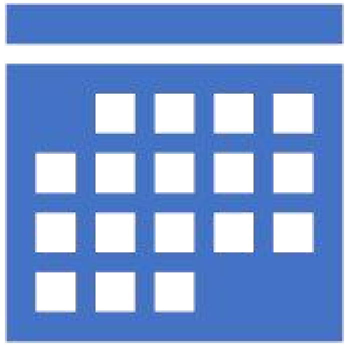
程序设计实训

南京大学智能科学与技术学院 史桀绮

课程形式

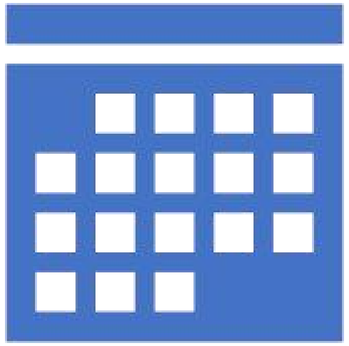


1-4周，6-9周每周二7-8节

南雍楼

每周二课上布置本周练习题，周六截止

课程形式

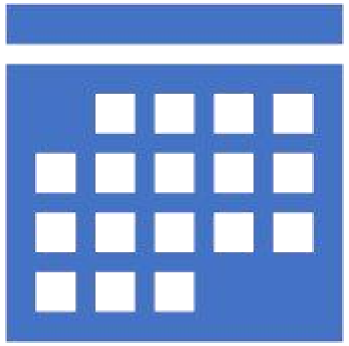


1-4周，6-9周每周二7-8节

南雍楼

每周二课上布置本周练习题，周六截止

课程形式



1-4周，6-9周每周二7-8节

南雍楼

由于中秋调休，最后一节课会在第九周进行。第八周、第九周课上进行pre和互评。提交通道于11.3关闭。

每周二课上布置本周练习题，周六截止

联系方式

jayceesjq@gmail.com

南雍楼

大作业

- 五人一组完成，结课提交源代码、实验报告和一份6分钟以内的介绍视频，主要解释实现的重要功能，并附上相关的代码片段，展示完整的编译-运行、试玩流程。

- 完成的题目选题(可作为参考):
  - 扫雷小游戏
  - 贪吃蛇小游戏
  - 2048
  - 自选

- 电脑端程序，使用C++完成，需要运用面向对象的编程方法

- 最后三节课会在课上播放大家的视频，每组附3分钟提问-答疑时间，并在教学平台上互相提交评分，作为大作业得分的20%

准备好了吗

准备好了吗

算法分类复习—动态规划

数字三角形问题

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

在上面的数字三角形中寻找一条从顶部到底边的路径，使得路径上所经过的数字之和最大。路径上的每一步都只能往左下或右下走。只需要求出这个最大和即可，不必给出具体路径。

三角形的行数大于1 小于等于 100 ，数字为 0 99

数字三角形问题

输入格式:

5　　//三角形行数
7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

要求输出最大和

递归解题思路

用二维数组存放数字三角形

数字三角形问题

用二维数组存放数字三角形

D(i, j)：第i行第j个数字，i, j从1开始算
MaxSum(i, j): 从D(i, j)到底边的各条路径中，最佳路径的数字和
问题：求MaxSum(1,1)

数字三角形问题

用二维数组存放数字三角形

D(i, j)：第i行第j个数字，i, j从1开始算
MaxSum(i, j): 从D(i, j)到底边的各条路径中，最佳路径的数字和
问题：求MaxSum(1,1)

典型的递归问题
D(i, j)出发，下一步只可以走D(i+1, j)或者D(i+1, j+1)。因此对于N行的三角形：

```
if (i == N)
    MaxSum(i, j) = D(i, j)
else
    MaxSum(i, j) = Max{ MaxSum(i+1, j), MaxSum(i+1, j+1) } + D(i, j)
```

数字三角形问题

递归方法深度遍历每条路径，存在大量重复计算，时间复杂度为$O(2^n)$。

改进：每次计算出一个MaxSum(i,j)都保存下来，时间复杂度是多少?

数字三角形问题

递归方法深度遍历每条路径，存在大量重复计算，时间复杂度为$O(2^n)$。

改进：每次计算出一个MaxSum(i,j)都保存下来，时间复杂度是多少?

存在TE

递归方法深度遍历每条路径，存在大量重复计算，时间复杂度为$O(2^n)$。

改进：每次计算出一个MaxSum(i,j)都保存下来，时间复杂度是多少？ $O(n^2)$

```
int maxsum[MAX][MAX];
int MaxSum(int i, int j){
    if (maxsum[i][j] != -1) return maxsum[i][j];
    if (i == n) maxsum[i][j] = D[i][j];
    else{
        int x = MaxSum(i+1, j);
        int y = MaxSum(i+1, j+1);
        maxsum[i][j] = max(x, y) + D[i][j];
    }
    return maxsum[i][j];
}
```

递归转换成递推

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

4	5	2	6	5

递归转换成递推

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7				
4	5	2	6	5

递归转换成递推

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7	12			
4	5	2	6	5

递归转换成递推

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7	12	10		
4	5	2	6	5

递归转换成递推

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7	12	10	10	
4	5	2	6	5

递归转换成递推

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

20				
7	12	10	10	
4	5	2	6	5

递归转换成递推

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

30				
23	21			
20	13	10		
7	12	10	10	
4	5	2	6	5

数字三角形问题

```
for (int i = n-1; i >= 1; i--)
    for (int j = 1; j <= i; j++)
        maxsum[i][j] = max( maxsum[i+1][j], maxsum[i+1][j+1]) + D[i][j];
```

数字三角形问题

```
for (int i = n-1; i >= 1; i--)
    for (int j = 1; j <= i; j++)
        maxsum[i][j] = max( maxsum[i+1][j], maxsum[i+1][j+1]) + D[i][j];
```

注意到每个数字(i,j)在计算完后，最后的调用在被正上方(i-1,j)对应的操作，因此从左往右处理时(i-1, j)的结果完全可以存储在(i, j)的位置

没有必要使用maxSum二维数组，只需要用一维数组存储一行

数字三角形问题

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7	5	2	6	5

数字三角形问题

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7	12	2	6	5

数字三角形问题

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7	12	10	6	5

数字三角形问题

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

7	12	10	10	5

数字三角形问题

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

20	12	10	10	5

数字三角形问题

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

20	13	10	10	5

空间优化方法

7
3 8
8 1 0
2 7 4 4
4 5 2 6 5

20	13	10	10	5

进一步考虑，可以直接舍弃maxsum数组，直接使用D的第n行代替maxsum。

节省空间，时间复杂度不变。

递推型动态规划：空间优化

~~for (int i = 1; i <= n; i++)~~

    ~~maxsum[n][i] = D[n][i];~~   maxsum = D[n];

for (int i = n-1; i >= 1; i--)

    for (int j = 1; j <= i; j++)

        maxsum[j] = max(maxsum[j], maxsum[j+1]) + D[i][j];

        ~~maxsum[i][j] = max(maxsum[i+1][j], maxsum[i+1][j+1]) + D[i][j];~~

动归解题

1. 将原问题分解为子问题

2. 定义状态

3. 确定边界值

4. 推导状态转移方程

动归解题

适合使用动态规划解决的问题的特点:

1. 问题具有最优子结构性质。如果问题的最优解所包含的子问题的解也是最优的，我们就称该问题具有最优子结构性质。

2. 无后效性。 当前的若干个状态值一旦确定，则此后过程的演变就只和这若干个状态的值有关，和之前是采取哪种手段或经过哪条路径演变到当前的这若干个状态，没有关系。

拦截导弹

某国为了防御敌国的导弹袭击，发展出一种导弹拦截系统。但是这种导弹拦截系统有一个缺陷：

虽然它的第一发炮弹能够到达任意的高度，但是以后每一发炮弹都不能高于前一发的高度。

某天，雷达捕捉到敌国的导弹来袭。由于该系统还在试用阶段，所以只有一套系统，因此有可能不能拦截所有的导弹。输入导弹依次飞来的高度（雷达给出的高度数据是不大于30000的正整数，导弹数不超过1000），计算这套系统最多能拦截多少导弹?

