Universitatea Politehnică Bucureşti

Facultatea de Automatică şi Calculatoare

Analiza Algoritimilor

-TEMĂ-

Student:
Nicuţă Loredana - Ionela 315 CD Anul II

2017-12-14

Cuprins

1 Tema aleasă

Tema aleasă este reprezentată de subiectul numărul 7: Analiza operaţiilor asociate unor structuri de date avansate

2 Structurile de date alese

2.1 AVL Tree

Un AVL Tree, este un arbore binar de căutare echilibrat, în care diferenţa înălţimilor dintre subarborele stâng şi cel drept nu poate fi mai mare decât 1. [1]

Complexitate timp: O(log n), unde log n reprezntă înălţimea maximă a arborelui.

Complexitate spaţială : O (n), unde n reprezintă numărul de noduri necesare construirii arborelui.

2.1.1 Avantajele folosirii unui AVL Tree

1. **Faţă de un arbore binar ne-echilibrat, la care complexitatea în cel mai rău caz va fi este O(n), unde n este numărul de noduri şi reprezintă înălţimea maximă, arborele AVL are complexitatea O(log n), având mereu înălţimea log n.**

2. **Un arbore AVL oferă posibilitatea de a căuta cu uşurinţă un element.**

2.1.2 Dezavantajele folosirii unui AVL Tree

1. **Codul pentru implementarea unui astfel de arbore este mult mai complex decât pentru cel al unui arbore binar ne-echilibrat, fiind nevoie să se trateze diferite excepţii.**

2. **Înălţimea arborelui fiind log n, aceasta trebuie menţinută, astfel sunt realizate destul de multe rotaţii, acest lucru făcând alocarea unui spaţiu în plus la input-uri mari.**

3. **Ştergerea unui element are costul mai mare. Complexitatea este în continuare O(log n) însă s-ar putea să fie necesare rotaţii ce se vor extinde până la rădăcina arborelui.**

2.1.3 Utilizări practice ale structurii AVL Tree

Se va folosi acest tip de structură în situaţiile în care avem deja un set de date iniţial apropae sortat, şi vor fi adăugate destul de puţine date pe parcurs sau şterse, astfel nefiind în cazul în care trebuie să se realizeze mai multe rotaţii (în acest caz, va fi utilizat mai bine un alt arbore binar şi anume arborele Roşu şi Negru). Astfel, vom putea folosi arborele AVL ţn implementarea unui sistem ce reţine rutele unui transport în comun, de exemplu trenul, nefiind modificate multe rute/adăugate.

2.2 Segment Tree

Un Segment Tree este un arbore binar folosit pentru a stoca intervale sau segmente. Fiecare nod este format din suma elementelor din subarborele stâng şi cele din subarborele drept, iar frunzele sunt elementele vectorului. [2]

Complexitate timp: O(log n), unde log n reprezntă înălţimea maximă a arborelui.

Complexitate spaţială : O (4 * n), unde 4 * n reprezintă numărul de noduri necesare construirii arborelui (numărul de noduri dintr-un arbore binar este de maxim 2 pow([(log n) + 1]) , unde log n + 1 reprezintă înălţimea) .

2.2.1 Avantajele utilizării unui Segment Tree

1. **Este o structură flexibilă ce permite diferite operaţii asupra elementelor dintr-un vector. (găsirea elementului minim, găsirea elementului maxim, calcularea sumei, calcularea celui mai mic divisor comun ...).**

2. **Operaţiile de adăugare, ştergere, căutare a unui element au complaxitatea O(log n).**

2.2.2 Dezvantajele utilizării unui Segment Tree

1. **Folosirea de memorie destul de multă comporativ cu alte implementări ale arborilor binari. În tot arborele, vom avea numai n elemente din vector, restul reprezentând suma dintre acestea. Astfel complexitatea spaţială va fi de O (4 * n), unde n reprezintă numărul de noduri din vectorul inţial.**

