Course Assignment

Filip Alina-Andreea
Calculatoare Romana
Anul I
Grupa 2

June 2, 2019

1 Cerinta

THE TASK-SCHEDULING PROBLEM Programeaza destule activitati care presupun folosirea exclusiva a unei resurse comune, cu scopul de a selecta un set de dimensiune maxima a unor activitati compatibile una cu cealalta. Dandu-ti faptul ca o activitate are un timp de incepere si un timp de terminare, doua activitati sunt compatibile daca intervalul intre timpul lor de incepere si cel de terminare nu se suprapune.

Problema ne ofera mai multe activitati cu un timp de incepere si unul de terminare iar noi trebuie sa le organizam astfel incat sa avem un numar maxim de activitati programate.

2 Algoritmi

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

2.1 Algoritmul de sortare a activitatilor dupa ora de terminare

sort(a, nr)

1. for(i= 0; i¡ nr-1;i++)
2. imax = i
3. for(j= i+1;j¡ nr; j++)
4. if (a[imax].end ¿ a[j].end)
5. imax = j
6. if (imax != i)
7. aux1=a[imax].end
8. aux2=a[imax].start
9. a[imax]=a[i]
10. a[i]=aux

Algoritmul sorteaza activitatile date in ordinea crescatoare a timpului lor de terminare. Urmeaza urmatorii pasi:

- iteratorul i ia prima pozitie
- iteratorul timpului maxim ia pozitia lui i
- iteratorul j inpepe de la urmatorul element din fata lui i

- daca timpul de terminare al activitatii de pe pozitia iteratorului timpului maxim este mai mare decat timpul de terminare al activitatii de pe pozitia j
- iteratorul maxim ia valoarea j unde se gaseste noul element maxim
- daca itertorul maxim este diferit de i atunci se face interschimbarea intre activitatea de pe pozitia imax si cea de pe pozitia i impreuna cu datele lor (timpul de terminare si cel de incepere)

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

Complexitatea algoritmului este

$$T(n) = O(n^2)$$

In cel mai bun caz , complexitatea algoritmului este O(1) cand nu este nici o activitate de programat.

2.2 Algoritmul de alegere a solutiei optime

greedy(a, nr)

1. a[0].start
2. a[0].end
3. for(i=1;i¡nr;i++)
4. if(a[i].start¿=a[ultim].end)
5. a[i].start
6. a[i].end
7. ultim=i

Algoritmul alege pe rand activitatea potrivita pentru a urma dupa ultima aleasa. Functioneaza astfel

- intai printeaza prima activitate din cele sortate deoarece aceasta are timpul de terminare minim deci prin urmare aceasta incepe cel mai devreme si se termina cel mai devreme deci ramane destul timp pentru a programa cat mai multe activitati.
- sarind peste prima activitate si luand-o pe prima ca fiind ultima activitate trecuta pe lista cu programari se cauta prima activitate care are un timp de start mai mare decat timpul de finish al ultimei activitati alese
- chiar daca in lista exista activitati care au tun timp de start mai apropiat de cel de finish al ultimei activitati adaugate acelea nu sunt optime deoarece se termina mai tarziu (de aceea apar mai jos in lista) si reduc timpul ramas pentru restul activitatilor.

- activitatea gasita ia rolul ultimei activitati adaugate pe lista

Complexitatea algoritmului este :

$$T(n) = O(n)$$

In cel mai bun caz, complexitatea algoritmului este O(1) cand nu exista nici o activitate de programat.

2.3 Algoritmul de printare a activitatilor

print(a, nr)

1. FILE* test-out
2. for (i= 0; i¡ nr ; i++)
3. fprintf(test-out," id: d time start: d time end: d", a[i].id, a[i].start, a[i].end)

Algoritmul scrie in 10 fisiere output-ul corespunzator fiecarui fisier in cu date astfel:

Intai apar activitatile sortate dupa timpul de finish apoi apare ordinea activitatilor programate impreuna cu timpul lor de start si finish in ore si minute.

Complexitatea este: O(n).