拦截导弹

输入数据
共两行。
第一行一个N,表示有n个导弹。
第二行，输入导弹依次飞来的高度。
输出
一个整数，表示最多能拦截的导弹数。

输入样例
8
389 207 155 300 299 170 158 65

输出样例
6

解题思路

1. 确定子问题

求序列的前n 个元素的最长下降(不上升)子序列的长度

解题思路

1. 确定子问题

求序列的前n个元素的最长下降子序列的长度

是子问题，但是没有无后效性

解题思路

1. 确定子问题

求序列的前n 个元素的最长下降子序列的长度

假设F(n) = x, 可能有多个序列满足 F(n) = x 。有的序列的最后一个元素比 a_{n+1} 大，则加上 a_{n+1} 就能形成更长的下降子序列；有的序列最后一个元素不比 a_{n+1} 大……如何达到状态 n，会影响之后的状态转移，因此不符合“无后效性”

解题思路

1. 确定子问题

求以a_k (k=1, 2, 3⋯N)为终点的最长下降子序列的长度

解题思路

1. 确定子问题

求以a_k (k=1, 2, 3…N)为终点的最长下降子序列的长度

满足无后效性，状态演变只取决于当前的状态a_k。

解题思路

2. 确定状态

子问题：求以a_k (k=1, 2, 3⋯N)为终点的最长下降子序列的长度

子问题只和数字的位置相关，因此序列中数字的位置k就是“状态”，状态k对应的值就是以a_k作为终点的最长下降子序列的长度。

状态一共有N个。

解题思路

3. 找出状态转移方程

maxlen(k)表示以ak作为终点的最长下降子序列的长度，那么：

初始状态：maxlen(1) = 1

转移方程：maxlen(k) = max{ maxlen(i): 1 <= i < k且ai >= ak, k≠1} + 1

如果找不到这样的i，则maxlen(k) = 1

maxlen(k)的值就是在ak左边，终点数值不小于ak，并且长度最大的那个下降子序列的长度+1。

最长下降序列

```
for ( int i = 2; i <= N; i++)
    for ( int j = 1; j < i; j++)
        if (a[i] <= a[j])
            maxlen[i] = max( maxlen[i], maxlen[j] + 1);
```

最长下降子序列：解法二

```
for ( int i = 2; i <= N; i++)
    for ( int j = 1; j < i; j++)
        if (a[i] <= a[j])
            maxlen[i] = max( maxlen[i], maxlen[j] + 1);
```

```
for ( int i = 1; i <= N; i++)
    for (int j = i + 1; j <= N; j++)
        if (a[j] <= a[i])
            maxlen[j] = max( maxlen[j], maxlen[i] + 1);
```

计算字符串距离

对于两个不同的字符串，我们有一套操作方法来把他们变得相同，具体方法为：
修改一个字符（如把“a”替换为“b”）
删除一个字符（如把“traveling”变为“travelng”）

比如对于“abcdefg”和“abcdef”两个字符串来说，我们认为可以通过增加/减少一个“g”的方式来达到目的。无论增加还是减少“g”，我们都仅仅需要一次操作。我们把这个操作所需要的次数定义为两个字符串的距离。
给定任意两个字符串，写出一个算法来计算出他们的距离。

计算字符串距离

输入
第一行有一个整数n。表示测试数据的组数，
接下来共n行，每行两个字符串，用空格隔开。表示要计算距离的两个字符串
字符串长度不超过1000。
输出
针对每一组测试数据输出一个整数，值为两个字符串的距离。

样例输入	样例输出
3	
abcdefg abcdef	1
ab ab	0
mnklj jlknm	4

解题思路

假设数组distance[i][j]代表字符串s1[1:i]与s2[1:j]的距离。distance[i][0] = i, distance[0][j] = j(删除i或j次)。存在几种可能:

解题思路

假设数组distance[i][j]代表字符串s1[1:i]与s2[1:j]的距离。distance[i][0] = i, distance[0][j] = j(删除i或j次)。存在几种可能：

1. s1[i] = s2[j]，distance[i][j] = distance[i-1][j-1]

解题思路

假设数组distance[i][j]代表字符串s1[1:i]与s2[1:j]的距离。distance[i][0] = i, distance[0][j] = j(删除i或j次)。存在几种可能:

1. s1[i] = s2[j]，distance[i][j] = distance[i-1][j-1]

2. 删除一个字符，distance[i][j] = min(distance[i-1][j], distance[i][j-1]) + 1

解题思路

假设数组distance[i][j]代表字符串s1[1:i]与s2[1:j]的距离。distance[i][0] = i, distance[0][j] = j(删除i或j次)。存在几种可能:

1. s1[i] = s2[j]，distance[i][j] = distance[i-1][j-1]

2. 删除一个字符，distance[i][j] = min(distance[i-1][j], distance[i][j-1]) + 1

3. 修改一个字符，使得s1[i] = s2[j]，distance[i][j] = distance[i-1][j-1] + 1

解题思路

假设数组distance[i][j]代表字符串s1[1:i]与s2[1:j]的距离。distance[i][0] = i, distance[0][j] = j(删除i或j次)。存在几种可能：

1. s1[i] = s2[j]，distance[i][j] = distance[i-1][j-1]

2. 删除一个字符，distance[i][j] = min(distance[i-1][j], distance[i][j-1]) + 1

3. 修改一个字符，使得s1[i] = s2[j]，distance[i][j] = distance[i-1][j-1] + 1

2和3在程序上需要合并，即distance[i][j] = min (min(distance[i-1][j], distance[i][j-1]), distance[i-1][j-1]) + 1

最佳加法表达式

有一个由1..9 组成的数字串，问如果将 m 个加号插入到这个数字串中 在各种可能形成的表达式中，值最小的那个表达式的值是多少。例如，在1234中摆放1个加号，最好的摆法就是12+34,和为36。

输入
有不超过15组数据，每组数据两行。
第一行是整数m，表示有m个加号要放(0<=m<=50)
第二行是若干个数字。数字总数n不超过50,且 m <= n-1

输出
对每组数据，输出最小加法表达式的值

解题思路

假定数字串长度是n，添完加号后，表达式的最后一个加号添加在第 i 个数字后面，那么整个表达式的最小值，就等于在前 i 个数字中插入 m-1个加号所能形成的最小值，加上第 i+1 到第 n个数字所组成的数的值（i 从 1 开始算）。

解题思路

假定数字串长度是n，添完加号后，表达式的最后一个加号添加在第 i 个数字后面，那么整个表达式的最小值，就等于在前 i 个数字中插入 m-1个加号所能形成的最小值，加上第 i + 1 到第 n个数字所组成的数的值（i 从 1 开始算）。

设V(m,n) 表示在 n 个数字中插入 m 个加号所能形成的表达式最小值，那么

$$V(m,n) = \begin{cases} n\text{个数字构成的整数}, & m = 0 \\ \infty, & n < m + 1 \\ \mathrm{Min}\{V(m-1,i) + \mathrm{Num}(i+1,n)\}\ (i = m, \ldots, n-1), & \text{else} \end{cases}$$

Num(i,j)表示从第 i 个数字到第 j 个数字所组成的数。数字编号从 1 开始算。此操作复杂度是 O(j-i+1)，可以预处理后存起来。

大盗阿福

阿福是一名经验丰富的大盗。趁着月黑风高，阿福打算今晚洗劫一条街上的店铺。

这条街上一共有 N 家店铺，每家店中都有一些现金。阿福事先调查得知，只有当他同时洗劫了两家相邻的店铺时，街上的报警系统才会启动，然后警察就会蜂拥而至。

作为一向谨慎作案的大盗，阿福不愿意冒着被警察追捕的风险行窃。他想知道，在不惊动警察的情况下，他今晚最多可以得到多少现金?

大盗阿福

输入

输入的第一行是一个整数 T (T <= 50) ，表示一共有 T 组数据。
接下来的每组数据，第一行是一个整数 N (1 <= N <= 100, 000) ，表示一共有 N 家店铺。第二行是 N 个被空格分开的正整数，表示每一家店铺中的现金数量。每家店铺中的现金数量均不超过 1000 。

输出

对于每组数据，输出一行。该行包含一个整数，表示阿福在不惊动警察的情况下可以得到的现金数量。

大盗阿福

样例输入
2
3
1 8 2
4
10 7 6 14

样例输出
8
24

解题思路

本题是典型的动态规划问题，沿街的房屋依次决定每一家是偷还是不偷。假设val[i]代表从第一家到第i家可以获得的最大金额，那么：

解题思路

本题是典型的动态规划问题，沿街的房屋依次决定每一家是偷还是不偷。假设val[i]代表从第一家到第i家可以获得的最大金额，那么：

1. 不偷第i家，则val[i] = val[i-1]

解题思路

本题是典型的动态规划问题，沿街的房屋依次决定每一家是偷还是不偷。假设val[i]代表从第一家到第i家可以获得的最大金额，那么:

1. 不偷第i家，则val[i] = val[i-1]

2. 偷第i家，则val[i] = val[i-2] + cash[i]

解题思路

本题是典型的动态规划问题，沿街的房屋依次决定每一家是偷还是不偷。假设val[i]代表从第一家到第i家可以获得的最大金额，那么:

1. 不偷第i家，则val[i] = val[i-1]

2. 偷第i家，则val[i] = val[i-2] + cash[i]

本问题中，由于i只和i-1, i-2状态有关，甚至可以省略val数组，用两个变量指代val[i-1]和val[i-2]，并将val[i]更新在val[i-1]上:

```
res = max( previous + cash[i], curr);
previous = curr;
curr = res;
```

孙悟空的背包

孙悟空找东海老龙王讨武器去了。老龙王拿出了 N 件武器，其中第 i 件武器的重量是 Wi，孙悟空对它的喜好值是 Vi。

孙悟空有一个虎皮背包，最多能承受的重量上限是 M。他想知道可以装上哪些武器，使得在总重量不超过 M 的前提下，总喜好值最大。每种物品只有一件，可以选择放或者不放。

