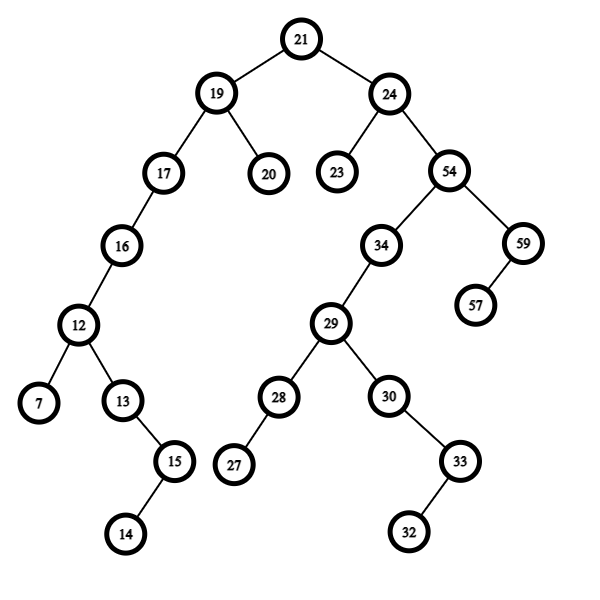
***Arbori binari de căutare***

Definiție: Se numește arbore binar de căutare un arbore binar în care fiecare nod are o cheie unică de identificare care respectă următoarele condiții:

* pentru orice subarbore, cheile nodurilor din subarborele stâng sunt mai mici decât cheia rădăcinii;
* pentru orice subarbore, cheile nodurilor din subarborele drept sunt mai mari decât cheia rădăcinii.

Arborii de căutare binară permit căutarea rapidă, adăugarea și eliminarea elementelor de date. Deoarece nodurile dintr-un ABC sunt așezate astfel încât fiecare comparație sări peste aproximativ jumătate din arborele rămas, performanța căutării este proporțională cu cea a logaritmului binar. ABC-urile au fost concepute în anii 1960 pentru problema stocării eficiente a datelor etichetate și sunt atribuite lui Conway Berners-Lee și David Wheeler.

Performanța unui arbore de căutare binar depinde de ordinea de inserare a nodurilor în arbore, deoarece inserțiile arbitrare pot duce la degenerare; mai multe variante ale arborelui de căutare binar pot fi construite cu performanță garantată în cel mai rău caz. Operațiile de bază includ: căutare, parcurgere, inserare și ștergere. ABC-urile cu complexități garantate în cel mai rău caz funcționează mai bine decât o matrice nesortată, care ar necesita timp de căutare liniar.



Vom exemplifica pe arborele de căutare următor:

