ALGORITMICA GRAFURILOR
Tema 1

Daniș Ciprian
Oloieri Alexandru
Anul II, Grupa A2

8 noiembrie 2019

1 Problema I

1.1 (a)

"$\Leftarrow$": G nu are punte $\Rightarrow \forall e \in E(G), G-e$ este conex $\Rightarrow G-e$ este k-muchie-conex, $k \geq 2$. Aplicăm DFS(G, x_1) $\Rightarrow$ ordinea $x_1, x_2, \ldots, x_n$. Muchiile traversate se orientează astfel: $x_i x_j$, $i < j$, iar muchiile netraversate(sau de întoarcere) se vor orienta în felul următor: $x_h x_k$, $h > k$ (x_k se află pe drumul de la x_1 la x_h *(1)*). Din faptul că G este conex $\Rightarrow \exists$ drum de la x_1 la x_i, $\forall i$.

Presupunem prin reducere la absurd că $\exists i$ a.î. $\nexists$ drum de la x_i la x_1 în $\overrightarrow{G}$ (alegem pe x_i la distanță dfs minimă față de x_1).

Cazul I: Dacă x_i nu este frunză $\Rightarrow G-e$ este conex, unde $e = yx_i$, y fiind un nod adiacent lui x_i și $\in$ drumului de la x_1 la x_i (G este k-muchie-conex, $k \geq 2$) $\Rightarrow \exists$ o muchie $x_j x_k$ în G a.î. $x_j \in V(T_{x_i}) \setminus \{x_i\}, x_k \in V(G) \setminus V(T_{x_i})$. Din *(1)* $\Rightarrow k < j$ și x_k se află pe drumul de la x_1 la x_j, deasupra lui x_i. $\exists$ drum de la x_k la x_1 (din alegerea lui x_1) $\Rightarrow \exists$ drum de la x_i la x_1(contradicție).

Cazul II: Presupunem prin reducere la absurd că x_i este frunză. $\exists x_k$, $k < i$ a.î. $x_i x_k \in E(\overrightarrow{G})$. $\exists$ drum de la x_k la x_1 în $\overrightarrow{G}$ (G este k-muchie-conex, $k \geq 2$) $\Rightarrow \exists$ drum și de la x_i la x_1 care trece prin x_k în $\overrightarrow{G}$(contradicție).

Din (**I**), (**II**) $\Rightarrow$ presupunerea făcută este falsă $\Rightarrow \exists$ drum de la x_i la x_1, $\forall i$.

"$\Rightarrow$": $\exists$ drum de la x_i la x_j, $\forall x_i$, $x_j \in V(\overrightarrow{G})$, $x_i \neq x_j$.
Presupunem prin reducere la absurd că G este 1-muchie-conex ($\exists e \in E(G)$ a.î. $G-e$ este neconex).

Cazul I: $\exists i$ a.î. $x_i \in V(G)$ este frunză și e este muchia care conectează x_i de restul grafului $\Rightarrow \exists$ drum de la x_j la x_i, dar $\nexists$ drum de la x_i la x_j în $\overrightarrow{G}, \forall j \neq i$(contradicție).

Cazul II: Fie $x_i, x_j \in V(G)$ a.î. $x_j x_i = e, x_j x_i \in E(\overrightarrow{G})$ și S, T componente conexe, $S, T \subset G-e, S \cup T = G-e, x_j \in S, x_i \in T$(capetele muchiei e se află în componente conexe diferite în $G-e$). Atunci $\exists$ drum de la $\forall y \in S$ la $\forall z \in T$ care trece prin x_j și x_i în $\overrightarrow{G}$, dar $\nexists$ drum de la $\forall z \in T$ la $\forall y \in S$ care trece prin x_i și x_j în $\overrightarrow{G}$(contradicție).

Din (**I**), (**II**) $\Rightarrow$ presupunerea făcută este falsă $\Rightarrow$ G este k-muchie-conex, $k \geq 2 \Leftrightarrow G$ nu are punte.

