程序设计与算法(二)
算法基础

郭 炜

微信公众号

微博：http://weibo.com/guoweiofpku

学会程序和算法，走遍天下都不怕!

讲义照片均为郭炜拍摄

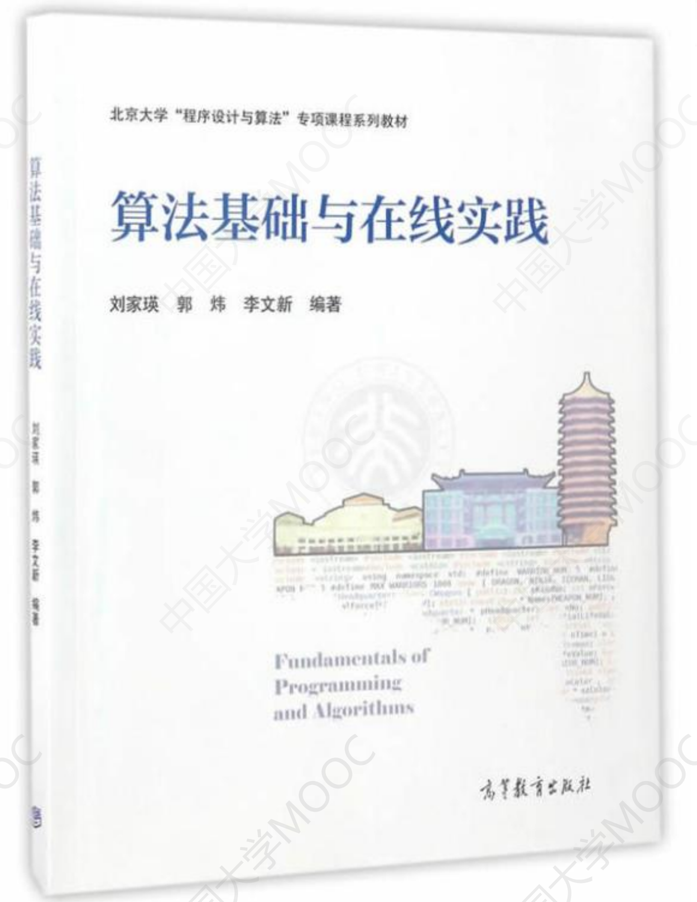
配套教材：

高等教育出版社

《算法基础与在线实践》

刘家瑛 郭炜 李文新 编著

本讲义中所有例题，根据题目名称在
http://openjudge.cn
“百练”组进行搜索即可提交



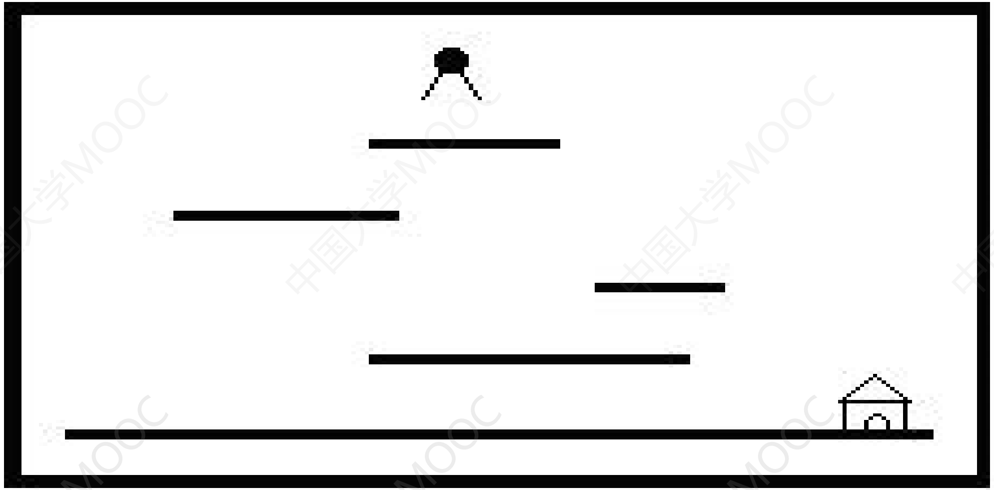
动态规划(二)

