Algoritmica grafurilor

Tema 2

Mihăeş Andrei - Grupa A1

Moniry-Abyaneh Daniel - Grupa A1

2 Decembrie 2015

Problema 1

a)

Presupunem prin reducere la absurd că $k_l(s,t,G) < p_l(s,t,G)$.

Pentru a elimina toate drumurile ar trebui să scoatem $k_l(s,t,G)$ noduri. Scoţând un nod, eliminăm unul sau mai multe drumuri, dar drumurile fiind disjuncte (nu au nici un nod comun) $\Rightarrow$ nu puteam scoate mai multe drumuri cu un nod $\Rightarrow$ Presupunerea e falsa $\Rightarrow$ $k_l(s,t,G) \geq p_l(s,t,G)$.

b)

Pentru graful anterior avem următoarele relaţii:

- Pentru l = 1 şi l = 2 nu există drum de la s la t.
- Pentru l = 3 $p_3(s,t,G) = 1$ şi $k_3(s,t,G) = 1 \Rightarrow p_3(s,t,G) = k_3(s,t,G)$
- Pentru l = 4 $p_4(s,t,G) = 1$ şi $k_4(s,t,G) = 2 \Rightarrow p_4(s,t,G) < k_4(s,t,G)$
- Pentru l = 5 $p_5(s,t,G) = 2$ şi $k_3(s,t,G) = 2 \Rightarrow p_5(s,t,G) = k_5(s,t,G)$
- Pentru l > 5 $p_l(s,t,G) = k_l(s,t,G)$.

  Observăm ca inegalitatea poate fi strictă pentru l = 4. Aşadar există un drum d de la s la t în G pentru care oricare ar fi 2 noduri neadiacente a, b care aparţin drumului d, există mai mult de un drum disjunct de la a la b cu proprietatea că $p_{l-(|d|-|d'|)}(a,b,G) \geq 2$ ("|d|" - lungimea minima a drumului între s şi t, "|d'|" - lungimea minimă a drumului de la a la b) . Prin urmare dacă am elimina numai unul dintre nodurile aflat pe drumul de la a la b, nu ar fi suficient ca să nu mai existe drumuri de la s la t.

c) Valorile lui l pentru care egalitatea $p_l(s,t,G) = k_l(s,t,G)$ este îndeplinită pentru orice graf G şi două vârfuri s şi t sunt $l \geq \frac{|G|-1}{2}$, unde |G| reprezintă numărul de noduri al grafului.

Atunci cand nu exista drumuri de lungime maxim l de la s la t vom avea $k_l(s,t,G) = p_l(s,t,G) = 0 \Rightarrow$ ∀l astfel încât ∄ drum de la s la t atunci avem egalitate pentru orice graf si orice 2 varfuri s si t.

Pentru l = |G|, drumurile de lungime maximă l vor cuprinde toate drumurile de la s la t in G. Aplicand teorema lui Menger se obtine : $p(s,t,G) = k(s,t,G)$ $\Rightarrow p_l(s,t,G) = k_l(s,t,G)$ pentru l = |G|.

Problema 2

a)

Presupunem prin absurd că graful G nu este conex. Deci există cel puţin un vârf expus relativ la cuplajul M, aşadar graful trebuie să fie conex pentru a exista cel mult un vârf expus relativ la cuplajul M.

Presupunem prin absurd că graful G nu este $K_{1,3}$-free. Deci există un subgraf indus de forma $K_{1,3}$în care ar rămâne 2 vârfuri expuse în urma unui cuplaj M, aşadar graful trebuie să fie $K_{1,3}$-free pentru a nu avea mai mult de un vârf expus.

Dacă :

1. |V| = 2k, k ∈ N atunci sunt 0 vărfuri expuse relativ la cuplajul M.
2. |V| = 2k+1, k ∈ N atunci exista un singur vârf expus relativ la cuplajul M.

b)

Cum structura grafului G este $K_{1,3}$-free şi conex, atunci nodurile fii sunt adiacente, doar când cardinalul părintelui > 2. Cum fii sunt adiacenţi, prin parcurgerea DFS se viziteaza mereu primul nod adiacent v, iar apoi DFS(v) (varianta recursiva). Se va vizita celalalt fiu al părintelui înainte ca acesta să fie vizitat chiar de părinte.

