树形动规

第一课

例1：选课 NKOJ1217

在大学里，为了达到规定的学分，学生们必须从很多课程里选择一些课程来学习，但有些课程必须在某些课程之前学习，如高等数学总是在其它课程之前学习。

现在有N门功课，每门课有个学分，**每门课最多只有一门直接先修课。每门课可是最多两门其它课程的直接先修课**（若课程a是课程b的先修课即只有学完了课程a，才能学习课程b）。一个学生要从这些课程里选择M门课程学习，问他能获得的最大学分是多少？(**只有一门课没有直接先修课**)

输入：
第一行有两个整数N,M用空格隔开(1<=N<=200,1<=M<=150)
接下来的N行,第I+1行包含两个整数ki和si,
ki表示第I门课的直接先修课，si表示第i门课的学分。
若ki=0表示没有直接先修课（1<=ki<=N, 1<=si<=20）。

输出：
只有一行，选M门课程的最大得分。

输入样例：
5 3
3 5
5 18
5 2
3 10
0 7

输出样例：
27

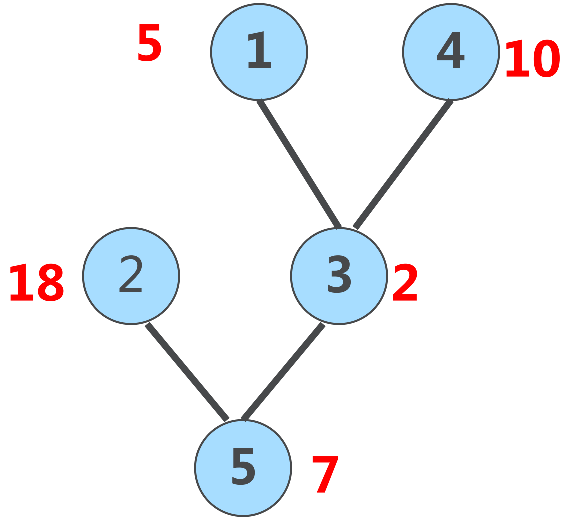
例1：选课 问题分析

1.每门课程最多只有1门直接先修课，如果我们把课程看成结点，也就是说每个结点最多只一个前驱结点。

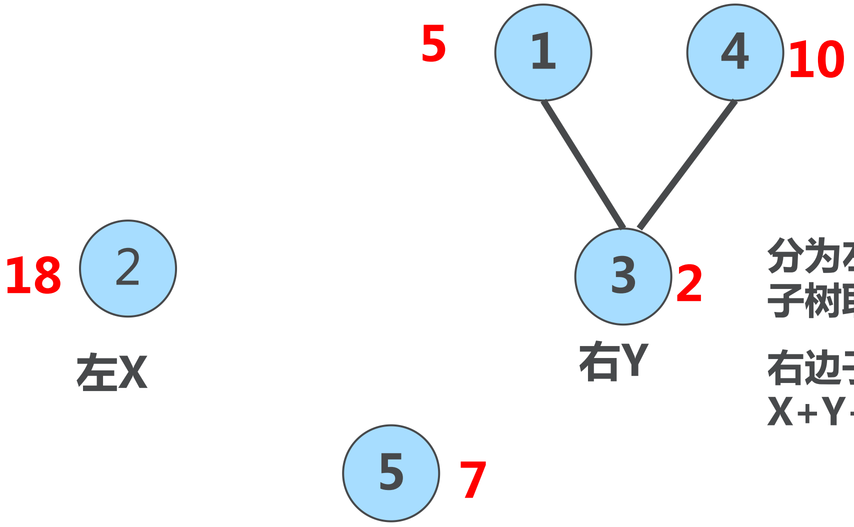
2.如果把前驱结点看成父结点，换句话说，每个结点只有一个父结点。同时，每门课程最多是其他两门课的直接先修课，并且只有一门课程没有直接先修课。显然具有这种结构的模型是树结构，而且是一棵**二叉树。没有先修课的这门课就是根节点。**