2. **Dintr-un Segment Tree nu se pot şterge propriu-zis elemente, ci vor fi marcate cu un număr negativ. O altă soluţie este crearea unui nou Segment Tree cu noul vector din care a fost eliminat elementul dorit. Această metoda este extrem de ineficientă.**

2.2.3 Utilizări practice ale structurii Segment Tree

Se va folosi acest tip de structură în situaţiile în care vom avea un set de date iniţial stabilit, asupra căruia nu vom mai dori sa adăugăm noi elemnte, ci doar să le modificăm valoarea eventual. Fiecare adăugare/ştergere reprezintă realizarea unui nou arbore de acest tip, existând alte structuri mai eficiente în acest caz. Astfel, vom putea utiliza această structură atunci când vom dori să realizăm pe un set de date, diferite operaţii dintr-un anumit interval. (Ex. Calcularea minimului subsirului dat de x si y, Calcularea sumei primelor x elemente dintr-un vector ...).

3 Analiza operaţiilor specifice

În realizarea acestor teste, am implementat structurile şi operaţiile necesare în Java. Pentru AVL, codul a fost luat de aici : [cod], iar pentru Segment de aici : [cod]. Pentru testare şi adaptare, au fost selectate metodele, respectiv adăugate.

3.1 Adăugarea elementelor

3.1.1 Adăugarea unui element într-un AVL Tree

Arborele AVL este un arbore echilibrat (avem diferenta înălţimilor subarborelui stang, respectiv celui drept de cel mult 1). Astfel, atunci când un element este inserat într-un astfel de arbore, se va realiza mereu la final re-echilibrarea arborelui, complaxitatea operaţiei de adăugare fiind mereu O(log n), unde log n reprezintă înălţimea arborelui AVL.

Re-echilibrarea unui arbore constă în realizarea unor rotaţii simple sau duble, urmate de recalcularea înălţimii fiecărui nod întâlnit parcurgând arborele de jos în sus, spre rădăcină.

Paşii ce vor fi urmăriţi la inserarea unui elemnt într-un arbore AVL:

1. **Se inserează elementul ca într-un arbore binar ordonat obişnuit: se porneşte de la rădăcină şi se urmează fiul stâng sau fiul drept, în funcţie de relaţia dintre cheia de inserat şi cheia nodurilor prin care se trece.**

2. **După ce elementul a fost introdus, se se parcurge drumul invers (unic) şi se caută pe acest drum primul nod care nu este echilibrat, adicua primul nod ai cărui subarbori diferă ca înălţime prin 2 unitaţi. Fiecare element adăugat ce necesită rotaţii se va afla în unul dintre cazurile ilustrate în figură. ([3]).**

