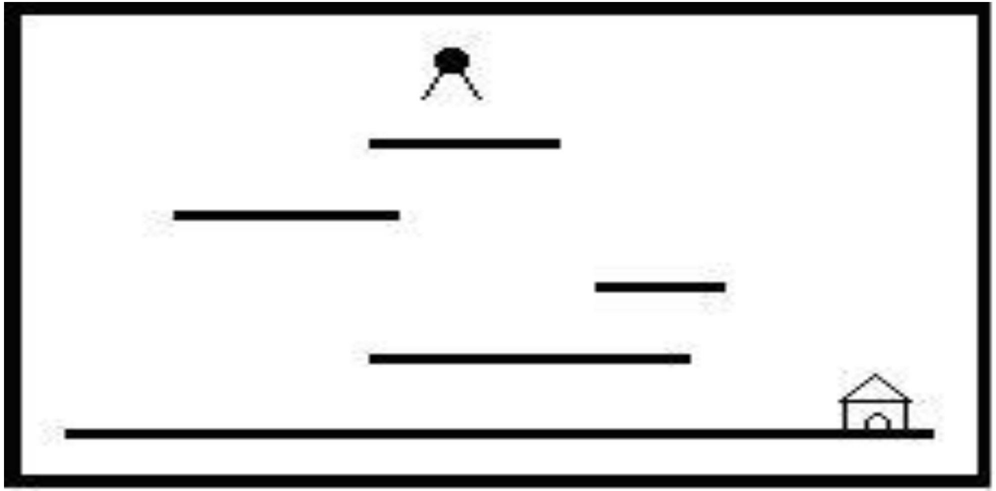
程序设计与算法(二)

郭 炜

动态规划(二)

Help Jimmy(POJ1661)

"Help Jimmy" 是在下图所示的场景上完成的游戏：



Help Jimmy(POJ1661)

场景中包括多个长度和高度各不相同的平台。

地面是最低的平台，高度为零，长度无限。

Jimmy老鼠在时刻0从高于所有平台的某处开始下落，它的下落速度始终为1米/秒。当Jimmy落到某个平台上时，游戏者选择让它向左还是向右跑，它跑动的速度也是1米/秒。当Jimmy跑到平台的边缘时，开始继续下落。Jimmy每次下落的高度不能超过MAX米，不然就会摔死，游戏也会结束。

设计一个程序，计算Jimmy到地面时可能的最早时间。

Help Jimmy(POJ1661)

输入数据

第一行是测试数据的组数t（0 <= t <= 20）。每组测试数据的第一行是四个整数N，X，Y，MAX，用空格分隔。N是平台的数目（不包括地面），X和Y是Jimmy开始下落的位置的横竖坐标，MAX是一次下落的最大高度。接下来的N行每行描述一个平台，包括三个整数，X1[i]，X2[i]和H[i]。H[i]表示平台的高度，X1[i]和X2[i]表示平台左右端点的横坐标。1 <= N <= 1000，-20000 <= X，X1[i]，X2[i] <= 20000，0 < H[i] < Y <= 20000（i = 1..N）。所有坐标的单位都是米。

Jimmy的大小和平台的厚度均忽略不计。如果Jimmy恰好落在某个平台的边缘，被视为落在平台上。所有的平台均不重叠或相连。测试数据保Jimmy一定能安全到达地面。

Help Jimmy(POJ1661)

输出要求

对输入的每组测试数据，输出一个整数，

Jimmy到地面时可能的最早时间。

输入样例

1

3 8 17 20

0 10 8

0 10 13

4 14 3

输出样例

23

解题思路

Jimmy跳到一块板上后，可以有两种选择，向左走，或向右走。
走到左端和走到右端所需的时间，是很容易算的。
如果我们能知道，以左端为起点到达地面的最短时间，和以右端为起点到达地面的最短时间，那么向左走还是向右走，就很容选择了。
因此，整个问题就被分解成两个子问题，即Jimmy所在位置下方第一块板左端为起点到地面的最短时间，和右端为起点到地面的最短时间。
这两个子问题在形式上和原问题是完全一致的。将板子从上到下从1开始进行无重复的编号(越高的板子编号越小，高度相同的几块板子，哪块编号在前无所谓)，那么，和上面两个子问题相关的变量就只有板子的编号。

