TEMA 1 ALGORITMICA GRAFURILOR

Anca Dorneanu Ţuţuianu Corneliu *grupa A7*

November 5, 2014

1 Problema 1.

1.1 A

Arătaţi că dacă **M** este un mers închis impar într-un digraf **D**, atunci **M** conţine un circuit impar în **D**.

Fie $\mathbf{D}\left(V\left(D\right),A\left(D\right)\right)$ un digraf. Fie $\mathbf{M}\left(v=\right)v_0,v_0v_1,v_1,\ldots,v_{r-1},v_{r-1}v_r,v_r(=u)$ cu $v=u\ \forall v_i\in V\left(D\right)$ un mers închis,impar de la v la u.$E'(M)=$ mulţimea de muchii din M.

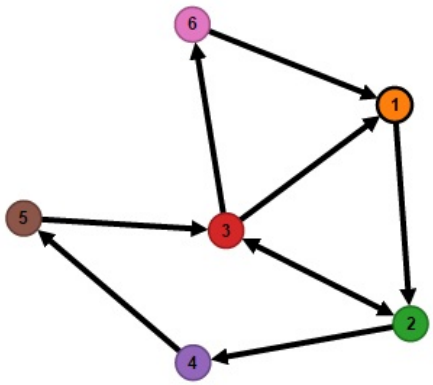
$$|E'(M)|=2*k+1,k\forall k\in\mathbb{N}$$

1. Caz 1: în mers fiecare vârf este parcurs o singură dată, deci prin definiţie mersul închis impar este un circuit impar şi $M\in D$

   $\forall v_i,v_j\in A\left(D\right),\forall v_iv_j\in E'\left(D\right)$,muchia v_iv_j parcurs o singura data sau varful v_i(sau v_j) este parcurs o singura data $\Rightarrow M$ devine un circuit impar prin definitie.

   Ex.

   Fie **D** digraful de mai jos



$$M=v_1,v_1v_2,v_2,v_2v_4,v_4,v_4v_5,v_5,v_5v_3,v_3,v_3v_1,v_1$$

   Având în vedere faptul că fiercare vârf e parcurs 1 singură dată, mersul închis ,impar **M** devine circuit şi $\mathbf{M}\subseteq\mathbf{M}\Rightarrow\mathbf{M}$ circuit impar în **D**.

2. Caz 2:

   În mersul **M** $\exists n, n \geq 2, n =$ număr de circuite ; si cum mersul **M** este impar si stiind că impar=impar+par si par=par+par sau impar+impar $\exists C \in \mathbf{M}$ circuit din **M** care să fie impar.

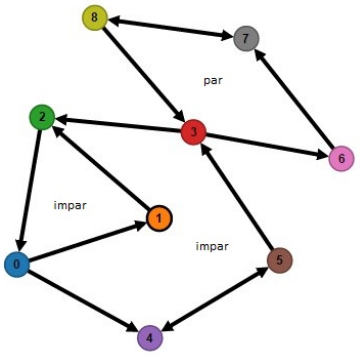
   $$M = v_i, v_i v_{i+1}, \ldots, v_k, v_k v_{k+1}, v_{k+1}, \ldots, v_l, v_l v_k, v_k, v_k v_x, v_x, \ldots, v_{j-1}, v_{j-1} v_j, v_j$$

   cu $v_i = v_j v_i, v_j \in V[D]$

   În vârful v_k se produce împărtirea mersului **M** în 2 circuite distincte, unul din ele având $2k+1$ muchii si unul $2k$ muchii cu $k \in \mathbb{N}$ cu alte cuvinte **M** va contine 2 circuite , unul par si unul imapar ,deci $\exists C \in \mathbf{M}$ circuit din **M** care să fie impar.

   Pe caz general , daca mersul **M** contine mai mult de 2 circuite, vor exista mai multe varfuri $v_{ki}, cui \leq$ numarul de circuite din M, deci in mersul **M** va exista cel putin un $\exists C \in \mathbf{M}$ circuit din **M** care să fie impar.

   Ex:



$$M = v_0, v_0 v_1, v_1, v_1 v_2, v_2, v_2 v_0, v_0, v_0 v_4, v_4, v_4 v_5, v_5, v_5 v_3$$

$$, v_3, v_3 v_6, v_6, v_6 v_7, v_7, v_7 v_8, v_8, v_8 v_3, v_3, v_3 v_2, v_2, v_2 v_0, v_0$$

Mersul prezinta 3 circuite, 2 circuite impare 0-1-2-0 si 0-4-5-3-2-0 , si 1 circuit par 3-6-7-8-3; $k_0 = v_0$ pt primul circuit, $k_1 = v_1$ pt circuitul par si $k_2 = v_0$ pt al doilea circuit impar; deci avem cel putin un $\exists C \in \mathbf{M}$ circuit din **M** care să fie impar.

