

计算机算法设计与分析

动态规划

易凯

2017 年 4 月 18 日

班 级 软件 53 班

学 号 2151601053

邮 箱 williamyi96@gmail.com

联系电话 13772103675

个人网站 <https://williamyi96.github.io>
williamyi.tech

实验日期 2017 年 4 月 18 日

提交日期 2017 年 6 月 6 日

目录	2
----	---

目录

1 基本声明	5
2 DP 问题的基本方法	5
2.1 DP 概念	5
2.2 动态规划算法的实现步骤	5
3 最长单调递增子序列	5
3.1 题目描述	5
3.2 试题分析	5
3.3 算法实现	6
4 编辑距离问题	8
4.1 问题描述	8
4.2 数据输入与结果输出	9
4.3 算法分析与设计	9
4.4 代码实现	9
4.5 运行结果	10
5 数字三角形问题	10
5.1 问题描述	10
5.2 数据输入与结果输出	10
5.3 算法分析与设计	11
5.4 代码实现	11
5.5 反思总结	11
6 汽车加油行驶问题	12
6.1 问题描述	12
6.2 算法设计	12
6.3 输入与输出	12
6.4 算法分析	12
6.5 程序实现	12
6.6 结果示例	14

目录	3
7 最少费用购物问题	14
7.1 习题描述	14
7.2 算法设计	15
7.3 数据输入	15
7.4 数据输出	16
7.5 分析	16
7.6 代码实现	16
7.7 样例测试	18
8 收集样本问题	18
8.1 习题描述	18
8.2 算法设计	19
8.3 数据输入	19
8.4 数据输出	19
8.5 代码实现	19
8.6 样例测试	21
9 参考资料	21

插图

1	编辑距离问题运行示例	10
2	汽车加油行驶问题示例运行	15
3	最少购物费用问题示例运行	19
4	机器人收集样本问题示例运行	21

1 基本声明

此动态规划的相关作业，是在完成《计算机算法设计与分析》的布置习题的基础上，对于《算法导论》对应章节的课后题的程序实现。

2 DP 问题的基本方法

2.1 DP 概念

动态规划方法通常用于求解最优化问题。这类问题往往有很多个解，但是我们需要找到这些解中的最优解（最优解就是最大值或者最小值的解）。

动态规划方法与分治法最大的差别是，分治法的子问题之间相互没有包含关系，而动态规划方法的子问题则存在着包含关系，分治法的求解会反复地求解公共子问题，而动态规划方法采用制表待查的方式使公共子问题只需要计算一次。从而将许多需要指数级时间才能够计算完成的算法只需要平方级别或者立方级别就可以完成。

2.2 动态规划算法的实现步骤

1. 刻画一个最优解的结构特征;
2. 递归地定义最优解的值;
3. 计算最优解的值，通常采用自底向上的方法
4. 利用计算出的信息构造一个最优解

3 最长单调递增子序列

3.1 题目描述

设计一个 $O(n^2)$ 时间的算法，找出由 n 个数组成的序列的最长单调递增子序列。

3.2 试题分析

我们可以使用数组 $b[0:n-1]$ 记录 $a[i]$, $0 \leq i < n$, 为结尾元素的最长递增子序列的长度。序列 a 的最长递增子序列的长度为 $\max b[i]$ 。

首先, $b[i]$ 满足最优子结构, 因为如果 $a[i]$ 不是 $b[j]$ 的最长单调递增子序列, 而 $a[k]$ 是它的最长单调递增子序列, 那么用 $a[k]$ 替换为 $a[i]$ 即可, 则最后 $b[i]$ 为最优子结构。

