BFS 0-1. Dijkstra. Dial

BUZATU GIULIAN & NIȚĂ ALEXANDROS

Problemă

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/camionas.

Problemă

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/camionas.

Hint: Extindem algoritmul BFS.

Problemă – BFS 0-1

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/camionas.

Soluție: Se realizează o extindere a algoritmului BFS. Putem reține în loc de greutatea fiecărei muchii, dacă aceasta este sau nu mai grea decât greutatea camionașului. Dacă aceasta este mai ușoară, reținem costul 1 pe muchia respectivă, altfel reținem costul 0. Acum, în loc de o coadă obișnuită, putem folosi un deque pentru implementarea algoritmului BFS. Nodurile în care intrăm folosind o muchie de cost 1 le adăugăm la capătul din dreapta, iar celelalte la capătul din stânga. Astfel, se păstrează proprietatea că algoritmul BFS ne oferă lanțul de lungime minimă de la un nod sursă la oricare altul.
Implementare: https://infoarena.ro/job_detail/3158978?action=view-source

Problemă

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/padure.

Problemă – BFS 0-1

Să rezolvăm următoarea problemă:https://infoarena.ro/problema/padure.

Soluție: Aplicăm algoritmul BFS 0-1 pe matricea noastră. Pornim din punctul de start, dacă vrem să mergem într-o celulă cu valoarea egală cu celula în care ne aflăm, atunci avem o muchie de cost 0 în „graful nostru”, iar dacă celula în care vrem să mergem are o valoare diferită, avem o muchie de cost 1.

Implementare:https://infoarena.ro/job_detail/3163844?action=view-source

Problemă

Am văzut data trecută cum putem să aflăm distanța minimă dintre un nod și celelalte noduri folosindu-ne de algoritmul BFS, dar algoritmul acesta obține rezultatul corect doar dacă toate muchiile sunt „egale”, i. e. au același cost. Ce facem dacă avem grafuri ponderate, unde costurile muchiilor sunt diferite?

Mai concret, dându-se un graf ponderat(cu ponderi nenegative), cum aflăm distanța minimă de la un nod la toate celelalte?

Algoritmul lui Dijkstra

Ideea principală este următoarea: dacă știm distanța minimă de la sursa s la un nod u, putem afla o posibilă distanță minimă de la sursă la toate nodurile adiacente lui u.

E important să înțelegem că există mai multe astfel de potențiale distanțe minime. Putem folosi fiecare nod adiacent cu cel pentru care încercăm să calculăm distanța pentru a încerca să o micșorăm.

Algoritmul lui Dijkstra

Descriere algoritm:

1. Marcăm distanța de la nodul sursă la el însuși cu 0 și restul distanțelor cu infinit.
2. Calculăm minimul dintre distanțele până la nodurile pentru care știm sigur că am găsit distanța minimă.
3. Relaxăm distanțele de la nodul găsit la pasul 2 la vecinii lui.
4. Repetăm pașii 2 și 3 de până nu mai putem îmbunătății nicio distanță.

Algoritmul lui Dijkstra - Complexitate

Complexitatea de spațiu: O(n) (folosim un vector să salvăm distanțele minime de la sursă la fiecare nod)
Complexitatea de timp diferă în funcție de modul în care calculăm minimul de la pasul 2. Putem parcurge vectorul de distanțe în timp liniar, rezultând într-o complexitate de O(n^2). Totuși putem mai bine. Folosind un min-heap (o structură de date arborescentă în care obținerea minimului se face in O(1), dar inserția și ștergerea se fac în O($log\ n$)). Se obține complexitatea O($m \log m$). Această structură de date este implementată în STL prin container-ul $std :: priority\_queue$. Se poate folosi, de asemenea, $std :: set$, o structură asemănătoare, dar cu performanțe diferite.

Algoritmul lui Dijkstra - Implementare

Problemă (implementare): https://infoarena.ro/problema/dijkstra

Soluții: 1. https://infoarena.ro/job_detail/3167751 ($std :: priority\_queue$ cu negative)

2. https://infoarena.ro/job_detail/3167752 ($std :: priority\_queue$ cu $std :: greater$)

3. https://infoarena.ro/job_detail/3167759 ($std :: priority\_queue$ cu funcție de comparare)

4. https://infoarena.ro/job_detail/3167750 ($std :: set$)

Algoritmul lui Dijkstra – Observație

Atenție!

