Tema 1

Mocanu Ada-Astrid
&
Ciripescu Teodor

Grupa A7

8 Noiembrie 2019

Problema 1

a) Ne dorim sa aratam ca $\forall u \in V(G)$ si $\forall v \in V(G), \exists$ un drum orientat de la u la v daca si numai daca $\forall$ ar fi $e \in E(G)$, $G \setminus \{e\}$ ramane conex.

($\Rightarrow$) Pp. ca $\exists e \in E(G)$ a.i. $G \setminus \{e\}$ nu e conex.

Atunci, $\exists u, v \in V(G)$ a.i. in $G \setminus \{e\}$ nu $\exists$ drum de la u la v si de la v la u.

Dar din ipoteza, $\forall u, v \in V(G), \exists$ un drum orientat de la u la $v \Rightarrow$ Contradictie.

Asadar, $\forall u, v \in V(G)$ a.i. $\exists$ un drum orientat de la u la v IMPLICA $\forall e \in E(G)$, $G \setminus \{e\}$ ramane conex.

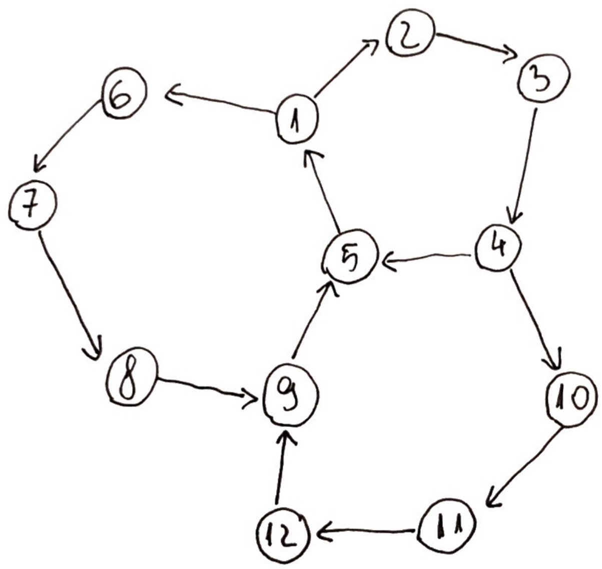
($\Leftarrow$) (este implicata de corectitudinea algoritmului de la subpunctul (b))

b)

```
procedure COMPUTE
    nr_circuit ← 1
    for each nod_curent ∈ V(G) do
        if nu am parcurs toate nodurile then
            parcurgere(nod_curent, nr_circuit)      ▷ incepem ciclul nr k din nodul u(daca ∃ drumul)
            nr_circuit ← nr_circuit + 1              ▷ trecem la ciclul/circuitul urmator
procedure PARCURGERE(nod_curent, nr_ciclu)
    vizitare[nod_curent] ← nr_circuit
    for each nod ∈ V(G) do
        if muchia nod_curent − nod are alocat un sens then
            pastram sensul
            continuam parcurgerea din nod
        else
            if ∃ muchia nod_curent − nod then
                ii dam sens de parcurs de la nod_curent spre nod
                continuam parcurgere din nod
    ne oprim cand ajungem din nodul din care am pornit
```

Grafurile care respecta proprietatile cerute sunt formate din mai multe cicluri alaturate, intre care pot exista coarde. (analogie cu "floricelele")
Prin parcurgerea fiecarui ciclu, dand sens fiecarei muchii si formand un circuit, vom ajunge sa vizitam fiecare nod pana la incheierea programului.
(Astfel incat pentru fiecare ciclu "vecin" cu un circuit deja format, vom incepe acel nou circuit urmand sensul circuitului sau "vecin", unde vecin se

refera la faptul ca au muchii comune).



Problema 2

a) $(\Rightarrow) X \subseteq V$ este uv - separatoare minimala si demonstram ca orice nod din X are vecini in ambele componenete.

