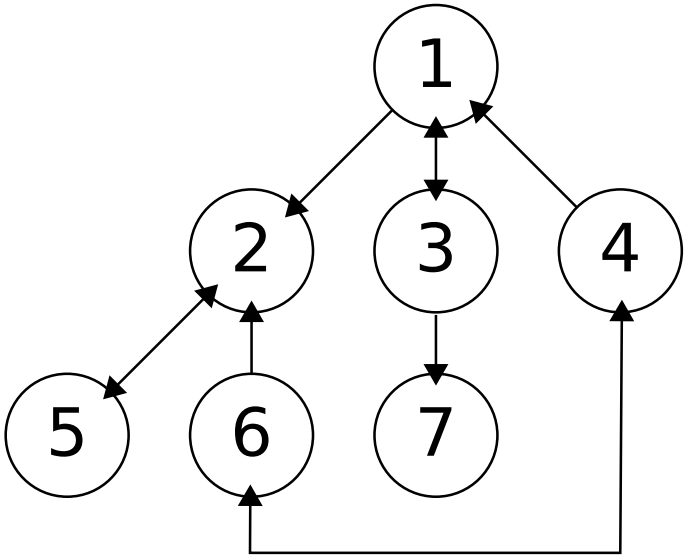
1. stahování obsahu webu do určité hloubky
2. pokrytí brněnských sídlišť linkami MHD
3. organizace na sebe navazujících činností
4. hledání občerstvení v nejbližším okolí zadaného bodu
5. hledání cesty v bludišti
6. nalezení vlakového spojení z vesnice A do vesnice B

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

▲ Prohledávání do hloubky, Depth-First Search

```
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.



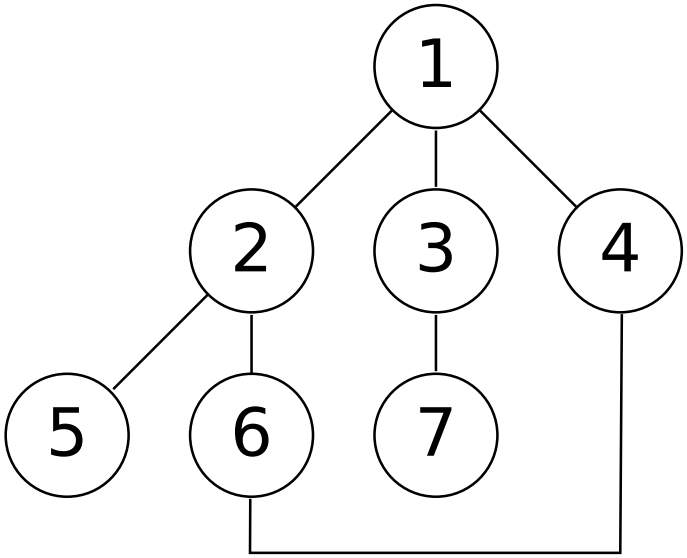
▲ Existuje vhodnější počáteční uzel pro projití daného grafu?

▲ Algoritmus prohledávání do šířky, Breadth-First Search:

```
procedure BFS(G: Graph, s: Node);
begin
  for each u ∈ 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;
```

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

▲ Na následujícím grafu ukažte procházení do šířky, počínaje uzlem č. 1.



▲ Jaká je složitost předchozího algoritmu, je-li graf zadaný 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?

▲ Poznáte algoritmus?

```
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
```

▲ Co daný algoritmus řeší?

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

▲ Dijsktrův algoritmus

```
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.

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

```
G = [ 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 ]
```

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:

```
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 := ∅;
        W := ∅;
        for v ∈ V do W := W ∪ {{v}};
        seřadíme množinu E podle ohodnocení;
        for (u, v) ∈ E {v pořadí od nejníže ohodnocené} do
           if W_u ≠ W_v
              {tj. u, v leží v různých komponentách}
               then begin A := A ∪ {(u, v)};
                          sjednotíme obě tyto komponenty
                    end
   end
```

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