21级实验室暑假第十讲

目录

状压dp

状态dp表示的不是一个有意义的数值，例如前面章节的花费、价值、长度等，而是代表了**集合的数量**。
这种处理复杂集合问题的DP，叫做状态压缩DP。
集合的状态很多，操作复杂，往往把方案用二进制数（“压缩”到这个二进制数中）来表示和操作。二进制操作有与、或、取反、移位（注意使用的时候**注意其优先级顺序**，记不清建议都加上括号）等。

状压dp

小国王

在n×n(1≤n≤10) 的棋盘上放k(0≤k≤n^2) 个国王

国王可攻击**相邻的 8 个格子**，求使它们 **无法互相攻击** 的 方案总数

八相邻:
通常意义上的八相邻指的是当前元素的上、下、左、右、左上、右上、左下、右下八个方向

这种 **棋盘放置类** 问题，在没有事先知道一些特定 性质 的情况下来做，都会想到 **爆搜**

本题的数据规模，也是向着 爆搜 去设置的

如果我们直接 **爆搜**，则 时间复杂度 为 $O(2^{n^2})$是会超时的，因此会想到用 **记忆化搜索** 来进行优化

状压dp

小国王

在$n \times n(1 \le n \le 10)$ 的棋盘上放$k(0 \le k \le n^2)$ 个国王

国王可攻击**相邻的 8 个格子**，求使它们 **无法互相攻击** 的 方案总数

考虑一下如何进行 **动态规划**

由于在第 i 层放置国王的行为，受到 i−1 层和 i+1 层以及 i 层的状态影响

那么我们就可以规定从上往下枚举的顺序，这样考虑第 i 层状态时，只需考虑 i−1 层的状态即可

于是乎我们可以考虑把层数 i 作为动态规划的 **阶段** 进行 **线性DP**

而**第 i 阶段需要记录的就是前 i 层放置了的国王数量 j，以及在第 i 层的 棋盘状态 k**

状压dp

小国王

在n×n(1≤n≤10) 的棋盘上放k(0≤k≤n^2) 个国王

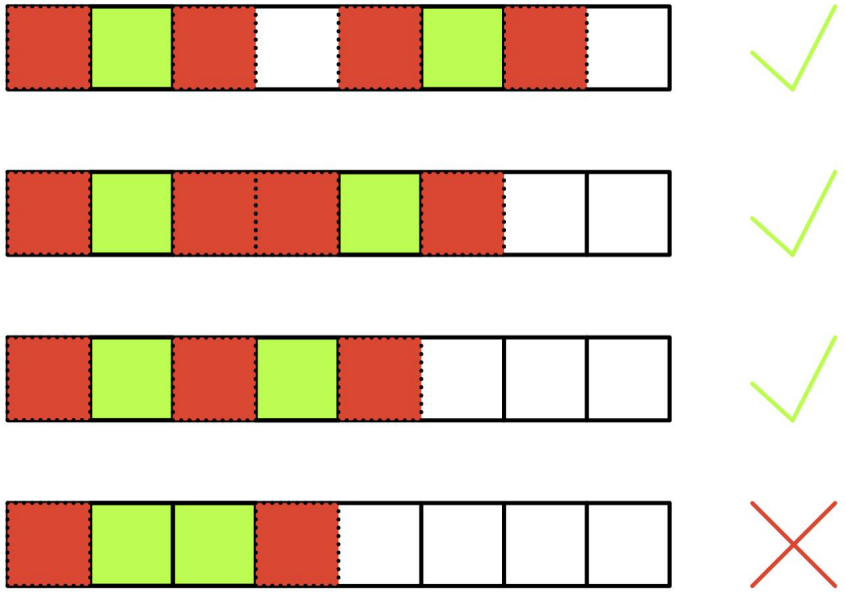
国王可攻击**相邻的 8 个格子**，求使它们 **无法互相攻击** 的 方案总数

这里先分析一下，哪些 **棋盘状态** 是合法的，哪些 **棋盘状态的转移** 是合法的

合法的棋盘状态

如右图所示，绿色方块为摆放国王的位置，红色方块为**王的 攻击范围**

只要任意王之间只要 **不相邻**，就是合法的状态



状压dp

小国王

在n×n(1≤n≤10) 的棋盘上放k(0≤k≤n^2) 个国王

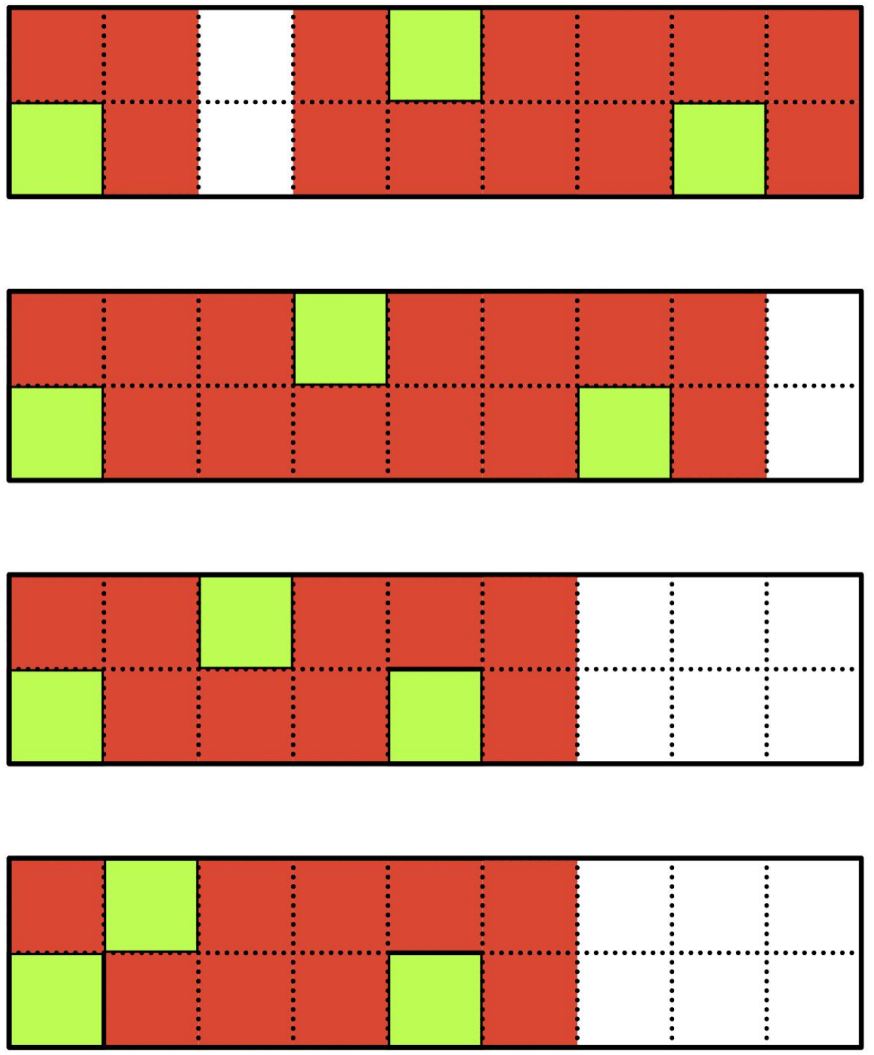
国王可攻击**相邻的 8 个格子**，求使它们 **无法互相攻击** 的 方案总数

