动态规划2——树型dp、状压dp

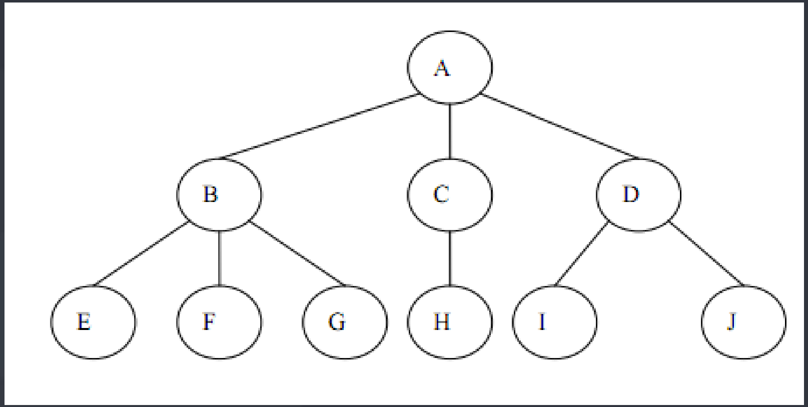
邓丝雨

树型dp

树型dp一般先算子树然后进行合并，在实现上与树的后序遍历（这个说法并不准确，因为其实很多都不是二叉树）类似——遍历子树，遍历完之后把子树的值合并给父亲。

复习：什么是树

- 父亲
- 儿子



引入

- 给你一棵n个点的树（1号点为根节点），求以点i为根的子树的大小

- f[i]以点i为根的子树的点的个数
- f[i] = 1+Σf[k] (k是i的儿子)

- Void dfs(i)
- {
-     If(i是叶子节点) f[i] = 1, 返回;
-     for (k 是i的儿子)
-     {
-         dfs(k);
-         f[i]+=f[k];
-     }
-     f[i]+=1;
- }

例1： NC15033 小G有一个大树

- 小G想要把自己家院子里的橘子树搬到家门口（QAQ。。就当小G是大力水手吧）
- 可是小G是个平衡性灰常灰常差的人，他想找到一个这个橘子树的平衡点。
- 怎么描述这棵树呢。。。就把它看成由一个个节点构成的树吧。结点数就代表树重。

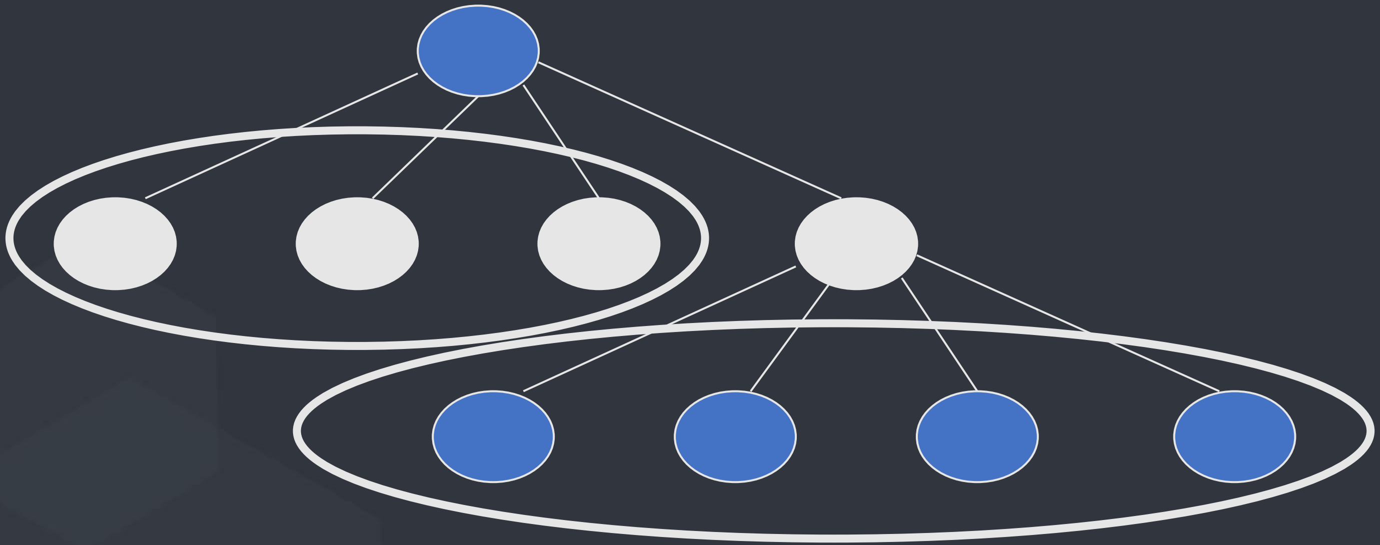
- 确定状态
- F[i]以将点i删掉以后最大联通块的大小
- 确定状态转移方程
- F[i] = max(n-tot[i], max(tot[k]))
- K是I 的儿子
- Tot[i]是以i为根的子树的大小