孙悟空的背包

输入
第一行包含两个整数 N 和 M (1<=N<=1000, 1<=M<=10000) 分别代表武器数量和背包重量上限。

接下来 N 行每行包含两个整数 Wi 和 Vi (1<=Wi<=2000, 1<=Vi<=1000) 分别代表一件武器的重量和喜好值。

输出
一个整数，表示最大总喜好值。

孙悟空的背包

用F[i][j]表示取前i种物品，使它们总体积不超过j的最优取法取得的喜好值总和。要求F[N][M]。

边界：

```
if(w[1] <= j)
    F[1][j] = v[1];
else
    F[1][j] = 0;
```

孙悟空的背包

用F[i][j]表示取前i种物品，使它们总体积不超过j的最优取法取得的喜好值总和。要求F[N][M]。

边界：

```
if(w[1] <= j)
    F[1][j] = v[1];
else
    F[1][j] = 0;
```

递推： F[i][j] = max(F[i-1][j], F[i-1][j-w[i]] + v[i])，取或者不取第i种物品

孙悟空的背包

$F[i][j] = \max(F[i-1][j], F[i-1][j-w[i]] + v[i])$， $1<=N<=1000, 1<=M<=10000$

本题如用记忆型递归，需要一个很大的二维数组，可能会超内存。注意到这个二维数组的下一行的值，只用到了上一行的正上方及左边的值，因此可用滚动数组的思想，只要一行即可。

孙悟空的背包

$F[i][j] = \max(F[i-1][j], F[i-1][j-w[i]] + v[i])$，1<=N<=1000, 1<=M<=10000

本题如用记忆型递归，需要一个很大的二维数组，可能会超内存。注意到这个二维数组的下一行的值，只用到了上一行的正上方及左边的值，因此可用滚动数组的思想，只要一行即可。

如果孙悟空还想知道达到总喜好值最大的最小重量是多少，应该如何修改？

输出

一行包含两个整数，分别表示最大总喜好值和达到这个值的最小重量。

滑雪

小明每周四都和朋友一起去滑雪，用滑雪成绩决定谁请客吃当晚的肯德基。为了获得速度，滑的区域必须向下倾斜。但是由于滑雪场设备老旧，每次小明滑到坡底，都不得不再次走上坡或者等待升降机来接人。小明想知道在一个区域中最长的滑坡。区域由一个二维数组给出。数组的每个数字代表点的高度。下面是一个例子：

1 2 3 4 5
16 17 18 19 6
15 24 25 20 7
14 23 22 21 8
13 12 11 10 9

一个人可以从某个点滑向上下左右相邻四个点之一(当且仅当高度减小)。在上面的例子中，一条可滑行的滑坡为24-17-16-1。当然25-24-23-...-3-2-1更长。

滑雪

输入：

输入的第一行表示区域的行数R和列数C(1 <= R,C <= 100)。下面是R行，每行有C个整数，代表高度h，0<=h<=10000。

输出：

输出最长区域的长度。

样例输入：

5 5

1 2 3 4 5

16 17 18 19 6

15 24 25 20 7

14 23 22 21 8

13 12 11 10 9

样例输出：

25

解题思路

$L(i, j)$ 表示从点 i,j 出发的最长滑行长度。一个点(i, j), 如果周围没有比它低的点，则$L(i, j) = 1$

解题思路

L(i, j) 表示从点 i,j 出发的最长滑行长度。一个点(i,j), 如果周围没有比它低的点，则L(i, j) = 1

否则：

L(i, j) = (i, j)周围四个点中 比 i,j 低，且L值最大的那个点的L值，再加 1

解题思路

L(i, j) 表示从点 i,j 出发的最长滑行长度。一个点(i, j), 如果周围没有比它低的点，则L(i, j) = 1

否则：

L(i, j) = (i, j)周围四个点中 比 i,j 低，且L值最大的那个点的L值，再加 1

解法1.

将所有点按高度从小到大排序，每个点的 L 值都初始化为 1。

从小到大遍历所有的点。经过一个点(i, j)时，用递推公式求 L(i, j)

解题思路

L(i, j) 表示从点 i,j 出发的最长滑行长度。一个点(i, j), 如果周围没有比它低的点，则 L(i, j) = 1

否则：

L(i,j) = (i,j)周围四个点中 比 i,j 低，且L值最大的那个点的L值，再加 1

解法2.

将所有点按高度从小到大排序，每个点的 L 值都初始化为 1。

从小到大遍历所有的点。经过一个点(i, j)时，要更新他周围的，比它高的点的 L 值。例如：

if H(i+1, j) > H(i, j)

L(i+1, j) = max(L(i+1, j), L(i, j)+1)

状态压缩动态规划

海贼王之伟大航路

“我是要成为海贼王的男人！”，路飞一边喊着这样的口号，一边和他的伙伴们一起踏上了伟大航路的艰险历程。

路飞他们伟大航路行程的起点是罗格镇，终点是拉夫德鲁（那里藏匿着“唯一的大秘宝”——ONE PIECE）。而航程中间，则是各式各样的岛屿。

因为伟大航路上的气候十分异常，所以来往任意两个岛屿之间的时间差别很大，从A岛到B岛可能需要1天，而从B岛到A岛则可能需要1年。当然，任意两个岛之间的航行时间虽然差别很大，但都是已知的。

现在假设路飞一行从罗格镇（起点）出发，遍历伟大航路中间所有的岛屿（但是已经经过的岛屿不能再次经过），最后到达拉夫德鲁（终点）。假设他们在岛上不作任何的停留，请问，他们最少需要花费多少时间才能到达终点?

海贼王之伟大航路

海贼王之伟大航路

输入

输入数据包含多行。

第一行包含一个整数$N(2 < N \le 16)$，代表伟大航路上一共有N个岛屿（包含起点的罗格镇和终点的拉夫德鲁）。其中，起点的编号为1，终点的编号为N。

之后的N行每一行包含N个整数，其中，第$i(1 \le i \le N)$行的第$j(1 \le j \le N)$个整数代表从第i个岛屿出发到第j个岛屿需要的时间$t(0 < t < 10000)$。第i行第i个整数为0。

输出

输出为一个整数，代表路飞一行从起点遍历所有中间岛屿（不重复）之后到达终点所需要的最少的时间。

海贼王之伟大航路

样例输入1：
4
0 10 20 999
5 0 90 30
99 50 0 10
999 1 2 0

样例输出1：
100

样例输入2：
5
0 18 13 98 8
89 0 45 78 43
22 38 0 96 12
68 19 29 0 52
95 83 21 24 0

样例输出2：
137

海贼王之伟大航路

对于样例输入1：路飞选择从起点岛屿1出发，依次经过岛屿3，岛屿2，最后到达终点岛屿4。花费时间为20+50+30=100。

对于样例输入2：可能的路径及总时间为：
1,2,3,4,5: 18+45+96+52=211
1,2,4,3,5: 18+78+29+12=137
1,3,2,4,5: 13+38+78+52=181
1,3,4,2,5: 13+96+19+43=171
1,4,2,3,5: 98+19+45+12=174
1,4,3,2,5: 98+29+38+43=208
所以最短的时间花费为137
单纯的枚举在N=16时需要14!次运算，一定会超时。

问题分析

题意：典型的Tsp问题，从1开始跑完1~n的所有岛屿，最终走到n，不能重复走，求最少时间。

定义当前在岛屿l，并且已经走过了一个岛屿集合s，那么需要求的dp[s][l]相当于：

s中任意一个岛屿i，求dp[s-i][i] + cost[i][l]的最小值

问题分析

题意：典型的Tsp问题，从1开始跑完1~n的所有岛屿，最终走到n，不能重复走，求最少时间。

定义当前在岛屿l，并且已经走过了一个岛屿集合s，那么需要求的dp[s][l]相当于：

s中任意一个岛屿i，求dp[s-i][i] + cost[i][l]的最小值

如何表示状态集合s?

