一、树的介绍

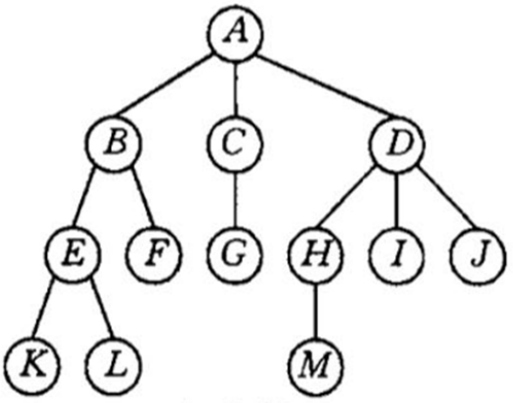
1、树的定义

树是一种数据结构，它是由n（n>=0）个有限节点组成一个具有层次关系的集合。

把它叫做“树”是因为它看起来像一棵倒挂的树，也就是说它是根朝上，而叶朝下的。

空树

只有根结点的树



一般的树

它具有以下的特点：

1.每个节点有零个或多个子节点;

2.没有父节点的节点称为根节点;

3.每一个非根节点有且仅有一个父节点;

4.除了根节点以外，每个子节点可以分为多个不相交的子树。

2、树的基本术语

若一个节点有子树，那么该节点称为子树根节点的“双亲“，子树的根是该节点的“孩子”。有相同双亲的节点互为“兄弟节点”。一个节点的所有子树上的任何节点都是该节点的后裔。从根节点到某个节点的路径上的所有节点都是该节点的祖先。

节点的度：节点拥有的子树的数目。

叶子：度为零的节点。

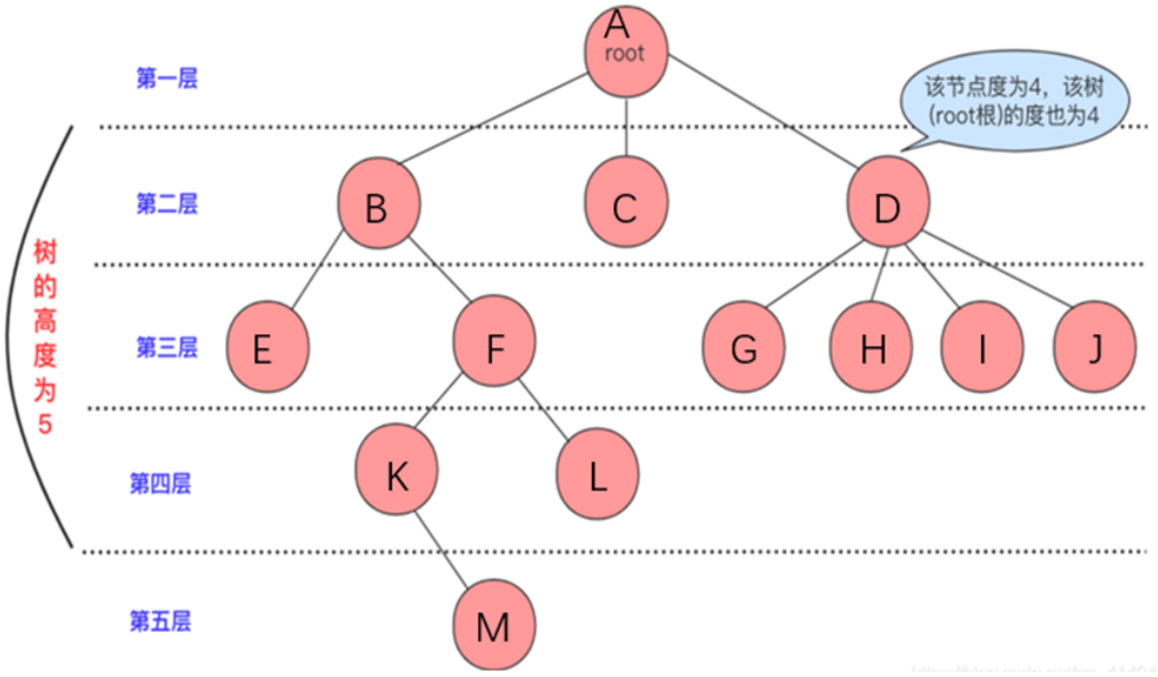
分支节点：度不为零的节点。

树的度：树中节点的最大的度。

层次：根节点的层次为1，其余节点的层次等于该节点的双亲节点加1。

树的高度：树中节点的最大层次。

森林：0个或多个不相交的树组成。对森林加上一个跟，森林即成为树；删去根，树即成为森林。



树的基本性质：

1.树的节点数=所有节点度数+1。

2.度为m的树第i层最多有m^(i-1)个节点。(i>=1)

第 i 层上的结点数 = 第 i-1 层结点的度数，也就是m*(第 i-1 层的结点个数) = m*m*(第 i- 2 层的结点个数) = ... = m^(i-1)*（第1层的结 点数）= m^(i-1)（共乘了i次）。

3.高度为h的m叉树最多(m^h-1)/(m-1)个节点。

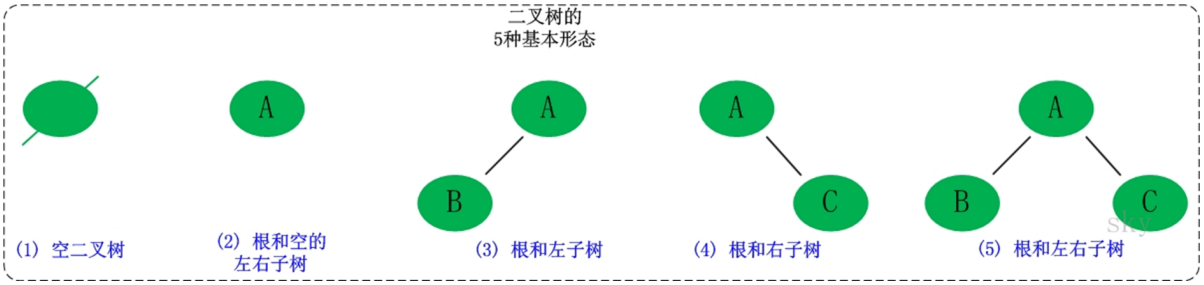
由树的上个性质可知，m叉树第 i 层至多有m^(i-1)个结点。那么高度为h的m叉树的结点数至多：S = m^(h-1) + m^(h-2) + m^(h-3) + ... + m + 1 = (m^h-1)/(m-1)。 该树现在是个满m叉树。

4.n个节点的m叉树最小高度为logm(n(m-1)+1)。

求最小高度也就是每层结点数最多时的高度，即该树是一棵完全m叉树，设其高度为h。
由树的第4个性质，有 n ≤ (m^h-1)/(m-1)，解得 h ≥ logm(n(m-1)+1) 。故h为⌈logm(n(m-1)+1)⌉。