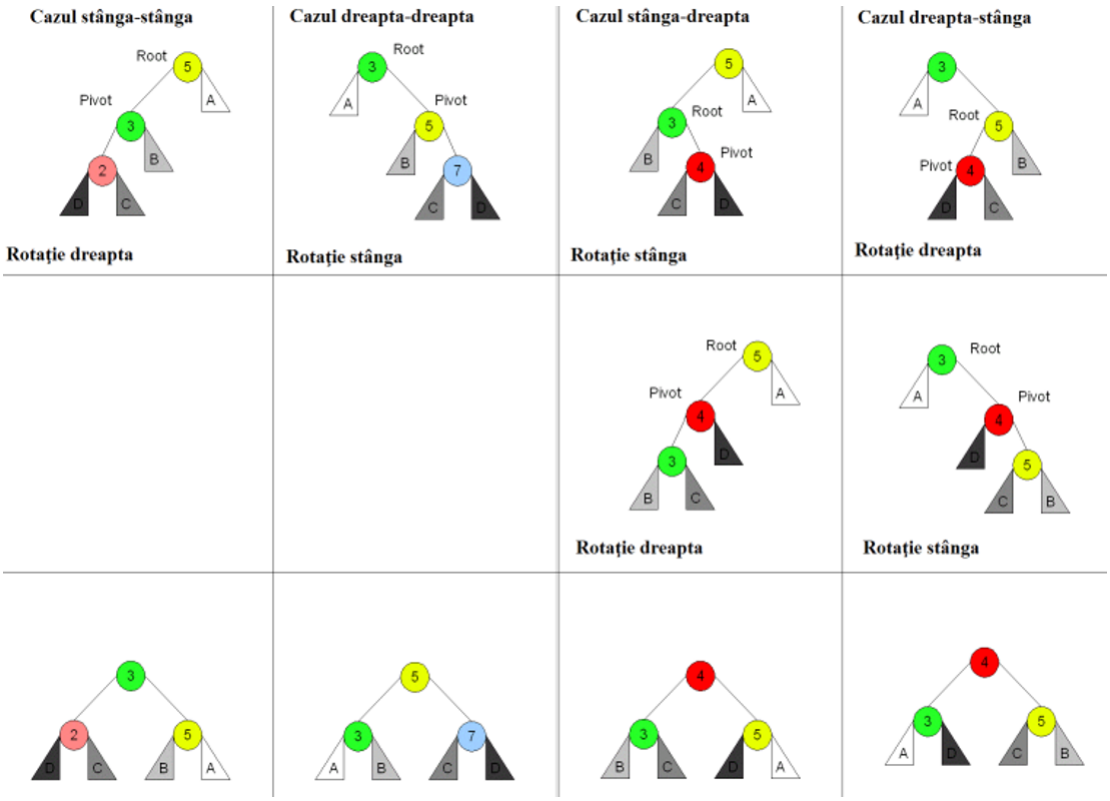

Figure 1: Cazurile de rotaţie într-un arbore AVL

3.1.2 Adăugarea unui element într-un Segment Tree

Arborele Segment este o structură de date folosită mai ales pentru a stoca informaţii despre intervale sau segmente. Această structură nu se poate modifica odata ce a fost creată, complaxitatea operaţii de adăugare fiind mereu O(log n), unde log n reprezintă înălţimea arborelui AVL.

Paşii ce vor fi urmăriţi la inserarea unui elemnt într-un arbore Segment:

1. **Vom adăuga mai întâi toate elementele dorite într-un vector de dimensiune n. Acesta va repreznta baza construcţiei arborelui Segment.**

2. **Pentru construirea arborelui, vom începe cu vectorul [0 ... n - 1] şi de fiecare dată când împărţim vectorul în jumătate, vom**

salva în nodul curent ceea ce ne interesează (suma, minim, maxim, etc.) şi continuăm până vectorul va fi de lungime 1. Astfel, vom ajunge să avem drept frunze toate elementele vectorului iniţial. [4]