例2：NC51178 没有上司的舞会 （最大独立集）

- Ural大学有N名职员，编号为1~N。
- 他们的关系就像一棵以校长为根的树，父节点就是子节点的直接上司。
- 每个职员有一个快乐指数，用整数 HiHi 给出，其中 1≤i≤N,1≤i≤N。
- 现在要召开一场周年庆宴会，不过，没有职员愿意和直接上司一起参会。
- 在满足这个条件的前提下，主办方希望邀请一部分职员参会，使得所有参会职员的快乐指数总和最大，求这个最大值。

例2：NC51178 没有上司的舞会

- 简单的染色统计是不正确的



例2：NC51178 没有上司的舞会

- *确定状态*
- i选或者不选会影响子树的结果
- 用f[i][0]表示不选择i点时，i点及其子树能选出的最多人数，f[i][1]表示选择i点时，i点及其子树的最多人数。
- *确定状态转移方程*
- $f[i][0] = \Sigma(\max(f[j][0], f[j][1]))$
- $f[i][1] = 1 + \Sigma f[j][0]$
- (j是i的儿子！！)
- 边界：$f[i][0] = 0, f[i][1] = 1$ --------i是叶子节点
- 结果为$\max(f[root][0], f[root][1])$

例3： poj1463 NC106060 Strategic game （树的最小点覆盖）

- 一城堡的所有的道路形成一个n个节点的树，如果在一个节点上放上一个士兵，那么和这个节点相连的边就会被看守住，问把所有边看守住最少需要放多少士兵。

例3： poj1463 NC106060 Strategic game

- 确定状态
- f[x][1]以x为根的子树在x上放置的士兵的最少所需的士兵数目
- f[x][0]以x为根的子树x上不放置的士兵的最少所需的士兵数目
- 确定状态转移方程
- f[x][1] =1 + Σ min(f[i][0],f[i][1]) // x上放置的士兵，于是它的儿子们可放可不放！
- f[x][0] = Σ f[i][1] //x上不放置的士兵，它的儿子们都必须放！
- (i是x的儿子！！)
- 结果为min(f[root][0], f[root][1])

例3： poj1463 NC106060 Strategic game

```
void dfs(long x)
{
    v[x] = 1;
    for (long i=0; i<n; i++)
    {
       if ((!v[i]) && (b[x][i]))
        {
            dfs(i);
            f[x][0] += f[i][1];
            f[x][1] += min(f[i][0], f[i][1]);
        }
    }
}
```

