Tema 2 AG

Oancea Ionut Eugen 3A5, Marius Dinu 3A5

November 2020

1. a) Pentru a aplica teorema lui Hall, trebuie demonstrat ca $|N_G(A)| \geq |A|, \forall A \in S$. Daca toate nodurile din S au gradele egale cu x, atunci numarul de vecini ai unei submultimi A este $x \cdot |A|$, care este mai mare ca $|A|$. Deci, teorema lui Hall poate fi aplicata, adica exista un cuplaj care satisface toate nodurile din S.

b) Jucatorii vor stabili urmatoarele reguli (inainte de joc): prima carte pusa jos va codifica simbolul cartii de ghicit, iar urmatoarele 4 valoarea. Este garantat ca exista macar 2 carti cu acelasi simbol intre cele 6 trase (pentru ca sunt doar 4 simboluri) - una va fi pastrata, iar a doua va fi pusa drept prima carte. Valorile cartilor sunt intre 1 si 13. Cele 4 carti pot codifica o valoare daca se considera permutarile lor: atunci cand toate 4 sunt in ordine, reprezinta valoarea 1; cand doar ultimele 2 sunt inversate, valoarea 2 s.a.m.d. Pentru a exista o relatie de ordine stricta intre carti, se poate stabili de la inceput si o ordine a simbolurilor. Astfel, se pot codifica $4! = 24$ de permutari, adcia 24 de numere diferite.

2. a) Conform enuntului, oricare 2 servere la distanta 2 vor fi conectate direct. Deci, oricare 2 vecini de-ai unui server vor fi conectati direct (pt ca sunt la distanta 2 unul de altul). Astfel, daca va cadea un server, toti vecinii lui vor ramane totusi conectati intre ei, deci nu se va deconecta reteaua. De asemenea, fiindca orice client e conectat la 2 servere, nu conteaza daca pica unul din ele, tot va ramane conectat la al doilea si deci si la restul retelei.

b) Fie e si f cele 2 legaturi care vor cadea, e fiind prima din ele care cade. Conform cerintei, e se afla pe 2 circuite - fie ele c_1 si c_2. Fiind pe un circuit, atunci cand e cade reteaua nu se va deconecta (se va putea ajunge in continuare la capetele lui e urmand restul ciclului). f nu se poate afla si pe c_1 si pe c_2 pentru ca cele 2 circuite pot avea doar o muchie in comun, deci exista si un al 3-lea circuit c_3 pe care se afla f, iar acest circuit ramane intact chiar si dupa ce cade e. Deci, atunci cand va cadea f reteaua va ramane conexa.

3. a) Mai intai, vom arata ca 2 b-subgrafuri nu se pot intersecta in mai mult de un nod. Conform enuntului, un b-subgraf este 2-conex, deci daca un nod este eliminat din el, acesta ramane conex. De asemenea, orice alt subgraf care il contine nu este 2-conex. Fie A si B 2 b-subgrafuri care se intersecteaza si fie $A \cup B$ subgraful generat de ele. $A \cup B$ nu este 2-conex, deoarece contine un b-subgraf. Dar, A si B sunt 2-conexe individual, deci inseamna ca punctul de articulatie care va face ca $A \cup B$ sa nu fie conex trebuie sa fie la intersectia celor 2. Teoretic, acel punct va deconecta A de B. Acest lucru este posibil doar daca la intersectia lor exista un singur nod, astfel fortand toate drumurile de la A la B sa treaca prin el - deci devenind punct de articulatie.

Apoi, vom arata ca nodul de la intersectia lui A cu B este punct de articulatie si in graful mare. Presupunem prin reducere la absurd ca nu ar fi punct de articulatie. In acest caz, daca il eliminam din graf trebuie sa mai existe un alt drum de la A la B, fie acest drum D. Astfel, se formeaza un ciclu din D si drumul deja existent de la A la B prin nodul de la intersectie. Nodurile din acest ciclu nu pot fi puncte de articulatie in G (deoarece daca este eliminat unul se poate ajunge in continuare de la oricare la oricare utilizand lantul ramas), deci se formeaza un subgraf care nu contine puncte de articulatie (adica este 2-conex) din A, B si D. Acest subgraf 2-conex il contine pe A strict, care este b-subgraf, ceea ce este in contradictie cu proprietatea lui. Deci, nodul de la intersectia lui A cu B trebuie sa fie punct de articulatie in G.