3.这样，问题就转化为**在一个N个结点的二叉树中选取M个结点，使得所选结点的权值之和最大**。同时满足每次选取时，若选儿子结点，必选根结点的条件。

例1：选课 问题分析

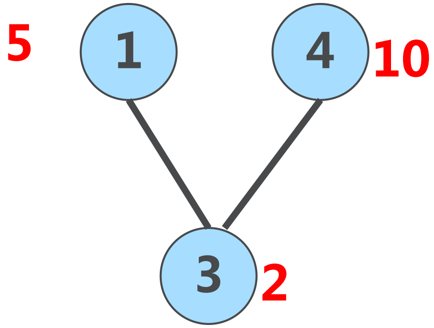


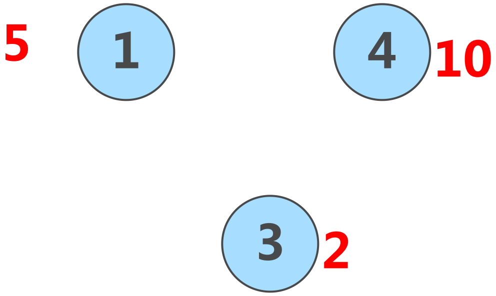
要在N个点中选M个节点



分为左右两棵子树，左边子树取X个节点

右边子树取Y个节点使得X+Y+1=M





对于划分出来的子树，可按同样的方法再递归划分为左X1, 右Y1两棵子树，使得X1+Y1+1=Y

例1：选课 解法

首先构建二叉树

阶段：依次讨论以每个节点为根的子树

状态：f[x][y]表示以x为根的子树保留(选)y个节点的最大值

决策：x的左右两棵子树分别保留多少个节点。

方程： **f[x][y]= max{ f[left[x]][k]+f[right[x]][y-1-k]+v[x] }**

v[x]表示x号节点的权值，left[x]为x左儿子的编号，right[x]为x的右儿子
f[left[x]][k]表示x的左子树保留k个节点的最优值
f[right[x]][y-1-k]表示x的右子树保留y-1-k个节点的最优值
满足： 左子树保留的节点数+右子树保留的节点数+1=y

边界条件：$1<=x<=n$ $1<=y<=m$ $0<=k<=y-1$

时间复杂度：$O(nm^2)$

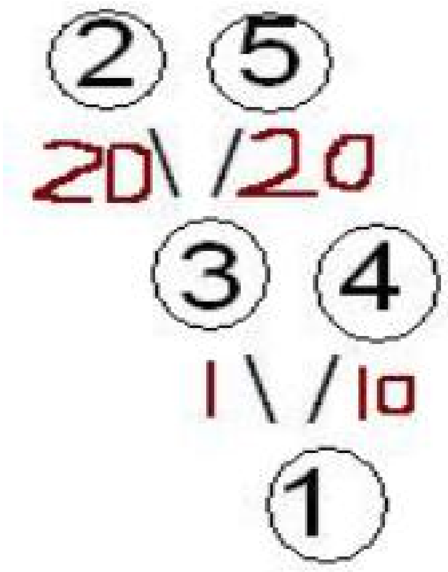
例1：选课 参考代码

```
//输入，建树
scanf("%d%d",&n,&m);
for(i=1;i<=n;i++)
{       //k为i的先修课，i的学分为s
        scanf("%d%d",&k,&s);
        v[i]=s;
        if(k!=0)
        {       //Left[k]记录k的左儿子
                if(Left[k]==0) Left[k]=i;
                else Right[k]=i;
        }else Root=i;
}
```

```
//动规
void TreeDP(int x,int y)  //求f[x][y]
{
    if(f[x][y]>0)return;     //已经被计算的就不再重复计算
    if(x>0)
    {
        for(int k=0;k<=y-1;k++)
        {
            TreeDP(left[x],k);           //求f[left[x]][k]
            TreeDP(right[x],y-1-k);  //求f[right[x]][y-1-k]
             if(f[left[x]][k]+f[right[x]][y-1-k]+v[x]>f[x][y])
                 f[x][y]=f[left[x]][k]+f[right[x]][y-1-k]+v[x];
        }
    }
}
```

例2：二叉苹果树(ural 1018)

有一棵苹果树，如果树枝有分叉，一定是分2叉,这棵树共有N个结点，编号为1-N,树根编号一定是1。我们用一根树枝两端连接的结点的编号来描述一根树枝的位置。下面是一颗有4个树枝的树。



现在这颗树枝条太多了，需要剪枝。但是一些树枝上长有苹果。现在告诉你每条树枝上的苹果数(左图红色数字)。给定需要保留的树枝数量M，求出最多能留住多少苹果。

1<=M<= N,1<N<=100

例2：二叉苹果树(ural 1018)

