

RAPORT TEHNIC

COPERTA

Titlul temei: Selectarea activităților

Numele: Drăghici Andreea-Maria

Grupa: CR1.1B

Anul de studiu: Anul I

Specializarea: Calculatoare Romana

Enuntul problemei

Se considera o multime de activitati astfel incat fiecare dintre ele necesita acces exclusiv la o resursa comuna. Pentru fiecare activitate se cunoaste timpul de start si durata. Se cere determinarea unei submultimi de cardinalitate maxima a acestor activitati de astfel incat oricare doua activitati selectate sunt mutual compatibile(nu se suprapun in timp).

Algoritmii propuși

Metoda 1

Presupunem ca detinem o multime $A=1,2,\dots,n$ unde n = numarul de activitati care folosesc aceeasi resursa comuna. Aceasta resursa poate fi folosita de o singura activitate la un moment dat, deoarece acestea nu trebuie sa se suprapuna in acelasi timp. Fiecare activitate are un timp de start si un timp de terminare ($\text{start}[\text{iterator}] \geq \text{finish}[\text{constant}]$).

Presupunem ca activitatile sunt ordonate crescator dupa timpul de terminare: $t_1 \leq t_2 \leq \dots \leq t_n$

Complexitatea timpului acestui algoritm este $\theta(n)$, unde n este numarul total de activitati. Metoda Greedy determina cea mai optima solutie posibila.

Denumirile variabilelor in limbajul C, respectiv Python difera fata de algoritmul in pseudocod, am incercat sa scriu algoritmul in pseudocod cat mai simplu si concis, iar implementarile pentru cod sunt adaptate in functie de fiecare algoritm in parte.

GREEDY-ALGORITHM(*start, finish*)

```
1  n=s.length
2  A={a1}
3  j=1
4  for i=2 to n
5      if s[i]>= f[j]
6          A=A U {ai}
7          j=i
8  return A
```

Figure 1: Metoda Greedy

Metoda 2

Implementam metoda Bubble pentru a sorta crescator activitatile dupa ora de final. Selectam prima activitate ,cea care se termina cel mai devreme, dupa aceea vom selecta, la fiecare pas, prima activitate neselectata, care nu se suprapune peste cele deja selectate. Afisam activitatile in functie de ora de inceput si final, selectand durata acestora

Complexitatea timpului acestui algoritm este $O(n)$, unde n este numarul total de activitati.

BUBBLE-SORT-ALGORITHM (*A*)

```
1  for i=0 to n-2
2      for j=0 to n-1
3          if A[j] > A[j+1]
4              aux=A[j]
5              A[j]=A[j+1]
6              A[j+1]=aux
```

Figure 2: Metoda Bubble Sort

Date experimentale

1 Random Generators

Am folosit o metodă pentru generarea automată de date de intrare, astfel încât determinarea unei submultimi de cardinalitate maximă a activităților trebuie să aibă oricare două activități mutual compatibile în funcție de începutul și durata fiecărei activități.

