华中科技大学

2023

算法设计与分析实验报告

专　　业：	计算机科学与技术
班　　级：	CS2207
学　　号：	U202215554
姓　　名：	付名扬
完成日期：	2024.1.6

目　录

1. 完成情况

1.1 实践内容

★本次实验一共完成了 24 道题目★

其中包括：**北京大学程序在线评测系统 Peking University Online Judge (POJ)** 平台上的 23 道题目，以及**洛谷 luogu** 平台上的 1 道题目。所有题目均来自于课程“**算法设计与分析实践 Algorithmic Design & Analysis**”所提供的学生用文件内要求选做的部分题目。完成题目的范畴为分治策略、动态规划、贪心算法、最短路径等理论课上讲授过的内容。详细完成情况如图 1-1 所示。

图 1-1 题目完成情况

1.2 报告内容

对于分治、动态规划、贪心算法、最短路径这四个解决问题的经典算法，分别选取了四道典型代表题目 **POJ.3269 Building A New Barn**、**POJ.1088 Snowboarding**、**POJ.1328 Radar Installation**、**POJ.1860 Currency Exchange**，分别对应分治、动态规划、贪心算法、最短路径。

首先对每道题目进行分析和建模，抽象出数据结构，进而进行算法设计，在设计的过程中附带流程图、伪代码、图解、图表等表现方式进行辅助理解。然后分析算法的时间复杂度和空间复杂度，并专门对每一个题目绘制了综合评估表进行分数评估。最后进行输入输出样例测试。

完成算法分析设计后，对报告进行了总结，说明自己收获与解题心得。

2. POJ.3269 Building A New Barn

经过多年的节衣缩食，农场主约翰决定建造一座新牛舍。他希望牛舍交通便利，而且他知道所有 N 头奶牛（$2 \le N \le 10,000$ 头）的放牧点坐标。每个放牧点的坐标都是整数（x_I, y_i）（$-10,000 \le x_I \le 10,000$；$-10,000 \le y_I \le 10,000$）。饥饿的奶牛从不在水平或垂直相邻的位置吃草。

牛舍必须放置在整数坐标上，并且不能位于任何奶牛的放牧点上。牛舍对任何一头奶牛的不便程度由曼哈顿距离公式 $|x - x_I| + |y - y_i|$ 得出，其中 (x, y) 和 (x_I, y_i) 分别是牛舍和奶牛放牧点的坐标。为了使牛舍对所有奶牛造成的不便之和最小，牛舍应该建在哪里？

2.1 题目分析

基于题干信息和输入输出信息，可以得到如下条件：

（1） 题目具体要求：给出 N 个互不相同的点的位置坐标，求一点到其它点曼哈顿距离总和**最小值**以及**解的个数**。

（2） N 个点的坐标是整数，且**不存在**不同的两点 $P_i(x_I, y_i)$与 $P_j(x_j, y_j)$，使得$|x_j - x_I| = 1$ 或$|y_j - y_I| = 1$，理由见题干首段末句“**从不在水平或垂直相邻的位置吃草。**”

（3） **曼哈顿距离** $d = |x - x_I| + |y - y_i|$ 是本题要求的距离表达形式。如图 2-1 所示。要注意不是欧氏距离！

通过上方分析，可以将题目进一步抽象转换为：求一坐标点 P (x , y)，使得点 P 到其他所有点的曼哈顿距离之和 $D = \sum_{i=1}^{n} |x - x_I| + |y - y_i|$ 的值最小。

给定二维坐标系上两点 P 和 Q，P 到 Q 的曼哈顿距离示意图如图 2-1 左所示，P 出发的路径可以理解为一个沿着 x 或 y 方向的坐标线路进行的垂直路程，因此对于给定的点 P 和 Q，它们之间的曼哈顿距离的路径形状可能会有多种不同的形式。图中给出了红色、绿色和蓝色三条路径。

回到问题本身，题目需要求牛棚（即出发点 P）到不同坐标位置的牛（即不同的目的地 Q）之间的曼哈顿距离之和的最小值，如图 2-1 右所示。

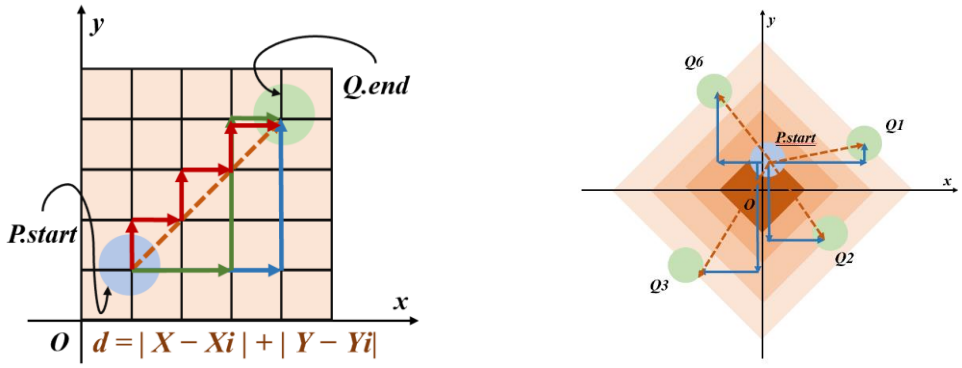


图 2-1 POJ.3269 曼哈顿距离示意图

2.2 算法设计

由于 x 和 y 方向具有一定的独立性，因此可以将问题再分割为单独求 $\sum_{i=1}^{n}|x-x_i|$ 与 $\sum_{i=1}^{n}|y-y_i|$ 的最小值。

函数 $f(x)=\sum_{i=1}^{n}|x-x_i|$ 取最小值当且仅当 x 取序列{X_n}的中位数（证明见 2.5）。如果 x 无法取到其中位数，例如有一些已经给定的点的横坐标和纵坐标恰好就是{X_n}序列和{Y_n}序列的中位数，则尽可能让 x 接近其中位数即可。

实际操作中，由于 N 分为奇数和偶数，其采取的策略也不同：N 是奇数时，x 和 y 坐标序列的中位数是一个确定的值。如果中位数上的坐标被占用了，那么由于“奶牛从不在水平或垂直相邻的位置吃草”，可以枚举该坐标的上下左右四个方向的坐标。N 是偶数时，存在两个中位数的 x，y 坐标，可以构成一个矩形。由于曼哈顿距离的性质，矩形内部以及边界的所有点都有相同距离之和，因此都是方案。考虑到部分点可能在这个矩形里面，那么仅用把矩形内所有点的数目减去被占用的点的数目即可，设计图如图 2-2 所示。相关变量定义见 Table1.1。

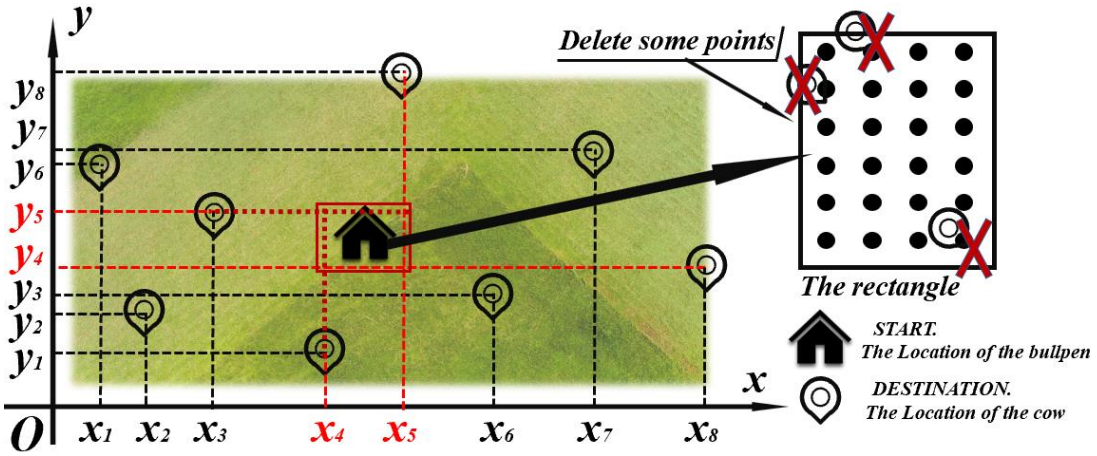


图 2-2 N 为偶数时的设计方案抽象

Table 1.1: Related definitions of variables in POJ.3269

Name	Functionality	Space Complexity
N	Number of Coordinates	O(1)
X[1...N]	Horizontal Coordinate	O(n)
Y[1…N]	Vertical Coordinate	O(n)
MIN	Minimum Distance	O(1)
PLAN	Number of Plans	O(1)