这里先分析一下，哪些 **棋盘状态** 是合法的，哪些 **棋盘状态的转移** 是合法的

合法的棋盘转移状态

如右图所示，绿色方块为**摆放国王的位置**，红色方块为王的 攻击范围

只要**任意王的 纵坐标不相邻**，就是 合法的转移状态



状压dp

小国王

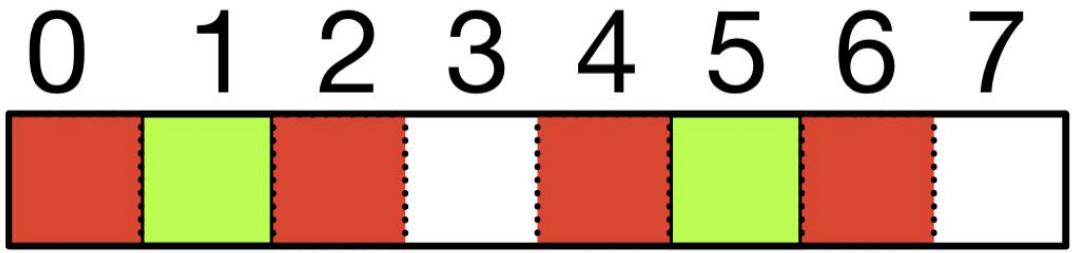
在$n \times n(1 \le n \le 10)$ 的棋盘上放$k(0 \le k \le n^2)$ 个国王

国王可攻击**相邻的 8 个格子**，求使它们 **无法互相攻击** 的 方案总数

我们可以用 二进制 来表示当前的状态

怎么用二进制表示状态?
若(state >> i) == 1 则表示在当前状态中，第 k(0≤i<n) 个位置放置了国王（下标从 0 开始）

再举一个简单的例子，见下图:



用二进制表示该状态，就是 $(00100010)_2$

状压dp

小国王

在n×n(1≤n≤10) 的棋盘上放k(0≤k≤n^2) 个国王

国王可攻击**相邻的 8 个格子**，求使它们 **无法互相攻击** 的 方案总数

状态表示：集合f[i,j,k]: 考虑前 i 层的棋盘，前 i 层放置了 j 个国王，且第 i 层**状态是 k(二进制)** 的方案

属性f[i,j,k]: 方案的总数 Sum

状态计算f[i,j,k]: $$f[i,j,k]=\sum_{pre} f[i-1,j-cnt_k,pre]$$

初始状态： f[0,0,0]=1

目标状态： $\sum f[n,k,st]$ (其中st为最后一行的合法状态)

其中pre是**枚举的能够与k合法存在于相邻行中的所有状态**，cnt_k表示状态k中的国王数量

状压dp

小国王

在n×n(1≤n≤10) 的棋盘上放k(0≤k≤n^2) 个国王

国王可攻击**相邻的 8 个格子**，求使它们 **无法互相攻击** 的 方案总数

状态表示：集合f[i,j,k]: 考虑前 i 层的棋盘，前 i 层放置了 j 个国王，且第 i 层**状态是 k(二进制)** 的方案

属性f[i,j,k]: 方案的总数 Sum

状态计算f[i,j,k]: $$f[i,j,k]=\sum_{pre} f[i-1,j-cnt_k,pre]$$

这样直接做，时间复杂度是 **O(nk2^n2^n)**是会超时的

但是我们可以通过预处理所有的 **合法状态**，以及**合法的 相邻转移状态**，以及 **合法状态 摆放的国王数量**

因为**虽然状态很多，但是 合法状态 并不多， 合法的转移状态 更不多**

状压dp

小国王

在n×n(1≤n≤10) 的棋盘上放k(0≤k≤n^2) 个国王

国王可攻击相邻的 8 个格子，求使它们 **无法互相攻击** 的 方案总数

必要函数

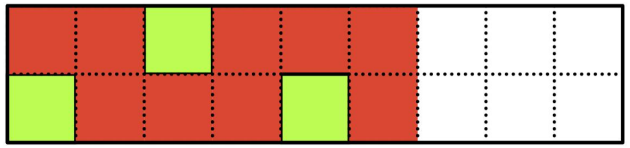
✓ 检测当前状态x是否不存在相邻的1

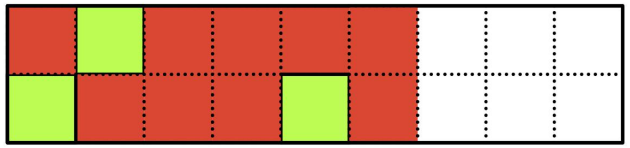
```
int check(int x){//检测x是否不存在相邻的1(存在相邻的1返回0,不存在返回1)
    for(int i=0;i<n;i++)
        if(((x>>i)&1)&&((x>>(i+1))&1))//存在相邻的1
            return 0;
    return 1;
}
```

