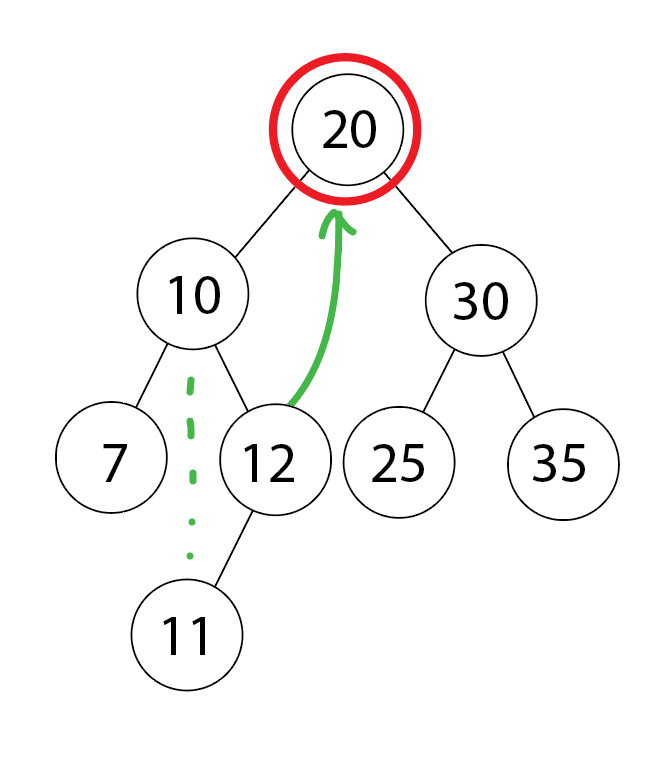
**Raport final proiect**

1. **Analiza timp a programului**
2. inserare(S, x):

Complexitatea pe timp este O(n)

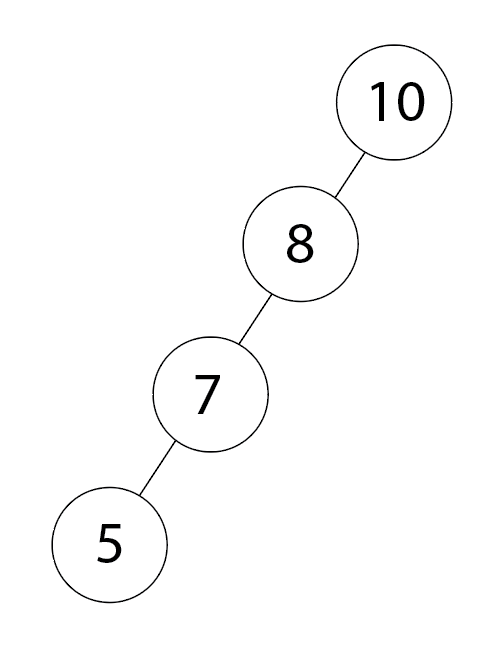
Worst-case: un arbore în care avem un număr de nivele egal cu numărul nodurilor -1, fiind nevoie sa treci prin fiecare nod existent.

1. ștergere(S, x):

Complexitatea pe timp este O(n)

Worst-case: ștergerea unui nod care are ambii fii. Dacă un nod are ambii fii nenuli va căuta un înlocuitor - cel mai mare număr mai mic decat el - adică se duce in stanga și apoi în dreapta pana intalneste null și se reapelează pe nodul înlocuitor; cand se reapeleaza se cauta iar nodul înlocuitor ( care are fii în stânga) și intra in if-ul pentru există doar fiu stang și dă return.

(Ex: la ștergerea lui 20, 12 devine rădăcina, iar 11 devine fiu drept al lui 10)

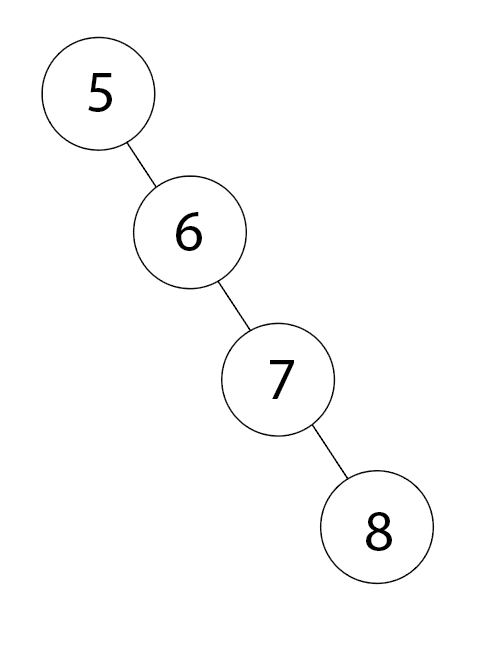


1. min(S, x):

Complexitatea pe timp este O(n)

Worst-case: un arbore în care avem un număr de nivele egal cu numărul nodurilor -1, iar toate nodurile au doar fii în stânga.