在基于问题分析和算法设计之后，解决问题的流程如下：

Algorithm 1: The Process of POJ.3269

```
Input: N, X[1…N], Y[1…N]
Output: MIN, PLAN
1   MIN = ∞, PLAN = 0
2   if N is an odd number then
3       Let P is the upper quartile of the coordinates
4       if P is already taken then
5           for the direction ↑ , ↓ ,←,→
6               Let dis is the sum of distances
7               if dis < MIN then
8                   MIN = dis
9                   PLAN = 1
10              else if dis = MIN than
11                  PLAN++
12              end if
13          end for
14          return MIN, PLAN
15      else
16          Let dis is the sum of distances
17          PLAN = 1
18          return MIN, PLAN
19      end if
20  else
21      Let P, Q is the upper quartile of the coordinates
22      Let MIN is the sum of distances
23      PLAN = Let PLAN be the number of points inside the rectangle formed by P, Q
24      for i = 1 to N do
25          if (X[i], Y[i]) is in the rectangle then
26              PLAN--
27          end if
28      end for
29      return MIN, PLAN
30  end if
end The process of POJ.3269
```

2.3 性能分析

由 Table 1.1 的 Space Complexity 列可知，该算法的空间复杂度为 O(n)，下面重点分析其时间复杂度。

排序的时间复杂度为 O(nlogn).由于 N 的取值不同，算法分为两种情况，每一个情况可能具有不一样的时间复杂度，因此不妨从 N 的取值进行分别分析。

N 是奇数时，对应给出的算法伪代码的第 2~19 行。4~13 行的枚举的时间复杂度为 O(4) = O(1)，但是每一次计算距离都需要 O(n)时间复杂度；15~17 行同样也需要 O(n)的时间复杂度去计算距离，因此奇数部分时间复杂度为 O(n)。

N 为偶数时，对应给出的算法伪代码的第 20~30 行。计算距离与判断重合部分均需要 O(n)时间复杂度。因此偶数部分时间复杂度为 O(n)。

综合上述分析，算法的空间复杂度是 O(n)，时间复杂度是 O(nlogn)。该算法的综合性能评估如图 2-3 所示。

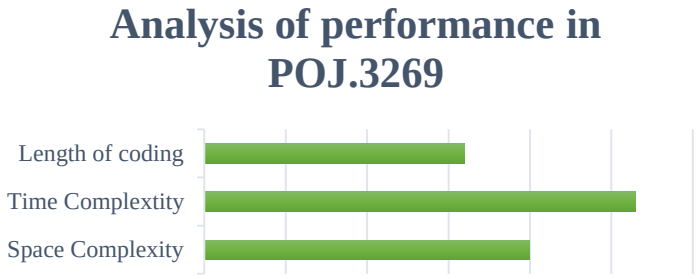


图 2-3 POJ.3269 综合评估

2.4 运行测试

输入：共 N+1 行。第一行是 N 的大小，接着是 N 行，每一行有两个输入，第一个输入是 x 方向的坐标，第二个输入是 y 方向的坐标。

输出：一行，两个数字。第一个是最小距离，第二个是可行方案数目。

样本输入与输出以及实际结果如 Table1.2 所示。

Table 1.2: Operational test in POJ.3269

Sample Input	Sample Output	Output
4 **1 -3** **0 1** **-2 1** **1 -1**	**10 4**	4 1 -3 0 1 -2 1 1 -1 10 4

*2.5 有关函数 $f(x)=|x-a_1|+|x-a_2|+\ldots+|x-a_n|$ 最小值的求法

给定 $f(x)=|x-a_1|+|x-a_2|+\ldots+|x-a_n|$，$x\in\mathbf{R}$，序列$\{a_n\}$为非严格上升序列。现需要求 $f(x)$取最小值时 x 的值或取值范围。

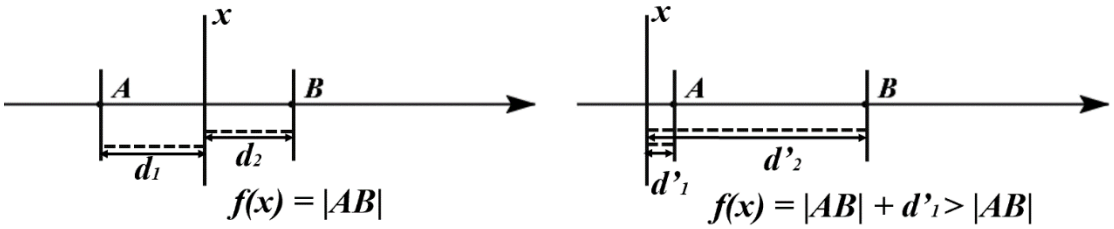
定理：对于 $f(x)=|x-a_1|+|x-a_2|+\ldots+|x-a_n|$，$x\in\mathbf{R}$，序列$\{a_n\}$为非严格上升序列。$n$ 为奇数时，x 取序列$\{a_n\}$的中位数；n 为偶数时，设序列$\{a_n\}$的中位数为a_k和a_{k+1}，$x\in[a_k\ ,\ a_{k+1}]$时均可取最小值。

下面对该定理进行证明。

先考虑最简单的情况：$n=1$ 时，$f(x)=|x-a_1|$，序列$\{a_n\}$的中位数为a_1，$f(a_1)=0$，由于 $f(x)=|x-a_1|\geqslant 0$ 在实数范围内成立，因此 $f(a_1)=0$ 是函数的最小值，定理成立。再考虑 $n=2$ 时，$f(x)=|x-a_1|+|x-a_2|$，序列$\{a_n\}$的中位数为 a_1 和a_2，直接观察不容易得出 x 的取值或取值范围，需要对问题进行转换。

考虑绝对值的几何意义：绝对值的几何意义是表示数轴上数与数之间的距离。从该角度考虑，$f(x)=|x-a_1|+|x-a_2|$的取值是数轴上一动点$(x\ ,0)$到另外两点 $A(a_1\ ,0)$与 $B(a_2,0)$的距离的和。

如图 2-4(a)所示，设序列$\{a_n\}$的中位数为a_k和a_{k+1}，$x\in[a_k\ ,\ a_{k+1}]$时，函数取值为$|\boldsymbol{AB}|$；而 x 在该区间之外时，如图 2-4(b)所示，该取值大于$|\boldsymbol{AB}|$，因此 $n=2$ 该定理是成立的。



(a) (b)

图 2-4 n = 2 时的函数几何意义

引理：如果 $x\in[a_1\ ,\ a_j]$，那么$|x-a_1|+|x-a_j|$为定值a_j-a_i.

$x\in[a_1\ ,\ a_j]$时，有$|x-a_1|+|x-a_j|=x-a_i+a_j-x=a_j-a_i$.引理得证。

下面利用该引理对普适情况进行证明。设序列$\{a_n\}$对应数轴上的点集 **Sn**，$A_i(a_i\ ,0)\in\mathbf{S_n}$.

n 为大于 1 的奇数时，考虑动点 P(*x* , 0)。$x\in(-\infty, a_i]$时，如图 2-5-1 所示，逐渐让 *x* 增大，*f(x)*的值是不断减小的。此时 $f(x) = nd_1 + f(a_1)$.函数值只取决于 P 到第一个点的距离。因此该距离越小，其值越小。因此在该情况下，取 $x = a_1$. 接下来继续让 P 增大并保持 P 在点 A_1 和 A_2 之间。

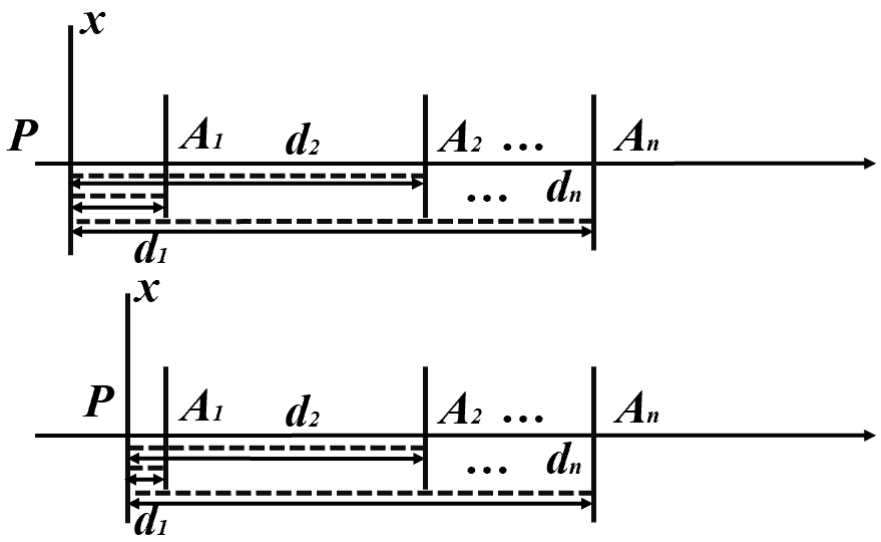


图 2-5-1 奇数情况 1

接下来继续让 *x* 增大并保持 P 在点 A_1 和 A_2 之间。由引理知此时函数的前两项和为一个定值。在 *x* 增大的时候，会产生与图 2-5-1 相同的情形：即随着 *x* 增大且保持 P 在点 A_1 和 A_2 之间时，函数值会逐渐变小。如图 2-5-2 所示。

