作业4

第1题：

代码如下

def movepattern(arr):

def movepattern(arr):

n=len(arr)

m=len(arr[0])

#无论多少种走法，到了终点可以算到走法的总数

map=[[1]] #将初始点的路线矩阵作为1

for i in range(n):

for j in range(m):

map[i].append(0) #当某点还没走，设为0

if arr[i][j]==1: #若遇到阻碍，则该点无法前进，将路线矩阵中该点设为0

map[i][j]=0

continue

if i>=1 and j >=0 and arr[i-1][j]==0: #若该点可以通行

map[i][j]+=map[i-1][j] #向下行走

if j>=1 and i>=0 and arr[i][j-1]==0:

map[i][j]+=map[i][j-1] #向右行走

map.append([])

return map[n-1][m-1] #终点路线矩阵值为实际可行走的路线数

arr=[[0,0,0,0,0],[0,1,0,1,0],[0,0,0,0,0]]

print movepattern(arr)

结果：



第2题：

代码如下：

def lowestcost(arr):

if arr is False:return 0

n=len(arr)

m=len(arr[0])

#以行为单位循环，计算从起点到地图矩阵该点的最小花费

#初始化pre列表，存储第一行的每点的花费

pre=[arr[0][0]]

for i in range(1,m):

pre.append(pre[i-1]+arr[0][i])

#从第二行开始

for i in range(1,n):

pre[0]=arr[i][0]+pre[0]

for j in range(1,m):

pre[j]=min(pre[j-1],pre[j])+arr[i][j]

#获得到第i行第j点的最低花费

return pre[m-1] #返回终点的最小花费

arr=[[1,1,1,4],[4,3,1,3],[3,1,2,1],[1,1,4,1]]

print lowestcost(arr)

结果：



第3题：

代码如下：

class Trituple: #define trituple

def \_\_init\_\_(self,a,b,c):

self.A=a

self.B=b

self.C=c

def printtri(self):

print [self.A,self.B,self.C]

def distance(tri): #define distance

return abs(tri.A-tri.B)+abs(tri.A-tri.C)+abs(tri.B-tri.C)

def findmindis(A,B,C):

map=[]

#put every trituple in a list

for i in A:

for j in B:

for x in C:

t=Trituple(i,j,x)

map.append(t)

min=map[0] #initialize the first trituple as the trituple with the shortest distance

for i in map:

if distance(i)<distance(min):

min=i

min.printtri()

print "The minimal distance is ",distance(min)

return

A = [13, 16, 19, 27]

B = [10, 14, 19, 24, 27, 35]

C = [15, 20, 23, 28, 31, 34, 38]

findmindis(A,B,C)

结果：



第4题：

def roadlength(arr):

n=len(arr)

data=[arr[n-1]] #从树塔底层开始计算

for i in range(n-2,-1,-1):

sum=[] #将第i层的每点最大值存入sum列表

k=len(data[0])

for j in range(0,k-1):

tmp=max(data[0][j],data[0][j+1])

sum.append(tmp+arr[i][j])

data.insert(0,sum)

return data[0][0] #树塔顶层即为所求最大路径长度

#以多维数组存储树塔

arr=[[17],[13,31],[11,20,15],[25,23,35,19],[14,22,37,18,14]]

print roadlength(arr)

结果如下：



第5题：

代码如下：

def submat(arr):

n=len(arr)

f=[[]]

#第0行为哨兵行

for i in range(1,n+1):

f.append([])

f[i].append(0)#第0列为哨兵列

f[i].append(arr[i-1][0])#存入第1列

for i in range(1,n+1):

for j in range(2,n+1):

f[i].append(f[i][j-1]+arr[i-1][j-1]) #将第i行第j列元素之前的所有元素（包括第j个元素）加和起来

p=[None] #p[k]为固定第i到j列求得包括从第1到k行范围内的最大子矩阵和

q=[None] #q[k]为固定第i到j列，从第1行道第k行的最大子矩阵

max=None #任何值大于None

#以下算法时间复杂度为O(n^3)

for i in range(1,n+1):

for j in range(i+1,n+1):

p.append(f[1][j]-f[1][i-1]) #p[k]为固定i和j求得包括k行的最大子矩阵和

x,y=1,1 #行的上下界从1开始

q.append(p[1])

for k in range(2,n+1):

if p[k-1]>0:

p.append(p[k-1]+f[k][j]-f[k][i-1])

else:

p.append(f[k][j]-f[k][i-1])

x=k #获得行上界

y+=1 #获得行下界

if q[k-1]>p[k]:

q.append(q[k-1])

else:

q.append(p[k])

if q[n]> max:

max=q[n] #第i到j列范围内最大的和

ii=i-1

ij=j-1

ix=x-1

iy=y-1

answer=[]

for i in range(ix,iy+1):

answer.append(arr[i][ii:ij+1])

return answer,max

arr=[[0,-2,-7,0],[9,2,-6,2],[-4,1,-4,1],[-1,8,0,-2]]

print submat(arr)

结果：