因此我们可以递归地定义最优解的值: $b[0] = 1, b[i] = \max b[k] + 1$

3.3 算法实现

Java 实现

```
1 package lis;
2
3 import java.util.Scanner;
4
5 public class LIS_DP {
6     //dp[i]记录[0:i]数组的LIS, lis表示LIS的长度
7     static int[] dp = new int[100];
8     static int lis;
9
10    static int LIS_dp(int arr[], int size) {
11        for(int i = 0; i < size; i++) {
12            dp[i] = 1;
13            for(int j = 0; j < i; j++) {
14                if(arr[i] > arr[j] && dp[i] < dp[j] + 1) {
15                    dp[i] = dp[j] + 1;
16                    if(dp[i] > lis) lis = dp[i];
17                }
18            }
19        }
20        return lis;
21    }
22
23    static void outputLIS(int arr[], int index) {
24        boolean isLIS = false;
25        if(index < 0 || lis == 0) return;
26        if(dp[index] == lis) {lis--; isLIS = true;}
27        if(isLIS) System.out.println(arr[index+1]);
28    }
29
30    public static void main(String[] args) {
```

```
31         //int [] arr = new int[100];
32         int arr[] = {1,2,3,5,4,9,8,7};
33         //         如何设置任意输出的字符串向整数的转化
34         //         Scanner sc = new Scanner(System.in);
35         //         System.out.println("Please input the array");
36         //         for(sc.nextInt()) {
37         //             arr = sc.nextInt();
38         //         }
39         System.out.println(LIS_dp(arr, arr.length));
40
41         outputLIS(arr, (arr.length - 1));
42         System.out.println();
43     }
44 }
```

Cpp 实现

```
1 #include <iostream>
2 using namespace std;
3
4 /* 最长递增子序列 LIS
5  * 设数组长度不超过 30
6  * DP
7  */
8
9 int dp[31]; /* dp[i]记录到 [0, i] 数组的 LIS */
10 int lis;    /* LIS 长度 */
11
12 int LIS(int * arr, int size){
13     for(int i = 0; i < size; ++i){
14         dp[i] = 1;
15         for(int j = 0; j < i; ++j){
16             if(arr[i] > arr[j] && dp[i] < dp[j] + 1){
17                 dp[i] = dp[j] + 1;
18                 if(dp[i] > lis) lis = dp[i];
19             }
20         }
21     }
22     return lis;
```

```
23 }
24
25 /* 输出 LIS */
26 void outputLIS(int * arr, int index) {
27     bool isLIS = 0;
28     if(index < 0 || lis == 0) return;
29
30     if(dp[index] == lis) {--lis; isLIS = 1;}
31
32     outputLIS(arr,--index);
33
34     if(isLIS) printf("%d ",arr[index+1]);
35 }
36
37 int main() {
38     int arr [] = {1,-1,2,-3,4,-5,6,-7};
39
40     /* 输出 LIS长度; sizeof 计算数组长度 */
41     printf("%d\n",LIS(arr,sizeof(arr)/sizeof(int)));
42
43     /* 输出 LIS */
44     outputLIS(arr,sizeof(arr)/sizeof(int) - 1);
45     printf("\n");
46 }
```

在使用 java 进行代码实现时, 只能够输出 LIS, 但是不能够输出 LIS 具体内容, 遇到这样的问题如何处理? 继续深入研究!

4 编辑距离问题

4.1 问题描述

设 A 和 B 是两个字符串。要用最少的字符操作将字符串 A 转换为字符串 B。这里所说的字符操作包括: 1). 删除一个字符; 2). 插入一个字符; 3) 将一个字符改为另一个字符。

将字符串 A 变换为字符串 B 所用的最少字符操作数称为字符串 A 到 B 的编辑距离, 记为 $d(A,B)$ 。试设计一个有效算法, 对任给的两个字符串 A 和 B, 计算出他们的编辑距离 $d(A,B)$ 。

也就是对于给定的字符串 A 和 B，计算其编辑距离 $d(A,B)$ 。

4.2 数据输入与结果输出

数据输入 文件的第一行是字符串 A，文件的第二行是字符串 B。

结果输出 将编辑距离 $d(A,B)$ 直接输出。

4.3 算法分析与设计

此题为使用动态规划的基础之上，然后利用相关 vector 完成基本的操作。值得注意的是，有效地字符操作数只有三种。

4.4 代码实现

```
1   #include <stdio.h>
2   #include <string.h>
3   using namespace std;
4
5   char s1[1000],s2[1000];
6
7   int min(int a,int b,int c) {
8       int t = a < b ? a : b;
9       return t < c ? t : c;
10  }
11  void editDistance(int len1,int len2) {
12      int** d=new int*[len1+1];
13      for(int k=0;k<=len1;k++)
14          d[k]=new int [len2+1];
15      int i,j;
16      for(i = 0;i <= len1;i++)
17          d[i][0] = i;
18      for(j = 0;j <= len2;j++)
19          d[0][j] = j;
20      for(i = 1;i <= len1;i++)
21          for(j = 1;j <= len2;j++) {
22              int cost = s1[i] == s2[j] ? 0 : 1;
23              int deletion = d[i-1][j] + 1;
```

```
24         int insertion = d[i][j-1] + 1;
25         int substitution = d[i-1][j-1] + cost;
26         d[i][j] = min(deletion, insertion, substitution);
27     }
28     printf("%d\n", d[len1][len2]);
29     for(int k=0; k<=len1; k++)
30         delete[] d[k];
31     delete[] d;
32 }
33 int main() {
34     while(scanf("%s %s", s1, s2) != EOF)
35         editDistance(strlen(s1), strlen(s2));
36 }
```

