《L4-CSP-J 复赛进阶算法与数据结构》

Day09 动态规划-Dynamic Programming

树形动态规划

主讲人：小武老师

可达信奥_小武老师_keda.ac

树形DP

树形DP就是在一棵树上做DP，这类DP是建立在树结构的基础上。整体的思路就是用树形的结构存储数据。一般将需要维护的答案信息一层一层地传递到根结点，然后得出整个问题的答案。

可达信奥_小武老师_keda.ac

DP的三要素

状态、阶段和决策是构成DP算法的三要素，而子问题重叠性、无后效性和最优子结构性质是问题能用DP求解的三个基本条件。动态规划是解决多阶段决策过程最优化问题的一种方法。

状态

某一阶段的出发位置称为状态。通常一个阶段包含若干状态。

阶段

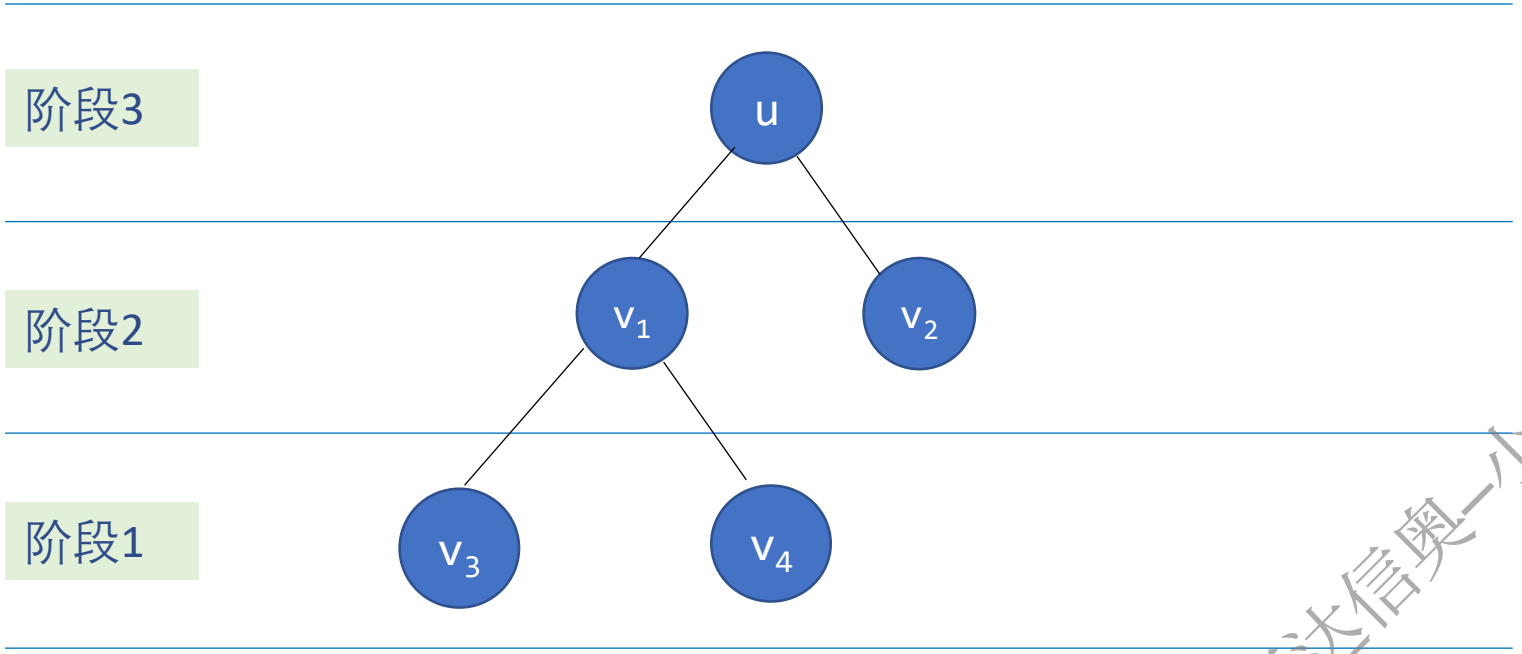
把问题分成几个相互联系的有顺序的几个环节，这些环节即称为阶段。

决策/转移方程

从某阶段的一个状态演变到下一个阶段某状态的选择。也就是从哪来要到哪去。前一阶段的终点就是后一阶段的起点，前一阶段的决策选择导出了后一阶段的状态，这种关系描述了由i阶段到i+1阶段状态的演变规律，称为状态转移方程。