✓ 统计当前状态x的1(放置)的个数

```
int count(int x){//统计状态x的1的个数
    int res=0;
    for(int i=0;i<n;i++)
        if((x>>i)&1)
            res++;
        //res+=(x>>i&1);//也可直接写成这种
    return res;
}
```

状压dp





小国王

在n×n(1≤n≤10) 的棋盘上放k(0≤k≤n^2) 个国王
国王可攻击相邻的 8 个格子，求使它们 **无法互相攻击** 的 方案总数

预处理阶段

```
vector<int> state;//保存所有合法状态
int cnt[MAXM];//统计每个状态的个数
vector<int> pre[MAXN];//保存当前状态的相邻合法状态的下标,注意不是相邻合法状态的值

for(int i=0;i<(1<<n);i++)
    if(check(i)==1){//是合法状态
        state.push_back(i);//保存合法状态
        cnt[i]=count(i);    //更新cnt值
    }
for(int i=0;i<state.size();i++)
    for(int j=0;j<state.size();j++){
        int a=state[i],b=state[j];
        if(((a&b)==0)&&check(a|b)==1)//注意要写成(a&b),不然先计算b==0(优先级问题)
            pre[i].push_back(j);//保存下标
    }
```

a&b表示检测状态之间是否直接相邻(有边相邻)，若为0，表示为无边相邻；
check(a|b)表示状态之间整合后是否有邻边

状压dp

小国王

在n×n(1≤n≤10) 的棋盘上放k(0≤k≤n^2) 个国王
国王可攻击相邻的 8 个格子，求使它们 **无法互相攻击** 的 方案总数

```
f[0][0][0]=1;
//循环n可写成n+1,此时直接输出f[n+1][m][0]的值
for(int i=1;i<=n;i++)//枚举每一层i
    for(int j=0;j<=m;j++)//枚举个数j
        for(int a=0;a<state.size();a++)//枚举当前状态
            for(int b=0;b<pre[a].size();b++){//枚举当前状态相邻的合法状态
                int temp_cnt=cnt[state[a]];
                if(j>=temp_cnt)//同时前i层的个数应大于等于当前行的个数
                    f[i][j][state[a]]+=f[i-1][j-temp_cnt][state[pre[a][b]]];
            }
ll res=0;
for(int i=0;i<state.size();i++)
        res+=f[n][m][state[i]];
printf("%lld\n",res);
//printf("%lld\n",f[n+1][m][0]);
```

执行阶段

虽然看着4层循环，但是通过**预处理所有的合法状态**，合法状态很少，合法的转移状态也少，故不会超时

状压dp

扩展: 玉米田

农夫约翰的土地由 M×N 个小方格组成，现在他要在土地里种植玉米。(1<=M,N<=12)

非常遗憾，部分土地是不育的，无法种植。

而且，**相邻的土地不能同时种植玉米**，也就是说种植玉米的所有方格之间都不会有公共边缘。

该题与小国王类似，它是距离长度为1的4个方向不能放置，故方法与小国王类似，当前第i行的状态是由上一行i-1的某些状态转移而来。

但是需注意的是，需考虑**田地中有不能种植的地方**，可通过**将当前行不能种植的地方用1表示，能种植的地方用0表示，形成一个二进制数，再将其与当前行的放置状态进行与运算**，如果结果不为0，存在不能种植的土地上进行了放置(存在冲突)。

状压dp

扩展: 炮兵阵地

司令部的将军们打算在 N×M 的网格地图上部署他们的炮兵部队。 **(N<=100，M<=10)**

一个 N×M 的地图由 N 行 M 列组成，地图的每一格可能是山地（用 H 表示），也可能是平原（用 P 表示），如下图。

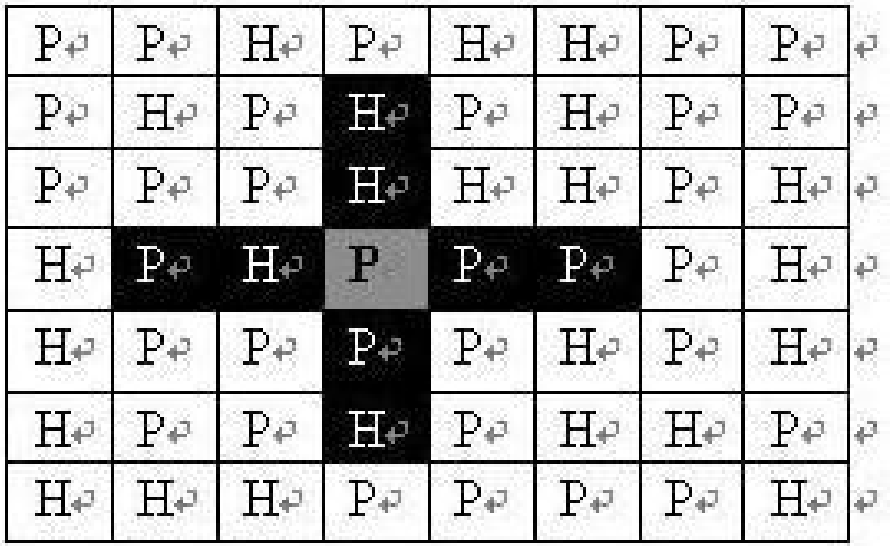
在每一格平原地形上最多可以布置一支炮兵部队（山地上不能够部署炮兵部队）；**一支炮兵部队在地图上的攻击范围如图中黑色区域所示**：