1.2 B

Demonstrati ca graful suport al unui digraf tare conex care nu are circuite impare este un graf bipartit.

Fie **D** digraf tare conex; $\mathbf{D} = (V(D), A(D))$ $A(E(D), E(D))$ si nu are circuite impare.

$\forall (v, u) \in (V(D), V(D)) \exists$ drum de la v la u.

Daca in $\mathbf{D}$ nu $\exists$ circuit impar $\Rightarrow$ in $\mathbf{D}$ $\exists$ doar circuite pare $\Rightarrow |V(D)| = 2k$ cu $k \in \mathbb{N}$.

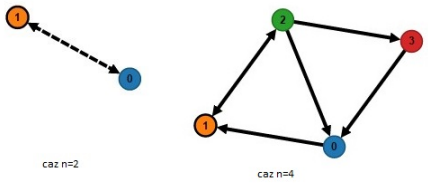
Daca $|V(D)| = 2k+1$ cu $k \in \mathbb{N}$ ar insemna ca circuitul care asigura tare conexivitatea ar fi impar (contradictie cu ipoteza).

Dat fiind un digraf $\mathbf{D} = (V(D), A(D))$ $A(E(D), E(D))$, atunci inlocuind fiecare arc $a = (u, v)$ cu multimea u, v, se obtine un graf $G(D)$ numit graful suport al digrafului D.

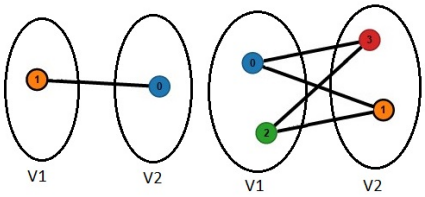
Reformuland definitia anterioară, se poate spune ca graful $\mathbf{G}$ suport al unui digraf $\mathbf{D}$ se obtine prin eliminarea sensului (sau a orientarii) arcelor.

Un graf $G = (V, E)$ este bipartit daca: $V = V_1 \cup V_2, a.i. V1 \cap V2 = \emptyset$ si $\forall i, j \in E$, atunci $i \in V_1, j \in V_2$ sau $j \in V_1, i \in V_2$.

1. Caz 1: $|V(D)| = 2$ banal;
2. Caz 2: $|V(D)| = 4$



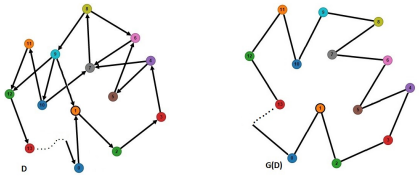
Aflam grafurile suport si varfurile le introducem in 2 multimi distincte si se poate observa ca formeaza grafuri bipartite:



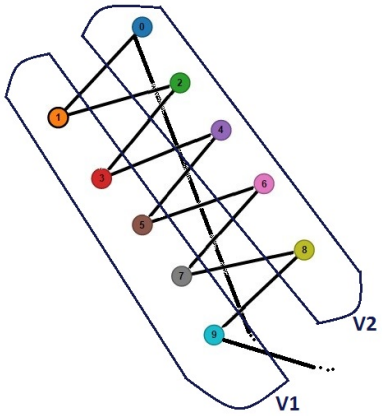
Stiind ca orice digraf tare conex am avea si datorita faptului ca acest digraf are $2k$ varfuri mereu vom putea organiza varfurile intr-un graf bipartit.

3. Caz general:

   Avem $|V(D)| = n; n \in \mathbb{N}$ $n = 2k$



Graful bipartit se formeaza cu cele n varfuri ,cate $n/2$ varfuri in fiecare multime,prin urmare avand un digraf $\mathbf{D} = (V(D), A(D))$ $A(E(D), E(D))$ tare conex fara circuite impare si pt $\forall n \in \mathbb{N}$ cu n numar par, graful suport al digrafului este un graf bipartit.



2 Problema 2

S-a propus urmatorul algoritm (cu timp de lucru polinomial) pentru determinarea numarului de stabilitate $\alpha(\mathbf{G})$ al unui graf $\mathbf{G}$:

GreedyStab (G) determina un varf de grad minim v_0 in $\mathbf{G}$;
if $V(G) = v_0 \cup N_G(v_0)$ then return 1
else return 1+**GreedyStab**$((G) - v_0 \cup N_G(v_0))$

2.1 A

Aratati ca algoritmul propus poate gresi oricat de mult: $\forall n \in \mathbb{N}$ exista un graf $\mathbf{G}$ astfel incat $\alpha(\mathbf{G}) -$ **GreedyStab** $(G) = n$.

