UNIVERSITATEA DIN CRAIOVA
FACULTATEA DE
AUTOMATICĂ, CALCULATOARE ŞI ELECTRONICĂ

PACHET SOFTWARE PENTRU MANAGEMENTUL UNEI LIBRĂRII

Autor : Zane M. Livia, Calculatoare romana, Anul 1, Grupa CR1.3B

1 DEFINIREA PROBLEMEI

Ca tema de casa in cadrul laboratorului la disciplina *Programarea Algoritmilor*, a fost alocat proiectul cu titlul *pachet software pentru managementul unei librarii.*

Conform cerintelor proiectului, libraria contine un set de carti. Fiecare carte are un titlu unic si un set de unu sau mai multi autori. Un autor poate fi coautor al mai multor carti. Un autor are o pereche unica continand numele si prenumele.

De asemenea, se precizeaza ca este important ca reprezentarea setului de carti al librariei nu trebuie sa contina informatii redundante.In acest sens, trebuie ca seturile de carti si de autori sa fie reprezentate prin liste simple inlantuite, astfel incat, fiecare autor si fiecare carte sa fie unic reprezentate in sistemul de management al librariei.

Trebuie definiti algoritmi pentru urmatoarele operatii:

i) adauga o carte la librarie;

ii) determina lista cartilor pentru care, un autor dat, este coautor;

iii) determina lista autorilor care au realizat o carte data.

Tot odata, s-a solicitat ca implementarea cerintelor sa fie realizata in programul C standard C99, ale carui elemente fundamentale au fost predate in cursul anului. Codul trebuie sa fie portabil, sens in care s-a solicitat sa fie compilat cu doua compilatoare diferite. Stilul de codare trebuie sa fie cel impus pentru acest curs (C coding style), iar fiecare functie, variabila si blocuri de program, sa fie comentate. Codul trebuie sa fie modular (minimum doua fisiere cu extensia .c si minim unu cu .h).

Un alt aspect solicitat este implementarea unei metode de testare, prin generearea automata a unor seturi de date non-triviale, vectori de lungime $10^3 - 10^8$, rezultatul experimentelor fiind prezentate grafic. De asemenea, proiectul trebuie insotit de o documentatie realizata in $LaTeX$.

Optional, pentru puncte suplimentare, codul poate fi scris si in Phyton2.7.x or Python 3.6.x., cu reguli similare privind codul (modulat, comentat, identat stil PEP 8, etc.).

2 ALGORITMI

Algoritmii implementati pentru realizarea cerintelor sunt urmatorii:

ADD-BOOK($head - book, authors[10], new - book - title$)

```
1  new − book = head − book                ▷ new-book is a new node of head-book type
2  new − book.book − title = new − book − title
3  for i = 1 to 10                         ▷ each book node has maximum 10 authors
4      new − book.authors[i] = authors[i]
5  new − book.next = head − book           ▷ head-book node became the second node
6  head − book = new − book                ▷ new-book node became the first node of the list
```

Algoritmul folosit este cel de inserare a unui nod la inceputul nei liste simplu inlantuite - $push - begining$, dupa modelul prezentat la cursul[1] si laboratorul de Proiectarea Algoritmilor.

Notand c1..c7 costurile pentru fiecare linie din algoritm calculam timpul de executie este dat de suma c1 + c2 + c3 + c4 + c5 + c6 + c7, deci timpul de executie este T(1). Cazurile defavorafil, favorabil si mediu sunt egale, deci $T_w = T_b = T_a$. In mod similar, in notatiile Θ, Ω si O avem egalitate respectiv, $O(1) = \Theta(1) = \Omega(1)$.

Calculul memoriei se face dupa tipul variabilelor, respectiv numarul de biti pe care este reprezentata inmultit cu numarul acestora.

Masurarea dimensiunii de intrare este numarul de biti necesari pentru a codifica intrarea inmultiti cu numarul de elemente. In cazul nostru pentru codificarea unui element sunt necesari 1.15kb avand in vedere ca: book-title - char[100] (8b*100), authors - long [10] (32b*10) si next-book -*(32b).

