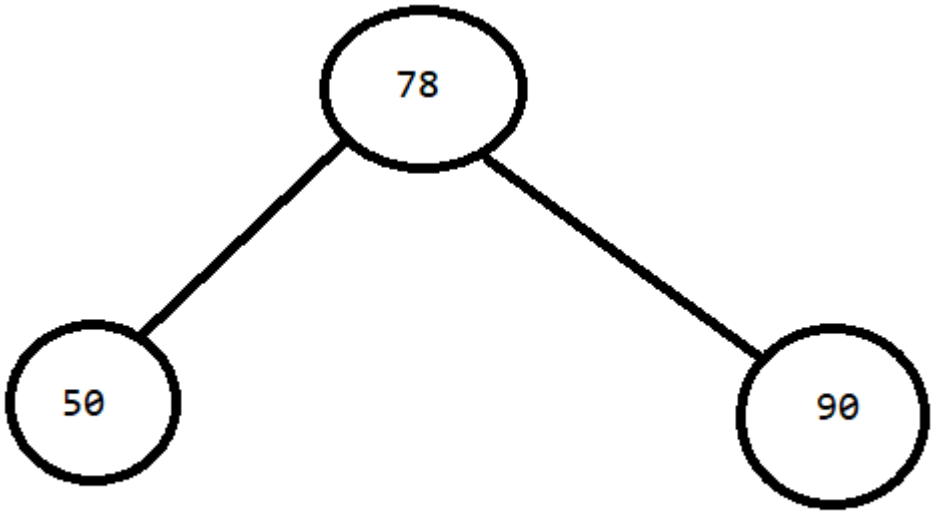
arbore binar de căutare ABC

Are proprietatea că, pentru fiecare nod, cheia din succesorul stâng este mai mică decât cheia din nod, iar cheia din succesorul drept este mai mare decât cheia din nod.



- Pentru orice nod, subarborele stâng conţine noduri cu valori mai mici ca ale cheii, iar subarborele drept conţine noduri cu valori mai mari ca ale cheii.
- Într-un arbore binar de căutare nu există două noduri cu aceeaşi valoare a cheii. Dacă este necesar se poate completa nodul cu un cîmp ce reține frecvența de apariție a cheii.

Algoritmi:

1. creare
2. parcurgere
3. actualizare
   - 3.1. inserarea unui nod
   - 3.2. ștergerea unui nod
4. căutarea unei valori
5. determinarea minimului/maximului