问题分析

有时，状态相当复杂，看上去需要很多空间，比如一个数组才能表示一个状态，那么就需要对状态进行某种编码，进行压缩表示。

本问题中，可以将岛屿表示为二进制数字，共16位，每一位代表某个岛屿是否去过。因此，状态转移方程可以写成：

问题分析

有时，状态相当复杂，看上去需要很多空间，比如一个数组才能表示一个状态，那么就需要对状态进行某种编码，进行压缩表示。

本问题中，可以将岛屿表示为二进制数字，共16位，每一位代表某个岛屿是否去过。因此，状态转移方程可以写成：

dp[s][l] = min(dp[s & ~(1 << i)][i] + dis[i][l], dp[s][l])

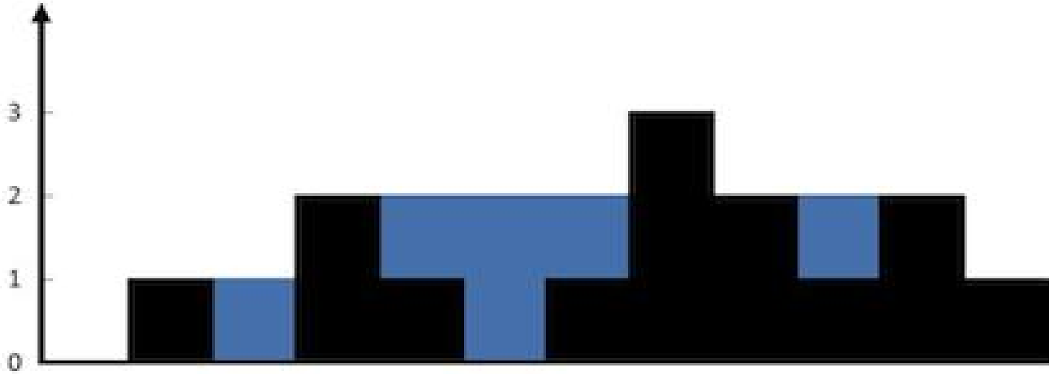
附加练习题

附加练习题

接雨水

给定 n 个非负整数表示每个宽度为 1 的柱子的高度图，计算按此排列的柱子，下雨之后能接多少雨水。

示例:



height = [0, 1, 0, 2, 1, 0, 1, 3, 2, 1, 2, 1]

由数组表示的高度图，在这种情况下，可以接 6 个单位的雨水（蓝色部分表示雨水）。

接雨水

输入

第一行包含一个整数n。1 <= n <= 2 * 10^4

第二行包含n个整数，相邻整数间以空格隔开。0 <= ratings[i] <= 2 * 10^5

输出

一个整数

样例输入

sample1 input:

12

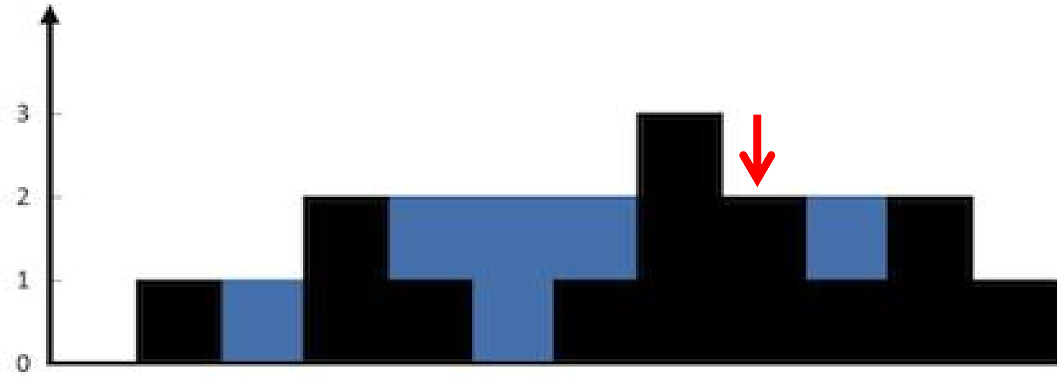
0 1 0 2 1 0 1 3 2 1 2 1

样例输出

6

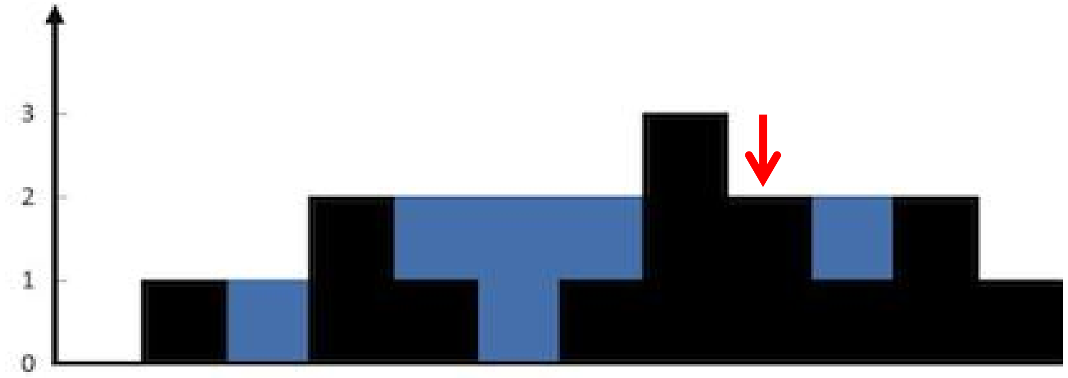
问题分析

本题的难点在于判断格子是否可以接住雨水，以及如何对雨水量进行更新。
一个直观的想法是对于所有柱子，都向左、向右寻找最高的柱子。考虑红色箭头所指位置，向左存在比它更高的柱子，可以接住雨水；但向右找不到更高的柱子，接不住雨水。



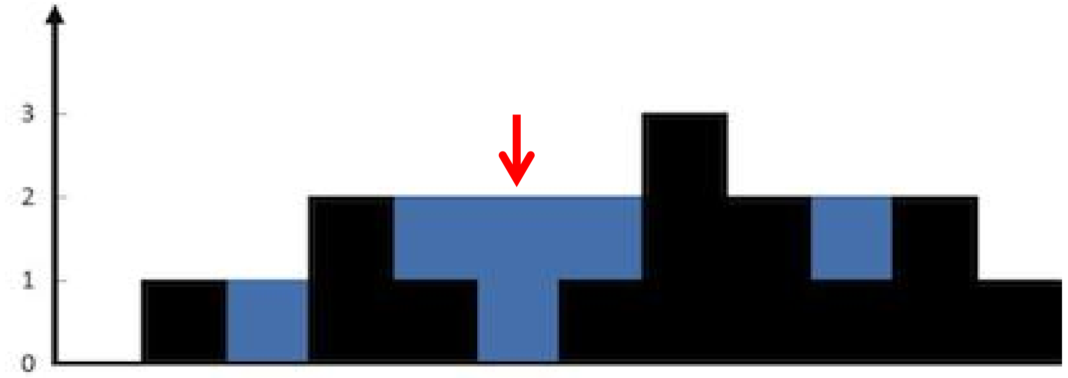
问题分析

本题的难点在于判断格子是否可以接住雨水，以及如何对雨水量进行更新。
一个直观的想法是对于所有柱子，都向左、向右寻找~~最高~~的柱子。考虑红色箭头所指位置，向左存在比它更高的柱子，可以接住雨水；但向右找不到更高的柱子，接不住雨水。



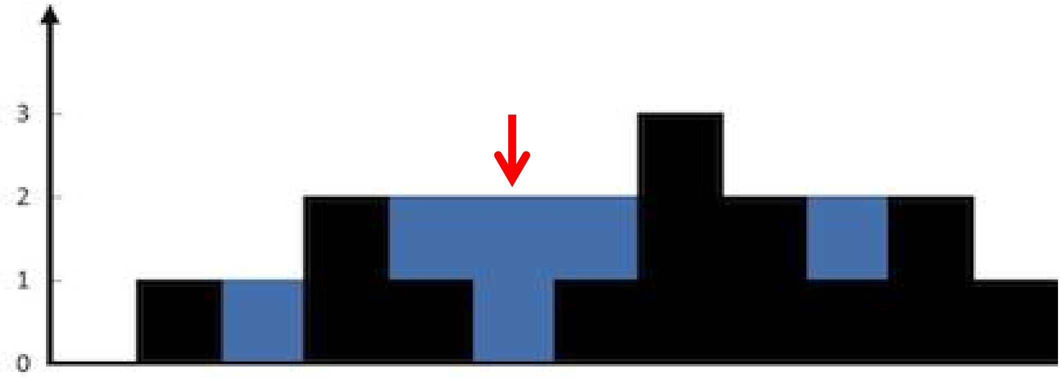
问题分析

最高不行。考虑红色箭头所指的位置，寻找最高柱子并不能用于计算雨水量，只能保证可以接住雨水。事实上，决定这个位置雨水量的是左右最高柱子的较低一个(木桶原理)。



问题分析

最高不行。考虑红色箭头所指的位置，寻找最高柱子并不能用于计算雨水量，只能保证可以接住雨水。事实上，决定这个位置雨水量的是左右最高柱子的较低一个(木桶原理)。

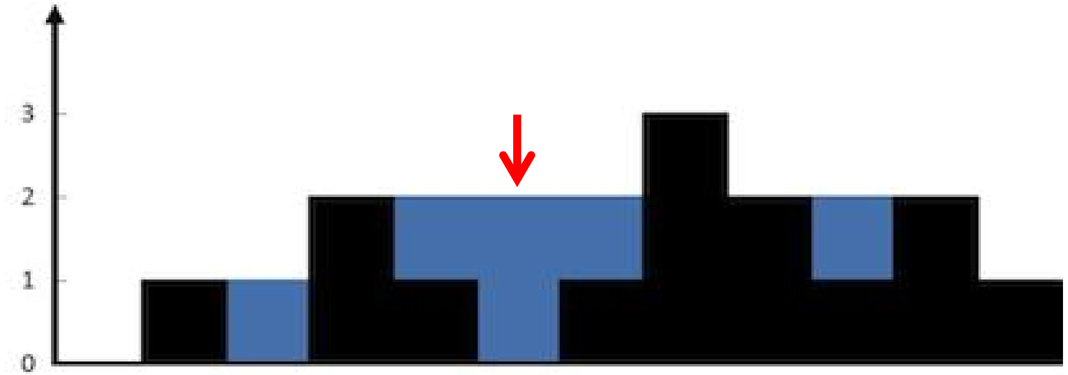


暴力枚举：对于每一个位置都向左、向右求最高的柱子中最低的一个，作为这一列的雨水量。可能TE。

问题分析

暴力枚举：对于每一个位置都向左、向右求最高的柱子中最低的一个，作为这一列的雨水量。可能TE。尝试优化暴力向左、向右求最高柱子的步骤。

子问题：对于每个位置i，左边的最高柱子和右边的最高柱子分别是什么?



