Cu Cărțile pe Raft

Proiectarea Algoritmilor - Tema de casă
Raport Tehnic

Sticlan Daiana-Valentina

Împărțirea cât mai echitabilă în trei secțiuni a unui raft de cărți
Se urmărește ca angajații să citească un număr cât mai apropiat de pagini, pentru distribuirea muncii într-un mod corect

Universitatea din Craiova
Facultatea de Automatică, Calculatoare și Electronică
Calculatoare și Tehnologia Informației
Anul I
CR1.3B

Mai 2024

1 Introducere

1.1 Enuntul problemei

Imaginați-vă un scenariu în care trei angajați primesc sarcina de a căuta informații specificate prin rafturile pline de cărți. Pentru a asigura o împărțire corectă și eficientă a muncii, **cărțile trebuie alocate celor trei fără a fi necesar să le miște sau să le împartă în grămezi separate**. Soluția cea mai simplă este să împartă raftul în **trei secțiuni distincte**, fiecare secțiune fiind alocată unui angajat.
Dar cum putem asigura că împărțirea este echitabilă?
Dacă fiecare carte ar avea aceeași dimensiune, soluția ar fi simplă: împărțim raftul în secțiuni egale,astfel:
100 100 100 – 100 100 100 – 100 100 100.
În acest mod, fiecare persoană se ocupă de 300 de pagini. Cu toate acestea, provocarea apare atunci când cărțile variază în dimensiune. Utilizând aceeași metodă pentru un raft cu cărți de dimensiunile următoare:
100 200 300 – 400 500 600 – 700 800 900,
este clar că distribuția este inegală.
Prima secțiune are un total de 600 de pagini, în timp ce ultima conține nu mai puțin de 2,400 de pagini.
Într-un astfel de caz, cu siguranță am prefera prima secțiune! Pentru a obține o încărcătură de muncă mai echilibrată, o împărțire mai bună ar putea arăta astfel:
100 200 300 400 500 – 600 700 – 800 900.
În acest aranjament, cea mai grea sarcină pe care o persoană ar avea-o este de 1,700 de pagini, iar cea mai ușoară este de 1,300 de pagini.
Acest exemplu ilustrează **necesitatea unei abordări strategice** pentru distribuirea cât mai echilibrată a sarcinilor de lucru atunci când se ocupă de sarcini cu dificultate sau dimensiune variabilă.

1.2 Descrierea problemei

Această problemă se referă la distribuirea echitabilă a unei sarcini de lucru între trei angajați. Sarcina constă în căutarea de informații într-un set de cărți aflate pe un raft. Fiecare carte are un număr variabil de pagini, fapt care determină cât de greu sau cât de ușor se citește aceasta.
Dimensiunea cărților citite de un angajat poate fi tradusă prin dificultatea sarcinii întreprinse de el. Din acest motiv, nu numărul cărților care revin fiecărei persoane este relevant în împărțirea convențională a raftului în trei secțiuni, ci numărul de pagini pe care angajații trebuie să le parcurgă este definitoriu în această problemă.
Așadar, provocarea constă în alocarea cărților astfel încât fiecare să citească pe cât posibil un număr aproximativ de pagini, fără a muta sau reordona cărțile de pe raft.

2 Algoritmi

2.1 Pseudocod

(BOOKS.C):

GenerateNumBooks()

START
Set **lower bound** to **1000**
Set **upper bound** to **100000000**
Initialize random number generator with current time
Generate a random number **num** between **lower bound** and **upper bound**
Return **num**
END

GeneratePages()

START
FOR i=0,numBooks-1 do:

Generate a random number of pages between MINpages and MAXpages

END

PrintBooks()

START
FOR i=0,numBooks-1 do:

Print the book's index and number of pages in a table format

END

NumberOfPages()

START
totalPages=0
FOR i=0,numBooks do:

totalPages=totalPages+currentPages

return **totalPages**
END

CreateSectionsFirstMethod()

START
totalPages=NumberOfPages
third=totalPages/3
pages[3]=NULL
FOR i=0,numBooks do:

