Assignment 1

Filip Alina-Andreea, CR 1.2.A

April 22, 2019

1 Cerinta

Luam o secventa de 2n numere ca input. Construieste un algoritm cu eficienta 0(nlogn) care imparte secventa in n perechi cu proprietatea ca partittia minimizeaza suma maxima a unei perechi. De exemplu, spunem ca avem secventa (6, 3, 5, 7) , iar posibile partitionari sunt: ((1,6);(3,7)); ((1,3);(6,7)); ((1,7);(3,6)). Suma perechilor acestor partitionari este:(4,13), (7,10), si (8,9).Asadar, a treia partitie are 9 ca suma maxima care este minima dintre cele ale celorlalte pertitii.

Problema ne cere sa afisam partitionarea care prezinta suna minima dintre sumele maxime ale fiecarei partitionari.In exemplul de mai sus:(1,7);(3,6).

2 Algoritmi

Pentru realizarea acestui program am utilizat mai multi algoritmi impartiti in mai multe coduri sursa.

2.1 Algoritmul de sortare QUICK-SORT

QUICK-SORT(A, p, r)

1. if(p¡r)
2. q=PARTITION(A,p,r);
3. QUICK-SORT(A,p,q);
4. QUICK-SORT(A,q+1,r);

Acest algoritm este unul recursiv. Sorteaza in ordine crescatoare numerele dintr-un sir dat. Urmeaza urmatorii pasi:

- Verifica daca pozitia pivotului este mai mica decat pozitia ultimului element in sir. El este considerat initial ca fiind primul element al sirului.
- Daca este indeplinita conditia atunci se calculeaza pozitia noului pivot

- Se reapeliaza QUICK-SORT pentru prima parte a sirului care contine valori mai mici ca pivotul
- Se reapeliaza QUICK-SORT pentru a doua parte a sirului care contine valori mai mari ca pivotul

Acesta este un algoritm implementat dupa modelul celui predat la curs.

Algoritmul are o complexitate egala cu :

$$O(nlogn)$$

Este unul dintre cei mai eficienti algoritmi de sortare.

2.2 Algoritmul de partitionare

PARTITION(A, p, r)

1. x=A[p]
2. i=p
3. j=r
4. while(i¡j)
5. while(A[i]¡x)
6. i++;
7. while(A[j]¿x)
8. j–;
9. if(i¡j)
10. aux=A[i];
11. A[i]=A[j];
12. A[j]=aux;
13. else
14. return j;

Algoritmul PARTITION organizeaza sirul in doua parti diferite, o parte care contine valori mai mici decat pivotul si o parte care contine valori mai mari ca pivotul. De asemenea el returneaza valoarea lui j, fiind pozitia unde i si j se intalnesc.

Algoritmul functioneaza astfel:

- Lui x ii da valoarea sirului in pozitia pivot

- i primeste pozitia primului elem in sirul ce trebuie sortat
- j primeste pozitia ultimului element din sirul ce trebuie sortat
- Cat timp i¡j
- i este incrementat cu 1 de fiecare data cand in sir se gaseste o valoare mai mica decat pivotul
- j este decrementat cu o valoare de fiecare data cand in sir se gaseste o valoare mai mare ca pivotul
- daca i¡j se face interschimbarea intre valorile vectorului in pozitiile lui i si j
- daca nu este indeplinita conditia atunci intoarcem valoarea lui j

Complexitatea algoritmului este:

$$T(n) = 3 + n[1 + m(1 + 1) + p(1 + 1) + 1 + 3l + 1] = O(n)$$

In cel mai bun caz, complexitatea algoritmului poate fi O(1) cand sirul are un singur element. In cel mai rau caz, complexitatea algoritmului poate fi $O(n^2)$ atunci cand elementele sun deja sortate si se intra pe un singur while de n ori (cum ar fi while-ul lui i) .

2.3 Algoritmul de partitionare al perechilor

PARTITIONARE-PERECHI(A, n-2)

1. $for(i = 2; i <= n + 1; i + +)$
2. $aux = A[1];$
3. $A[1] = A[i];$
4. $A[i] = aux;$

Algoritmul realizeaza n-2 interschimbari intre a doua valoare din sirul dat si a treia, a 4-a ...pana la ultima valoare din sir. Algoritmul urmeaza urmatorii pasi:

- i primeste pozitia celui de-al 3-lea elem din vector pana la pozitia ultimului elem din vector
- aux primeste valoarea celui de-al doilea elem din vector
- al doilea elem din vector primeste valoarea elementului de pe pozitia i
- elementul de pe pozitia i primeste valoarea celui deal doilea element din vector

Vectorul este modificat la fiecare incrementare a lui i si fiecare interschimbare intre elemente iar la final este transmis rezultatul final.

Complexitatea algoritmului este :

$$T(n) = 1 + n(3 + 2) = O(n)$$

In cel mai bun caz, complezitatea algoritmului este O(1) cand vectorul are dimensiunea 2 sau mai putin.

2.4 Algoritmul de afisarea a partitionarii(a rezultatului)

AFISARE(A, n)

1. $printf(\text{"}partitionareacusumaminimadintresumelemaximeeste : \text{"})$;
2. $i = 0$;
3. $while(i <= n)$
4. $j = i + 1$;
5. printf(A[i], A[j]);
6. $i = i + 2$;

Algoritmul acesta realizeaza afisarea elementelor din vector doua cate doua. Complexitatea algoritmului este

$$T(n) = 1 + 1 + n/2(1 + 1 + 1 + 1) = O(n)$$

Avem n/2 in loc de n deoarece de atatea ori se intra in while. In cel mai bun caz, complexitatea algoritmului este O(1) cand n=0, adica nu vectorul este vid.

3 Date experimentale

Datele experimentale au fost create cu ajutorul functiei random din main. Un set de astfel de date este urmatorul:14 36 86 95.

3.1 Algoritmul de randomizare al sirului introdus

Acest algoritm se gaseste in main

1. srand((unsigned) time(t));
2. $n = rand()procent10$;
3. $for(i = 0; i < n; i + +)$
4. $A[i] = rand()procent100$;

Algoritmul de randomizare returneaza un sir de numere la intamplare, de o dimensiune aleasa la intamplare.

4 Rezultate si concluzii

La inceput am construit acest program mult mai complicat, utilizand mai multe functii care puteau fi excluse. La inceput algoritmul functiona astfel :

- se genera un vector aleatoriu
- se apela functia quick-sort
- se apela functia determinare-vector-sume-max care returna un vector alcatuit din sumele maxime ale tuturor combinatiilor vectorului. In interiorul acestei functii se regasea un vector al pozitiilor elementelor in vector care ajuta la calculul sumelor fara a modifica vectorul initial. Suma se calculeaza prin apelul unei functii care calculeaza suma elem din vector doua cate doua.
- se apela functia pozitie-minimul-maximelor care returna cate iteratii trebuie sa aiba functia de interschimbare a valorilor sirului
- se apela afisarea vectorului dupa interschimbare

Dupa mai multe teste am ajuns la concluzia ca de fiecare data numarul de interschimbari ce trebuie facute este n-2 si am hotarat renuntarea la functiile ce erau in plus pentru a face programul mult mai eficient. De asemenea am hotarat sa adaug in main niste if-uri pentru a selecta ce trebuie sa faca programul in cazul in care numarul de variabile este egal cu 0, este impar sau este egal cu 2. Am decis sa nu includ in raportul tehnic si sursele din codul initial deoarece ele nu fac parte din codul final. Codul initial cu functiile mentionate poate fi regasit in codul sursa sub forma de comentarii.