树形DP

顾名思义：树形dp就是在树上进行动态规划，由于树是递归定义的，树形DP的关键和实现方法是dfs. 动态规划是多阶段决策问题，而树形结构有明显的层次性，正好对应动态规划的多个阶段。树形DP求解过程一般为自底向上，将子树从小到大作为DP的“阶段”。DFS, 回溯时向上进行状态转移。



例题选讲

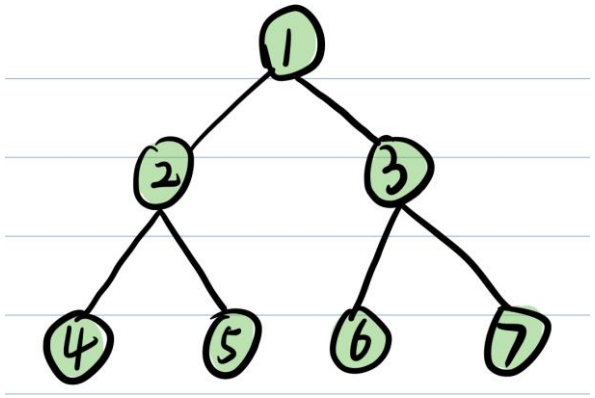
P0498　没有上司的舞会　　P0497　选课

P0324　攻克城堡

P0498 没有上司的舞会

首先，很显然公司的人员关系构成了一棵树。设：
dp[a][1]为节点a参加舞会的情况下，以他为根的子树取得的最大快乐值；
dp[a][0]为root不参加舞会的情况下，以他为根的子树取得的最大快乐值。那么，状态转移方程就是：