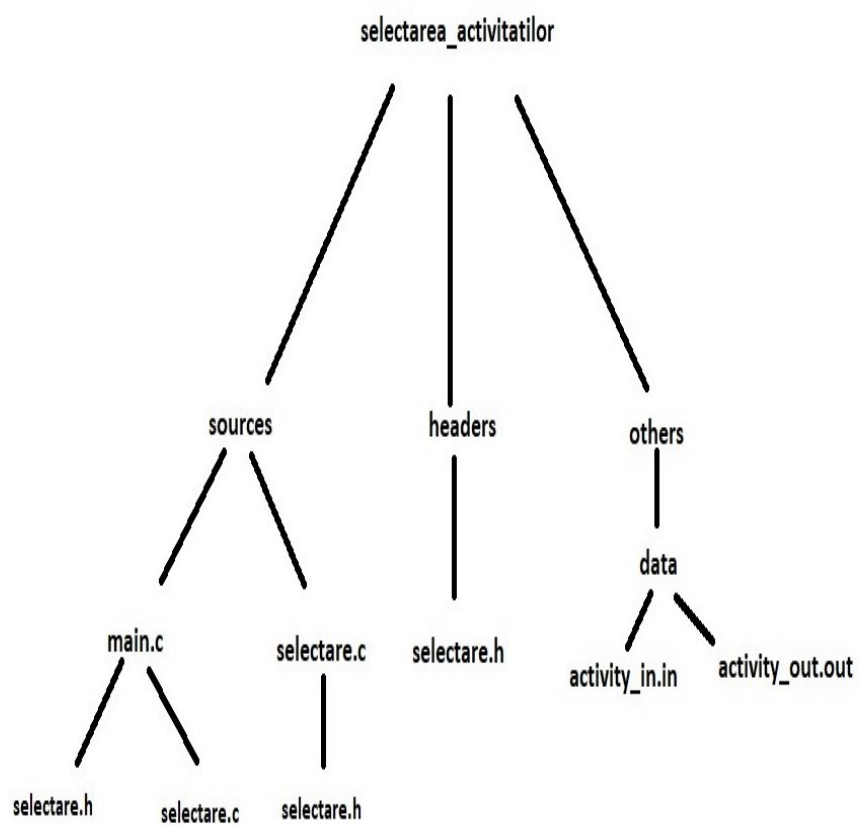
Random Generator Algorithm

1. $\text{limit} = \text{RAND_MAX} - (\text{RAND_MAX} \% n)$
2. **while** $r = \text{rand}() \geq \text{limit}$
3. **return** $r \% n$
4. $\text{srand}(\text{time}(0))$
5. **for** $i = 1$ to 40
6. **return** randRange

Am folosit funcția *rand* pentru a returna un număr întreg aleator situat în intervalul $[0, \text{RAND_MAX}]$, unde RAND_MAX este egal cu $2^{31}-1$. Cu ajutorul buclei *for* putem specifica limita unei secvențe de valori, astfel funcția va returna un număr aleator selectat din intervalul nostru, dar fără să includă limita sfârșitului, adică valoarea 40.

Proiectarea aplicației