输入格式

第1行2个数，N和M(1<=M<= N,1<N<=100)。

N表示树的结点数，M表示要保留的树枝数量。

接下来N-1行描述树枝的信息。

每行3个整数，前两个是它连接的结点的编号。

第3个数是这根树枝上苹果的数量。

每根树枝上的苹果不超过30000个。

输出格式

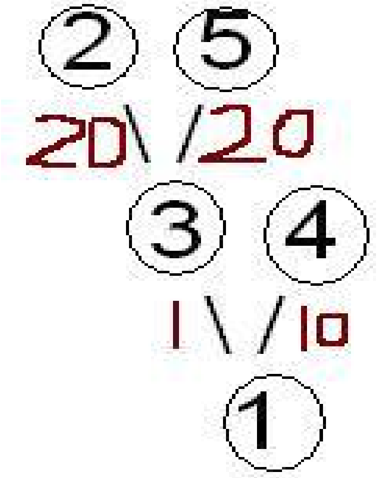
一个数，最多能留住的苹果的数量。

样例输入

5 2
1 3 1
1 4 10
2 3 20
3 5 20

样例输出

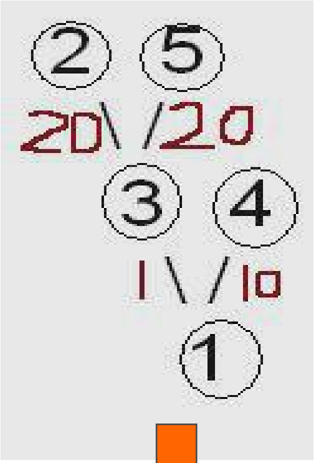
21



例2：二叉苹果树 解题分析

```
2   5
 \ /
  3
   \
    1
```

剪掉1的右枝




```
 4
 /
1
```

剪掉1左枝

```
2   5
 \ /      4
  3
     \ /
      1
```

保留1的左右枝，
左子树保留X条枝，右子树保留Y条枝，
使得X+Y+2=M

由此推出：对于任意子树，若要求其保留M条枝，均可按剪掉左子树，剪掉右子树（剪后判断剩下的枝数是否等于M）和将左右子树分为两部分，分别保留X和Y条边，使得**X+Y+2=M** 来进行讨论.

例2：二叉苹果树 解题分析

首先DFS一遍建好二叉树

阶段：依次讨论以每个节点为根的子树

状态：设f[x][y]表示以x为根的子树共保留y条树枝的最大苹果数

决策：剪掉x的左枝，还是剪掉右枝，还是保留左右枝。

方程： **f[x][y]=**

```
max{
        f[left[x]][y-1]+a[x][left[x]]        //保留x的左枝，剪掉x的右枝
        f[right[x]][y-1]+a[x][right[x]]      //保留x的右枝，剪掉x的左枝
        f[left[x]][k]+f[right[x]][y-2-k]+a[x][left[x]]+a[x][right[x]]
    }   //保留x的左右两条枝，x的左子树保留k条枝，右子树保留y-2-k条枝
```

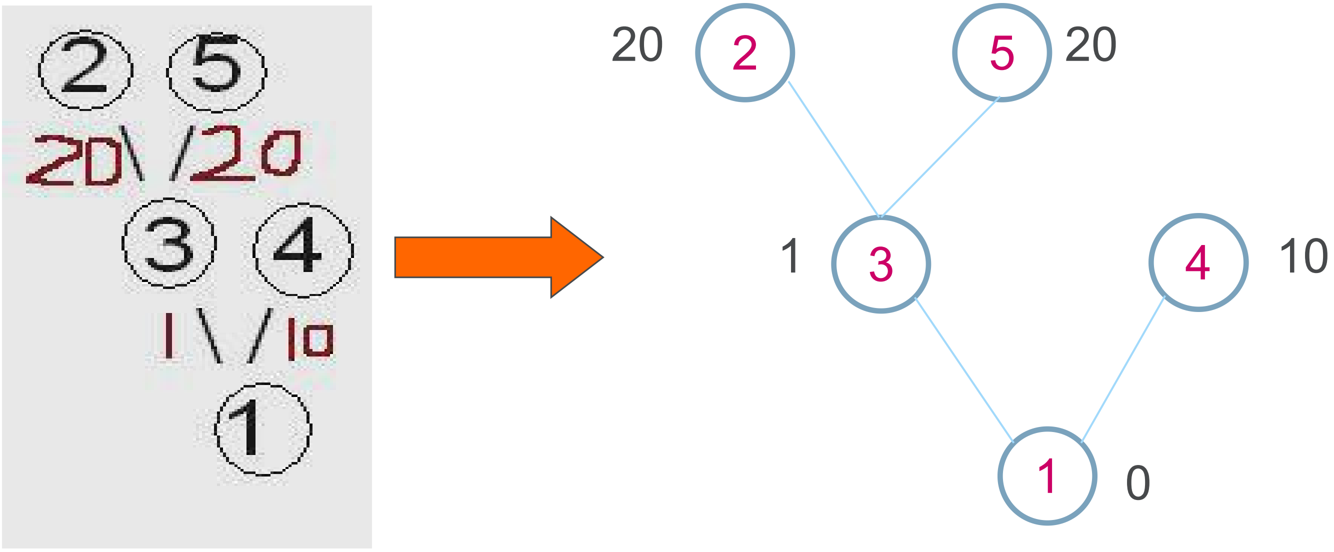
a[x][left[x]]表示x的左枝的苹果数，a[x][right[x]]对应x右枝的苹果数