例4： NC24953 Cell Phone Network （树的最小支配集）

- 给你一棵无向树，问你最少用多少个点可以覆盖掉所有其他的点。
- （一个点被盖，它自己和与它相邻的点都算被覆盖）

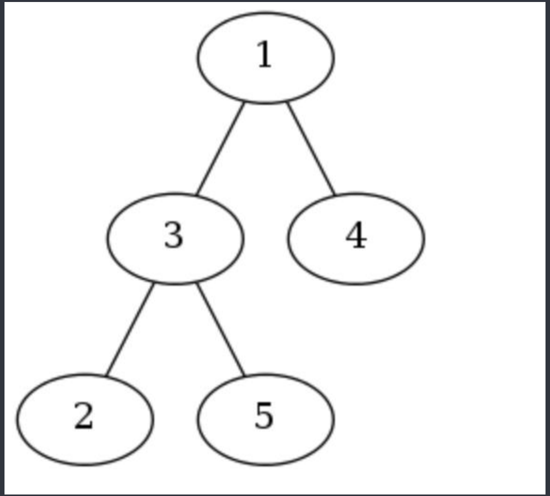
例4： NC24953 Cell Phone Network

- 确定状态
- 选他，选他儿子，选他父亲都对子树答案有影响
- dp[i][0]：选点i，并且以点i为根的子树都被覆盖了。
- dp[i][1]：不选点i，i被其儿子覆盖
- dp[i][2]：不选点i，被其父亲覆盖（儿子可选可不选）
- 确定状态转移方程
- dp[i][0]=1+Σmin(dp[u][0],dp[u][1],dp[u][2]) （u是i的儿子）
- dp[i][2]=Σ(dp[u][1],dp[u][0])
- 对于dp[i][1]的讨论稍微复杂一点——他的所有儿子里面必须有一个取dp[u][1]
- 那么：if(i没有子节点)dp[i][1]=INF else dp[i][1]=Σmin(dp[u][0],dp[u][1])+inc
- 其中对于inc有：
- if(上面式子中的Σmin(dp[u][0],dp[u][1])中包含某个dp[u][0])inc=0;
- else inc=min(dp[u][0]-dp[u][1])。

```
void dfs(int x)
{
    v[x] = 1;
    f[x][0] = 1;  f[x][1] = f[x][2] = 0;
    int tmp = INF;
    bool flag = 1;
    for (int i = head[x]; i != 0; i = edge[i].next)
    {
        int u = edge[i].t;
        if ((!v[u]))
        {
            dfs(u);
            f[x][2] += min(f[u][1], f[u][0]);
            f[x][0] += min(min(f[u][0], f[u][1]), f[u][2]);
            if(f[u][0] <= f[u][1])
            {
                flag = false;
                f[x][1] += f[u][0];
            }
            else
            {
                f[x][1] += f[u][1];
                tmp = min(tmp, f[u][0] - f[u][1]);
            }
        }
    }
    if(flag)  f[x][1] += tmp;
}
```

例5：NC50505 二叉苹果树

- 有一棵二叉苹果树，如果数字有分叉，一定是分两叉，即没有只有一个儿子的节点。这棵树共N个节点，标号1至N，树根编号一定为1。
- 我们用一根树枝两端连接的节点编号描述一根树枝的位置。一棵有四根树枝的苹果树，因为树枝太多了，需要剪枝。但是一些树枝上长有苹果，给定需要保留的树枝数量，求最多能留住多少苹果。



例5：二叉苹果树

- 确定状态
- f[u][j]表示在以u为根的子树保留j个分支可以得到的最大苹果数量
- 确定状态转移方程

变形：

- 如果是多叉树怎么办？
- ——树上背包
- $F[u][j] = \max(f[u][k] + f[v][j - k - 1] + W)$
- v分别是u的儿子，w为u到v边上的苹果数目，k属于[0, j]
- （类似于背包的思想）

```
void dfs(int u){
    vis[u]=1;
    int i,v,w,j,k,son=0;
    for(i=head[u];i!=-1;i=e[i].next) {
        v=e[i].ed;w=e[i].w;
        if(vis[v]==1)continue;
        dfs(v);
        for(k=m;k>=1;k--) {
            for(j=1;j<=k;j++)//在v节点的子树中选择j条边
                if(f[u][k]<f[u][k-j]+f[v][j-1]+w)
                    f[u][k]=f[u][k-j]+f[v][j-1]+w; //u与v有一条边，所以加上dp[v][j-1]
        }
    }
}
```