IF pages[section]+bookPages[i]is less than third then or section=2:

pages[section]=pages[section]+bookPages[i]

ELSE

section=section+1
pages[section]=pages[section]+bookPages[i]

Print the results
END

CreateSectionsSecondMethod()

START
totalPages=NumberOfPages
pages[3]=NULL
firstBreak=0
secondBreak=0
FOR i=0,numBooks-2 do:

pages[0]=pages[0]+bookPages[i]
pages[1]=0
FOR j=i+1,numBooks-1 do:

pages[1]=pages[1]+bookPages[j]
pages[2]=totalPages-pages[0]-pages[1]
IF pages[0] is less than pages[1]

IF pages[0] is less than pages[2]

MinPages=pages[0]

ELSE

MinPages=Pages[2]

ELSE

IF pages[1] is less than pages[2] then:

MinPages=pages[1]

ELSE

MinPages=pages[2]

IF pages[0] is more than pages[1]

IF pages[0] is more than pages[2]

MaxPages=pages[0]

ELSE

MaxPages=Pages[2]

ELSE

IF pages[1] is more than pages[2] then:

MaxPages=pages[1]

ELSE

MaxPages=pages[2]

IF diff is less than minDiff then:

minDiff=diff
firstBreak=i
secondBreak=j

FOR i=0,numBooks do:

IF i is less or equal to firstBreak then:

pages[0]=pages[0]+bookPages[i]

ELSE IF i is less or equal tosecondBreak

pages[1]=pages[1]+bookPages[i]

ELSE

pages[2]=pages[2]+bookPages[i]

Print the results
END

(MAIN.C):
START
numBooks=GenerateNumBooks
GeneratePages()
Print the number of books
PrintBooks()
pages=NumOfPages()
Print the total number of pages
pages=NumOfPages()/3
Print what it means a third of the total number of pages
CreateSectionsFirstMethod()
CreateSectionsSecondMethod()
END

2.2 Functionalitatea programului

Programul este alcătuit din funcții care sunt apelate atât în fișierul main, cât și în interiorul altor funcții. În ceea ce privește rolul pe care îl îndeplinesc în program, funcțiile pot fi clasificate în două categorii:
1) Funcții cu rol organizatoric, care se ocupă de felul în care sunt create datele cu care lucrează programul într-un final și felul în care sunt prezentate, spre a fi ușor de urmărit.

a)funcția **GenerateNumBooks()** - generează random un număr între 1000 și 100000000, care reprezintă cărțile aflate pe raft
b)funcția **GeneratePages()** - prin intermediul unei instrucțiuni FOR, parcurge fiecare carte și generează un numar random din intervalul [MINpages, MAXpages], două macrodefiniții ce pot fi foarte ușor modificate dacă se dorește un alt minim sau maxim de pagini avute de o carte
c)funcția **PrintBooks()** - ajută la orientarea utilizatorului, parcurgând prin intermediul unei instrucțiuni FOR fiecare carte și afișează într-un format prietenos, simulând un tabel, indicele cărților și numărul de pagini pentru fiecare

d)funcția **NumberOfPages()** - parcurge fiecare carte ca și funcțiile anterioare și generează un număr foarte important în cadrul programului, acela al numărului total de pagini

2) Funcții cu rol metodic, care lucrează cu datele generate și obținute pentru a oferi soluția proprie-zisă. Acestea sunt în număr de două, ambele propunându-și să rezolve problema împărțirii echitabile a raftului în trei și având atât minusuri cât și plusuri. Cele două metode lasă loc de comparație.