ADD-AUTHOR($head-author, new-id-author, new-author-name, new-author-given-name$)

1 $new - author = head - author$ ▷ new-author is a new node of head-author type
2 $new - author.author - name = new - author - name$
3 $new - author.author - given - name = new - author - given - name$
4 $new - author.id - author = new - id - author$
5 $new - author.next = head - author$ ▷ head-author node became the second node
6 $head - author = new - author$ ▷ new-author node became the first node of the list

Algoritmul folosit este cel de inserare a unui nod la inceputul nei liste simplu inlantuite - $push - begining$.

Notand c1..c7 costurile pentru fiecare linie din algoritm calculam timpul de executie este dat de suma c1 + c2 + c3 + c4 + c5 + c6, deci timpul de executie este T(1). Cazurile defavorafil, favorabil si mediu sunt egale, deci $T_w = T_b = T_a$. In mod similar, in notatiile Θ, Ω si O avem egalitate respectiv, $O(1) = \Theta(1) = \Omega(1)$.

Calculul memoriei se face dupa tipul variabilelor, respectiv numarul de biti pe care este reprezentata inmultit cu numarul acestora.

Masurarea dimensiunii de intrare este numarul de biti necesari pentru a codifica intrarea inmultiti cu numarul de elemente. In cazul nostru pentru codificarea unui element sunt necesari 384b avand in vedere ca: id-author long(32b), author-name - char[20] (8b*20), author-given-name - char[20] (8b*20), next-author - *(32b).

SEARCH-BOOK-AUTHORS($head - book, head - author, book - title$)

```
1  iterator − book = head − book
2  iterator − author = head − author
3  while iterator − book ≠ NIL
4      if iterator − book.book − title = book − title
5          for i = 1 to 10          ▷ each book node has maximum 10 authors, 0 for no authors
6              if iterator − author.id − author ≠ 0
7                  while iterator − author ≠ NIL
8                      if iterator − book.authors[i] = iterator − book.authors[i]
9                          return iterator − book.book − title
10                 iterator − author = iterator − author.next
11         iterator − author = head − author
12     break
13     iterator − book = iterator − book.next
```

Primul while parcurge lista de carti cu un timp de executie $T(n) = n$, iar al doi-lea while are acelasi timp de executie. Timpul de executie al algoritmului este $T(n) = c1 + c2 + n + c3 + 10 * (c4 + c5 + n + c6 + c7) + c8 = n + 10n + 44c$ (considerand costurile fixe ca fiind egale), deci in notatie asimptotica avem $O(n)$.

Cazul cel mai favorabil este atunci cand cartea cautata este prima in lista de carti, iar autorii sunt primii in lista de autorii, deci $\Omega(1)$.

Cazul mediu este atunci cand nu se parcurge lista de carti pana la final, cartea fiind gasita mai devreme si se iese din prima bucla while. Similar, al doi-lea while parcurge lista pana gaseste id_autor, apoi iese din bucla. Timpul de executie al algoritmului este $T(n) = \log n * 10 * \log n$, deci avem $\Theta(\log n)$.

SEARCH-AUTHOR-BOOKS($head - book, head - author, book - title$)

```
1  iterator − book = head − book
2  iterator − author = head − author
3  while iterator − author ≠ NIL
4      if iterator − author.author − name = author − name and
5      iterator − author.author − given − name = author − given − name
6          while iterator − author ≠ NIL
7              for i = 1 to 10
8                  if iterator − author.id − author = iterator − book.authors[i]
9                      return iterator − book.book − title
10             iterator − book = iterator − book.next
11     break
12     iterator − author = iterator − author.next
```

Pentru acest algoritm, cand costurile sunt considerate egale, timpul de executie este $T(n) = c + c + n * (c + c + n(10c + c)) + c$, deci in notatie asimptotica avem $O(n)$.

Cazul cel mai favorabil este atunci cand autorul cautat este primul in lista de autori, lista de carti fiind totusi parcursa pana final, deci $\Omega(n)$.

Cazul mediu este atunci cand nu se parcurge lista de autori pana la final, autorul fiind gasit mai devreme si se iese din prima bucla *while*, insa al doi-lea *while* parcurge lista de carti pana la capat. Timpul de executie al algoritmului este $T(n) = \log n + n$, deci avem $\Theta(n)$.

