

Proiect Analiza Algoritmilor

Structuri de Date - Cozi de prioritate

Arbore AVL și Max-Heap

Duțu Alin Călin
323 CD

Facultatea de Automatică și Calculatoare
Universitatea Politehnică București

18 decembrie 2020

Abstract. *Contextul acestei lucrări este prezentarea temei 1 la disciplina Analiza Algoritmilor parcursă în primul semestru al anului universitar 2020 - 2021. Această lucrare are ca scop analizarea anumitor operații efectuate de anumite structuri de date abstracte. Se iau în vedere implementarea, testarea și analiza algoritmilor ce se află în legătură cu scopul structurilor de date, dar și analiza beneficiilor pe care le aduc acești algoritmi cu scopul găsirii celei mai bune opțiuni pentru diverse situații. Raportul conține 13 pagini structurate pe 5 capitole care oferă o analiză amplă a temei abordate.*

Cuvinte cheie: AVL Tree, Max Heap, Element maxim, Complexitate, Timp de execuție

Cuprins

1	Introducere	2
1.1	Descrierea problemei rezolvate și specificarea soluțiilor alese	2
1.2	Specificarea criteriilor de evaluare	2
1.3	Exemple de aplicații practice pentru problema aleasă	2
2	Prezentarea Soluțiilor	3
2.1	Descrierea modului în care funcționează algoritmi	3
2.1.1	AVL Tree	3
2.1.2	Max - Heap	4
2.2	Analiza complexității soluțiilor	4
2.2.1	AVL Tree	4
2.2.2	Max-Heap	5
2.3	Avantaje și dezavantaje	6
3	Evaluare	7
3.1	Modalitatea de construire a testelor	7
3.2	Rezultatele testelor	7
3.3	Prezentarea rezultatelor testelor	12
3.4	Specificațiile sistemului de calcul	12
4	Concluzii	13
	Bibliografie	13

1 Introducere

1.1 Descrierea problemei rezolvate și specificarea soluțiilor alese

Problema aleasă este cea a analizei operațiilor unor structuri de date abstracte. Structurile alese fac parte din categoria cozilor de prioritate, astfel s-au ales pentru comparație arborele AVL și Max-Heap-ul și se vor compara prin utilizarea operațiilor comune acestora.

AVL tree este un arbore binar de căutare care are proprietatea de auto-echilibare ce îi permite să execute operațiile unui arbore binar de căutare obișnuit mult mai eficient. Pentru AVL se iau în vedere operațiile de creare a structurii, adăugare a unui element, operațiile de rotație stânga și rotație dreapta, găsirea elementului maxim și minim, eliminarea elementului maxim și a celui minim din structură și ștergerea structurii. Max-Heap-ul este o structură de date bazată pe un arbore complet cu proprietatea că nodul cu valoarea maximă se găsește chiar în rădăcină. Pentru Max - Heap se vor implementa operațiile de creare a structurii, adăugare a unui element, găsirea elementului maxim, swift Up și swift Down, extragerea elementului maxim și ștergerea structurii.

1.2 Specificarea criteriilor de evaluare

Pentru evaluarea structurilor se vor folosi teste variate, crescătoare, descrescătoare și generate random folosind un instrument online de generare a numerelor în ordine aleatoare[9] atât pentru AVL cât și pentru Max - Heap. Principalele criterii de evaluare luate în calcul pentru ambele structuri sunt timpii de execuție care vor testa rapiditatea structurilor pentru diferite input-uri, calculul complexităților și dificultatea implementării structurii propriu-zise.