二、 二叉树

二叉树是每个节点最多有两个子树的树结构。它有五种基本形态：二叉树可以是空集；根可以有空的左子树或右子树；或着左、右子树皆为空。



1、二叉树的性质

性质1：二叉树第i层上的节点数目最多为 2^(i-1)。 (i≥1)。

性质2：深度为k的二叉树至多有2^k-1个节点（k>=1）。

性质3：包含n个节点的二叉树的高度至少为log2 (n+1)。

性质4：在任意一颗二叉树中，若终端节点的个数为n0，度为2的节点数为n2，则n0=n2+1。

节点总数为n=n0+n1+n2

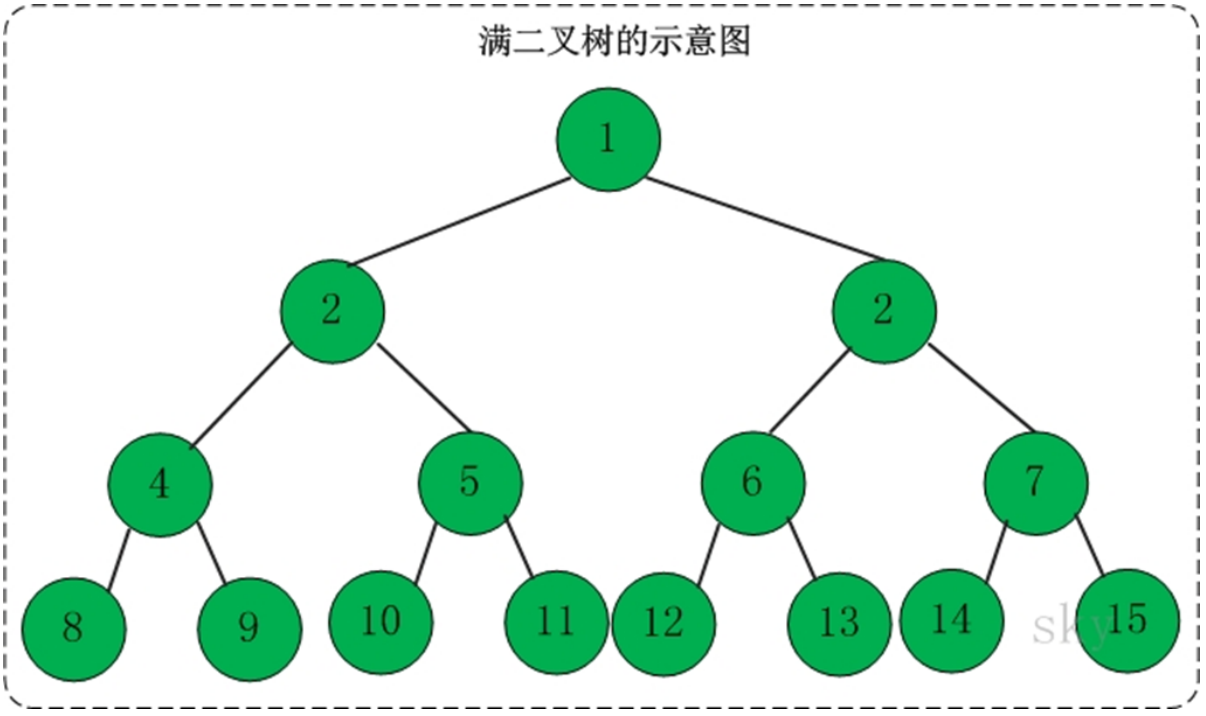
边数T=n-1

度为1的结点产生一个边，度为2的结点产生两个边，叶子结点不产生边，边数T=n1+2*n2

T = n1+2*n2 = n-1= n0+n1+n2-1 --> n0=n2+1

2、 二叉树的种类

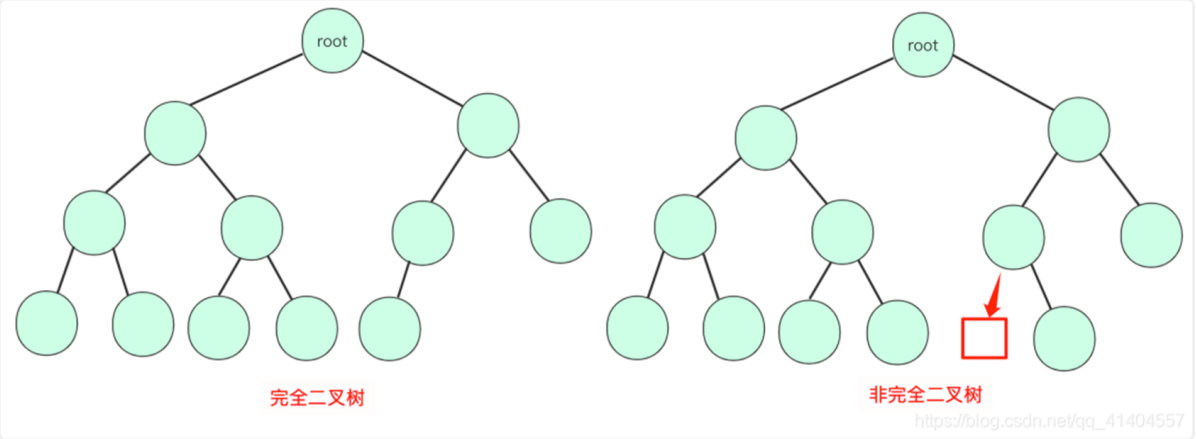
（1）满二叉树 高度为h，并且由2^h –1个结点的二叉树，被称为满二叉树。



(2)完全二叉树

