**算法训练 方格取数**

时间限制：1.0s   内存限制：256.0MB

**问题描述**  
　　设有N\*N的方格图(N<=10),我们将其中的某些方格中填入正整数,而其他的方格中则放入数字0。  
　　某人从图的左上角的A 点(1,1)出发，可以向下行走，也可以向右走，直到到达右下角的B点(N,N)。在走过的路上，他可以取走方格中的数（取走后的方格中将变为数字0）。  
　　此人从A点到B 点共走两次，试找出2条这样的路径，使得取得的数之和为最大。  
**输入格式**  
　　输入的第一行为一个整数N（表示N\*N的方格图），接下来的每行有三个整数，前两个表示位置，第三个数为该位置上所放的数。一行单独的0表示输入结束。  
**输出格式**  
　　只需输出一个整数，表示2条路径上取得的最大的和。  
**样例输入**  
　　8  
　　2 3 13  
　　2 6 6  
　　3 5 7  
　　4 4 14  
　　5 2 21  
　　5 6 4  
　　6 3 15  
　　7 2 14  
　　0 0 0  
**样例输出**  
　　67

锦囊1

动态规划。

锦囊2

把一个人走两次看成两个人走一次，并且是并排走的，则这两个人除开左上和右下位置外其他的位置均不会重合。而左上和右下必取，所以只用算剩下的部分。 他们每次都在同一斜线上。 用F[i,j,k]表示走到第i条斜线上，第一个人在j位置，第二个人在k位置的最优值，则F[i,j,k]=F[i-1,j',k']+V(i,j)+V(i,k)，其中j',k'分别能走到j和k，V(i,j)表示第i条斜线上j位置的值。

本题的C++参考代码如下：

#include<stdio.h>

const int maxn = 10;

int map[maxn][maxn], dp[maxn][maxn][maxn][maxn];

#define max(a, b) (a > b ? a : b)

int main()

{

int n, x, y, w;

scanf("%d", &n);

while(scanf("%d%d%d", &x, &y, &w) == 3 && x && y && w)

{

map[x][y] = w;

}

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

{

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

{

for(int k = 1; k <= n; k++)

{

for(int l = 1; l <= n; l++)

{

int & tmp = dp[i][j][k][l];

int p = max(max(dp[i-1][j][k-1][l], dp[i-1][j][k][l-1]), max(dp[i][j-1][k-1][l], dp[i][j-1][k][l-1]));

tmp += (i == k && j == l) ? (p + map[i][j]) : (p + map[i][j] + map[k][l]);

}

}

}

}

printf("%d\n", dp[n][n][n][n]);

return 0;

}

本题的C参考代码如下：

#include <stdio.h>

#define max(a,b) a>b?a:b

#define min(a,b) a<b?a:b

int main()

{

int map[11][11]={0},f[11][11]={0};

int i,j,k,N,t;

scanf("%d",&N);

while (scanf("%d%d%d",&i,&j,&k)&&(i||j||k)) map[i][j]=k;

for (i=2;i<=2\*N;i++)

for (t=min(i,N),j=t;j>0;j--)

for (k=t;k>0;k--)

{

f[j][k]=max(f[j][k],f[j-1][k-1]);

f[j][k]=max(f[j][k],f[j][k-1]);

f[j][k]=max(f[j][k],f[j-1][k]);

if (j==k) f[j][k]+=map[j][i-j];

else f[j][k]+=map[j][i-j]+map[k][i-k];

}

printf("%d",f[N][N]);

return 0;

}

本题的Java参考代码如下：

import java.io.BufferedReader;

import java.io.IOException;

import java.io.InputStreamReader;

public class Main{

static int x;

static int y;

static int n;

public static void main(String[] args) throws IOException {

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));

n = Integer.parseInt(br.readLine());

int[][] tag = new int[2 \* n + 1][2 \* n + 1];

int[][] arr = new int[n \* n][2];

out: for (int i = 1; ; i++) {

String[] str = br.readLine().split(" ");

for (int j = 0; j < 1; j++) {

x = arr[i][0] = Integer.parseInt(str[0]);

y = arr[i][1] = Integer.parseInt(str[1]);

tag[x][y] = Integer.parseInt(str[2]);

if (x == 0 && y == 0 && tag[x][y] == 0){

dp(tag);

break out;

}

}

}

}

public static void dp(int[][] tag) {

int[][] temp = new int[2 \* n][2 \* n];

int k;

for (int i = 2; i <= 2 \* n; i++) {

for (int t = min(i, n), j = t; j > 0; j--) {

for (k = t; k > 0; k--) {

temp[j][k] = max(temp[j][k], temp[j - 1][k - 1]);

temp[j][k] = max(temp[j][k], temp[j - 1][k]);

temp[j][k] = max(temp[j][k], temp[j][k - 1]);

if (j == k)

temp[j][k] += tag[j][i - j];

else

temp[j][k] += tag[j][i - j] + tag[k][i - k];

}

}

}

System.out.println(temp[n][n]);

}

public static int max(int a, int b) {

return a > b ? a : b;

}

public static int min(int a, int b) {

return a > b ? b : a;

}

}