Analiza algoritmilor

Tema 1

Deadline: ~~18.11.2016~~ 22.11.2016
Ora 23:55

Responsabil temă: ***Mihai Nan***

Profesor titular: ***Andrei-Horia Mogoș***

Facultatea de Automatică şi Calculatoare
Universitatea Politehnica din Bucureşti
Anul universitar 2016 - 2017
Seria CC

1 Problema 1

1.1 Clase de complexitate

1. Arătați că:
   - $n \log^3 n = o(n^2)$
   - $\log n! = \Theta(n \log n)$
   - $n! = \Omega(5^{\log n})$

2. Dându-se doi vectori sortați crescător, V_1 și V_2, fiecare conținând n numere naturale, și un număr natural k, scrieți un algoritm, în pseudocod, care să verifice dacă există un element x în V_1 și un element y în V_2 astfel încât $x + y = k$. Specificați complexitatea algoritmului propus.

1.2 Găsirea și rezolvarea unei recurențe

3. Fie următorii doi algoritmi care calculează produsul a două numere naturale, având n cifre.

Algorithm 1 Inmultire 1

```
procedure ALGORITHM1(x, y, n)
    if n = 1 then
        return x * y
    else
        m ← ⌈n/2⌉
        p ← 10^m
        a ← x div p
        b ← x mod p
        c ← y div p
        d ← y mod p
        e ← algorithm1(a, c, m)
        f ← algorithm1(b, d, m)
        g ← algorithm1(b, c, m)
        h ← algorithm1(a, d, m)
        return p^2 * e + p * (g + h) + f
```

Algorithm 2 Inmultire 2

```
procedure ALGORITHM2(x, y, n)
    if n = 1 then
        return x * y
    else
        m ← ⌈n/2⌉
        p ← 10^m
        a ← x div p
        b ← x mod p
        c ← y div p
        d ← y mod p
        e ← algorithm2(a, c, m)
        f ← algorithm2(b, d, m)
        g ← algorithm2(a − b, c − d, m)
        return p^2 * e + p * (e + f − g) + f
```

Găsiți recurența de complexitate pentru fiecare algoritm și aplicați una dintre cele patru metode predate la curs (iterativă, arbore de recurență, substituție sau Master) pentru a determina clasa de complexitate.

1.3 Rezolvarea unei recurențe

4. Găsiți clasele de complexitate pentru:
   - $T(n) = \begin{cases} 2aT(n-1) & \text{dacă } n > 1, a \in N^* \text{ și } a = \text{const} \\ 2 & \text{dacă } n = 1 \end{cases}$
   - $T(n) = \begin{cases} 3T(n^{\frac{1}{3}}) + \log n & \text{dacă } n > 1 \\ 1 & \text{dacă } n = 1 \end{cases}$

5. Găsiți clasa de complexitate, folosind metoda substituției, pentru:
   - $T(n) = \begin{cases} 3T(\frac{n}{5}) + k_2 n^2 & \text{dacă } n > 1, k_2 \in R_+^* \\ k_1 & \text{dacă } n = 1, k_1 \in R_+^* \end{cases}$

2 Problema 2

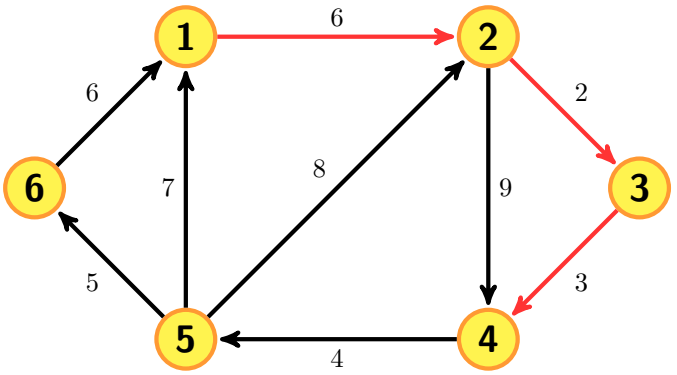
2.1 Drumuri minime în graf

Fiind dat un graf orientat $G = (V, E)$, se consideră funcția $w : E \to W$, numită funcție de cost, care asociază fiecărui arc o valoare numerică. Domeniul funcției poate fi extins, pentru a include și perechile de vârfuri între care nu există arc direct, caz în care valoarea este ∞.

Costul unui drum format din arcele $p_{12}p_{23}...p_{(n-1)n}$, având costurile $w_{12}, w_{23}, ..., w_{(n-1)n}$, este suma $w = w_{12} + w_{23} + ... + w_{(n-1)n}$.