如果在地图中的灰色所标识的平原上部署一支炮兵部队，则**图中的黑色的网格表示它能够攻击到的区域：沿横向左右各两格，沿纵向上下各两格**。

图上其它白色网格均攻击不到。

从图上可见炮兵的攻击范围不受地形的影响。

现在，将军们规划如何部署炮兵部队，在防止误伤的前提下（保证任何两支炮兵部队之间不能互相攻击，即任何一支炮兵部队都不在其他支炮兵部队的攻击范围内），在整个地图区域内最多能够摆放多少我军的炮兵部队。



状压dp

扩展: 炮兵阵地

该题无非是玉米田的**一个再扩展**，它是距离长度为2的4个方向不能放置，此时当前第i行的状态是由上一行i-1和上上一行i-2的某些状态转移而来，故状态表示还需扩展一层，即f[i,k,l],该题未规定数量限制，可去掉前i行的数量j,k表示当前行i的状态,l表示当前行的状态

直接开所有的状态空间，**空间复杂度是 $O(N\times 2^M\times 2^M)$是会爆内存的**，故需要采用**滚动数组实现**

同时也需要考虑田地的问题，与玉米田采用的思想类似，此处不作解释

P	P	H	P	H	H	P	P
P	H	P	H	P	H	P	P
P	P	P	H	H	H	P	H
H	P	H	P	P	P	P	H
H	P	P	P	P	H	P	H
H	P	P	H	P	H	H	P
H	H	H	P	P	P	P	H

树形dp

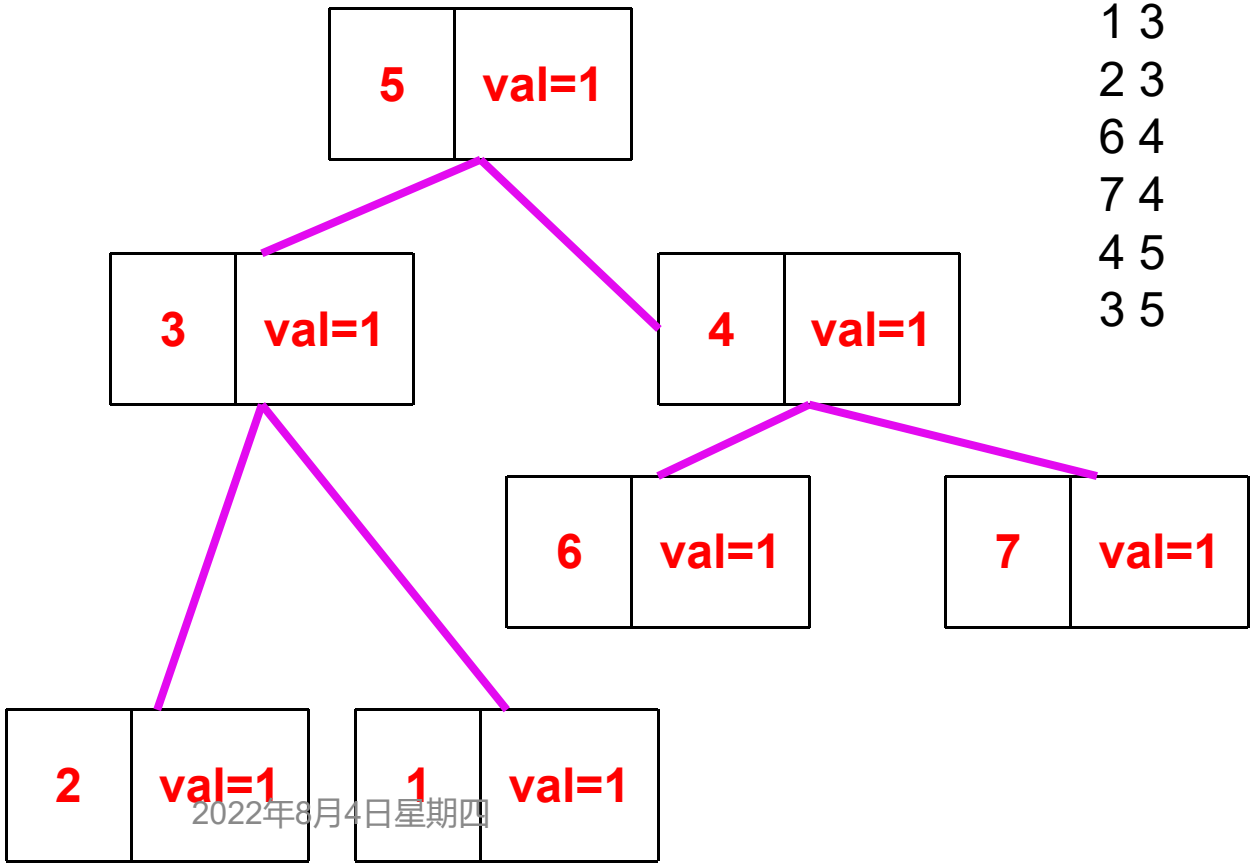
在树这种数据结构上进行的DP：给出一棵树，要求以最少的代价（或取得最大收益）完成给定的操作。

在树上做动态规划，显得非常合适：树本身有“子结构”性质（树和子树），具有递归性，符合DP的性质。

树形dp

没有上司的舞会

一棵有根树上，每个结点有一个权值。**相邻的父结点和子结点只能选择一个，问如何选择，使得总权值之和最大。**（邀请员工参加宴会，为了避免员工和直属上司发生尴尬，规定员工和直属上司不能同时出席。）