边界条件：$1<=x<=n$　$1<=y<=m$　$0<=k<=y-2$

时间复杂度：$O(nm^2)$

例2：二叉苹果树 解题分析

换一种解决方法，把边的权值存在点上！



状态：设f[x][y]表示以x为根的子树共保留y个节点的最大苹果数

方程： f[x][y]= max{ f[left[x]][k]+f[right[x]][y-1-k]+v[x] }
0<=k<y

最后答案为:f[1][m+1]

例2：二叉苹果树 参考代码

```
//DFS建树
void make_tree(int v)
{
    visit[v]=true;
    for(int i=1;i<=N;i++)
      if(!visit[i]&&map[v][i]!=-1)
      {
           if(!Tree[v].L) Tree[v].L=i;
           else Tree[v].R=i;
           Tree[i].data=map[v][i];
           make_tree(i);
      }
}
```

```
//动规
int DP(int x,int y)   //求f[x][y],x为根的子树中保留y个节点
{
    if(f[x][y]) return f[x][y];          //已经算过的直接返回结果
    if(y==0) return 0;
    if(y==1) return Tree[v].data;
    int Max=0;
    for(int k=0;k<=y-1;k++)
        Max=max(Max,DP(Tree[v].L,k)+DP(Tree[v].R,y-k-1));
    f[x][y]=Max+Tree[v].data;
    return f[x][y];
}
```

树形动规 小结

树形动态规划就是在“树”的数据结构上的动态规划

阶段：以每节点所代表的子树作为一个阶段。

状态：以每棵子树的最优值作为状态

决策：当前节点的最优值由其子节点代表的子树的最优值选择而来

实现：递归程序实现DP（记忆化搜索）

例3：选课2.0 NKOJ2317

在大学里每个学生，为了达到一定的学分，必须从很多课程里选择一些课程来学习，但有些课程必须在某些课程之前学习，如高等数学总是在其它课程之前学习。

现在有N门功课，每门课有个学分，**每门课最多只有一门直接先修课。**要选择某门课程，必须先选修它的先修课。**一门课程可能是多门其他可能的直接先修课。**

一个学生要从这些课程里选择M门课程学习，问他能获得的最大学分是多少？（**可能有多门课程没有直接先修课**）

1<=M,N<=300

例3：选课2.0 问题分析1

1.每门课程最多只有1门直接先修课，如果我们把课程看成结点，也就是说每个结点最多只一个前驱结点。

2.如果把前驱结点看成父结点，换句话说，每个结点最多有一个父结点。跟选课1.0不同，题目没有明确告知每个节点的儿子数，所以不一定是二叉树。可能有多个节点没有父亲(先修课)，显然具有这种结构的模型是树结构，要么是一棵**多叉树**，要么是一个**森林**。

3.这样，问题就转化为在一个**N 个结点的森林中选取M个结点**，使得所选结点的权值之和最大。同时满足每次选取时，若选儿子结点，必选根结点的条件。

例3：选课2.0 问题分析 森林

给出的数据呈**森林**状，怎么处理？

如下图图1，为两棵树构成的森林。
我们可以**虚拟一个根结点**，将这些树连接起来，那么森林就转会为了1棵树，如下图图2
显然选M=4时，选3，2，7，6几门课程最优。