4.5 运行结果

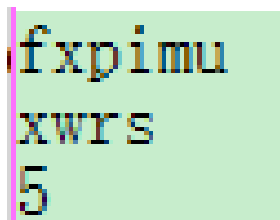


图 1: 编辑距离问题运行示例

5 数字三角形问题

5.1 问题描述

给定一个由 n 行数字组成的数字三角形。试设计一种算法，计算出从三角形的顶到底的一条路径，使该路径经过的数字总和最大。

5.2 数据输入与结果输出

数据输入 文件的第一行是数字三角形的行数，接下来的 n 行是数字三角形各行中的数字。所有数字都在 0-99 之间。

结果输出 将计算结果输出中的第一行就表示所得到的计算出的最大值。

5.3 算法分析与设计

该问题只需要在三角形的每一层中选择最大的数字，然后构成一条路径即可。

5.4 代码实现

```
1 package maxtriangle;
2
3 public class MaxTriangle {
4
5     static int maxTriangle(int mat[][] , int row, int col) {
6         int max = 0, sum = 0;
7         if(row == 0) return sum;
8         for(int k = 0; k < col; k++) {
9             if(mat[row][k] > max) max = mat[row][k];
10            sum += maxTriangle(mat, row-1, col-1);
11        }
12        return sum;
13    }
14
15    public static void main(String[] args) {
16        int [][] mat= {{7},{3,8},{8,1,0},{2,7,4,4},{4,5,2,6,5}};
17        System.out.println(maxTriangle(mat, 5, 5));
18
19    }
20 }
```

5.5 反思总结

相比较而言，此动态规划问题较为简单，思考如果按照上述格式完成有可能不会得到预期结果的根本原因。

6 汽车加油行驶问题

6.1 问题描述

给定一个 $N \times N$ 的网格，设其左上角为起点，坐标为 $(1,1)$ ，X 轴向右为正，Y 轴向下为正，每个方格边长为 1。一辆汽车从起点驶向终点，其坐标为 (N,N) 。如果在若干网格的交叉点处，设置了油库，可供汽车在行驶途中加油。汽车在行驶过程中应该遵守如下规则：

1. 汽车只能沿着网格行驶，装满油后汽车能够行驶 K 条网格边。出发时汽车已经装满油，在起点和终点处不设油库。
2. 当汽车行驶经过一条网格边时，若其 X 坐标或者 Y 坐标减小，则应付费 B ，否则免付费。
3. 汽车在行驶过程中遇油库则应该加满油并支付加油费用 A 。
4. 在需要时可在网格点处增设油库，并付增设油库费用 C (不含加油费用 A)。
5. 其中上述四步的所有数均为正整数。

6.2 算法设计

求汽车从起点出发到终点的一条所付费用最少的行驶路线。

6.3 输入与输出

输入为每个一组 $NKABC$ 正整数，然后接着 N 行的加油点情况 (0 表示没有，1 表示有)。

输出是支付费用最少的路线。

6.4 算法分析

此题为典型的使用动态规划求解的例子，当全局的费用最少时，也就是达到全局最优时，那么局部一定是最优的。

6.5 程序实现

```
1 #include <stdio>
2 #include <queue>
```

```

3 #include <cstring>
4 #include <algorithm>
5 #define N 105
6 #define K 15
7 #define inf 0x3f3f3f3f
8 using namespace std;
9 const int dx[4]={1,0,-1,0};
10 const int dy[4]={0,1,0,-1};
11 struct Lux
12 {
13     int k,x,y;
14     Lux(int a,int b,int c):k(a),x(b),y(c){}
15     Lux(){}
16 };
17
18 int map[N][N],id[N][N],cnt;
19 int n,p,A,B,C;
20 int dist[K][N][N];
21 bool in[K][N][N];
22 Lux s,t;
23
24 int spfa() {
25     int i,vx,vy,fee,fee2;
26     queue<Lux>q;
27     memset(dist,0x3f,sizeof(dist));
28     dist[s.k][s.x][s.y]=0;
29     in[s.k][s.x][s.y]=1;
30     q.push(s);
31     while(!q.empty())
32     {
33         Lux U=q.front();q.pop();in[U.k][U.x][U.y]=0;
34         if(!U.k)continue;
35         for(fee=i=0;i<4;i++) {
36             vx=U.x+dx[i];
37             vy=U.y+dy[i];
38             if(i==2)fee=B; /* 往回走要多付的费用在这里处理。 */
39             if(!id[vx][vy])continue;
40             if(!map[vx][vy])
41                 {