Algoritmul lui Dijkstra oferă răspunsul corect doar când costurile muchiilor sunt numere nenegative. Acest lucru se datorează faptului că algoritmul este unul de tip Greedy, ceea ce înseamnă că odată ce algoritmul consideră că a găsit o distanță minimă pentru un nod, nu mai verifică.

Algoritmul lui Dijkstra

Să rezolvăm următoarea problemă: https://codeforces.com/problemset/problem/20/C.

Algoritmul lui Dijkstra

Să rezolvăm următoarea problemă: https://codeforces.com/problemset/problem/20/C.

Soluție: Trebuie să găsim drumul de lungime minimă între nodul 1 și nodul n. Vom folosi un vector unde $pred[i]$ este nodul pe unde am găsit drumul. Când găsim o distanță mai mică, vom actualiza și acest vector.

Implementare: https://codeforces.com/contest/20/submission/232325862

Algoritmul lui Dijkstra

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/catun.

Algoritmul lui Dijkstra

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/catun.

Hint: Seamănă cu o problemă de data trecută unde aplicam BFS.

Algoritmul lui Dijkstra - multisource

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/catun.

Soluție: Vom aplica algoritmul lui Dijkstra, dar, în loc să punem 0 în vectorul de distanțe doar pentru un nod sursă, vom punem pentru toate fortărețele, pe care, de asemenea, le vom adăuga în heap. Rezultatul îl vom stoca într-un vector în care $v[i]$ este cea mai apropiată fortăreața de cătunul i și îl vom actualiza când relaxăm distanța la nodul i.

Implementare: https://infoarena.ro/job_detail/3167757?action=view-source

Problemă

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/car.

Problemă

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/car.

Hint: Putem extinde algoritmul BFS 0-1 pentru mai multe costuri?

Problemă – Algoritmul lui Dial

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/car.

Soluție: Putem observa că operațiile de tip 3 și 4 se reduc, de fapt, la o combinație de operații de tip 1 și 2. Așadar, trebuie să vedem cum facem să modelăm aceste operații. Putem să ne folosim de principiul de la algoritmul BFS 0-1. Știm că distanțele din coadă vor fi diferite prin cel mult valoarea 2. Deci, putem ține 3 cozi, iar de fiecare dată când nu mai avem elemente într-o coadă, mergem la următoarea, în mod ciclic.

Implementare: https://infoarena.ro/job_detail/3167665?action=view-source

Problemă

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/car.

Putem obține o altă soluție?

Soluție mai bună cu BFS 0-1

Să rezolvăm următoarea problemă: https://infoarena.ro/problema/car.

Soluție: Putem să facem și observația că operațiile de tip 1 și 2 pot fi simulate prin simple rotiri ale direcției, în timp ce noi rămânem pe aceeași celulă. Practic, avem 2 variante: continuăm să mergem pe direcția actuală (deci costul este 0) sau schimbăm direcția (fie la stânga, fie la dreapta, cu costul 1 în ambele cazuri). Așadar, problema se reduce la algoritmul BFS 0-1.

Implementare: https://infoarena.ro/job_detail/3167681?action=view-source

Soluții diferite la problema Camionaș

Soluții: 1. https://infoarena.ro/job_detail/3158978?action=view-source (cu BFS 0-1)
2. https://infoarena.ro/job_detail/3167598?action=view-source (cu Dijkstra)
3. https://infoarena.ro/job_detail/3167622?action=view-source (cu Dial)

În cazul de față, nu se observă o diferență considerabilă între timpii de execuție, dar, în practică, dacă putem folosi algoritmul lui Dial, atunci preferăm să facem asta, decât să folosim Dijkstra.

Temă

- https://leetcode.com/problems/shortest-bridge/description/
- https://infoarena.ro/problema/distante
- https://cses.fi/problemset/task/1195
- https://cses.fi/problemset/task/1196
- https://pbinfo.ro/probleme/2933/tollroads

Probleme suplimentare

- https://infoarena.ro/problema/rover
- https://infoarena.ro/problema/dmin
- https://infoarena.ro/problema/lanterna
- https://infoarena.ro/problema/base3
- https://codeforces.com/gym/103081/problem/C

Lectură suplimentară

- https://cp-algorithms.com/graph/dijkstra.html
- https://cp-algorithms.com/graph/dijkstra_sparse.html
- https://cp-algorithms.com/graph/01_bfs.html