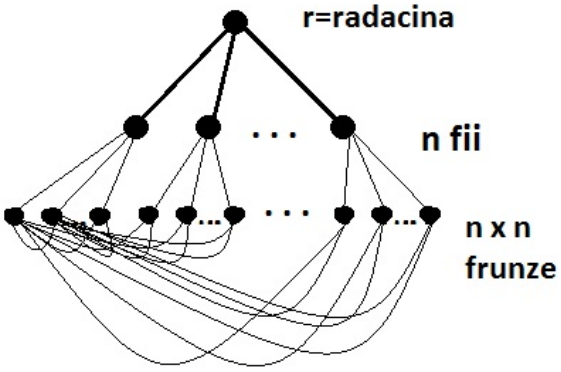
Intelegerea algoritmului

1. Pasul 1 : se alege varful de grad minim din $\mathbf{G}$;
2. Clauza IF : verifica daca reuniunea varfului ales la pasul 1 impreuna cu toti vecinii acestuia formeaza multimea de varfuri din $\mathbf{G}$;
3. Ramura true : In caz de adevar returneaza 1;
4. Ramura false : In caz de fals returneaza 1 si re apeleaza algoritmul pentru graful din care s-au eliminat varful ales la pasul 1 si vecinii acestuia.

Cazul general al executiei algoritmului:
Fie graful $\mathbf{G}$ construit astfel :

- nivelul 0 : radacina v_0;
- nivelul 1 : n fii ai radacinii ,neconectati intre ei;
- nivelul 2 : fiecare nod de la nivelul 1 are in nivelul 2 alti n fii; toate nodurile de la nivelul 3 sunt conectate intre ele .

Desen simplificat (conexiunile frunzelor sunt facute doar pt primele 3 noduri



)

1. Caz n=1: In G exista 3 noduri, unul pentru fiecare nivel; Primul pas : se alege radacina; Se intra pe ramura de false deoarece $r \cup N_G(r) \neq V(G)$,1 + se elimina r si vecinul sau apoi se apeleaza iar GreedyStab pt ultimul nod aflat $\rightarrow$ se intra pe true deoarece a ramas doar ultimul nod in G $\Rightarrow$ GreedyStab returneaza valoarea 2 . Algoritmul greseste cu 1 : $\alpha(\mathbf{G}) = 1$,**Greedystab** returneaza 2 , diferenta lor fiind -1.

2. Caz n=2 : avem 1 radacina ,2 fii si 2 x 2 = 4 nepoti .Primul pas : se alege radacina; Se intra pe ramura de false deoarece $r \cup N_G(r) \neq V(G)$,1 + se elimina r si vecinii sai apoi se apeleaza iar GreedyStab pt un nod frunza $\rightarrow$ se intra pe true deoarece au ramas doar nodurile frunza care sunt conectate intre eele $\Rightarrow$ GreedyStab returneaza valoarea 2 . Algoritmul greseste cu 2 : $\alpha(\mathbf{G}) = 2$,**Greedystab** returneaza 2 , diferenta lor fiind 0

3. Caz n=3 : avem 1 radacina ,3 fii si 3 x 3 = 9 nepoti .Primul pas : se alege radacina; Se intra pe ramura de false deoarece $r \cup N_G(r) \neq V(G)$,1 + se elimina r si vecinii sai apoi se apeleaza iar GreedyStab pt un nod frunza $\rightarrow$ se intra pe true deoarece au ramas doar nodurile frunza care sunt conectate intre eele $\Rightarrow$ GreedyStab returneaza valoarea 2 . Algoritmul greseste cu 1 : $\alpha(\mathbf{G}) = 3$,**Greedystab** returneaza 2 , diferenta lor fiind 1.

4. Caz $\forall n$: avem 1 radacina ,n fii si n x n nepoti .Primul pas : se alege radacina; Se intra pe ramura de false deoarece $r \cup N_G(r) \neq V(G)$,1 + se elimina r si vecinii sai apoi se apeleaza iar GreedyStab pt un nod frunza $\rightarrow$ se intra pe true deoarece au ramas doar nodurile frunza care sunt conectate intre eele $\Rightarrow$ GreedyStab returneaza valoarea 2 . Algoritmul greseste cu n-2 : $\alpha(\mathbf{G}) = n$,**Greedystab** returneaza 2 , diferenta lor fiind $n - 2$.