1.3 Exemple de aplicații practice pentru problema aleasă

Având în vedere proprietățile AVL-ului față de alte structuri cum ar fi Arborii Roșu - Negru se poate constata că AVL-ul nu mai este foarte folosit în practică deoarece există structuri mult mai eficiente, însă un exemplu care ar fi perfect pentru folosirea acestei structuri este gestionarea datelor pentru inventarul trenurilor dintr-o rețea feroviară deoarece exemplul se bazează pe necesitatea căutării unui tren în funcție de greutate sau de inventar și nu necesită foarte multe inserări și ștergeri de obiecte. Max heap-ul are avantajul de a avea nodul cel mai mare în rădăcină și de aceea am putea avea ca exemplu de aplicație gestionarea pacienților dintr-un centru de analize în funcție de prioritatea lor întrucât se poate afla foarte repede care va fi următorul pacient. Aceste două exemple au fost implementate suplimentar pe lângă testele pentru operații și vor oferi informații dintr-o perspectivă practică care vor contribui la compararea structurilor.

2 Prezentarea Soluțiilor

Pentru acest proiect s-au implementat structurile de arbore AVL și Max-heap. Pentru ambele structuri s-a urmărit, în principiu, comportamentul operațiilor de: creare a structurii, adăugare a unui element, găsirea elementului maxim, extragerea elementului maxim sau ștergerea elementului maxim din structură și ștergerea structurii.

2.1 Descrierea modului în care funcționează algoritmi

2.1.1 AVL Tree

Arborele AVL este un arbore binar de căutare care are în plus proprietatea de auto-echilibrare. Această proprietate se referă la faptul că diferența înălțimii subarborelui stâng cu înălțimea subarborelui drept nu poate fi mai mare de un nod. Dacă această condiție este încălcată atunci arborele va fi reechilibrat pentru a se păstra proprietatea de echilibru[8].

O altă proprietate a arborelui AVL este factorul de balanță care este egal pentru oricare nod din arbore cu diferența dintre înălțimea subarborelui stâng și înălțimea subarborelui drept și poate lua valorile -1, 0, 1, -1 pentru un AVL left-heavy, 0 pentru un AVL echilibrat și 1 pentru un AVL right-heavy[8].

Pentru implementarea algoritmilor s-au folosit diferite surse de inspirație pentru implementarea structurii propriu-zise și pentru implementarea operațiilor acestuia, dar și cunoștințele dobândite până acum la cursurile de Structuri de date incluzând cursurile și laboratoarele parcurse[6, 8]. Implementarea acestei structuri cuprinde doar nodul rădăcină. La început s-a optat și pentru implementarea cu santinelă nil, însă pentru ușurință s-a renunțat la această variantă. Implementarea operațiilor este în mare parte iterativă cu excepția operațiilor de inserare și eliminare de elemente care sunt recursive.

Pentru verificarea operațiilor de inserare a unui element și de găsim a maximului respectiv minimului din structură s-a folosit funcția Inorder care afișează în fișierul de output toate nodurile arborelui Inorder. În plus s-au adăugat în fișier și rezultatele operațiilor de găsim a maximului și minimului, pentru comparare cu rezultatul funcției Inorder demonstrând astfel corectitudinea operației. Pentru operația de ștergere se mai afișează încă odată noile valori extreme și se verifică tot cu rezultatul funcției Inorder. De asemenea, pentru testarea inserării am folosit un instrument online care mi-a permis să verific arborele structura arborelui[1].

2.1.2 Max - Heap

Max-heap-ul este un arbore binar complet cu proprietatea că: pentru orice nod, cheia nodului este mai mare decât cheile din nodurile copii, dacă există copii [8].

Pentru implementarea Heap-ului s-au folosit în principal cunoștințele acumulate, dar și laboratoarele făcute la cursul de Structuri de Date[8]. Toate operațiile implementate sunt iterative.

Pentru verificarea inserării s-a folosit un instrument de vizualizare[3] cu care se poate vedea aranjarea elementelor pe Heap. Pentru maxim s-a folosit un instrument online care să afișeze maximul dintr-o listă de numere, în acest caz din input[5]. Iar pentru delete s-a efectuat aceeași verificare ca la maxim cu precizarea că s-a eliminat vechiul maxim din listă.