Pentru implementarea algoritmilor au fost folosite urmatoarele functii:

*void add-book (struct book-node **head-book, long authors[], char *new-book-title)* - realizeaza inserarea unui nod in lista simplu inlantuita pentru stocarea cartilor;

*void add-author (struct author-node **head-author, long new-id-author, char *new-author-name, char *new-author-given-name)* - realizeaza inserarea unui nod in lista simplu inlantuita pentru stocarea autorilor;

*void search-book-authors(struct book-node *head-book, struct author-node *head-author, char *title)* - realizeaza cautarea unei carti date si afisarea autorilor acesteia;

*void search-author-books(struct book-node *head-book, struct author-node *head-author, char *author-name, char *author-given-name)* - realizeaza cautarea unui autor dat in lista de autori iar dupa id-ul acestuia afiseaza cartile la carea acesta este autor;

*long find-book (struct book-node *head-book, char *title)* - realizeaza cautarea unui titlu de carte dat in lista de carti si returneaza 1 daca gaseste sau 0 daca nu gaseste;

*long find-author (struct author-node *head-author, char *searched-author-name, char *searched-author-given-name)* - realizeaza cautarea dupa nume si prenume autor dat, in lista de autori si returneaza id autor daca gaseste sau 0 in caz contrar;

*void list-library (struct book-node *current-book)* - afiseaza toate titlurile cartilor din lista de carti si vectorul de 10 autori (cand id-author $\neq 0$) pentru fiecare carte;

*void list-authors(struct author-node *current-author)* - afiseaza numele, prenumele si id-ul tuturor autorilor din lista simplu inlantuita corespondenta;

De asemenea, au mai fost implementate functii pentru salvarea celor 2 liste simplu inlantuite in fisiere de tip text, respectiv citirea din fisiere a datelor si popularea listelor.

Pentru partea de testare, functiile pentru cei doi algoritmi de cautare au fost folosite cu modificarea de a nu afisa titlurile cartilor si numele, prenumele autorilor, in loc de *printf* fiind pusa instructiunea *continue*. De asemenea, a mai fost realizata o functie de generare aleatorie a unor siruri de caractere care va fi detaliata in capitolul urmator.

In fisierele de cod, conform cerintelor, au fost comentate toate variabilele importante, toate functiile si blocurile de program. Comentariile, pentru usurinta in depanarea/modificarea codului, sunt in limba engleza.

3 DATE DE TEST

Pentru obtinerea unui set de date aleatorii, a fost folosita functia *rand()*. Avand in vedere ca aceasta genereaza numere pseudo-aleatorii, pentru a evita generarea aceluiasi set de numere la fiecare rulare a programului, am folosit si functia *srand()*, care, la fiecare apel, seteaza un alt punct de plecare pentru *rand()*. Pentru generarea secventelor alfanumerice aleatorii, am folosit[2] functia ***random-string()***, de tip pointer la caracter, al carui cod si mod de functionare sunt prezentate

in cele ce urmeaza:

```
char *random_string(int size){
    #define ASCII_START 32
    #define ASCII_END 126

    char *rand_string = malloc(size);

    for(int i = 0; i < size; i++) {
        rand_string[i] = (char) (rand()%(ASCII_END-ASCII_START))+ASCII_START;
    }

    return rand_string;
}
```

Analizand tabela ASCII, se obesrva ca sunt utile pentru algoritmul de testare, caracterele de la codul 32 la 126. Astfel, functia *rand()* va genera numere de la 0 la 94 (126-32),la care se adaga 32, apoi sunt transformate (cast) in date de tip caracter. Lungimea *size* a secventelor alfanumerice generate aleatoriu, este transmisa din programul principal la apelarea functiei, potrivit destinatiei (100 pentru titlul cartilor si 40 pentru nume/prenume). Tot in programul principal este apelata si functia *srand()* la fiecare rulare a programului.

