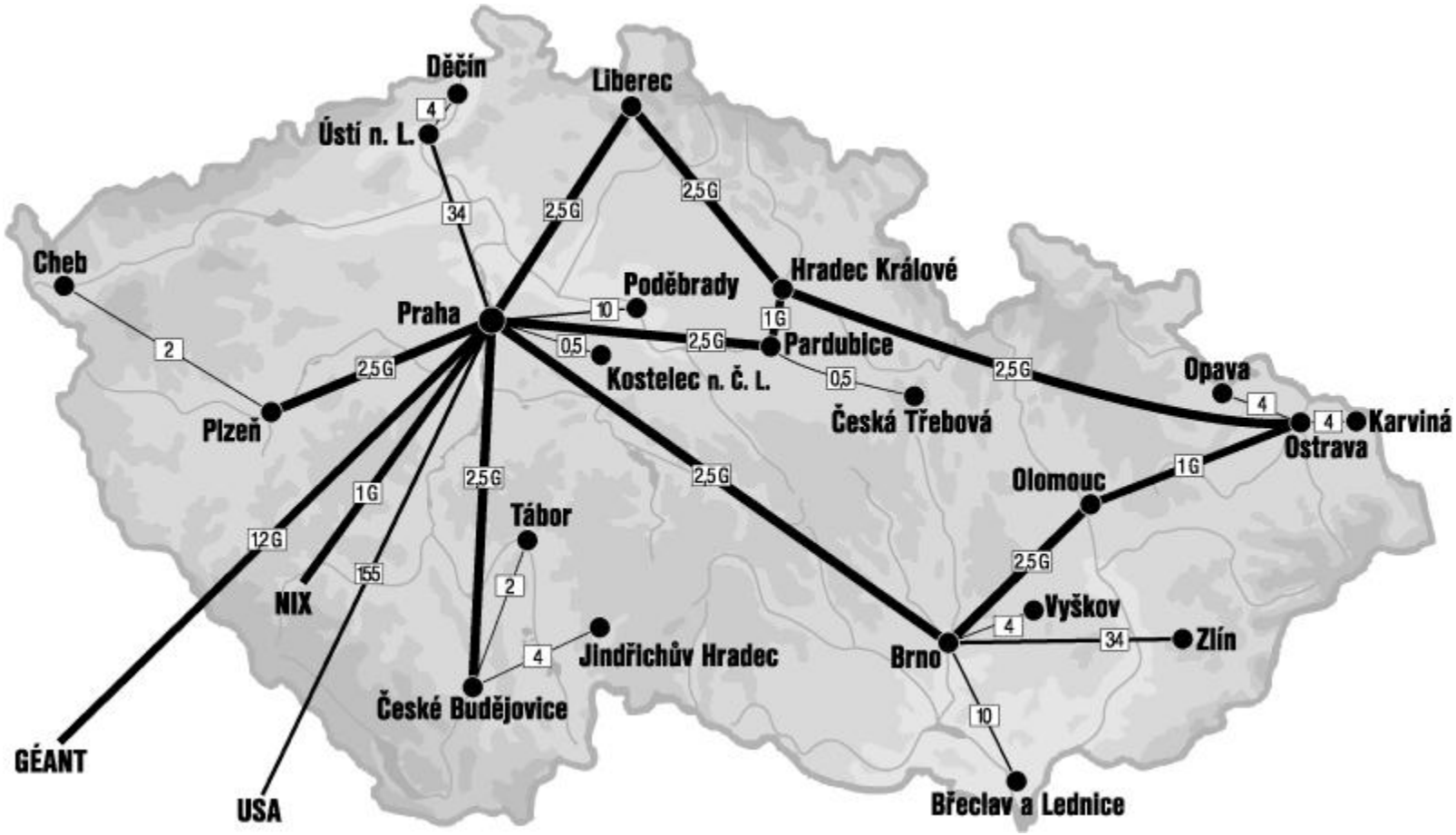
ADT GRAF
GRAPH

GRAF

Zdroj:

www.cesnet.cz



GRAF

G = (U, H)

U – možina uzlů |U| - počet uzlů

H – množina hran |H| - počet hran

U(G) – množina uzlů grafu G

H(G) – množina uzlů grafu G

REPREZENTACE GRAFU

Dva standardní způsoby v informatice:

- soubor (*collection* ne *file)* seznamů sousednosti
- matice sousednosti

Oba způsoby použitelné pro orientované i neorientované grafy

REPREZENTACE GRAFU

Je-li $|H| << |U|^2$ graf se nazývá *řídký*

obvykle je vhodnější použít seznam sousednosti

Je-li $|H| \sim |U|^2$ graf se nazývá *hustý*

obvykle je vhodnější použít matici sousednosti

také je-li nutno rychle zjistit existenci hrany

SEZNAM SOUSEDNOSTI

seznam sousedů pro každý uzel

```
grafss = array [ 1:|U| ] of  ukaz_sous;
ukaz_sous=^uzel;
uzel = record
          uzel: 1..|U|;
          dalsi_uzel: ukaz_sous
       end;
```

SEZNAM SOUSEDNOSTI

Je-li ***G*** orientovaný graf, délka všech seznamů sousednosti je |H|

Je-li ***G*** neorientovaný graf, délka všech seznamů sousednosti je 2|H|

Požadovaná paměť ~ (|U| + |H|)

SEZNAM SOUSEDNOSTI

Ohodnocené grafy

ohodnocení grafu ***G*** se nazývá funkce

w: H(G) -> R

Je-li ***(u,v)∈H(G), w(u,v)*** se uloží v uzlu ***v*** seznamu sousedů uzlu ***u***

Zjistit existenci hrany ***(u,v)*** znamená hledat uzel ***v*** v seznamu sousedů uzlu ***u***

MATICE SOUSEDNOSTI

matice $\boldsymbol{S = (s_{ij}),\ i,j = 1, \ldots, |U|}$

je-li $\boldsymbol{(i,j) \in H,\ s_{ij} = 1}$

jinak $\boldsymbol{s_{ij} = 0}$

grafms = **array** [1..|U|, 1..|U|] **of** 0..1;

Požadovaná paměť ~ $|U|^2$

MATICE SOUSEDNOSTI

pro neorientovaný graf je $\boldsymbol{S = S^T}$

pro ohodnocený graf

$\boldsymbol{s_{uv} = w(u,v)}$ je-li $\boldsymbol{(u,v) \in H(G)}$,

$\boldsymbol{s_{uv} \notin R}$ je-li $\boldsymbol{(u,v) \notin H(G)}$

pro neohodnocený graf lze prvky matice $\boldsymbol{S}$ uložit v bitech

PROHLEDÁVÁNÍ DO ŠÍŘKY
BREATH-FIRST SEARCH

Z vybraného uzlu *s* nalezneme všechny uzly ve vzdálenosti *k* od uzlu *s* před tím, než nalezneme uzly ve vzdálenosti *k+1*

Nalezený uzel obarvíme šedě a uložíme do fronty – začneme uzlem *s*

Po nalezení všech sousedů uzel obarvíme černě

Není-li fronta prázdná, pokračujeme dalším uzlem z fronty

PROHLEDÁVÁNÍ DO ŠÍŘKY
BREATH-FIRST SEARCH

Prvky pole seznamů sousedů uzlů rozšíříme o položky

barva – barva uzlu

d – vzdálenost od uzlu *s*

pred – číslo předcházejícího uzlu (0 = nemá předchůdce)

```
grafss = array [ 1:|U| ] of  ukaz_sous;
ukaz_sous=^uzel;
uzel = record
            uzel: 1..|U|;
            dalsi_uzel: ukaz_sous
       end;
grafss = array [ 1:|U| ] of record
                              zac:ukaz_sous;
                              barva:(b,š,č);
                              d:0..maxint;
                              pred:0..|U|
                            end;
```

```
BFS(GSS:grafss; s:1..|U|); …
F:fronta;    {ve frontě jsou šedé uzly}
x:ukaz_sous;  {prochází seznam sousedů}
u,v:1..|U|; …

for u:=1 to |U| do begin
   {na začátku jsou uzly bílé}
   if u<>s then with GSS[u] do begin barva:=b;
                                     d:=maxint;
                                     pred:=0
                                end
end;

{první navštívíme uzel s}
with GSS[s] do begin barva:=š;
                     d:=0;
                     pred:=0
end;
```

```
PRIDEJ(F,s);
while not PRAZDNA(F) do
 begin
    u:=UBER(F);
    x:=GSS[u].zac;
    while x<>NIL then begin
        v:=x^.uzel;
        with GSS[v] do
            if barva=b then begin
                                    barva=š;
                                    d:=GSS[u].d+1;
                                    pred:=u;
                                    PRIDEJ(F,v)
                                end;
        x:=x^.dalsi_uzel
    end;
    GSS[u].barva:=č;
 end;
```

PROHLEDÁVÁNÍ DO ŠÍŘKY
BREATH-FIRST SEARCH

- nalezneme všechny dosažitelné vrcholy z vybraného vrcholu *s*
- vypočteme vzdálenost (počet hran) k nalezeným vrcholům od *s*
- vytvoříme BFS strom všech dosažitelných uzlů, kterého kořen je *s*
- cesta z kořene do uzlu v BSF stromě je nejkratší