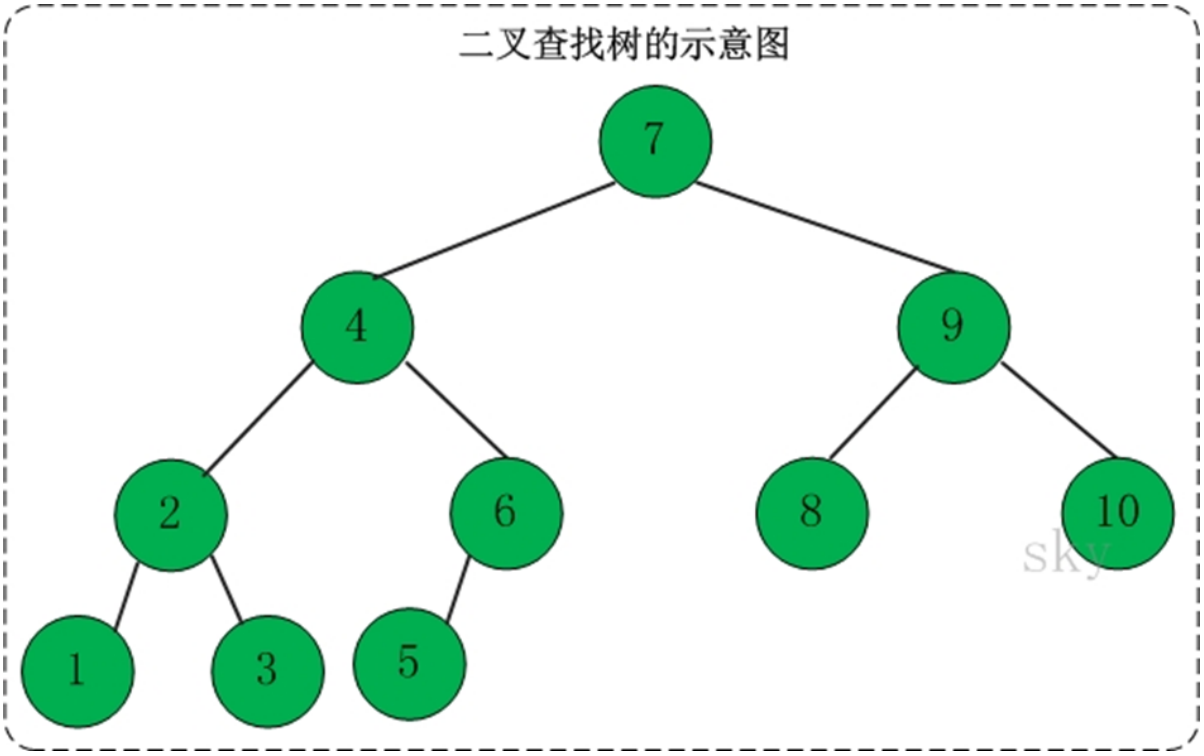
一颗二叉树中，只有最下面两层节点的度可以小于2，并且最下层的叶节点集中在靠左的若干位置上。

叶子节点只能出现在最下层和次下层，且最下层的叶子节点集中在树的左部。显然，一颗满二叉树必定是一颗完全二叉树，而完全二叉树不一定是满二叉树。



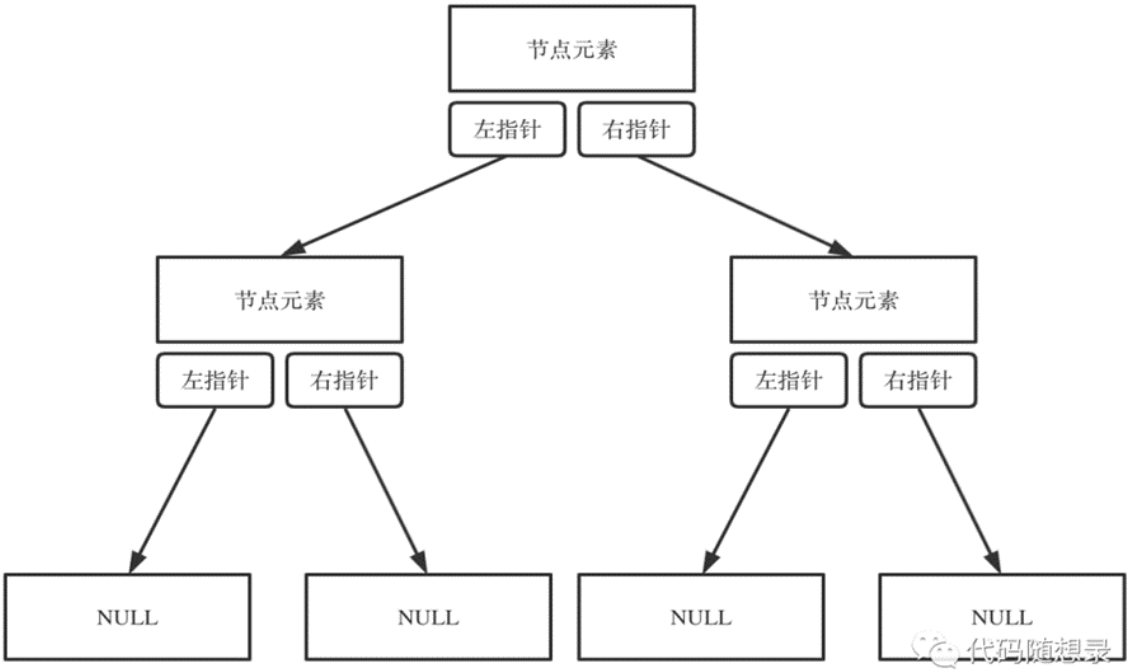
(3) 二叉查找树

二叉查找树（Binary Search Tree）,又被称为二叉搜索树。设x为二叉树中的一个节点，x节点包含关键字Key，节点x的Key值记为Key[x]。如果y是x的左子树中的一个节点，则Key[y]<=Key[x]；如果y是x的有子树的一个节点，则Key[y]>=Key[x]。



3、二叉树的存储方法

(1)链式存储方法



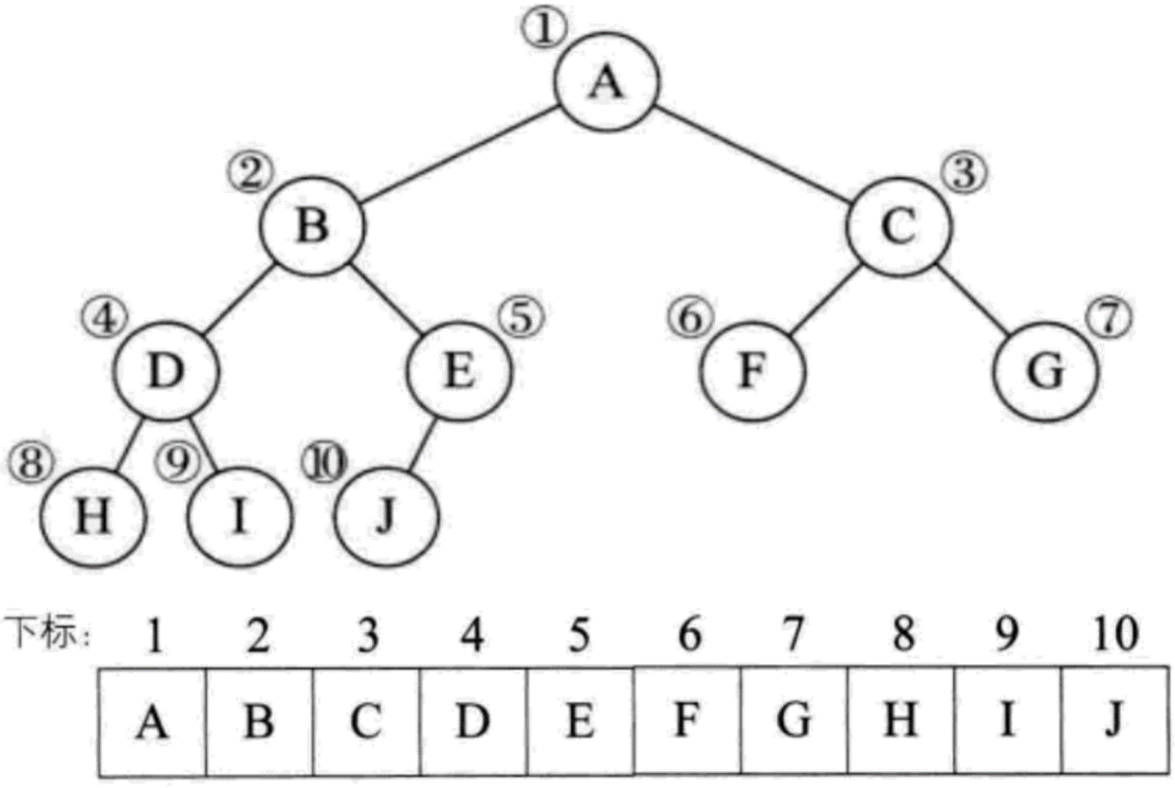

```
struct node
{
    int data;
    int left;
    int right;
    int fa;
}tree[N];
```