……

```
#include <iostream>
using namespace std;
struct nod
{
int info;
nod *st, *dr;
};

void Creare (nod *&r, int val)
{
  if(r!=NULL)
  {
    if(val<r->info)
        Creare(r->st,val);
    else
        if(val>r->info)
             Creare(r->dr,val);
        else
            return;//cout<<”Valoarea exista!\n”;
  }
  else
  {
    r=new nod;
    r->info=val;
    r->st=NULL;
    r->dr=NULL;
  }
}

void RSD(nod *r)
{if(r!=NULL)
  {
    cout<<r->info<<' ';
    RSD(r->st);
    RSD(r->dr);
  }
}

void SRD(nod *r)
{
    if(r!=NULL)
    {
      SRD(r->st);
```

```
      cout<<r->info<<' ';
      SRD(r->dr);
    }
}

nod *Cautare(nod *r, int x)
{
    if(r!=NULL)
    {
      if(r->info==x)
         return r;
      else
         if(r->info>x)
            return Cautare(r->st,x);
         else
            return Cautare(r->dr,x);
    }
    else return NULL;
}

int Cautare1(nod *r, int x)
{
    if(r!=NULL)
    {
      if(r->info==x)
         return 1;
      else
         if(r->info>x)
            return Cautare1(r->st,x);
         else
            return Cautare1(r->dr,x);
    }
   else return 0;
}

int Val_Min(nod *r)
{
  if(r->st!=NULL)
     return Val_Min(r->st);
  else
     return r->info;
}

int Val_Max(nod *r)
{
  if(r->dr!=NULL)
     return Val_Max(r->dr);
```

```
  else
     return r->info;
}

void Inserare(nod *&r, int x)
{
    if(r!=NULL)
    {
        if(r->info==x)
            return ;// cout<<"Valoarea exista in arbore!";
        else
            if(r->info>x)
                Inserare(r->st,x);
            else
                Inserare(r->dr,x);
    }
    else
    {
        r=new nod;
        r->info=x;
        r->st=NULL;
        r->dr=NULL;
    }
}

void Sterge(nod *&p,nod *&cmax)
{
    nod *q;
    if(cmax->dr!=NULL) Sterge(p, cmax->dr);
    else
    {
        p->info=cmax->info;
        q=cmax;
        cmax=cmax->st;
        delete q;
    }
}

void Stergere(nod *&r, int k)
{
    nod *q;
    if(r!=NULL)
    {
        if(r->info==k)
        {
            if(r->st==NULL&&r->dr==NULL)
            {
```

```
                delete r;
                r=NULL;
            }
            else
                if(r->st==NULL)
                {
                    q=r->dr;
                    delete r;
                    r=q;
                }
                else
                    if(r->dr==NULL)
                    {
                        q=r->st;
                        delete r;
                        r=q;
                    }
                    else
                        Sterge(r,r->st);
        }
        else
            if(r->info>k)
                Stergere(r->st,k);
            else
                Stergere(r->dr,k);
    }
    else return;//cout<<"Cheia NU exista"<<endl;
}
int main()
{
    int x;
    nod *r=NULL;
    int v[]={8,3,5,7,2,6,9};
    for(int i=0;i<=6;i++)
        Creare(r,v[i]);
    RSD(r);
    cout<<endl;
    cout<<Val_Max(r)<<endl;
    Stergere(r,x);
    RSD(r);

    return 0;
}
```

```
……..
#include <iostream>
#include <fstream>
using namespace std;
ifstream fin("date.in");
ofstream fout("date.out");

struct nod
{
    int info;
    nod *st,*dr;
};

int h,k;

void Creare (nod *&r, int x)
{
    if(r!=NULL)
    {
        if(x<r->info)
            Creare(r->st,x);
        else if(val>r->info)
            Creare(r->dr,x);
        else
            return;//cout<<"Valoarea exista!\n";
    }
    else
    {
        r=new nod;
        r->info=x;
        r->st=NULL;
        r->dr=NULL;
    }
}
void SRD(nod *r)
{
    if(r!=NULL)
    {
        SRD(r->st);
        cout<<r->info<<" ";
        SRD(r->dr);
    }
}
void SDR(nod *r)
{
    if(r!=NULL)
```

```
    {
        SDR(r->st);
        SDR(r->dr);
        cout<<r->info<<" ";
    }
}
int Max (int a,int b)
{
    if(a>b)
        return a;
    else
        return b;
}
int Inalt(nod *r)
{
    if(r==NULL)
        return -1;
    else
        return 1+Max(Inalt(r->st),Inalt(r->dr));
}
int Cautare(nod *r, int x)
{
    if(r!=NULL)
    {
        if(r->info==x)
            return 1;
        else if(r->info>x)
            return Cautare(r->st,x);
        else
            return Cautare(r->dr,x);
    }
    else
        return 0;
}
void Frunze_Niv(nod *r)
{
    if(r!=NULL)
    {
        Frunze_Niv(r->st);
        if(r->st==NULL&r->dr==NULL)
            cout<<r->info<<" ";
        Frunze_Niv(r->dr);
    }
}
int Val_Min(nod *r)
{
    if(r->st!=NULL)
```

```
        return Val_Min(r->st);
    else
        return r->info;
}
int Val_Max(nod *r)
{
    if(r->dr!=NULL)
        return Val_Max(r->dr);
    else
        return r->info;
}
void Inserare(nod *&r,int x)
{
    if(r!=NULL)
    {
        if(r->info==x)
            cout<<"Valoarea exista in arbore!";
        else if(r->info>x)
            Inserare(r->st,x);
        else
            Inserare(r->dr,x);
    }
    else
    {
        r=new nod;
        r->info=x;
        r->st=NULL;
        r->dr=NULL;
    }
}
void StergeNod(nod *&p,nod *&cmax)
{
    nod *q;
    if(cmax->dr!=NULL)
        StergeNod(p,cmax->dr);
    else
    {
        p->info=cmax->info;
        q=cmax;
        cmax=cmax->st;
        delete q;
    }
}
void Stergere(nod *&r, int k)
{
    nod *q;
    if(r!=NULL)
```

```
    {
        if(r->info==k)
            if(r->st==NULL&&r->dr==NULL)
            {
                delete r;
                r=NULL;
            }
            else if(r->st==NULL)
            {
                q=r->dr;
                delete r;
                r=q;
            }
            else if(r->dr==NULL)
            {
                q=r->st;
                delete r;
                r=q;
            }
            else
                StergeNod(r,r->st);
        else if(r->info>k)
            Stergere(r->st,k);
        else
            Stergere(r->dr,k);
    }
    else
        cout<<"Cheia NU exista"<<endl;
}
int main()
{
    int x,cmin,cmax;
    nod *r;
    r=NULL;
    cin>>x;
    while(x!=0)
    {
        Creare(r,x);
        cin>>val;
    }
    SRD(r);
    SDR(r);
    h=Inalt(r);
    cout<<h;
    cout<<"x=";  cin>>x;
    if(Cautare(r,x)==1)
        cout<<"s-a gasit!";
```

```
    else
        cout<<"nu s-a gasit!";
    Frunze_Niv(r);
    cmin=Val_Min(r);
    cmax=Val_Max(r);
    cout<<cmin<<" "<<cmax;
    Inserare(r,52);
    SDR(r);
    Stergere(r,52);
    SDR(r);
    Stergere(r,r->info);
     SDR(r);
    return 0;
}
```