例题
Help Jimmy

圣彼得堡彼得霍夫宫

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$O(n^2)$

例题
神奇的口袋

圣彼得堡阿芙乐尔号巡洋舰

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（$1 \le n \le 20$）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][50];//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;
}
```

例题
Charm Bracelet

美国圣地亚哥中途岛号航母

例六、 Charm Bracelet 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。

例题:滑雪

圣彼得堡炮兵博物馆

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)

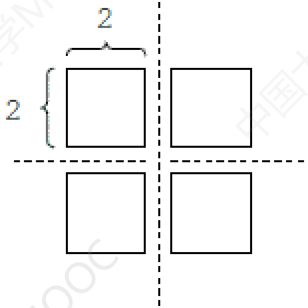
例题:分蛋糕

瑞典斯德哥尔摩

例八、分蛋糕

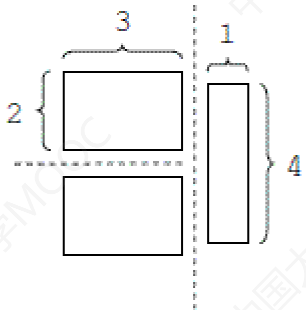
- 问题描述

- 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 ≤ W, H, M ≤ 20 ， M ≤ 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 ... 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... m-1 }

例题：灌溉草场

瑞士马特洪峰

例九、Dividing the Path 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。 John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。

John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， John 最少需要安装多少个喷水头。

Dividing the Path 灌溉草场(POJ2373)

```
            |---- c2----|-c1|      奶牛活动范围
|--- 1 ---|-------  2 -------|---3 ---|    喷水头
+---+---+---+---+---+---+---+---+
0   1   2   3   4   5   6   7   8
```

在位置2和6，喷水头的喷洒范围不算重叠

- **输入**

  第1行：整数N、L。

  第2行：整数A、B。

  第3到N+2行：每行两个整数S、E (0 <= S < E <= L)，表示某头牛活动范围的起点和终点在线段上的坐标(即到线段起点的距离)。

- **输出**：最少需要安装的多少个喷水头；若没有符合要求的喷水头安装方案，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
- 显然，X应满足下列条件
  - X为偶数
  - X所在位置不会出现奶牛，即X不属于任何一个(S,E)
  - X≥2A
  - 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

解题思路

- 递推计算f(X)
  - $f(X) = \propto$ ： X 是奇数
  - $f(X) = \propto$ ： $X < 2A$
  - $f(X) = \propto$ ：X处可能有奶牛出没
  - $f(X)=1$： $2A \le X \le 2B$ 、且X位于任何奶牛的活动范围之外
  - $f(X)=1+\min\{f(Y): Y \in [X-2B \quad X-2A]$、Y位于任何奶牛的活动范围之外$\}$： $X>2B$

解题思路

f(X)=1+min{f(Y): Y∈[X-2B   X-2A]、Y位于任何奶牛的活动范围之外}： 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，太慢
- 快速找到[X-2B   X-2A]中使得f(Y)最小的元素是问题求解速度的关键 。

解题思路

- 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。

解题思路

- 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，故不能抛弃之，于是算法就无法继续了。

- 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。

- 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

```
#include <iostream>
#include <cstring>
#include <queue>
using namespace std;
const int INFINITE = 1<<30;
const int MAXL = 1000010;
int F[MAXL];  // F[L]就是答案
int cowThere[MAXL]; //cowThere[i]为1表示点i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) { }
};// 在优先队列里，f值越小的越优先
priority_queue<Fx> qFx;
```

```
int main()
{
        cin >> N >> L;
        cin >> A >> B;
        A <<= 1; B <<= 1; //A,B的定义变为覆盖的直径
        memset(cowThere,0,sizeof(cowThere));
        for( int i = 0;i < N; ++i ) {
                int s,e;
                cin >> s >> e;
                ++cowThere[s+1]; //从s+1起进入一个奶牛区
                --cowThere[e]; //从e起退出一个奶牛区
        }
        int  inCows = 0; //表示当前点位于多少头奶牛的活动范围之内
        for( int i = 0;i <= L ; ++i) { //算出每个点是否有奶牛
                F[i] = INFINITE;
                inCows += cowThere[i];
                cowThere[i] = inCows > 0;
        }
```

```
for( int i = A;i <= B ; i += 2 ) //初始化队列
        if(! cowThere[i] ) {
                F[i] = 1;
                if( i <= B + 2 - A )
                //在求F[i]的时候，要确保队列里的点x， x <= i - A
                        qFx.push(Fx(i,1));
        }
for( int i = B + 2 ; i <= L; i += 2 ) {
        if( !cowThere[i] ) {    Fx fx;
                while(!qFx.empty()) {
                        fx = qFx.top();
                        if( fx.x < i - B )
                                qFx.pop();
                        else
                                break;
                }
                if ( ! qFx.empty() )
                        F[i] = fx.f + 1;
        }
```