例题：P4913 【深基16.例3】二叉树深度

```
#include<bits/stdc++.h>
#define IOS std::ios::sync_with_stdio(false);std::cin.tie(0);std::cout.tie(0)
#define int long long
using namespace std;

const int N = 1e6+10;
const int INF = 1e18;

typedef pair<int,int> PII;

int n,m,k,p[N],l[N],r[N],ans=0;

void dfs(int root,int dep)
{
    if(l[root]==0&&r[root]==0)  //叶子节点
    {
        ans=max(ans,dep);
        return;
    }
    dfs(l[root],dep+1);
    dfs(r[root],dep+1);

}
```

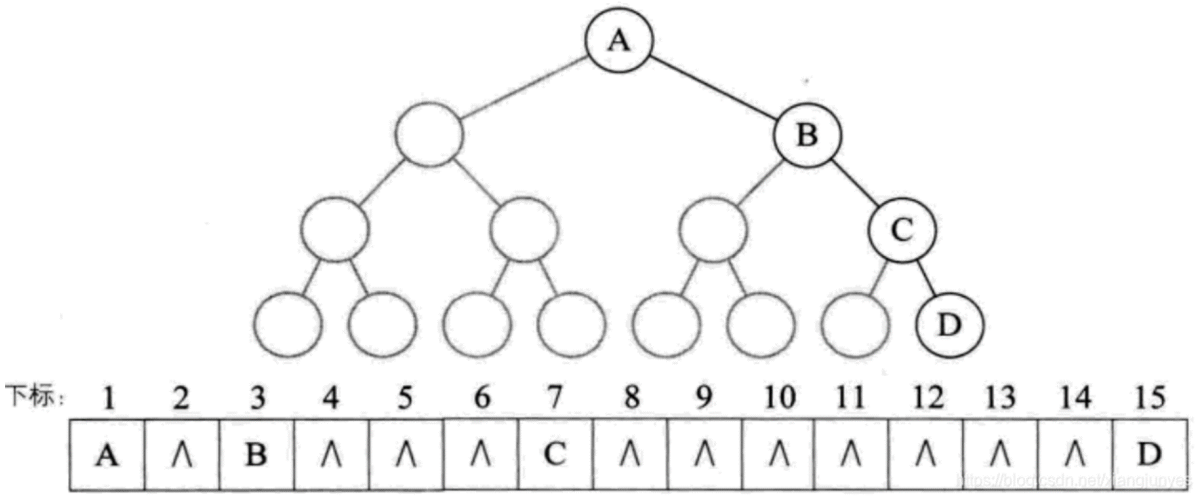
```
void solve()
{
    cin>>n;
    for(int i=1;i<=n;i++)
    {
        int x,y;
        cin>>x>>y;
        p[x]=i;
        p[y]=i;
        l[i]=x;
        r[i]=y;
    }
    dfs(1,1);
    cout<<ans<<"\n";


}

signed main()
{
    IOS;
    solve();
    return 0;
}
```

(2)顺序存储方法

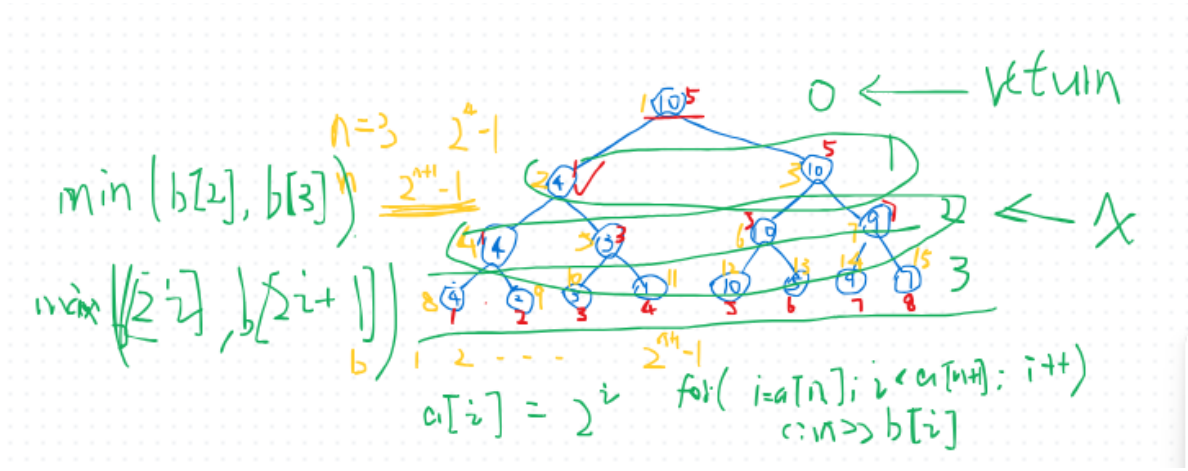


如果数组从下标1开始存，若当前节点为i，那么它的左孩子的节点编号为*2i*，*右孩子的节点编号为*2i+1。

如果数组从下标0开始存，若当前节点为i，那么它的左孩子的节点编号为*2i+1*，*右孩子的节点编号为2i+2*。



题目：洛谷P4715



AC代码：

```
#include<bits/stdc++.h>
#define IOS std::ios::sync_with_stdio(false);std::cin.tie(0);std::cout.tie(0)
#define int long long
using namespace std;

const int N = 1e6+10;
const int INF = 1e18;

typedef pair<int,int> PII;

int n,m,k,a[N],b[N];   //a[i]存储2^i

void build(int x)   //第x层
{
    if(x==0) return;
    for(int i=a[x-1];i<a[x];i++) b[i]=max(b[2*i],b[2*i+1]);
    build(x-1);
}


void solve()
{
    cin>>n;
    a[0]=1;
```

```
    for(int i=1;i<=8;i++) a[i]=a[i-1]*2;

    for(int i=a[n];i<a[n+1];i++) cin>>b[i];

    build(n);

    int t=min(b[2],b[3]);
    for(int i=a[n],j=1;i<a[n+1];i++,j++)
    {
        if(b[i]==t)
        {
            cout<<j<<"\n";
            return;
        }
    }

    return;
}

signed main()
{
    IOS;
    solve();
    return 0;
}
```