a)funcția **CreateSectionsFirstMethod()**

Folosește un algoritm de tip Greedy. Algoritmii Greedy sunt o clasă de algoritmi de optimizare care fac alegerea optimă la fiecare pas, cu speranță că aceste alegeri vor duce la o soluție globală optimă. Cea mai bună alegere se face la momentul curent, fără a se ține cont de consecințele acestei alegeri asupra alegerilor viitoare.
În cazul nostru, la fiecare pas (parcurgerea fiecărei cărți), adăugăm paginile cărții curente în secțiune dacă însumate fiind cu cele deja existente nu depășesc o treime din numărul total de pagini. În momentul în care secțiunea a fost completată, procesul se reia pentru a doua secțiune, prin incrementarea index-ului acesteia pe ramura ELSE și însumarea paginilor cărții curente. În cazul secțiunii 3, cu indexul 2 (indexarea se realizează începând cu 0), se va actualiza numarul de pagini conform IF-ului, datorită condiției cu caracter disjunctiv "or section=2".
Acest algoritm este util în ceea ce privește timpul de execuție, având o complexitate O(n) dată de o buclă FOR, însă soluțiile generate nu sunt întotdeuna cele mai optime la nivel global. Adesea, numărul de pagini din a doua și a treia secțiune se dovedește destul de mare față de numărul de pagini din prima.

b)funcția **CreateSectionsSecondMethod()**

Fucția se bazează pe un algoritm de căutare exhaustivă, care evaluează fiecare posibilitate existentă. Acest tip de algoritmi sunt cunoscuți și sub numele de metode brute-force, o denumire sugestivă în ceea ce privește modul riguros în care datele sunt obținute.
În cazul nostru, pentru a străbate cele trei secțiuni în mod corespunzător, se folosesc două FOR-uri imbricate. Se adaugă numărul de pagini ale carții curente treptat, în fiecare dintre primele două secțiuni, ce-a de-a treia fiind actualizată cu restul de pagini din totalul știut. În cadrul primului FOR se actualizează mereu prima secțiune, iar cea de-a este resetată la 0, urmând a fi incrementat în al doilea FOR, împreună cu imediata actualizare a secțiunii trei. De fiecare dată, se va afla minimul și maximul dintre cele trei secțiuni, relevantă pentru problema noastră fiind diferențe dintre aceste două numere obținute. În acest fel ajungem prin comparare la un minim global. Este evident că ne dorim cea mai mică diferență dintre secțiuni. Odată acest minim descoperit, se salvează indicii ce ne ajută să separăm secțiunile, iar acestea sunt create într-un alt FOR.
Deși metoda este una anevoioasă de o complexitate $O(n^2)$. Aceasta oferă soluții optime, spre deosebire de metoda amintită anterior, care lasă loc de unele incertitudini și poate conduce la soluții suboptimale.

În ceea ce privește memoria, aceasta este alocată dinamic în funcție de variabila **numBooks**, cerințele depinzând de numărul random de cărți. Se poate afirma că memoria are o complexitate O(n), unde n reprezintă numărul de cărți. Acest lucru înseamnă că spațiul de memorie necesar crește liniar cu mărimea imputului. Rezultatul este bun, deoarece înseamnă că programul poate gestiona seturi de date mari într-un mod eficient al memoriei.
În cadrul aplicației realizate în Python, nu a fost necesara alocarea și eliberarea de memorie. Python gestionează memoria în mod automat și o eliberează atunci când obiectele nu mai sunt folosite. De asemenea, Python are un sistem de tipuri dinamic, ceea ce înseamnă că nu trebuie să declari tipuri de date înainte de a utiliza o variabilă. O altă diferență notabilă între cele două limbaje de programare o reprezintă faptul că Python stochează informații suplimentare, cum ar fi tipul de date și numărul de referințe pentru fiecare obiect, ceea ce poate duce la un consum mai mare de memorie.
Așadar, deși complexoitatea memoriei este O(n), diferențele în stocare se datorează felului în care fiecare program lucrează cu memoria.