Acum, vom arata ca orice punct de articulatie din G este la intersectia mai multor b-subgrafuri. Fie un punct de articulatie u. In jurul sau sunt mai multe subgrafuri distincte care se intersecteaza in u si care daca ar fi eliminat u s-ar deconecta unul de altul. Vom demonstra ca in interiorul lor exista un

b-subgraf care il contine pe u. Pentru asta, vom defini un proces de construire a unui b-subgraf pornind de la un subgraf 2-conex. u impreuna cu un vecin de-al sau formeaza un subgraf indus 2-conex, X. Atunci cand gasim un subgraf care este 2-conex si il contine strict pe X, il extindem pe X a.i. sa fie egal cu acest subgraf. Astfel, repetand procesul vom obtine un b-subgraf. Folosind aceasta metoda de constructie cu cate un vecin de-al lui u din fiecare componenta, vom obtine mai multe b-subgrafuri care se vor intersecta in u, deci proprietatea a fost demonstrata.

b) Mai intai, vom arata ca $\mathcal{G}$ este conex. Un drum in $\mathcal{G}$ reprezinta un drum din G care trece prin anumite puncte de articulatie si b-subgrafurile dintre ele. Construind b-subgrafuri in jurul punctelor de articulatie din G precum la subpunctul anterior, G poate fi impartit in mai multe componente 2-conexe maximale legate prin puncte de articulatie. Astfel, daca nu ar exista un drum in $\mathcal{G}$ de la un punct de articulatie a la alt punct b, ar insemna ca nici in G nu exista acel drum - imposibil, deoarece G este conex. Deci, si $\mathcal{G}$ este conex.

Apoi, vom arata ca $\mathcal{G}$ nu contine niciun ciclu. Presupunem prin absurd ca exista un ciclu c in acest graf. Acest ciclu ar semnifica ca exista de asemenea un ciclu in G care sa contina mai multe puncte de articulatie. Dar, acest lucru este imposibil deoarece un punct de articulatie nu poate fi pe un ciclu (dupa cum am mai aratat la **a)**). Deci, $\mathcal{G}$ nu contine niciun ciclu.

Din cele demonstrate anterior, rezulta ca $\mathcal{G}$ este un arbore (e un graf conex aciclic).

c) Radacina, fiii si tabloul *parent* se refera la informatii legate de arborele parcurgerii DFS a grafului. In acest algoritm, pargurgerea DFS se face post ordine. Daca radacina are 2 (sau mai multi) fii, inseamna ca nu se poate ajunge la niciunul din nodurile din subarborele celui de-al doilea fiu din nodurile din subarborele primului fiu si invers (daca se putea, atunci subarborele fiului 2 ar fi fost inclus in subarborele fiului 1). Deci, singurul drum de la unul dintre subarbori la altul este chiar prin radacina, ceea ce inseamna ca aceasta este punct de articulatie.

Pentru a demonstra partea a 2-a din cerinta, vom explica ce reprezinta tablourile *order* si *high*. In *order* se retine in ce ordine au fost vizitate nodurile de catre DFS, astfel: primul nod vizitat are $order[nod]$ 1, al doilea $order[nod]$ 2 s.a.m.d. Pentru a explica rolul tabloului *high*, mai intai vom defini conceptul de "muchie de intoarcere": aceasta este o muchie care apare in graf dar nu si in arborele DFS, adica nu a fost parcursa de acesta pentru ca ar fi dus intr-un nod deja vizitat. Ea leaga 2 noduri din acelasi subarbore (daca ar fi dus in alt subarbore ar fi fost explorata de DFS) si practic ar putea fi folosita pentru a "urca" in acel subarbore. In *high* se retine ordinea unui nod x, cu proprietatea ca x este nodul pozitionat cel mai sus in arbore in care se poate ajunge din subarborele nodului curent folosind "muchii de intoarcere". Astfel, daca $order[x] > high[fiu]$, pentru un fiu oarecare, inseamna ca din subarborele fiului nu ne putem intoarce intr-un nod mai sus in arbore decat x. Acest lucru are semnificatia ca singurul drum de la restul grafului la subarborele fiului trece prin x, deci acesta este punct de articulatie. In caz contrar, daca toti fiii lui x au $high[fiu] \geq order[x]$, exista muchii de intoarcere pentru toti pentru a ajunge mai sus de x (sau chiar in x), adica sunt mai multe drumuri pana la ei, deci x nu este punct de articulatie.