2.4 Algoritmul de generare a activitatilor

generate(a, nr)

1. fscanf(test-in,"d", nr);
2. for(i = 0; i ¡ nr; i++)
3. activities[iterator].id = iterator;
4. fscanf(test-in, "d", a[i].start);
5. fscanf(test-in, "d", a[i].end);

Aceasta functie citeste din fisierul dat numarul de activitati din lista si timpurile pentru fiecare si le intorduce in structura unei activitati.

Complexitatea este: O(n).

3 Date experimentale

Datele experimentale au fost create cu uneui program ce scrie date random in 10 fisiere sursa ce sunt folosite la inceputul programului ca date de intrare.

3.1 Algoritmul de randomizare a datelor de intrare

Acest algoritm se gaseste in alt program pus in fisierul cu codurile sursa.

1. for(j=1;j¡=10;j++)
2. sprintf(temp-name-in, "teste
   test-procentd.in", j+1);
3. FILE* test-in = fopen(temp-name-in, "wt");
4. n=rand()30;
5. while(n¡=0)
6. n=rand()30;
7. fprintf(test-in, "d", n)
8. for(i=1; i¡=n; i++)
9. timp-end=rand()1440
10. while(time-end¡=0)
11. timp-end=rand()1440
12. timp-start=rand()timp-end+1;
13. while(time-start¡0 or time-start==1440)
14. timp-start=rand()timp-end+1;
15. fprintf(test-in, "d", timp-start);
16. fprintf(test-in, "d", timp-end);

Programul scrie in 10 fisiere numarul evenimentelor ce sunt pe lista iar pe fiecare linie, pentru fiecare activitate se scrie timpul de incepuere si timpul de terminare a activitatii.

Numarul de activitati ia valori pana la 30 deoarece am decis ca asa exte mai usor sa verific fiecare aoutput al problemei.

Timpul de finish ia valori pana la 1440 deoarece atatea minute exista intr-o zi, iat timpul de start ia valori pana la valoarea aleasa ca timp de finish deoarece nu putem incepe o activitate dupa timpul de finish.

In functie se mai gasesc si multiple while-uri care trateaza cazurile rele de input cum ar fi daca se da un numer negativ sau 0 de activitati sau daca timpul de incepere si de finish sunt negative.

Complexitatea algoritmului este O(n).

4 Aplication design

4.1 Input

Input-ul este reprezentat de numarul de activitati date si de timpul de start si finish al fiecareia.

Numarul de activitati este de tip intreg si ia valori intre 0 si 30.

Timpurile sunt tot de timp int si sunt reprezentate in minute pentru a fi mai usor sa facem comparatia intre ele si deoarece activitatile se programeaza in ore si minute, nu se tine cont de secunde. Daca am trece timpul in secunde ar fi mult mai greu de verificat daca output-ul este corect deoarece am avea numere mult prea mari.

4.2 Output

Output-ul este reprezentat de organizarea activitatilor date in ordinea crescatoare a timpului de finish. Am hotarat sa adaug si asta la output deoarece este usor sa verifica daca functia de sortare functineaza pentru fiecare caz si deoarece este mai usr de verificat raspunsul problemei.

In output apare ordinea evenimentelor impreuna cu orele de start respectiv finish iar in paranteza este precizat id-ul de pe lista initiala a activitatilor.

4.3 Module

Problema este organizata in 2 fisiere sursa , unul contine functia main si nu face altceva decat sa citeasca fiecare fisier si sa apeleze functiile in ordinea data pentru acesta. Al doilea fisier contine functiile apelate in scopul rezolvarii problemei si se numeste greedy deoarece el contine metoda greedy ce consta in sortarea si alegerea solutiei optime a probelemi date.

4.4 Functions

4.4.1 Fisierul sursa Main

In main gasim apelul unor functii din greedy pe care le utilizam la rezolvarea problemei.