2.2 Analiza complexității soluțiilor

2.2.1 AVL Tree

Conform teoriei un arbore AVL are înălțimea $O(\log n)$. Urmează să fie demonstrată această afirmație. Fie h , înălțimea arborelui și $T(h)$ numărul minim de noduri cu înălțimea h . Pornim de la premisa că: Dacă subarborele stâng este umplut până la înălțimea $h-1$, atunci subarborele drept trebuie umplut până la înălțimea $h-2$. Astfel numărul minim de noduri cu înălțimea h este:

$$T(h) = T(h-1) + T(h-2) + 1; \text{ cu } T(0) = 1, T(1) = 2$$

$$T(h-1) = T(h-2) + T(h-3) + 1 \Rightarrow T(h) = 2 * T(h-2) + T(h-3) + 2$$

$$T(h-2) = T(h-3) + T(h-4) + 1 \Rightarrow T(h) = 3 * T(h-3) + T(h-4) + 2$$

$$\dots \text{ De unde rezultă că } T(h) = h * T(h-h) + (h-1) * T(h-h+1) + 2^h = 3h - 2 + 2^h \Rightarrow T(h) > 2^h \\ \Rightarrow \log T(h) > h. \text{ Fie } T(h) = n, \text{ unde } n \text{ este numărul de noduri} \Rightarrow \log n > h \Rightarrow h = O(\log n)$$

Create AVL

Crearea unui AVL gol este o operație foarte simplă fără apeluri recursive, fără structuri repetitive și care conține instrucțiuni de complexitate $O(1)$, deci crearea arborelui AVL în cel mai rău caz are complexitatea $O(1)$.

Rotate left & Rotate right

Operațiile de rotire ale nodurilor sunt necesare pentru echilibrarea arborelui în cazul în care factorul de balanță este mai mare decât 1 sau mai mic decât -1. De asemenea, rotirea propriu-zisă este o operație care nu necesită recurențe sau bucle repetitive, deci în cel mai rău caz rotirea fie ea stângă sau dreaptă are complexitatea $O(1)$.

Insert

Inserarea unui nod are trei etape: prima reprezintă crearea nodului care are complexitate $O(1)$ deoarece operația se execută iterativ și fără structuri repetitive, a doua etapă constă în găsirea locului unde trebuie

adăugat nodul. În worst case, se parcurge arborele pe toată înălțimea lui, asta însemnând complexitatea $O(\log n)$ și apoi urmează reechilibrarea arborelui, operație ce are complexitatea în cel mai rău caz tot $O(\log n)$, deoarece ajungând la frunze algoritmul se întoarce și face rotirile necesare pentru echilibrare ($O(1)$) de la frunză până la rădăcină, practic o parcurgere inversă a arborelui care are complexitate $O(\log n)$. Astfel Insert-ul ar avea complexitatea $O(2 * \log n) = O(\log n)$.

Get Min & Max

Pentru elementul minim și maxim se știe din teorie că ele se găsesc la extremitățile arborelui, în sensul că nodul minim este frunza cea mai din stânga, iar nodul maxim este frunza cea mai din dreapta. Deci, pentru găsirea fie a minimumului, fie a maximumului este necesară o parcurgere pe toată înălțimea arborelui care are complexitatea $O(\log n)$. Astfel complexitatea operației de găsire a minimumului și maximumului dintr-un AVL este $O(2 * \log n) = O(\log n)$.