1.2 (b)

Fie $G = (V, E)$ graful neorientat conex asociat rețelei stradale a orașului. Pentru algoritmul ce urmează a fi descris vom presupune că muchiile acestuia sunt reprezentate cu ajutorul listelor de adiacență: pentru un nod u, $A[u]$ este mulțimea tuturor nodurilor adiacente cu u.

Ideea algoritmului: se face o parcurgere DFS a grafului G, fie u nodul curent (ce a fost scos din stivă) și uv muchia curentă. Dacă nodul v a fost vizitat, atunci muchia uv este muchie de întoarcere, deci va fi transformată în arcul vu, iar dacă nu a fost vizitată atunci nodul v va fi introdus în stivă și muchia va fi transformată în arcul uv. Verificarea dacă un nod a fost vizitat se face în $O(1)$ cu ajutorul unui vector caracteristic: $visited[u] = true$ dacă nodul a fost vizitat, inițial $visited[v] = false, \forall v \in V$.

Întrucât G este neorientat, $\forall u, v \in V$ dacă $u \in A[v]$ atunci și $v \in A[u]$, deci în timpul parcurgerii grafului trebuie să ne asigurăm că atunci când analizăm muchia uv, ștergem din $A[v]$ pe u, pentru a nu analiza aceeași muchie de 2 ori, și deci să introducem arce în plus în digraful rezultat.

Graful $G = (V, E)$ este k-muchie conex, $k \geq 2$, deci pentru transformarea lui G într-un digraf tare conex $G' = (V, E')$ putem începe parcurgerea dfs din orice nod $u, u \in V$, iar în algoritmul de mai jos am notat acest nod aleator cu s. Mulțimea E este reprezentată prin liste de adiacență, însă algoritmul de mai jos va întoarce ca rezultat o pereche $G' = (V, E')$, unde V este mulțimea de noduri din G iar E' este o mulțime de arce.

Complexitatea timp a algoritmului: Orice nod este introdus în stivă o singură dată, iar orice muchie este analizată o singură dată (atunci cand se decide orientarea sa), deci complexitatea timp a algoritmului este $O(N+M)$, unde $N = |V|$ și $M = |E|$.

```
1:  E' ← ∅; // inițial mulțimea arcelor e vidă
2:  for (v ∈ V) do
3:      visited[v] ← false;
4:  S ← stivaVida();
5:  push(S, s);
6:  while (S ≠ ∅) do
7:      u ← top(S);
8:      visited[u] ← true;
9:      if ((v ← next[A[u]]) ≠ NULL) then
10:         if (visited[v] = false) then
11:             push(S, v);
12:             A'[v] ← A'[v]\{u};
13:             E' ← E' ∪ {uv}; // arcul uv
14:         else
15:             E' ← E' ∪ {vu}; // muchie de întoarcere, arcul vu
16:     else
17:         delete(S, u);
18: G' ← (V, E')
19: return G';
```

2 Problema II

2.1 (a)

"⇒": $X \subseteq V$ este mulțime uv-separatoare minimală $\Rightarrow \exists S, T \subset (G - X), S \neq T$ și S, T conexe a.î. $u \in S, v \in T$ și $\forall X' \subsetneq X, u, v \in S', S' \subset (G - X')$ conex *(1)*$\Leftrightarrow$ $\nexists X'' \subsetneq X, |X''| < |X|$ a.î. X'' mulțime uv-separatoare minimală. Din *(1)* $\Rightarrow X$ este o mulțime ST-separatoare minimală, $|X| = k(S, T; G)$. Conform Teoremei lui Megner $\Rightarrow p(S, T; G) = k(S, T; G) = |X| \Rightarrow \exists$ maxim $|X|$ drumuri disjuncte de la S la T*(2)*. Din faptul că S, T sunt conexe și *(2)* $\Rightarrow \exists$ maxim $|X|$ drumuri disjuncte de la u la v*(3)*. Din *(3)*, $u \in S, v \in T$ și $S, T \subset (G - X), S \neq T \Rightarrow$ pentru fiecare $x_i \in X$ și $x_i \in P_i$, unde P_i este drum de la u la $v, 1 \leq i \leq |X|, x_i$ va avea 2 noduri adiacente $w_1, w_2, w_1 \in S, w_2 \in T$.