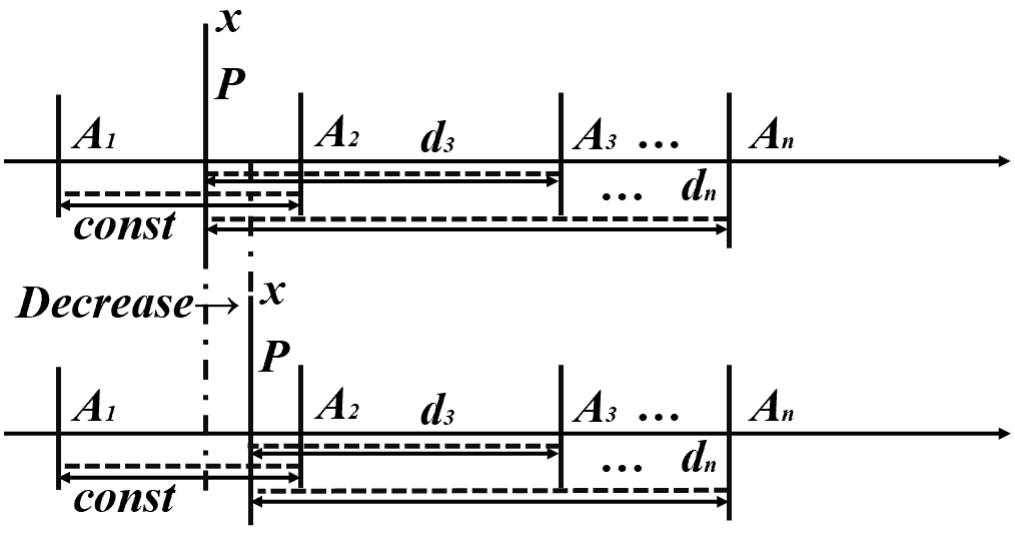


图 2-5-2 奇数情况 2

因此 P 点在 A_2 时该情况取最小值。继续持续这样的操作，设让 *x* 增大并保持 P 在点 A_{i-1} 和 A_i 之间，且 A_i 在 A_k 左侧，使得 A_{2i} 也在点集内部。这么做使得函数的前 2(i-1)项和均是一个常数（因为可以看成 P 在最左端点和最右端点

间，也在次左端点和次右端点间，第三左端点和第三右端点间…这样的点有 i 组并且没有交集，根据引理可知这是一个定值），且每次 P 移动到一个新的区间，都呈现函数值下降的情况。因此这样的操作可以一直持续到 P 在点 A_{k-2} 和 A_{k-1} 的时候。

继续右移 P，使得 P 在 A_{k-1} 和 A_k 之间。这时函数前 2 (k - 1)项的和为一个常数，由于 *n* 为一奇数，因此 2(k-1) + 1 = *n*，即函数只剩下最后一项为变量。则 P 在 A_k 时最后一项取最小值。

而继续右移 P 的话，这时候给出的前提条件“函数的前 2(i-1)项和均是一个常数”已经不成立了，因此无法再找到第 2i – 2 个点。实际上，这时可以将坐标系进行镜像反转，则此时进行右移 P 的操作其实是等价于对 P 进行满足前提条件的情况下进行左移，值反而会不断增大。**因此 *n* 取奇数的情况下，*x* 取中位数时，函数取最小值**。

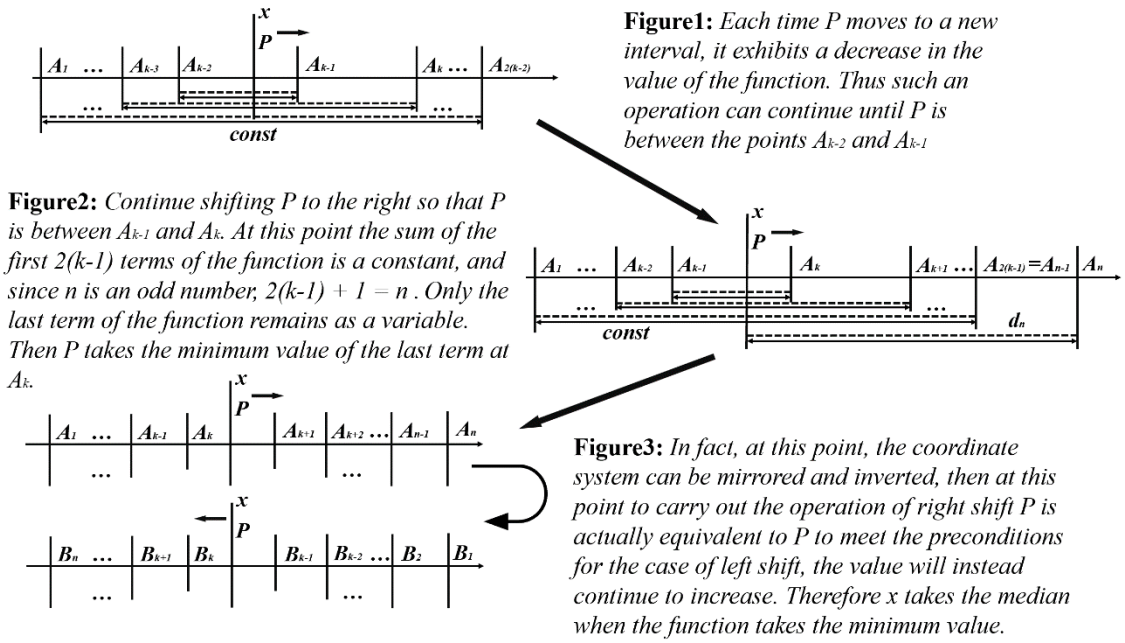


图 2-5-3 奇数情况 3

n 为偶数时，采取的证明方法类似，但在 P 移动到 A_{k-1} 和 A_k 间的情况之后略微不同。这时序列有两个中位数，因此和奇数情况不同的地方在于 P 在 A_k 和 A_{k+1} 间时，基于引理，依旧满足前提条件“函数的前 2(i-1)项和均是一个常数”，且这种情况是前提条件的临界情况，再移动 P 则就会出现“镜像反转”的情况，得到的值更大。因此 P 在该区间内部时，函数恒取最小值。

至此命题得证。

3. POJ.1088 Snowboarding

Michael 喜欢滑雪，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底滑坡。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。

现在需要求出最长路径的长度。

3.1 题目分析

基于题干信息和输入输出信息，可以得到如下条件：

（1） 输入信息为一个行为 R、列为 C 的矩阵 M，代表地图。矩阵的每一个元素 m_{ij} 代表该点的雪山高度大小（也可以理解为坡度）。对地图进行抽象为矩阵的过程如图 3-1 所示。

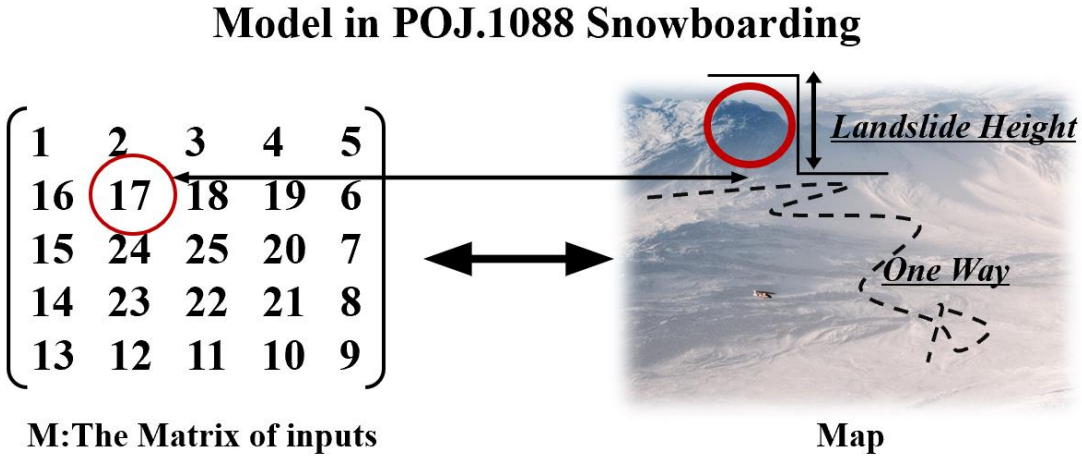


图 3-1 POJ.1088 地图-矩阵抽象概念图

（2） 每个位置的步数都与周围的步数有关。

（3） 每一次运动都只能往上下左右四个方向。且不可以触碰边界。

最长路径的长度，意味着需要在矩阵内找到一个连续的（即上下左右方向连续）的下降序列（也可以是上升序列）。直接在矩阵中进行一个个的枚举显然是极其耗费时间的，可以采取动态规划的思想。

