NTIN060 | Homework 05/5

Milan Wikarski (milan@wikarski.sk)

Zadanie

6. Revoluce u Bobří řeky. Chcete zavést KaraNET i ve své domovské vesnici Kravařích. Síť možných spojení je bohužel trochu řidší a spojení odpovídá rovinnému grafu. Jak dlouho bude trvat nalézt páteřní síť v tomto rozložení?

Riešenie a dôkaz správnosti

Majme súvislý rovinný graf `G = (V, E)`, zobrazenie `E -> w`, ktoré každej hrane priradí nejakú váhu. Hľadáme minimálnu kostru grafu `G`.

Pre nájdenie minimálnej kostry grafu `G` použijeme Jarníkův algoritmus. Vyberieme si nejaký vrchol `v<sub>0</sub>` (náhodne) a začneme so stromom `T = ({v<sub>0</sub>}, {})`, ktorý obsahuje iba vrchol `v<sub>0</sub>` a žiadne hrany.

Stav vrcholov

Každý vrchol `v ∊ V` sa bude nachádzať v jednom z troch stavov:

1. **IN** - ak `v` patrí do T.
2. **NEIGHBOR** - ak `v` nepatrí do T a zároveň existuje hrana `e={u, v} ∊ G` medzi `T` a zvyškom grafu `G` (tzn. `u ∊ V(T), v ∊ V(G) \ V(T)`).
3. **OUT** - v ostatných prípadoch.

Tieto stavy budú uložené v poli `state`.

Ohodnotenie vrcholov

Každý vrchol `v ∊ V` budú mať ohodnotenie `eval[v]` - to odpovedá:

- `+∞`, ak `state[v] = OUT`.
- najnižšej váhe z hrán, ktoré vedú z vrchola `v` do nejakého vrchola `u ∊ V(T)`, ak `state[v] = NEIGHBOR`.
- ak `state[v] = IN`, hodnota `eval[v]` nás nezaújíma a nebude uložená

Ohodnotenie hrán si uložíme do minimálnej haldy, ktorá bude mať operácie ExtractMin a Decrease.

Predchodcovia

Ďalej si budeme držať pole predchodcov `pred[v]`. Bude platiť, že ak `eval[v] < +∞`, potom hrana `e = {v, pred[v]}` je hrana s najnižším ohodnotením, ktorá vedie z vrchola `v` do nejakého vrchola `u ∊ V(T)`.

Beh algoritmu

V každom kroku algoritmu si vyberieme vrchol `v ∊ V` s najnižším ohodnotením `eval[v]` z haldy a pripojíme ho ku grafu `T` hranou `e = {v, pred[v]}`. Ďalej prepočítame hodnotu `eval[u]` pre všetkých susedov `u` vrchola `v`, ktorí nepatria do `T` (konkrétne ju znížime, ak hrana medzi vrcholmi `u` a `v` má nižšiu váhu, ako je hodnota `eval[u]`).

Správnosť

Tu je dôležité, že ani hrana `e`, ani vrchol `v` nepatria do grafu `T`, a teda ku `T` pripájame listy. Z toho dostávame, že graf `T` bude počas celého behu algoritmu strom.

Algoritmus sa po najviac `n` krokoch zastaví, keďže v každom kroku sme ku grafu `T` pridali jeden vrchol.

V každom kroku vyberáme najľahšiu hranu elementárneho rezu medzi stromom `T` a zvyškom grafu. Všetky vybrané hrany teda ležia v minimálnej kostre, a teda strom `T` je minimálnou kostrou grafu `G`.

Algoritmus

```
ALGORITMUS: PravdepodobnostUspechuPrenosu
---------------------------------------------------------------
 INPUT: graf G = (V, E), zobrazenie E -> w
OUTPUT: Strom T, ktorý je minimálnou kostrou grafu G

for ∀ v ∈ V:
  eval[v] <- +∞
  pred[v] <- null
  state[v] <- OUT
v0 <- random vertex from V
eval[v0] <- 0
state[v0] <- NEIGHBOR
while (exists v ∈ V such that state[v] = NEIGHBOR):
  v <- ExtractMin(pred)                         ⊲ Vyberieme vrchol s najnižšou hodnotou eval z ha
  state[v] <- IN
  if (pred[v] is not null):                     ⊲ Nutná kontrola, pretože pred[v0] is null
    V(T) <- V(T) ∪ {v}
    E(T) <- E(T) ∪ {v, pred[v]}                 ⊲ Pridáme list ku stromu T
  for (∀ u; u is neighbour of v):               ⊲ Každý vrchol u spojený hranou s vrcholom v
    if (state[u] != IN and eval[u] > w({u, v})):
      state[u] <- NEIGHBOR
      eval[u] <- w({u, v})
      pred[u] <- v
return T
```

Časová a priestorová zložitosť

Označme si `n = |V|` počet vrcholov a `m = |E|` počet hrán.

Časová zložitosť

Jarníkův algoritmus nájde minimálnu kostru pri použití halde v čase `O(m \* log(n))`. Pre rovinné grafy platí `m ≤ 3n - 6`, z čoho dostávame `O(m \* log(n)) = O((3n - 6) \* log(n)) = O(n \* log(n))`

Priestorová zložitosť

Algoritmus pracuje s pomocnými poľami `pred` a `state` a haldou `eval` ktoré majú dĺžku `n`. Ďalej potrebujeme pamäť na ukladanie kostry `T`, čo je najvaic `O(n + m)`. Z hore spomenutého odhadu ale máme `O(n + m) = O(n + 3n - 6)`. Dokopy to všetko je `O(3n + n + 3n - 6) = O(n)`.