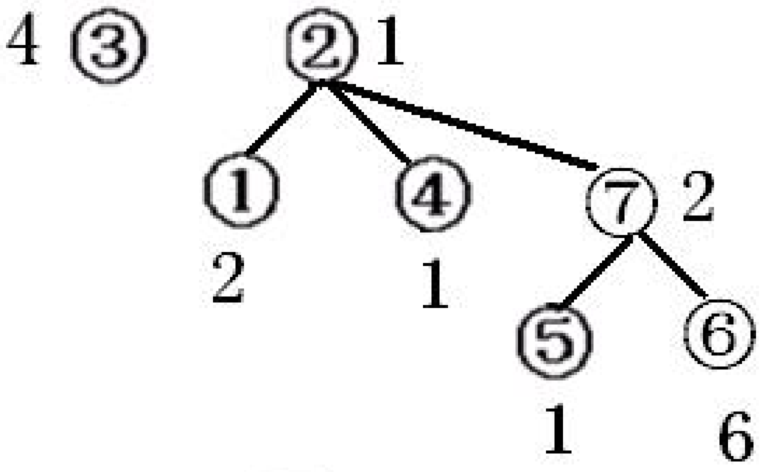


图1

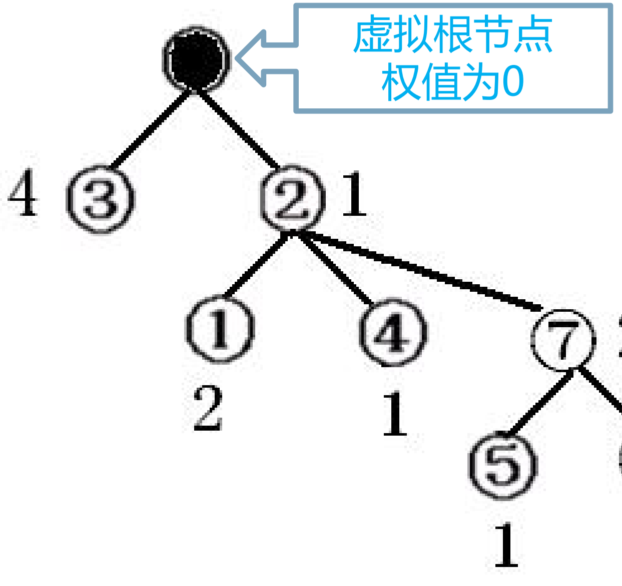


图2

例3：选课2.0 问题分析 多叉树

设状态f[x][y]表示x为根的子树保留y个节点的最优值。

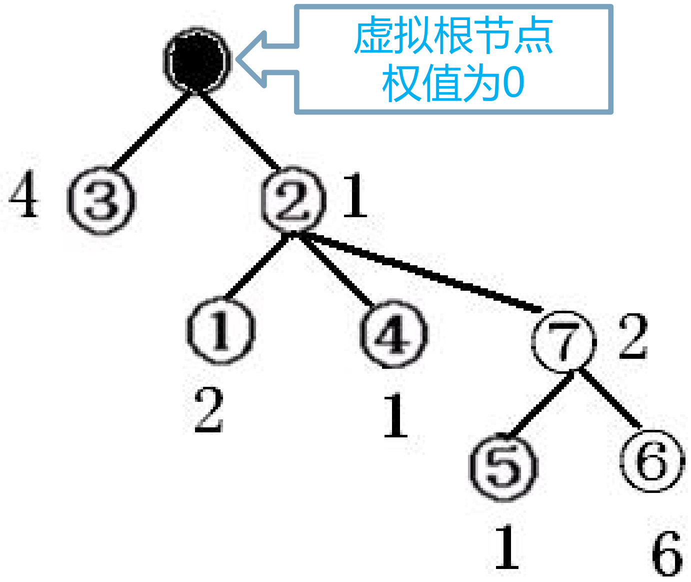


图2

但我们要处理的是一棵多叉树。

对于x节点，我们要讨论将y个节点如何分配在x的所有儿子节点为根的子树上，讨论起来十分麻烦，并且时间复杂度会达到O（N^2M^2），显然不可行。

我们需要将**多叉树转换成二叉树**来处理！

例3：选课2.0 多叉树转二叉树

多叉树转二叉树的规则：每个节点的左起第一个孩子变成它的左孩子，右边第一个兄弟变成它的右孩子。

例3：选课2.0 多叉树转二叉树 代码

```
cin>>n;
for (i=1; i<=n; i++)
{
    cin>>x>>y;            //x是y的父亲
    right[y]=left[x];     //x的左儿子是y的兄弟，成为y的右儿子
    left[x]=y;            //y成为x新的左儿子
}
```

