Meno: Tomáš Belan
Príklad 4**, kat.** O
Škola: ŠpMNDaG Teplická, Bratislava

Najprv je dobré si všimnúť jednu vec: kvôli pravidlu 5 musia všetky hrany záverečného stromu byť rozhádané, t.j. byť medzi tými v pôvodnom zozname hrán, ktorý je na vstupe.

Riešenie pozostáva z toho, určiť, kto bude starý koreň, a zo vstupného zoznamu hrán nejaké odstrániť, nech z grafu vznikne strom. Pri odstraňovaní si dávam pozor, nech je stále nejaká iná cesta, ako sa dostať z jedného konca hrany na druhú, takže medzi všetkými bodmi, kde existovala cesta na začiatku, bude existovať na konci. Výsledný strom nejde vytvoriť práve vtedy, keď sa vo vstupnom grafe nedá dostať ku všetkým bodom (ako napríklad v prvom príklade v zadaní).

Zaujímavá vec je, že je úplne jedno, kto bude starý koreň. Ak existuje vôbec nejaké riešenie, tak pre každý bod existuje riešenie, kde je koreňom. Len treba šikovne vybrať, ktoré hrany odstrániť, nech to s daným starým koreňom nespraví spor.

V mojej implementácii je starým koreňom vždy človek 1.

/* Algoritmus funguje takto: určím si nejaký ľubovoľný koreň, povedzme že bod 1. (Áno, ľubovoľný bod grafu môže byť starý koreň.) „Zdvihnem ho" a zdvihnem za neho celý graf. (Podobne ako pri Dijkstrovom algoritme.) Pozriem sa na tie hrany, čo od neho visia dolu. To budú podriadení. Ale keď medzi nimi nájdem nejaké koliečko (loop), tak nejakú hranu z koliečka, čo priamo od neho visí, prestrihnem. */

Keď chcem spraviť strom, musím dať preč loopy (koliečka). Ale nemôžem len tak hocijako ničiť hrany, kým tam žiadne nezostanú – pri každej zrušenej hrane sa treba uistiť, že sa zachová štvrté pravidlo (že je stále jeden nadriadeným druhého). Príklad 1: Zoberme si 5 ľudí, čo sa nemajú radi 1-2, 2-3, 3-4, 4-5, 5-1. Keby som zrušil hranu 3-4, tak v strome zostanú dve vetvy „visiace" od 1, jedna 1-2-3 a druhá 1-5-4, a pri vzťahu 3-4 neplatí pravidlo štyri. Príklad 2: V druhom príklade v zadaní je koreň 1 šéfom 5. Keby som zrušil 2-5 a 2-4, tak potom 5 je šéfom 3, a 3 je šéfom 2 a 4, čo ale nerieši vzťah 2-4.

Tento algoritmus na toto dokáže dať pozor. Graf držím za koreňa, zvyšok „visí" dolu. Vždy zoberiem hrany, ktoré idú od daného bodu „dolu". Keď je tam koliečko, tak to koliečko preseknem (jednu hranu zruším). Ale preseknem ho hneď hore, na vrchu, za ktorý to celé koliečko visí. Potom zvyšok koliečka spadne dolu. Ukážem na minulom príklade 1: ak preseknem hranu 1-2, tak vznikne priama lineárna hierarchia šéfov 1>5>4>3>2, a aj vzťah 1-2 je vyriešený, lebo 1 je nepriamym nadriadeným 2.

Riešenie formálnejšie: graf je uchovaný v zozname okolí 'susedia'. Toto je pole, ktoré obsahuje N rôzne veľkých polí, a každé vnorené pole susedia[i] je zoznam susedov daného bodu i. Toto využíva, že v Ruby nemusia polia mať pevnú veľkosť, ale rovnako dobre by to šlo implementovať bežným dvojrozmerným poľom, ako vo vzorovom riešení „Tentokrát o grafe".

Postupne, ako graf mením na strom, bodom určujem šéfov. Keď sa bodu určí šéf, tak sa ten šéf odstráni z jeho susedov (miesto jeho čísla sa na dané

miesto poľa susedia[i] dá nil, čiže NULL). Na konci je pole susedia vlastne zoznam podriadených.