```
            if( F[i- A + 2] != INFINITE) {
                //队列中增加一个+1可达下个点的点
                qFx.push(Fx(i- A + 2, F[i- A + 2]));
            }
    }
    if( F[L] == INFINITE )
            cout  << -1 <<endl;
    else
            cout << F[L] << endl;
    return 0;
} // 复杂度：O(nlogn)
```

手工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，当然队头的元素，一定是队列中最小的

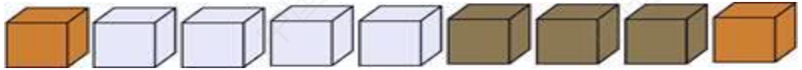
例题:方盒游戏


瑞典斯德哥尔摩

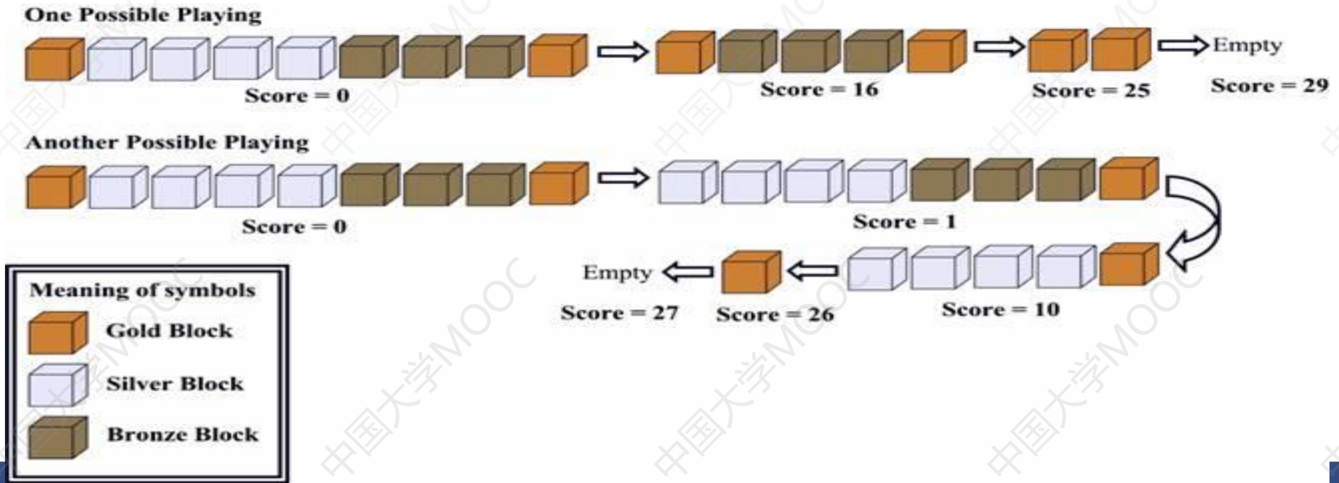
例八、POJ1390 方盒游戏

- 问题描述

N个方盒(小块)摆成一排，每个小块有自己的颜色。连续摆放的同颜色小块构成一个"大块" 。下图中共有四个大块，每个大块分别有1、4、3、1个小块



玩家每次点击一个小块，则该小块所在大块就会消失。若消失的大块中共有k个小块，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?
- 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),，表示共有多少个小块
  - 第二行包括n个整数，表示每个小块的颜色。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**

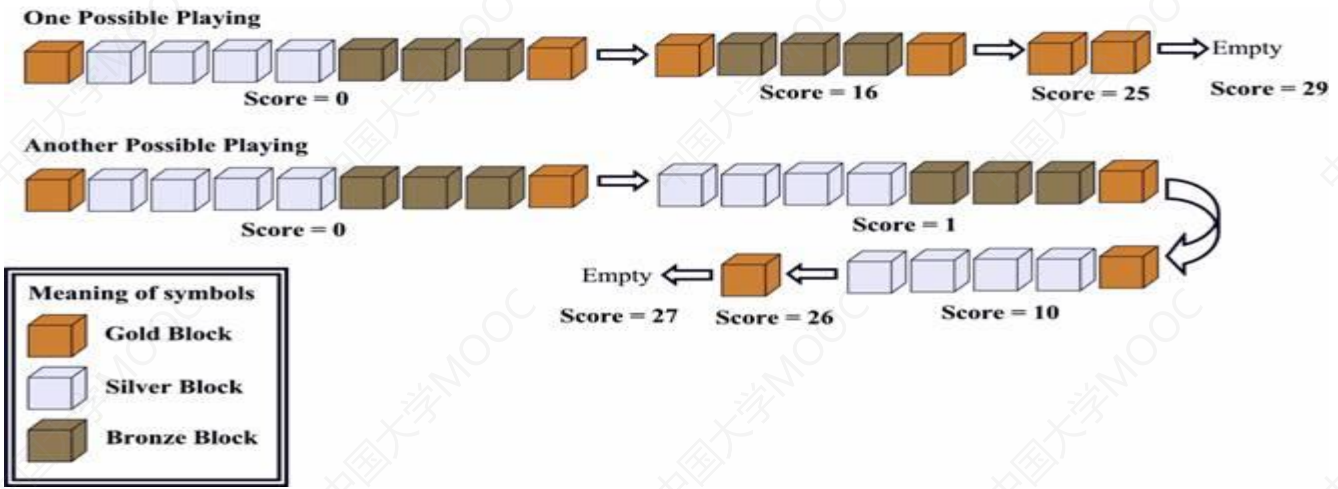
2
9
1 2 2 2 2 3 3 3 1
1
1

- **样例输出**

Case 1: 29
Case 2: 1

问题分析

- 当同色的小块摆放在不连续的位置时，小块的点击顺序影响玩家获得的积分



- 同种颜色的小块被点击的次数越少，玩家获得的积分越高
- 明显的递归问题：每次点击之后，剩下的小块构成一个新的小块队列，新队列中小块的数量减少了。然后计算玩家从新队列中可获得的最高积分

问题分析

- 点击下图中黑色小块之前，先点击绿色小块可提高玩家的积分：同颜色小块A和B被其他颜色的小块隔开时，先点击其他颜色小块，使得A和B消失前能够在同一个大块中
- 点击下图中红色和蓝色小块可获得的积分
  - 所有红色小块合并到同一个片段：49+1+36=86
  - 所有蓝色小块合并到同一个片段：49+16+9=74

问题分析

- 思路:

将连续的若干个方块作为一个“大块”考虑，假设开始一共有 n个“大块”，编号0到n-1

第i个大块的颜色是 color[i],包含的方块数目，即长度,是len[i]

用ClickBox(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

则整个问题就是： ClickBox(0,n-1)

问题分析

要求ClickBox(i,j)时，考虑最右边的大块j，对它有两种处理方式，要取其优者：

1) 直接消除它，此时能得到最高分就是：

ClickBox(i,j-1) + len[j]*len[j]

2) 期待以后它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，到底和哪个合并最好，不知道，只能枚举。假设大块j和左边的大块k(i<=k<j-1) 合并，此时能得到的最高分是多少呢?

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，到底和哪个合并最好，不知道，只能枚举。假设大块j和左边的大块k(i<=k<j-1) 合并，此时能得到的最高分是多少呢?

是不是:

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

问题分析

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

不对!

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，才符合上述式子，但直接将其消去，未必是最好的，也许它还应该和左边的同色大块合并，才更好

递推关系无法形成，怎么办?

问题分析

需要改变问题的形式。

ClickBox(i,j) 这个形式不可取，因为无法形成递推关系

考虑新的形式：

ClickBox(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，不妨就称其为ex_len)，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。

于是整个问题就是求：ClickBox(0,n-1,0)

问题分析

求ClickBox(i,j,ex_len)时，有两种处理方法，取最优者

假设j和ex_len合并后的大块称作 Q

1) 将Q直接消除，这种做法能得到的最高分就是：

ClickBox(i,j-1,0) + $(len[j]+ex\_len)^2$

2) 期待Q以后能和左边的某个同色大块合并。需要枚举可能和Q合并的大块。假设让大块k和Q合并，则此时能得到的最大分数是：

ClickBox(i,k,len[j]+ex_len) + ClickBox(k+1,j-1,0)

递归的终止条件是什么？

问题分析

ClickBox(i,j,ex_len) 递归的终止条件：

i == j

