Problema Cangrena - Soluție Oficială

Autor: Student Tamio Nakajima-Vesa

Notații:

- Vom modela orașul TownsVille ca pe un graf **G**, unde intersecțiile sunt noduri (mulțimea **V**) și străzile muchii (mulțimea **E**).
- Pentru un nod `v ∈ V`, numim „valoarea” nodului **v** coeficientul de aglomerație din enunț (**`val[v]`**).
- Pentru o muchie `e ∈ E`, numim „costul” muchiei *e* modulul diferenței dintre valorile nodurilor incidente (**`cost[e]`**).
- Numim un mod de a asocia valori nodurilor inițial nedefinite „atribuire”.
- Numim „costul” unei atribuiri ca fiind costul maxim al vreunei muchii, după atribuirea respectivă. Astfel, pentru o atribuire costul este `max{cost[e] / e ∈ E}`.

Observație: Dacă există o atribuire de cost **C** pentru graful dat, va exista o atribuire pentru orice cost **`C'`** mai mare sau egal cu **C**.

Demonstrația observației este trivială: luăm o atribuire de cost **C**, ne alegem o muchie de cost **C** din graf, și îi creștem „artificial” costul cu **`C' - C`** (adunăm **`C' - C`** la valoarea nodului cu valoare mai mare din cele două incidente muchiei). Această observație ne permite să căutăm binar costul minim posibil al unei atribuiri: fixăm un cost **C** și încercăm să construim o atribuire de cost **C**. Dacă reușim, putem căuta un cost mai mic. Altfel, niciun cost mai mic nu poate aduce o atribuire, deci suntem obligați să alegem mai departe un cost mai mare.

Mai avem de rezolvat următoarea problemă: pentru un cost fixat **C**, trebuie să construim o atribuire validă de cost **C**, sau să semnalăm că nu se poate.

Soluție în $O(M * \log_2 N * \log_2 VMAX)$

Fie $a_1, a_2, .. a_k \in V$ mulțimea de noduri care aveau costul fixat inițial. Observăm că orice vecin al lui a_1 va putea avea valoarea asociată maxim $val[a_1] + C$. Extinzând această observație, vecinii vecinilor săi vor putea avea valoarea asociată maxim $val[a_1] + 2C$. În general, nodul a_1 va impune fiecărui nod **v** o restricție de forma $val[x] <= val[a_1] + C * dist(a_1, x)$, unde $dist(a_1, x)$ = distanța (măsurată în număr de muchii) de la a_1 la **x**. Dacă scriem restricțiile impuse de fiecare nod din $\{a_1, a_2...a_k\}$ asupra lui **x**, vom obține restricția generală

$$val[x] \le min\{val[a_i] + C * dist(a_i, x) / \text{i între 1 și k}\}.$$

Notăm această limită superioară a lui **`val[x]`** ca **`MAX[x]`**.

În mod similar vom defini valoarea **`MIN[x]`**.

Putem folosi algoritmul lui Dijkstra pentru a afla aceste limite superioară/inferioară a fiecărui nod: atribuim fiecărei muchii din graf costul **C**, și pornim algoritmul cu nodurile $a_1, a_2...a_k$ în coada de priorități cu costurile $val[a_1], val[a_2]...val[a_k]$. La sfârșitul algoritmului, distanța spre fiecare nod va fi **`MAX[nod]`**. Pentru a afla valorile **`MIN`**, singurele lucruri care se schimbă sunt costurile muchiilor din **C** în **-C** și direcția cozii de priorități (din **`min-priority-`**

queue în **max-priority-queue**). Se observă foarte ușor acum că există soluție dacă și numai dacă **MIN[x] ≤ MAX[x], ∀ x ∈ V**. O posibilă atribuire este ca fiecărui nod să i se asocieze valoarea **MAX[x]**. Complexitatea totală este $O(M\log_2 N \log_2 VMAX)$.

Reducere la $O(M * \log_2 VMAX + N * \log_2 N)$

Din particularitățile grafului pe care trebuie rulat algoritmul lui Dijkstra, coada cu priorități poate fi implementată mai bine decât cu un heap: putem înlocui heap-ul cu două cozi, deoarece toate muchiile au același cost: costurile inițiale pot fi ordonate crescător și descrescător înainte de căutarea binară în $O(N * \log_2 N)$, pentru ca apoi să le inserăm direct în prima din cele două cozi. Apoi, extinderea prin muchii din Dijkstra va adăuga noduri doar în cea de a doua coadă, care va rămâne mereu ordonată crescător (algoritmul este identic cu cel de la Coduri Huffman). *Observație*: Algoritmul Bellman-Ford va merge tot în O(M) pe acest graf.

Reducere la un singur algoritm de căutare

Observăm că în soluție, atribuirea de valori pune doar **MAX[v]** în orice nod **v**. Natural, ne vine ideea să nu mai calculăm deloc valorile **MIN**. Se poate demonstra destul de ușor că restricția **MIN[x] ≤ MAX[x], ∀ x ∈ V** este echivalentă cu **MAX[a_i] = val[a_i], i** între **1** și **k**. Astfel, avem nevoie doar de valorile **MAX**, deci avem nevoie doar de un singur algoritm de parcurgere.