2.3 Utilizarea aplicației

Programul realizează împărțirea echitabilă a unui raft de cărți între trei angajați în funcție de numărul de pagini. Utilizatorul nu trebuie să introducă manual numărul de cărți sau numărul de pagini ale acestora. Datele cu care lucrează programul sunt generate în mod aleator. Numărul de cărți este foarte mare, aplicația dorindu-și să testeze algoritmii de împărțire pentru date de intrare mari. O serie de rezultate sunt salvate în fișiere de tip text, atât în aplicația realizată în C, cât și în aplicația realizată în Python. Formatul este unul prietenos cu utilizatorul. Cu caractere specifice, se simulează un tabel în care este trecut numărul fiecărei cărți și câte pagini are aceasta. Este afișat numărul total de pagini, dar și cât înseamnă o treime din acesta, pentru a putea judeca felul în care împărțirea se realizează, o treime din total reprezentând o secțiune perfectă. În final, apar date legate de numărul de pagini repartizat fiecarui angajat, prin două metode de împărțire care au plusuri și minusuri.
Privit în mod practic, programul are caracter speculativ, raftul de cărți fiind unul exagerat de consistent. Acest lucru se datorează necesității de a studia cum se comportă cei doi algoritmi când vine vorba de date mari.

3 Rezultate

Aplicația corespunzătoare descrierilor se găsește în următorul **REPOSITORY**, atât în Python, cât și în C.

3.1 Aplicația realizată în C

Tabelul următor reprezintă o serie de rezultate obținute în cazul programului C. Este afișat numărul de cărți, totalul paginilor, o treime din acestea și câte pagini îi revin fiecarui angajat odată realizată împărțirea prin ambele metode. Se observă ca metoda 2, cea exhaustivă, duce la o diferență mai mică între numărul de pagini ce-i revin fiecărui angajat.

Nr. Crt.	Program C										
	Timp de execuție	Nr. de carți	Pagini	O treime	Nr Angajat	Rezultate		Dif 1	Dif 2	Total dif 1	Total dif 2	(Total dif 1 > Total dif 2)?
						Metoda 1	Metoda 2	(1-2,2-3,1-3)				
1	0.083	5417	4122120	1374040	1	1373614	1373614	364	726	1828	1452	DA
					2	1373978	1374340	550	174			
					3	1374528	1374166	914	552			
2	1.518	26092	19807140	6602380	1	6602052	6602052	152	458	2576	1052	DA
					2	6601900	6602510	1288	68			
					3	6603188	6602578	1136	526			
3	1.55	28793	21957849	7319283	1	7318762	7319420	810	114	6366	594	DA
					2	7317952	7319306	3183	183			
					3	7321135	7319123	2373	297			
4	2.024	3386	2572780	857593	1	857174	857174	225	551	2066	1414	DA
					2	857399	857725	808	156			
					3	858207	857881	1033	707			
5	0.038	3468	2637167	879055	1	879043	879216	1084	47	4412	868	DA
					2	877959	879169	2206	387			
					3	880165	878782	1122	434			
6	1.762	30432	23158631	7719543	1	7719197	7719793	460	389	3920	778	DA
					2	7718737	7719404	1960	30			
					3	7720697	7719434	1500	359			
7	2.082	31196	23840127	7946709	1	7946344	7946344	225	654	1740	1308	DA
					2	7946569	7946998	645	213			
					3	7947214	7946785	870	441			
8	0.238	2582	1952768	650922	1	650432	651150	372	252	2200	860	DA
					2	650804	650898	728	178			
					3	651532	650720	1100	430			
9	1.224	25307	19189250	6396416	1	6396006	6396437	380	210	3984	962	DA
					2	6395626	6396647	1992	481			
					3	6397618	6396166	1612	271			
10	1.338	24631	18744600	6248200	1	6247796	6248934	74	388	2276	1352	DA
					2	6247870	6248546	1064	288			
					3	6248934	6248258	1138	676			

În cadrul cazului 3 se remarcă o diferență semnificativă de 6366 de pagini între angajați. Utilizând a doua metodă, pentru același caz, diferența este de 594 de pagini. Diferența obținută prin prima metodă este de aproximativ 10 ori mai mare, inificiența având aici un grad crescut.