$$dp[a][1] = dp[b_1][0] + dp[b_2][0] + \cdots + dp[b_k]$$

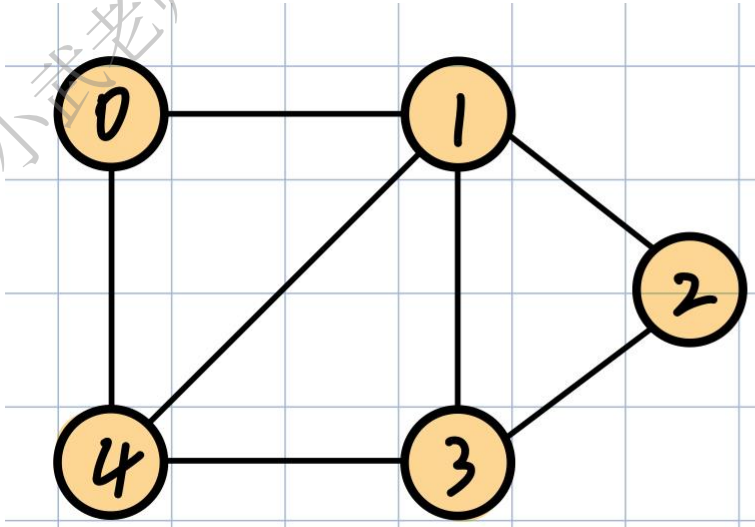


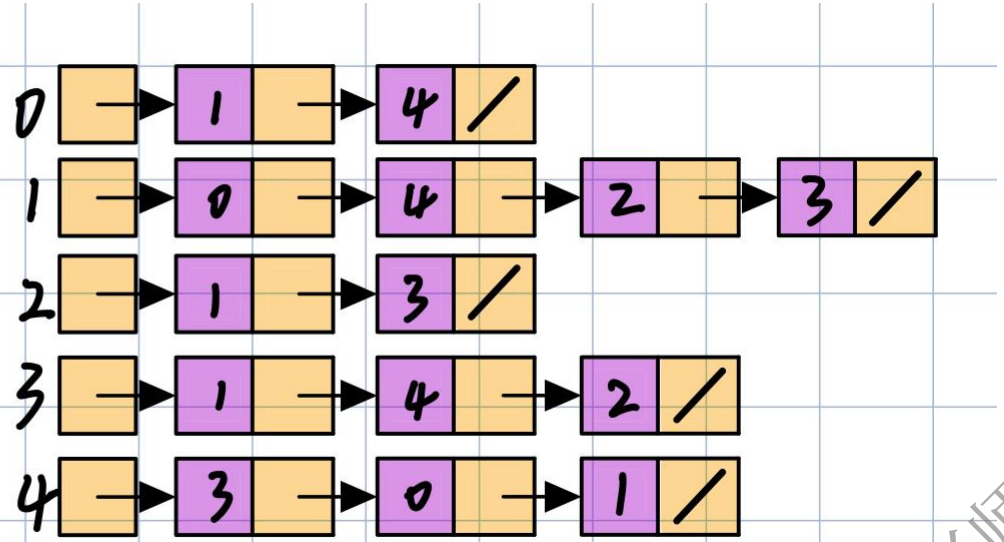
$$dp[a][0] = max(dp[b_1][1], dp[b_1][0]) + max(dp[b_2][1], dp[b_2][0]) + \ + max(dp[b_k][1], dp[b_k][0])$$

其中 $b_1, b_2, \cdots, b_k$ 为a的子结点

最后的答案就是root $max(dp[root][1], dp[root][0])$

P0498 没有上司的舞会






```
vector<int> g[101];
```

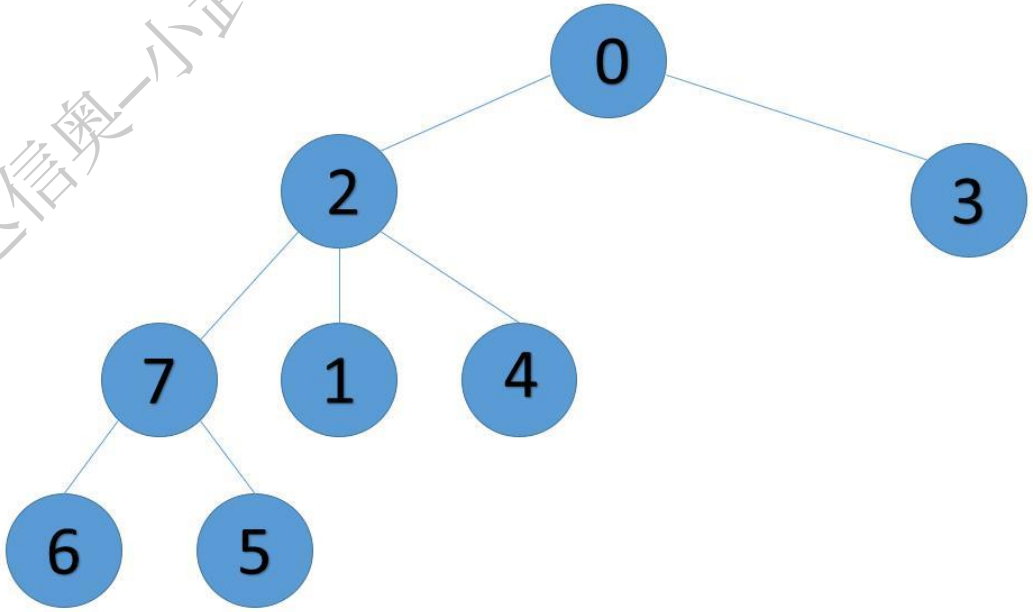
P0498 没有上司的舞会

```
void dfs(int x){
    dp[x][0] = 0;
    dp[x][1] = r[x];
    for(int i = 0; i < g[x].size();i++){
        int y = g[x][i];
        dfs(y);
        dp[x][0] += max(dp[y][0], dp[y][1]);
        dp[x][1] += dp[y][0];
    }
}
```

```
int main(){
    int n;
    cin >> n;
    for(int i = 1; i <= n; i++) cin >> r[i];
    for(int i = 1; i < n; i++){
        int x, y;
        cin >> x >> y;
        g[y].push_back(x);
        v[x] = 1;
    }
    int root;
    for(int i = 1; i <= n; i++){
        if(!v[i]){
            root = i;
            break;
        }
    }
    dfs(root);
    cout << max(dp[root][0], dp[root][1]) << endl;
    return 0;
}
```