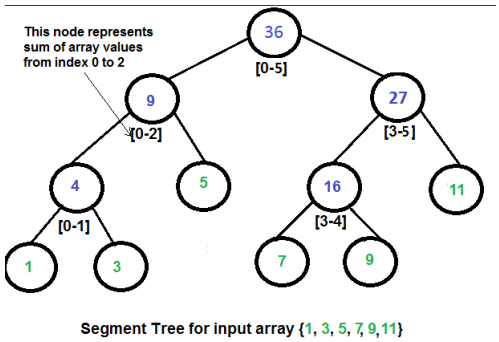

Figure 2: Exemplu de inserare într-un arbore Segment

3.1.3 Testarea implementărilor pe diferite teste

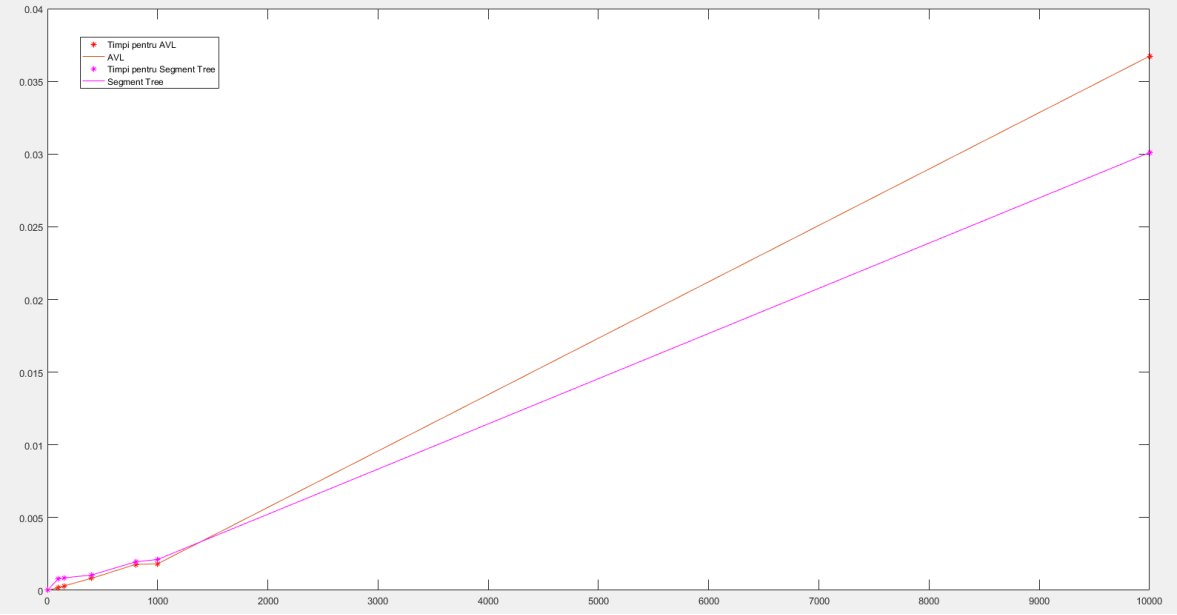

Figure 3: Evoluţia temporală la inserarea elementelor

Se observă şi de pe grafic faptul că arborele AVL, faţă de arborele Segment este o bună implemntare atunci când avem teste de dimensiuni relativ mici (100 - 1000) de elemente, însă cu cât datele sunt mai multe, cu atât durează mai mult fiind nevoie de multe re-echilibrari.

3.2 Ştergerea elementelor

3.2.1 Ştergerea elementelor dintr-un AVL Tree

Atunci când se doreşte ştrgerea unui element dintr-un AVL Tree, va trebui mai întâi să se caute acel element. În cazul în care acesta a fost găsit se va rupe legătura cu părintele elementului şi se va realiza o nouă legătură cu potenţialii copii al elementului eliminat.

După ce se şterge fizic elementul, vom realiza re-echilibrarea arborelui, în acelaşi mod în care am realizat-o la adăugarea unui element. Această operaţie are complexitatea O(log n), deoarece în cel mai rău caz, vom avea de parcurs arborele până la înălţimea maximă pentru a găsi elementul ce se doreşte a fi şters.[5]

3.2.2 Ştergerea elementelor dintr-un Segment Tree

Atunci când se doreşte ştergerea unui element dintr-un Segment Tree, există două posibilităţi de abordare asupra acestei probleme:

1. **Se caută elementul în vectorul iniţial, se şterge, iar mai apoi se realizează un NOU arbore de tip Segment, ce nu va conţine elementul şters.**
2. **Se caută în arbore elementul ce se doreşte a fi şters şi se înlocuieşte cu un număr negativ. Această metodă se poate folosi atunci când avem doar numere pozitive în cadrul vectorului.**

Această operaţie are complexitatea maximă O(log n), unde log n reprezintă înălţimea meaximă a arborelui.