问题分析

在第i个位置，记i左边的最高柱子为leftMax[i]，右边的最高柱子为rightMax[i]，那么从左、右两边分别更新:

leftMax[i] = max(leftMax[i-1], height[i-1]),
rightMax[i] = max(rightMax[i+1], height[i+1])

问题分析

在第i个位置，记i左边的最高柱子为leftMax[i]，右边的最高柱子为rightMax[i]，那么从左、右两边分别更新：

leftMax[i] = max(leftMax[i-1], height[i-1]),
rightMax[i] = max(rightMax[i+1], height[i+1])

注意rightMax需要从右往左更新，因为rightMax[i-1]可能对应i位置本身，不能直接用于更新。

问题分析

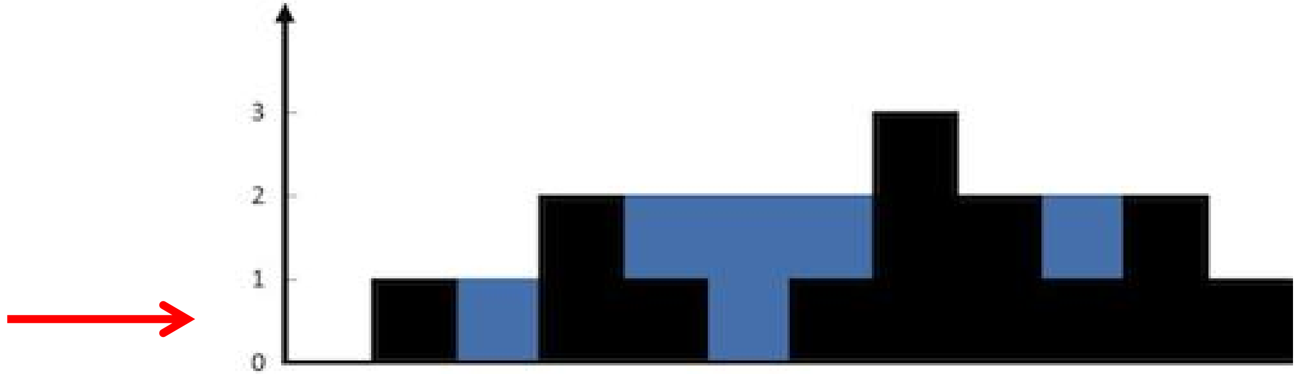
```
int leftMax[MAXN];
int rightMax[MAXN];
for (int i = 1; i < len - 1; i++)
    leftMax[i] = max(leftMax[i - 1], height[i - 1]);

for (int i = len - 2; i >= 0; i--)
    rightMax[i] = Math.max(rightMax[i + 1], height[i + 1]);

for (int i = 1; i < len - 1; i++) {
    int minHeight = min(leftMax[i], rightMax[i]);
    result += minHeight > height[i] ? minHeight - height[i] : 0;
}
```

接雨水:另一个解法

是否可以从行的角度解决接雨水的问题?



关注最后一行，在第三格碰到一个凹槽，雨水+1；第6格也碰到一个凹槽，雨水+1。

求第i层的水，遍历每个位置，如果当前的高度小于i，并且两边有高度大于等于i的，说明这个地方一定有水，水就可以增加 。

接雨水:另一个解法

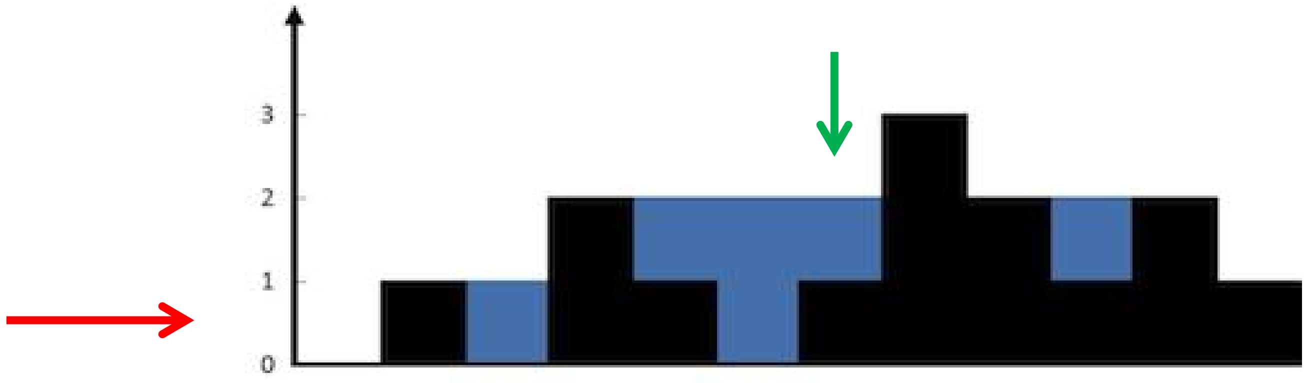
暴力解法:

如果求高度为i 的水，首先用临时一个变量water保存当前累积的水，初始化为0。从左到右遍历墙的高度，遇到h >= i的时候，开始更新。

已经开始更新后，第一次遇到h < i，就把water + 1，再次遇到h >= i时，说明这个地方一定有水，就把water加到最终的答案里，并把water重新设置为0，停止更新，继续循环。

接雨水:另一个解法

进一步分析，当位置i出现一个高柱子后，i左边的矮柱子就不会影响到[i, i+k]区域的雨水量，因此可以直接被忽略。但是i无法遮挡i左边的高柱子。



如图，当第七列(绿箭头)出现一个比较高的柱子，第六列平地被遮挡了，无法遮挡第四列更高的柱子。但是[5,7]之间的雨水已经固定，后续不变。

当位置i出现一个矮柱子，i左边的高柱子都可能与后续某个[i, i+k]形成接雨水。

接雨水:另一个解法

考虑:

1. 每次出现高柱子i，找到左边与它形成凹陷的两根柱子j < k，只计算(i - j) * (height[i] - height[k])的雨水。

2.找一个数据结构，只保存递减(不增)的柱子。

接雨水:另一个解法

单调栈/单调队列：栈内元素按照某个策略排序

单调栈存储下标，满足从栈底到栈顶的下标对应的柱子高度递减，即栈顶是当前位置左边最矮的柱子。

从左到右遍历数组，遍历到下标i时，如果栈内至少有两个元素，记栈顶元素为top，栈顶第二个元素为next，则一定有height[next] > height[top]。

如果height[i] > height[top]，则得到一个可以接雨水的区域，范围在柱子next与柱子i之间。该区域的宽度是i - next - 1，高度是min(height[i], height[next]) - height[top]，根据宽度和高度即可计算得到该区域能接的雨水量。

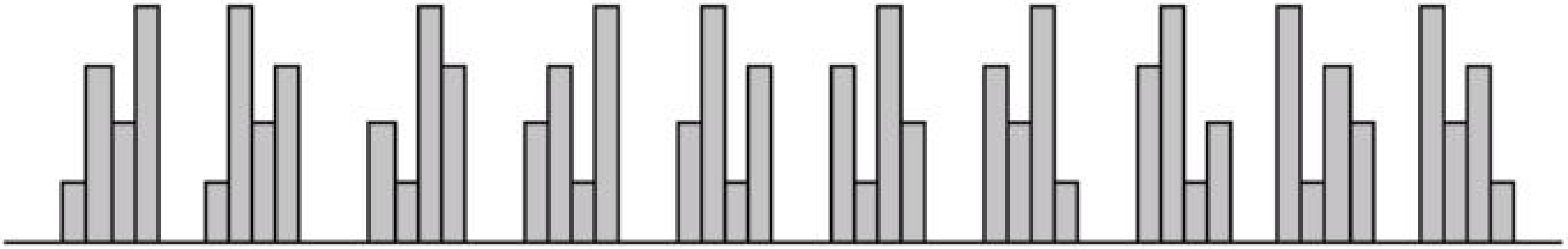
接雨水:另一个解法

```
Deque<Water> stack = new LinkedList<Water>();
  for (int i = 0; i < len; i++) {
    while (!stack.isEmpty() && height[i] > height[stack.peek()]) {
      int top = stack.pop();
      if (stack.isEmpty()) break;
      int next = stack.peek();
      int currWidth = i - next - 1;
      int currHeight = min(height[left], height[i]) - height[top];
      ans += currWidth * currHeight;
    }
    stack.push(i);
  }
```

美妙的栅栏

N 个木棒 长度分别为 1, 2, N. 构成美妙的栅栏。除了两端的木棒外，每一跟木棒，要么比它左右的两根都长，要么比它左右的两根都短，即木棒呈现波浪状分布。符合上述条件的栅栏建法有很多种，对于满足条件的所有栅栏,按照字典序 从左到右 ,从低到高 排序。

给定一个 栅栏的排序号，请输出该栅栏，即每一个木棒的长度。



All cute fences made of $N = 4$ planks, ordered by their catalogue numbers.