Costul minim al drumului dintre două vârfuri este minimul dintre costurile drumurilor existente între cele două vârfuri.

Exemplu:



$w_{14} = w_{12} + w_{23} + w_{34} = 6 + 2 + 3 = 11$

Pseudocod

```
Dijkstra(sursa, dest):
    selectat(sursa) = true
    foreach nod in V // V = multimea nodurilor
        daca exista muchie[sursa, nod]
            // initializam distanta pana la nodul respectiv
            d[nod] = w[sursa, nod]
            introdu nod in Q
            // parintele nodului devine sursa
            P[nod] = sursa
        altfel
            d[nod] = +∞ // distanta infinita
            P[nod] = null // nu are parinte

    // relaxari succesive
    cat timp Q nu e vida
        u = extrage_min (Q)
        selectat(u) = true
        foreach nod in vecini[u] // (*)
            // drumul de la s la nod prin u este mai mic
            daca !selectat(nod) si d[nod] > d[u] + w[u, nod]
              // actualizeaza distanta si parinte
                d[nod] = d[u] + w[u, nod]
                P[nod] = u
                //actualizeaza pozitia in coada
                actualizeaza (Q,nod)

    // gasirea drumului efectiv
    Initializeaza Drum = {}
    nod = P[dest]
    cat timp nod != null
        insereaza nod la inceputul lui Drum
        nod = P[nod]
```

Algoritmul lui Dijkstra este un algoritm ce se poate folosi pentru a determina drumurile de cost minim de la un vârf de start s la restul vârfurilor în care se poate ajunge într-un graf ponderat cu costuri pozitive. Un exemplu de pseudocod pentru acest algoritm este descris in blocul anterior.

2.2 Cerință

Să se scrie un program C care să implementeze algoritmul lui Dijkstra în patru moduri: folosind un vector sortat la fiecare inserare a unui vârf pentru care s-a determinat distanța de la nodul de start până la el, pentru un graf reprezentat prin matrice de adiacență, respectiv pentru unul reprezentat prin liste de adiacență, și folosind un min-heap, pentru un graf reprezentat prin matrice de adiacență, respectiv pentru unul reprezentat prin liste de adiacență. Se dorește să se determine distanța minimă de la vârful de start la un vârf destinație din graf și vârfurile care compun acest drum de cost minim. În plus, pentru fiecare variantă a algoritmului, programul va număra operațiile elementare care s-au realizat. Prin operație elementară se înțelege o operație foarte simplă: operație aritmetică, comparație, atribuire, indexare etc. (s-a discutat la curs, la metrica unitară și omogenă, despre operații elementare).

Veți primi o serie de teste și veți rula cele patru variante ale algoritmului pentru fiecare test, obținându-se, la fiecare rulare, numărul de operații elementare efectuate de algoritm. Pe baza rezultatelor obținute în urma acestor rulări ale algoritmului, veți realiza un tabel, în fișierul **README**, care va conține toate rezultatele celor 4 variante ale algoritmului. Astfel, fiecare linie a tabelului este asociată unui mod de implementare, iar fiecare coloană este asociată unui test.

Pornind de la acest tabel, veți realiza o comparație a celor 4 implementări realizate, în fișierul **README**, specificând, pentru fiecare test, care variantă a fost optimă și oferind o explicație succintă a acestei concluzii. De asemenea, puteți măsura și timpul de execuție și să specificați cum variază în funcție de implementare, explicând de ce se întâmplă acest lucru.

Rezolvați această problemă în câte fișiere C doriți (puteți adăuga headere sau nu, este la libera voastră alegere), însă realizați un fișier **Makefile** care:

- la comanda **build** va genera un executabil pentru fiecare din cele patru variante de implementare;
- la comanda **clean** se vor sterge fișierele executabile și orice alt fișier generat la compilare;
- la comanda **run-var1** se va rula prima implementare a algoritmului (graf reprezentat prin matrice de adiacență și vector pentru reținerea vârfurilor pentru care s-a determinat distanța minimă față de vârful de start);
- la comanda **run-var2** se va rula a doua implementare a algoritmului (graf reprezentat prin matrice de adiacență și min-heap pentru reținerea vârfurilor pentru care s-a determinat distanța minimă față de vârful de start);
- la comanda **run-var3** se va rula a treia implementare a algoritmului (graf reprezentat prin liste de adiacență și vector pentru reținerea vârfurilor pentru care s-a determinat distanța minimă față de vârful de start);
- la comanda **run-var4** se va rula a patra implementare a algoritmului (graf reprezentat prin liste de adiacență și min-heap pentru reținerea vârfurilor pentru care s-a determinat distanța minimă față de vârful de start);

Numele fișierului de intrare este ***dijkstra.in***, iar cel al fișierului de ieșire este ***dijkstra.out***.

