Proiect Analiza Algoritmilor
Structuri de Date - Cozi de prioritate
Arbore AVL şi Max-Heap

Duţu Alin Călin
323 CD

Facultatea de Automatică şi Calculatoare
Universitatea Politehnică Bucureşti

12 aprilie 2021

Abstract. *Contextul acestei lucrări este prezentarea temei 1 la disciplina Analiza Algoritmilor parcursă în primul semenstru al anului universitar 2020 - 2021. Această lucrare are ca scop analizarea anumitor operaţii efectuate de anumite structuri de date abstracte. Se iau în vedere implementarea, testarea şi analiza algoritmilor ce se află în legătură cu scopul structurilor de date, dar şi analiza beneficiilor pe care le aduc aceşti algoritmi cu scopul găsirii celei mai bune opţiuni pentru diverse situaţii. Raportul conţine 13 pagini strcturate pe 5 capitole care oferă o analiză amplă a temei abordate.*
Cuvinte cheie: *AVL Tree, Max Heap, Element maxim, Complexitate, Timpi de execuţie*

Cuprins

1 Introducere

1.1 Descrierea problemei rezolvate și specificarea soluțiilor alese

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

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

1.2 Specificarea criteriilor de evaluare

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

1.3 Exemple de aplicații practice pentru problema aleasă

Având în vedere proprietăţile AVL-ului faţă de alte structuri cum ar fi Arborii Roşu - Negru se poate constata că AVL-ul nu mai este foarte folosit in practică deoarece există structuri mult mai eficiente, însa un exemplu care ar fi perfect pentru folosirea acestei structuri este gestionarea datelor pentru inventarul trenurilor dintr-o reţea feroviară deoarece exemplul se bazează pe necesitatea căutării unui tren în funcţie de greutate sau de inventar şi nu necesită foarte multe inserări şi ştergeri de obiecte. Max heap-ul are avantajul de a avea nodul cel mai mare în rădăcină şi de aceea am putea avea ca exemplu de aplicaţie gestionarea pacienţior dintr-un centru de analize în functie 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 operaţii şi vor oferi informaţii dintr-o prespectivă practică care vor contribui la compararea structurilor.

2 Prezentarea Soluțiilor

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

2.1 Descrierea modului în care functionează algoritmii

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 diferenţa înălţimii subarborelui stâng cu înălţimea suboarborelui drept nu poate fi mai mare de un nod. Dacă această condiţie este încălcată atunci arborele va fi reechilibrat pentru a se păstra proprietatea de echilibru[8].

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

Pentru implementarea algoritmilor s-au folosit diferite surse de inspiraţie pentru implementarea structurii propriu-zise şi pentru implementarea operaţiilor acestuia, dar si cunoştinţele dobândite până acum la cursurile de Structuri de date incluzând cursurile şi laboratoarele parcurse[6, 8]. Implementarea acestei structuri cuprinde doar nodul rădăcină. La început s-a optat şi pentru implementarea cu santinelă nil, insă pentru uşurinţă s-a renunţat la această variantă. Implementarea operaţiilor este in mare parte iterativă cu excepţia operaţiilor de inserare şi eliminare de elemente care sunt recursive.

Pentru verificarea operaţiilor de inserare a unui element şi de găsire a maximului respectiv minimului din structură s-a folosit funcţia Inorder care afişează în fişierul de output toate nodurile arborelui Inorder. În plus s-au adăugat în fişier şi rezulatele operaţiilor de găsire a maximului şi minimului, pentru comparare cu rezultatul functiei Inorder demonstrând astfel corectitudinea operaţiei. Pentru operaţia de ştergere se mai afişează inca odată noile valori extreme şi se verifică tot cu rezultatul functiei 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 cunoştinţele acumulate, dar şi laboratoarele făcute la cursul de Structuri de Date[8]. Toate operaţiile 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 afişeze maximul dintr-o listă de numere, în acest caz din input[5]. Iar pentru delete s-a efectuat aceeaşi 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 înălţimea O(log n). Urmează să fie demonstrată această afirmaţie. Fie h, înălţimea arborelui şi T(H) numărul minim de noduri cu înălţimea h. Pornim de la premisa că: Dacă subarborele stăng este umplut până la înălţimea h-1, atunci subarborele drept trebuie umplut până la înălţimea h-2. Astfel numărul minim de noduri cu înălţimea h este:
$T(h) = T(h-1) + T(h-2) + 1$; 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 * $T(h-3) + T(h-4) + 2$
... 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. Fie $T(h) = n$, unde n este numărul de noduri $\Rightarrow log\ n >$ h $\Rightarrow h = O(log\ n)$

Create AVL

Crearea unui AVL gol este o operaţie foarte simplă fără apeluri recursive, fără structuri repetitive şi care conţine instructiuni de complexitate O(1), deci crearea arborelui AVL în cel mai rău caz are complexitatea O(1).