1 3
2 3
6 4
7 4
4 5
3 5

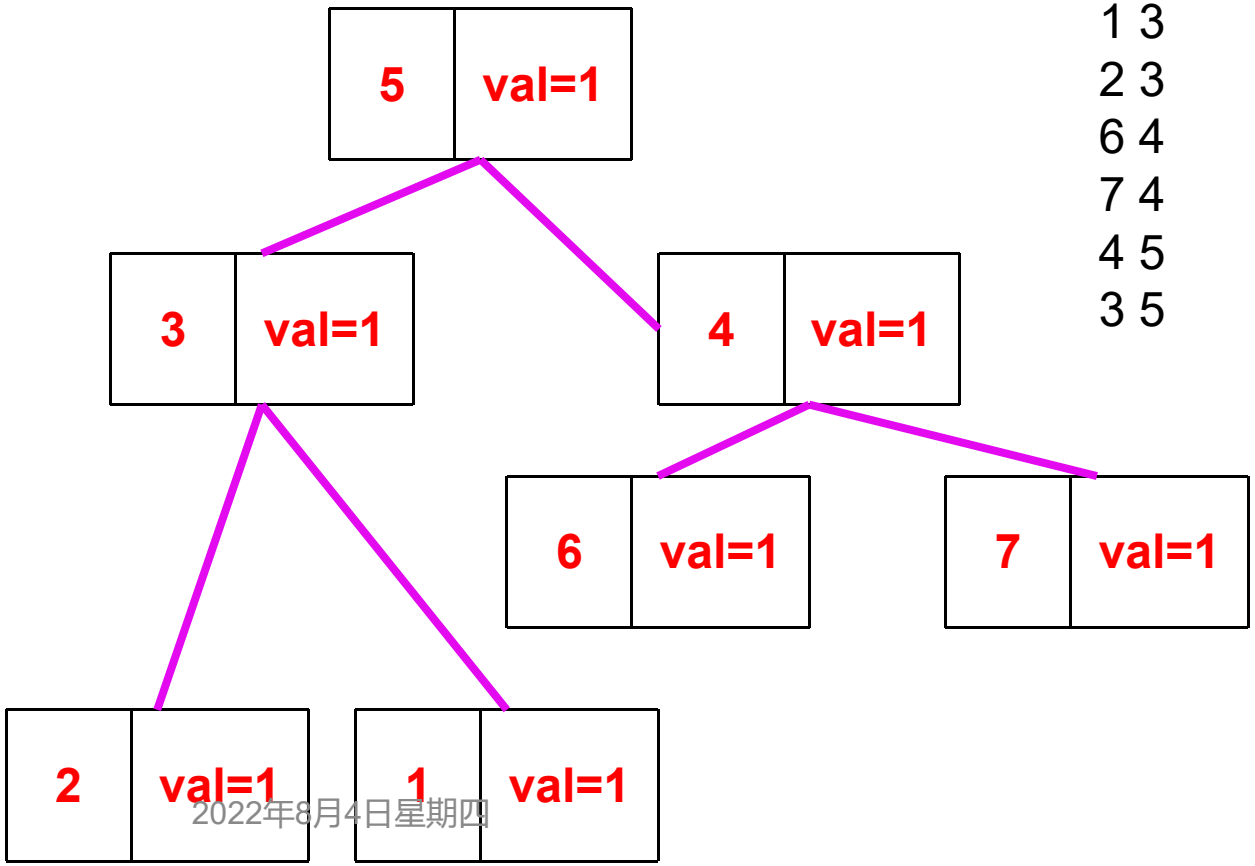
后者的数是前者的数的上司

结点选1、2、5、6、7时，有最大值5

树形dp

没有上司的舞会

一棵有根树上，每个结点有一个权值。**相邻的父结点和子结点只能选择一个，问如何选择，使得总权值之和最大。**（邀请员工参加宴会，为了避免员工和直属上司发生尴尬，规定员工和直属上司不能同时出席。）



1 3
2 3
6 4
7 4
4 5
3 5

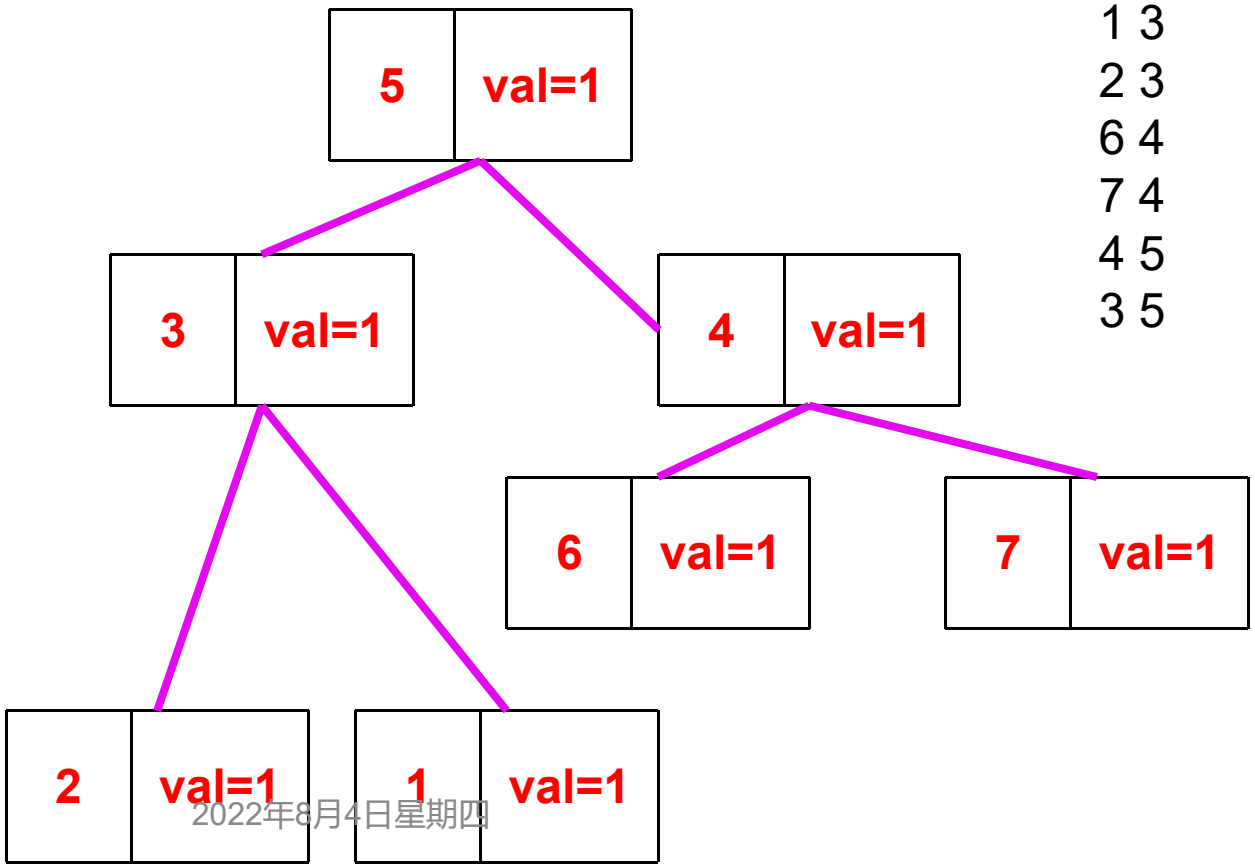
定义状态：
dp[i][0]：不选择当前结点的最优解；
dp[i][1]：选择当前结点时的最优解。

