

Raport tehnic

June 8, 2021

1 Coperta

Titlul temei: Planificarea sarcinilor

Numele: Bucă-Ghebaură Elizabetha Alexandrina

Grupa: CR 1.1 A

Anul de studiu: Anul I

Specializarea: Calculatoare și Tehnologia Informației în Limba Română

2 Enunțul problemei

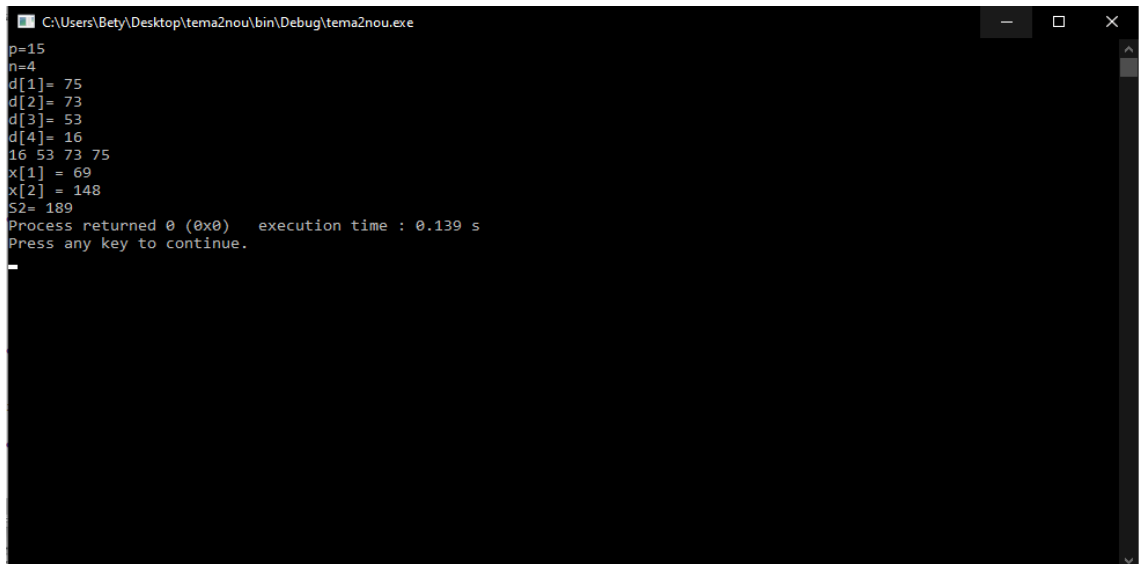
Se considera $p \geq 1$ procesoare identice și $1 \leq n \leq k \times p$ sarcini de lucru, unde k este un parametru fixat, număr natural. Se cere determinarea planificării optimale a celor n sarcini pe cele p procesoare disponibile, astfel încât pe fiecare procesor să fie executate cel mult k sarcini. Se cunoaște durata de execuție d_i a fiecărei sarcini. Se notează cu $d_M = (\sum_{i=1}^n d_i)/p$ media timpului de execuție al tuturor sarcinilor pe cele p procesoare. Dacă se notează cu x_j durata cumulată de execuție a sarcinilor planificate pe procesorul $1 \leq j \leq p$ atunci planificarea optimă a sarcinilor trebuie să minimizeze dezechilibrul total față de timpul mediu, calculat prin $\sum_{j=1}^p |x_j - d_M|$. Se vor implementa doi algoritmi, primul pentru $k=2$ și al doilea pentru $k=3$.

3 Algoritmii propuși

Pentru prima metodă avem datele pe care le-am generat random, și anume numărul procesoarelor și cel al sarcinilor de lucru. Am implementat metoda MERGE SORT pentru a sorta în ordine crescătoare timpii de execuție ai fiecărei sarcini. În acest fel, pentru a doua parte a problemei unde trebuie să se calculeze media timpului de execuție, am însumat fiecare timp de execuție al fiecărei sarcini, în final împărțind la numărul de procesoare. Mai departe, am calculat folosind o buclă durată cumulată de execuție astfel încât pe fiecare procesor să fie cel mult două sarcini de lucru.