例6：NC202475树上子链（树的直径）

- 给定一棵树 T ，树 T 上每个点都有一个权值。
- 定义一颗树的子链的大小为：这个子链上所有结点的权值和 。
- 请在树 T 中找出一条最大的子链并输出。

状压dp

状态压缩只是一种手段，一种直观而高效地表示复杂状态的手段

复习：状压dp预备知识——位运算

- << 左移
- >> 右移
- |　或
- &　和
- ^　异或——两个值相同为0， 不同为真

位运算基础

- 去掉最后一位
- x >> 1
- 在最后加一个0
- x << 1
- 在最后加一个1
- (x<< 1)+1
- 把最后一位变成1
- x | 1
- 把最后一位变成0
- (x | 1) - 1
- 最后一位取反
- x ^ 1
- 把右数第k位变成1
- x | (1 << (k-1))
- 把右数第k位变成0
- x & (~ (1 << (k-1)))
- 右数第k位取反
- x xor (1 shl (k-1))

位运算基础

- 取末k位
- x & ((1<<k)-1)
- 取右数第k位
- (x >> (k-1)) & 1
- 把末k位变成1
- x | ((1 << k)-1)
- 末k位取反
- x ^ ((1 << k)-1)
- 把右边连续的1变成0
- (x & (x+1))
- 把右起第一个0变成1
- x | (x+1)
- 把右边连续的0变成1
- x | (x-1)
- 取右边连续的1
- (x ^ (x+1)) >> 1
- 去掉右起第一个1的左边
- x&(-x)

引入：

- 在n*n(n≤20)的方格棋盘上放置n个车(可以攻击所在行、列)，求使它们不能互相攻击的方案总数。
- 给大家30秒时间思考！

引入：

- 我们一行一行放置，则第一行有n种选择，第二行n-1，……，最后一行只有1种选择，根据乘法原理，答案就是n!
- 可是如果这个题变化一下呢？还能用排列组合么？

例1:

- 在n*n(n≤20)的方格棋盘上放置n个车(可以攻击所在行、列)，某些格子不能放，求使它们不能互相攻击的方案总数。

例1:

- 我们知道这个车是需要一行一行放的
- 取棋子的放置情况作为状态，某一列如果已经放置棋子则为1，否则为0。这样，一个状态就可以用一个最多20位的二进制数表示。
- 例如n=5,第1、3、4列已经放置，则这个状态可以表示为01101(从右到左)。
- 那么，令f[i][st]表示前i行的状态为st的方法数
- f[i][st] = Σf[i- 1][st’]　(第i行放在第j列)
- 保证满足 (st ’ &(1 << (j - 1))) == 0 且 st’ + (1 << (j - 1)) == st

例2： NC20240 [SCOI2005]互不侵犯

- 在N×N的棋盘里面放K个国王，使他们互不攻击，共有多少种摆放方案。国王能攻击到它上下左右，以及左上左下右上右下八个方向上附近的各一个格子，共8个格子。
- 1 <=N <=9, 0 <= K <= N * N

例2： NC20240 [SCOI2005]互不侵犯

- 你可以搜索……但是一共最多81个格子，100%tle……
- 一般的动如果规也解决不了这个问题，因为一个格子一个格子地扩展，不满足无后效性……
- 但是如果一行一行扩展呢?
- 根据上一行的状态我们自然就能知道当前行的状态是否合法
- 那么第一个问题就是如何表示每一行的状态了……
- 由于每一行的格子小于等于9个，每一个格子又只有两种状态——放国王和不放国王，假设我们把放了国王记为1，没有放国王记为0，很显然每一行的状态就是一个01串，如果把这个串看成一个二进制数，那么它显然不会超过（111111111），即2^9-1

