Prim. Bellman-Ford

BUZATU GIULIAN & NIȚĂ ALEXANDROS

Algoritmul lui Prim

Vom considera aceeași problemă ca în cazul algoritmului lui Kruskal: dându-se un graf ponderat cu n noduri și m muchii, aflați un arbore parțial de cost minim ce conține toate nodurile din graf.

Algoritmul lui Prim

Vom considera aceeași problemă ca în cazul algoritmului lui Kruskal: dându-se un graf ponderat cu n noduri și m muchii, aflați un arbore parțial de cost minim ce conține toate nodurile din graf.

Hint 1: Să presupunem că avem un arbore format din k noduri, $k \leq n$. Cum adăugăm următorul nod în arbore?

Algoritmul lui Prim

Vom considera aceeași problemă ca în cazul algoritmului lui Kruskal: dându-se un graf ponderat cu n noduri și m muchii, aflați un arbore parțial de cost minim ce conține toate nodurile din graf.

Hint 1: Să presupunem că avem un arbore format din k noduri, $k \leq n$. Cum adăugăm următorul nod în arbore?
Hint 2: Intuitiv, vom folosi muchia de cost minim ce leagă un nod care nu este în arbore de arborele deja format pentru a minimiza costul total.

Algoritmul lui Prim

Descriere algoritm:

1. Inițial arborele este vid și putem adăuga orice nod.

2. Adăugăm un nod în arborele format folosind muchia de cost minim ce unește un nod din arbore de alt nod care nu este în arbore.

3. Repetăm pasul 2 până adăugăm toate nodurile în arbore, adică de $n - 1$ ori.

Algoritmul lui Prim

Descriere algoritm:

1. Inițial arborele este vid și putem adăuga orice nod.
2. Adăugăm un nod în arborele format folosind muchia de cost minim ce unește un nod din arbore de alt nod care nu este în arbore.
3. Repetăm pasul 2 până adăugăm toate nodurile în arbore, adică de $n - 1$ ori.

Pasul 2 este descris vag, deoarece există mai multe metode de a-l efectua, ducând la complexități diferite.

Algoritmul lui Prim

Metoda 1

Vom parcurge toate nodurile din arborele creat și, pentru fiecare nod din arbore, toate nodurile din graf care nu fac parte din arbore. Apoi alege minimul muchiilor cu un capăt în arbore și celălalt în afara arborelui. Această implementare are complexitatea $O(nm)$, deoarece pentru fiecare nod din arbore vom verifica toate muchiile adiacente. În cazul unui graf dens, această complexitate tinde spre $O(n^3)$.

Implementare: https://infoarena.ro/job_detail/3194695?action=view-source

Algoritmul lui Prim

Metoda 2

Vom reține într-un vector distanța minimă din arbore la un nod din afară. Când trebuie să decidem prin ce muchie adăugăm noul nod, vom folosi valoarea minimă din vectorul de distanțe minime și după adăugarea lui, vom actualiza acest vector. Această abordare conduce la complexitatea de $O(n^2)$, deoarece căutăm un minim într-un vector de n ori. Această soluție este de preferat pentru grafuri dense.

Implementare: https://infoarena.ro/job_detail/3194704?action=view-source

Algoritmul lui Prim

Metoda 3

Vom folosi un $std :: priority\_queue$ pentru a determina minimul. Apoi parcurgem lista de adiacență a nodului găsit pentru a actualiza distanțele. Conduce la o complexitate de $O(m \log n)$ deoarece pentru fiecare muchie actualizată trebuie să o adaug în $std :: priority\_queue$.

Implementare: https://infoarena.ro/job_detail/3194710?action=view-source

Problemă

Să rezolvăm problema: https://infoarena.ro/problema/cablaj

Problemă

Să rezolvăm problema: https://infoarena.ro/problema/cablaj

Soluție: Problem constă în aplicarea algoritmului lui Prim. Graful este complet, deci, implicit, este și dens, deci metoda a 2-a este cea pe care vrem să o folosim.

Implementare ($O(n^2)$): https://infoarena.ro/job_detail/3171297?action=view-source
Implementare ($O(m \log n)$): https://infoarena.ro/job_detail/3194713?action=view-source

Observație: Se poate observa o diferență de timp și de memorie între cele două surse.

Problemă

Să rezolvăm problema: https://infoarena.ro/problema/rusuoaica

Problemă

Să rezolvăm problema: https://infoarena.ro/problema/rusuoaica

Hint: Cum putem aplica cele trei operații eficient pe graful nostru?