Prin urmare $\forall n, n \in \mathbb{N}$ $\exists grafG$ pentru care algoritmul sa greseasca oricat de mult: cu valori de la -1 la $n - 2$.

3 Problema 3

Fie $G = (V, E)$ un digraf, X o multime finita si $c : E \to 2^X$ o functie care asociaza fiecarui arc $e = vw \in E$ o submultime a lui X: $c(vw) \subseteq X$. Functia c poate fi extinsa la drumurile lui **G**, considerand pentru orice drum **P** al lui **G**,

$$c(P) = \emptyset \cup \bigcup_{e \in E(P)} c(e)$$

(in particular, daca$E(P) = \emptyset$ – adica P este un varf – avem $c(P) = \emptyset$). Pentru orice $v, w \in V$, notam $P_{v,w} = \{P | P$ drum in **G** de la v la $w\}$. Notam cu $|A|$ numarul de elemente ale multimii A, iar # este un element care nu apartine multimii X. Consideram urmatoarea problema:

P: Dat digraful $G = (V, E)$, functia c si $s \in V$, sa se determine pentru fiecare $v \in V$, un drum $P^{\star}_{sv}$ astfel incat $\left|c\left(P^{\star}_{v,w}\right)\right| = min\{|c(P)| : P \in \mathcal{P}_{s,v}\}$

Adevarat sau Fals? "*Urmatorul algoritm rezolva problema P*".

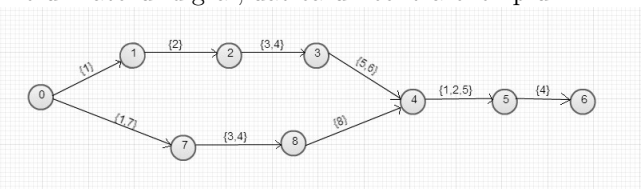
1. $u[s] \leftarrow \emptyset$; $parent[s] \leftarrow 0; S \leftarrow \{s\}$;
for $v \in V - S$do
if $sv \in E$ **then** $\{u[v] \leftarrow c(sv); parent[v] \leftarrow s\}$
else $\{u[v] \leftarrow X \cup \{\#\}; parent[v] \leftarrow -1\}$;
2. while $S \neq V$ **do** find $v^{\star} \in V - S$ s.t. $|u[v^{\star}]| = min\{|u[v]| : v \in V - S\}$
$S \leftarrow S \cup \{v^{\star}\}$; **for** $v \in V - S$**do**

if $v \star v \in E \& |u[v]| > |u[v^{\star}] \cup c(v \star v)|$ **then** $u[v] \leftarrow u[v^{\star}] \cup c(v \star v)$; parent$[v] \leftarrow v^{\star}$;

Algoritmul de mai sus in esenta calculeaza un drum de cost minim pentru un digraf **D**,muchiile sale fiind multimi de elemente, insa in anumite cazuri drumul aflat nu este un drum minim.

Presupun prin reducere la absurd ca algoritmul este adevarat si ca rezolva problema P;

Fie urmatorul digraf,luat ca un contra exemplu:



In digraful de mai sus ,aplicand algoritmul, obtinem urmatorii pasi:

- incepem cu nodul 0;
- ne uitam in vecinii sai si alegem drumul cel mai optim, in cazul nostru spre nodul 1 cu costul = 1;
- ne uitam in vecinii lui 1 si alegem drumul optim, adica cel spre 2 cu cost =1 ;
- suntem in nodul 2, avem un cost al drumului pana acum = 2; ne uitam inapoi in digraf si vedem ca acelasi cost avem si pentru nodul 7 si pentru

cele 2 noduri ne uitam la vecini si la costurile eventuale ale drumurilor, ambele sunt de 2 deci alegem unul dintre ele , am ales nodul 2 ,facem drumul pana la nodul 3 si rezulta un cost =4;

- suntem in nodul 3 si avem un cost total al drumului =4 ; ne uitam inapoi in digraf si vedem ca si drumul spre 8 are acelasi cost, deci ne uitam la vecinii ambelor noduri si observar costurile spre acei vecini si alegem drumul de la 8 la 4 deoarece are cost =1;
- suntem in nodul 4,avem un cost total al drumului = 5 ,mai bun decat celalalt drum care ar fi avut un cost total 6;
- din nodul 4 ajungem in nodul 5 cu un cost total = 7 (cardinalul multimii rezultate prin reuniune) si din 5 in 6 cu costul total al drumului= 8 ;
- acesta este rezultatul algoritmului, el fiind presupus optim; insa el este fals deoarece daca am fi ales celalalt drum chiar daca drumul pana in nodul 4 nu era mai bun decat celalalt,in urma parcurgerii ulterioare spre 5 si 6 ar fi rezultat un drum total de cost = 6. Deci algoritmul nu rezolva problema **P**.