3.2.3 Testarea implementărilor pe diferite teste

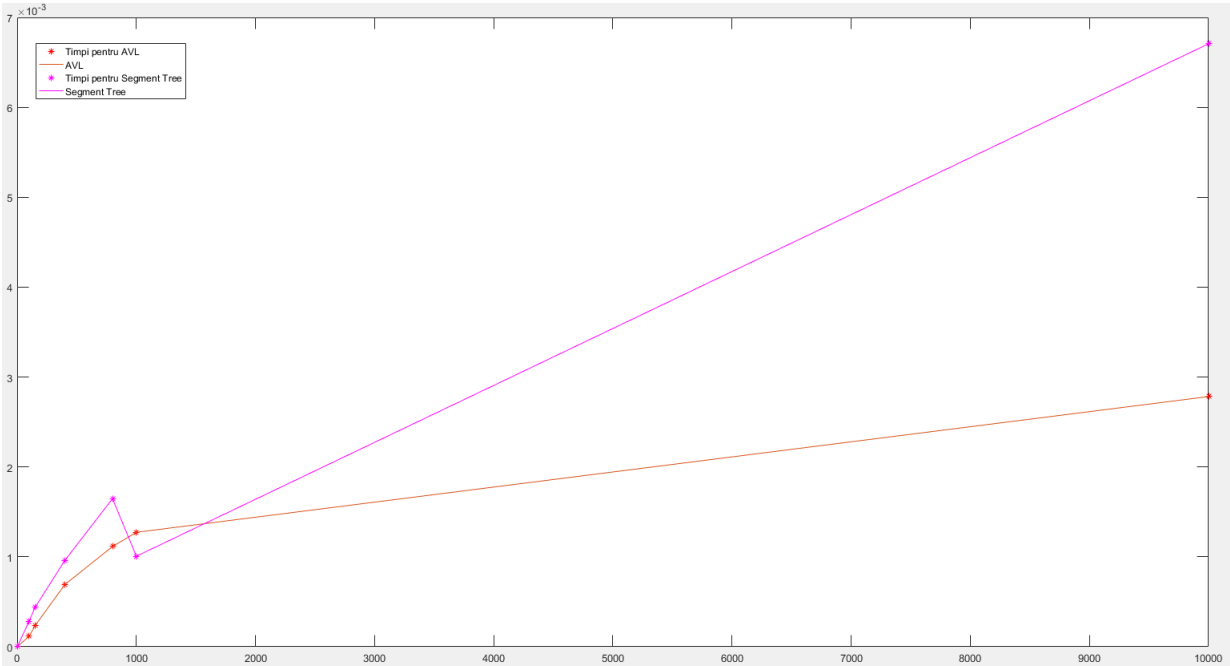

Figure 4: Evoluţia temporală la ştergerea elementelor

Se observă şi de pe grafic faptul că atunci când se vor şterge elementele din vector, este mult mai eficient arborele AVL. Acest lucru se datorează în primul rând datorită complexităţilor spaţiale diferite.

3.3 Găsirea elementului minim

3.3.1 Găsirea elementului minim într-un AVL Tree

Pentru a determina minimul din arbore, se va parcurge toată structura începând cu rădăcina şi se va parcurge cât mai la "stânga" posibil.

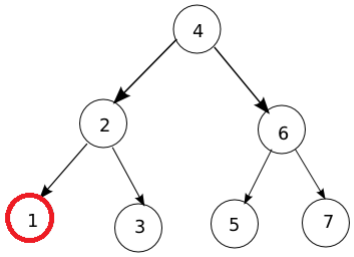

Figure 5: Elementul minim într-un arbore AVL

3.3.2 Găsirea elementului minim într-un Segment Tree

Pentru a determina minimul dintr-un anumit interval, se vor alege mai întâi cele două capete, se va căuta în arbore zonele unde se găsesc acele valori şi se determină minimul frunzelor. ! Se poate optimiza acest task prin păstrarea în nodurile interne, minimul dintre cei doi copii. Această operaţie are complexitatea de O(log n), unde log n reprezintă înălţimea meaximă a arborelui. [6]