2 Structura de nivel înalt a aplicației



Apelul functiei principale :

```
#include <stdlib.h >
#include <stdio.h >
#include <time.h >
#include <selectare.h >
#include <selectare.c >
```

selectarea_activitatilor : main.o selection.o

```
< T > gcc - o selectarea_activitatilor main.o se-
lectare.o main.o : main.c selectare.c selectare.h
```

```
< T > gcc - c main.c
```

objects = main.o selection.o

selectarea_activitatilor : \$(objects)

```
< T > gcc - o selectarea_activitatilor $(objects)
```

\$(objects):selectare.h

selectare.c : selectare.h

3 Specificarea datelor de intrare

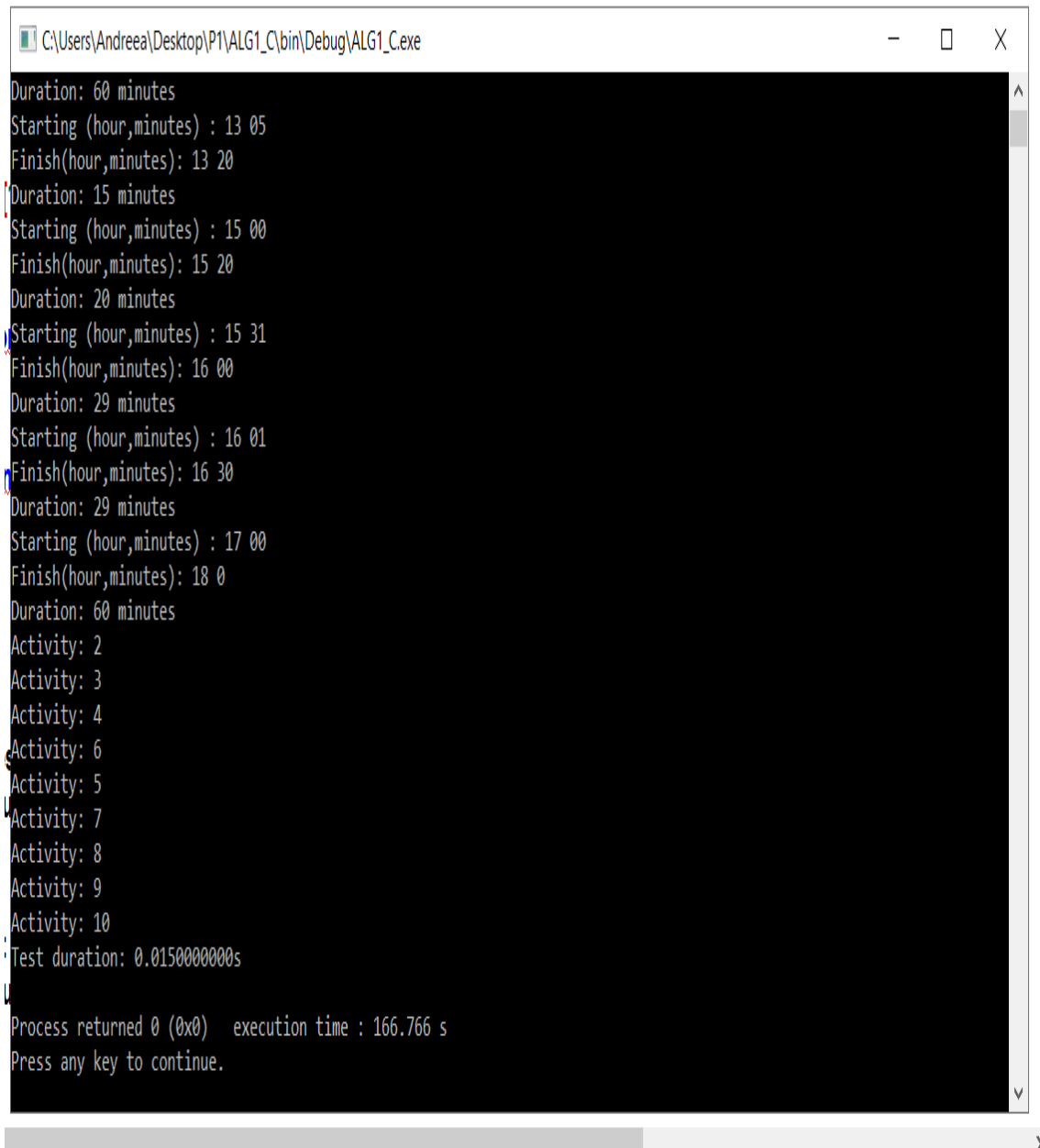
Sortam crescator activitatile dupa ora de final.