父亲的老儿子成为新儿子的右儿子；
新儿子取代老儿子成为父亲的左儿子；

例3：选课2.0 转二叉树后的DP分析

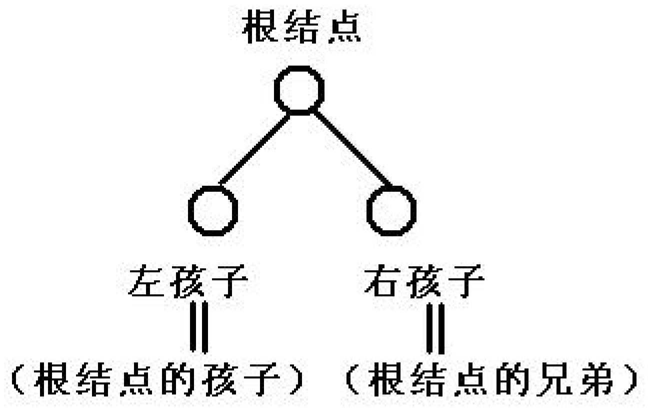
仔细理解转二叉树后左右孩子的意义（如右图）：

左孩子：原根结点的孩子

右孩子：原根结点的兄弟

也就是说，左孩子分配选课资源时，

根结点必须要选修，而与右孩子无关。



因此，设f[i][j]表示以i为根结点的二叉树分配j门课程的所获得的最大学分，则有，

```
f[i][j]= max{
             f[left[i]][k]+f[right[i]][j-k-1]+v[i]    //选第i门课
             f[right[i]][j]                           //不选第i门课
         }
```

边界条件：0<=k<j<n, 1<=i <=n

时间复杂度：$O(nm^2)$

例4：通往自由的钥匙 NKOJ1469

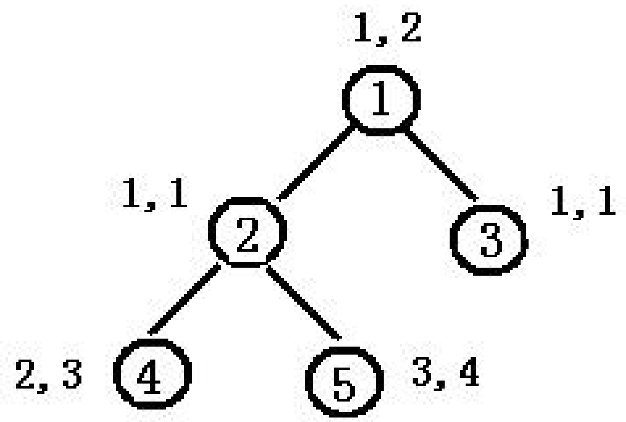
通向自由的钥匙被放n个房间里，这**n个房间由n-1条走廊连接**。但是每个房间里都有特别的保护魔法，在它的作用下，我们无法通过这个房间，也无法取得其中的钥匙。

我们可以通过消耗能量来破坏房间里的魔法，但是我们的能量是有限的。一个房间的魔法被破坏后，就不再有魔法保护了。

那么，如果我最先站在1号房间（1号房间的保护魔法依然是有效的，也就是，如果不耗费能量，我们无法通过1号房间，也无法取得房间中的钥匙），如果我们拥有的能量为P，现在告诉你每个房间的需要消耗的能量和钥匙数量，我们最多能取得多少钥匙？

1<=p,n<=100

如右图，当P=5
可取1,2,5三间房的钥匙，消耗为1+1+3 = 5，
获得的钥匙为2+1+4 = 7



例4：通往自由的钥匙

1.这n个房间由n-1条走廊连接，说明该问题的模型是一棵树

2.也就是说，给出P的资源，如何在树中分配这些资源，得到最大值，即钥匙。这是典型的树型动态规划问题(**树型背包**)。

3.将树转化为孩子兄弟表示法(二叉树)，由于根的左孩子还是它的孩子，右孩子是它的兄弟，因此：
树根获取资源，则左右孩子均可获取资源
树根不获取资源，则左孩子不能获取资源，右孩子可获取资源。

例4：通往自由的钥匙 解法

首先构建二叉树

阶段：依次讨论以每个节点为根的子树

状态：f[x][y]表示以x为根的子树分配y个能量能得到的最多钥匙数量

方程： **f[x][y]=**

max{

f[left[x]][k]+f[right[x]][y-k-cost[x]]+key[x]

f[right[x]][y] //不选根节点x

}

边界条件：1<=x<=n 1<=y<=p 0<=k<=y-cost[x]

时间复杂度：$O(np^2)$

例4：通往自由的钥匙 参考代码

```
//输入
for(i=1;i<n;i++)
{
    scanf("%d%d",&x,&y);
    map[x][y]=map[y][x]=true;
}
```

```
//DP
int dp(int x,int y)
{
      if(x<0) return 0;
      if(f[x][y]>0) return f[x][y];
      f[x][y]=dp(r[x],y);
      for(int i=0;i<=y-cost[x];i++)
         f[x][y]=max(f[x][y],dp(l[x],i)+dp(r[x],y-i-cost[x])+key[x]);
      return f[x][y];
}
```

```
//DFS建树
void build_tree(int v)
{
    vis[v]=true;
    for(int i=1;i<=n;i++)
      if(!vis[i]&&map[v][i])
       {
          r[i]=l[v];
          l[v]=i;
          build_tree(i);
       }
}
```