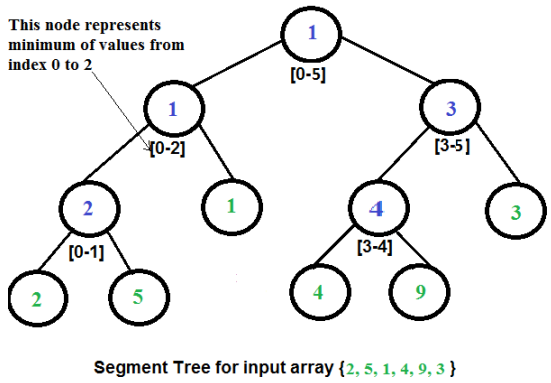

Figure 6: Elementul minim într-un arbore Segment

3.3.3 Testarea implementărilor pe diferite teste

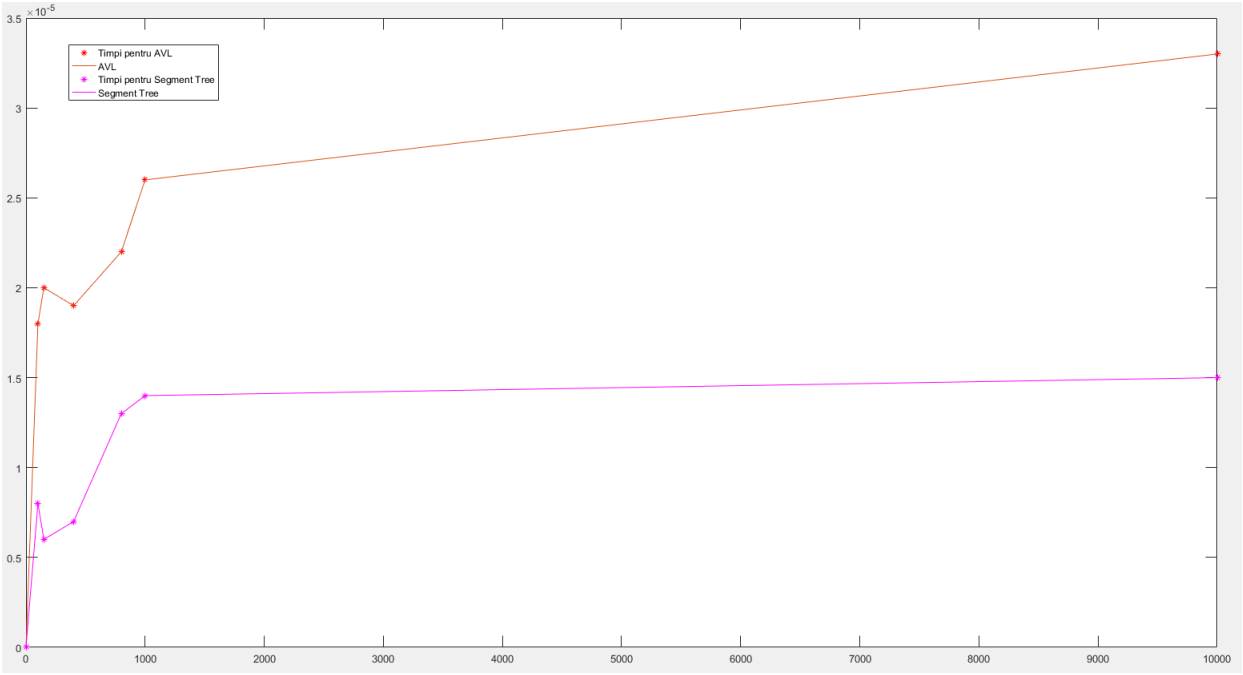

Figure 7: Evoluţia temporală la găsirea elementului minim

Se observă şi de pe grafic faptul că atunci când vine vorba de operaţii de tipul căutarea minumului, arborele Segment va avea un

timp de execuţie mult mai bun decât cel al arborelui AVL.