In vederea formarii setului de date utilizat pentru testare, programul genereaza cate 10^4 siruri alfanumerice aleatorii, de lungimi cuprinse intre minim trei si urmatoarele lungimi maxime:

i) 100 caractere pentru titlul cartilor;

ii) 40 caractere pentru numele autorului;

iii) 40 caractere pentru prenumele numele autorului.

Datele generate aleatoriu, reprezentand nume de carti, nume si prenume, sunt inserate in cele doua liste simplu inlantuite, utilizand codul urmator:

```
for (int i = 1;  i <= 10000; i++){
rand_book_title = random_string(40);
for (int j = 0; j < 10; j++){
        authors[j] = i - j;
    }
    find = find_book (head_book, authors, rand_book_title);
    if (find == 0){
        add_book (&head_book, authors, rand_book_title);
        } else {
            printf ("Carte existenta\n");
        }
}
end = clock();
time_alghoritm = (double)(end - start)/CLOCKS_PER_SEC;
printf("%f", time_alghoritm);
```

Codul insereaza in lista simplu inlantuita corespondenta 10^4 titluri de carti generate aleatoriu.Pentru fiecare titlu, se adauga si cate 10 autori generati anterior dupa un algoritm similar. Identificatorul (*id_author*) pentru cei 10 autori ai fiecarei carti se aloca automat dintre id-urile generate la incarcarea listei de autori.

Totodata, prin intermediul functiei *clock()* din biblioteca standard `<time.h>`, inainte de adaugarea in lista, se retine timpul procesorului (CPU time) in variabila *start*, iar la terminarea celor 10^4 inregistrari, timpul procesorului se retine in variabila *end*, pentru determinarea, in secunde, a timpului de executie a algoritmului. Timpii inregistrati pe mai multe seturi de date (rulari repetate ale programului), vor fi prezentati in capitolul urmator.

Dupa ce sunt inserate titlurile de carti, lista corepondenta este traversata, iar pentru fiecare carte, autorii se cauta in lista de autori dupa id_author. Codul acestui algoritm este urmatorul:

```
struct book_node *iterator_book = malloc (sizeof (*iterator_book));
if (iterator_book == NULL) {
     printf("Memory not allocated!\n");
     exit(0);
}
iterator_book = head_book;
struct author_node *iterator_author = malloc (sizeof (*iterator_author));
if (iterator_author == NULL) {
printf("Memory not allocated!\n");
     exit(0);
}
iterator_author = head_author;
start = clock();
while (iterator_book != NULL){
     search_book_authors_nodisplay(head_book, head_author, iterator_book->book_t
     iterator_book = iterator_book->next;
}
end = clock();
time_alghoritm = (double)(end - start)/CLOCKS_PER_SEC;
printf("%f", time_alghoritm);
```

Dupa definirea unui nod *iterator* pentru lista *carti* si *autori*, respectiv alocarea de memorie catre acesti vectori catre o structura, este traversata lista de carti, iar pentru fiecare din cele 10^4 titluri, se cauta autorii fiecarei carti in lista de autori prin intermediul functiei *search_book_authors_nodisplay*.Fun este o copie a celei de cautare a autorilor unei carti date, in care a fost comentata comanda *printf()* de afisare a autorilor, iar in locul ei a fost pusa instructiunea *continue* pentru a nu scurta artificial timpul de executie al algoritmului. Am facut acest artificiu deoarece operatiile de afisare au un cost mare de executie, cu repercusiuni negative asupra timpului real de executie al algoritmului. Astfel, prin intermediul codului de test sunt *"listate"* toate cele 10^4 titluri de carti, fiecare cu cate 10 autori.

De retinut este faptul ca aceste operatii sunt executate cu datele generate aleatoriu din plaja tabelei ASCII de la codul 32 la 126, sirurile avand lungimi cuprinse intre 3 si 100 caractere (titlurile cartilor) si 40 caractere (numele si prenumele). Am ales minimul de trei caractere, deoarece la introducerea de la tastatura a numelui/prenumelui autorului, programul paraseste sesiunea de

lucru daca se introduce un nume cu lungimea de un caracter. Prin aceasta varianta se asigura usurinta in introducerea datelor de catre operator, astfel incat, daca o carte are un singur autor, la numele celui de-al doilea, daca se introduce un singur caracter (*lenght = 1*), programul nu mai solicita datele pentru alti 9 autori.