```

```

42         fee2=C; /*新开油站，准确的说这里的思想是把全图都开成加油站，原加油站强制加
43         if (U.k&&dist[U.k-1][vx][vy]>dist[U.k][U.x][U.y]+fee)
44         { /*因为加油站强制加油，所以不是加油站才能这么转移。*/
45             dist[U.k-1][vx][vy]=dist[U.k][U.x][U.y]+fee;
46             if (!in[U.k-1][vx][vy]) in[U.k-1][vx][vy]=1, q.push(Lux(U.k-1, vx, vy));
47         }
48     }
49     else fee2=0; /*已经有原加油站了就不需要再付额外费用了*/
50
51     if (dist[p][vx][vy]>dist[U.k][U.x][U.y]+A+fee+fee2)
52     { /*加油的转移*/
53         dist[p][vx][vy]=dist[U.k][U.x][U.y]+A+fee+fee2;
54         if (!in[p][vx][vy]) in[p][vx][vy]=1, q.push(Lux(p, vx, vy));
55     }
56 }
57 }
58 int ret=inf;
59 for(i=0; i<=p; i++) ret=min(ret, dist[i][t.x][t.y]);
60 return ret;
61 }
62
63 int main() {
64     scanf("%d%d%d%d", &n, &p, &A, &B, &C);
65     for(int i=1; i<=n; i++) for(int j=1; j<=n; j++) scanf("%d", &map[i][j]), id[i][j]=++cnt;
66     s=Lux(p, 1, 1), t=Lux(0, n, n);
67     printf("%d\n", spfa());
68     return 0;
69 }

```

6.6 结果示例

7 最少费用购物问题

7.1 习题描述

某商店中每种商品都有一个价格。例如，一朵花的价格是 2 元；一个花瓶的价格是 5 元。为了吸引更多的顾客，商店提供了特殊优惠价。

特殊优惠商品是把一种或几种商品分成一组。并降价销售。例如：3 朵花

9	3	2	3	6					
0	0	0	0	1	0	0	0	0	0
0	0	0	1	0	1	1	0	0	0
1	0	1	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	1
1	0	0	1	0	0	1	0	0	0
0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	0
12									

图 2: 汽车加油行驶问题示例运行

的价格不是 6 而是 5 元;2 个花瓶加 1 朵花是 10 ICU 不是 12 元。

编一个程序, 计算某个顾客所购商品应付的费用。要充分利用优惠价以使顾客付款最小。请注意, 你不能变更顾客所购商品的种类及数量, 即使增加某些商品会使付款总数减小也不允许你作出任何变更。假定各种商品价格用优惠价如上所述, 并且某顾客购买物品为:3 朵花和 2 个花瓶。那么顾客应付款为 14 元因为:

1 朵花加 2 个花瓶: 优惠价:10 元; 2 朵花, 正常价:4 元

7.2 算法设计

对于给定预购商品的价格和数量, 以及优惠商品价, 计算所购商品应付的最少费用。

7.3 数据输入

用两个文件表示输入数据。第一个文件 INPUT. TXT 描述顾客所购物品 (放在购物筐中); 第二个文件描述商店提供的优惠商品及价格 (文件名

为 OFFER. TXT)。两个文件中都只用整数。

第一个文件 INPUT. TXT 的格式为: 第一行是一个数字 B (0 B 5), 表示所购商品种类数。下面共 B 行, 每行中含 3 个数 C, K, P。C 代表商品的编码 (每种商品有一个唯一的编码), 1 C 999。K 代表该种商品购买总数, 1 K 5。P 是该种商品的正常单价 (每件商品的价格), 1 P 999。请注意, 购物筐中最多可放 $5*5 = 25$ 件商品。