Problemă

Să rezolvăm problema: https://infoarena.ro/problema/rusuoaica

Soluție: Aplicăm algoritmul lui Kruskal, dar folosindu-ne de operațiile suplimentare pe care ni le oferă problema. Dacă putem lua o muchie în arbore, avem două cazuri: aceasta are costul strict mai mare decât A, atunci putem să o înlocuim cu o muchie de cost A, altfel, pur și simplu luăm muchia în arbore. În ambele cazuri, actualizăm corespunzător răspunsul. Dacă o muchie nu este aleasă, atunci scădem costul ei din răspuns. La final, este posibil să nu avem doar un APM, ci o pădure de „APM-uri”. Deci conectăm arborii între ei cu muchii de cost A.

Implementare: https://infoarena.ro/job_detail/3170000?action=view-source

Problemă

Să rezolvăm problema: https://infoarena.ro/problema/ninjago

Problemă

Să rezolvăm problema: https://infoarena.ro/problema/ninjago

Hint: Încercați să vedeți care muchii sunt prioritare.

Problemă

Să rezolvăm problema: https://infoarena.ro/problema/ninjago

Soluție: Pentru început, să constatăm că cerința 1 nu cere nimic legat de distanță. Deci vom parcurge graful printr-un DFS/BFS prin muchiile în care nu avem E. Pentru cerințele 2 și 3 vom folosi Algoritmul lui Kruskal, dar trebuie să vedem cum să facem sortarea. Vom sorta în funcție de numărul de E-uri din cod, iar, dacă acestea sunt egale, în funcție de costul muchiei. Astfel, minimizăm numărul de E-uri (cel mai important) și distanța parcursă (mai puțin important).

Implementare: https://infoarena.ro/job_detail/3170250?action=view-source

Bellman-Ford

Avem aceeași problemă ca în cazul Algoritmului lui Dijkstra: găsiți distanța de la un nod u la toate celelalte noduri într-un graf orientat cu n noduri și m muchii.

Doar un mic detaliu în plus: costurile muchiilor pot fi negative.

Bellman-Ford

În primul rând, să observăm că există posibilitatea să nu existe o distanță minimă de la un nod la toate celelalte. Acest lucru se întâmplă în cazul unui ciclu în care costul muchiilor este negativ. Numim acest caz ciclu de cost negativ.

Algoritmul pe care îl vom prezenta poate să detecteze aceste cazuri și chiar să deducă ciclurile.

Bellman-Ford

Algoritmul funcționează asemănător cu Algoritmul lui Dijkstra. Încercăm să folosim muchia (a, b) cu cost c pentru a relaxa distanța până la b.

Se poate dovedi matematic că aplicând această metodă de n – 1 ori vom determina distanțele minime.

Pentru determinarea ciclurilor negative parcurgem muchiile și dacă găsim încă o relaxare, înseamnă că am găsit unul.

Bellman-Ford

Descriere algoritm:

1. Parcurgem lista muchiilor.
2. Pentru fiecare muchie verificăm dacă poate să relaxeze drumul.
3. Repetăm pașii 1 și 2 de n – 1 ori.
4. Parcurgem muchiile și dacă găsim o relaxare, ne oprim.
   Am găsit un ciclu negativ.

Complexitate: O(nm)

Bellman-Ford

Să rezolvăm problema: https://infoarena.ro/problema/bellmanford

Implementare: https://infoarena.ro/job_detail/3182394?action=view-source

Bellman-Ford

Ne putem folosi de faptul că nu toate încercările noastre de relaxare au succes. Creăm o coadă de priorități care conține doar nodurile pentru are am găsit deja răspunsul dar care pot să își relaxeze vecinii. La fiecare relaxare, adăugăm nodul în coadă.

Implementare: https://infoarena.ro/job_detail/3182423?action=view-source

Temă

- https://codeforces.com/problemset/problem/1468/J
- https://open.kattis.com/problems/shortestpath3
- https://cses.fi/problemset/task/1197
- https://cses.fi/problemset/task/1673
- https://infoarena.ro/problema/ciclu

Probleme suplimentare

- https://infoarena.ro/problema/ciob
- https://oj.uz/problem/view/APIO17_merchant

Lectură suplimentară

- https://cp-algorithms.com/graph/mst_prim.html
- https://cp-algorithms.com/graph/bellman_ford.html
- https://usaco.guide/CP2.pdf#page=109