Pe prima linie apare declaratia unei structuri a activitatilor care contine variabile de timp intreg (id, timp start, timp end) si care contine detaliile despre fiecare activitate data. Pe a doua linie sunt declarate variabile de tip intreg care reprezinta numarul de activitati introduse si un iterator care contorizeaza fisierele sursa introduse si cele scrise astfel incat pentru fisierul test1.in se va scrie un fisier test1.out care contine rezolvarea corespunzatoare datelor din test1.in. Fisierele de output sunt deschise cu wt astfel incat daca fisierul nu a fost creat acesta se va crea iar daca a fost creat acesta va fi suprascris. Fisierele de input sunt deja create de functia random dar daca nu au fost create inca existaun if care indica eroarea.

Prima data , din fisier se citeste numarul de activitati date ce urmeaza a fi folosit in apelul functiilor.

Se aloca spatiu peuntru toate activitatile unde o activitate are dimensiunea structurii.

De aici se trece la executia unor functii existente in cel de al 2-lea fisier sursa al programului.

4.4.2 Fisierul sursa Greedy

Se apeleaza functia de generare a activitatilor, aceasta nu face altceva decat sa citeasca timpurile de start si finish ale activitatilor si ii atribuie fiecarei activitati un id.

Se apeleaza functia sort care sorteaza activitatile dupa timpul de finish.

Functia compara fiecare activitate cu cele din fata ei iar daca se gaseste una care are un timp de finish mai mic atunci se face interschimbarea intre elementele primei activitati si cele ale celei gasite. Variabilele din functie sunt urmatoarele: iterator-1 si iterator-2 de tip intreg si ajuta la parcurgerea listei de activitati, index-max (tip intreg) reprezinta pozitia activitatii care contine timpul maxim de finish si variabilele aux-time-start, aux-id, aux-time-end sunt variabile de tip intreg care retin elemnetele din structura activitatii pentru a ajuta la a interschimbarea pozitiilor activitatilor in lista fara a pierde valori.

Se apeleaza functia de printare a activitatilor in fisier. Fisierul out este deschis cu at astfel incat ce este deja scris in fisier nu se va pierde si se va scrie in continuarea sa.

Se apeleaza functia de rezolvare a problemei si alegere a solutiei optime care va scrie solutia problemei tot in continuarea fisierului out.

Functiile de generare, printare si greedy au ca parametru si variabila de tip intreg i astfel incat functia sa stie in ce fisier trebuie sa fie introduse datele returnate.

Toate functiile utilizate sunt de tip void si nu intorc dini un parametru in functia main dar realizeaza schimbari asupra datelor.

Dupa apelarea functiei greedy se inchide fiecare fisier deschis.

Cele de mai sus se repeta de 10 ori caci atatea ficiere avem ca in.

5 Rezultate si concluzii

La fiecare proiect am invatat lucruri noi pe care nu le stiam si mi-am dat seama ca nu toti pasi spre rezolvarea uni probleme din viata reala se aplica si la pasi pe care ii descriem pe calculator prin coduri ci trebuie sa gasim cai pentru a construi un program mai eficient si mai usor de inteles.

Nu am reusit sa ofer o rezolvare modulara si in care sa folosesc functii in python insa doresc sa invat cat mai mult despre acest limbaj pe viitor deoarece mi se pare un limbaj asemanator cu c-ul insa tot odata diferit. Multe coduri se aseamana insa modul in care le scrii este diferit fata de c.

Ca sugestie pe viitor as dori sa se preadea python in laboratoare pentru ca studentii sa inteleaga mult mai bine acest limba. Una este cand citesti singur si incerci sa te lamuresti si una este cand un om care deja stie limbajul iti explica codul.

6 References

https://www.infoarena.ro/metoda-greedy-si-problema-fractionara-a-rucsaculu

https://www.youtube.com/watch?v=cHa85et7LK0&list=PLS1QulWo1RIYt4e0WnBp-ZjCNq8X0FX0J&index=8

http://www.aut.upt.ro/~rraul/PC/2009-2010/PC-lab10.pdf

https://sites.google.com/site/eildegez/home/clasa-xi/prezentarea-metodei-greedy

https://www.dyclassroom.com/dynamic-programming/0-1-knapsack-problem

https://iliepanait.weebly.com/metoda-greedy.html