状态转移方程，有两种情况：
（1）不选择当前结点，那么**它的子结点可选可不选**，取其中的最大值：
dp[u][0] += max(dp[son][1], dp[son][0])
（2）选择当前结点，那么**它的子结点不能选**：
dp[u][1] += dp[son][0];

树形dp

没有上司的舞会

一棵有根树上，每个结点有一个权值。**相邻的父结点和子结点只能选择一个，问如何选择，使得总权值之和最大。**（邀请员工参加宴会，为了避免员工和直属上司发生尴尬，规定员工和直属上司不能同时出席。）



（1）建树。采用链式前向星，建立关系树。

（2）树的遍历。可以用DFS，从根结点开始进行记忆化搜索。

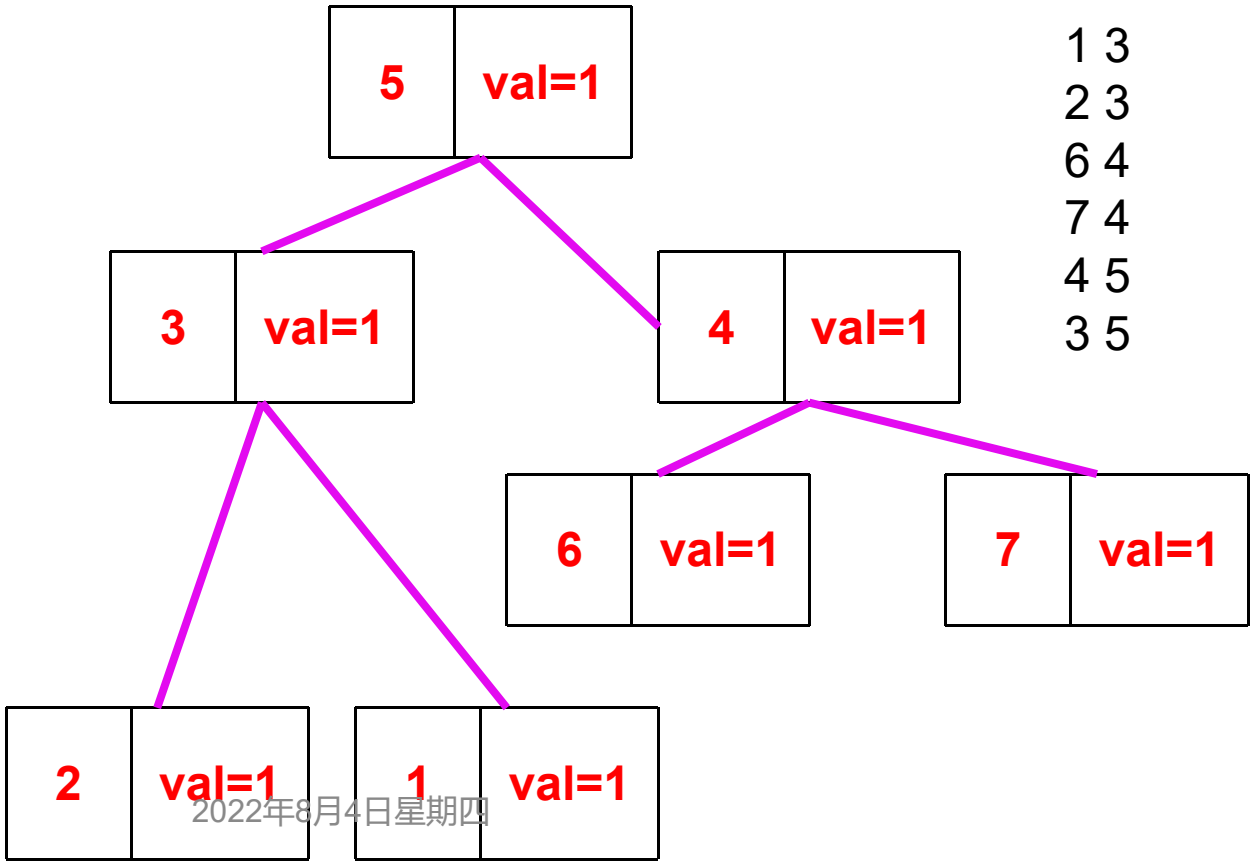
（3）DP。

复杂度：O(n)