Un fișier de intrare va conține:

- pe prima linie două numere naturale m și n separate printr-un spațiu, fiecare având următoarea semnificație: m - numărul de arce are grafului, n - numărul de vârfuri ale grafului;
- pe a doua linie două numere naturale s și d, separate printr-un spațiu, reprezentând indicele vârfului de start și indicele vârfului destinație;
- pe fiecare dintre următoarele m linii se găsesc câte 3 numere naturale, x, y, z, cu proprietatea: arcul (x, y) al grafului are asociat costul z.

Fișierul de ieșire va conține:

- pe prima linie costul drumului minim de la vârful de start s la vârful destinație d;

- pe a doua linie se vor afișa vârfurile care compun acest drum de cost minim, despărțite printr-un spațiu;
- pe ultima linie a fișierului se va găsi numărul de operații elementare.

3 Punctaj

⚠ IMPORTANT !

⚠ Punctajul complet pe fiecare task se acordă doar dacă task-ul a fost **complet** și **corect** rezolvat.
Tema valorează **1 punct** din nota finală de la AA.

Problema	Punctaj
Problema 1.1.1	15 puncte (3 x 5p)
Problema 1.1.2	10 puncte
Problema 1.2	10 puncte
Problema 1.3.4	15 puncte (5p + 10p)
Problema 1.3.5	10 puncte
Problema 2	40 puncte (30p implementare + 10p comparație)

Atenție!

Tema este individuală! Toate soluțiile trimise vor fi verificate, folosind o unealtă pentru detectarea plagiatului. Orice parte a temei copiată, de pe Internet sau de la colegi, duce la anularea punctajului pentru temă. În cazul copierii de la un alt coleg se anulează și punctajul colegului respectiv pentru această temă.
Dacă o temă este uploadată dupa termenul limită, tema se anulează.

Observație

Se va acorda și punctaj parțial pe exercițiile propuse la **Problema 1** (1), în funcție de cât de mult s-a rezolvat corect.
În cazul **Problemei 2** (2), dacă fișierele sursă nu compilează sau nu rulează corect (conform specificațiilor din enunț), punctajul maxim pe task va deveni 50% din punctajul maxim specificat anterior și se va acorda punctaj parțial, în funcție de cât de mult din task a fost rezolvat. Dacă nu se va realiza o variantă de implementare propusă, nu se va acorda nici punctajul aferent explicațiilor din comparație pentru aceasta.
De asemenea, nerespectarea indicațiilor legate de organizarea codului în foldere, absența fișierului **Makefile** sau reguli greșite în fișierul **Makefile** vor însemna cod care nu funcționează și, prin urmare, se vor aplica regulile de notare menționate în paragraful anterior.

4 Precizări generale pentru trimiterea temei

4.1 Cum se trimite tema?

Creați o arhivă cu denumirea **nume_prenume_grupa.zip**. De exemplu, studentul *Ionel Popescu* de la grupa *326CC* va crea arhiva *popescu_ionel_326CC.zip*.

Încărcați arhiva pe cs.curs.pub.ro până la data indicată (**18.11.2016, ora 23:55**). **NU** se acceptă teme trimise pe e-mail sau altfel decât prin intermediul platformei de curs.

Atenție! Formatul arhivei trebuie să fie **zip**.

4.2 Ce trebuie să conțină arhiva?

Un fișier **README**, având formatul **PDF**, care să fie semnat cu **nume**, **prenume** și **grupă**. Acest fișier trebuie să conțina răspunsul la intrebările și cerințele din enunț. Pentru redactarea fișierului **PDF**, puteți utiliza, de exemplu, LaTeXsau orice procesor de documente pentru desktop sau online în care să puteți edita ecuații (exemple ar fi: Google Docs, LibreOffice Writer, Microsoft Word etc.).

Un folder **aa-tema1-2** care să conțină sursele și Makefile-ul cerute la **Problema 2**(2).

⚠ IMPORTANT !

⚠ Acestea trebuie să se afle în rădăcina arhivei și **NU** în alte foldere!
Fișierul **README NU** poate conține imagini cu rezolvările exercițiilor realizate pe foi.