P0497 选课

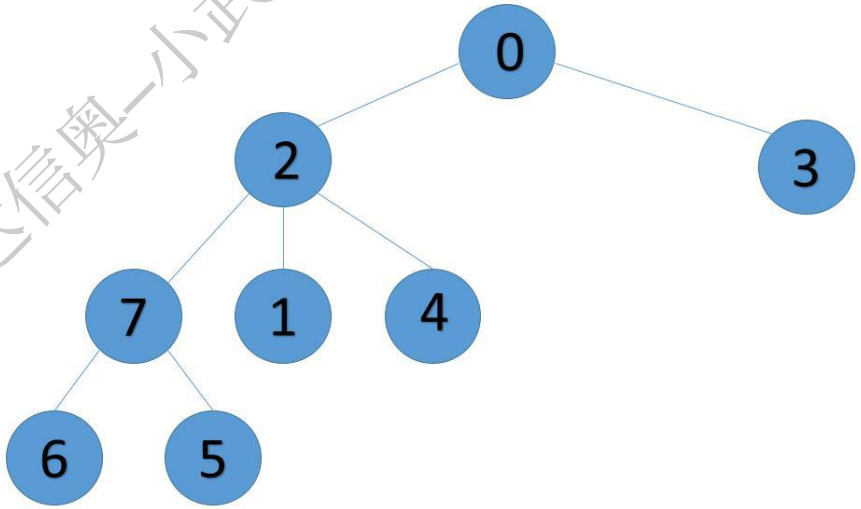
dp[i][j]表示以i为根的子树，选j门课能获得的最大学分



这道题输入的是森林，并不是一棵树，需要加一个结点，把每个子树连到根节点，构成一棵树，即加入0结点作为虚拟课程，学分为0

P0497 选课（树形背包DP）

dp[i][j]表示以i为根的子树，选j门课能获得的最大学分



状态转移数组

dp[i][j]为以i为根的子树选j门课程获得的最大学分。

0结点是必选的，所以需要m+1。

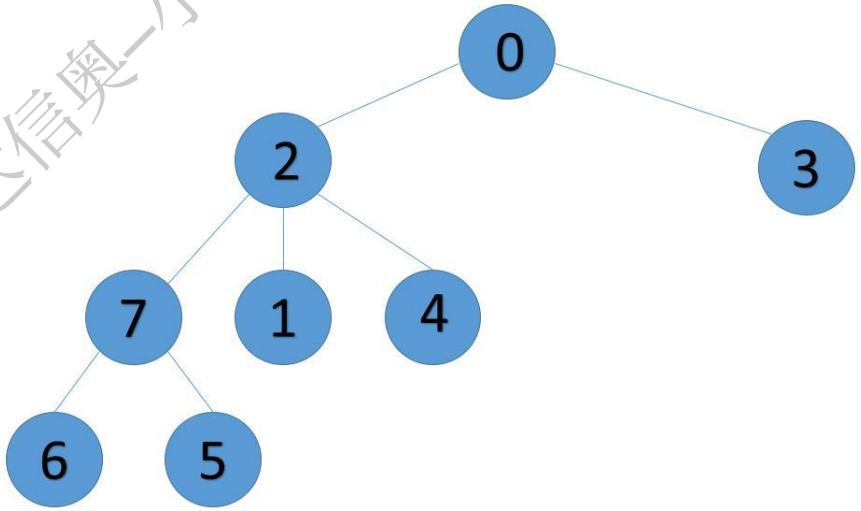
状态转移方程

```
dp[u][j] = max(dp[u][j],dp[v][k] + dp[u][j-k]);
```