"⇐": Fie S, T componente conexe, $S, T \subset (G - X), S \neq T$ și u, v 2 noduri a.î. $u \in S, v \in T$. Pentru $\forall x \in X, x$ are vecini în S, respectiv T*(4)* $\Rightarrow X$ mulțime ST-separatoare $\Rightarrow X$ mulțime uv-separatoare*(5)*.

Presupunem prin reducere la absurd că $\exists X' \subsetneq X$ a.î. X' este mulțime uv-separatoare minimală de cardinal $|X'| < |X|$. Dar cum are loc *(4)* $\Rightarrow \exists x_k \in X$ a.î. $x_k \notin X'$ din cauza căruia S, T vor rămâne conectate $\Rightarrow u, v \in S', S' \subset (G - X'), S' = S \cup T, S'$ conex(contradicție) $\Rightarrow$ presupunerea făcută este falsă $\Rightarrow \forall X' \subsetneq X, u, v \in S', S' \subset (G - X'), S'$ conex*(6)*.

Din *(5)*, *(6)* $\Rightarrow X$ este mulțime uv-separatoare minimală.

2.2 (b)

X_1, X_2 mulțimi uv-separatoare minimale $\Rightarrow X_1 \neq X_2, X_1 \not\subset X_2, X_2 \not\subset X_1$. Fie $S_1, T_1 \subset (G - X_1)$ componente conexe distincte a.î. $u \in S_1, v \in T_1$ și $\forall x_1 \in X_1$ are vecini în S_1 și T_1, respectiv $S_2, T_2 \subset (G - X_2)$ componente conexe distincte a.î. $u \in S_2, v \in T_2$ și $\forall x_2 \in X_2$ are vecini în S_2 și T_2.

Presupunem prin reducere la absurd că X_1 intersectează cel mult una din componentele conexe S_2 și T_2.

Cazul I: X_1 nu intersectează nici una din componentele conexe S_2 și $T_2 \Rightarrow X_1$ este o mulțime independentă de cele 2 componente*(1)*. Dar X_1 este mulțime uv-separatoare minimală $\Rightarrow \forall x_1 \in X_1$ are vecini în S_2 și T_2 *(2)*. Din *(1)*, *(2)* $\Rightarrow S_2, T_2$ fac parte din aceeași componentă conexă $\Rightarrow u, v$ fac parte din aceeași componentă conexă $\Rightarrow X_2$ nu este mulțime uv-separatoare minimală(contradicție).

Cazul II: X_1 intersectează una dintre componentele conexe S_2, T_2. Dar X_1 intersectează cel puțin 2 componente din $(G - X_2) \Rightarrow X_1$ intersectează măcar o altă componentă conexă, independentă de cele 2. Fie Q respectiva componentă. Avem că X_1 este mulțime uv-separatoare minimală, dar u, v fac parte din componente conexe diferite în $G - (X \setminus Q) \Rightarrow (X \setminus Q)$ este mulțime uv-separatoare de cardinal $|X \setminus Q| < |X|$(contradicție).

Din (**I**), (**II**) $\Rightarrow$ presupunerea facută este falsă $\Rightarrow X_1$ intersectează pe S_2, T_2, unde $S_2, T_2 \subset (G - X_2)$ conexe și $u \in S_2, v \in T_2$.

3 Problema III

3.1 (a)

Fie următorul algoritm:

1: $V' \leftarrow V$; $A' \leftarrow A$;
2: construct the array $d_G[u]$, $\forall u \in V'$;
3: $Q \leftarrow \{u \in V : d_G[u] < m/n\}$; // Q este o coadă
4: **while** $(Q \neq \varnothing)$ **do**
5: $\quad v \leftarrow pop(Q)$;
6: $\quad V' \leftarrow V' \backslash \{v\}$;
7: $\quad$ **for** $(w \in A'[v])$ **do**
8: $\quad\quad A'[v] \leftarrow A'[v] \backslash \{w\}$;
9: $\quad\quad A'[w] \leftarrow A'[w] \backslash \{v\}$;
10: $\quad\quad -- d_G[w]$;
11: $\quad\quad$ **if** $(d_G[w] < m/n)$ **then**
12: $\quad\quad\quad push(Q, w)$;
13: return (V, A);