PROHLEDÁVÁNÍ DO ŠÍŘKY
BREATH-FIRST SEARCH

Čas výpočtu ~ |U|+|H|

BFS je pro orientované i neorientované grafy

BFS je základem dalších algoritmů

- Primův algoritmus minimální kostry
- Dijkstrův algoritmus minimální cesty

PROHLEDÁVÁNÍ DO HLOUBKY DEPTH-FIRST SEARCH

Hledáme, když je to možné, napřed do „hloubky“, tj. do větší vzdálenosti a nalezené uzly obarvíme šedě

Po průchodu všemi hranami z vyšetřovaného uzlu, uzel obarvíme černě a vrátíme se k jeho předchůdci nebo skončíme

PROHLEDÁVÁNÍ DO HLOUBKY DEPTH-FIRST SEARCH

Jestli zůstaly nenalezené uzly, jeden vybereme a postup opakujeme – vznikne les DFS stromů

Kromě položek barva a pred zavedeme dvě časové značky (jednotka „času“ = přechod na další uzel)

PROHLEDÁVÁNÍ DO HLOUBKY
DEPTH-FIRST SEARCH

n – čas nalezení uzlu, kdy je obarven šedě

k – čas konce procházení seznamu sousedů uzlu, kdy je uzel obarven černě

Hodnoty časových značek jsou mezi 1 a 2|U|.
Pro každý uzel *u* platí $n(u) < k(u)$.

```
ukaz_sous=^uzel;
uzel = record
            uzel: 1..|U|;
            dalsi_uzel: ukaz_sous
       end;
grafss' = array [ 1:|U| ] of record
                                zac:ukaz_sous;
                                barva:(b,š,č);
                                pred: 0..|U|;
                                n: 1..2*|U|;
                                k: 1..2*|U|
                             end;
```

```
DFS(GSS:grafss');…
cas:Integer;
x:ukaz_sous;  {prochází seznam sousedů}
u,v:1..|U|; …

 for u:=1 to |U| do
  with GSS[u] do begin barva:=b;
                       pred:=0
                 end;
cas:=0;
for u:=1 to |U| do
  if GSS[u].barva=b then DFS_NAVSTIV(u);
```

```
DFS_NAVSTIV(u:1..|U|);…
GSS[u].barva:=š;
cas:=cas+1;
GSS[u].n:=cas;
x:=GSS[u].zac;
while x<>NIL do begin v:=x^.uzel;
                      with GSS[v] do
                        if barva=b then begin
                          pred:=u;
                          DFS_NAVSTIV(v)
                        end;
                      x:=x^.dalsi_uzel
                end;
GSS[u].barva:=č;
cas:=cas+1;
GSS[u].k:=cas;
```