3.2 算法设计

本题采用了动态规划的思想。

使用动态规划的前提条件是满足无后效性和最优子结构性。本题中先走哪一个位置对之后的前进选择没有任何影响，因此无后效性。

对于最优子结构性：不妨设 D[x][y]是矩阵上位置为(x, y)处所拥有的最长下降路径的长度，它对应一条最长路径为 P[x][y]；再设 D[x’][y’]是最长路径 P[x][y]上减去坐标(x, y)部分的路径长度。容易知道

$$D[x'][y'] = D[x][y] - 1$$

$$P[x'][y'] = P[x][y] - (x , y)$$

也即

$$D[x'][y'] + 1 = D[x][y]$$

$$P[x'][y'] + (x , y) = P[x][y]$$

问题的结构如图 3-2 所示。

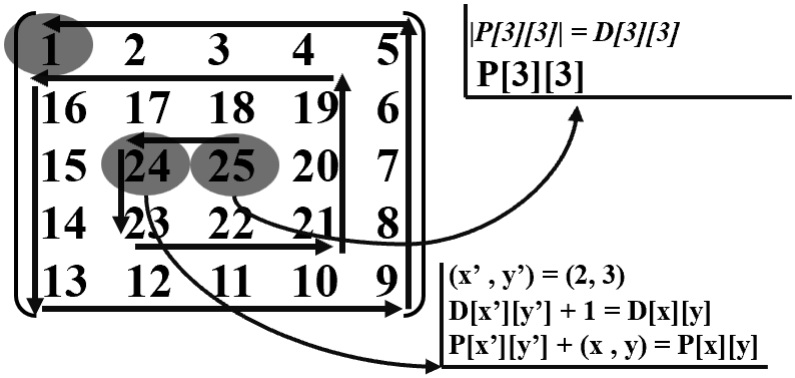


图 3-2 D/P[x , y]与 D/P[x’][y’]

即如果要求出(x , y)处的最长下降路径长度，则可以将其转化为子问题(x’ , y’)处的最长下降路径长度。下面证明最优子结构性成立：

证明：**采用“剪切—粘贴”证明法。**

假设 D[x][y] 是矩阵上位置为(x, y)处所拥有的最长下降路径的长度，它对应一条最长路径为 P[x][y]。有 D[x'][y'] = D[x][y] – 1，P[x'][y'] = P[x][y] – (x , y)。如果对于 P[x'][y']，存在一条更长的路径 P' [x'][y']，其长度为 D' [x'][y']，将这条路径“剪切”到原本的路径里，即有 D'[x'][y'] + 1 > D[x'][y'] + 1 = D[x][y] ，那么 D[x][y] 就不再是矩阵上位置为(x, y)处所拥有的最长下降路径的长度，如图 3-3 所示。这与假设矛盾，因此最优子结构性是满足的。

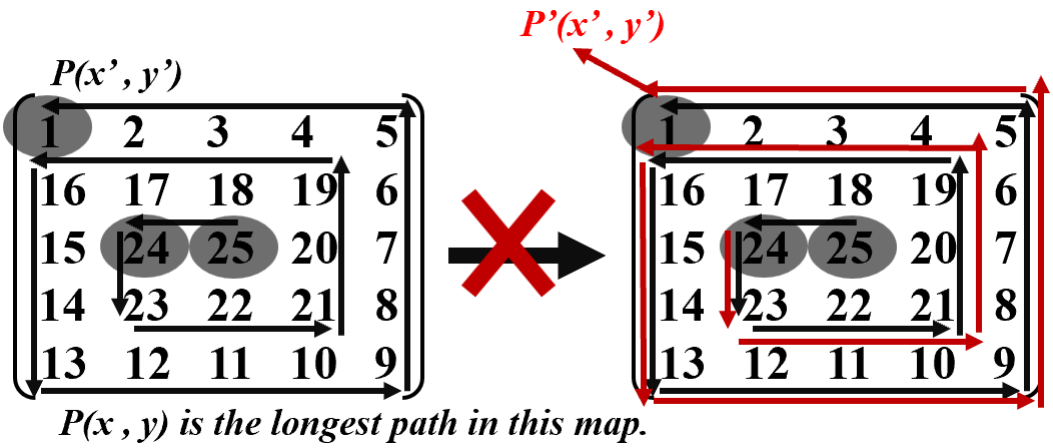


图 3-3 最优子结构性的证明

综合上述，可以采用动态规划对问题进行求解。

首先进行边界条件的判断。这里的判断相对容易：对于某个坐标点(x , y)，如果这个点周围（即上下左右四个方向）都是边界或者其高度值 m_{xy} 小于其周围非边界的所有点，那么这个点的最长下降路径长度是 1，因为整个路径只有它本身。

对应其他坐标点，即周围存在高度值小于该点高度的一些点，那么这个坐标点(x , y)的最长下降路径 P[x][y]取决于这些周围高度较低的点。由于 P 是最长的一条路径，考虑之前的最优子结构性质，P 最长取决于下一个最长的路径。设点集合 S_{ij} 是坐标点(x , y)周围的那些高度小于该点的点，则有|P[x][y]| = D[x][y] = $\max_{(x_k,y_k)\in S_{ij}}\{D[x_k][y_k]+1\}$，这里的“+1”是因为(x , y)本身也算了一个路径的长度。综上所述，可以得出状态转移方程：

$$f(x)=\begin{cases}1 & \text{boundary and no another way}\\ \max\limits_{(x_k,y_k)\in S_{ij}}\{D[x_k][y_k]+1\} & \text{else}\end{cases}$$

算法的相关变量定义见 Table2.1。

Table 2.1: Related definitions of variables in POJ.1088

Name	Functionality	Space Complexity
M	Save the height of each coordinate of the map	$O(n^2)$
D	Save the length of the longest path of each coordinate of the map	$O(n^2)$
R	Rows of the matrix M	O(1)
C	Cols of the matrix M	O(1)
Max	The length of the longest path	O(1)

在基于问题分析和算法设计之后，对函数进行了相关设计。获取坐标(i , j)的函数 GetLength(R,C,M,D,i,j)流程如下：

Algorithm 2.1: GetLength(R,C,M,D,i,j)

```
Input: R , C, M[1…R][1…C] , D[1…R][1…C] , i , j
Output: Max
1    if (i , j) has no way to go ahead then
2        D[i][j] = 1
3        return 1
4    else
5        Max= 0
6        for the direction↑,↓,←,→
7            Let (i' , j') is the next direction
8            D[i'][j'] = GetLength(R,C,M,D, i', j')
9            if Max < D[i'][j']+ 1 then
10               Max = D[i'][j']+ 1
11               D[i][j] = Max
12           end if
13       end for
14   end if
15   return Max
end GetLength
```

主程序对函数进行调用并对结果进行打印的流程如下：

Algorithm 2.2: The Process of POJ.1088

```
Input: R , C, M[1…R][1…C] , D[1…R][1…C]
Output: NIL
1    Get R,C,M,D
2    for every dij in D
3        dij = -1
4    Let max = 0
5    for i = 1 to R
```

```
6         for j = 1 to C
7             if max < GetLength(R,C,M,D, i, j) then
8                 max = GetLength(R,C,M,D, i, j)
9             end if
10        end for
11    end for
12    print max
end The Process of POJ.1088
```

核心算法 GetLength 的流程图如图 3-4 所示。

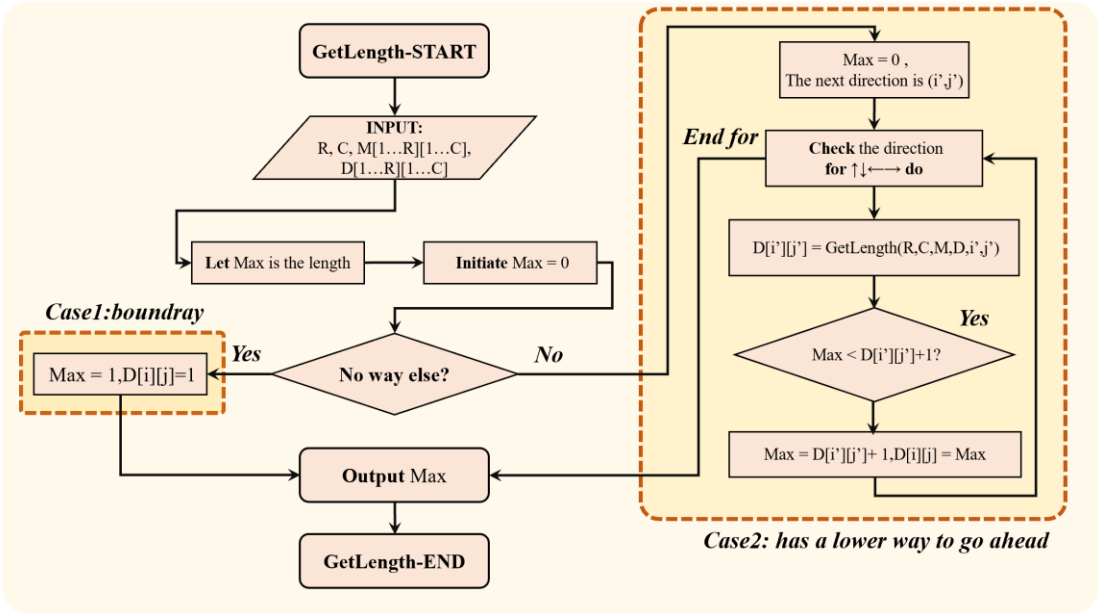


图 3-4 GetLength 算法流程图

3. 3 性能分析

据 Table2.1 所示，存储矩阵需要 $O(n^2)$的空间复杂度。算法的 7-11 行进行了递归的操作。考虑最坏情况，每一次操作都枚举四个方向，每一次递归也同样这么做，那么需要 $O(4n^2) = O(n^2)$的时间复杂度。综合评估如图 3-5 所示。