Graf celý prehľadám po šírke. Keď prídem na nejaký bod, najprv nejaké loopové hrany zruším, a potom všetkým hranám, čo zostali, nastavím ten bod ako ich šéfa. Toto centrálne prehľadávanie po šírke má zložitosť O(N+K).

Teraz, ako sa zisťuje, ktoré hrany sú loopové a teda ich môžem zrušiť: proste skúsim tú hranu na chvíľu zrušiť, a ak stále existuje iná cesta medzi jednou a druhou stranou hrany, tak je tam loop a tá hrana má byť zrušená. Existenciu inej cesty zisťujem ďalším prehľadávaním po šírke, a keďže táto zrušiteľnosť sa overuje pre každú hranu, má to zložitosť O(K(N+K)). Takže celý program má zložitosť O((K+1)(N+K)), čo je, ak sa nemýlim, O(K(N+K)).

Tu je moja implementácia popísaného algoritmu v Ruby.

```
print "N="; N = readline.to_i; print "K="; K = readline.to_i
# susedia je pole N poloziek kde kazda polozka je prazdne pole
# susedia[i] je pole obsahujuce susedov vrchola i
# susedia[i].length je pocet susedov vrchola i
susedia = Array.new(N).collect{Array.new}
sef = Array.new(N)   # pole sef je podobne ako 'from' z "Tentokrat o grafe"
                     # (t.j. je to zaroven aj pole 'visited')
K.times do
  r = readline.split; x = r[0].to_i-1; y = r[1].to_i-1
  susedia[x] << y; susedia[y] << x   # pridam y na koniec pola susedia[x]
end

def mame_cestu(odkial, kam, susedia)
  queue = Array.new     # v Ruby je Array aj pole aj fronta
  visited = Array.new(N, false)
  visited[odkial] = true
  return true if odkial == kam
  queue.push(odkial)
  while not queue.empty?
    v = queue.shift
    susedia[v].each do |sused|
      if sused != nil and visited[sused] == false
        queue.push(sused)
        visited[sused] = true
        return true if sused == kam  # prisli sme tam
      end
    end
  end
  return false
end

# teraz ideme spravit z toho celeho strom
sef[0] = -1   # stary koren bude clovek 1 (pri cislovani od nuly clovek 0)
queue = Array.new
queue.push(0)
while not queue.empty?
  ja = queue.shift
  # teraz sa pole susedov postupne zmeni na pole podriadenych
  # najprv odstranim tych, co nie su priami podriadeni
  susedia[ja].length.times do |i|
    # on je cislo suseda, i je jeho miesto v poli susedia[ja]
    on = susedia[ja][i]
    next if on == nil
    susedia[ja][i] = nil   # docasne zrusime tu priamu cestu ja–on
    # (toto zrusilo len ja->on, tu on->ja mozem nechat lebo mame_cestu aj tak
    # skonci akonahle sa ku bodu on dostane a susedov bodu on odignoruje)
    if mame_cestu(ja, on, susedia)   # ak stale existuje ina cesta
```

```
      # tak vztah ja--on zrusime (aj jeho druhu polovicu on->ja)
      # zistim, na ktorom mieste sa nachadzam ja vnutri susedia[on]
      j = susedia[on].index(ja)
      susedia[on][j] = nil
    else
      susedia[ja][i] = on    # obnovim tu docasne zrusenu cestu
    end
  end
  # teraz si podriadim tych, co zostali
  susedia[ja].length.times do |i|
    on = susedia[ja][i]
    next if on == nil
    # odteraz nie som jeho sused, ale som jeho sef
    j = susedia[on].index(ja)
    susedia[on][j] = nil  # vymazem sa zo susedia[on]
    sef[on] = ja
    # pridam ho do fronty, nech si aj on nejakych podriadi
    queue.push(on)
  end
end

# kontrola - maju vsetci sefa? (cize presli sme vsetky body?)
N.times { |i| if sef[i] == nil then puts "nejde"; exit end }

N.times do |i|
  print "#{i+1}:"
  susedia[i].each { |sused| print " #{sused+1}" if sused != nil }
  print "\n"
end
```