例2： NC20240 [SCOI2005]互不侵犯

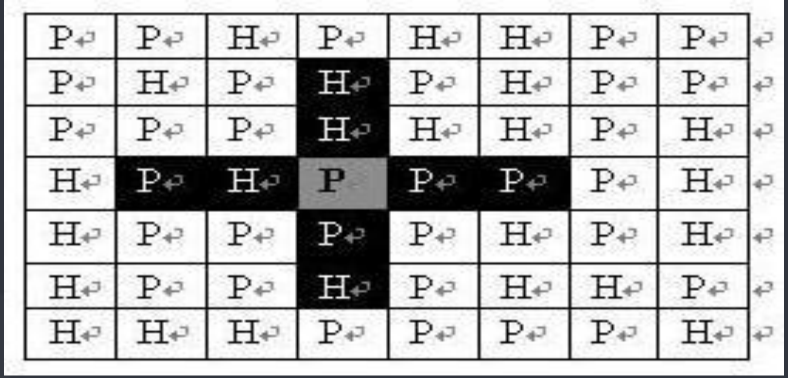
- 那么我们就完全可以拿一个int类型的数来表示每一行的状态。然后，怎样的状态才是合法的呢?
- 没有相邻的两个1!
- 如何判断?
- (x & (x<<1))==0
- 友情提示：位运算的优先级可能出乎你的意料，一定记得加括号！！
- 如果已知上一行的状态是x当前行状态是y，xy满足什么条件才是合法的?
- (x & y) == 0 (x & (y << 1)) == 0 (x & (y >> 1)) == 0

例2： NC20240 [SCOI2005]互不侵犯

- 确定状态
- f[i][j][k]表示第i行的状态为k 且已经放了j个国王的方案数
- 确定状态转移方程
- f[i][j][k] += f[i - 1][j - num[k]][p]　（ (k & p) == 0 ， (x & (p << 1)) == 0 (x & (p >> 1)) == 0
- 且k和p都合法
- (num[k]表示k状态的国王数)

例3：NC16886 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。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：



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

例3： NC16886 炮兵阵地

- 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。

例3： NC16886 炮兵阵地

- 确定状态
- 由于每一个炮都可以打到两行，所以每一行的放置方法都与他放置的情况有关
- 所以 f[i][j][k] 表示第i行为状态j，第i - 1行为状态为k时所用的最大炮兵数
- 确定状态转移方程
- 所以 f[i][j][p] = max(f[i][j][p]，f[i - 1][p][q] + num[j])
- q和p和j均不发生冲突
- Pqj均为符合要求的状态，即任意1左右两边两位都不是1判断条件是((i & (i >> 1)) == 0) && ((i & (i >> 2)) == 0)，且为1的地方都是平原（也用位运算判断）

例4：TSP问题 NC16122郊区春游 NC16544简单环

- 旅行商问题（Traveling Saleman Problem，TSP）又译为旅行推销员问题、货郎担问题，简称为TSP问题
- 给你一张图（你可以认为是抽象了的地图，由若干点和边组成），求从某个起点出发，经过所有的点的最短路径。

例4：TSP问题

- 确定状态
- F[st][i]表示当前状态为st，最后到达的一个点是i，所经过的最短距离
- 确定状态转移方程
- $F[st][i] = minn(f[st'][j] + a[j][i])$
- 其中 $st' + (1<<(j-1)) == st$

扩展：

- 如果要求走完所有的点回到原点怎么办？

例5： NC15832   Most Powerful

- 不超过10种气体，两两之间相互碰撞可以产生一定的能量，如a碰b，那么b气体就消失，自身不能碰自身，问最后所能得到的最大能量。

例5： NC15832 Most Powerful

- 第一步：确定状态
- F[i]表示状态为i时所能获得的最大能量
- i的第k位若等于1 则表示第k个气体已经用了并消失了，为0则没有用或是用了没消失。
- 第二步：确定状态转移方程
- F[k | (1 << (i-1))]=max(f[k]+a[j][i])

例6： 棋盘覆盖 poj2411 NC107008 Mondriaan's Dream

- 一个N*M的矩阵（N,M<=15)，用1*2和2*1的砖块密铺，问：有多少种方法?