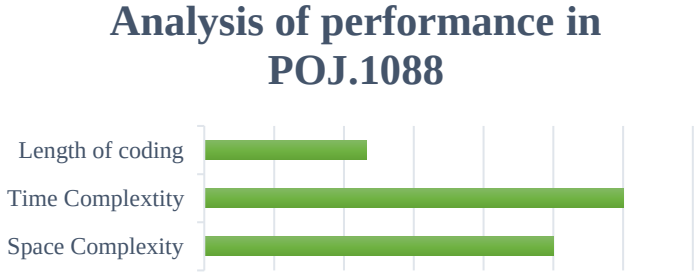


图 3-5 POJ.1088 综合评估

以上是该题目完成时的情况，事实上，可以根据动态规划的特性来造表对函数的时间复杂度进行简化。可在算法伪代码 **Algorithm 2.1** 的第 8 行修改为如 **Algorithm 2.3** 所示的代码:

Algorithm 2.3: Add in Line8 in GetLength

```
1   if D[i'][j'] == -1 then
2       D[i'][j'] = GetLength(R,C,M,D, i', j')
3   end if
end Add in Line8 in GetLength
```

这么做的好处是在每次枚举的时候检索这个方向的最长距离是否存在，这么一来就不再重新调用函数，将一部分递归操作的时间复杂度改成了 O(1)，达到了提高效率的目的。

3.4 运行测试

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。

样本输入与输出以及实际结果如 Table2.2 所示。

Table 2.2: Operational test in POJ.1088

Sample Input	Sample Output	Output
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	25	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 25 请按任意键继续. . .

4. POJ.1328 Radar Installation

假设海岸线是一条无限长的直线。陆地在海岸线的一边，海洋在另一边。每个小岛都是位于海边的一个点。而位于海岸线上的任何雷达装置都只能覆盖 d 的距离，因此，如果两者之间的距离最多为 d，那么海中的一个小岛就可以被一个半径为 d 的装置覆盖。

我们使用直角坐标系，将海岸线定义为 x 轴。海面在 x 轴上方，陆地在 x 轴下方。给定每个岛屿在海中的位置，并给定雷达装置的覆盖距离，你的任

务是编写一个程序，找出覆盖所有岛屿的最小雷达装置数量。请注意，岛屿的位置由其 x-y 坐标表示。

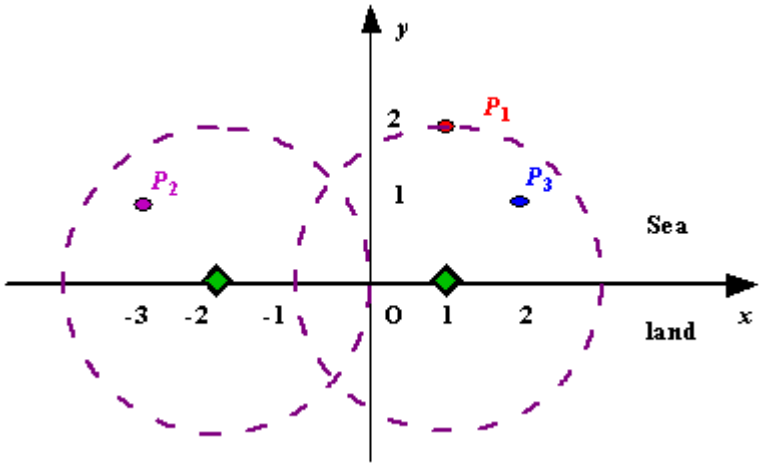


图 4-1 雷达装置输入示例

4.1 题目分析

基于题干信息和输入输出信息，可以得到如下条件：

（1） 题目具体要求：给出 n 个互不相同的点（岛屿）的位置坐标，再给出雷达扫描的最大距离 d，需要求出至少几个雷达的扫描范围可以覆盖整个海平面上存在的所有岛屿。

（2） 雷达只能建在海岸线上。因此其纵坐标为 0.

（3） 雷达的扫描范围是一个半径为 d 的圆。

（4） 存在一些没有解决方案的情况，这时候需要输出-1。因此得知每次进入全新情景后需要对岛屿的位置进行一些判断再正式求解，不符合要求的就不再求解。

综合上述分析，该岛屿-雷达可以抽象为一个覆盖问题，即在 xoy 坐标系空间中给出两个不同的点集 A，B；A 是一些纵坐标 y≥0 的点的集合，B 是一些纵坐标 y = 0 的点的集合，A 和 B 的横坐标 x 满足-∞ ≤ x ≤ +∞；其中 B 集合具有属性 d：每个处于 B 集合里的点可以生成一个半径为 d 的圆。

现给出 A 集合的基数 n，现在要求找出集合 B 的最小基数，使得 B 集合里的点集生成的圆域可以覆盖 A 集合的所有点，如果不存在最小基数则令 B 的基数为-1。

根据问题情景抽象出数学模型的过程如图 4-2 所示。

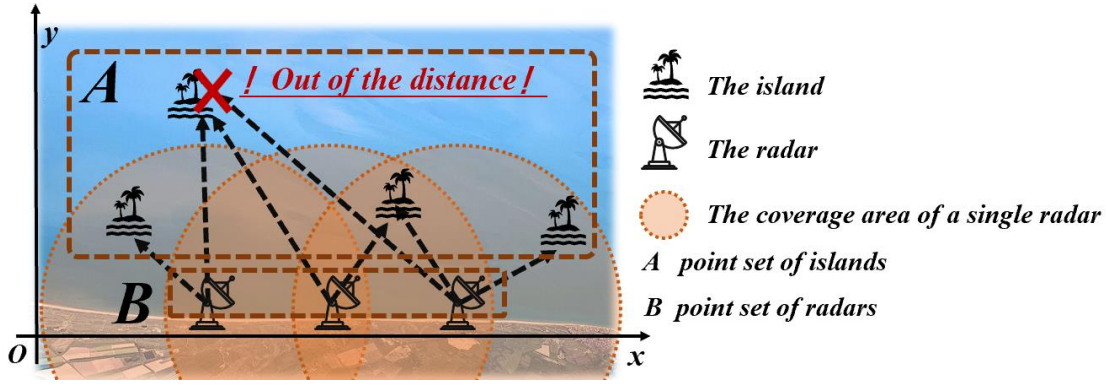


图 4-2 岛屿-雷达问题的抽象过程

4.2 算法设计

考虑到有一些不合理的情况，岛屿距离海岸线过远导致没有一个雷达能够覆盖到这个岛屿，这时候问题是无解的，需要特判。因此需要在开始解决问题之前先对每个岛屿进行一次检索。当岛屿到海岸线的距离 y 大于 d 时，离它最近的雷达也没法探测到它。所以判断的依据是岛屿的 y 轴坐标分量是否满足 y ≤ d。检查完毕后，确认不存在太远的岛屿后，再设法解决问题。

所以，首先对于每个岛 Piont[i]，检查 Piont[i].y（即岛屿 y 轴坐标分量值）与 d 的大小关系，如果发现有一个不满足上述条件，那么直接停止该情景的解决过程，并输出-1 表示问题无解。无解的情况如图 4-3 所示。

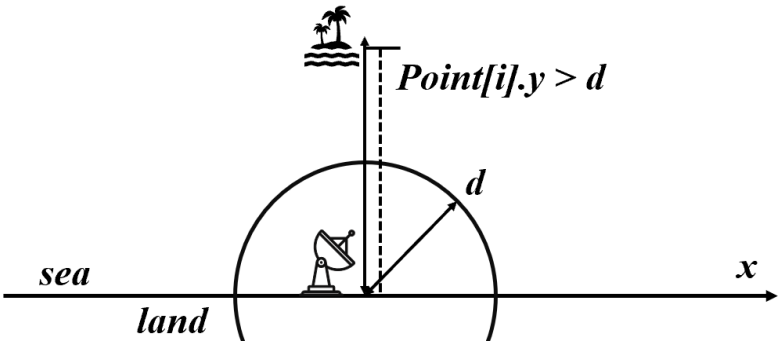


图 4-3 无解情况的判断

当所有岛屿都满足时，进行下一步的求解。

在所有的岛屿 Point[i]满足 Piont[i].y ≤ d 时，以该岛屿 Point[i]为圆心，作出一个半径为 d 的圆 i，那么圆 i 与 x 轴一定有 1 或 2 个交点。只有一个交点，当且仅当 Piont[i].y = d。