美妙的栅栏

输入数据

第一行是测试数据的组数 K (1 <= K <= 100 ）。 接下来 的 K行每一行描述一组输入数据。
每一组输入数据包括两个整数N和C. (1 <= N <= 20) 表示栅栏的木棒数，C表示要找的栅栏的排列号

输出数据

输出第C个栅栏每一个木棒的长度

美妙的栅栏

输入样例
2
2 1
3 3

输出样例
1 2
2 3 1

解题思路

首先对问题进行抽象：

给定1到N这N个数字，将这些数字高低交替进行排列，把所有符合情况的进行一个字典序排列。问第C个排列是一个怎样的排列？首先理解排列计数问题

解题思路

首先对问题进行抽象：

给定1到N这N个数字，将这些数字高低交替进行排列，把所有符合情况的进行一个字典序排列。问第C个排列是一个怎样的排列？

如1，2，3，4的全排列，一共4！种，求第10个排列。

解题思路

首先对问题进行抽象:

给定1到N这N个数字，将这些数字高低交替进行排列，把所有符合情况的进行一个字典序排列。问第C个排列是一个怎样的排列?

如1，2，3，4的全排列，一共4！种，求第10个排列。

首先尝试首位1，234有3!=6 < 10种，说明1 偏小，转换成2开头的第(10-6=4)个排列。

解题思路

首先对问题进行抽象:

给定1到N这N个数字，将这些数字高低交替进行排列，把所有符合情况的进行一个字典序排列。问第C个排列是一个怎样的排列?

如1， 2， 3， 4的全排列，一共4！ 种，求第10个排列。

首先尝试首位1，234有3!=6 < 10种，说明1 偏小，转换成2开头的第(10-6=4)个排列。

3!=6>4，首位确实是2.第二位依次尝试1,3，确定第二位是3，...最后得出第10个排列是2-3-4-1。

解题思路

回到本题：

对于每一组栅栏，首先全局对比第一根的长度，作为第一层排序；

解题思路

回到本题:

对于每一组栅栏，首先全局对比第一根的长度，作为第一层排序;

在确定第一根长度的大类之后，在大类内再使用第二根进行排序;

......

解题思路

回到本题:

对于每一组栅栏，首先全局对比第一根的长度，作为第一层排序;

在确定第一根长度的大类之后，在大类内再使用第二根进行排序;

使用M(i,j)表示：有i跟木棒，以其中第j短的作为第一根，能够构成的美妙栅栏的树木

假设第1短 的木棒作为第一根，看此时方案数量M(N,1)是否不小于C；如果否，则使用第2短的作为第一根，C - M(N,1)作为最终目的， ...

解题思路

使用M(i,j)表示：有i跟木棒，以其中第j短的作为第一根，能够构成的美妙栅栏的数目

假设第1短 的木棒作为第一根，看此时方案数量M(N,1)是否不小于C；如果否，则使用第2短的作为第一根，C - M(N,1)作为最终目的， ...

如果发现以第i短作为第一根，方案数量已经不少于C，则确定i为第一根，开始计算M(N-1, 1)， M(N-1, 2), ...

关键是需要求出所有M(i, j)

解题思路

令S(i)表示由i根木棒构成的合法方案集合。M[i][k]表示S(i)中以第k短的木棒开头的方案数量。

在选定了某根木棒x作为i根木棒中的第一根木棒的情况下，假定剩下i-1根木棒的合法方案数是 A[i-1] ，这 A[i-1] 种方案并不是每种都能和 x 形成新的合法方案。将第一根比第二根长的方案称为 DOWN 方案，第一根比第二根短的称为 UP 方案，则，S(i-1) 中，第一根木棒比 x 长的 DOWN 方案，以及第一根木棒比 x 短的 UP 方案，才能和 x 构成 S(i) 中的方案。

解题思路

令S(i)表示由i根木棒构成的合法方案集合。M[i][k]表示S(i)中以第k短的木棒开头的方案数量。

在选定了某根木棒k作为i根木棒中的第一根木棒的情况下，假定剩下i-1根木棒的合法方案数是 A[i-1] ，这 A[i-1] 种方案并不是每种都能和k形成新的合法方案。将第一根比第二根长的方案称为 DOWN 方案，第一根比第二根短的称为 UP 方案，则，S(i-1) 中，第一根木棒比 x 长的 DOWN 方案，以及第一根木棒比 x 短的 UP 方案，才能和 x 构成 S(i) 中的方案。

无法直接细化

解题思路

给M[i][k]加一维度M[i][k][flag]，flag分为up和down，分别代表第一根比第二根短和长的方案。

那么B[i][k] = B[i-1][k, ..., i-1][down]　//第k到i-1根木棒长度大于等于第k根
　　　　　+ B[i-1][1, ..., k-1][up]　//第1到k-1根木棒长度小于第k根

解题思路

给M[i][k]加一维度M[i][k][flag]，flag分为up和down，分别代表第一根比第二根短和长的方案。

那么M[i][k] = M[i-1][k, ..., i-1][down] //第k到i-1根木棒长度大于等于第k根
+ M[i-1][1, ..., k-1][up] //第1到k-1根木棒长度小于第k根

进一步细化，M[i][k][up] = M[i-1][k, ..., i-1][down]
M[i][k][down] = M[i-1][1, ..., k-1][up]

M[1][1][up] = M[1][1][down] = 1，对M进行动归

解题思路

```
void init(int n){
    for ( int i = 2; i <= n; i++)
        for (int k = 1; k <= i; k++){
            for ( int j = k; j < i; j++)
                M[i][k][UP] += M[i-1][j][DOWN];
            for ( int j = 1; j <= k-1; j++)
                M[i][k][DOWN] += M[i-1][j][UP];

}
```

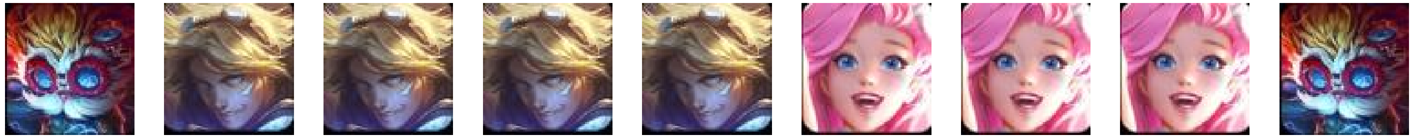
M[i][k][up] = M[i-1][k, ..., i-1][down]

M[i][k][down] = M[i-1][1, ..., k-1][up]

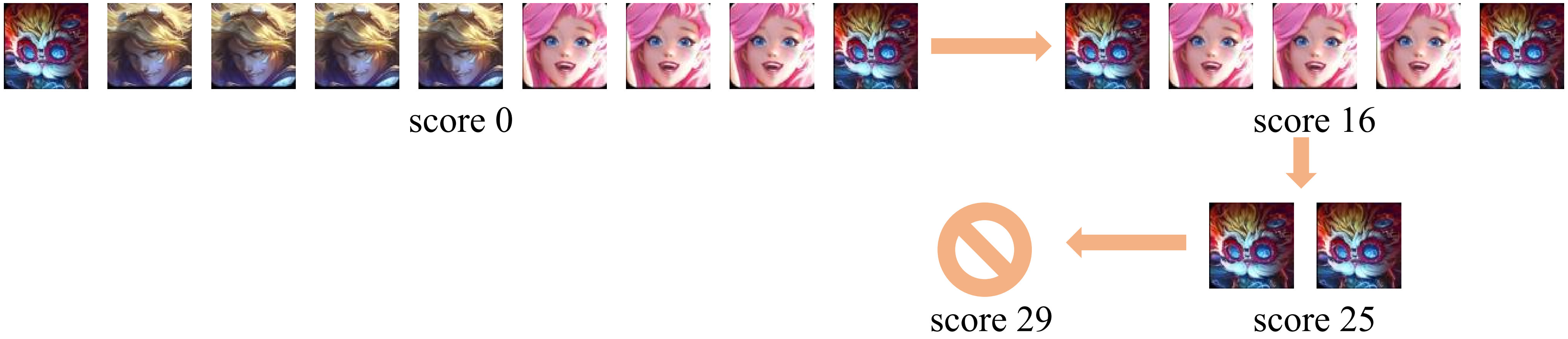
总方案是Sum(M[n][k][down] + M[n][k][UP]}, k = 1, ..., n，寻找第k个方案时也需要记得up和down相加

棋子消除(POJ1390方盒游戏)

N个棋子摆成一排，每个棋子都是不同的角色。连续摆放的同角色棋子构成一个棋子片段。下图中共有四个棋子片段，每个片段分别有1, 4, 3, 1个棋子。



玩家每次点击一个棋子，则该棋子所在片段就会消失。若消失的棋子片段中共有 k 个棋子，则玩家获得 k*k 个积分。请问：给定游戏开始时的状态，玩家可获得的最高积分是多少?