File Edit View

Numărul de cărți este 28793

Cartea	Pagini
1	747
2	649
3	1323
4	1006
5	406
6	1157
7	1360
8	448
9	705
10	1167
11	1162
12	1017
13	667
14	1458
15	1279
16	232
17	972
18	1062
19	707
20	732
21	1240
22	589
23	1328
24	693
25	465
26	405
27	732

File Edit View

28772	326
28773	1218
28774	696
28775	536
28776	1244
28777	1187
28778	432
28779	314
28780	1208
28781	361
28782	576
28783	939
28784	1203
28785	1035
28786	557
28787	114
28788	885
28789	1008
28790	313
28791	233
28792	1438
28793	844

Numerul TOTAL de pagini este 21957849
O TREIME de pagini este reprezentata 7319283
===Rezultatele in cazul unei metode greedy===
Angajatul 1 este responsabil de 7318762 pagini.
Angajatul 2 este responsabil de 7317952 pagini.
Angajatul 3 este responsabil de 7321135 pagini.
===Rezultatele in cazul unei metode de cautare exhaustive===
Angajatul 1 este responsabil de 7319420 pagini.
Angajatul 2 este responsabil de 7319306 pagini.
Angajatul 3 este responsabil de 7319123 pagini.

Tot pe baza tabelului putem observa eficacitatea metodei în ceea ce privește o primă secțiune echitabilă. În cazul 1, 2, 4 și 7, prima secțiune din raft are număr egal de pagini în urma calculului cu ambele metode. Așadar, metoda 1 se dovedejte eficientă în unele cazuri când vine vorba de primul angajat. Totuși, tot metoda doi realizează o împărțire corectă generală.

1	0.083	5417	4122120	1374040	1	1373614	1373614	364	726	1828	1452	DA
					2	1373978	1374340	550	174			
					3	1374528	1374166	914	552			
2	1.518	26092	19807140	6602380	1	6602052	6602052	152	458	2576	1052	DA
					2	6601900	6602510	1288	68			
					3	6603188	6602578	1136	526			

...

					3	7321135	7319123	2373	297			
4	2.024	3386	2572780	857593	1	857174	857174	225	551	2066	1414	DA
					2	857399	857725	808	156			
					3	858207	857881	1033	707			

...

7	2.082	31196	23840127	7946709	1	7946344	7946344	225	654	1740	1308	DA
					2	7946569	7946998	645	213			
					3	7947214	7946785	870	441			

Cel mai mare timp de execuție este 2.082 s, lucru motivat de numărul mare de cărți generate (31196).

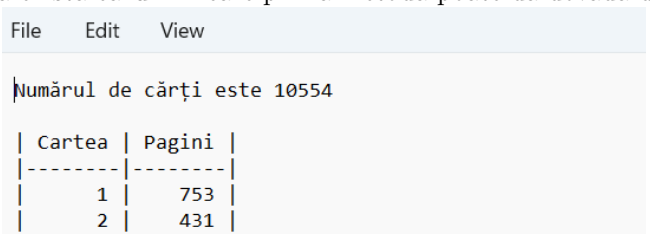
3.2 Aplicația realizată în Python

În tabelul următor sunt înregistrate rezultate ai acelorași algoritmi, dar de data asta implementați în Python.
Și din acest tabel se poate deduce eficiența metodei 2 și observațiile făcute anterior în legătura cu prima secțiune a raftului.