De precizat ca la lansarea programului exista optiunea de rulare a codului de testare, astfel incat pot fi verificati toti algoritmii prezentati mai sus, inclusiv cei de testare.

Corectitudinea codului rezulta si din faptul ca acesta a fost compilat, fara **erori** sau **atentionari**, in aplicatiile Code::Blocks/Windows si GCC/Linux.

4 REZULTATE SI CONCLUZII

4.1 Rezultate

Rezultatele obtiunute prin rularea, de cate 10 ori, a cate unui set de date de test de 10.000 carti (cu cate 10 autori fiecare) si 10.000 autori, generate aleatoriu, sunt prezentate in graficele din fig.1, fif.2 si fig.3, dupa cum urmeaza:

a) fig.1, timpii procesor la rulare cod C pe un calculator cu procesor Pentium 5 care ruleaza Windows 10, comparativ cu alt calculator cu Windows 10 si Pentium 3, respectiv pe Linux cu procesor Pentrium 3;

b) fig.2, codul Python pe aceleasi trei calculatoare;

c) fig.3, comparativ codurile C si Python pe un calculator cu procesor P5 si sitem de operare Windows 10

Dupa cum se observa, timpii procesor ai celor trei algoritmi nu difera in functie de calculator (cel putin intre P3 si P5), insa difera in functie de sistemul de operare.

Graficele au fost generate cu aplicatia GNU-Octave (similar Matlab, care este licentiat).

4.2 Concluzii

Cea mai complicata parte a programului a fost, pentru mine, citirea din fisiere a listelor salvate anterior in respectivele fisiere. Ar fi trebuit sa folosesc separatori pentru variabile la scriere si functia *strtok* dupa citirea linie cu linie, pentru a memora datele in variabile, insa imi propun ca pe viitor sa aprofundez scrierea/citirea in fisiere a variabilelor de tip diferit. Pentru ca am vrut sa lucrez si cu functiile pentru fisiere, am renuntat in codul C la implementarea listelor simplu inlantuite (autorii unei carti) in lista simplu inlantuita a cartilor, insa in varianta Python campul *authors* este de tip lista, cauza pentru care si timpul de executie al algoritmului de cautare a cartilor pentru un autor dat.

De asemenea am intampinat greutati in lucrul cu aplicatia L^AT_EX, cu care am lucrat prima oara acum, insa imi propun ca pe viitor sa imi aprofundez cunoastintele in ceea ce priveste acest program. Tot legat de aceasta aplicatie, am constatat ca nu pune figurile in locatia in care au fost