棋子消除

输入：
第一行是一个整数t(1<=t<=15)，表示共有多少组测试数据。每组测试数据包括两行。第一行是一个整数 n(1<=n<=200), 200),，表示共有多少个方盒；第二行包括 n 个整数，表示每个方盒的颜色。这些整数的取值范围是[1, n]。

输出：
对每组测试数据，分别输出该组测试数据的序号、以及玩家可以获得的最高积分

样例输入：

2

9

1 2 2 2 2 3 3 3 1

1

1

样例输出：

Case1: 29

Case2: 1

问题分析

当同角色的棋子摆放在不连续的位置时，棋子的点击顺序影响玩家获得的积分

问题分析

当同角色的棋子摆放在不连续的位置时，棋子的点击顺序影响玩家获得的积分

同种颜色的棋子被点击的次数越少，玩家获得的积分越高

明显的递归问题：每次点击之后，剩下的棋子构成一个新的队列，新队列中棋子的数量减少了。然后计算玩家从新队列中可获得的最高积分。

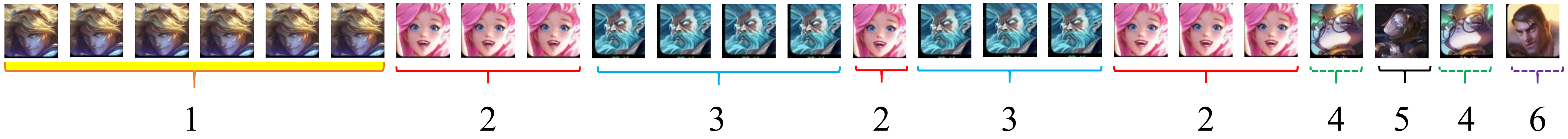
问题分析

点击下图中4号角色之前，先点击5号角色可提高玩家的积分：同角色的两个4号角色被5号角色隔开时，先点击5号角色，使得 两个4号角色消失前能够在同一个片段中

点击下图中2号和3号角色共可获得的积分：

所有3号角色合并到同一个片段： 49+1+36=86

所有2号角色合并到同一个片段： 49+16+9=74



问题分析

将连续的若干个棋子作为一个“大块”考虑，假设开始一共有n 个“大块”，编号 0 到 n-1，第i 个大块的角色是 name[i]，包含的棋子数目，即长度是len[i]

用click[i, j]表示从大块 i 到大块 j 这一段消除后所能得到的最高分，则最终需要求解的问题为click(0, n-1)

问题分析

将连续的若干个棋子作为一个“大块”考虑，假设开始一共有n 个“大块”，编号 0 到 n-1，第i 个大块的角色是 name[i]，包含的棋子数目，即长度是len[i]

用click[i, j]表示从大块 i 到大块 j 这一段消除后所能得到的最高分，则最终需要求解的问题为click(0, n-1)

考虑(i, j)最右边的大块j， 对它有两种处理方式：

1. 直接消除，此时的最高分是click(i, j-1) + len[j] * len[j]
2. 未来可期，看看能不能和左边的某个同色大块合并

问题分析

考虑和左边的某个同色大块合并：

左边的同色大块可能有很多个，只能枚举计算和哪个合并最好。假设大块 j 和左边的大块k(i <=k<j-1)合并，此时能够得到的最高分为：

问题分析

考虑和左边的某个同色大块合并:

左边的同色大块可能有很多个，只能枚举计算和哪个合并最好。假设大块 j 和左边的大块k(i <=k<j-1)合并，此时能够得到的最高分为:

$$click(i, k-1) + click(k+1, j-1) + (len[k] + len[j])^2$$

问题分析

考虑和左边的某个同色大块合并：

　　左边的同色大块可能有很多个，只能枚举计算和哪个合并最好。假设大块 j 和左边的大块k(i <=k<j-1)合并，此时能够得到的最高分为：

$$\text{click}(i, k-1) + \text{click}(k+1, j-1) + (\text{len}[k] + \text{len}[j])^2$$

因为将大块k 和大块 j 合并后，形成的新大块会在最右边。将该新大块直接将其消去的做法符合上述式子，但直接将其消去未必是最好的，也许它还应该继续和左边的同色大块合并。递推关系无法形成。

问题分析

考虑放弃click(i, j)的形式， 使用click(i, j, ex_len)

click(i, j, ex_len)表示大块j的右边已经有一个长度为 ex_len 的大块。该大块可能是在合并过程中形成的，且 j 的颜色和 ex_len相同 ，在此情况下将 i 到 j 以及 ex_len 都消除所能得到的最高分。

最终求解的问题变为： click(0, n-1, 0)。

问题分析

假设j 和 ex_len合并后的大块称作Q，对于Q有两种处理方法：

1. 直接消除，能够得到的最高分是click(i, j-1, 0) + (len[j] + ex_len)^2

2. 未来可期，希望Q能够和左边的某个同色大块合并，同样需要枚举可能和Q合并的大块。假设让大块K和Q合并，则此时能够得到的最大分数是：
    click(i, k, len[j] + ex_len) + click_box(k+1, j-1, 0)

递归的终止条件： i == j

问题分析

```
struct Segment{
    int color;
    int len;        //segment在读取输入数据时进行初始化
};
Segment segments[MAXN];
int score[MAXN][MAXN][MAXN];
```

问题分析

```
int Click(int i, int j, int len){
    if(score[i][j][len] != 1)      //减少重复计算
        return score[i][k][len];
    int result = (segments[j].len + len) * (segments[j].len + len);      //合并大块
    if (i == j) return result;
    result += Click(i, j-1, 0);    //选择1，直接消除大块
    for (int k = i; k <= j-1; k++){
        if ( segments[k].name != segments[j].name) continue;
        int r = Click(k+1, j-1, 0) + Click(i, k, segments[j].len + len);
        result = max(result, r); //选择2，枚举后合并更大的块
    }
    score[i][j][len] = result;
    return result;
}
```

灌溉草场(POJ2373)

在一片草场上有一条长度为 L (1 <= L <= 1,000,000, L 为偶数)的线段。李华的 N (1 <= N <= 1000)头奶牛都沿着草场上这条线段吃草，每头牛的活动范围是一个开区间 (S, E), S, E都是整数。不同奶牛的活动范围可以有重叠。

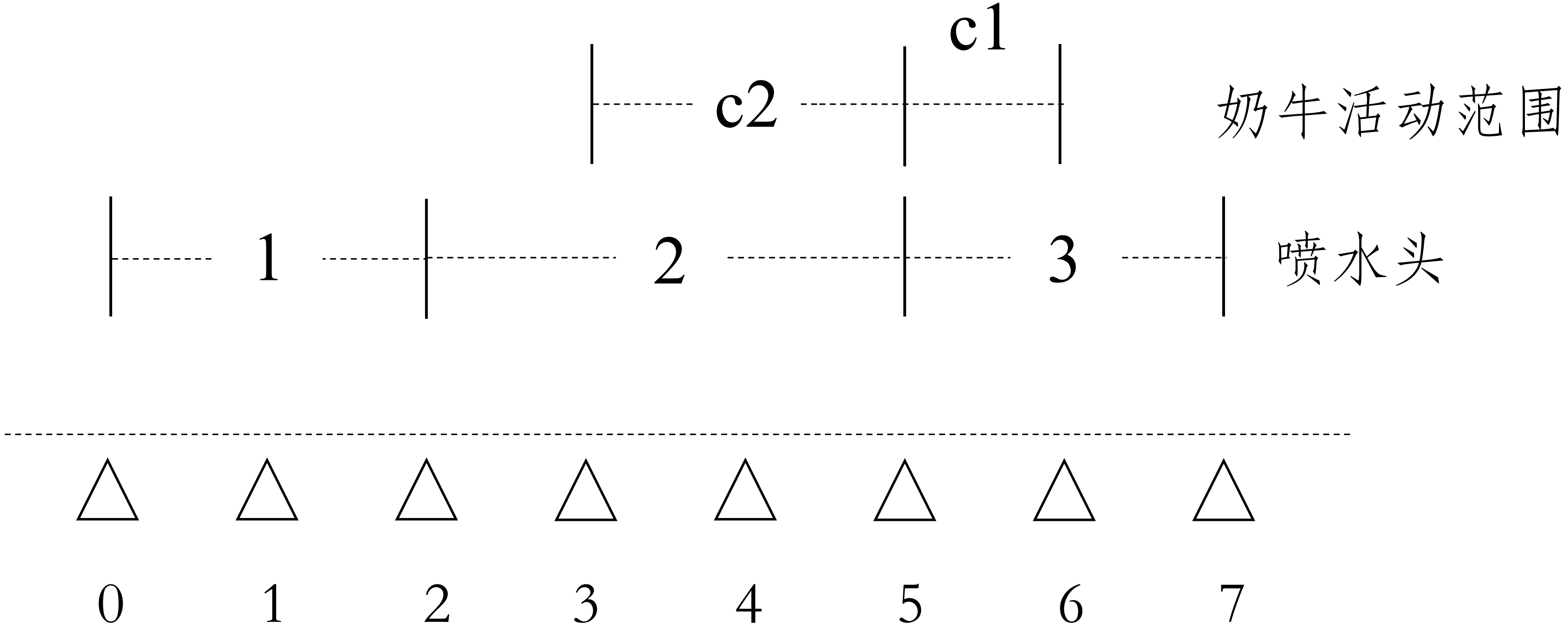
李华要在这条线段上安装喷水头灌溉草场。每个喷水头的喷洒半径可以随意调节，调节范围是[A, B](1 <= A <= B <= 1000), A,B 都是整数。要求:

线段上的每个整点恰好位于一个喷水头的喷洒范围内;
每头奶牛的活动范围要位于一个喷水头的喷洒范围内;
任何喷水头的喷洒范围不可越过线段的两端，左端是 0, 右端是 L。

请问，李华最少需要安装多少个喷水头?