PROHLEDÁVÁNÍ DO HLOUBKY
DEPTH-FIRST SEARCH

Čas výpočtu ~ $|U|+|H|$

Vytvoří les DFS stromů

DFS využívají jiné algoritmy

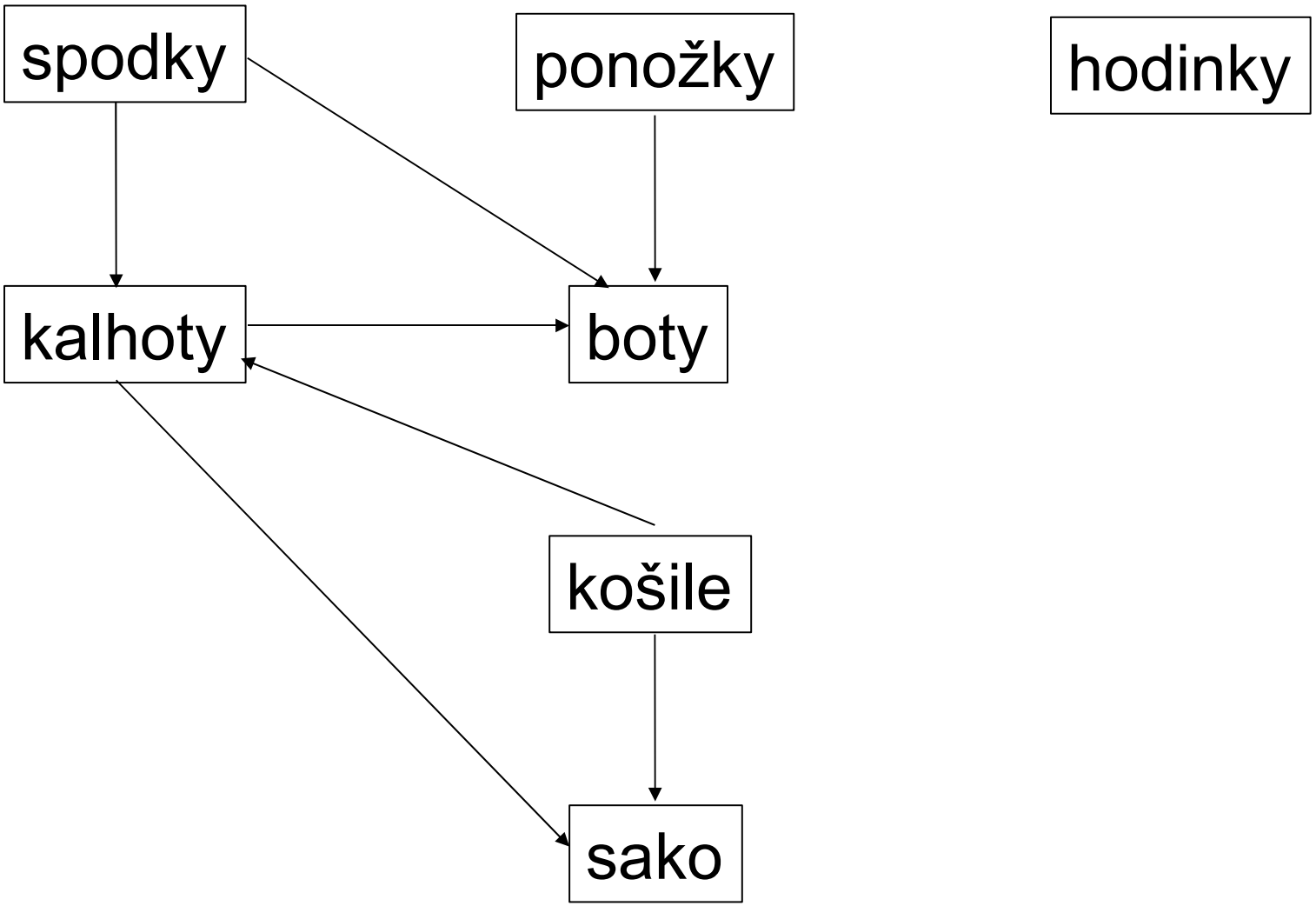
TOPOLOGICKÉ ŘAZENÍ

Máme množinu prvků, ve které je definované uspořádaní jen pro některé dvojice – např. činností v čase

Uspořádaná dvojice prvků (u,v) potom tvoří hranu orientovaného acyklického grafu

(*directed acyclic graph – DAG*)