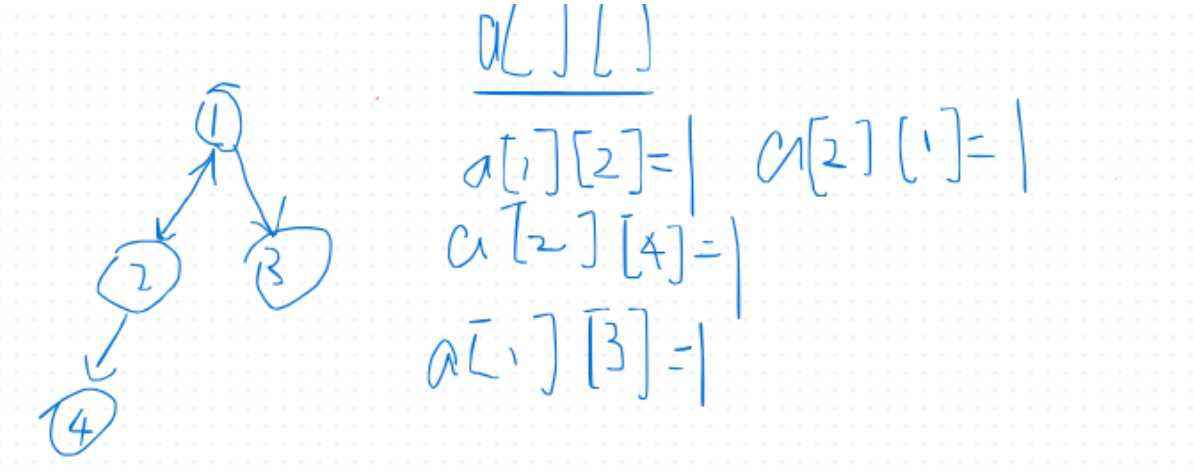
(3)存图

1）邻接矩阵：

用二维数组a[i][j]来存图

如果x到y有一条边，那么a[x][y]=1

例如上图：a[1][2]=1,a[2][3]=1,a[3][4]=1



2）邻接表

数组模拟邻接表：

```
#include<bits/stdc++.h>
#define int long long
```

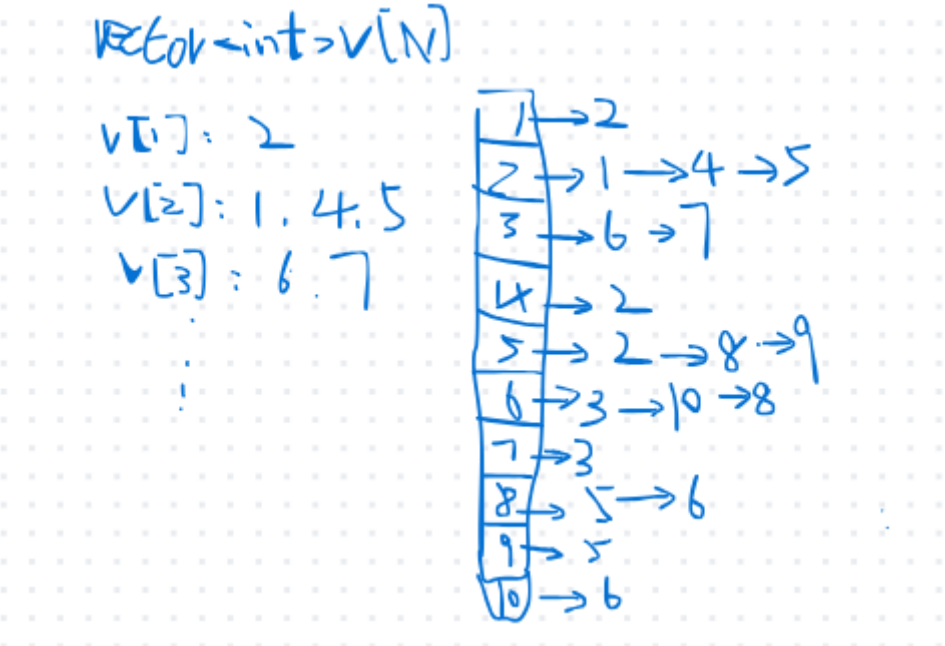
```cpp
using namespace std;

const int N = 1e6+10;
const int INF = 1e18;

typedef pair<int,int> PII;

int n,m;

int e[N],ne[N],h[N],idx;
//idx表示当前的边是第几条边
//h[x]表示节点x目前指向哪一条边
//ne[i]表示第i条边指向哪条边
//e[i]表示指第i条边的终点节点的编号


void add(int x,int y)
{
    e[idx]=y;
    ne[idx]=h[x];
    h[x]=idx++;
}

signed main()
{
    cin>>n>>m;
    memset(h,-1,sizeof h);
    for(int i=1;i<=m;i++)
    {
        int x,y;
        cin>>x>>y;
        add(x,y);
    }
    for(int i=1;i<=n;i++)
    {
        cout<<i<<":  ";
        for(int j=h[i];j!=-1;j=ne[j])
        {
            int k=e[j];
            cout<<k<<" ";
        }
        cout<<"\n";
    }
    return 0;
}
```

vector实现邻接表：

```cpp
#include<bits/stdc++.h>
#define int long long
using namespace std;

const int N = 1e6+10;
```

```
const int INF = 1e18;

typedef pair<int,int> PII;

int n,m;

vector<int>v[N];

signed main()
{
    cin>>n>>m;
    for(int i=1;i<=m;i++)
    {
        int x,y;
        cin>>x>>y;
        v[x].push_back(y);  //x->y有一条边
    }
    for(int i=1;i<=n;i++)
    {
        cout<<i<<":  ";
        for(auto j:v[i]) cout<<j<<" ";
        cout<<"\n";
    }
    return 0;
}
```