作出圆 1…n 之后，设圆 i 与 x 轴的交点的横坐标为 X[i][1]与 X[i][2]，其中 X[i][1]是圆 i 与 x 轴交点的从左往右数的第 1 个点的横坐标值，X[i][2]是圆 i 与 x 轴交点的从左往右数的第 2 个点的横坐标值。如果 Piont[i].y = d，不妨 X[i][1] = X[i][2]即可。

接着，对所有的岛屿 Point[i…n]，根据 X[i][1]的值的大小，对序列 Point 进行升序排序，即满足任意 Point[i]，Point[j]，i < j，有 X[i][1]≤X[j][1]。这一步操作是便于后续任务的执行。

作出圆 i 并对岛屿序列进行排序的操作结果图如图 4-4 所示。

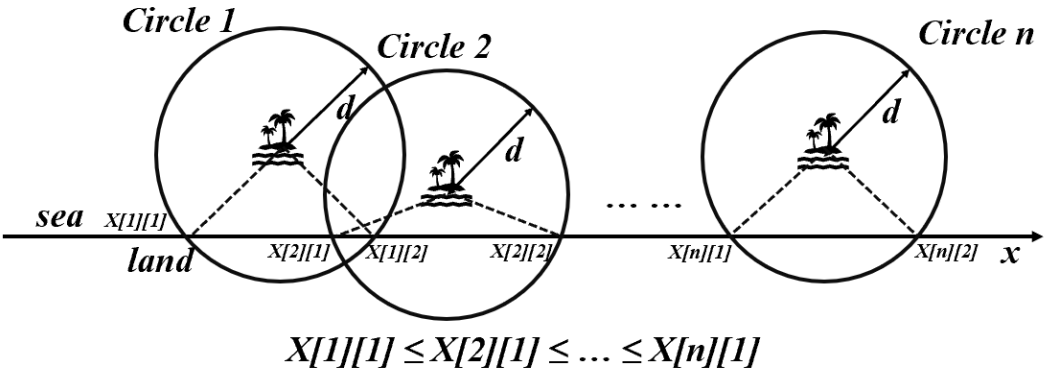


图 4-4 作圆并对岛屿进行升序排序

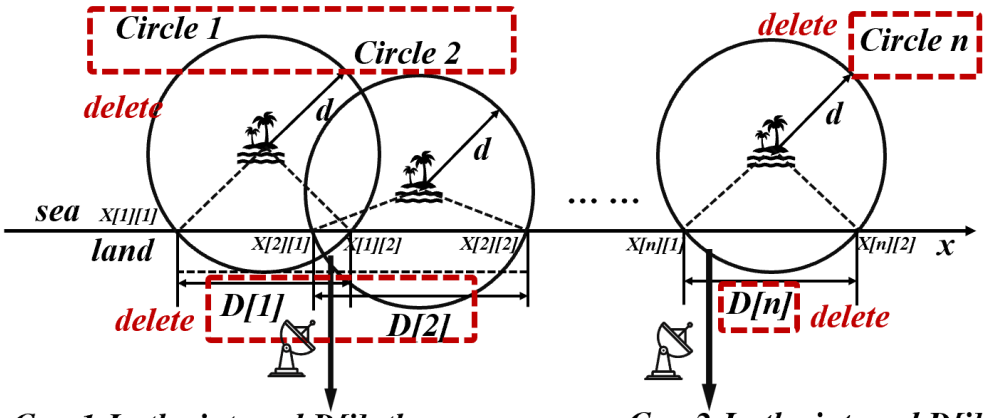
接下来开始加入雷达。设雷达初始数量为 0，对于每个岛屿 Point[i]，考察其生成的圆 i 在 x 轴上制造的闭区间 D[i]:[**X[i][1]，X[i][2]**]。检索从岛屿 1 一直进行到岛屿 n 为止，检索结束时的雷达数目为最终雷达数目。在检索的过程中将会面临如下情况：

- 情况 1：在区间 D[i]中，存在 N[i]个其他圆生成的区间与该区间 D[i]有重合部分。
- 情况 2： 在区间 D[i]中，没有与其他区间重合的部分。

情况 1 和 2 覆盖了整个问题所有的情况。对这两种情况采取不同的策略。

- 对应情况 1：雷达数目+1，在 Point 序列删除该岛屿，以及其他所有生成区间与 D[i]有重合的 N[i]个岛屿，再删除这些岛屿生成的圆以及该圆生成的区间。删除完成后考察下一个没有被删除的编号最小的岛屿。
- 对应情况 2：雷达数目+1，在 Point 序列删除该岛屿，并且删除这个岛屿生成的圆以及该圆生成的区间。删除完成后考察下一个没有被删除的编号最小的岛屿。

这其实是一种贪心策略。因为每一次面对情况，雷达数目都会增加 1，因此希望雷达的数目最少，即尽可能在每一次操作的时候删掉尽可能多的岛屿。这样使得操作次数最少。对于该算法运行的图解如图 4-5 所示。



Case1:In the interval D[i], there exist N[i] other circle-generated intervals that have overlapping parts with this interval D[i].

Radar number +1, delete that island in the Point sequence, as well as all other N[i] islands whose generated intervals have overlap with D[i], and then delete the circle generated by those islands as well as the interval in which that circle was generated. After the deletion is complete examine the next smallest numbered island that has not been deleted.

Case2:In the interval D[i], there is no part of the interval that overlaps with other intervals.

Radar number +1, deletes the island in the Point sequence and deletes the circle generated by this island as well as the interval in which the circle was generated. After the deletion is complete examine the next smallest numbered island that has not been deleted.

图 4-5 贪心算法运行图解

算法的相关变量定义见 Table3.1。

Table 3.1: Related definitions of variables in POJ.1328

Name	Functionality	Space Complexity
n	The number of islands	O(1)
d	Coverage distance of radars	O(1)
Point[1…n]	Storing coordinates of islands	O(n)
X[1…n][1…2]	Save the value of the horizontal coordinate of the intersection of circle i with the x-axis	O(2n)=O(n)
D[1…n]	Intervals of circle	O(n)
num	The minimum nuber of radars	O(1)

为便于算法操作，定义函数 Add 与 Pop 来辅助主算法。

函数 Add 的参数为集合 S 与岛屿 island。功能是将 island 加入集合 S。

Add 的算法流程如下：

Algorithm 3.1: Add(S, island)

```
Input: S, island
Output: NIL
1    S = S∪{island}
end Add(S, island)
```

函数 Pop 的参数为集合 S 与一个数字 i。功能是当 i = 0 时，删除集合里下标最小的岛屿并返回该岛屿；i 为其他值时返回下标为 i 的岛屿。如果 i 大于所有岛屿的下标，或者 S 是空集，返回错误。

Pop 的算法流程如下：

Algorithm 3.2: Pop(S, i)

```
Input: S, i
Output: island
1    if S is ∅ or i > the max subscript in S then
2        return ERROR
3    else
4        if i = 0 then
5            Let island is the one having minimum subscript in S
6            Delete island in S
7            return island
8        else
9            Let island is the one whose subscript is i in S
10           Delete island in S
11           return island
12       end if
13   end if
end Pop(S, i)
```

该贪心算法的完整流程如下：

Algorithm 3.3: The Process of POJ.1088

```
Input: n, d, Point[1…n], X[1…n][1…2], D[1…n]
Output: num
1    num = 0
2    Get n, d
3    Let S is ∅
4    for i = 1 to n
5        Get Point[i].x, Point[i].y
6        Add(S, Point[i])
```

```
7       if Point[i].y > d then
8           return ERROR
9       end if
10      Make a circle with Point[i] as center and d as radius
11      Let X[i][1] is the first intersection of the circle with horizontal coordinate axis
12      Let X[i][2] is the second intersection of the circle with horizontal coordinate axis
13  end for
14  Sort X[1…n][1…2] and Point[1…n] according to X[1…n][1] in Ascending order
15  while !S is empty do
16      Let P = Pop(S, 0)
17      Let t = the subscripts of P in Point[1…n]
18      num++
19      for i = t+1 to n
20          if X[i][1] is in D[t] or X[i][2] is in D[t] then
21              Pop(S, i)
22          end if
23      end for
24  end while
25  return num
end The Process of POJ.1088
```