| ***Documentație funcții:*** | |
| --- | --- |
| Nodul 31 a fost inserat. | **Funcția de adăugare a valorilor din opțiuni:**  Se disting două cazuri:   * Elementul apare deja în arbore, caz în care se incrementează frecvența elementului și se afișează mesajul specific * Elementul nu apare în arbore, caz în care se crează un nou nod care se adaugă la arbore, iar mai apoi se afișează mesajul specific   La revenirea în funcția main, se afișează arborele în inordine |
| Pe exemplul dat urmează să se mai citească noduri din fișier. | **Funcția de adăugare a nodurilor arborelui inițial:**  Se disting două cazuri:   * Elementul apare deja în arbore, caz în care se incrementează frecvența elementului și se afișează mesajul specific * Elementul nu apare în arbore, caz în care se crează un nou nod care se adaugă la arbore, iar mai apoi se afișează mesajul specific |
| Nodul 7 a fost șters. | **Funcția de ștergere a unui nod:**  Pentru a șterge un nod Z, trebuie sa avem în vedere 3 cazuri:   * *Nodul Z este frunza*: Se reține valoarea într-o variabila aux, nodul Z devine invalid și se eliberează memoria din aux; * *Nodul Z are un singur fiu:* Fiul lui Z devine fiul tatălui lui Z și se eliberează memoria pentru Z. Se folosesc cazuri separate pentru tipul de fiu (dreapta, stanga). * *Nodul Z are 2 fii*: Se găsește cel mai mic nod din subarborele drept al nodului de șters sau se găsește cel mai mare nod din subarborele stâng al nodului Z, pe acest nod îl vom numi succesorul în ordine. Se schimbă valoarea nodului de șters cu valoarea succesorului în ordine. Se șterge succesorul în ordine din arbore recursiv. Valoarea succesorului în ordine este mai mare decât oricare altă valoare din subarborele stâng și mai mică decât orice valoare din subarborele drept. |
| Nodul 33 a fost șters. |
| Nodul 54 a fost șters. |
| Nodul 33 apare în arbore. | **Funcția de verificarea a apariției unui nod în arbore:**  Începe căutarea de la rădăcina arborelui și comparăm valoarea căutată cu valoarea rădăcinii. Dacă valoarea căutată este mai mică decât valoarea rădăcinii, ne deplasăm în subarborele stâng al rădăcinii, altfel, dacă este mai mare decat valoarea rădăcinii, ne deplasăm în subarborele drept. Continuăm să ne deplasăm în subarborii corespunzători în funcție de valoarea căutată, până când găsim valoarea căutată sau până când ajungem într-un nod invalid. În funcție de apariția nodului în arbore, afișăm cu format un mesaj specific (în cazul în care valoarea nu apare, mesajul e afișat cu format). |
| Nodul 33 apare o singură dată în arbore. | **Funcția ce afișează frecvența elementului:**  Începem parcurgerea din rădăcina și comparăm valoarea căutată cu valoarea rădăcinii. Dacă valoarea căutată este egală cu valoarea rădăcinii, se afișează numărul de apariții ale elementului. Dacă valoarea căutată este mai mică decât valoarea rădăcinii, ne deplasăm în subarborele stâng, altfel, dacă este mai mare, în subarborele drept. Continuăm astfel până când ajungem la un nod invalid. În cazul în care nu găsim valoarea în arbore, afișăm (cu format) un mesaj corespunzător. |
|  | **Funcția de afișare a unui mesaj specific:**  Primește ca parametri un șir de caractere și o valoare care indica tipul de afișare. În funcție de valoarea celui de-al doilea parametru, apar două cazuri:   * Cazul 1: Operația solicitată produce eroare pentru că arborele nu are niciun nod sau valoarea căutată nu apare în arbore și se afișează un text formatat * Cazul 2: Operația solicitată nu produce erori și se afișează un text neformatat   În finalul funcției, în cazul în care s-a afișat o eroare, se reseteaza culoarea fontului la cea inițială |
|  |
| Nodurile se afișează în următoarea ordine: 7, 12, 13, 14, 15, 16, 17, 19, 20, 21, 23, 24, 27, 28, 29, 30, 32, 33, 34, 54, 57, 59. | **Funcția de afișarea a valorilor arborelui în ordine crescătoare:**  Se disting două cazuri:   * Arborele este nul, când se afișează un mesaj specific, cu font formatat. * Arborele nu este nul; funcția afișează șirul sortat, folosind o altă funcție, care parcurge arborele în inordine și afișează valorile memorate în acestea. De asemenea, se utilizează o culoare specifică pentru afișarea acestora.. |
| Nodurile se afișează în următoarea ordine:  21, 19, 17, 16, 12, 7, 13, 15, 14, 20, 23, 24, 54, 34, 29, 28, 27, 30, 33, 32, 59, 57. | **Funcția de afișare în preordine a arborelui.**  Funcția afișează valorile din arbore astfel:  **RSD** – se parcurge mai întâi rădăcina (**R**), apoi subarborele stâng (**S**), apoi subarborele drept (**D**). |
| Nodurile se afișează în următoarea ordine: 7, 12, 13, 14, 15, 16, 17, 19, 20, 21, 23, 24, 27, 28, 29, 30, 32, 33, 34, 54, 57, 59. | **Funcția de afișare în inordine a arborelui.**  Funcția afișează valorile din arbore astfel:  **SRD** – se parcurge mai întâi subarborele stâng (**S**), apoi rădăcina (**R**), apoi subarborele drept (**D**). |
| Nodurile se afișează în următoarea ordine:  7, 14, 15, 13, 12, 16, 17, 20, 19, 27, 28, 32, 33, 30, 29, 34, 57, 59, 54, 24, 23, 21. | **Funcția de afișare în postordine a arborelui.**  Funcția afișează valorile din arbore astfel:  **SDR** – se parcurge mai întâi subarborele stâng (**S**), apoi subarborele drept (**D**), apoi rădăcina (**R**). |
|  | **Funcția ce setează culoarea fontului din consolă:**  Funcția primește un parametru, în funcție de care se realizează diferite operații   1. Dacă parametrul ia valoarea 1, culoarea fontului devine roșu 2. Dacă parametrul ia valoarea 2, culoarea fontului devine alb (default) |