Ideea algoritmului: Se calculează gradele fiecărui nod din graf. La fiecare moment vom menține o mulțime de noduri despre care știm că trebuie eliminate din graf, vom implementa acest lucru cu ajutorul unei cozi (structura de date abstractă ce are complexitate O(1) pe extragerea unui element). La început se introduc într-o coadă toate acele noduri u, cu $d_G[u] < m/n$. Apoi, cât timp coada e nevidă, efectuăm următoarele operații: extragem u primul nod din coadă, parcurgem lista lui de adiacență și decrementăm gradele nodurilor v adiacente cu u, dacă $d_G[v]$ devine $< m/n$, introducem și nodul v în coadă.

Afirmație: Algoritmul descris mai sus este o implementare eficientă a algoritmului din enunțul problemei.

Demonstrație: Un nod u se poate afla în situația să îndeplinească condiția $d_G[u] < m/n$ în două cazuri:

Cazul I: În graful inițial G acesta este adiacent cu mai puțin de m/n noduri.

Cazul II: După ce din graful G au fost eliminate un număr de noduri adiacente cu u, deci gradul acestuia a scăzut.

Afirmație: Algoritmul de mai sus ce folosește o coadă nu poate "rata" nici un nod u ce a ajuns să aibă la un moment dat $d_G[u] < m/n$:

Cazul I: La început se introduc în coadă toate nodurile cu grad $< m/n$.

Cazul II: Fie u nodul scos din coadă la momentul t. Fie w, pe rând, fiecare nod adiacent cu u. Dacă în urma instrucțiunii $-- d_G[w]$ gradul acestuia ajunge să fie $< m/n$, acesta este introdus imediat în coadă, deci va fi eliminat din graf, iar dacă gradul rămâne $>= m/n$, atunci la momentul t știm sigur că el nu trebuie eliminat. Fie x, pe rând, orice nod din G neadiacent cu u, ce nu se află în coadă și ce nu a fost încă eliminat. La momentul t gradul acestuia nu este afectat de eliminarea lui u, iar la momentul $t-1$ știm sigur că acesta avea $d_G[x] \geq m/n$ (altfel x ar fi fost introdus în coadă), deci nici la momentul t acesta nu trebuie eliminat. Deci putem afirma că nici un nod nu va fi ratat în timpul procesului de eliminare a nodurilor.

Complexitate: Fiecare nod poate fi introdus în coadă cel mult o dată, iar fiecare muchie poate fi parcursă tot o singură dată, deci complexitatea timp a algoritmului este $O(N+M)$, unde N = numărul de noduri și M = numărul de muchii, iar complexitatea memoriei este $O(N)$, întrucât se folosește o coadă în care nu pot fi introduse mai mult de N noduri și un vector de dimensiune N pentru a ține minte gradele fiecărui nod. (ignorăm memoria utilizată pentru memorarea grafului).

3.2 (b)

Pentru cerința dată, vom presupune că $n \geq 2$ și $m \geq 1$, adică graful nostru admite cel puțin o muchie. Avem relațiile $d_G(v) \in \{0, 1, 2, \ldots, n-1\}, \forall v \in V, 1 \leq m \leq \binom{n}{2}$ și $0 \leq m/n \leq |E(K_n)|/n, |E(K_n)| = \binom{n}{2}$, unde $n \geq 2$.
Presupunem prin reducere la absurd că graful G' obținut în urma executării algoritmului este nul, adică $G' = K_0$.

Cazul I: $m < n$ în graful inițial $G \Rightarrow 0 < m/n < 1 \Rightarrow \forall Q$ componentă conexă din G cu $|Q| \geq 2$ nu va fi afectată de algoritm $\Rightarrow G'$ nu este nul(contradicție).