(Ex: min = 5)

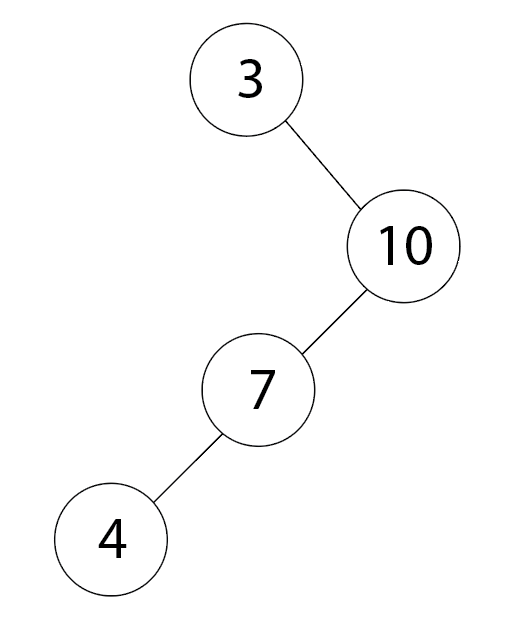


1. max(S, x):

Complexitatea pe timp este O(n)

Worst-case: un arbore în care avem un număr de nivele egal cu numărul nodurilor -1, iar toate nodurile au doar fii în dreapta.

(Ex: max = 8)

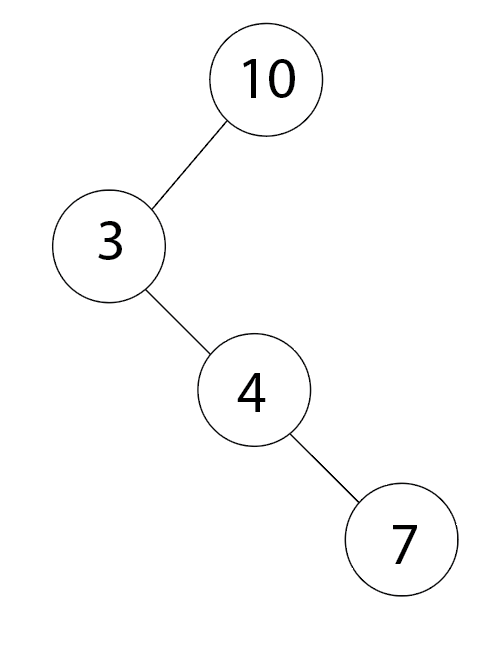


1. succesor(S, x):

Complexitatea pe timp este O(n)

Worst-case: succesorul rădăcinii unui arbore care are un număr de nivele egal cu numărul nodurilor -1, iar toate nodurile au fii doar in stanga cu excepția rădăcini, care are un fiu în dreapta.

(Ex: succesorul lui 3 este 4)

1. predecesor(S, x):

Complexitatea pe timp este O(n)

Worst-case: succesorul rădăcinii unui arbore care are un număr de nivele egal cu numărul nodurilor -1, iar toate nodurile au fii doar în dreapta cu excepția rădăcinii, care are un fiu în stânga.

(Ex: Predecesorul lui 10 este 7)

1. k\_element(x):

Complexitatea pe timp este O(n)

Worst-case: atunci cand dorim sa returnam indicele unui nod care este frunza, iar arborele are numărul de nivele egal cu numărul nodurilor -1.

1. cardinal(S, x):

Complexitatea pe timp este O(n)

Worst-case este și best-case, deoarece trebuie neapărat să parcurgă toate nodurile.

1. este\_în(S, x):

Complexitatea pe timp este O(n)

Worst-case: căutarea frunzei a unui arbore care are un număr de nivele egal cu numărul de noduri -1.

**2. Motivatia structurii folosite (Binary Search Tree)**

Am folosit arborele binar de căutare, deoarece pentru anumite funcții precum succesor și predecesor are complexitate mai buna, iar per total BST-ul are o

**3. Avantaje si dezavantaje**

1. Dezavantaje:

-fata de stiva/coada, BST-ul nu are inserare/stergere la fel de rapidă

-fata de min heap, funcția de min este O(n) fata de O(1)

1. Avantaje:

-fata de Heap, afișarea elementelor în ordine este mai eficienta

-nu se folosesc vectori, ceea ce ne permite sa alocam memoria pe parcursul inserarii elementelor

-

**4. Descrierea modului de testare:**

Fișierele de intrare:

1. Ordine crescătoare de la 1 la 40
2. Ordine descrescătoare de la 1000 la 1
3. Random de la 1 la 10000
4. De la 1 la 10000 destinate predecesorului si succesorului
5. Random de la 1 la 1000000

Testele au fost generate pentru a maximiza timpul fiecărei funcții. Au fost realizate folosind biblioteca chrono care a permis preluarea a doua puncte în timp înainte și după folosirea funcțiilor. Urmatorii timpi de rulare reprezinta worst- case al tuturor numerelor din fișiere.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| nr. sec | exemplu1 | exemplu2 | exemplu3 | exemplu4 | exemplu5 |
| inserare | 0.016 | 0.017 | 0.015 | 0.096 | 2.699 |
| cardinal | 0 | 0 | 0 | 0.000955 | 0.000955 |
| max | 0 | 0 | 0 | 0.000964 | 0.00032 |
| min | 0 | 0 | 0 | 0 | 0.000964 |
| este în | 0 | 0.000994 | 0.001073 | 0.001044 | 0.005984 |

Pentru funcția de inserare a fost testat și un fișier randomized cu 10 000 000 de numere si a durat 19 minute și 31 de secunde.

**5. Sales pitch:**

Grupa noastra este formata din Neagu Marian-Madalin, Ignat Eduardo și Pîslaru Alin, suntem în seria 14, grupa 144, iar impreuna am realizat un BST(Binary Search Tree).

Programul reuseste sa faca operatii pe multimi cu date mari într-un timp scurt, ceea ce îl determina sa scoata o complexitate optimă.

Spre deosebire de alte programe, precum Heap, afișarea **ordonată** a elementelor

este mai eficienta și nu se folosesc vectori, ceea ce ne permite sa alocam memoria

pe parcursul inserarii elementelor.