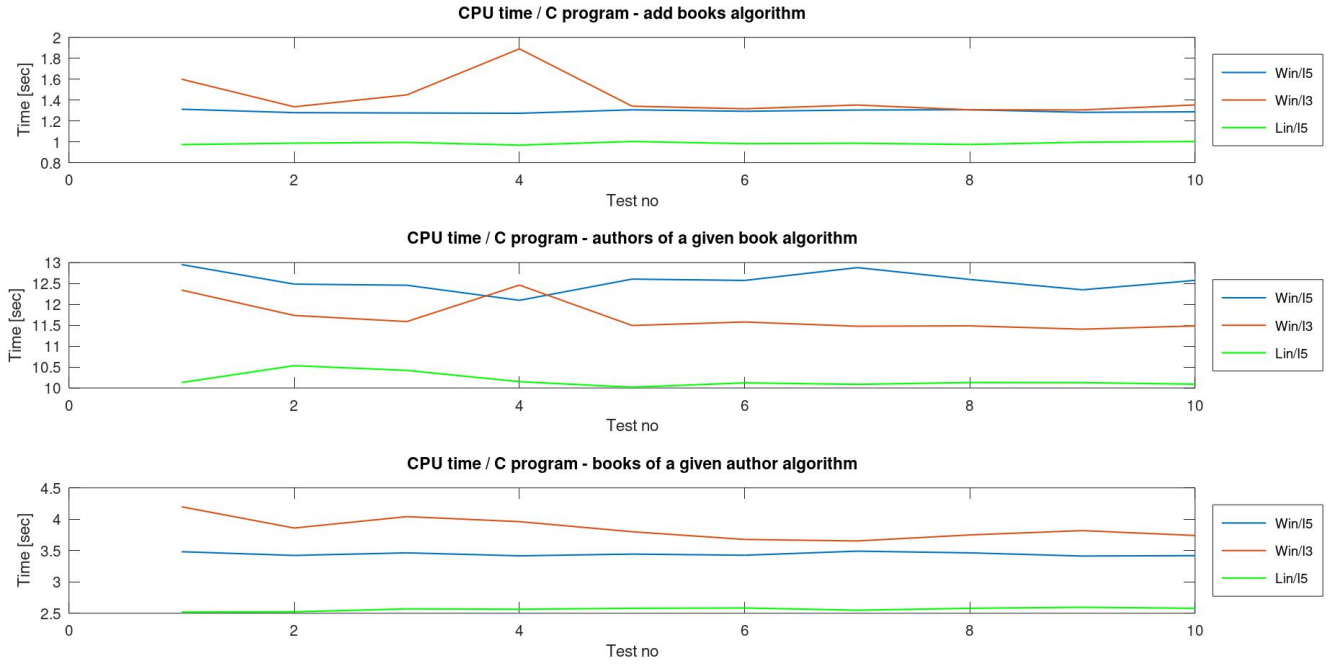

Figure 1: Timpii masurati pentru cod C

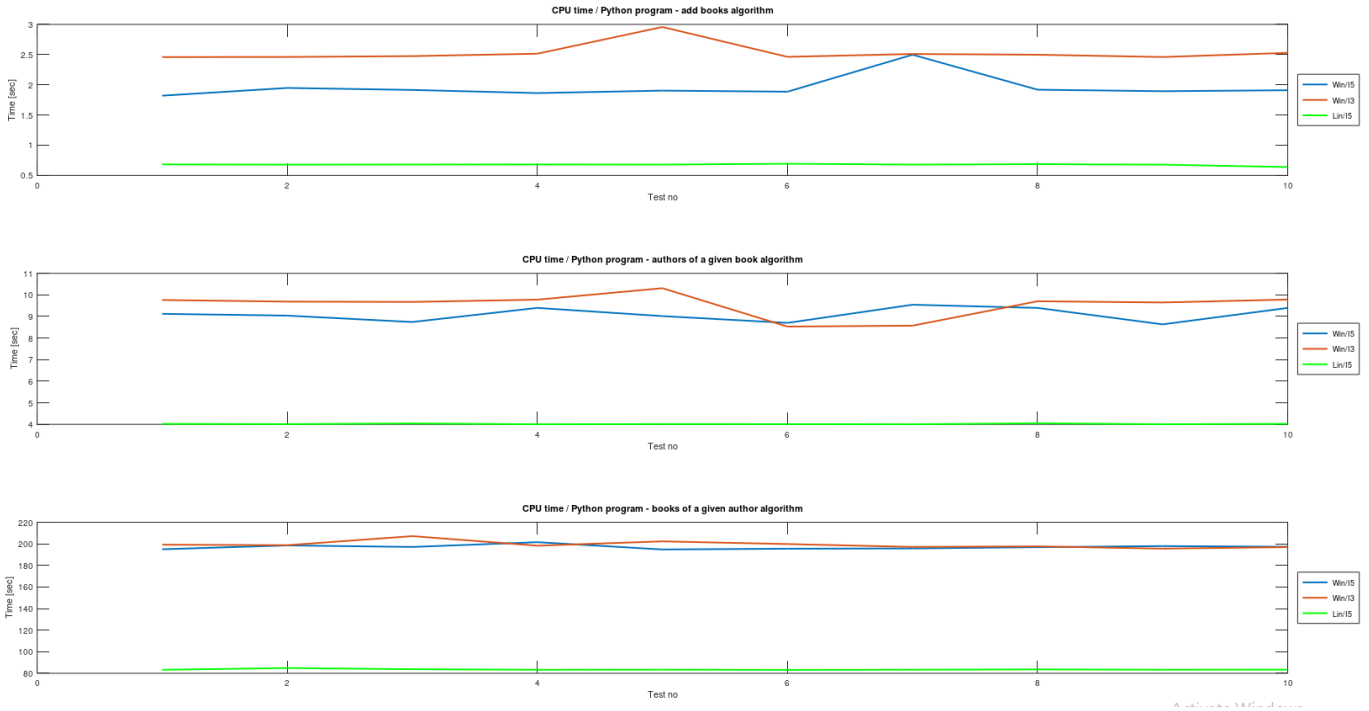

Figure 2: Timpii masurati pentru cod Python

inserate in sursa, ci le cauta un loc convenabil pentru aplicatie. O alta problema intampinata a fost aceea ca nu toate imaginile/graficele generate de Octave au fost acceptate de aplicatia L^AT_EX, fiind nevoita sa incerc diferite