Mergesort

- a) if ($p < r$) // avem cel puțin două elemente
- 1. $q = (p+r)/2$
- 2. mergesort(x, p, q) // sortează x[p:q]
- 3. mergesort(x, q+1, r) // sortează x[q+1:r]
- 4. mergesort(x[p:q], x[q+1:r]) // combină subșirurile



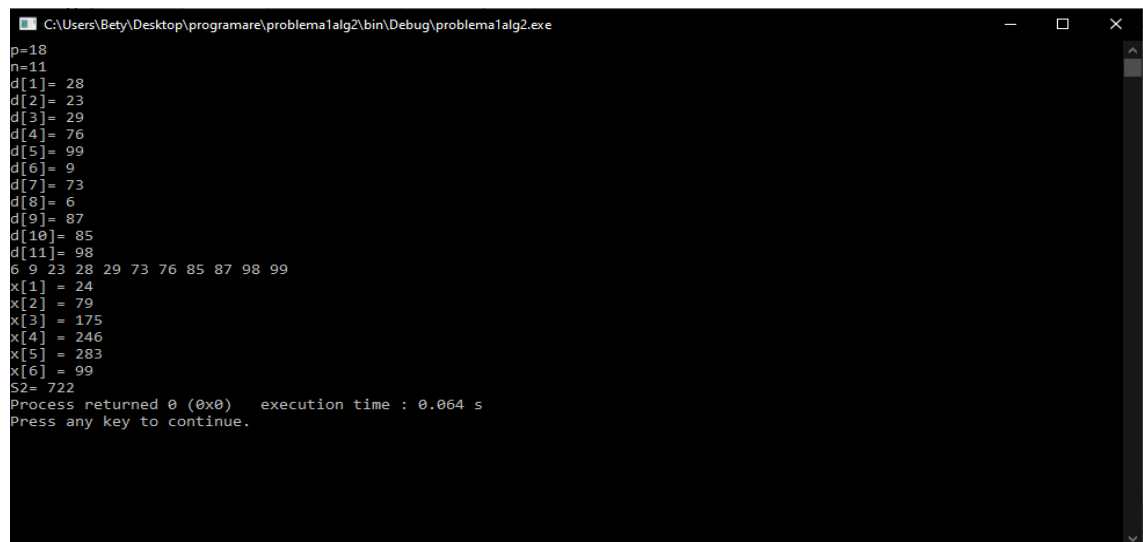
```
C:\Users\Bety\Desktop\tema2nou\bin\Debug\tema2nou.exe
p=15
n=4
d[1]= 75
d[2]= 73
d[3]= 53
d[4]= 16
16 53 73 75
x[1] = 69
x[2] = 148
S2= 189
Process returned 0 (0x0) execution time : 0.139 s
Press any key to continue.
```

Figura 1

Pentru a doua metodă, ca și pentru prima metodă, am folosit funcția de generare aleatorie a procesoarelor și sarcinilor de lucru. Quicksort efectuează sortarea bazându-se pe o strategie divide et impera. Astfel, el împarte lista de sortat în două subliste mai ușor de sortat. Am încercat să modific prima metodă, astfel încât pe fiecare procesor să fie executate cel mult $k=3$ sarcini de lucru.

QuickSort

1. quickSort(arr[], low, high)
2. if (low < high)
3. pi = partition(arr, low, high)
4. quickSort(arr, low, pi - 1)
5. quickSort(arr, pi + 1, high)



```
C:\Users\Bety\Desktop\programare\problema1alg2\bin\Debug\problema1alg2.exe
p=18
n=11
d[1]= 28
d[2]= 23
d[3]= 29
d[4]= 76
d[5]= 99
d[6]= 9
d[7]= 73
d[8]= 6
d[9]= 87
d[10]= 85
d[11]= 98
6 9 23 28 29 73 76 85 87 98 99
x[1] = 24
x[2] = 79
x[3] = 175
x[4] = 246
x[5] = 283
x[6] = 99
S2= 722
Process returned 0 (0x0)   execution time : 0.064 s
Press any key to continue.
```

Figura 2

4 Date experimentale

Am folosit această funcție pentru a genera aleatoriu numărul de procesoare și numărul sarcinilor de lucru.