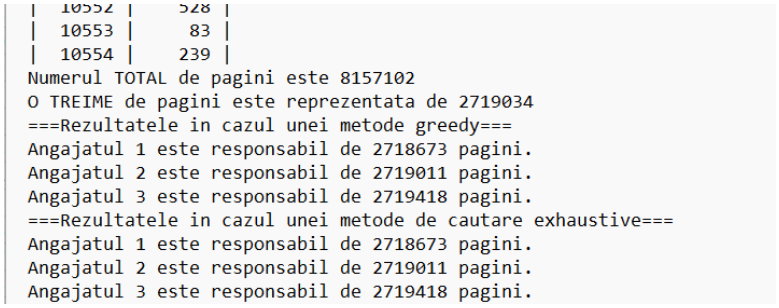
	Program Python								
				Metoda 1			Metoda 2		
Nr. Crt	Nr. Carti	Pagini	O treime	Angajat 1	Angajat 2	Angajat3	Angajat 1	Angajat 2	Angajat3
1	422	342532	114177	112802	113482	116303	114199	113940	114393
2	298	229072	76357	76163	75711	77198	76163	76452	76457
3	994	729999	243333	242953	242836	244210	242953	243963	243083
4	1325	1011494	337164	336727	335976	338791	336727	337413	337354
5	2572	1992231	664077	663475	663757	664999	664239	664214	663778
6	3158	2456352	818784	817907	817629	820816	818833	819252	818267
7	4229	3274087	1091362	1090868	1090599	1092620	1091696	1090984	1091407
8	7397	5714322	1904774	1904318	1904568	1905436	1904318	1904824	1905180
9	10554	8157102	2719034	2718673	2719011	2719418	2718673	2719011	2719418
10	37717	29022341	9674113	9673669	9673755	9674917	9674250	9673804	9674287

Metoda 1			Metoda 2					
Dif 1 (1-2)	Dif 2 (2-3)	Dif 3 (1-3)	Dif 1 (1-2)	Dif 2 (2-3)	Dif 3 (1-3)	Total dif 1	Total dif 2	Total dif 1 > Total dif 2
680	2821	3501	259	453	194	7002	906	Da
452	1487	1035	289	5	294	2974	588	Da
117	1374	1257	1010	880	130	2748	2020	Da
751	2815	2064	686	59	627	5630	1372	Da
282	1242	1524	25	436	461	3048	922	Da
278	3187	2909	419	985	566	6374	1970	Da
269	2021	1752	712	423	289	4042	1424	Da
250	868	1118	506	356	862	2236	1724	Da
338	407	745	338	407	745	1490	1490	Nu => EGALITATE
86	1162	1248	446	483	37	2496	966	Da

Se poate observa un caz pe care nu l-am întâlnit în simularea anterioară folosind limbajul de programare C. Penultimul caz, datorită numărului convenabil de carți și pagini, se dovedește a avea un rezultat bun prin ambele metode. Particular, prima metodă se dovedește a fi la fel de eficientă ca a doua. Acest lucru demonstrează că există cazuri în care prima metodă poate da dovadă de eficacitate crescută.



...



4 Concluzii

Sarcina împărțirii unui raft de cărți în trei secțiuni aproximativ egale ca număr de pagini a fost realizată prin două metode, supuse mai apoi analizei. În urma rezultatelor, s-a putut nota eficacitatea fiecăreia. Realizând lucrarea de față am avut ocazia să înțeleg că o metodă eficientă poate fi anevoioasă ca execuție, iar una simplă și intuitivă se poate să ofere rezultate suboptimale. Implementarea realizată atât în limbajul C, cât și în limbajul Python, a fost un exercițiu bun care m-a ajutat

să sesizez diferențe și similarități între cele două limbaje. Cea mai interesantă constatare făcută o reprezintă felul în care este gestionată memoria de către Python, sarcina de alocare și eliberare nerevedindu-i programatorului.