Cazul II: $n \leq m < 2n$ în graful inițial $G \Rightarrow 1 \leq m/n < 2$. Fie $Q \subset V$ o mulțime de noduri ce descrie fie un graf complet cu $d_{[Q]_G}(v) \geq 2$, fie un circuit $C_k, k \geq 3$. Cum gradul nodurilor lui $[Q]_G$ este deja $\geq 2 \Rightarrow$ nici după ștergerea tuturor nodurilor $y \in G-Q, y \in N_G(v), v \in Q$ cu $d_G(y) < m/n$ din G, $d_G(v) \geq 2, \forall v \in Q$. Asta înseamnă că, în urma execuției algoritmului, Q nu va fi șters $\Rightarrow G'$ nu este nul(contradicție).

Cazul III: $m \geq 2n$ în graful inițial $G \Rightarrow 2 \leq m/n \leq |E(K_n)|/n$. Fie $Q \subset V$ o mulțime de noduri ce descrie un graf complet cu $d_{[Q]_G}(v) \geq \lceil m/n \rceil, \forall v \in Q$ în G. Cum gradul nodurilor lui $[Q]_G$ este deja $\geq \lceil m/n \rceil \Rightarrow$ nici după ștergerea tuturor nodurilor $y \in G - Q, y \in N_G(v), v \in Q$ cu $d_G(y) < m/n$ din G, $d_G(v) \geq \lceil m/n \rceil, \forall v \in Q$. Asta înseamnă că, în urma execuției algoritmului, Q nu va fi șters $\Rightarrow G'$ nu este nul(contradicție).

Din (**I**), (**II**), (**III**) $\Rightarrow$ presupunerea făcută este falsă $\Rightarrow G'$ este nenul(mai exact, are cel puțin o muchie).

3.3 (c)

Fie G un graf oarecare cu $m \geq 0, n \geq 1$, pe care îl supunem algoritmului descris în ipoteză. Conform **(b)** $\Rightarrow \exists Q \subset V$ o mulțime de noduri a.î. $[Q]_G$ este:

I. o componentă conexă cu $|Q| \geq 2$ pentru $m < n$;

II. fie un circuit $C_k, k \geq 3$, fie un graf complet $K_m, m \geq 3$ pentru $n \leq m < 2n$;

III. un graf complet $K_m, m \geq \lceil m/n \rceil + 1$ pentru $2n \leq m$.

În toate cazurile, $\exists D$ un drum în $[Q]_G$ de lungime cel puțin $m/n \Rightarrow \exists D$ un drum de lungime cel puțin m/n în G pentru $n \geq 2, m \geq 1$. Cum într-un graf gol(fără muchii) orice nod izolat determină un graf complet K_1 cu un drum D de lungime $0/n = 0 \Rightarrow \exists D$ un drum de lungime cel puțin m/n în G pentru $n \geq 1, m \geq 0$.

4 Problema IV

4.1 (a)

Afirmația 1: Algoritmul lui Dijkstra clasic poate fi modificat astfel încât, pentru un digraf $G = (V, E)$, o funcție $a : E \to R_+$ și un nod s din care toate celelalte noduri sunt accesibile, acesta să calculeze atât un vector ce conține costul minim al unui drum de la nodul de start s la oricare dintre noduri, cât și o submulțime de arce $A, A \subseteq E, |A| = |V| - 1$, iar $T = (V, A)$ să fie un arbore.

Demonstrație: Fie următorul algoritm:

Algorithm 1 Algoritmul lui Dijkstra, cursul 4

```
1: S ← s; before[s] ← 0; u_s ← 0;
2: for (i ∈ V\{s}) do
3:     u_i ← a_si; before[i] ← s;
4: while (S ≠ V) do
5:     find j* ∈ V\{S} s. t. u_j* = min{u_j : j ∈ V\S};
6:     S ← S ∪ {j*};
7:     for (j ∈ V\S) do
8:         if (u_j > u_j* + a_j*j) then
9:             u_j ← u_j* + a_j*j; before[i] ← j*;
```

