Proiect Structuri de Date

Realizat de: Buță Gabriel, Lăzăroiu Teodora, Roman Andrei

1 Motivația structurii de date

În cadrul proiectului nostru am ales să folosim un *treap* ca și structură de date. Un treap reprezintă o combinație între un arbore binar de căutare și un max-heap în care fiecare nod conține 3 valori: *o cheie*, *o prioritate* și *o greutate*. Astfel, cheia urmărește ordinea de parcurgere în inordine a unui arbore binar de căutare iar prioritatea este aleasă la întâmplare cu scopul menținerii arborelui echilibrat. Greutatea unui nod reprezintă marimea subarborelui care are ca rădăcină nodul respectiv.

Utilizarea unui treap pentru reprezentarea unei mulțimi de numere întregi reprezintă un avantaj datorită unicității elementelor din această structură de date și a rapidității operațiilor implementate.

2 Analiza complexității

Prin proprietatea unui arbore balansat orice element poate fi accesat într-un timp maxim de *O(log n)*, unde n este numărul de elemente din mulțime. Așadar toate operațiile pe o mulțime de numere vor avea complexitatea de *O(log n)*, mai puțin *cardinal* care are complexitatea *O(1)*.

2.1 Inserare și ștergere

Un treap folosește rotații pentru a balansa arborele și pentru a păstra proprietatea de max-heap la realizarea operațiilor ce modifică structura acestuia. Astfel, la inserare se coboară pe arbore până la un nod terminal astfel încât să păstreze proprietatea arborelui binar de căutare după cheie, după care se folosește de rotații pentru a se păstra regula max-heapului. În cazul ștergerii se foloseste proprietatea arborelui binar de căutare pentru a se găsi nodul apoi se folosesc rotații până când nodul dorit ajunge frunză, iar apoi se șterge.

2.2 Căutare

Pentru operațiile de căutare a unui element din mulțime, inclusiv aflarea minimului si maximului, se realizează căutarea standard pentru un arbore binar de căutare balansat care are complexitatea de timp medie de *O(log n)*.

3 Avantaje și dezavantaje

Avantajele unui treap sunt: memoria alocată dinamic și efectuarea tuturor operațiilor cerute în cel mult *O(log n)*. De asemenea, această structură necesită introducerea de elemente distincte, ceea ce se suprapune cu proprietatea unei mulțimi de numere întregi. Aceasta poate fi văzută atât ca un avantaj cât și ca un dezavantaj deoarece în cazul unui fișier de testare ce conține numere indentice, algoritmul va ignora aceste valori.

4 Modalitatea de testare

Pentru testarea programului am creat un generator care returnează un număr între 1 și 9 care reprezintă una dintre operațiile cerute de problemă: inserare (1), ștergere (2), minim (3), maxim (4), cardinal (5), este_in (6), succesor (7), predecesor (8) și k_element (9). În cazul inserării, ștergerii, succesorului, predecesorului, funcției k_element și funcției este_in returnează încă un număr random necesar acelei operații folosind niște optimizări.

Timpul de rulare al testelor în CodeBlocks:

- primul test: 19 operații - aproximativ 0.02 secunde
- al doilea test: 100 de operații cu valori de maxim 5 cifre - aproximativ 0.02 secunde
- al treilea test: 100,000 de operații cu valori de maxim 5 cifre - aproximativ 0.12 secunde
- al patrulea test: 1,000,000 de operații cu valori de maxim 5 cifre - aproximativ 1 secundă
- al cincilea test: 10,000 de operații cu valori de maxim 10 cifre - aproximativ 0.03 secunde
- al șaselea test: 1,000,000 de operații cu valori de maxim 10 cifre - aproximativ 1.2 secunde

În urma testelor realizate se poate observa că programul va rula pentru orice valori în intervalul int, pentru oricâte operații pe mulțimi, în limita resurselor calculatorului pe care este rulat programul și a timpului care îi este pus la dispoziție.

5 Sales pitch

În programare timpul reprezintă un factor extrem de important iar un interval de câteva milisecunde poate face diferența între un succes și un eșec. Din acest motiv am ales o structură de date ce rezolvă toate cerințele într-un timp minimal. Deoarece reprezintă o combinație între un arbore binar de căutare și un max-heap, treap-ul reprezintă o structură complexă ce îmbina avantajele celor două. Această structură are o complexitate de *O(log n)* pentru a implementa operațiile pe o mulțime de n numere întregi în mod eficient datorită balansării automate oferită de folosirea priorităților, aceasta fiind dovedită probabilistic.