Random number generator

```
int main()
{
    srand(time(NULL));
    int i;
    for(i=0; i<7; i++)
        printf("%d ", rand());
    return 0;
}
```

Funcția `rand` returnează un număr natural pseudo-random. Pentru a limita intervalul său de valori, putem folosi operația modulo (%), așa cum am folosit-o și în algoritmul propus de mine în proiectul C. Pentru a evita generarea acelorasi numere random la rularea programului, se folosește `time(NULL)`.

Astfel că, în programul meu, am setat ca numărul procesoarelor și numărul sarcinilor de lucru să nu depășească 2 cifre, pentru fi mai ușor de calculat ceea ce este cerut în problemă.

5 Proiectarea aplicației experimentale

Algoritmul MergeSort

Considerăm șirul: $x[1], x[2], \dots, x[n]$.

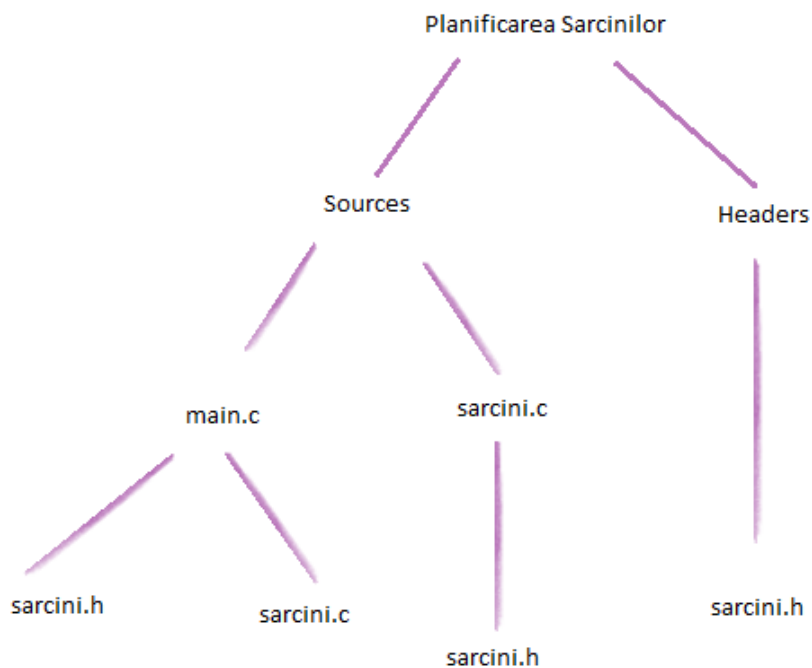
- * Se descompune șirul inițial nesortat în subșiruri de lungime 1.
- * Se interclasează subșirurile obținute la pasul precedent până rezultă un singur șir.

Algoritmul QuickSort

- * se alege un element al listei, denumit pivot
- * se reordonează lista astfel încât toate elementele mai mici decât pivotul să fie plasate înaintea pivotului și toate elementele mai mari să fie după pivot; după această partiționare, pivotul se află în poziția sa finală
- * se sortează recursiv sublista de elemente mai mici decât pivotul și sublista de elemente mai mari decât pivotul

5.1 Structura de nivel înalt a aplicației

Această figură reprezintă programul meu, prima metoda de rezolvare a problemei, cu toate sursele create; similar această schiță arată și pentru cea de-a doua metodă.