$$dp[u][j] = \max(dp[u][j], dp[v][k] + dp[u][j-k]);$$

P0497 选课

dp[i][j]表示以i为根的子树，选j门课能获得的最大学分



状态转移数组

dp[i][j]为以i为根的子树选j门课程获得的最大学分。

0结点是必选的，所以需要m+1。

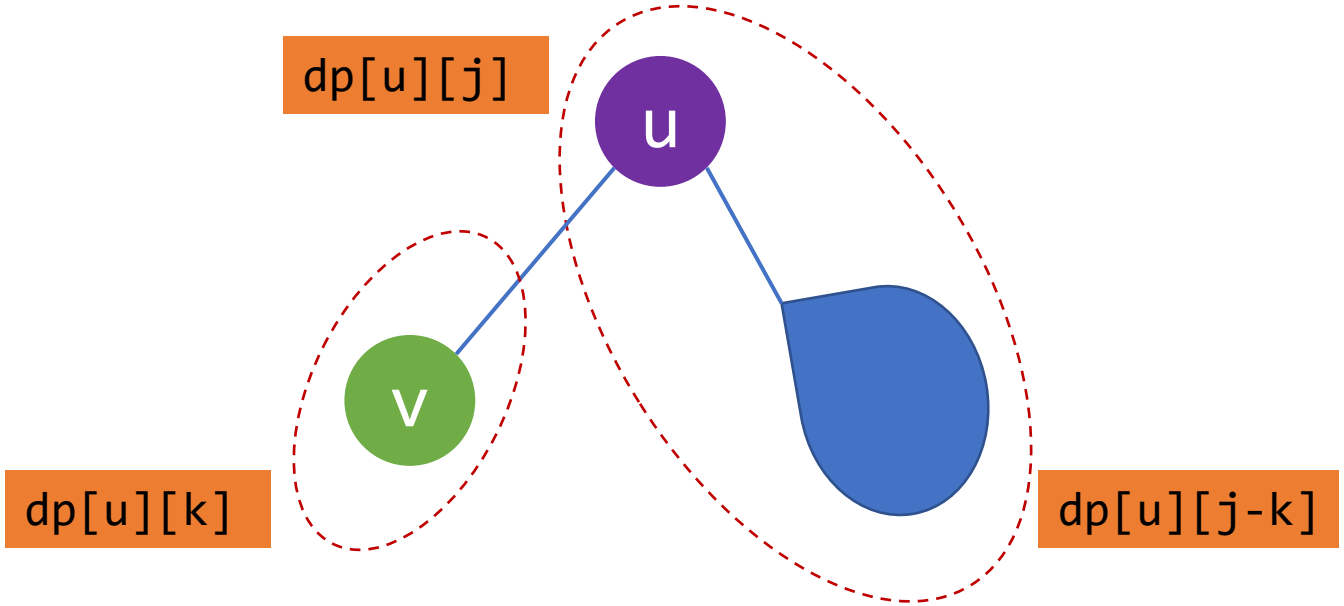
状态转移方程

```
dp[u][j] = max(dp[u][j],dp[v][k] + dp[u][j-k]);
```

$$dp[u][j] = \max(dp[u][j], dp[v][k] + dp[u][j-k])$$

P0497 选课

对节点u的每一个子结点v，若在以v为根的子树中选择k个节点获得的最大和值为dp[v][k]，则在以节点u为根的其余部分获得的最大和值为dp[u][j-k]，两部分求和之后取最大值。



$$dp[u][j] = \max(dp[u][j], dp[v][k] + dp[u][j-k])$$

0-1 背包

dp[i][w]　当背包容量在w的时候，前i个物品放入时候背包中获得的最大价值

不同的阶段 i

V	W	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0
4	1	0	4	4	4	4	4	4
8	3	0	4	4	8	12	12	12
4	5	0	4	4	8	12	12	12
6	1	0	6	10	10	14	18	18

$$dp[i][w] = \begin{cases} dp[i-1][w] & , wt[i] > w \\ \max\{dp[i-1][w], dp[i-1][w-wt[i]] + v[i]\} & , wt[i] \le w \end{cases}$$

完全背包

P0457. 完全背包问题

状态表示：dp[i][w] 表示将第i个物品放入容量为w的背包中可以获得的最大价值

v	w	0	1	2	3	4	5	6
0	0	0	0	0	0	0	0	0
4	1	0	4	8	12	16	20	24
8	3	0	4	8	12	16	20	24
4	5	0	4	8	12	16	20	24
6	1	0	6	12	18	24	30	36

$$dp[i][w] = \begin{cases} dp[i-1][w] & , wt[i] > w \\ \max\{dp[i-1][w], dp[i][w-wt[i]] + v[i]\} & , wt[i] \le w \end{cases}$$

P0497 选课

01背包和完全背包的区别?