OBLÉKÁNÍ



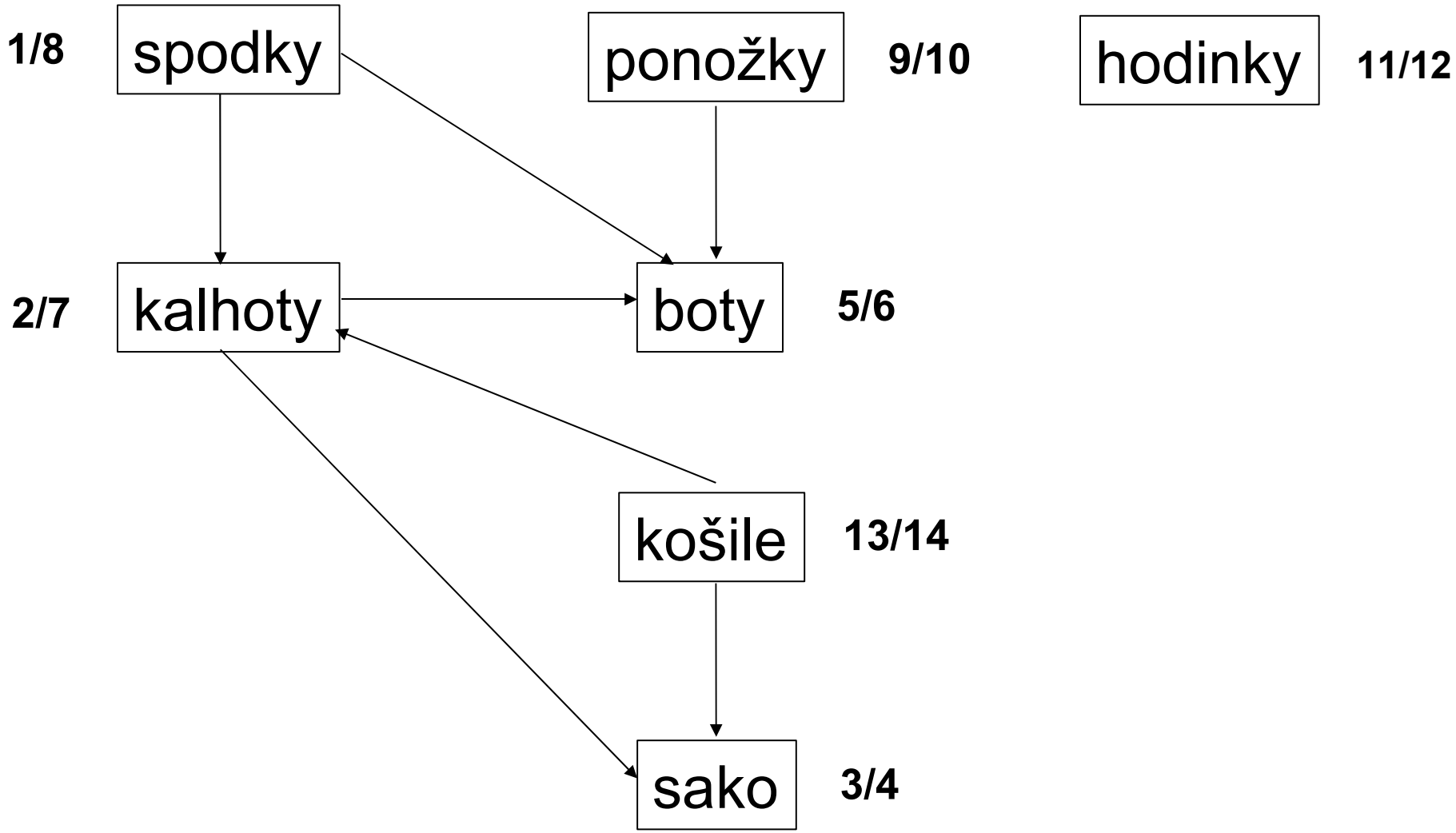
ALGORITMUS TOPOLOGICKÉHO ŘAZENÍ

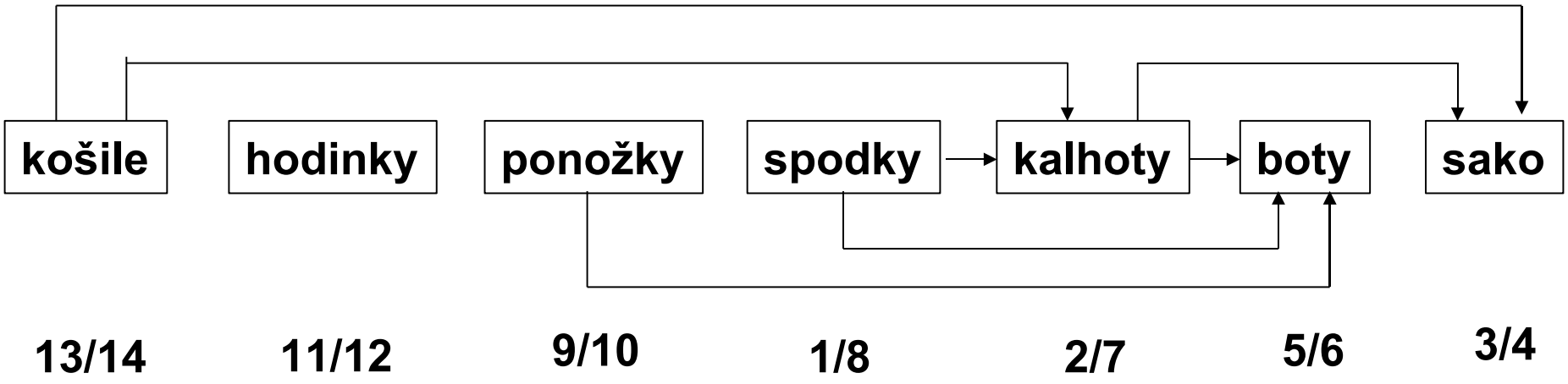
Úkol: Lineárně seřadit uzly (činnosti) tak, aby vzájemné pořadí dvojic zůstalo zachováno

Algoritmus:

6. Zavolej DSF
7. Když je uzel ukončen přidej ho do seznamu

OBLÉKÁNÍ


1/8
spodky
ponožky
9/10
hodinky
11/12
2/7
kalhoty
boty
5/6
košile
13/14
sako
3/4


košile
hodinky
ponožky
spodky
kalhoty
boty
sako
13/14
11/12
9/10
1/8
2/7
5/6
3/4