Rotate left & Rotate right

Operaţiile de rotire ale nodurilor sunt necesare pentru echilibrarea arborelui în cazul în care factorul de balanţă este mai mare decât 1 sau mai mic decât -1. De asemeneta, rotirea propriu-zisă este o operaţie care nu necesită recurenţe sau bucle repetitive, deci in 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 operaţia se execută iterativ şi fără structuri repetitive, a doua etapă constă în gasirea locului unde trebuie

adăugat nodul. În worst case, se parcurge arborele pe toata înalţimea lui, asta însemnând complexitatea O(log n) şi apoi urmează reechilibrarea arborelui, operaţie ce are complexitatea in cel mai rău caz tot O(log n), deoarece ajungand la frunze algoritmul se intoarce şi 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 şi maxim se ştie din teorie că ele se găsesc la extremităţile 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 minimului, fie a maximului este necesară o parcurgere pe toată înălţimea arborelui care are complexitatea O(log n). Astfel complexitatea operaţiei de găsire a minimului şi maximului dintr-un AVL este O(2 * log n) = O(log n).

Delete

Operaţia de ştergere a unui element, desi este mai complexă faţă de inserare, ea are aceeaşi complexitate. Motivul este că operaţia incepe prin căutarea nodului de eliminat (in worst case O(log n)), apoi inlocuirea nodului eliminat care are 3 cazuri: Când nodul eliminat nu are copii şi atunci operaţia are complexitatea O(1), când are un copil şi atunci operaţia are complexitatea tot O(1) şi când are 2 copii, caz în care se poate alege fie valoarea maximă din subarborele stâng, fie valorea minimă din subarborele drept (in 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 operaţiei este reechilibrarea arborelui (O(log n)). În total avem complexitatea O(3 * log n) = O(log n).

2.2.2 Max-Heap

Create Heap

Operaţia de creare a Heap-ului nu are bucle repetitive sau recursivităţi, de aceea complexitatea operaţiei este O(1).

Sift Up & Sift Down

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

Insert

În cel mai rău caz arborele adaugă un nod la frunze si după se face sift up până când nodul nou ajunge la rădăcină. Stiind 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

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

Extract Max

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

2.3 Avantaje și dezavantaje

Principalele avantajele ale structurilor AVL Tree şi Max-Heap sunt rapiditatea operaţiilor de inserare, căutare şi găsire a maximului, dar şi eliminarea maximului deoarece toate aceste operaţii au complexitate O(log n) care este mult mai bună faţa de complexităţea unui arbore binar de căutare obişnuit de exemplu şi oferă mai multă flexibilitate. Principalul dezavantaj pentru AVL este dificultatea implementării codului pentru operaţii, iar pentru Max-Heap principalul dezavantaj ar fi organizarea memoriei.

3 Evaluare

3.1 Modalitatea de construire a testelor

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

3.2 Rezultatele testelor

Tabela 3.1: Timpii de execuţie 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

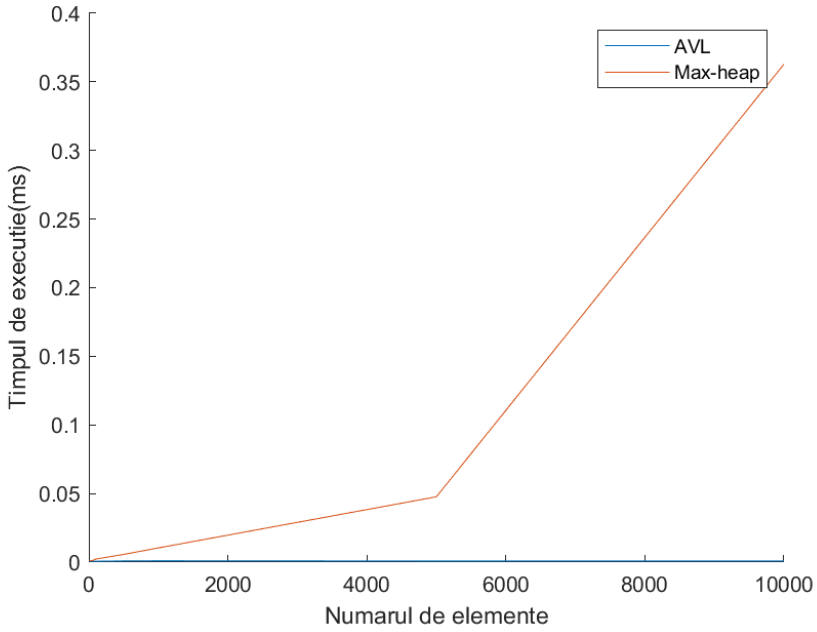

Figura 3.1: Diagrama pentru crearea structurilor

Tabela 3.2: Timpii de execuţie 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