Delete

Operația de ștergere a unui element, deși este mai complexă față de inserare, ea are aceeași complexitate. Motivul este că operația începe prin căutarea nodului de eliminat (în worst case $O(\log n)$), apoi înlocuirea nodului eliminat care are 3 cazuri: Când nodul eliminat nu are copii și atunci operația are complexitatea $O(1)$, când are un copil și atunci operația are complexitatea tot $O(1)$ și când are 2 copii, caz în care se poate alege fie valoarea maximă din subarborele stâng, fie valoarea minimă din subarborele drept (în această implementare s-a luat în considerare valoarea maximă din subarborele stâng). Acest caz are complexitatea $O(\log n)$ deoarece se parcurge arborele până la frunze. Ultima etapă a operației este reechilibrarea arborelui ($O(\log n)$). În total avem complexitatea $O(3 * \log n) = O(\log n)$.

2.2.2 Max-Heap

Create Heap

Operația de creare a Heap-ului nu are bucle repetitive sau recursivități, de aceea complexitatea operației este $O(1)$.

Sift Up & Sift Down

Operația de Sift Up în această implementare nu este recursivă și nu are blocuri repetitive, deci complexitatea este $O(1)$. În schimb Sift Down în cel mai rău caz duce nodul rădăcină până la frunze, practic o parcurgere a arborelui, deci complexitatea este $O(h)$, unde h este înălțimea arborelui, însă dat fiind faptul că în teorie se spune că heap-ul este un arbore complet se poate afirma că $O(\log n)$ este complexitatea operației de Sift Down.

Insert

În cel mai rău caz arborele adaugă un nod la frunze și după se face sift up până când nodul nou ajunge la rădăcină. Știind că complexitatea lui Sift Up este $O(1)$ se deduce că insertul este echivalent în acest caz cu parcurgerea arborelui de sus în jos ceea ce înseamnă complexitate $O(\log n)$. Deci insertul are complexitatea $O(\log n)$.

Get Max

Știind că valoarea maximă se găsește la rădăcină rezultă complexitate $O(1)$, deci complexitatea găsirii elementului maxim este $O(1)$.

Extract Max

Pentru Extract Max, operația propriu-zisă de extragere este echivalentă cu operația Get Max care are complexitatea $O(1)$ împreună cu operația de Sift Down care mai adaugă complexitatea $O(\log n)$. Deci, complexitatea operației de extragere a elementului maxim în cazul cel mai nefavorabil este $O(\log n + 1) = O(\log n)$.

2.3 Avantaje și dezavantaje

Principalele avantaje ale structurilor AVL Tree și Max-Heap sunt rapiditatea operațiilor de inserare, căutare și găsim a maximului, dar și eliminarea maximului deoarece toate aceste operații au complexitate $O(\log n)$ care este mult mai bună față de complexitatea unui arbore binar de căutare obișnuit de exemplu și oferă mai multă flexibilitate. Principalul dezavantaj pentru AVL este dificultatea implementării codului pentru operații, iar pentru Max-Heap principalul dezavantaj ar fi organizarea memoriei.

3 Evaluare

3.1 Modalitatea de construire a testelor

Pentru evaluarea operațiilor structurilor de AVL Tree și Max - Heap s-au implementat în etapa trecută câte 5 teste care au fost construite folosind un generator de numere distincte[9] și un instrument de sortare online[2], numerele fiind generate într-o ordine aleatoare și mai mici de 1000. Aceste teste au un număr de 5, 10, 25, 50 și 100 de elemente distincte. Pentru o mai bună acuratețe a performanței operațiilor se vor mai adăuga un set de 5 teste pentru fiecare structură. Un test va fi de 250 de elemente generate în ordine aleatoare, două teste vor fi de 500 și 1000 de elemente și vor fi generate descrescător, iar celelalte două teste vor fi de 5000 și 10000 de elemente generate crescător. Fiecare test va rula toate operațiile structurii de date reprezentate. Pentru AVL se vor executa: Crearea structurii, Inserarea elementelor, Căutarea maximului și minimului, Eliminarea maximului și minimului din structură și Ștergerea structurii, iar pentru Heap se vor executa: Crearea structurii, Inserarea elementelor, Preluarea nodului cu valoare maximă, Extrgerea nodului cu valoare maximă și ștergerea structurii.