阶段：第i类物品就是第i阶段
目标：放入第i类物品，有无提升背包价值
每个物品，都对应容量大小为v[i]-m到个背包

设dp[v]表示重量不超过v公斤的最大价值

```
int n,m;
Cin >> n >> m;
for(int i=1;i<=n;i++) cin>>v[i]>>w[i];
for(int i=1; i <= n;i++){
    for(int j = m; j >= v[i];j--){
        dp[j] = max(dp[j], dp[j-v[i]]+w[i]);
    }
}
cout<<dp[m];
return 0;
```

i相同时，未取值

每选一个物品，背包容量都遍历一次

01背包（逆序遍历,保证不重复放）

设dp[v]表示重量不超过v公斤的最大价值

```
int n,m;
cin>>n>>m;
for(int i=1;i<=n;i++) cin>>v[i]>>w[i];
for(int i=1;i<=n;i++){
    for(int j=v[i];j<=m;j++){
        dp[j]=max(dp[j],dp[j-v[i]]+w[i]);
    }
}
cout<<dp[m];
return 0;
```

i相同时，已取值

每选一个物品，背包容量都遍历一次

完全背包（正序遍历）

P0497 选课

dp[i][j]表示以i为根，选j门课能获得的最大学分

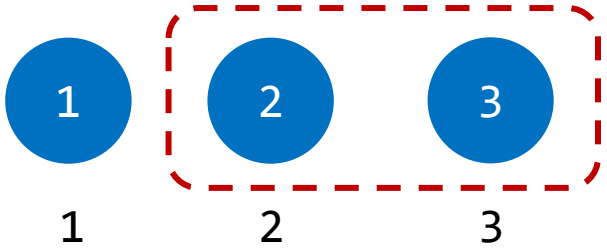
```
int main() {
    cin >> n >> m;
    for(int i = 1; i <= n; i++){
        int x, y;
        cin >> x >> y;
        dp[i][1] = y;
        g[x].push_back(i);
    }
    dfs(0);
    cout << dp[0][m+1];
}
```

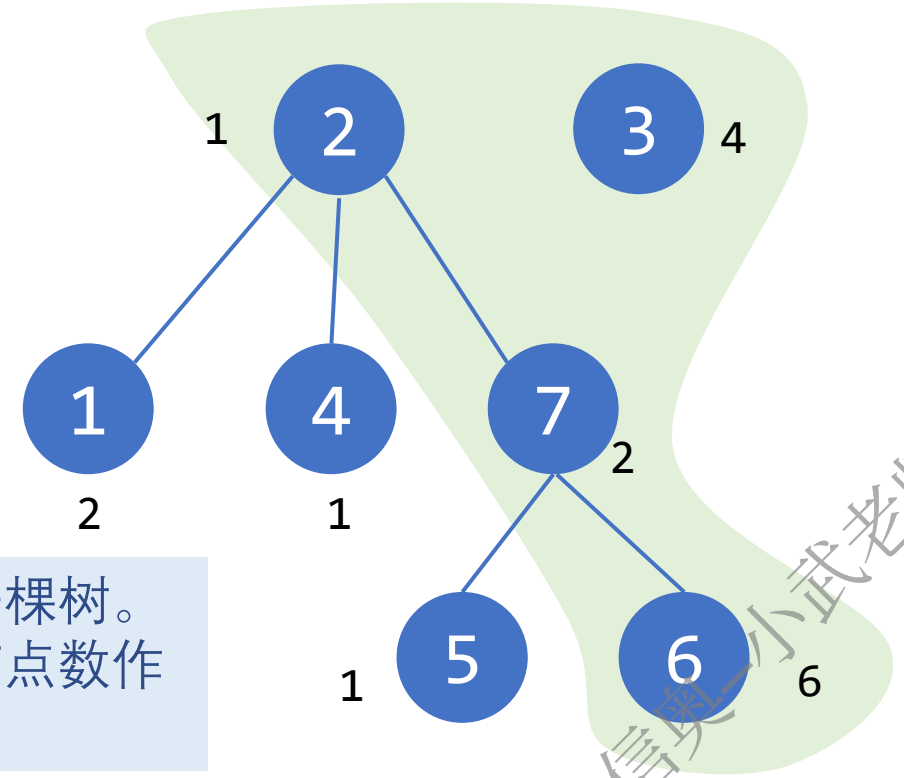
```
void dfs(int x) {
    for(long unsigned int i = 0; i < g[x].size(); i++){
        int y = g[x][i]; //节点x的儿子
        dfs(y);
        for(int j = m+1; j >= 1; j--){//DP
            for(int k = 0; k<j; k++){
                dp[x][j]=max(dp[x][j],dp[y][k]+dp[x][j-k]);
            }
        }
    }
}
```