不妨认为Jimmy开始的位置是一个编号为0，长度为0的板子，假设LeftMinTime(k)表示从k号板子左端到地面的最短时间，RightMinTime(k)表示从k号板子右端到地面的最短时间，那么，求板子k左端点到地面的最短时间的方法如下：

```
if ( 板子k左端正下方没有别的板子) {
    if( 板子k的高度 h(k) 大于Max)
        LeftMinTime(k) = ∞;
    else
        LeftMinTime(k) = h(k);
}
else if( 板子k左端正下方的板子编号是m )
    LeftMinTime(k) = h(k)-h(m) +
      Min( LeftMinTime(m) + Lx(k)-Lx(m),
           RightMinTime(m) + Rx(m)-Lx(k));
}
```

上面，h(i)就代表i号板子的高度，Lx(i)就代表i号板子左端点的横坐标，Rx(i)就代表i号板子右端点的横坐标。那么 h(k)-h(m)　当然就是从k号板子跳到m号板子所需要的时间，Lx(k)-Lx(m) 就是从m号板子的落脚点走到m号板子左端点的时间，Rx(m)-Lx(k)就是从m号板子的落脚点走到右端点所需的时间。

求RightMinTime(k)的过程类似。

不妨认为Jimmy开始的位置是一个编号为0，长度为0的板子，那么整个问题就是要求LeftMinTime(0)。

输入数据中，板子并没有按高度排序，所以程序中一定要首先将板子排序。

时间复杂度：

一共 n个板子，每个左右两端的最小时间各算一次 O(n)

找出板子一段到地面之间有那块板子，需要遍历板子 O(n)

总的时间复杂度O(n^2)

例五、神奇的口袋(百练2755)

- 有一个神奇的口袋，总的容积是40，用这个口袋可以变出一些物品，这些物品的总体积必须是40。
- John现在有n（1≤n ≤ 20）个想要得到的物品，每个物品的体积分别是a_1，a_2……a_n。John可以从这些物品中选择一些，如果选出的物体的总体积是40，那么利用这个神奇的口袋，John就可以得到这些物品。现在的问题是，John有多少种不同的选择物品的方式。

- **输入**

输入的第一行是正整数n (1 <= n <= 20)，表示不同的物品的数目。接下来的n行，每行有一个1到40之间的正整数，分别给出a_1，a_2……a_n的值。

- **输出**

输出不同的选择物品的方式的数目。

- **输入样例**

3
20
20
20

- **输出样例**

3

枚举的解法：

枚举每个物品是选还是不选，共2^{20}种情况

递归解法

```
#include <iostream>
using namespace std;
int a[30];  int N;
int Ways(int w ,int k )  { // 从前k种物品中选择一些，凑成体积w的做法数目
        if( w == 0 )    return 1;
        if( k <= 0 )    return 0;
        return Ways(w, k -1 ) + Ways(w - a[k], k -1 );
}
int main()    {
        cin >> N;
        for( int i = 1;i <= N; ++ i )
                cin >> a[i];
        cout << Ways(40,N);
        return 0;
}
```

动规解法

```
#include <iostream>
using namespace std;
int a[30];  int N;
int Ways[50][40];//Ways[i][j]表示从前j种物品里凑出体积i的方法数
int main()    {
        cin >> N;
        memset(Ways,0,sizeof(Ways));
        for( int i = 1;i <= N; ++ i ) {
                cin >> a[i];   Ways[0][i] = 1;
        }
        Ways[0][0] = 1;
        for( int w = 1 ; w <= 40; ++ w ) {
                for( int k = 1; k <= N; ++ k ) {
                        Ways[w][k] = Ways[w][k-1];
                        if( w-a[k] >= 0)
                                Ways[w][k] += Ways[w-a[k]][k-1];
                }
        }
        cout << Ways[40][N];
        return 0;
}
```

例六、0-1背包问题(POJ3624)

有N件物品和一个容积为M的背包。第i件物品的体积w[i]，价值是d[i]。求解将哪些物品装入背包可使价值总和最大。每种物品只有一件，可以选择放或者不放

(N<=3500,M <= 13000)。

0-1背包问题(POJ3624)