4. a) In urma unei iteratii **while**, sunt selectate $p-1$ muchii care vor lega unele dintre componentele conexe. In cel mai bun caz, se vor alege $p-1$ muchii distincte si tot graful va deveni conex, dar in cel mai rau caz, numarul minim de muchii distincte va fi $p/2$ - fiecare muchie poate fi aleasa de maxim 2 ori, cate o data din fiecare capat; astfel, jumatate dintre muchii se vor repeta, deci de fapt se vor uni doar $p/2$ componente conexe. Deci, numarul de componente conexe se va injumatati in cel mai rau caz.

b) Vom presupune, prin reducere la absurd, ca T nu este aciclic, adica contine macar un ciclu. Fie c un ciclu din T. Pentru ca acesta sa se poata forma, trebuie ca muchiile din el sa aiba costuri din ce in ce mai mici (daca a fost aleasa o muchie e intre T_i si T_j, urmatoarea muchie, care va fi aleasa din T_j, trebuie sa aiba costul mai mic decat e, altfel s-ar alege din nou e si nu s-ar mai continua ciclul) - strict mai mici, deoarece toate muchiile au costuri distincte. Dar, acest lucru devine imposibil atunci cand trebuie selectata ultima muchie, intre ultima componenta si prima: daca ultima muchie are costul mai mic decat prima, atunci ar fi trebuit aleasa aceasta muchie si din prima componenta - contradictie. Deci, T este aciclic.

c) Fie T_s si T_j 2 componente conexe care urmeaza sa fie unite printr-o muchie $v_s v_j$. Cele 2 componente sunt aciclice (conform **b**) si conexe, deci sunt arbori. Astfel, aceasta situatie este identica cu cea descrisa in algoritmul pentru metoda generala MST descris in cursul 6, deci poate fi aplicata aceeasi demonstratie (descrisa de asemenea in curs) pentru a arata ca muchia aleasa de cost minim $v_s v_j$ face parte dintr-un MST de-al lui G. Acest rezultat este valabil pentru oricare 2 arbori care urmeaza sa fie uniti, deci algoritmul va alege intotdeauna muchii din MST. Astfel, in final T va fi conex, aciclic si va contine doar

muchii din MST-ul lui G, deci este un MST pentru G.

d) Utilizand rezultatul de la **a)**, este logic ca **while**-ul va rula de $O(logn)$ ori (numarul initial de componente conexe este n si algoritmul se opreste cand mai este doar una). Pentru a calcula complexitatea operatiilor din interiorul buclei, vom stabili ca se vor folosi anumite implementari de Union-Find: $find$ va fi a 3-a implementare descrisa in cursul 6, care, folosind colapsarea drumurilor va avea complexitatea $O(1)$ in practica, iar $union$, folosind implementarea a 2-a din curs va avea complexitate $O(1)$. In interiorul **for**-ului se va verifica pentru o muchie uv din ce componente fac parte u si v apeland $find$ pe acestea (deci in $O(1)$), iar in total se pot verifica maxim $2 \cdot m$ muchii - atunci cand fiecare muchie este verificata din ambele capete. Deci, **for**-ul se executa in $O(m)$. Actualizarea grafului cu muchiile alese se face in maxim $2 \cdot m$ pasi (daca fiecare muchie este aleasa din ambele capete si toate muchiile sunt alese), iar adaugarea propriu-zisa a unei muchii se face folosind $union$ (in $O(1)$), deci complexitatea actualizarii este $O(m)$. In concluzie, complexitatea totala a buclei **while** este $O(m)$.