3.2 Rezultatele testelor

Tabela 3.1: Timpii de execuție pentru crearea structurilor

N	AVL Tree	Max - Heap
5	0.001178 ms	0.001106 ms
10	0.000726 ms	0.001051 ms
25	0.000682 ms	0.000986 ms
50	0.000677 ms	0.00114 ms
100	0.000676 ms	0.002253 ms
250	0.000707 ms	0.003481 ms
500	0.000898 ms	0.005623 ms
1000	0.000984 ms	0.010363 ms
5000	0.000797 ms	0.04761 ms
10000	0.000861 ms	0.362846 ms

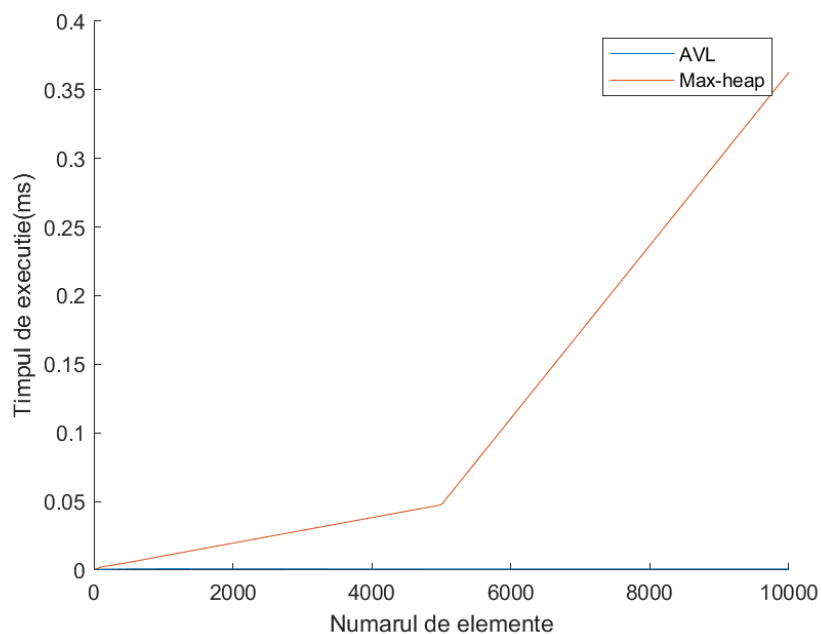


Figura 3.1: Diagrama pentru crearea structurilor

Tabela 3.2: Timpii de execuție pentru inserarea elementelor

N	AVL Tree	Max - Heap
5	0.001159 ms	0.010295 ms
10	0.017719 ms	0.010691 ms
25	0.041883 ms	0.02151 ms
50	0.096804 ms	0.052967 ms
100	0.221705 ms	0.109183 ms
250	0.591891 ms	0.225686 ms
500	1.29952 ms	0.379961 ms
1000	3.05291 ms	0.692936 ms
5000	19.6206 ms	11.6697 ms
10000	59.8021 ms	24.26 ms