5 Appendix

5.1 Program C

```c
void sections (struct Book* books, int num_books, FILE* file)
{
    long long total_pages = no_of_pages(books, num_books, file); //salveaza numarul total de pagini
    long long third = total_pages / 3; //calculeaza o treime din pagini, pentru a sti orientativ cate pagini i-ar reveni unui angajat
    long long pages[3] = {0, 0, 0}; //un vector cu trei el;emente care va reprezenta numarul de pagini pentru fievare angajat
    int section = 0; //sectiunea curenta, care poate sa fie 0, 1 sau 2

    for(int i = 0; i < num_books; i++) //parcurgerea cartilor
    {
        //daca suma dintre numarul curent de pagini si paginile cartii curente e mai mica decat o treime din totalul de pagini
        // sau am completat primele doua sectiuni si suntem la sectiunea 2, nr de pagini din sectiune se actualizeaza
        if(pages[section] + books[i].pages <= third || section == 2)
        {
            pages[section] += books[i].pages;
        }
        else //se trece la urmatoarea sectiune si se actualizeaza nr curent de pagini
        {
            section++;
            pages[section] += books[i].pages;
        }
    }
    fprintf(file, "===Rezultatele in cazul unei metode greedy===\n");
    fprintf(file, "Angajatul 1 este responsabil de %lld pagini.\n", pages[0]);
    fprintf(file, "Angajatul 2 este responsabil de %lld pagini.\n", pages[1]);
    fprintf(file, "Angajatul 3 este responsabil de %lld pagini.\n", pages[2]);
}
```

```c
void sections2 (struct Book* books, int num_books, FILE* file)
{
    long long total_pages = no_of_pages(books, num_books, file); //se afla numarul total de pagini
    long long pages[3] = {0, 0, 0}; //se realizeaza vectorul reprezentand sectiunile
    long long min_pages, max_pages, diff, min_diff = total_pages; //se declara variabile pentru a calcula diferenta minima globala, aceasta fiind initializata cu nr maxim de magini
    int first_break = 0, second_break = 0; //pozitiile despartitoare dintre sectoiuni

    for(int i = 0; i < num_books - 2; i++) //parcurge toate cartile incercand fiecare ppunct de impartire
    {
        pages[0] += books[i].pages; //paginile cartii curente sunt adaugate in prima sectiune
        pages[1] = 0; //nr de pagini pentru a doua sectiune este resetat la 0 pentru fiecare i
        for(int j = i + 1; j < num_books - 1; j++)
        {
            pages[1] += books[j].pages;//paginile cartii curente sunt adaugate in a doua sectiune
            pages[2] = total_pages - pages[0] - pages[1]; //restul de pagini ramase sunt adaugate in sectiunea a treia
            // Determinarea numrului minim de pagini
            if(pages[0] < pages[1])
            {
                if(pages[0] < pages[2])
                {
                    min_pages = pages[0];
                }
                else
                {
```

```
                    min_pages = pages[2];
                }
            }
            else
            {
                if(pages[1] < pages[2])
                {
                    min_pages = pages[1];
                }
                else
                {
                    min_pages = pages[2];
                }
            }

            // Determinarea numrului maxim de pagini
            if(pages[0] > pages[1])
            {
                if(pages[0] > pages[2])
                {
                    max_pages = pages[0];
                }
                else
                {
                    max_pages = pages[2];
                }
            }
            else
            {
                if(pages[1] > pages[2])
                {
                    max_pages = pages[1];
                }
                else
                {
                    max_pages = pages[2];
                }
            }


            diff = max_pages - min_pages; //aflarea diferentei dintre numarul maxim de pagini si cel minim
            if(diff < min_diff) //aflarea diferentei minime globale
            {
                min_diff = diff; //actualizarea diferentei minime globale
                first_break = i; //primul marcator de pozitie, dintre sectiunea 0 si sectiunea 1
                second_break = j; //al doilea marcator de pozitie, dintre sectiunea 2 si sectiunea 3
            }
        }
    }

    //resetarea sectiunilor, in vederea recalcularii lor in functie de marcatorii de pozitie
    pages[0] = pages[1] = pages[2] = 0;

    for(int i = 0; i < num_books; i++) //parcurgerea cartilor
    {
        if(i <= first_break)
            pages[0] += books[i].pages; //aflarea primei sectiuni
        else if(i <= second_break)
            pages[1] += books[i].pages;
        //aflarea celei de-a doua sectiuni
        else
            pages[2] += books[i].pages; //aflarea celei de-a treia sectiuni
    }
    fprintf(file, "===Rezultatele in cazul unei metode de cautare exhaustive===\n");
    fprintf(file, "Angajatul 1 este responsabil de %lld pagini.\n", pages[0]);
    fprintf(file, "Angajatul 2 este responsabil de %lld pagini.\n", pages[1]);
```

```
    fprintf(file, "Angajatul 3 este responsabil de %lld pagini.\n", pages[2]);
}
```