E suficient să introducem o singură instrucțiune în acest algoritm pentru ca acesta să construiască și mulțimea A, și anume, vom introduce in bucla *while*, după instrucțiunea numărul 6, următoarea instrucțiune:

$$7 : A \leftarrow A \cup \{before[j^*]j^*\}$$

Întrucât instrucțiunea 4 se repetă de $|V| - 1$ ori, înseamnă că la final $|A| = |V| - 1$, iar cum orice nod din graful G este accesibil din nodul s, atunci sigur există și cel puțin un drum între s și $i \in V\backslash\{s\}$, drum al cărui cost este mai mic decât ∞ (valoarea a_{si} atunci când arcul si nu se găsește în graf), deci $A \subseteq E$ (vom alege numai arce din graful G). Cum știm că nu se formează un ciclu? Pentru ca acest lucru să se întâmple ar trebui ca cel puțin un nod j^* găsit în bucla while să fie adiacent cu 2 noduri ce au fost deja introduse în mulțimea S, însă noi adăugăm în mulțimea A câte o singură muchie la fiecare pas: $before[j^*]j^*$.

Fie $G = (V, E)$ digraful din enunț, $a : E \to R_+$ funcția din enunț și $x_0 \in V$ un nod din digraful G din care toate celelalte noduri sunt accesibile. Fie $T = (V, A)$ arborele returnat de algoritmul descris în afirmația 1, algoritm aplicat pe digraful G, funcția a și $s = x_0$.

Afirmația 2: În arborele T, pentru orice nod $u \in V$, costul minim al drumului de la x_0 la u este egal cu costul minim al unui drum drum de la x_0 la u în digraful G.

Demonstrație:

Presupunem că pentru un nod $u \in V$ există $D_2(x_0, u)$ în G de cost mai mic decât $D(x_0, u)$ în T. Dar drumul $D(x_0, u)$ în T găsit cu ajutorul algoritmului descris în afirmația 1, aplicat pe digraful G, ceea ce înseamnă că algoritmul descris în afirmația 1 este greșit, pentru că nu a găsit drumul de cost minim $D_2(x_0, u)$, ci a găsit un alt drum, $D(x_0, u)$, care are cost mai mare. Dar algoritmul descris în afirmația 1 este

exact algoritmul lui Dijkstra, care este corect, deci presupunerea inițială este falsă, deci drumurile din arborele T sunt de cost minim.

Arborele $T = (V, A)$ returnat de algoritm respectă criteriile din enunț, deci T este un $SP - arbore$, iar cum algoritmul descris în afirmația 1 găsește mereu un arbore T într-un digraf ce respectă specificațiile, înseamnă că un $SP-arbore$ există întotdeauna.

4.2 (b)

1: $A \leftarrow \varnothing$; // inițial mulțimea arcelor e vidă
2: S $\leftarrow x_0$; before$[x_0] \leftarrow 0$; $u_{x_0} \leftarrow 0$;
3: **for** $(i \in V \backslash \{x_0\})$ **do**
4: $\quad u_i \leftarrow a_{x_0 i}$; before[i] $\leftarrow x_0$;
5: **while** $(S \neq V)$ **do**
6: $\quad$ find $j^* \in V \backslash \{S\}$ s. t. $u_{j^*} = min\{u_j : j \in V \backslash S\}$;
7: $\quad$ S $\leftarrow$ S $\cup \{j^*\}$;
8: $\quad A \leftarrow A \cup \{before[j^*]j^*\}$
9: $\quad$ **for** $(j \in V \backslash S)$ **do**
10: $\quad\quad$ **if** $(u_j > u_{j^*} + a_{j^* j})$ **then**
11: $\quad\quad\quad u_j \leftarrow u_{j^*} + a_{j^* j}$; before[i] $\leftarrow j^*$;
12: $T \leftarrow (V, A)$;
13: *return* T;

După cum am arătat la **(a)**, algoritmul de mai sus găsește întotdeauna un $SP - arbore$ într-un digraf în care există un nod s din care toate celelalte noduri sunt accesibile.

Construirea mulțimii A se face în $(n-1)$ pași, unde $n = |V| - 1$: folosind vectorul $before$, $before[nod]$= nodul aflat înainte de nodul nod pe drumul de cost minim de la s la nod, se adaugă $n-1$ arce de tipul $before[nod] \rightarrow nod$. Tabloul unidimensional $before$ este numit și "vector de tați", iar faptul că algoritmul descris construiește un astfel de vector confirmă faptul că $T = (V, A)$ este un graf conex aciclic maximal.