树形dp

没有上司的舞会

一棵有根树上，每个结点有一个权值。**相邻的父结点和子结点只能选择一个，问如何选择，使得总权值之和最大。**（邀请员工参加宴会，为了避免员工和直属上司发生尴尬，规定员工和直属上司不能同时出席。）




```
1 3
2 3
6 4
7 4
4 5
3 5
```

```
void dfs(int root){
    f[root][1]=w[root];
    for(int i=head[root];i!=-1;i=nex[i]){
        int j=ed[i];
        dfs(j);
        f[root][0]+=max(f[j][0],f[j][1]);
        //当前子树的根不选,它的儿子可选可不选
        f[root][1]+=f[j][0];
        //当前子树的根选,它的儿子必不选
    }
}
```

树形dp

树的最长路径

给定一棵树，树中包含 n 个结点（编号1~n）和 n−1 条无向边，每条边都有一个权值。

现在请你找到树中的一条最长路径。

换句话说，要找到一条路径，使得使得路径两端的点的距离最远。

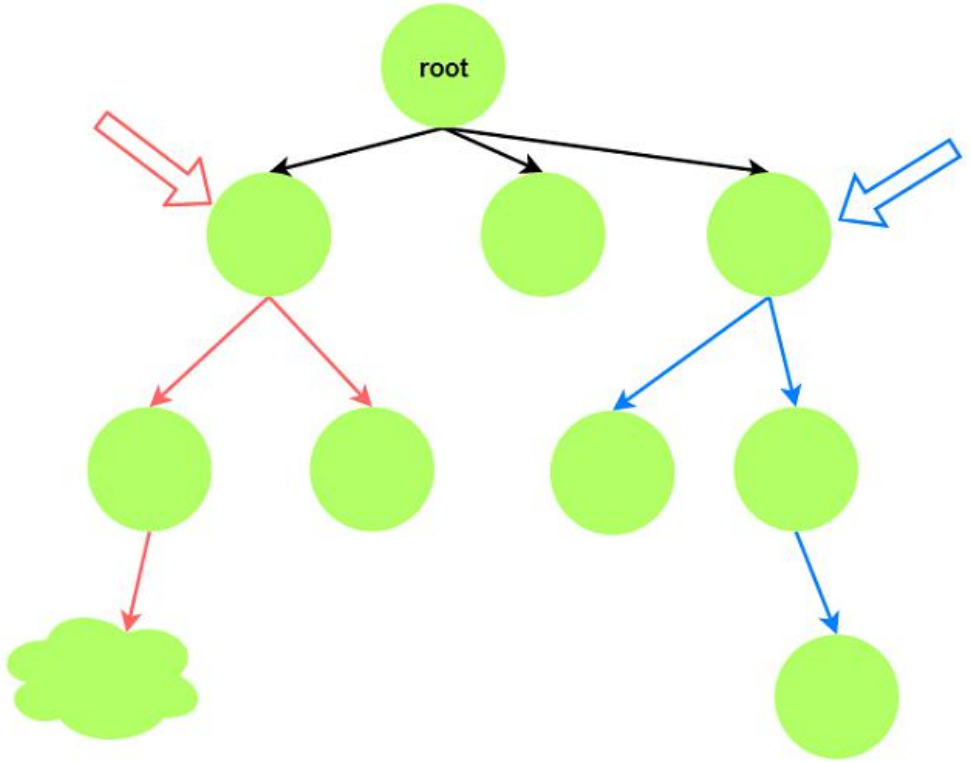
注意：路径中可以只包含一个点。

树形dp

树的最长路径

状态表示 dp(u)

- 集合：任意以一点作为树根后, 树节点相对根就有了高低之分.**dp(u)表示最高节点为u的所有路径集合**.
- 属性: Max路径长度的最大值.

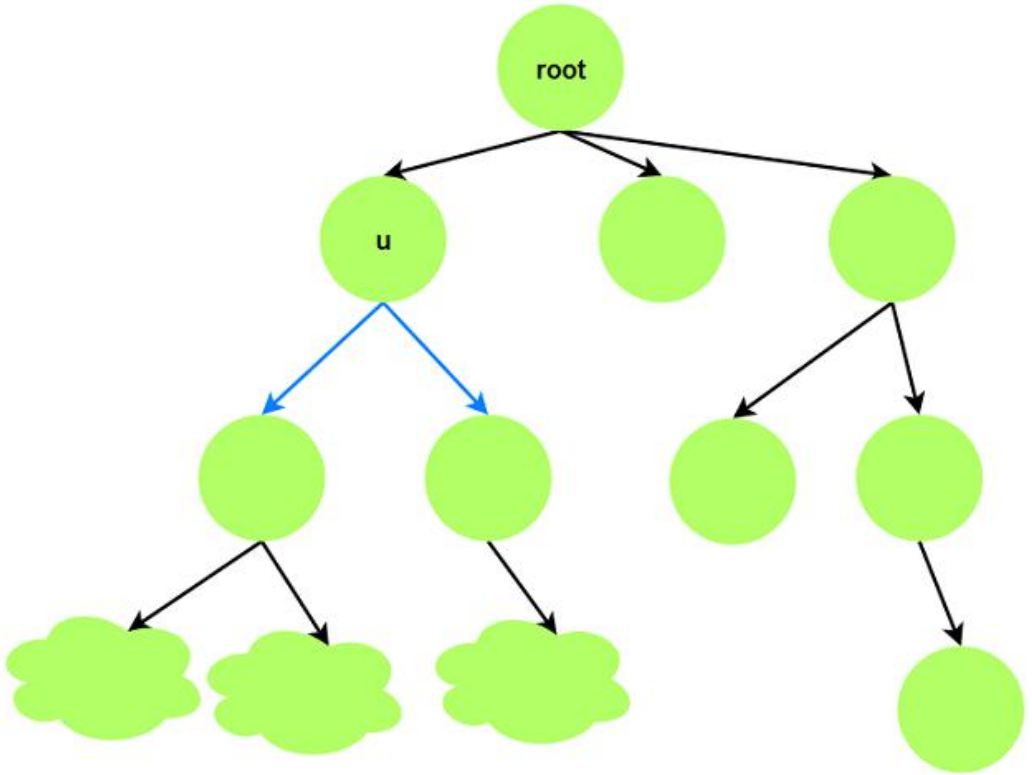


树形dp

树的最长路径

状态计算:

- ✓ 为保证当前节点u是路径最高点, 向下枚举以子节点为根到叶节点的最长路径.
- ✓ dp(u) 即**所有其子节点向下的路径的最长与次长值之和**, 若路径值为负数可以忽略.
- ✓ 在确定根后, 树的边应该为有向边. 为**防止节点“回头”向上搜索**, 在dfs过程中加入参数fa —当前节点的父节点.