例题：[P3884 JLOI2009] 二叉树问题



Ac代码：

```
//#include<bits/stdc++.h>
//#define IOS std::ios::sync_with_stdio(false);std::cin.tie(0);std::cout.tie(0)
//#define int long long
//using namespace std;
//
//const int N = 1e6+10;
```

```
//const int INF = 1e18;
//
//typedef pair<int,int> PII;
//
//int n,m,k,x,y,wid[N],st[N],deep[N],p[N];
//
//vector<int>v[N];
//
//int lca(int x,int y)
//{
//  st[x]=1;
//  while(x!=0)
//  {
//      x=p[x];
//      st[x]=1;
//  }
//
//  while(st[y]!=1)
//  {
//      y=p[y];
//  }
//
//  return y;
//}
//
//void dfs(int root,int fa)
//{
//  deep[root]=deep[fa]+1;
//  p[root]=fa;
//  for(auto j:v[root])
//  {
//      if(j!=fa) dfs(j,root);
//  }
//}
//
//void solve()
//{
//  cin>>n;
//  for(int i=1;i<n;i++)
//  {
//      int a,b;
//      cin>>a>>b;
//      v[a].push_back(b);
//      v[b].push_back(a);
//  }
//  cin>>x>>y;
//  dfs(1,0);
//  int maxdep=1,maxwid=1;
//  for(int i=1;i<=n;i++)
//  {
//      wid[deep[i]]++;
//      maxdep=max(maxdep,deep[i]);
//  }
//  for(int i=1;i<=n;i++) maxwid=max(maxwid,wid[i]);
//  int la=lca(x,y);
```

```
//  cout<<maxdep<<"\n"<<maxwid<<"\n"<<abs(deep[x]-deep[la])*2+abs(deep[y]-
deep[la])<<"\n";
//  return;
//}
//
//signed main()
//{
//  IOS;
//  solve();
//  return 0;
//}



#include<bits/stdc++.h>
#define IOS std::ios::sync_with_stdio(false);std::cin.tie(0);std::cout.tie(0)
#define int long long
using namespace std;

const int N = 1e6+10;
const int INF = 1e18;

typedef pair<int,int> PII;

int n,m,k,x,y,wid[N],st[N],deep[N],p[N];

int e[N],ne[N],h[N],idx;

//idx表示当前的边是第几条边
//h[x]表示节点x目前指向哪一条边
//ne[i]表示第i条边指向哪条边
//e[i]表示指第i条边的终点节点的编号

void add(int a,int b)
{
    e[idx]=b;
    ne[idx]=h[a]++;
    h[a]=idx++;
}

int lca(int x,int y)
{
    st[x]=1;
    while(x!=0)
    {
        x=p[x];
        st[x]=1;
    }

    while(st[y]!=1)
    {
        y=p[y];
    }

    return y;
}
```

```
void dfs(int root,int fa)
{
    deep[root]=deep[fa]+1;
    p[root]=fa;
    for(int i=h[root];i!=-1;i=ne[i])
    {
        int j=e[i];
        if(j!=fa) dfs(j,root);
    }
}

void solve()
{
    cin>>n;
    memset(h,-1,sizeof h);
    for(int i=1;i<n;i++)
    {
        int a,b;
        cin>>a>>b;
        add(a,b);
        add(b,a);
    }
    cin>>x>>y;
    dfs(1,0);
    int maxdep=1,maxwid=1;
    for(int i=1;i<=n;i++)
    {
        wid[deep[i]]++;
        maxdep=max(maxdep,deep[i]);
    }
    for(int i=1;i<=n;i++) maxwid=max(maxwid,wid[i]);
    int la=lca(x,y);
    cout<<maxdep<<"\n"<<maxwid<<"\n"<<abs(deep[x]-deep[la])*2+abs(deep[y]-
deep[la])<<"\n";
    return;
}

signed main()
{
    IOS;
    solve();
    return 0;
}
```