```
#include <iostream>
#include <cstring>
using namespace std;
const int M = 210;
struct Segment { //Segment就是大块
   int color;
   int len;
};
Segment segments[M];
int score[M][M][M];
```

```
int ClickBox(int i,int j,int len)  {
    if( score[i][j][len] != -1)
         return score[i][j][len];
    int result = (segments[j].len + len)  *
                 (segments[j].len + len);
    if( i == j )
         return result;
    result += ClickBox(i,j-1,0);
    for(int k = i;k <= j-1; ++k ) {
         if( segments[k].color != segments[j].color )
                continue;
         int r = ClickBox(k+1,j-1,0);
         r += ClickBox(i,k,segments[j].len + len);
         result = max(result,r);
    }
```

```
    score[i][j][len] = result;
    return result;
}
int main()
{
    int T;
    cin >> T;
    for(int t = 1; t <= T; ++ t) {
        int n;
        memset(score,0xff,sizeof(score));
        cin >> n;
        int lastC = 0;
        int segNum = -1;
```

```
        for(int i = 0;i < n; ++ i) {
                int c;
                cin >> c;
                if( c != lastC ) {
                        segNum ++;
                        segments[segNum].len = 1;
                        segments[segNum].color = c;
                        lastC  = c;
                }
                else segments[segNum].len ++;
        }
  cout << "Case " << t << ": " << ClickBox(0,segNum,0) << endl;
    }
    return 0;
}
```