Cod

| #include <bits/stdc++.h>  #include <windows.h>  ​  using namespace std;  ​  struct Nod  {  int info;  int frq;  Nod \*st, \*dr;  }\*r;  ​  struct AB  {  HANDLE p = GetStdHandle (STD\_OUTPUT\_HANDLE);  void Set\_Culoare (int caz)  {  **if**(caz == 1)  SetConsoleTextAttribute (p , 4);  **else** **if**(caz == 2)  SetConsoleTextAttribute (p , 7);  **else**  **return**;  }  void Afis (**const** char str[], int caz)  {  **if**(caz == 1)  {  Set\_Culoare(1);  cout << "EROARE: ";  }  cout << str;  **if**(caz == 1)  Set\_Culoare (2);  }  void Adauga(Nod \*&p, int X)  {  **if**(p)  {  **if**(X < p->info)  Adauga(p -> st, X);  **else** **if**(X > p->info)  Adauga(p -> dr, X);  **else**  {  ++p -> frq;  Afis("Elemetul exită în arbore\n", 1);  **return**;  }  }  **else**  {  p = **new** Nod;  p -> info = X;  p -> st = NULL;  p -> dr = NULL;  p -> frq = 1;  Afis("Element adăugat cu succes\n", 2);  **return**;  }  }  void AdaugaI(Nod \*&p, int X)  {  **if**(p)  {  **if**(X < p->info)  AdaugaI(p -> st, X);  **else** **if**(X > p->info)  AdaugaI(p -> dr, X);  **else**  {  ++p -> frq;  **return**;  }  }  **else**  {  p = **new** Nod;  p -> info = X;  p -> st = NULL;  p -> dr = NULL;  p -> frq = 1;  **return**;  }  }  void cmd (Nod \*&c, Nod \*&f)  {  Nod \*aux;  **if**(f -> dr)  cmd(c, f -> dr);  **else**  {  c -> info = f -> info;  aux = f;  f = f->st;  delete aux;  }  ​  }  void Sterge (Nod \*r, int X)  {  **if**(r == NULL)  {  Afis("Arbore nul\n" , 1);  **return**;  }  sterg(r , X);  }  void sterg(Nod \*& c, int k)  {  Nod \*aux;  **if**(c)  {  **if**(c -> info == k)  **if**(c -> st==0 && c -> dr == 0) /// daca e nod terminal  {  delete c;  c = 0;  }  **else** **if**(c -> st == 0) /// are numai subordonat drept  {  aux = c -> dr;  delete c;  c = aux;  }  **else** **if**(c -> dr == 0) /// are numai subordonat drept  {  aux = c -> st;  delete c;  c = aux;  }  **else**  cmd(c, c -> st); /// are ambii subordonati  **else** **if**(c -> info < k)  sterg(c -> dr, k);  **else**  sterg(c -> st, k);  }  **else**  Afis("Valoarea de șters nu se găseste în arbore\n", 1);  ​  }  void Frecventa (int X)  {  Nod \*q = r;  **while**(q != NULL)  {  **if**(q -> info == X)  {  Afis("Elementul apare de " , 2);  cout << q -> frq << " ori\n";  **return**;  }  **else** **if**(X < q -> info)  q = q -> st;  **else**  q = q -> dr;  }  Afis("Elementul nu apare în arbore\n" , 1);  **return**;  }  void Apare (int X)  {  Nod \*q = r;  **while**(q != NULL)  {  **if**(q -> info == X)  {  Afis ("Elementul apare în arbore\n" , 2);  **return**;  }  **else** **if**(X < q -> info)  q = q -> st;  **else**  q = q -> dr;  }  Afis("Elementul nu apare în arbore\n" , 1);  **return**;  }  void Afisare (Nod \*r, int caz)  {  **if**(r == NULL)  {  Afis("Arbore