tipuri de fisiere (png, jpg, pdf, eps) pana am reusit sa inserarea lor.

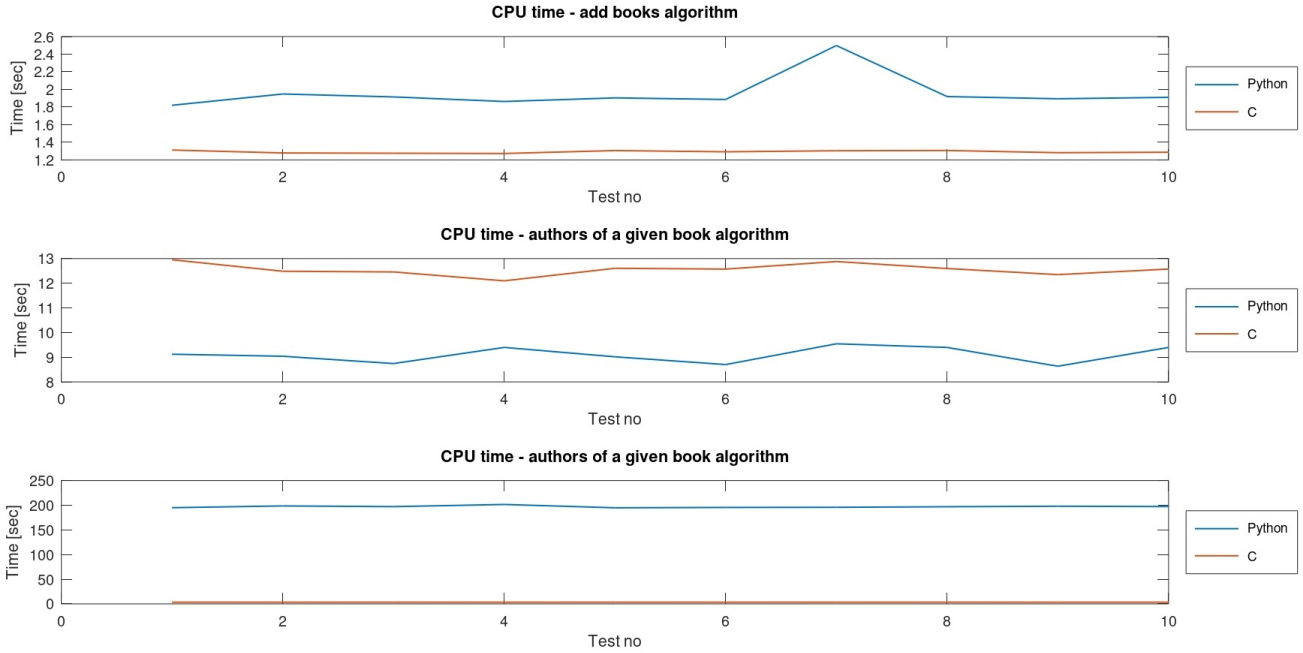

Figure 3: Timpii masurati pentru cod Python comparativ cu cod C

Sursa prezentului document, poate fi accesata aici.

Bibliografie

[1] Badica Costin. *Note de curs*. Google Classroom, 2021.

[2] https://codereview.stackexchange.com/questions/29198/random-string-generator-in c.