4、二叉树的遍历

深度优先搜索：

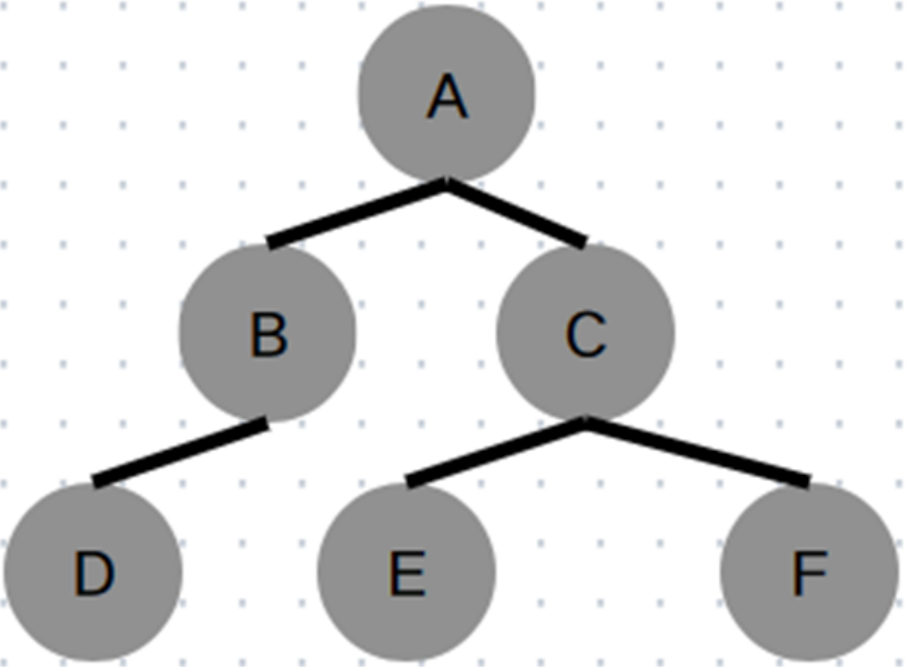
（1）先(根)序遍历（根左右） ABDCEF

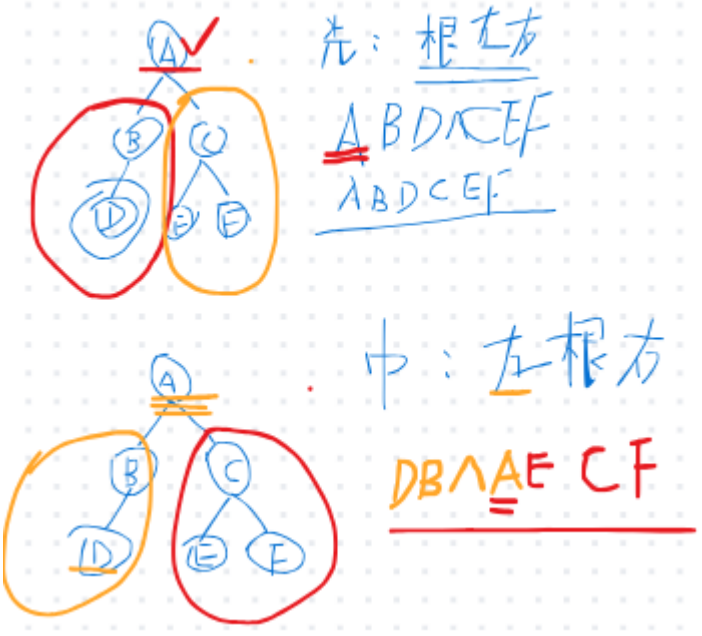
（2）中(根)序遍历（左根右） DBAECF

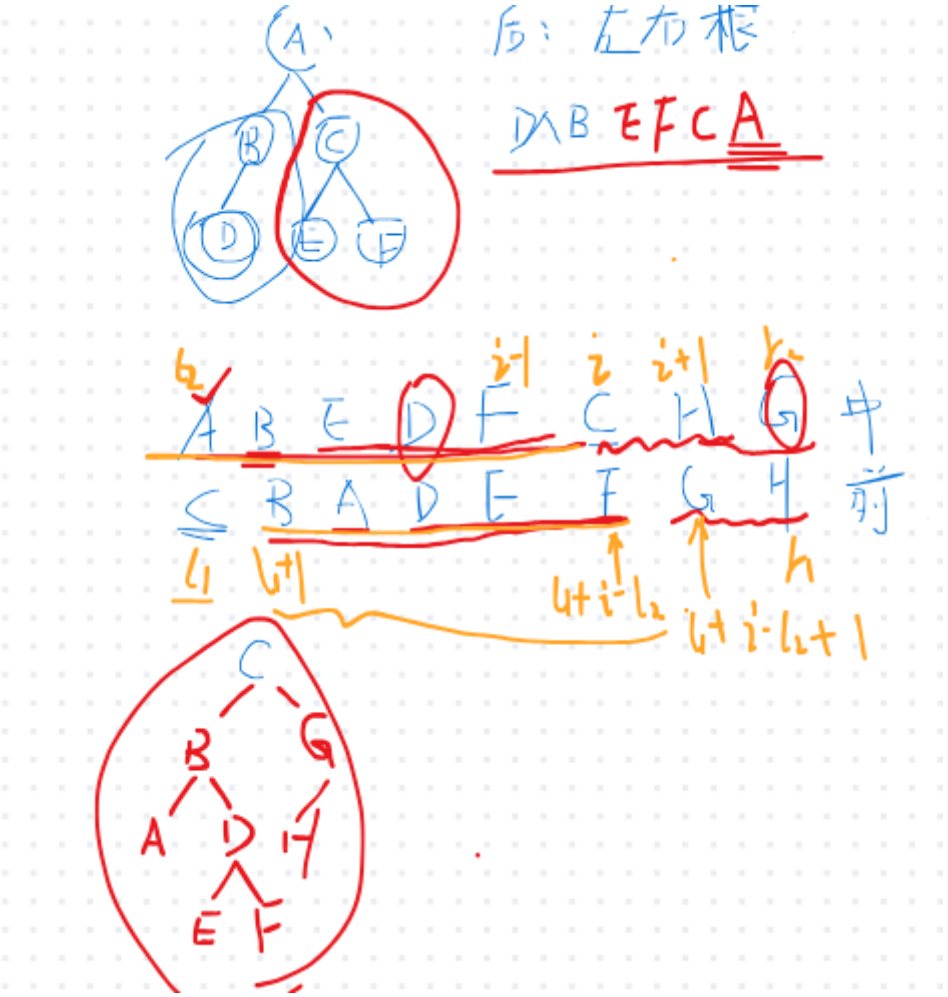
（3）后(根)序遍历（左右根） DBEFCA

广度优先搜索：

层序遍历 ABCDEF







例题：洛谷P1305

Ac代码：

```
#include<bits/stdc++.h>
using namespace std;

int n,l[30],r[30],p[30],vis[30];

void dfs(int u)
{
    cout<<char(u+'a'-1);
    if(l[u]) dfs(l[u]);
    if(r[u]) dfs(r[u]);
}

int main()
{
    cin>>n;
    int root;
    for(int i=1;i<=n;i++)
    {
        string s;
        cin>>s;
        int t=s[0]-'a'+1;
        vis[t]=1;
        if(s[1]!='*')
```

```
        {
            l[t]=s[1]-'a'+1;
            p[l[t]]=t;
            vis[l[t]]=1;
        }
        if(s[2]!='*')
        {
            r[t]=s[2]-'a'+1;
            p[r[t]]=t;
            vis[r[t]]=1;
        }
    }
    for(int i=1;i<=26;i++)
    {
        if(vis[i]==1&&p[i]==0)
        {
            root=i;
            break;
        }
    }
    dfs(root);
    return 0;
}
```

5、哈夫曼树

路径：在一棵树中，一个结点到另一个结点之间的通路，称为路径。

路径长度：在一条路径中，每经过一个结点，路径长度都要加 1 。例如在一棵树中，规定根结点所在层数为1层，那么从根结点到第 i 层结点的路径长度为 i - 1 。图 1 中从根结点到结点 c 的路径长度为 3。

结点的权：给每一个结点赋予一个新的数值，被称为这个结点的权。例如，图 1 中结点 a 的权为 7，结点 b 的权为 5。

结点的带权路径长度：指的是从根结点到该结点之间的路径长度与该结点的权的乘积。例如，图 1 中结点 b 的带权路径长度为 2 * 5 = 10 。

树的带权路径长度为树中所有叶子结点的带权路径长度之和。通常记作 “WPL” 。例如图 1 中所示的这颗树的带权路径长度为：WPL= 7 * 1 + 5 * 2 + 2 * 3 + 4 * 3