综合分析：

- 纵观上文讨论的题目，几乎都是普普通通的一个递推公式或者状态转移方程，只不过其中的一维或多维是“压缩的”，即把一个状态(一个方案、一个集合等)压缩成一个整数。
- 这很明显是一个Hash的过程，所以状压DP又被称为Hash DP或集合DP。
- 除去这一点区别，我们发现它和普通递推/DP没有什么差别，都是从已有的状态推知新的状态，所做的决策都没有后效……那我们为什么要用状态压缩？
- 因为一般的状态没办法满足我们的需求，我们的一个状态中包含了很多的信息，而这些信息又可以压缩成一个二进制数（或者是三进制四进制数……）

综合分析：

- 看到什么样的问题的时候考虑用状压dp?
- 在棋盘格子上的覆盖
- 类似于TSP的路径问题
- N比较小，变成n位二进制后还很合理……

一些题目

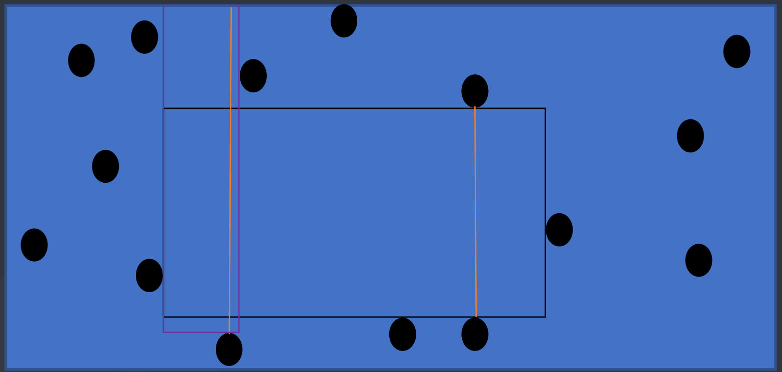
例1：最大全0子矩形

- 在一个0,1方阵中找出其中最大的全0子矩阵。
- 010010
- 100010
- 001000
- 111000

例1：最大全0子矩形

- 解法一：枚举上下左右四个边界，判断中间有没有1
- 解法二： 枚举左右边界，对处在边界内的1按y排序，每两个相邻的点和左右边界组成一个矩形

- **悬线法：**



```
    for(int i = 1; i <= n; i++)
    {
        for(int j = 1; j <= n; j++)
        {
            if(a[i][j]) h[i][j] = l[i][j] = 0;
            else
            {
                h[i][j] = h[i-1][j] + 1;
                l[i][j] = l[i][j-1] + 1;
            }
        }
        for(int j = n; j > 0; j--)
        {
            if(a[i][j]) r[i][j] = 0;
            else r[i][j] = r[i][j+1] +1;
        }
    }
    for (int i = 1; i <= n; i++)
    for (int j = 1; j <= n; j++)
    {
        if (h[i][j] > 1)
        {
            l[i][j] = min(l[i][j], l[i-1][j]);
            r[i][j] = min(r[i][j], r[i-1][j]);
        }
         ans = max(ans, (r[i][j] + l[i][j] - 1) * h[i][j]);
    }
```

变形1：最大全0子正方形

变形2：棋盘制作

- 国际象棋是世界上最古老的博弈游戏之一，和中国的围棋、象棋以及日本的将棋同享盛名。据说国际象棋起源于易经的思想，棋盘是一个8*8大小的黑白相间的方阵，对应八八六十四卦，黑白对应阴阳。
- 而我们的主人公小Q，正是国际象棋的狂热爱好者。作为一个顶尖高手，他已不满足于普通的棋盘与规则，于是他跟他的好朋友小W决定将棋盘扩大以适应他们的新规则。
- 小Q找到了一张由N*M个正方形的格子组成的矩形纸片，每个格子被涂有黑白两种颜色之一。小Q想在这种纸中裁减一部分作为新棋盘，当然，他希望这个棋盘尽可能的大。
- 不过小Q还没有决定是找一个正方形的棋盘还是一个矩形的棋盘（当然，不管哪种，棋盘必须都黑白相间，即相邻的格子不同色），所以他希望可以找到最大的正方形棋盘面积和最大的矩形棋盘面积，从而决定哪个更好一些。
- 于是小Q找到了即将参加全国信息学竞赛的你，你能帮助他么？