În concluzie, structura grafului G ($K_{1,3}$-free şi conex) denotă faptul că arborle obţinul in urma parcurgerii DFS este binar.

c)

Presupunem prin reducere la absurd că există 3 fii descendenţi ce nu formează o clică in graful G.

Aşadar în subgraful indus de cei 3 fii, există 2 vârfuri care nu sunt adiacente.

Prin urmare subgraful indus nu este $K_{1,3}$-free şi conex. (CONTRADICŢIE)

d)

1. Obţinerea unui arbore binar în urma parcurgerii DFS. (similar punctului b)
2. Obţinerea unui nou subarbore cu fii descendenţi a oricărui nod care sunt diferiţi de rădăcină şi formează o clică. (similar punctului c)
3. Parcurgerea în postordine, obţinându-se astfel cuplajul de cardinal $[\frac{V}{2}]$. (similar punctului a)

Complexitatea algoritmului este $2 \cdot O(|E| + |V|) \simeq O(|E| + |V|)$.

Problema 3

a)

Algoritmul Kruskal-clustering are ca input o mulţime de puncte U = {P_1, P_2, ... , P_n} ce poate fi asociată unui graf G=(V,E) si k reprezentând numărul de clase (numărul de componente conexe). Output-ul este o partiţie a mulţimii U cu k clase (S_1,S_2, ... , S_n) Ide calitate maximă reprezentând un subgraf H al lui G de cost minim.

În primele 2 linii ale algoritmului se formează vectorul e ce conţine distanţele între oricare 2 puncte P_i şi P_j ordonate crescător. În linia 3 se construieşte partiţia lui U cu n clase, ({P 1 }, {P 2 }, . . . , {P n }), iniţializând o structura de date pentru utilizarea procedurilor union-find. Prin funcţia union(P, Q) ne putem la referi la adăugarea muchiei intre punctele P si Q din graful G, unde P, Q $\in$ V.

Invariantul buclei while de la linia 5, asigură numărul minim de muchii adăugate pentru a se forma cele k componente conexe. Datorită faptului că structura de distanţe e este sortată crescător asigura proprietatea de drum minim (calitate maximă a partiţiei U). Prin $if\ Find(P) \neq Find(Q) then$ este asigurat faptul că subgraful H obţinut nu are cicluri deorece întotdeauna va fi adăugată o muchie între P şi Q daca aceasta nu se afla în aceaşi componentă conexă.

În concluzie, graful returnat H este un subgraf al grafului G şi de cost minim.

b)

Prima linie a algoritmului are complexitate de $O(\frac{n(n-1)}{2}) \simeq O(n^2)$.

A doua linie are complexitatea în cazul cel mai nefavorabil de $O(n^2)$.

A treia linie are complexitatea de $O(n)$.

Considerăm că structura de date pentru utilizarea procedurilor union-find este implementată pe baza listelor.

Find(x) returnează x->head de complexitate O(1).

Union(x, y) are complexitate $O(L_x + L_y)$, unde L_x şi $L_y$sunt lungimele listelor x şi y. Putem reduce la $O(L_x, L_y) \simeq O(min(L_x, L_y))$. Lungimea maximă a unei liste este de maxim $n - 2 \Rightarrow O(min(L_x, L_y)) \simeq O(n)$.

Incrementarea variabilei added este afectată de alegerea muchiei. Şi cum numărul elementelor din structura de distanţe e este $\frac{n(n-1)}{2}$ atunci complexitatea este de $O(\frac{n(n-1)}{2}) \simeq O(n^2)$.

În concluzie, operaţia predominantă se află în bucla while de complexitate $O(n^2)$ în cadrul căreia se execută funcţia Union de complexitate $O(n)$. Complexitatea algoritmului este $O(n) \cdot O(n^2) = O(n^3)$.