Tema Algoritmi Fundamentali

Buhai Darius

November 20, 2020

Exercitiul 1

Fie graful G = (V, E). In primul rand, putem demonstra ca orice graf conex aciclic cu n noduri, nu poate avea mai mult de n-1 muchii. Fie V_1 = multimea nodurilor de grad 1 si k_1 = cardinalul multimii V_1.

Daca $k_1 = 0$, atunci inseamnna ca avem un ciclu (putem ajunge de la cel putin un nod la un altul pe drumuri diferite). Din graful curent conex, cream un nou graf, eliminand nodurile din multimea V_1. Aplicand acesti pasi, vom putea ajunge la 2 cazuri posibile:

1. Graful ramane format dintr-un singur nod, caz in care au fost eliminate n-1 noduri, deci graful are n-1 muchii.
2. Graful ramane format din 2 noduri si o singura muchie, caz in care am eliminat n-2 noduri, deci avem n-2+1 = n-1 muchii.

Daca la niciun pas nu am avut multimea V_1 vida (graf aciclic), atunci am scos n-1 muchii, deci rezulta ca orice graf aciclic conex poate sa aiba maxim n-1 muchii.

Ramane de demonstrat ca orice graf aciclic poate sa aiba maxim n-1 muchii. Daca nu ar fi conex, ar fi compus din multiple componente conexe care au k_x noduri si $k_x - 1$ muchii (conform argumentului precedent). Deci in total ar avea $(k_1 - 1) + (k_2 - 1) + ... < (n - 1)$ muchii.

Exercitiul 2

1. Un cub rubic poate fi reprezentat ca un graf de stari, unde:
   - V = Starea in care se afla cubul
   - E = Muchiile care fac legatura dintre starile cubului, reprezentant starea la care poti ajunge printr-o singura mutare.

   Graful este neorientat, deoarece orice mutare este reversibila.
2. Un algoritm eficient de rezolvare a cubului, este unul care pleaca din starea initiala si pargurge graful pana ajunge intr-o stare finala (in care cubul rubic este rezolvat) folosind algoritmul BFS si adaugand in coada starile curente. Drumul minim de la starea initiala la starea finala, este de maxim 26 de mutari, pentru un cub rubik 3x3, numar care se mai numeste si God's number (Sursa: Wikipedia).

Exercitiul 3

Putem elimina maxim |E| - n + 1 muchii. Astfel aplicam BFS pentru a vedea cat de repede putem ajunge la fiecare dintre nodurile 2, 3, .., n din nodul 1.

Fie T_i multimea posibilitatilor pentru care d(1, nod) = i. Ca sa ajungem la aceste noduri, vom pleca din nodurile T_{i-1} pentru care d(1, nod) = i-1, deci pentru fiecare nod la distanta minima i, il vom lega printr-o muchie existenta de un alt nod la distanta i-1.

Pe logica aceasta, vom crea arborele BFS ce va avea doar n-1 muchii, celelalte putand fi eliminate.

Exercitiul 4

Putem modela graful in felul urmator:

1. Avem d + 1 noduri, semnificand numerele de la 0 la d, unde fiecare nod va avea 2 muchii: muchie catre nod + a si muchie catre nod + b. Graful este deci orientat.
2. Pentru a rezolva problema data, se observa ca x + y reprezinta numarul minim de muchii de tipul (nod, nod + a) plus numarul de muchii de tipul (nod, nod + b) pentru a ajunge intr-un nod', cu proprietatea ca nod' este multiplu al lui d. Mai direct, trebuie sa aflam numarul minim de muchii de la 0 la un numar nod'. Pentru asta vom folosi algoritmul BFS.

Exercitiul 5

Fiecare dintre noduri mai poate fi legat de toti fii sai de grad 2 astfel:

- Nodul 1 mai poate fi legat de nodurile 8, 4, 5, 2, 11, 10, 7 si 9
- Nodul 3 mai poate fi legat de nodurile 8, 5, 2
- Nodul 4 mai poate fi legat de nodul 2
- Nodul 6 mai poate fi legat de nodurile 10 si 9.

In total mai putem adauga 14 muchii, iar numarul maxim de muchii pe care il poate avea graful dat este de 10 + 14 = 24.

Ordinea de procesare a vecinilor unui varf trebuie sa fie cea din arborele DFS initial.

Exercitiul 6

Pentru ca graful nostru sa respecte $T_{DFS} = T_{BFS}$, el trebuie sa fie un arbore, adica un graf neorientat, conex și fără cicluri.

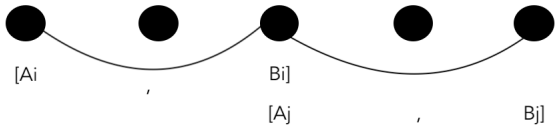
Prin urmare o parcurgere in adancime (DFS) va avea acelasi rezultat cu una in latime (BFS), deoarece ambele parcurgeri se comporta identic atunci cand nu exista cicluri in graf.

Exercitiul 7

Pentru a rezolva problema eficient putem considera variabilele $m_1 = min(a_i)$ si $m_2 = max(b_i)$ si graful G = (V, E), unde:

- V = multimea nodurilor reprezentate de numere de la m_1 la m_2.
- E = multimea de muchii ce reprezinta legaturile dintre intervale. Pentru $[a_i, b_i]$, vom avea muchie de la nodul a_i la nodul b_i.

Algoritmul va parcurge toate componentele conexe create, cautand drumul de lungime maxima format din noduri in ordine crescatoare (asemanator cu BFS). Mai jos am desenat un exemplul vizual pentru 2 intervale date $[a_i, b_i]$ si $[a_j, b_j]$:



In exemplul dat, pornim din nodul 1 si putem ajunge in nodul 5, parcurgand cele 2 muchii (date de intervalele i si j).

Exercitiul 8

Fie graful G_A cu n noduri conectate intre ele, astfel incat pentru $\forall i, j \in V, Q(i, j) = 1$.

Pentru a determina muchiile de legatura dintre fiecare nod, algoritmul A va porni din nodul 1 pana in nodul (n-1), interogand cate (n-i) muchii. La fiecare pas putem exclude nodurile vizitate, deoarece fiind un graf neorientat, muchia (x, y) = (y, x).

Astfel numarul total de interogari va fi $(n - 1) + (n - 2) + ... + 1 = \frac{n(n-1)}{2}$.

Exercitiul 9

Fie graful $G = (V, E)$, unde V = multimea punctelor de interes si E = distanta dintre puncte.

Un algoritm eficient care gaseste un itinerariu corespunzator, foloseste Arbori Partiali de Cost Minim, printr-o parcurgere DFS din (x_1, y_1).

Pentru a ajunge intr-un nod, trebuie sa folosim muchia catre el, iar recursivitatea din DFS ne va da din nou aceeasi muchie, cand trebuie sa iesim din nodul respectiv. Astfel, vom marca fiecare muchie de 2 ori (la intrare si iesire).