- 法1：修改转移条件（前一个状态和自己颜色不同时转移）
- 法2：棋盘黑白染色颠倒01

例2:NC16615 传纸条

- 小渊和小轩是好朋友也是同班同学，他们在一起总有谈不完的话题。一次素质拓展活动中，班上同学安排做成一个m行n列的矩阵，而小渊和小轩被安排在矩阵对角线的两端，因此，他们就无法直接交谈了。幸运的是，他们可以通过传纸条来进行交流。纸条要经由许多同学传到对方手里，小渊坐在矩阵的左上角，坐标(1,1)，小轩坐在矩阵的右下角，坐标(m,n)。从小渊传到小轩的纸条只可以向下或者向右传递，从小轩传给小渊的纸条只可以向上或者向左传递。
- 在活动进行中，小渊希望给小轩传递一张纸条，同时希望小轩给他回复。班里每个同学都可以帮他们传递，但只会帮他们一次，也就是说如果此人在小渊递给小轩纸条的时候帮忙，那么在小轩递给小渊的时候就不会再帮忙。反之亦然。

- 还有一件事情需要注意，全班每个同学愿意帮忙的好感度有高有低（注意：小渊和小轩的好心程度没有定义，输入时用0表示），可以用一个0-100的自然数来表示，数越大表示越好心。小渊和小轩希望尽可能找好心程度高的同学来帮忙传纸条，即找到来回两条传递路径，使得这两条路径上同学的好心程度只和最大。现在，请你帮助小渊和小轩找到这样的两条路径。
- 1<=m,n<=50

例3：NC 210520 Min酱要旅行

- 从前有个富帅叫做Min酱，他很喜欢出门旅行，每次出门旅行，他会准备很大一个包裹以及一大堆东西，然后尝试各种方案去塞满它。
  然而每次出门前，Min酱都会有个小小的烦恼。众所周知，富帅是很讨妹子喜欢的，所以Min酱也是有大把大把的妹子，每次出门都会有一只妹子随行。然而这些妹子总是会非常排斥Min酱准备的众多东西中的一件（也许是因为这件东西是其它妹子送给Min酱的），这件东西Min酱是万万不敢带上的，否则的话……嘿嘿嘿。另外，妹子们嫌Min酱的包裹太丑了，会自带一个包裹去换掉Min酱的包裹。
  Min酱是个控制欲很强的人，然而这样一来，Min酱就不知道可以用多少种方案去填充包裹了，所以Min酱很郁闷。
  于是Min酱找到了聪明的你，希望你能帮助他解决这些问题。
  另外，Min酱是个典型的懒人，他不希望每次带不同的妹子出去都麻烦你，所以他希望你能给出有K1..Kn件物品，第 i件不能带并且包裹大小为 1..M的所有方案数。

例4：NC210249打砖块(brike)

- 在一个凹槽中放置了n 层砖块，最上面的一层有n 块砖，第二层有n–1块，……最下面一层仅有一块砖。第 i 层的砖块从左至右编号为1，2,…i，第i层的第j块砖有一个价值a[i,j]（a[i,j]≤50）。下面是一个有5层砖块的例子：如果你要敲掉第 i层的第j 块砖的话，若 i=1，你可以直接敲掉它，若 i>1，则你必须先敲掉第 i–1 层的第j和第j+1 块砖。你的任务是从一个有n（n≤50）层的砖块堆中，敲掉 m(m≤500)块砖，使得被敲掉的这些砖块的价值总和最大。