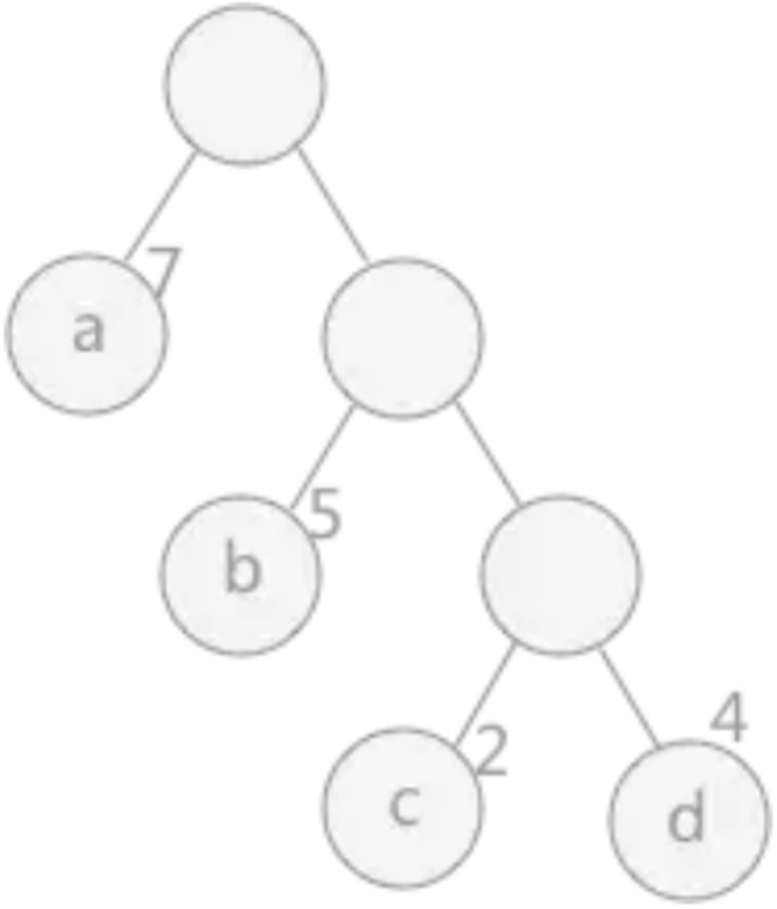


图1 哈夫曼树

(1)什么是哈夫曼树

当用 n 个结点（都做叶子结点且都有各自的权值）试图构建一棵树时，如果构建的这棵树的带权路径长度最小，称这棵树为“最优二叉树”，有时也叫“赫夫曼树”或者“哈夫曼树”。

在构建哈弗曼树时，要使树的带权路径长度最小，只需要遵循一个原则，那就是：权重越大的结点离树根越近。在图 1 中，因为结点 a 的权值最大，所以理应直接作为根结点的孩子结点。

(2)哈夫曼树构建方法

哈夫曼树的构建方法是通过贪心算法实现的。基本步骤如下：

1、将给定的一组权值作为叶子节点构建成n棵只有根节点的二叉树。

2、每次选择权值最小的两棵树合并为一棵新树，合并后的树的权值为两棵子树权值之和。

3、重复步骤2，直到只剩下一棵树，即哈夫曼树。

有n个叶节点的哈夫曼树一定有2*n-1个节点。

(3)存储结构

可以用一个大小为2n-1的一维数组来存放哈夫曼树的各个结点。

```
#define N 20//叶子结点的最大值
#define M 2*N-1//所有结点的最大值
typedef struct{
    int weight;//结点的权值
    int parent;//双亲的下标
    int LChild;//左孩子结点的下标
    int RChild;//右孩子结点的下标
}HTNode,HuffmanTree[M+1];//HuffmanTree是一个结构数组类型，0号单元不用
```

对于有n个叶子结点的哈夫曼树，结点总数为2n-1个，为实现方便，将叶子结点集中存储在前面部分n个位置，而后面的n-1个位置中存储其余非叶子结点。

(4)算法实现

```
#define INF 0x7FFFFFF//无穷

/*从当前森林中（在森林中树的根结点的双亲为0）选择两棵根的权值最小的树*/
void select(HuffmanTree ht,int n,int * s1,int * s2){//在ht[1]~ht[i-1]的范围内选择两个parent为0且weight最小的结点，其序号分别赋给s1,s2
    int i,m1=INF,m2=INF,indexm1,indexm2;//初始化m1,m2为INF用来存储两个最小的weight，用index来记录其下标
    for(i=1;i<=n;++i){//遍历结点
        if(ht[i].parent==0){//当前遍历的结点还没有被选走，即parent为0
            if(ht[i].weight<m1){//假设m1<m2，如果当前结点比m1还小，则先把目前m1的所有信息转移给m2，再更新m1的信息
                m2 = m1;
                indexm2 = indexm1;
                m1 = ht[i].weight;
                indexm1 = i;
            }
            else if(ht[i].weight<m2){//如果当前结点比m1大，却比m2小，则直接更新m2的信息
                m2 = ht[i].weight;
                indexm2 = i;
            }
        }
    }
    *s1 = indexm1; *s2 = indexm2;//两个weight最小的结点的序号分别赋给s1,s2
}

/*创建哈夫曼树算法*/
void CrtHuffmanTree(HuffmanTree ht,int w[],int n){//构造哈夫曼树ht[M+1],w[]存放n个权值
    int i,s1,s2,m;
    for(i=1;i<=n;++i){//1~n号单元存放叶子结点，初始化
        ht[i].weight = w[i]; ht[i].parent = 0; ht[i].LChild = 0; ht[i].RChild = 0;
    }
    m = 2*n-1;
    for(i=n+1;i<=m;++i){//n+1~m号单元存放非叶结点，初始化
        ht[i].weight = 0; ht[i].parent = 0; ht[i].LChild = 0; ht[i].RChild = 0;
    }
    for(i=n+1;i<=m;++i){
        select(ht,i-1,&s1,&s2);//在ht[1]~ht[i-1]的范围内选择两个parent为0
```

```
                                //且weight最小的结点，其序号分别赋给s1,s2
        ht[i].weight = ht[s1].weight+ht[s2].weight;
        ht[s1].parent = i; ht[s2].parent = i;
        ht[i].LChild = s1; ht[i].RChild = s2;
    }//哈夫曼树建立完毕
}

int main()
{
    HuffmanTree T;
    int w[N+1];
    int i,n,m;
    scanf("%d",&n);//读取叶子结点个数
    m = 2*n-1;
    for(i=1;i<=n;++i) scanf("%d",&w[i]);//读取权值
    CrtHuffmanTree(T,w,n);
    for(i=1;i<=m;++i){//打表
        printf("%d || %d  %d  %d
%d\n",i,T[i].weight,T[i].parent,T[i].LChild,T[i].RChild);
    }
    return 0;
}
```

//5 7 3 2 8

```
T[]
下标      权值    父亲    l    r
1         5     7 &
2         7     8 &
3         3     6 &
4         2     6 &
5         8     8 &
6         5     7      3    4    &
7         10    9      1    6    &
8         15    9      2    5    &
9         25           7    8
```

例题：[P1090 NOIP2004 提高组] 合并果子

Ac代码：

```
#include<iostream>
#include<algorithm>
#include<queue>
using namespace std;

int a[1000000],sum[1000000];

priority_queue<int, vector<int>, greater<int>>q;

int main()
{
    int n,s=0;
    cin>>n;
    for(int i=1;i<=n;i++)
```

```
    {
        cin>>a[i];
        q.push(a[i]);
    }
    while(1)
    {
        int a=q.top();
        q.pop();
        int b=q.top();
        q.pop();
        s+=(a+b);
        q.push(a+b);
        if(q.size()<=1) break;
    }
    cout<<s;
    return 0;
}
```

课后习题：(图论)树 课后习题