Apelul funcției principale

Pentru metoda 1, similar și pentru metoda 2. #include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <sarcini.c>
#include <sarcini.h>

5.2 Specificarea datelor de intrare

Am implementat prima metoda pentru a sorta crescator durata de execuție a fiecărei sarcini.

Datele de intrare sunt reprezentate de numărul procesoarelor și numărul sarcinilor de lucru.

Pentru prima metoda, din prima figura, **INPUT**-urile vor fi urmatoarele:

- p=15
- n=4
- d[1]= 75
- d[2]= 73
- d[3]= 53
- d[4]= 16

5.3 Specificarea ieșirilor/rezultatelor

Datele de ieșire sunt reprezentate de media timpului de execuție a tuturor sarcinilor, precum și dezechilibrul total din ultima parte a enunțului.

Pentru prima metoda, cea din prima figura, acesta va fi **OUTPUT**-ul:

- S2=189

5.4 Lista tuturor modulelor aplicației și descrierea lor

```
void merge()          // combină două subșiruri
void mergeSort()      // procedură recursivă care ordonează un
subșir precizând limitele acestuia
void printArray()      // afișează vectorul sortat
srand(time(NULL))     // generează numere random
void swap()           // interschimbă două elemente
int partition ()      // reordonează lista in funcție de fiecare ele-
ment
void quickSort()      // procedură recursivă care ordonează un subșir
```


5.5 Lista tuturor funcțiilor aplicației, grupate pe module

5.5.1 Descrierea scopului funcției

Ambele metode conțin funcții care prin divizarea acestora în sub-programe ajută la rezolvarea mai rapida a problemei.

```
void merge()          // combină două subșiruri arr[]. primul este
arr[1...n]. al doilea este arr[m+1...r]
void mergeSort()      // procedură recursivă care ordonează subșirul
arr[]
void printArray()     // afișează vectorul sortat (timpul de execuție
al fiecărei sarcini)
srand(time(NULL))     // generează numărul procesoarelor, cel
al sarcinilor de lucru și timpul de execuție pentru fiecare sarcină de
lucru, random
```

5.5.2 Descrierea fiecărui parametru

```
int p                // numărul procesoarelor
int n                // numărul sarcinilor de lucru
int d[i]             // timpul de execuție al fiecărei sarcini de pe cele p
procesoare
int x[j]             // durata cumulata de execuție a sarcinilor planificate
pe procesorul j
```

5.5.3 Semnificația valorii de return

```
return S2           // returnează valoarea dezechilibrului total
```

6 Concluzii

- * Am încercat să fac ambele implementări ale algoritmilor, atât în C, cât și în Python.
- * În Python am întâmpinat câteva probleme, și anume niște erori pe care nu le pot desluși, din păcate nu am reușit să termin algoritmul în acest limbaj.
- * Am folosit algoritmul de sortare Merge Sort pentru a sorta crescător timpul de execuție al fiecărei sarcini.
- * Prima metoda pe care am folosit-o pentru a rezolva problema o consider optimă și mai ușoară decât a doua metodă.
- * Am folosit și un algoritm de generare aleatorie a numerelor, ceea ce mi-a dat puține bătăi de cap la început, însă mi-am dat seama mai târziu cum se folosește.
- * Pentru metoda a doua am încercat să folosesc un alt algoritm de sortare decât Merge Sort, anume Quick Sort.
- * Din nefericire, nu am reușit să rezolv până la capăt a doua metoda încercată, dar consider că este un început.
- * De asemenea, am încercat să fac și un fișier de tip OUT, astfel încât atunci când am rulat programul, în consolă să nu apară nimic, dar să fie mutate datele de acolo în fișierul creat de mine.

7 Referințe bibliografice

- * Chapter 6/ Capitolul 6
- * <https://www.overleaf.com/learn>
- * <https://infogenius.ro/biblioteca-standard-c/>
- * Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Cliff Stein, Introduction to Algorithms (3rd edition), MIT Press and McGraw-Hill, 2009
- * <https://www.geeksforgeeks.org/>