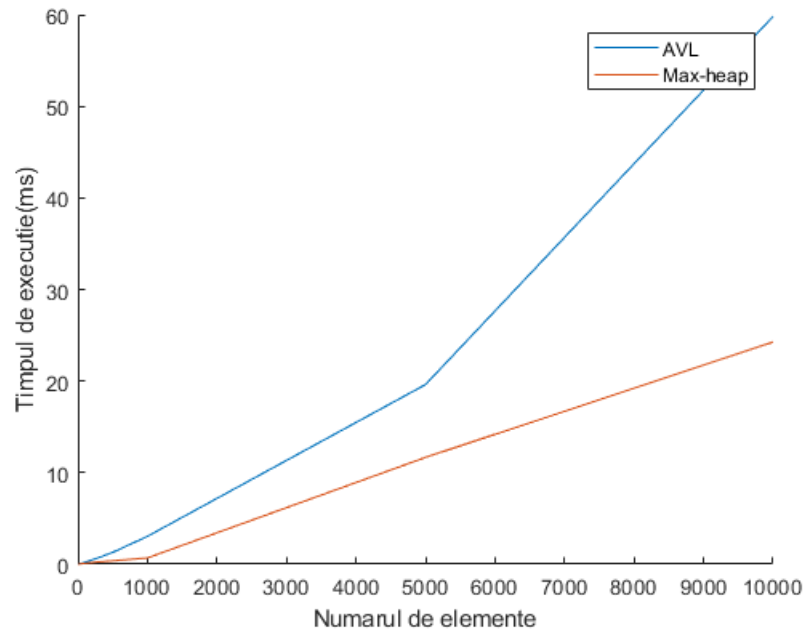


Figura 3.2: Diagrama pentru inserarea elementelor

Tabela 3.3: Timpii de execuție pentru găsirea elementelor maxime

N	AVL Tree	Max - Heap
5	0.007937 ms	0.000431 ms
10	0.008015 ms	0.000409 ms
25	0.008226 ms	0.000412 ms
50	0.008193 ms	0.000427 ms
100	0.008362 ms	0.00044 ms
250	0.008447 ms	0.000453 ms
500	0.010743 ms	0.000468 ms
1000	0.01005 ms	0.000466 ms
5000	0.011588 ms	0.000485 ms
10000	0.012177 ms	0.000631 ms

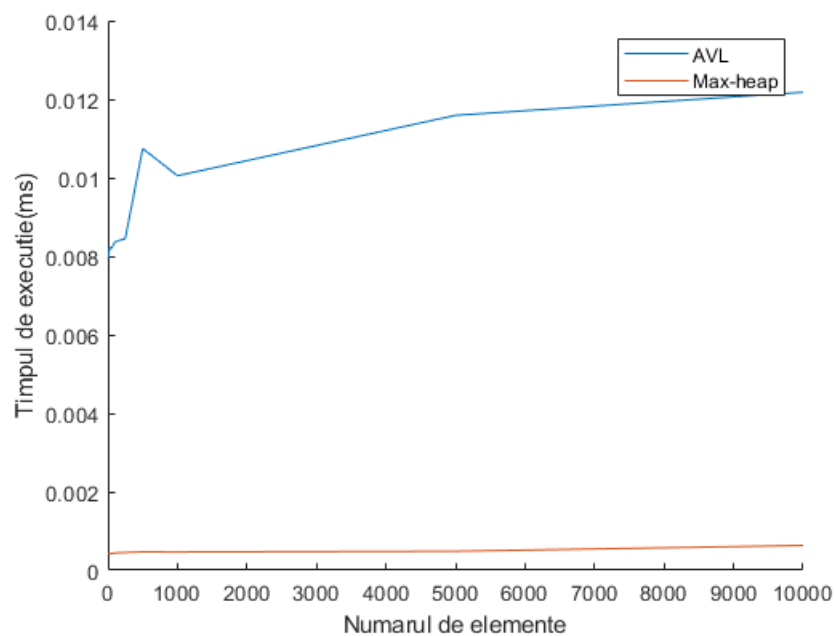


Figura 3.3: Diagrama pentru găsirea elementelor maxime

Tabela 3.4: Timpii de execuție pentru eliminarea elementelor maxime

N	AVL Tree	Max - Heap
5	0.000388 ms	0.000667 ms
10	0.000677 ms	0.000737 ms
25	0.000573 ms	0.001418 ms
50	0.000555 ms	0.001529 ms
100	0.00108 ms	0.001674 ms
250	0.001119 ms	0.001795 ms
500	0.000832 ms	0.001991 ms
1000	0.000983 ms	0.002041 ms
5000	0.005285 ms	0.00266 ms
10000	0.00594 ms	0.002867 ms