01背包和完全背包的区别?

P0324. 攻克城堡

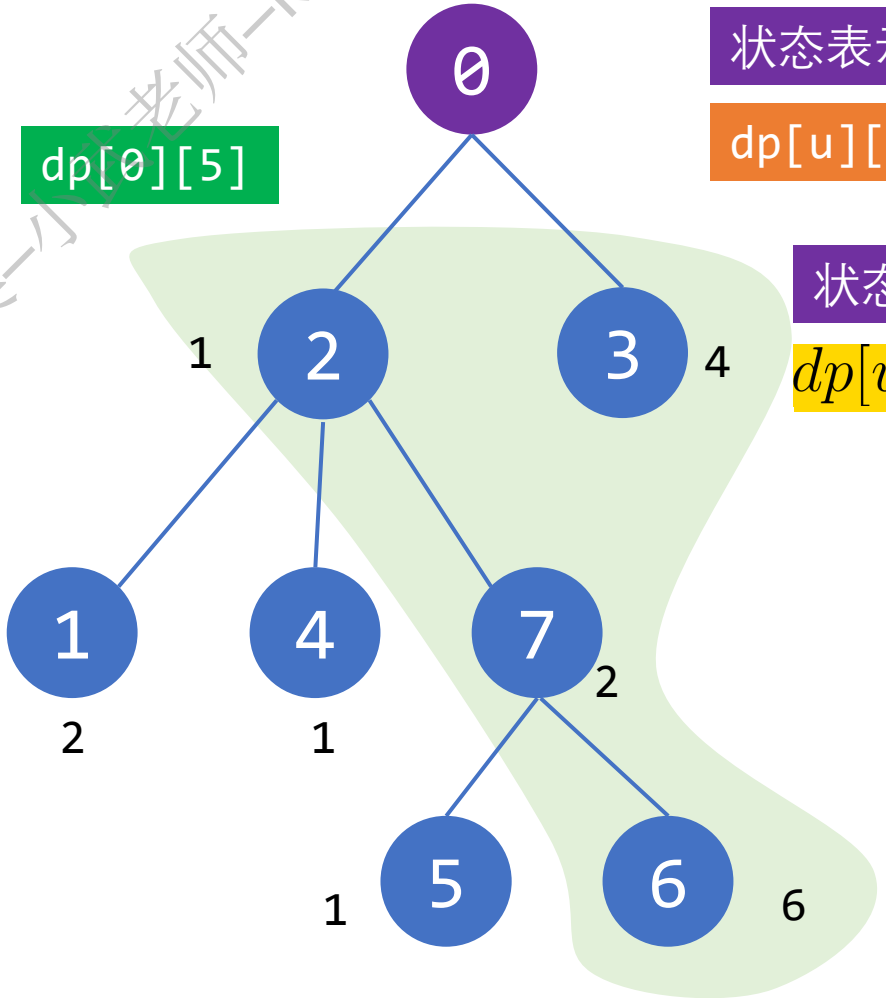
分析：本题属于背包类树形DP，在城堡数量有限制的情况下获得的宝物数量最多，只是加个拓扑限制。