Datele de intrare sunt reprezentate de numarul de activitati pe care le avem, se cunoaste timpul de inceput, durata si timpul de sfarsit.

Fiecare din aceste activitati necesita acces exclusiv la o resursa comuna.

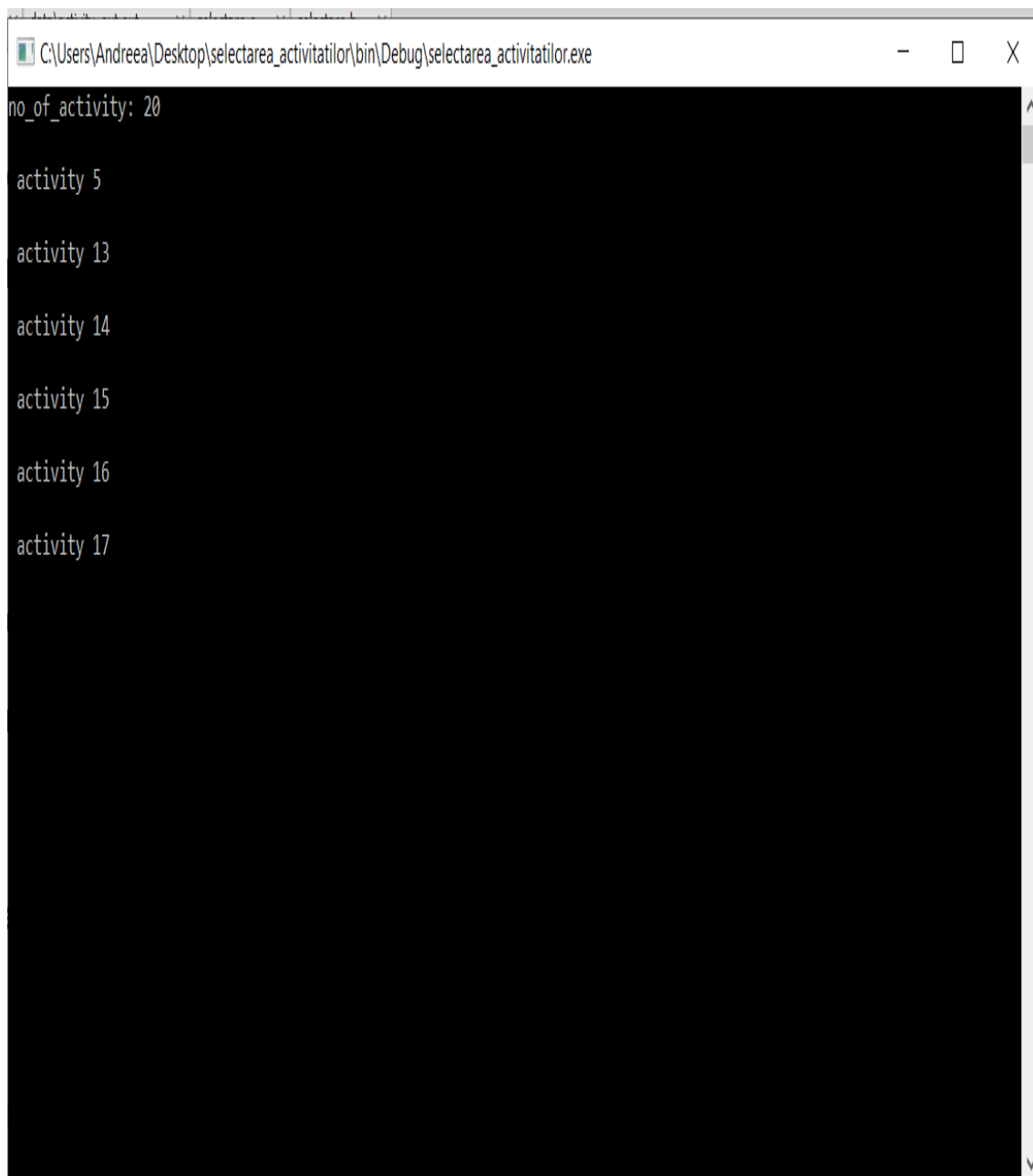
4 Specificarea datelor de iesire

Determinam submultimea de cardinalitate maxima a acestor activitati, astfel incat oricare doua activitati selectate sunt mutual compatibile, adica exploram toate submultimile de activitati astfel incat oricare doua activitati din aceeasi submultime sa nu se suprapuna in acelasi timp. Selectam prima activitate, cea care se termina cel mai devreme, dupa aceea vom selecta, la fiecare pas, prima activitate neselectata, care nu se suprapune peste cele deja selectate.



```
C:\Users\Andreea\Desktop\P1\ALG1_C\bin\Debug\ALG1_C.exe
Duration: 60 minutes
Starting (hour,minutes) : 13 05
Finish(hour,minutes): 13 20
Duration: 15 minutes
Starting (hour,minutes) : 15 00
Finish(hour,minutes): 15 20
Duration: 20 minutes
Starting (hour,minutes) : 15 31
Finish(hour,minutes): 16 00
Duration: 29 minutes
Starting (hour,minutes) : 16 01
Finish(hour,minutes): 16 30
Duration: 29 minutes
Starting (hour,minutes) : 17 00
Finish(hour,minutes): 18 0
Duration: 60 minutes
Activity: 2
Activity: 3
Activity: 4
Activity: 6
Activity: 5
Activity: 7
Activity: 8
Activity: 9
Activity: 10
Test duration: 0.0150000000s
Process returned 0 (0x0)  execution time : 166.766 s
Press any key to continue.
```

Figure 3: Rezultate Metoda 2_C



```
C:\Users\Andreea\Desktop\selectarea_activitatilor\bin\Debug\selectarea_activitatilor.exe
no_of_activity: 20
activity 5
activity 13
activity 14
activity 15
activity 16
activity 17
```

Figure 4: Rezultate Metoda 1_C

The image shows a Python IDE window titled 'ALG1_Python' with a file named 'main.py'. The code defines a function 'activities()' that takes a list of activities and returns an array of selected activities. The code uses a greedy algorithm to select activities based on their start and finish times. The output window shows the execution of the 'activities()' function with the following input and output:

```
activities()
no of activity 6
start's hour 5 45
finish's hour 12 15
Process finished with exit code 0
```

```
10 arr[0] = 0
11 arr[1] = duration[1]
12 arr[2] = duration[2]
13 iterator: int
14 constant: int
15 constant = 1
16 aux: int
17 aux = 1
18 for iterator in range(no_of_activity):
19     if start[iterator] >= finish[constant]:
20         aux = aux + 1
21         arr[aux] = arr[iterator]
22         constant = iterator
23         print('Activity %d \n ' % iterator)
24 arr[0] = aux
25 return arr
```

Figure 5: Rezultate Metoda 1_Python

The image shows a screenshot of a Python IDE window titled "ALG2_Python". The editor displays a file named "main.py" with the following code:

```
33 def write():
34     last: int
35     i: int
36     iterator: int
37     for last in range(n):
38         if start[iterator[i]] >= finish[iterator[last]]:
39             print('Activity %d \n' % iterator[i] + 1)
40
41 def number_generate(min,max):
42     number=random.randint(min,max)
43     return number
44 n=number_generate(1,10)
45 print("No of activities: " ,n)
46
47 hour1=number_generate(0,23)
48 minutes1=number_generate(0,59)
49 hour2=number_generate(0,23)
50 minutes2=number_generate(0,23)
51
52 print( "Finish(hour,minutes): " , hour1,minutes1)
53 print( "Starting(hour,minutes): " , hour2, minutes2)
```