贪心算法的流程图如图 4-6 所示。

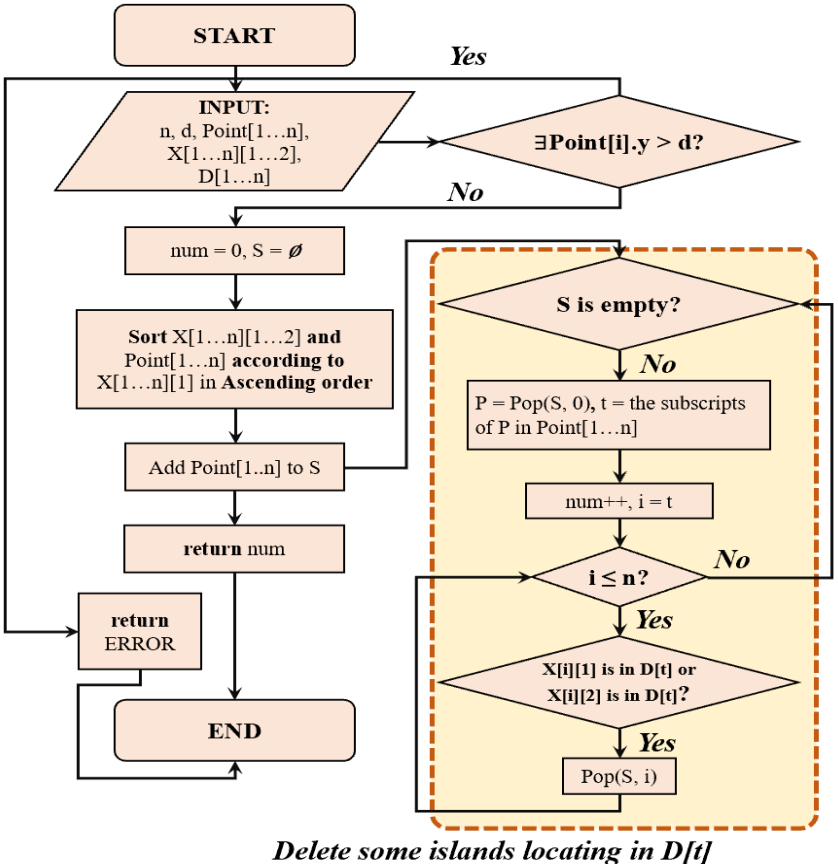


图 4-6 POJ.1328-贪心算法流程图

4.3 性能分析

辅助函数 Add 的时间复杂度为 O(1)，函数 Pop 主要起查找功能，时间复杂度依据选择的数据结构而变化。如果考虑数组类型的数据结构，并且对下标也进行存储，那么该函数的时间复杂度为 O(1).

算法主体的读取数据需要 O(n)的时间复杂度；计算每个圆生成的区间需要 O(n)的时间复杂度；算法 14 行的排序需要 O(nlogn)的时间复杂度。

算法核心操作是 15~24 行，考虑最坏的情况，即每次都不存在区间重合的岛屿。这样一来，while 循环将会进行 n 次，附带内部 for 循环，并且每次调用函数 Pop 都会有 O(1)的时间复杂度的开销，而每次 while 循环和 for 循环都会执行一次 Pop。那么 15~24 行将会进行 $n + (n) + (n - 1) + \ldots + 2 + 1 = \frac{n(n+3)}{2}$次运算，因此算法该部分的时间复杂度为 $O(n^2)$。

综上所述，总的时间复杂度为 $O(n^2)$，空间复杂度为 O(n)

综合评估如图 4-7 所示。

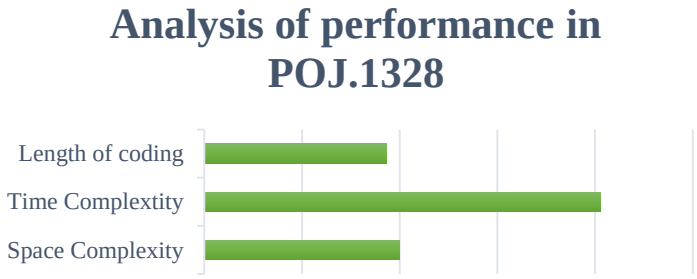


图 4-7 POJ.1328 综合评估

4.4 运行测试

输入：输入由多个测试案例组成。每个案例的第一行包含两个整数 n（1<=n<=1000）和 d，其中 n 是海中岛屿的数量，d 是雷达装置的覆盖距离。随后是 n 行，每行包含两个整数，分别代表每个岛屿的位置坐标。然后用一行空行来分隔情况。

输入以一行包含一对 0 的行结束。

输出：对每个测试用例输出一行，包括测试用例编号和所需安装雷达的最小数量。”-1 “表示该案例没有解决方案。

样本输入与输出以及实际结果如 Table3.2 所示。

Table 3.2: Operational test in POJ.1328

Sample Input	Sample Output	Output
0 2 1 2 -3 1 2 1 1 2 0 2 0 0	Case 1: 2 Case 2: 1	3 2 1 2 -3 1 2 1 Case 1: 2 1 2 0 2 Case 2: 1 0 0 请按任意键继续. . .

5. POJ.1860 Currency Exchange

我们所在的城市有几个货币兑换点。假设每个兑换点都专门兑换两种货币，并且只对这两种货币进行兑换。同一对货币可以有多个兑换点。每个兑换点都有自己的汇率，A 兑换 B 的汇率就是用 1A 换取 B 的数量。此外，每个兑换点都有一定的佣金，即您在进行兑换操作时必须支付的金额。佣金总是以来源货币收取。

例如，如果您想在汇率为 29.75 的兑换点将 100 美元兑换成俄罗斯卢布，而佣金为 0.39，您将得到 (100 - 0.39) * 29.75 = 2963.3975 卢布。

您肯定知道，在我们的城市里有 N 种不同的货币可以交易。让我们为每种货币分配从 1 到 N 的唯一整数。这样，每个兑换点就可以用 6 个数字来描述：整数 A 和 B——兑换的货币数量，实数 R_{AB}、C_{AB}、R_{BA} 和 C_{BA}——分别是 A 兑换 B 和 B 兑换 A 时的汇率和手续费。

尼克有一些货币 S 的钱，他想知道在进行一些兑换操作后，能否以某种方式增加他的资本。当然，他最终还是想把钱存成 S 货币。请帮助他回答这个难题。尼克在进行操作时必须始终拥有非负数的资金。

5.1 问题分析

基于题干信息和输入输出信息，可以得到如下条件：

（1） 尼克希望对资金进行扩增。

（2） S 是尼克的起始货币。

（3） 尼克可以对货币进行交换，由于汇率之间的关系，在某些情况下，尼克所有的货币数目可以增值，但他每一次交换货币都需要付一定手续费。

（4） 通过一系列的交换，回到尼克这边的资金应该大于初始资金并且类型为 S。

问题可以建模为一个图结构。

N 种货币可以视为 N 个不同的结点（结点集为 P），尼克拥有的货币 S 是 N 个结点里的其中一个；而 M 个兑换点可以视为 **2M** 个不同的有向边（边集为 E）。这里边数为 2M 的原因是：如果 A 可以换 B，那么 B 也可以换 A，因此一个兑换点对应两个边。这里，有向边<A, B>内蕴含的信息为 R_{AB}、C_{AB}，分别代表 A 兑换 B 时的汇率和手续费。这里每个边储存的信息不能直接表示边权值。对图 G(P, E)的建模过程如图 5-1 所示。

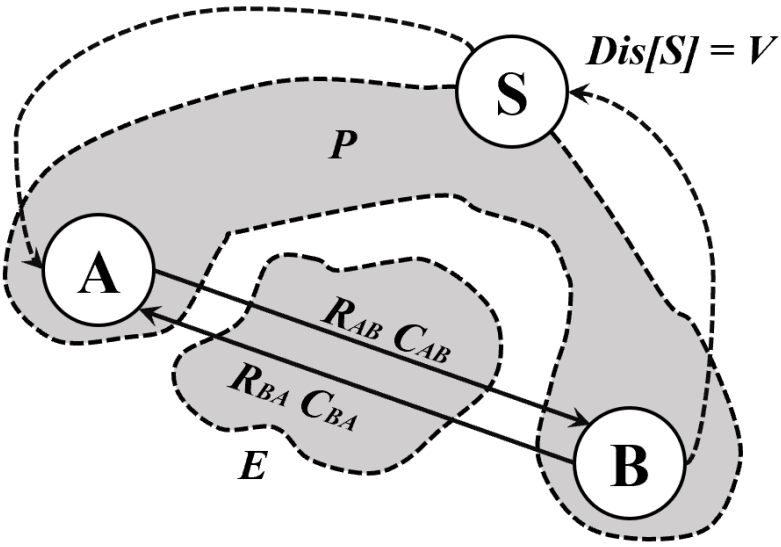


图 5-1 图建模过程示意图

5.2 算法设计

考虑初始资金为 V，设初始结点 S 的资金 Dis[S] = V，则现在需要找到一个回路，使得资金在这条回路上流通之后回到原点，最后的资金数额要大于 V。因此这里实际上需要找到一个图中的“正权值回路”。对于回路的判定，可以采用 Bellman-Ford 算法。

由于在本题图建模中不显含权值函数，并且这里并不是需要求出最短路径相关。因此需要对路径的松弛操作进行一些调整。

问题开始时，置其他结点 p 的 Dis[p] = 0，置 S 结点（出发点）的 Dis[S] = V。每一次“松弛操作”时，进行的工作为：

对每个边 e，首先计算(Dis[e.A] – e.C_{AB})×R_{AB} 的值，这是进行兑换操作之后获得的价值。如果这个价值要大于 Dis[e.B]本身的价值，那么对这个边进行“松弛”（实际上这里是让 Dis[e.B]变成比较大的那个值，因此这里的松弛不如说是“紧凑”），即让 Dis[e.B] = (Dis[e.A] – e.C_{AB})×R_{AB}.