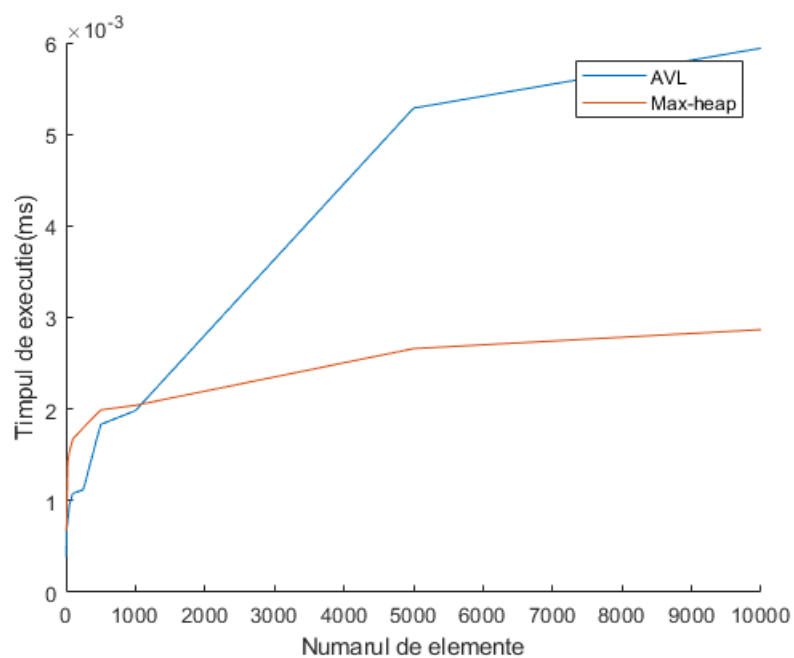


Figura 3.4: Diagrama pentru eliminarea elementelor maxime

Tabela 3.5: Timpii de execuție pentru ștergerea structurii

N	AVL Tree	Max - Heap
5	0.000501 ms	0.000473 ms
10	0.000808 ms	0.000328 ms
25	0.001794 ms	0.000268 ms
50	0.003458 ms	0.000297 ms
100	0.007003 ms	0.000362 ms
250	0.013364 ms	0.000416 ms
500	0.02463 ms	0.000444 ms
1000	0.04781 ms	0.00032 ms
5000	0.369463 ms	0.00076 ms
10000	0.753662 ms	0.00088 ms