Pp. Reducere la Absurd
$X \subseteq V$ este uv - separatoare minimala a.i. $\exists x \in V(X)$, care nu are vecini in ambele componente.
$\Rightarrow$
Caz 1. x nu are vecini in niciuna din componenete $\Rightarrow u$ poate fi eliminat din multimea $X \Rightarrow X \setminus \{x\}$ este uv - separatoare si are cardinal $< X \Rightarrow X$ nu e minimal.
Caz 2. x are vecini intr-o singura componenta $C_1 \Rightarrow$ putem muta x din X a.i. $x \in C_1 \Rightarrow \exists Y = X \setminus \{x\}$ care este uv - separatoare si are cardinal $< X \Rightarrow X$ nu este minimal.

$\Rightarrow$Ambele False $\Rightarrow \forall x \in X$, x are vecini in ambele componente rezultate prin eliminarea tuturor nodurilor din X.

$(\Leftarrow)$ Daca $\forall$ nod din X are vecini in ambele componente, atunci $X \subseteq V$ nu este uv - separatoare minimala (adica $X \subseteq V$ este doar uv - separatoare).

$\Rightarrow \exists Y \subseteq X$ submultime a lui X.
$\Rightarrow \exists x \in X$, $x \notin Y$ ai $xu \in E(G), xv \in E(G)$ unde $u \in C_1, v \in C_2$

$\Rightarrow$ cum $x \notin Y \Rightarrow x$ poate apartine uneia dintre cele 2 componente, spre exemplu, $x \in C_1$.
Dar asta inseamna ca prin eliminarea lui Y nu putem obtine 2 componente conexe (pentru ca xv actioneaza ca punte) $\Rightarrow$ Contradictie.
$\Rightarrow$ Daca $\forall$ nod din X are vecini in ambele componente, atunci X este uv - separatoare minimala.

b) -

Problema 3

a) Vom alege o implementare cu matrice de adiacenta, deoarece observam ca implementarea cu liste simplu inlantuite genereaza o complexitate egala cu $O(n^2) + O(n * m)$. Pe de alta parte, utilizand o matrice de adiacenta, obtinem o complexitate de $O(n^2)$. De mentionat ca utilizam un vector d, pentru a memorarea gradelor nodurilor. Crearea matricei are complexitate $O(n^2)$. Parcurgerea nodurilor se face in $O(n)$, iar verificarea gradului nodului in $O(1)$. Eliminarea nodului din graf se face in $O(1)$, insa este necesara si refacerea constanta a vectorului de grade, in $O(n)$, fapt ce este repetat pentru fiecare nod $\Rightarrow O(n) * O(n) = O(n^2)$. Asadar, complexitatea timp a algoritmului este $O(n^2)$.

```
G' ← G                                          ▷ O(n²)
while ∃u ∈ V(G') a.i. d_G'(u) < m/n do          ▷ O(n) + O(1)(parcurgere + interogare vector grade)
    G' ← G' − u                                 ▷ O(1) + O(n)(eliminare + refacere vector grade)
return G'
```

b) Dorim sa observam evolutia raportului muchii-noduri in graful G'.
Pp ca la Pasul nr. 1 eliminam un nod si implicit k muchii. Astfel, vom avem $n_1 = n - 1$ si $m_1 = m - k$.
Comparand raportul m/n cu m_1/n_1, obtinem ca acesta este echivalent cu compararea lui $m * n - m$ cu $m * n - k * n$
$\Rightarrow$ avem de comparat $k*n$ cu m. Dar $k < m/n$ deoarece k este gradul nodului pe care il eliminam $\Rightarrow m/n < m_1/n_1$
Aplicam INDUCTIE
Presupunem ca $m_{p-1}/n_{p-1} < m_p/n_p$.
Fie $m_{p+1} = m_p - y$, unde y este gradul unui nod eliminat si $n_{p+1} = n_p - 1$.
Vom compara m_p/n_p cu m_{p+1}/n_{p+1}. Fapt care se reduce la compararea lui $y * n_p$ cu $m_p \Rightarrow y$ trebuie comparat cu m_p/n_p
Cum $m_p/n_p > m/n$, dar $m/n > y \Rightarrow y < m_p/n_p \Rightarrow m_p/n_p < m_{p+1}/n_{p+1}$.
Fie k pasul final din executarea algoritmului.
Pp ca graful rezultat are 0 muchii$\Rightarrow m_k = 0 \Rightarrow m_k/n_k = 0$. Dar $m_k/n_k > m/n \Rightarrow 0 > n/m$
$\Rightarrow$Contradictie $\Rightarrow$ Graful nu poate fi niciodata nul.