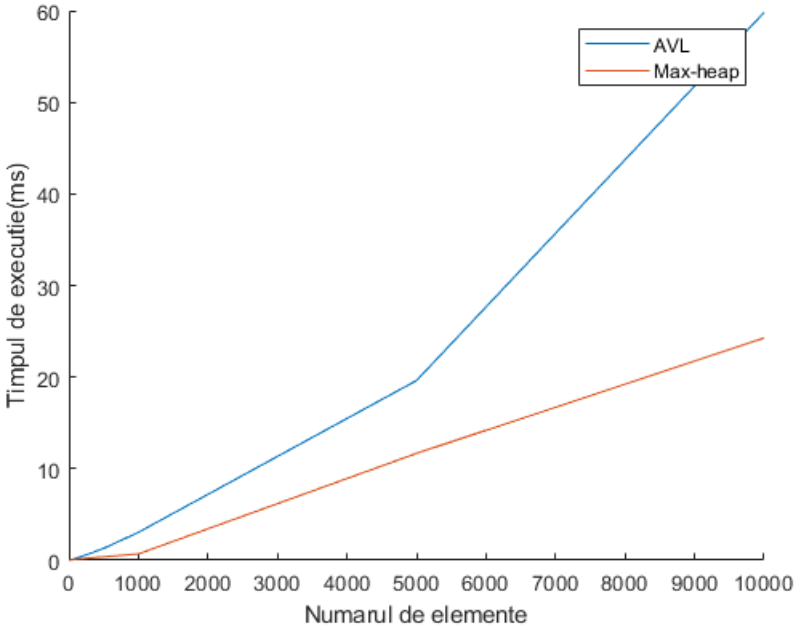

Figura 3.2: Diagrama pentru inserarea elementelor

Tabela 3.3: Timpii de execuţie 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

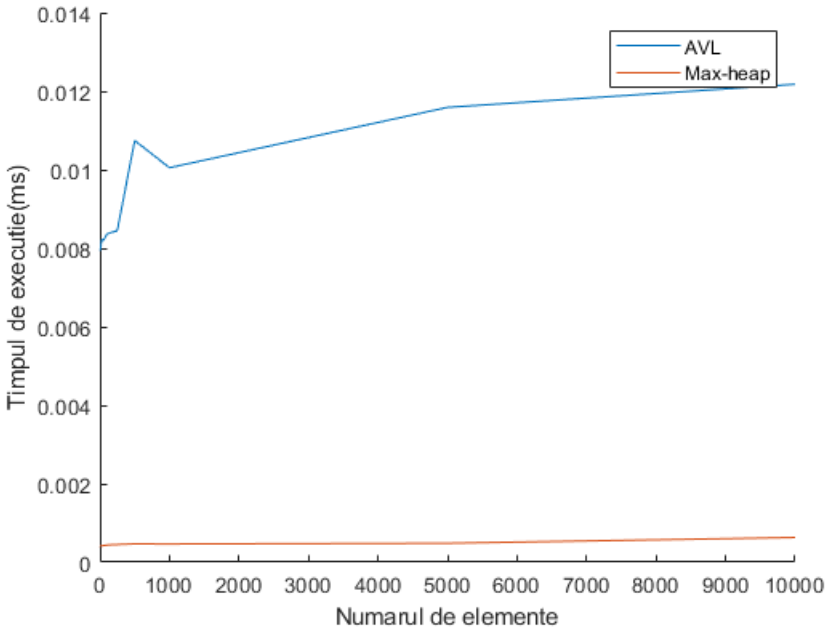

Figura 3.3: Diagrama pentru găsirea elementelor maxime

Tabela 3.4: Timpii de execuţie 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