4 Problema 4

Fie $\mathbf{G} = (V, E)$ graf cu $|V| = n$ $|E| = m$, reprezentat cu ajutorul listelor de adiacenta. O ordonare a varfurilor lui **G** este o aplicatie injectiva $\pi : V \rightarrow \{1, 2, 3, \cdots, n\}$, ($\pi((v) = i$ are semnificatia ca varful v se afla pe locul i in ordonarea π) π este o ordonare lexicografica daca pentru orice doua varfuri $x, y \in Vcu\pi\left(x\right) < \pi\left(y\right)$, daca multimea $D_{x,y} = \{z \in V \| \pi\left(z\right) < \pi\left(x\right)$ si z este adiacent cu exact unul dintre x si y $\}$

4.1 A

Aratati ca algoritmul de mai jos construieste o ordonare lexicografica:

Lexicographic (G) initializeaza lista $\mathbb{L}$ de multimi avand o singura multime: **V** ; (fiecare multime din L se reprezinta ca o lista dublu inlantuita)
for $i := 1$ to n do
$v :=$ primul varf al primei multimi din $\mathbb{L}$;
sterge v din acea multime ;
$\pi\left(v\right) := i$;
for each $L_j \in \mathbb{L}$ do
inlocuieste L_j cu $L_j \cap N_G\left(v\right)$ urmata de $L_j - N_G\left(v\right)$;
sterge din $\mathbb{L}$ multimile vide

Fie orice doua varfuri $x, y \in V$ cu $\pi\left(x\right) < \pi\left(y\right)$: Stim ca π este o ordonare lexicografica daca multimea $D_{x,y} = \{z \in V \| \pi\left(z\right) < \pi\left(x\right)$ si z este adiacent cu exact unul dintre x si y $\}$ este nevida. Atunci exista un z0 apartinand multimii descrise mai sus, cu $\pi\left(z_0\right) = min\pi\left(z\right)$, care satisface faptul ca exista muchia $z_0x \in E$ si muchia $z_0ynu \in E$.

Presupunem prin reducere la absurd ca π este o ordonare lexicografica generata incorect de algoritmul **Lexicographic** (G), pentru graful G. Atunci exista doua varfuri $x, y \in V, \pi\left(x\right) < \pi\left(y\right)$, cu multimea $D_{x,y}$ descrisa anterior, si exista

un z_0 apartinand multimii $D_{x,y}$ astfel incat $\pi(z_0) = min\pi(z)$, dar care satisface faptul ca exista muchia z_0x si NU apartine lui E si muchia z_0y care apartine lui E (invers decat in definitie).

Parcurgand algoritmul pentru graful **G**, la un moment dat, se va sterge acel z_0, obtinandu-se doua multimi: una care va contine vecinii lui z_0 si una care va contine nodurile ce nu sunt adiacente cu z_0.

Deoarece ordonarea este generata incorect si pentru ca muchia z_0y exista si apartine multimii de muchii E, inseamna ca y va fi inclus in prima multime, adica cea cu nodurile vecine, iar pentru ca muchia z_0x nu apartine multimii de muchii E, x va fi inclus in cea de-a doua multime, adica in cea care contine nodurile neadiacente cu z_0. Deci mai intai algoritmul va ajunge la nodul y si mai apoi la nodul x, ceea ce inseamna $\pi(y) < \pi(x) \Rightarrow$ contradictie la proprietatea de injectivitate.

Asadar, presupunerea este falsa, deci π este o ordonare lexicografica generata corect de algoritmul **Lexicographic** (G) pentru graful **G**.

4.2 B

Argumentati ca algoritmul se poate implementa ın timpul $O(n+m)$.

Algoritmul descris parcurge toate nodurile grafului. Deoarece multimile sunt reprezentate cu ajutorul listelor de adiacenta, stergerea ("sterge v din acea multime") se realizeaza in timp constant, si anume $O(1)$. De asemenea, crearea multimii care include vecinii nodului v si a celei care nu include vecinii nodului v ("inlocuieste L_j cu $L_j \cap N_G(v)$ urmata de $L_j - N_G(v)$"), este realizata tot in timp constant, ca si stergerea multimilor vide.

Dat fiind faptul ca multimile sunt reprezentate prin liste de adiacenta, parcurgerea grafului are complexitatea $O(n+m)$, astfel ca in cazul cel mai nefavorabil, vor fi explorate toate muchiile si toate nodurile.