例5：经典问题“电脑 NKOJ3694”

给你一棵由n个节点构成的树，树中每条边都有一定的长度值。

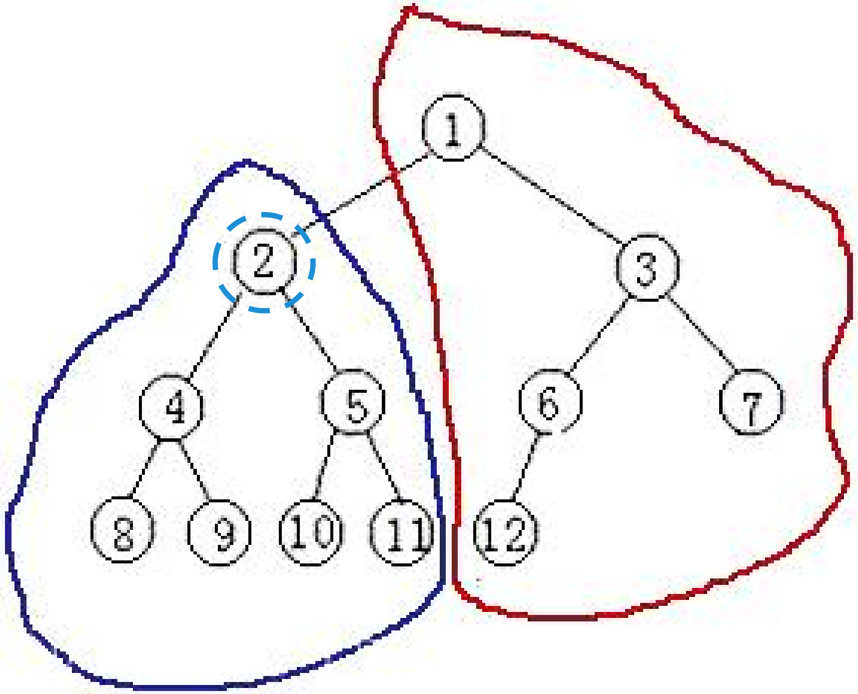
对于树中的每个节点，请你找出树中离它最远的点，并输出它们的距离。

1<=n<=10000

0<=边的长度<=1000000000

改编自HDU2196

例3：电脑 问题分析1



观察左边这棵树，如果我们要找距离点2最远的点的距离MaxDis[2]。

图中1是2的父亲。
我们以1到2这条边为界，将这颗树分为红蓝两部分。

假设dis[2]为2所在这棵蓝色子树中，距离2最远的点的距离值。

dis[1]为1所在的红色子树中，距离1最远的点的距离值。

MaxDis[2]= max{ dis[2], dis[1]+Len[1][2] } Len[1][2]为1到2这条边的长度

结论： 对于任意节点i,与i最远的点的距离
要么是i的父亲往下能到的最远点的距离+i与父亲的边长；
要么是i的儿子往下能到的最远点的距离+i与儿子的边长；

例3：电脑 问题分析2

结论： 对于任意节点i,与i最远的点的距离
要么是i的父亲往下能到的最远点的距离+i与父亲的边长；
要么是i的儿子往下能到的最远点的距离+i与儿子的边长；

状态：

dp[i][0]:从点i**往上**经过**父亲节点**，能到达的最长距离。

dp[i][1]:从点i**往下**到叶子节点，能到达的**最长**距离。

dp[i][2]:从点i**往下**到叶子节点，能到达的**次长**距离。

为什么要记录次长距离？
因为dp[i][0]的路线有可能与dp[i][1]重合，这个时候我们就取次长距离来计算dp[i][0]

方程：

dp[i][1]=max{ dp[j][1]+Len[i][j] } j是i的儿子

dp[i][2]=max{ dp[j][1]+Len[i][j] } j是i的儿子 并且 j不是**BestSon[i]**

***BestSon[i]**记录i往下最长距离来自哪个儿子所在子树*

dp[i][0]=max{ dp[fa][1] , dp[fa][0] }+Len[fa][i] fa是i的父亲 并且 i!=BestSon[fa]

dp[i][0]=max{ dp[fa][2] , dp[fa][0] }+Len[fa][i] fa是i的父亲 并且 i==BestSon[fa]