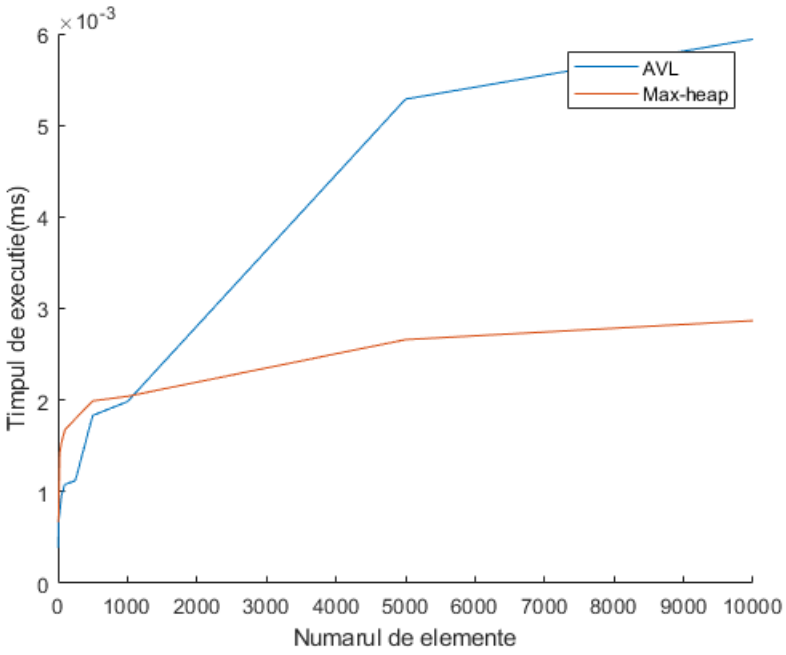

Figura 3.4: Diagrama pentru eliminarea elementelor maxime

Tabela 3.5: Timpii de execuţie pentru ştergerea 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

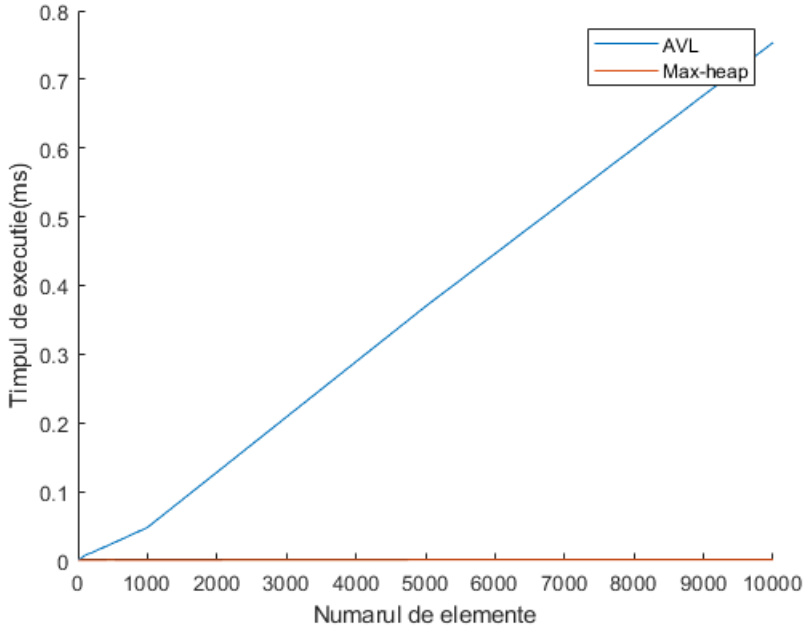

Figura 3.5: Diagrama pentru ştergerea structurilor

3.3 Prezentarea rezultatelor testelor

Pentru crearea structurii propriu-zise se poate observa că AVL-ul are pentru orice set de date o eficienţă mult mai bună din punctul de vedere al timpului de execuţie faţă de Max-Heap, însă diferenţa nu este foarte mare. Pentru Insert pentru teste sub 1000 de elemente structurile sunt la fel de eficiente, insă pentru teste mai mari Max - Heap - ul este mult mai bun faţă de AVL. Pentru găsirea elementului maxim Max-Heap-ul având valoarea maximă in nodul rădăcină este clar că vă scoate timpi foarte mici de execuţie datorită complexităţii O(1) care în comparaţie cu O(log n) este diferenţă semnificativă in 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, însa pentru testele mai mari Max-Heap-ul este mai bun. În final, pentru ştergerea structurilor, Max-Heap-ul are o performanţă mult mai bună în comparaţie cu AVL.

3.4 Specificațiile sistemului de calcul

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

4 Concluzii

În concluzie, pentru situaţiile în care este nevoie de o bază de date nu foarte populată cu puţine adăugări şi ştergeri 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 performanţă 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 Bucureşti. 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/.