灌溉草场

在位置2和5，喷水头的喷洒范围不算重叠



灌溉草场

输入：
第一行：整数N, L。
第二行：整数A, B。
第三行到第N+2行：每行两个整数S, E，表示某头牛活动范围的起止点在线段上的坐标。
输出：
最少需要安装的喷水头个数。如果没有符合要求的安装方案，则输出-1。

输入样例：
2 8
1 2
6 7
3 6

输出样例：
3

问题分析

从线段的起点向终点安装喷水头，令 f(X) 表示：所安装喷水头的喷洒范围恰好覆盖直线上的区间 [0, X] 时，最少需要多少个喷水头

X需要满足下列条件：

1. X是偶数
2. X所在位置不会出现奶牛，即X不会截断任何一个(S，E)
3. X >= 2A
4. 当X > 2B时，存在Y ∈ [X-2B, X-2A] 且 Y 满足上述三个条件，使得 f(X)=f(Y)+1

问题分析

递推计算 f(X):

1. $f(X) = \infty$， X 是奇数

2. $f(X) = \infty$， $X < 2A$

3. $f(X) = \infty$，X 处可能有奶牛出没

4. $f(X)=1$， $2A \le X \le 2B$，且 X 位于任何奶牛的活动范围之外

5. $f(X)=1+\min\{f(Y): Y\in[X-2B, X-2A]$, Y 位于任何奶牛的活动范围外$\}$， $X > 2B$

问题分析

f(X)=1+min{f(Y): Y∈[X-2B, X-2A], Y 位于任何奶牛的活动范围外}， X >2B

对每个 X 求 f(X)，都要遍历区间[X-2B, X-2A]去寻找其中最小的f(Y), 则时间复杂度为： L * B = 1000000 * 1000 ，TE

如何快速找到 [X-2B， X-2A] 中使得 f(Y) 最小的元素Y?

问题分析

f(X)=1+min{f(Y): Y∈[X-2B, X-2A], Y 位于任何奶牛的活动范围外}， X >2B

对每个 X 求 f(X)，都要遍历区间[X-2B, X-2A]去寻找其中最小的f(Y), 则时间复杂度为： L * B = 1000000 * 1000 ，TE

如何快速找到 [X-2B， X-2A] 中使得 f(Y) 最小的元素Y?

可以使用优先队列 priority_queue(multiset 也可以，比priority_queue 慢一点)

求 F(X) 时，若坐标属于 [X-2B, X-2A] 的二元组 (i,F(i)) 都保存在一个 priority_queue 中 并根据 F(i) 值排序，则队头的元素就能确保是 F(i) 值最小的。

问题分析

在求 X 点的 F(X) 时，必须确保队列中包含所有属于 [X-2B,X-2A] 的点。而且，不允许出现坐标大于 X-2A 的点，因为这样的点对求 F(X) 无用。如果这样的点出现在队头，因其对求后续点的 F 值有用，故不能抛弃，算法就无法继续了。

队列中可以出现坐标小于 X-2B 的点。这样的点若出现在队头，则直接将其抛弃，后续也不会再利用到。

求出 X 点的 F 值后，将 (X-2A+2, F(X-2A+2)) 放入队列，为求 F(X+2) 作准备。

队列里只要存坐标为偶数的点即可。

问题分析

```
int F[MAXL];                //最终目的是求F[L]
int cowExist[MAXL];         //记录i点是否有奶牛
int N, L, A, B;
struct Fx{
    int x;
    int f;
    bool operator<(const Fx & a) const{
        return f > a.f;
    }
    Fx(int xx=0, int ff=0): x(xx), f(ff) { }
};
priority_queue<Fx> qFx;     //优先队列定义为f越小越靠前
```

问题分析

```
for (int i = 2*A; i <= 2*B; i += 2)
    if (! cowExist[i]){
        F[i] = 1;
        if (i <= 2*B + 2 - 2*A) qFx.push(Fx(i, 1)); }
for (int i = 2*B + 2; i <= L; i += 2){
    if (!cowExist[i]) {
        Fx fx;
        while (!qFx.empty()){
            fx = qFx.top();
            if (fx.x < i - 2*B) qFx.pop();
            else break; }
        if (!qFx.empty()) F[i] = fx.f + 1;
    }
```

初始化优先队列，确保队列中的点x <= i - 2A

f(X)=1+min{f(Y): Y∈[X-2B, X-2A], Y 位于任何奶牛的活动范围外}， X>2B

问题分析

队列中增加一个可达下个点的点

```
    if (F[i - 2*A + 2] != INF){
        qFx.push(Fx(i - 2*A + 2, F[i - 2*A + 2]));
    }
}
if (F[L] == INF)
    cout << -1 << endl;
else
    cout << F[L] << endl;
```

手动实现优先队列的方法

如果一个队列满足以下条件:

1.开始为空
2.每在队尾加入一个元素 a 之前，都从现有队尾往前删除元素，一直删到碰到小于 a 的元素为止，然后再加入 a

那么队列就是递增的，队头的元素一定是队列中最小的

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

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。

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

炮兵阵地

输入

第一行包含两个由空格分割开的正整数，分别表示N和M；

接下来的N行，每一行含有连续的M个字符('P'或者'H')，中间没有空格。按顺序表示地图中每一行的数据。N <= 100；M <= 10。

输出

仅一行，包含一个整数K，表示最多能摆放的炮兵部队的数量。

样例输入

```
5 4
PHPP
PPHH
PPPP
PHPP
PHHP
```

样例输出

```
6
```

问题分析

如果使用dp[i]表示前i行存放的最多炮兵数目，是否能满足递推关系?

问题分析

如果使用dp[i]表示前i行存放的最多炮兵数目，是否能满足递推关系？不能，放置方法会影响到后续放置。除了数目，还需要记录摆放的状态。

问题分析

如果使用dp[i]表示前i行存放的最多炮兵数目，是否能满足递推关系？不能，放置方法会影响到后续放置。除了数目，还需要记录摆放的状态。

添加一个维度来处理多余的限制条件，使用dp[i][j]表示前i行在j的布局下最多能存放多少炮兵。

j同样使用一种简单编码来表示，不然需要存放二维数组画图。这里同样考虑二进制数，只需要10位(M <= 10)，用0,1代表对应位置是否存放炮兵。

问题分析

如果使用dp[i]表示前i行存放的最多炮兵数目，是否能满足递推关系？不能，放置方法会影响到后续放置。除了数目，还需要记录摆放的状态。

添加一个维度来处理多余的限制条件，使用dp[i][j]表示前i行在j的布局下最多能存放多少炮兵。

j同样使用一种简单编码来表示，不然需要存放二维数组画图。这里同样考虑二进制数，只需要10位(M <= 10)，用0,1代表对应位置是否存放炮兵。

还是不满足条件！仅从dp[i-1][k](k = 0, ..., 1023)无法推导出dp[i][k]，因为i-2行的布局也会影响到第i行。

问题分析

dp[i][j][k]，表示第i行布局为j，第i-1行布局为k时，前i行最多的炮兵数目。必须满足：

1. j, k相容，否则dp[i][j][k] = 0

2. dp[i][j][k] = max { dp[i-1][k][m], m = 0, ..., 1023 } + Num(j)。Num(j)为布局j中炮兵的数目，j必须与m相容，k也必须与m相容。

3. dp[0][j][0] = Num(j)
   dp[1][i][j] = max{ dp[0][j][0] } + Num(i)

问题分析

进一步问题： dp[100][1024][1024]可能还是过大，需要进一步压缩

问题分析

进一步问题： dp[100][1024][1024]可能还是过大，需要进一步压缩

每一行全是平地的情况下，每行最多放置4个炮兵，炮兵放置方案不超过60。

可以计算每一行在全部平地下的所有炮兵方案，并存入数组status[70]，在进行dp时直接进行dp[100][70][70]，进一步压缩。dp[i][j][k]表示第i行布局为status[j]，第i-1行布局为status[k]时，前i行最多的炮兵数目。