Below the code editor, the "Run" panel shows the execution output for the "main" script:

```
"C:\Users\Andreea\Desktop\activities selection-9.4.2020\venv\Script"
No of activities:  1
Finish(hour,minutes):  13 48
Starting(hour,minutes):  11 23
Process finished with exit code 0
```

Figure 6: Rezultate Metoda 2_Python

5 Lista tuturor modulelor aplicatiei si descrierelor

```
void read() ; //functia pentru citit
```

```
void sort(); //functia pentru sortarea activitatilor
```

```
void write(); //functia pentru afisare
```

```
int randRange(int value); //functia pentru care cardinalitatea maxima este stocata
```

```
int main(int argc, char **argv); //functia care genereaza random numerele
```

6 Descrierea scopului pentru fiecare functie

Fiecare implementare contine anumite functii care "impart" programul principal in anumite subprograme separate. Aceste functii sunt grupate in acelasi cod, cod care corespunde elementelor(parametrilor) declarate in interiorul acestor functii. (de ex: "int value", "int argc, char **argv"...restul parametrilor care corespund fiecarei functii din cod) .

Fiecare functiei este folosita cu un anumit scop in programul principal :

`void read() ; //afișăm activitățile în funcție de ora de început și de sfârșit, selectând durata lor`

`void sort(); //sortăm activitățile în ordine crescătoare după ora de sfârșit, selectând activitatea care se termina cel mai devreme`

`void write(); //afișează activități care nu se suprapun în același timp`

`int randRange(int value); // returneaza o valoare uniformă în intervalul (0, n-1) pentru orice n pozitiv`

7 Descrierea parametrilor

`int value // stocheaza cardinalitatea maxima a activitatilor`

`int no_of_activity // reprezinta numarul total de activitati intr-un anumit timp`

8 Semnificatia valorilor de return

`return random % value; //returneaza o solutie random`
`return arr; //returneaza multimea de activitati`

Concluzii

- Am incercat sa fac ambele implementari ale algoritmilor, atat in limbajul C, cat si in limbajul Python.
- Am incercat sa respect fiecare cerinta din metodologie, astfel incat sa pot descrie fiecare parametru in functie de implementarile pe care le am.
- Am folosit Metoda Greedy, deoarece aceasta determina cea mai optima solutie.
- Am folosit Metoda Bubble Sort pentru a sorta crescator activitatile.
- Consider ca a fost o provocare(din care am incercat sa invat cat mai multe), atat gandirea celor doua metode, cat si implementarea acestora in cele doua limbaje de programare.
- Am incercat sa folosesc si un algoritm pentru generarea aleatoare a numerelor, dar consider ca nu este cea mai buna metoda pe care am incercat sa o folosesc.
- Ca o extindere a studiului pe termen mai lung, consider ca referinta ajutatoare in intelegerea si parcurgerea temelor de casa, cartea Introduction to Algorithms(3rd edition)-Thomas H. Cormen

Rezumatul rezultatelor

-Rezultatele care trebuiau obtinute in cazul de fata erau reprezentate de cardinalitatea maxima a activitatilor astfel incat oricare doua activitati sa nu se suprapuna in acelasi timp.

-Rezultatele obtinute pentru cele doua implementari se pot observa in Figurile 3, 4, 5 si 6. In figura 3 rezultatele obtinute sunt ordonate crescator in functie de timpul de sfarsit, pe cand in figura 4, rezultatele obtinute reprezinta cardinalitatea maxima a activitatilor, adica am explorat toate submultimile de activitati care nu se suprapun in acelasi timp.

Bibliografie

Mai jos se pot observa cateva referinte bibliografice, capitole din cursuri, carti, site-uri web, referinte ce au ajutat in parcurgerea si intelegerea mai ampla a temei de casa:

References

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein, Introduction to Algorithms (3rd edition), MIT Press and McGraw-Hill, 2009

Chapter12/Capitolul12

Chapter6/Capitolul6

Infoarena — www.infoarena.ro

Limbajul LaTeX:lista principalelor comenzi

<http://andrei.clubcisco.ro/cursuri/>

[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))