nul\n", 1);  **return**;  }  **if**(caz == 1)  AfisarePO(r);  **else** **if**(caz == 2)  AfisareIO(r);  **else**  AfisarePSO(r);  }  void AfisarePO (Nod \*r)  {  **if**(r == NULL)  **return**;  cout << r-> info << " ";  AfisarePO (r -> st);  AfisarePO (r -> dr);  }  void AfisareIO (Nod \*r)  {  **if**(r == NULL)  **return**;  AfisareIO (r -> st);  cout << r-> info << " ";  AfisareIO (r -> dr);  }  void AfisarePSO (Nod \*r)  {  **if**(r == NULL)  **return**;  AfisarePSO (r -> st);  AfisarePSO (r -> dr);  cout << r-> info << " ";  }  void Sortare (Nod \*r)  {  **if** (r == NULL)  **return**;  Sortare(r -> st);  cout << r -> info << " ";  Sortare (r -> dr);  }  void AfisareSortata (Nod \*r)  {  **if**(r == NULL)  {  Afis("Graful este nul\n" , 1);  **return**;  }  SetConsoleTextAttribute (p , 14);  cout << "Șirul valorilor sortate este:\n";  Sortare(r);  SetConsoleTextAttribute (p , 7);  cout << "\n";  }  } T;  ​  void prezentaison()  {  cout << "Proiect realizat de:\n";  cout << "Comșa Mirela Elena\n";  cout << "Maxim Sabin\n";  cout << "Miron Victor\n";  cout << "Onuț Alexandru Ioan\n";  cout << "Rîșca Ioan\n";  cout << "Poftă bună!!!!!!!\n";  cout << "\n";  }  ​  int main()  {  SetConsoleOutputCP(65001);  ifstream fin ("test.in");  prezentaison();  int optiune, X;  **while**(fin >> X)  T.AdaugaI(r, X);  HANDLE p = GetStdHandle (STD\_OUTPUT\_HANDLE);  SetConsoleTextAttribute (p, 100);  cout << "Tema: Arbori Binari de Căutare\n";  SetConsoleTextAttribute (p, 7);  **do**  {  cout << "1 - Adăugare\n";  cout << "2 - Ștergere\n";  cout << "3 - Apariție\n";  cout << "4 - Frecvență\n";  cout << "5 - Afișare\_Preordine\n";  cout << "6 - Afișare\_Inordine\n";  cout << "7 - Afișare\_Postordine\n";  cout << "8 - Afișare\_Sortata\n";  cout << "0 - exit\n";  cout << "\n";  cout << "Introduceți opțiunea: \n";  cin >> optiune;  cout << '\n';  **switch**(optiune)  {  **case** 1:  {  cout << "Introduceți valoarea de inserat:\n";  cin >> X;  T.Adauga(r, X);  T.AfisareIO(r);  **break**;  }  **case** 2:  {  cout << "Introduceți valoarea de șters:\n";  cin >> X;  T.Sterge(r, X);  T.AfisareIO(r);  **break**;  }  **case** 3:  {  cout << "Introduceți valoarea:\n";  cin >> X;  T.Apare(X);  **break**;  }  **case** 4:  {  cout << "Introduceți valoarea:\n";  cin >> X;  T.Frecventa(X);  **break**;  }  **case** 5:  T.Afisare(r, 1);  **break**;  **case** 6:  T.Afisare(r, 2);  **break**;  **case** 7:  T.Afisare(r, 3);  **break**;  **case** 8:  T.AfisareSortata(r);  **break**;  **case** 0:  ;  **break**;  **default** :  cout << "Opțiune invalidă!\n";  }  cout << "\n";  ​  }  **while**(optiune);  **return** 0;  } |
| --- |

Bibliografie:

<https://en.wikipedia.org/wiki/Binary_search_tree>

Highlight la cod: <https://www.syntax-highlighter.dev/>