树形dp

树的最长路径

状态计算:

- ✓ 为保证当前节点u是路径最高点, 向下枚举以子节点为根到叶节点的最长路径.
- ✓ dp(u) 即**所有其子节点向下的路径的最长与次长值之和**, 若路径值为负数可以忽略.
- ✓ 在确定根后, 树的边应该为有向边. 为**防止节点“回头”向上搜索**, 在dfs过程中加入参数fa —当前节点的父节点.

```
int dfs(int u,int fa){
    int dis=0;//从当前点往下走的最大长度
    int d1=0,d2=0;//d1表示最大值,d2表示次大值
    for(int i=head[u];i!=-1;i=nex[i]){
        int j=ed[i];
        if(j==fa)
            continue;
        int d=dfs(j,u)+val[i];
        dis=max(dis,d);
        if(d>=d1)//d>=最大值,更新最大值和次大值
            d2=d1,d1=d;
        else if(d>d2)//d>次大值,更新次大值
            d2=d;
    }
    res=max(res,d1+d2);//更新最终结果
    return dis;
}
```

树形dp

树的最长路径

扩展：关于无权树的直径(最长路径)求解

对于无权树(可认为边权均为1), 可以按如下方法求树的直径:

- ✓ **取任意一点a为起点, 找到距离a最远的一点u.**
- ✓ **找距离u最远的节点v.**

u→v的路径即为树的一条直径.

树形dp

树的最长路径

扩展：关于无权树的直径(最长路径)求解

证明：若u→v不是树的一条直径, 即u不是一条直径(最长路径)的端点.
假设树中直径c→d, 与树中直径a→u存在两者关系.

1. 二者无交点

树上任意两点联通, 所以从a存在一条到d的路径.

由于距离a最远的顶点是u(用$[x,y]$表示x到y的长度),故

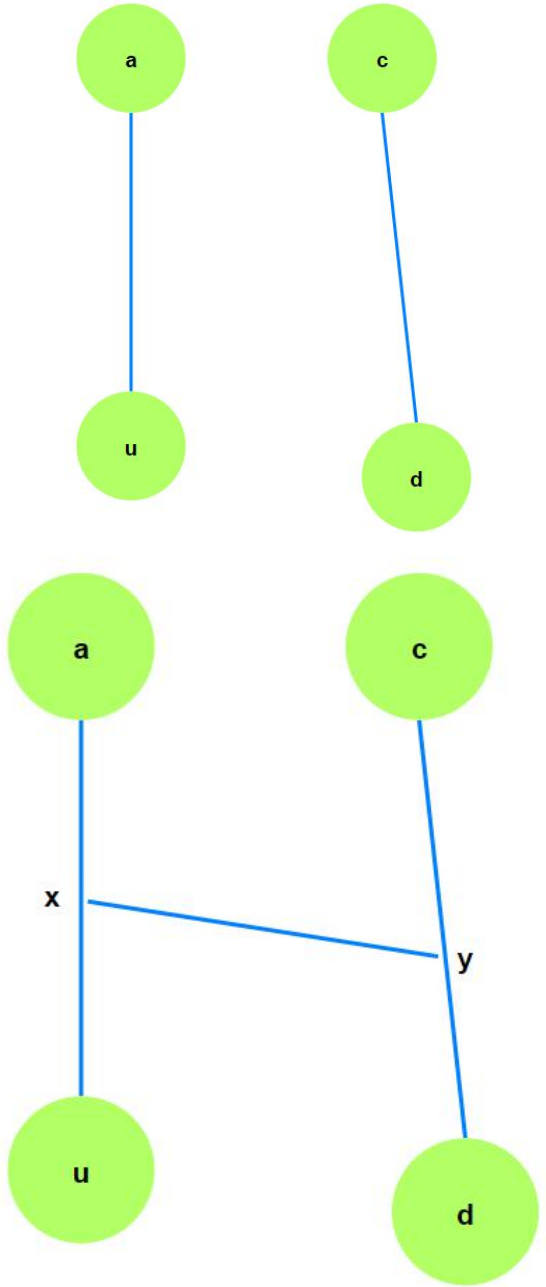
$$[a,x]+[x,u]\geq[a,x]+[x,y]+[y,d]$$

可推出$[x,u]\geq[x,y]+[y,d]$

该式移项$[x,u]-[x,y]\geq[y,d]$,此时$[x,u]+[x,y]\geq[y,d]$也成立

两边同时加上$[y,c]$可得到$[x,u]+[x,y]+[y,c]\geq[y,d]+[y,c]=[c,d]$,

由于$[y,d]+[y,c]$即树的直径$c\rightarrow d$,所以从u到c也是直径,即u作为直径的一个端点



树形dp

树的最长路径

扩展：关于无权树的直径(最长路径)求解

证明：若u→v不是树的一条直径, 即u不是一条直径(最长路径)的端点.
假设树中直径c→d, 与树中直径a→u存在两者关系.

2. 二者有交点

可以认为x→y退化为一个点. 可以用类似的方式证明.

$$[a,x]+[x,u]\geq[a,x]+[x,d]\Rightarrow[x,u]\geq[x,d]$$

两边同时加上$[x,c]$

$$[x,u]+[x,c]\geq[x,d]+[x,c]=[c,d]$$

故u亦可作为c的直径的另一个端点