可以添加一个0号节点作为超根，转化为一棵树。将节点编号作为状态的第1维，将选择的节点数作为状态的第二维。

P0324. 攻克城堡

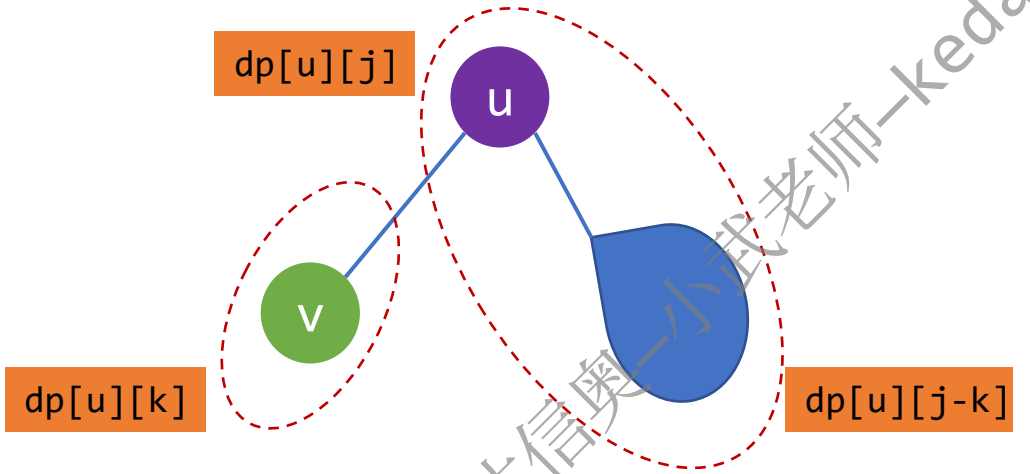


状态表示：

dp[u][j]表示以u为根的子树中选择j个节点获得的最大和值。

状态转移方程：

$$dp[u][j] = \max(dp[u][j], dp[v][k] + dp[u][j-k]);$$



P0324. 攻克城堡

dp[u][j]表示以u为根的子树中选择j个节点获得的最大和值。

u:以u为根节点

m: 选出m个城堡，背包最大容量

i: 节点u的第i个儿子

j: 相当于背包容量（从m到1），

k: 第i个儿子里选k个城堡，剩下的选j-k个城堡

P0324. 攻克城堡

```
int main() {
    while(cin>>n>>m, n+m){
        memset(dp, 0, sizeof(dp));
        for(int i = 0; i <= n; i++)
                g[i].clear();
        for(int i = 1; i <= n; i++){
            int x, y;
            cin >> x >> y;
            dp[i][1] = y;
            g[x].push_back(i);
        }
        dfs(0);
        cout << dp[0][m+1] << endl;
    }
    return 0;
}
```

```
void dfs(int x) {
    for(int i = 0; i < g[x].size(); i++){
        int y = g[x][i];
        dfs(y);
        for(int j = m+1; j >= 1; j--){//DP
            for(int k = 1; k<j; k++){
                dp[x][j]=max(dp[x][j],dp[y][k]+dp[x][j-k]);
            }
        }
    }
}
```

P0324. 攻克城堡

方法二

```cpp
int main() {
    int N, M;
    while (scanf("%d%d", &N, &M), N + M) {
        for (int i = 0; i <= N; i++) {
            E[i].clear();
        }
        memset (dp, 0, sizeof(dp));
        M++;
        val[0] = 0;
        int u;
        for (int i = 1; i <= N; i++) {
            scanf("%d%d", &u, &val[i]);
            E[u].push_back(i);
        }
        dfs(0, M);
        printf("%d\n", dp[0][M]);
    }
    return 0;
}
```

```cpp
void dfs(int u, int M) {
    dp[u][1] = val[u];
    for (int i = 0; i < E[u].size(); i++) {
        int v = E[u][i];
        dfs(v, M - 1);
        for (int j = M; j >= 1; j--) {
            for (int k = 1; k < j; k++) {
                dp[u][j] = max(dp[u][j], dp[v][k] + dp[u][j - k]);
            }
        }
    }
}
```

课后习题与实验

Talk is cheap, show me the code !

下节课见啦！