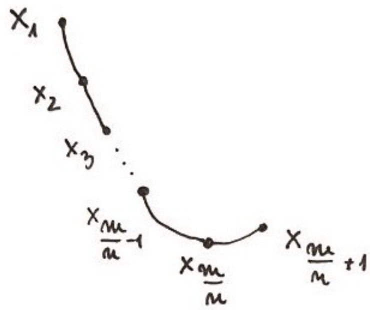
c) $\forall v, d(v) \geq \frac{m}{n}$

Din punctul b) $\Rightarrow \forall$ graf rezultat prin taierea tuturor nodurilor cu grad$< \frac{m}{n}$ este nenul.

Cu alte cuvinte, $\forall v, d(v) \geq \frac{m}{n} \Rightarrow \forall v \in V(G')$ are macar $\frac{m}{n}$ vecini $\Rightarrow \exists$ cel putin $\frac{m}{n} + 1$ noduri in G'.

Un nod x_1 are $\frac{m}{n}$ vecini dintre care unul este x_2 (generand astfel un drum de lungime 1).

$x_2 \in V(G') \Rightarrow d(x_2) \geq \frac{m}{n} \Rightarrow x_2$ are si el $\frac{m}{n}$ vecini $\Rightarrow$ are macar $\frac{m}{n} - 1$ vecini diferiti de x_1; unul dintre ei fiind x_3, iar x_3 are macar $\frac{m}{n} - 2$ vecini, diferiti de x_1, x_2 (generand drum de lungime 3).



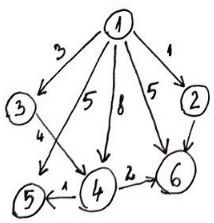
Astfel, ajungem la $x_{\frac{m}{n}-1}$ care are macar 2 vecini diferiti, de $x_1, x_2, ..., x_{\frac{m}{n}-2}$, unul fiind $x_{\frac{m}{n}}$ (gen. drum de lungime $\frac{m}{n} - 1$).

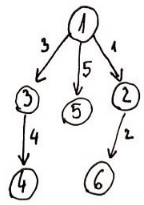
Asadar, $x_{\frac{m}{n}}$ are macar 1 vecin diferit de $x_1, x_2, ..., x_{\frac{m}{n}-1}$, notat $x_{\frac{m}{n}+1}$ si el genereaza un drum de lungime $\frac{m}{n}$.

$\Rightarrow$ Graful contine un drum de lungime cel putin $\frac{m}{n}$.

Problema 4

a) $x_0 \in V$ din care toate celelalte noduri sunt accesibile $\Rightarrow G$ conex
Avem si o functie de cost $a : E \to \mathbb{R}+ \Rightarrow \nexists$ cicluri negative
$\Rightarrow$ putem aplica algoritmul lui Dijkstra, astfel generand un arbore cu drumuri minime(de la un varf la toate celelalte)





In exemplul alaturat am pornit din nodul 1

b) Fie a o matrice de dimensiune $n \times n$, care reprezinta matricea cost drumuri.

$$a[i][j] = \begin{cases} 0, & \text{daca i=j (nu exista cost de la un nod la el insusi)} \\ cost(arc), & \text{daca } \exists \text{ arc de la i la j} \\ \infty, & \text{altfel} \end{cases}$$

Fie x_0 nodul din care vom porni pentru aflarea drumului de la x_0 la oricare alt nod. $drum[i]$ este vectorul in care obtinem drumul de la x_0 la i de cost minim.

```
procedure COMPUTE
    for each  u ∈ V(G) a.i. u ≠ x0  do
        drum[u] ← a[x0][u]
selectat[v] ← 1
k ← 1
    while k ≤ n − 1 do
        min ← ∞
        for each  u ∈ V(G)  do
            if drum[u] < min && (!selectat[u]) then
                min ← drum[u]
                pozmin ← u
        selectat[pozmin] ← 1
        for each  u ∈ V(G)  do
            if !selectat[u] then
                if drum[u] > drum[pozmin] + a[pozmin][u] then
                    drum[u] ← drum[pozmin] + a[pozmin][u]
            k ← k + 1
```