第二个文件 OFFER. TXT 的格式为: 第一行是一个数字 S (0 S 99), 表示共有 S 种优惠。下面共 S 行, 每一行描述一种优惠商品的组合中商品的种类。下面接着是几个数字对 (C, K), 其中 C 代表商品编码, 1 C 9 99。K 代表该种商品在此组合中的数量, 1 K 5。本行最后一个数字 P (1 P 999) 代表此商品组合的优惠价。当然, 优惠价要低于该组合中商品正常价之总和。

7.4 数据输出

将计算出的所购商品应付的最少费用输出到文件 OUTPUT.TXT。

7.5 分析

通过动态规划进行求解, 我们可以得到状态转移方程: $F[a, b, c, d, e] = \text{Min } F[a-S1, b-S2, c-S3, d-S4, e-S5] + \text{SaleCost}[S]$

初始条件为: $F[a, b, c, d, e] = \text{Cost}[1]*a + \text{Cost}[2]*b + \text{Cost}[3]*c + \text{Cost}[4]*d + \text{Cost}[5]*e$ 。即不用优惠的购买费用。

7.6 代码实现

```
1  #include <iostream>
2  #include <cstdio>
3  #include <algorithm>
4  #include <cstring>
5  #include <queue>
6  #include <set>
7  #include <map>
8  #include <time.h>
9  using namespace std;
10
```



```

11 int sale[1000][6] = {0};    //分别表示每个优惠中每个商品数量
12 int saleprice[1000] = {0}; //优惠总价
13 int salelength[1000] = {0}; //优惠总共有几个商品
14 int salenumber[1000][1000] = {0}; //优惠商品的ID
15 int good[6][4] = {0};      //1 -> number 2 -> price 3 -> last num
16 int num[1000];             //商品ID
17 int dp[6][6][6][6][6];
18 int n,m;
19
20 void input() {
21     cin>>n;
22     for(int i = 1; i <= n; i++) {
23         cin>>good[i][1]>>good[i][3]>>good[i][2];
24         num[i] = good[i][1];
25     }
26     cin>>m;
27     for(int i = 1; i <= m; i++) {
28         cin>>salelength[i];
29         for(int j = 1; j <= salelength[i]; j++) {
30             cin>>salenumber[i][j];
31             cin>>sale[i][salenumber[i][j]];
32         }
33         cin>>saleprice[i];
34     }
35 }
36
37 void output() {
38     for(int i = 1; i <= n; i++)
39         cout<<"goodnum: "<<good[i][1]<<" goodprice: "
40             <<good[i][2]<<" goodlast: "<<good[i][3]<<endl;
41     for(int i = 1; i <= m; i++) {
42         cout<<"sale"<<i<<" : ";
43         for(int j = 1; j <= salelength[i]; j++)
44             cout<<"num: "<<salenumber[i][j]<<"
45                 count: "<<sale[i][salenumber[i][j]]<<" ";
46         cout<<endl;
47         cout<<" price: "<<saleprice[i]<<endl;
48     }
49 }

```

```

50
51 int main() {
52     //freopen("in2","r",stdin);
53     input();
54     // output();
55     dp[0][0][0][0][0] = 0;
56     for(int i = 0; i <= good[1][3]; i++)
57         for(int j = 0; j <= good[2][3]; j++)
58             for(int k = 0; k <= good[3][3]; k++)
59                 for(int l = 0; l <= good[4][3]; l++)
60                     for(int p = 0; p <= good[5][3]; p++) {
61                         int minx = i * good[1][2] + j
62                             * good[2][2] + k * good[3][2]
63                             + l * good[4][2] + p * good[5][2];
64                         for(int q = 1; q <= m; q++) {
65                             if(i - sale[q][num[1]] < 0 || i - sale[q][num[2]] < 0 ||
66                                 i - sale[q][num[3]] < 0 || i - sale[q][num[4]] < 0 ||
67                                 i - sale[q][num[5]] < 0) continue;
68                             int t = dp[i - sale[q][num[1]]][j -
69                                 sale[q][num[2]]][k - sale[q][num[3]]
70                                 [l - sale[q][num[4]]][p - sale[q][num[5]]] + saleprice[q];
71                             if(t < minx) minx = t;
72                         }
73                         dp[i][j][k][l][p] = minx;
74                     }
75     cout << dp[good[1][3]][good[2][3]]
76         [good[3][3]][good[4][3]][good[5][3]] << endl;
77     return 0;
78 }