完成 Bellman-Ford 算法需要的 N 次松弛操作之后，再进行一次操作。原版的算法是判定负权回路的存在。这里如果在第 N+1 次操作后发现仍然有被松弛的边，那么代表图中存在“正权回路”，这时输出 Yes，否则输出 No。

对问题的图解和“松弛操作”如图 5-2 所示。

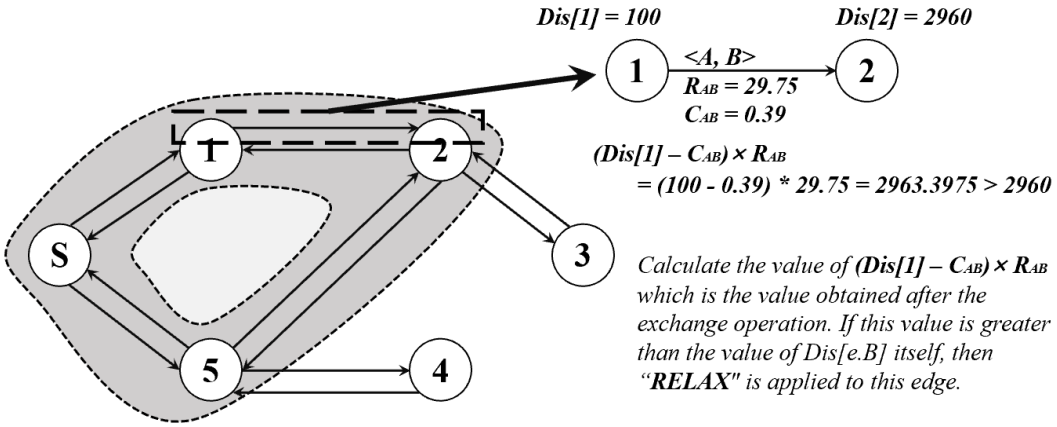


图 5-2 “松弛”操作示意图

算法的相关变量定义见 Table4.1。

Table 4.1: Related definitions of variables in POJ.1860

Name	Functionality	Space Complexity
N	The kind of money	O(1)
M	The number of exchange points	O(1)
RAB	Exchange rate from A to B	O(1)
RBA	Exchange rate from B to A	O(1)
CAB	Commission from A to B	O(1)
CBA	Commission from B to A	O(1)
flag	Check the answer	O(1)
Dis[1…N]	Preservation of value	O(n)

对 Bellman-Ford 算法进行修改后的伪代码算法流程如下：

Algorithm 4.1: Bellman-Ford-revised version(G, S)

Input: G, S

```
Output: “Yes” or “No”
1    Dis[S] = V
2    for all p in P except S
3        Dis[p] = 0
4    end for
5    for i = 1 to N
6        for every p in G
7            Let distance = (Dis[e.A] – e.CAB)×RAB
8            if distance > Dis[e.B] then
9                Dis[e.B] = distance
10           end if
11       end for
12   end for
13   flag = 0
14   for every p in G
15   Let distance = (Dis[e.A] – e.CAB)×RAB
16       if distance > Dis[e.B] then
17           flag = 1
18       end if
19   end for
20   if flag = 1 then
21       print”Yes”
22   else
23       print”No”
24   end if
end Bellman-Ford-revised version(G, S)
```

5.3 性能分析

据 Table4.1 所示，空间复杂度为 O(N+M)

算法采用了 Bellman-Ford 的思想，因此时间复杂度为 O(NM)。

综合评估结果如图 5-3 所示。

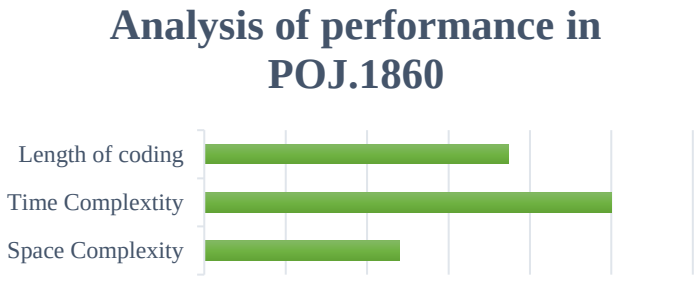


图 5-3 POJ.1860 综合评估

5.4 运行测试

输入：输入的第一行包含四个数字： N - 货币数量，M - 兑换点数量，S - 尼克拥有的货币数量，V - 他拥有的货币单位数量。接下来的 M 行按上述顺序各包含 6 个数字，即相应兑换点的说明。数字之间用一个或多个空格隔开。1<=S<=N<=100, 1<=M<=100, V 为实数，0<=V<=1000。

每个点的汇率和佣金都是实数，小数点后最多两位数，0.01<=汇率<=100，0<=佣金<=100。

如果某个交换操作序列中没有一个交换点被使用超过一次，我们就称该序列为简单序列。你可以假定，在任何简单的交换运算序列中，末尾和开头的和的数值之比都小于 10000。

输出：如果尼克能增加财富，则输出 "是"，否则输出 "否 "。

样本输入与输出以及实际结果如 Table4.2 所示。

Table 4.2: Operational test in POJ.1860

Sample Input	Sample Output	Output
3 2 1 20.0 **1 2 1.00 1.00 1.00 1.00** **2 3 1.10 1.00 1.10 1.00**	**YES**	3 2 1 20.0 1 2 1.00 1.00 1.00 1.00 2 3 1.10 1.00 1.10 1.00 YES

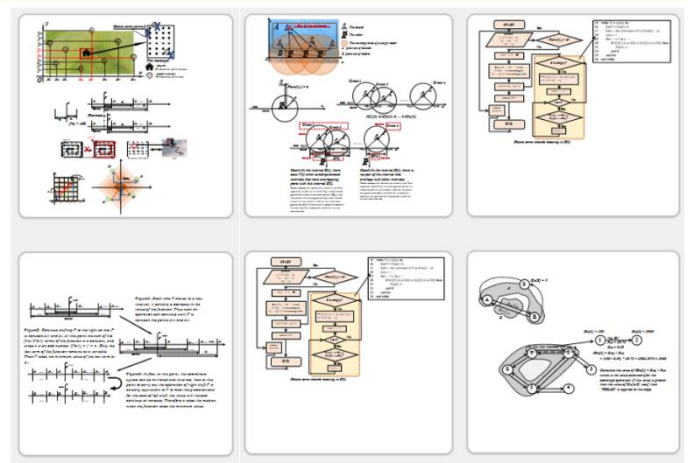
6. 总结

6.1 实验总结

本次实验一共完成了 24 道题目，囊括了分治、动态规划、贪心算法、最短路径等在算法理论课上讲授过的内容。有一些题目并不好做，花了很长时间才想出来，也有一些题目借鉴了网络上的思路。这些题目是对于理论课的知识进行了非常好的运用和实践。

在撰写实验报告的时候，也有意识地选择了以上四块不同领域的算法题目进行分析。对于每一个题目，采用问题分析+建模+设计算法+伪代码+流程图的描述。这样写起来使得整个问题清晰了不少。同时在撰写报告的过程中也是对算法精髓部分的很好的回顾复习。

6.2 心得体会和建议



分治、动态规划、贪心算法等思想是处理问题的几大经典算法。笔者在思考的过程中把这些算法充分地拓展应用，大大提高了编程能力。

对于该报告，采用的撰写手段是插入了大量的图片和表格。为了锻炼自身的英文文献阅读和写作能力。同时笔者极其重视排版（当然在这里可能有一些班门弄斧的嫌疑），花了大量的时间消磨在对页面的布局上。目的同样也是希望锻炼能力。

这些图片和表格以及伪代码等部分的设计都是第一次做，技术上有很多不成熟的地方，采用的制作方法也很野蛮粗暴，就是直接一个字一个字打出来的。确实耗费了大量的时间，导致最后在 deadline 时间才做完，写到最后，赶工的痕迹也很明显…总共写作时间持续了快一个月…

下面给出些建议：相信老师也知道 poj 平台过于老旧，其 c++的标准居然是 98 年的。而且用不了万能头文件库，这让很多同学都被“初见杀”。可以使用洛谷，力扣等刷题平台，上面也有 poj 的题目。另外，也可以出一些算法导论题目的变式。比如找零问题或活动教师安排问题，让大家在电脑上实现一些代码，这样想必会让教学更顺利。

感谢您的阅读！再见！