答案： **ans[i]=max{ dp[i][1] , dp[i][0] } 1<=i<=n**

例3：电脑 参考代码1

dp[i][1]:从点i**往下**到叶子节点，能到达的**最长**距离。

dp[i][2]:从点i**往下**到叶子节点，能到达的**次长**距离。

```
void DP_Down(int i,int fa)                        //往下求dp[i][1]和dp[i][2], fa为i的父亲节点
{
        for (int  j .......)                      //j依次枚举i号点的所有儿子，求dp[i][1]
        {
            if(j==fa)continue;
            DP_Down(j);                           //先递归求出儿子节点j的dp[j][ ]值
            if (dp[i][1]<dp[j][1]+Len[i][j])
            {
                dp[i][1]=dp[j][1]+Len[i][j];
                BestSon[i] = j;                   //记录i往下的最远路径走的哪个儿子的方向
            }
        }
        for (int j .......)                       //j依次枚举i号点的所有儿子，求dp[i][2]
        {
                if(j==fa)continue;
                if (j==BestSon[i]) continue;
                if(dp[i][2]<dp[j][1]+Len[i][j])dp[i][2]<dp[j][1]+Len[i][j];
        }
}
```

例3：电脑 参考代码2

dp[i][1]:从点i**往下**到叶子节点，能到达的**最长**距离。

dp[i][2]:从点i**往下**到叶子节点，能到达的**次长**距离。

dp[i][0]:从点i**往上经过父亲节点**，能到达的最长距离。

```
void DP_Up(int fa)                                   //往上计算fa的儿子节点i的dp[i][0]值
{
        for (int  i ......)                          //i依次枚举fa点的所有儿子
        {
            if (i == BestSon[fa])dp[i][0] = max(dp[fa][2],dp[fa][0])+Len[fa][i];
            else  dp[i][0] = max(dp[fa][1],dp[fa][0])+Len[fa][i];
            DP_Up(i);                                //思考：DP_Up(i)能不能放到if前面？
        }
}
```

```
int main()
......
    for (int i = 1; i <= n; i++)cout<<max(dp[i][1],dp[i][0]);
```

例3：补充的知识“树的直径”

树的直径：树中距离最远的两个点间的距离，称为该树的直径。

树的直径的求法：

1.在树中**任选一点S**,从它出发，找出离S最远的点X(从S出发做BFS或DFS);

2.从X出发，找出离X最远的点Y（从X出发做BFS或DFS）;

X到Y的距离就是该树的直径。

上述结论，很简单，自己证明一下！

例3：电脑 第二种解法

树的直径：树中距离最远的两个点间的距离，称为该树的直径。

可以证明：**对于树上任意节点i，与它距离最远的节点j一定是树中某条直径的端点。**

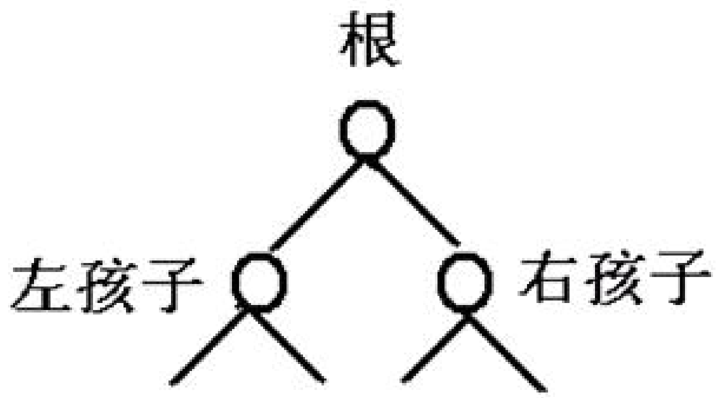
于是我们得到解法：

1. 从树中任意点s出发，通过搜索，找到离s最远的点x(x一定是一条直径的端点);
2. 从x出发，通过搜索，找到离x最远的点y(y一定是x所在直径的另一个端点)；
   搜索时，记录下x到每个点的距离DisX[i]
3. 从y出发，通过搜索，找出y到每个点的距离DisY[i]
4. 对于任意点i,从它出发能到达的最远点的距离就是max{ DisX[i]，DisY[i] }

结论：三次搜索即可解决问题。

本课小结

树型动态规划有一个共性，那就是它的基本模型都是一棵树或者森林。为了考虑方便，一般情况下都将这个树或森林转化为二叉树，如下图，然后整个问题的最优只会涉及到左右孩子的最优，然后考虑根结点的情况，这样化简了问题，最终很容易写出状态转移方程，从而问题得到解决。



另外，并不是所有的问题一定要转化为二叉树来解决，要仔细思考的就是每个结点有些什么状态，这些结点的状态与父、子结点的状态有什么联系，也就是如何由子结点的最优值推出父节点的最优值(即状态转移方程)。

课后习题

讲课例题：1217，2317
基础训练：2318，3688，3504
思维提高：3689，2306，3691