3.4 Găsirea elementelor mai mici decât un element într-un arbore AVL

Pentru a determina toate elementele mai mici decât un alt element, se va căuta prima dată elementul în arbore, iar dacă va fi găsit se va afişa întreg arborele stâng. Această operaţie are complexitatea de O(log n), unde log n reprezintă înălţimea maximă a arborelui.

3.4.1 Testarea implementării pe diferite teste

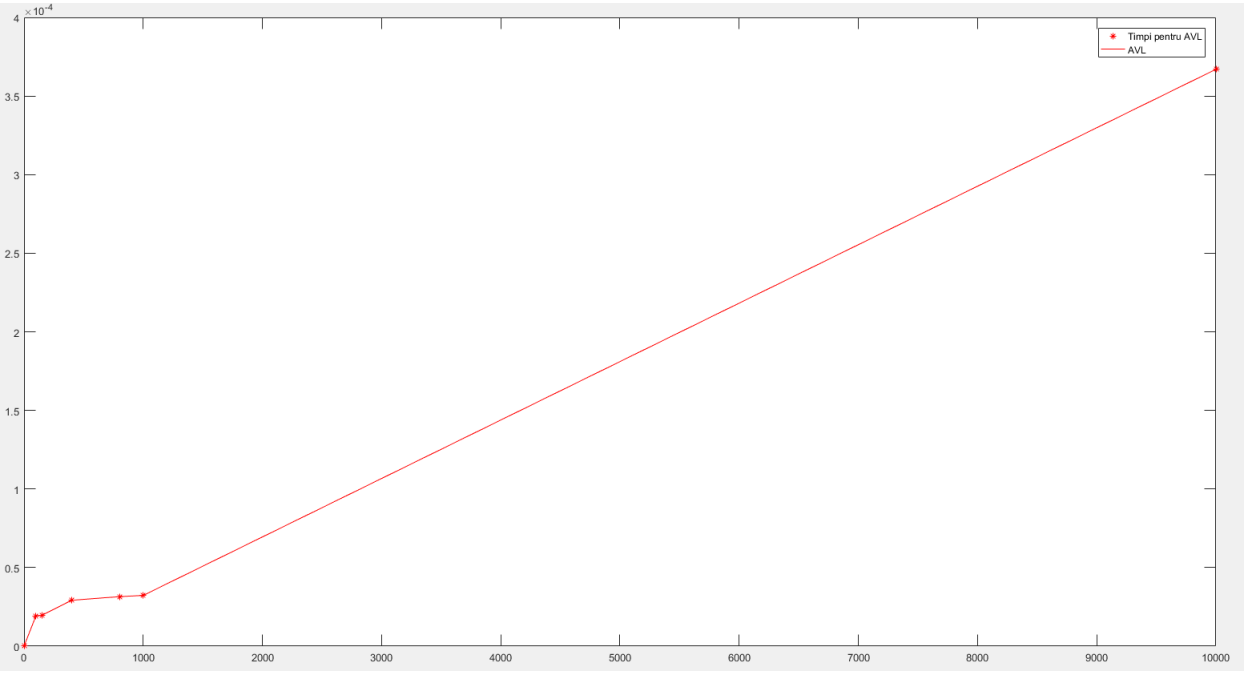

Figure 8: Evoluţia temporală la gasirea elementelor mai mici decât un element dat într-un AVL Tree

3.5 Înlocuirea unui element x cu y într-un arbore Segment

Pentru a putea înlocui un element dintr-un arbore Segment, se va parcurge lista de elemente, se va salva indexul elementului şi se va parcurge arborele până se va ajunge la intervalul de tip [index, index], înlocuidu-se valoarea iniţială cu cea finală. Aceastaă operaţie are complexitatea de O(log n), unde log n reprezintă înălţimea maximă a arborelui. [2]