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

边界：
```
if (w[1] <= j)
        F[1][j] = d[1];
    else
        F[1][j] = 0;
```

0-1背包问题(POJ3624)

用 F[i][j] 表示取前i种物品，使它们总体积不超过j的最优取法取得的价值总和

递推： F[i][j] = max(F[i-1][j],F[i-1][j-w[i]]+d[i])

取或不取第 i种物品，两者选优

(j-w[i] >= 0才有第二项)

0-1背包问题(POJ3624)

F[i][j] = max(F[i-1][j],F[i-1][j-w[i]]+d[i])

本题如用记忆型递归，需要一个很大的二维数组，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。即可以用一维数组，用“人人为我”递推型动归实现。

例七、滑雪(百练1088)

Michael喜欢滑雪百这并不奇怪， 因为滑雪的确很刺激。

可是为了获得速度，滑的区域必须向下倾斜，而且当你滑到坡底，

你不得不再次走上坡或者等待升降机来载你。

Michael想知道载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。事实上，这是最长的一条。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。

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，L(i,j) = 1

否则

递推公式： L(i,j) 等于(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

复杂度：$O(n^2)$

解题思路

解法1） “人人为我”式递推

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，L(i,j) = 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

解法2）“我为人人”式递推

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，L(i,j) = 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，要更新他周围的，比它高的点的L值。例如：

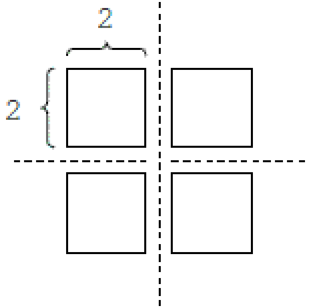
if H(i+1,j) > H(i,j) // H代表高度

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

例八、分蛋糕

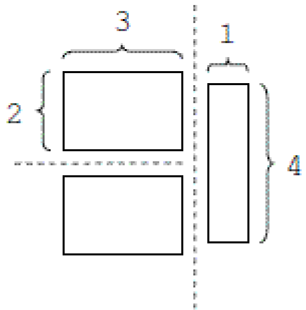
- 问题描述

- 有一块矩形大蛋糕，宽和高分别是整数w 、h。现要将其切成m块小蛋糕，每个小蛋糕都必须是矩形、且宽和高均为整数。切蛋糕时，每次切一块蛋糕，将其分成两个矩形蛋糕。请计算：最后得到的m块小蛋糕中，最大的那块蛋糕的面积下限。
- 假设w= 4，h= 4，m= 4，则下面的切法可使得其中最大蛋糕块的面积最小。



例八、分蛋糕

- 假设w = 4，h = 4，m = 3，则下面的切法可使得其中最大蛋糕块的面积最小。



例八、分蛋糕

- 输入

共有多行，每行表示一个测试案例。每行是三个用空格分开的整数W，H，M，其中$1 \leqslant W, H, M \leqslant 20$，$M \leqslant WH$. 当 $W = H = M = 0$ 时不需要处理，表示输入结束。

- 输出

每个测试案例的结果占一行，输出一个整数，表示最大蛋糕块的面积下限。

- 样例输入

4 4 4

4 4 3

0 0 0

- 样例输出

4

6

例八、分蛋糕

- 解题思路
- 设 ways(w,h,m)表示宽为w,高为h的蛋糕，被切m刀后，最大的那块蛋糕的面积最小值
- 题目就是要求 ways(W,H,M-1)

边界条件：

w * h < m + 1	INF
m == 0	w*h

例八、分蛋糕

递推式：

SV为第一刀竖着切时能得到的最好结果，SH为第一刀横着切时能得到的最好结果，则ways(w,h,m) = min(SV,SH)

SV = min{ S_i, i = 1 ... w-1 },

其中： S_i = 为第一刀左边宽为i的情况下的最好结果

例八、分蛋糕

递推式：

SV为第一刀竖着切时能得到的最好结果，SH为第一刀横着切时能得到的最好结果，则ways(w,h,m) = min(SV,SH)

$SV = \min\{ S_i,\ i = 1 \ldots w-1 \}$,

其中： S_i = 为第一刀左边宽为i的情况下的最好结果

$S_i = \min\{ \max(ways(i,h,k), ways(w-i,h,m-1-k)),\ k = 0 \ldots m-1 \}$