5.2 Program Python

```
def sections(books, num_books, file): #functia care imparte cartile intre angajati folosind o metoda greedy
    total_pages = no_of_pages(books) #se calculeaza numarul total de pagini, prin apelarea functiei anterioare
    third = total_pages // 3 #se afla o treime din numarul total de pagini
    pages = [0, 0, 0] #se initializeaza numarul de pagini ce-i revin fiecarui angajat
    section = 0 #initializam sectiunea curenta cu 0

    for book in books: #parcurgem lista de carti
        if pages[section] + book.pages <= third or section == 2: #adaugam paginile urmatoarei carti daca suma nu depaseste o treime din total sau suntem la ultima sectiune
            pages[section] += book.pages
        else: #altfel
            section += 1 #trecem la noua sectiune
            pages[section] += book.pages #adaugam paginile la noua sectoiune

    file.write("===Rezultatele in cazul unei metode greedy===\n") #se afiseaza rezultatele
    file.write(f"Angajatul 1 este responsabil de {pages[0]} pagini.\n")
    file.write(f"Angajatul 2 este responsabil de {pages[1]} pagini.\n")
    file.write(f"Angajatul 3 este responsabil de {pages[2]} pagini.\n")
```

```
def sections2(books, num_books, file): #functie care imparte cartile prin o metoda exhaustiva
    total_pages = no_of_pages(books) #se afla numarul total de pagini
    pages = [0, 0, 0] #initializam nr de pagini pentru fiecare angajat cu 0
    min_diff = total_pages #initializam diferenta minima cu numarul total de pagini
    first_break = 0 #initializam prima pauza din lista cu 0
    second_break = 0 #initializam a doua pauza din lista cu 0

    for i in range(num_books - 2): #parcurgerea listei de carti pana la penultima carte
        pages[0] += books[i].pages #adaugam paginile cartii curente in sectiunea primului angajat
        pages[1] = 0 #resetam numarul de pagini pentru al doilea angajat
        for j in range(i + 1, num_books - 1): #parcurgem lista de carti de la cartea curenta pana la ultima carte
            pages[1] += books[j].pages #adaugam paginile cartii curente la al doilea angajat
            pages[2] = total_pages - pages[0] - pages[1] #calculam numarul de pagini pentru al treilea angajat ca fiind restul de pagini ramase
            min_pages = min(pages) #calulam numarul minim de pagini
            max_pages = max(pages) #calculam numarul maxim de pagini
            diff = max_pages - min_pages #diferenta dintre numarul minim si numarul maxim de pagini

            if diff < min_diff: #daca diferenta obtinuta e mai mica decat diferenta minima
                min_diff = diff #actualizam diferenta minima
                first_break = i #actualizam prima pauza
                second_break = j #actualizam a doua pagina

    pages[0] = pages[1] = pages[2] = 0 #resetam numarul de pagini pentru fiecare angajat

    for i in range(num_books): #parcurgem lista de carti
        if i <= first_break: #daca suntem inainte de prima pagina, adaugam paginile la primul angajat
            pages[0] += books[i].pages
        elif i <= second_break: #daca suntem inainte de a doua pauza, adaugam paginile la al doilea angajat
            pages[1] += books[i].pages
        else: #altfel, adaugam paginile la al treilea angajat
            pages[2] += books[i].pages

```

```
33      file.write("===Rezultatele in cazul unei metode de cautare exhaustive===\n") #se afiseaza
            rezultatele
34      file.write(f"Angajatul 1 este responsabil de {pages[0]} pagini.\n")
35      file.write(f"Angajatul 2 este responsabil de {pages[1]} pagini.\n")
36      file.write(f"Angajatul 3 este responsabil de {pages[2]} pagini.\n")
```

References

1. Liste în Python
2. Funcții în Python
3. Realizarea de instrucțiuni buclă în Python
4. **Implementarea C și implementarea Python**