3.5.1 Testarea implementărilor pe diferite teste

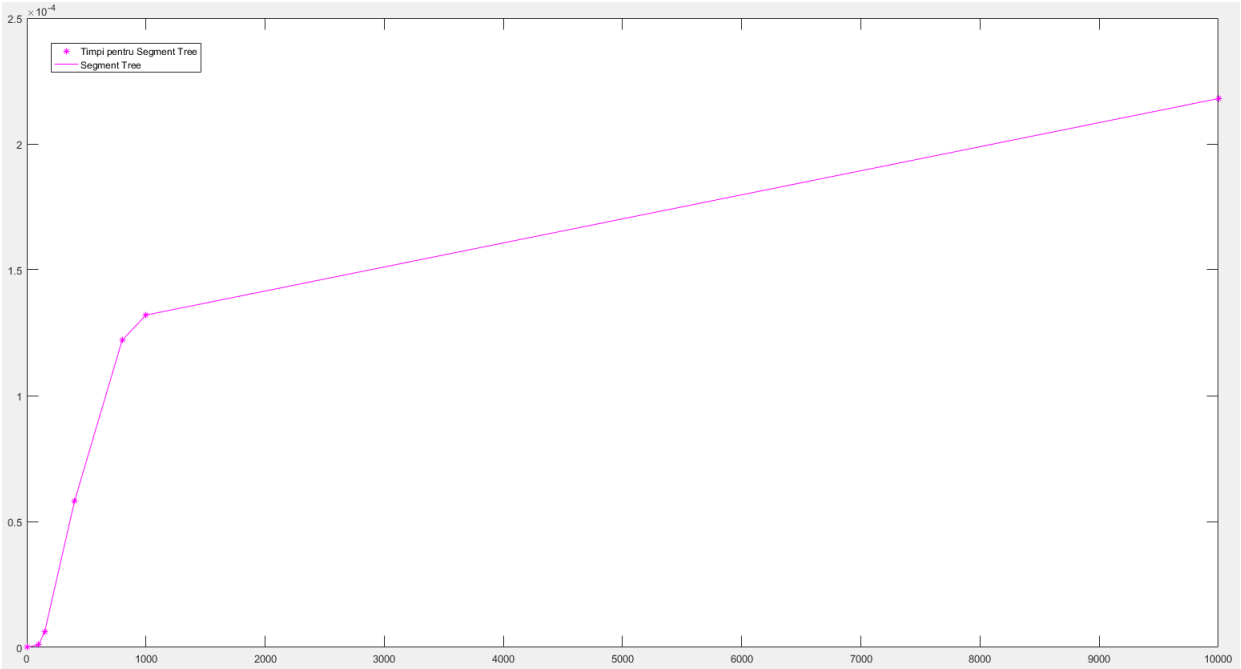

Figure 9: Evoluţia temporală la înlocuirea unui element existent într-un arbore Segment

4 Concluzie

Atât arborele AVL, cât şi arborele Segment sunt două structuri de date avansate, ce au scopul de a facilita şi de a îmbunătăţi modalitatea de rezolvarea a problemelor. După cum s-a observat în graficele de mai sus, fiecare structură are un comportament diferit, ce poate fi, sau nu, benefic în anumite situaţii.

Arborele AVL are timpi de execuţie asupra adăugării elementelor, respectiv asupra determinării minimului, mai mari, însă atunci când avem un set de date asupra căruia nu dorim să realizăm alte operţii în afară de cele declarate mai sus, este o structură uşor de înţeles, respectiv accesibilă.

Arborele Segment are timpi de execuţie mai slabi la "ştergere", respectiv la adăugarea elementelor, pe teste de dimensiuni mai mari (> 1000) , însă atunci când avem operaţii de procesare a elementelor, arborele Segment este o variantă mult mai bună.

5 Bibliografie

[1]. Detalii AVL
[2]. Detalii Segment Tree
[3]. Explicaţii contruire arbore AVL
[4]. Explicaţii contruire arbore Segment
[5]. Explicaţii pentru ştergerea elementelor dintr-un arbore AVL
[6]. Explicaţii pentru calcularea minimului într-un arbore Segment