```

7.7 样例测试

8 收集样本问题

8.1 习题描述

机器人在一个正方形方格之内收集样本。试找出两条行走路径，使其取得的样本总价值最大。

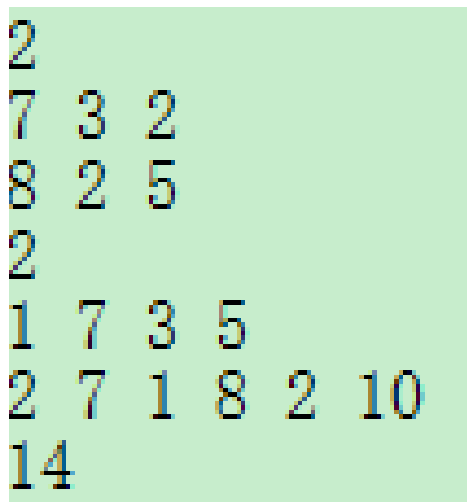


图 3: 最少购物费用问题示例运行

8.2 算法设计

给定方形区域 F 的样本分布，计算 Rob 的两条行走路径，使其取得的样本总价值最大。

8.3 数据输入

首先输入矩阵的规模，然后在输入样本所在的位置以及价值。

8.4 数据输出

计算得到的最大的样本总价值。

8.5 代码实现

```
1 #include <iostream>
2 #include <cstdio>
3 #include <string.h>
4 using namespace std;
5
6 #define MAXN 22
7
```

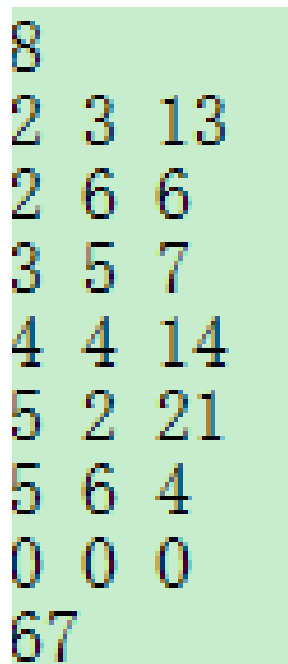
```

8  int h[MAXN][MAXN][MAXN][MAXN];
9  int v[MAXN][MAXN];
10 int n;
11
12 void update(int x1, int y1, int x2, int y2, int val) {
13     if(y1 >= n || y2 >= n) return;
14     if(x1 >= n || x2 >= n) return;
15     if(x1 == x2 && y1 == y2) {
16         h[x1][y1][x2][y2] = max(h[x1][y1][x2][y2],
17                                   val + v[x1][y1]);
18     } else {
19         h[x1][y1][x2][y2] = max(h[x1][y1][x2][y2],
20                                   val + v[x1][y1] + v[x2][y2]);
21     }
22 }
23
24 int main() {
25     scanf("%d", &n);
26     memset(v, 0, sizeof(v));
27     int x, y, val;
28     while(scanf("%d %d %d", &x, &y, &val) && x != 0) {
29         v[x - 1][y - 1] = val;
30     }
31     memset(h, 0, sizeof(h));
32     h[0][0][0][0] = v[0][0];
33
34     for(int s = 0; s < 2 * n - 2; ++s) {
35         for(int x1 = 0; x1 < n && x1 <= s; ++x1) {
36             for(int x2 = 0; x2 < n && x2 <= s; ++x2) {
37                 int y1 = s - x1;
38                 int y2 = s - x2;
39                 int v = h[x1][y1][x2][y2];
40
41                 update(x1 + 1, y1, x2 + 1, y2, v);
42                 update(x1 + 1, y1, x2, y2 + 1, v);
43                 update(x1, y1 + 1, x2 + 1, y2, v);
44                 update(x1, y1 + 1, x2, y2 + 1, v);
45             }
46         }

```

```
47     }  
48     printf("%d\n", h[n - 1][n - 1][n - 1][n - 1]);  
49     return 0;  
50 }
```

8.6 样例测试



8		
2	3	13
2	6	6
3	5	7
4	4	14
5	2	21
5	6	4
0	0	0
67		

图 4: 机器人收集样本问题示例运行

9 参考资料

1. LIS.<http://www.ahathinking.com/archives/117.html>
2. Thomas H.Cormen, Charles E.Leiserson, Ronald L.Rivest, Clifford Stein. Introduction to Algorithms.