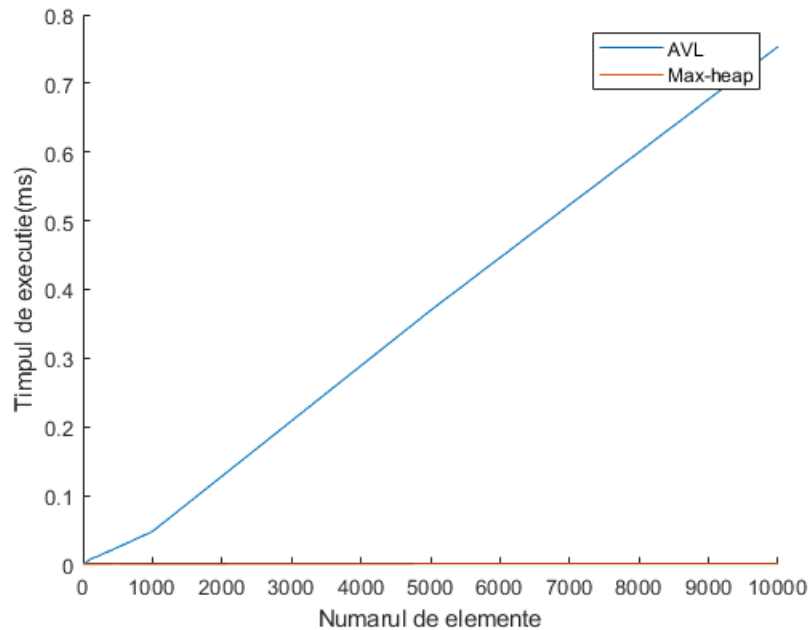


Figura 3.5: Diagrama pentru ștergerea structurilor

3.3 Prezentarea rezultatelor testelor

Pentru crearea structurii propriu-zise se poate observa că AVL-ul are pentru orice set de date o eficiență mult mai bună din punctul de vedere al timpului de execuție față de Max-Heap, însă diferența nu este foarte mare. Pentru Insert pentru teste sub 1000 de elemente structurile sunt la fel de eficiente, însă pentru teste mai mari Max - Heap - ul este mult mai bun față de AVL. Pentru găsirea elementului maxim Max-Heap-ul având valoarea maximă în nodul rădăcină este clar că vă scoate timpi foarte mici de execuție datorită complexității $O(1)$ care în comparație cu $O(\log n)$ este diferență semnificativă în special la seturi foarte mari de date. Pentru eliminarea elementului maxim AVL-ul este mai eficient decât Max Heap-ul pentru input-uri mai mici de 1200 de elemente, însă pentru testele mai mari Max-Heap-ul este mai bun. În final, pentru ștergerea structurilor, Max-Heap-ul are o performanță mult mai bună în comparație cu AVL.

3.4 Specificațiile sistemului de calcul

Tema a fost implementată folosind C++, compilată folosind G++ versiunea 9.3.0 și a fost rulată pe o mașină virtuală cu Ubuntu 20.04 cu 6GB RAM alocată. Configurația host-ului este Windows 10 cu Procesor Intel Core i7-5500U cu 2 core-uri de 2,4 GHZ, 16 GB RAM și placă Video NVidia cu 2GB.

4 Concluzii

În concluzie, pentru situațiile în care este nevoie de o bază de date nu foarte populată cu puține adăugări și ștergeri de date se poate opta pentru o structură de date de tip AVL, iar pentru cazul în care este nevoie de o bază de date al căror principal atribut este un număr prioritar se poate folosi pentru cea mai bună performanță un Max - Heap.

Bibliografie

- [1] AVL Tree Visualization (Ultima accesare: 17.12.2020). <https://www.cs.usfca.edu/~galles/visualization/AVLtree.html>.
- [2] Endmemo - Online Number Sorter (Ultima accesare: 18.12.2020). <http://www.endmemo.com/math/numsort.php>.
- [3] Heap Visualization (Ultima accesare: 17.12.2020). <http://btv.melezinek.cz/binary-heap.html>.
- [4] Lncs Template, (Ultima accesare: 17.12.2020). <https://www.overleaf.com/latex/templates/springer-lecture-notes-in-computer-science/kzwwpvhwnvfj#.WtR5Hy5ua71>.
- [5] Minimum and maximum of a list online (Ultima accesare: 17.12.2020). <https://pinetools.com/minimum-maximum-list>.
- [6] Sandeep Jain; Shikhar Goel; Dharmesh Singh; Shubham Baranwal. GeeksforGeeks AVL (Ultima accesare: 16.12.2020). <https://www.geeksforgeeks.org/avl-tree-set-1-insertion/> , <https://www.geeksforgeeks.org/avl-tree-set-2-deletion/>.
- [7] Gabriela Ciuprina. Template Latex v5 (Ultima accesare: 16.12.2020). <https://acs.curs.pub.ro/2019/mod/resource/view.php?id=19677>.
- [8] Echipa de Structuri de Date Universitatea Politehnică București. Structuri de Date (ultima accesare: 17.12.2020). <https://acs.curs.pub.ro/2019/course/view.php?id=145>.
- [9] Dr Mads Haahr; Dr Svend Haahr. Random.org, Integer - Sets (Ultima accesare: 17.12.2020). <https://www.random.org/integer-sets/>.