Part1：阶梯型定价

1. 计算分区矩阵 Omega

首先，我们以校园的布局为基础，给出P区、M区、F区和不可停放区域G区，用矩阵Ω表示，元素分别为2、1、0、-1 。此外，将P区的长、宽以及中心存储在一个数组里，以便得到车辆的分布状况和产生随机数，从而获得事件矩阵。

【主要思路】

1. 首先给定一个矩形区域Ω（即校园），其长、宽均为整数，分别记为width=74、length=110;
2. 再由gly的辛苦劳作得到各个P区、M区、不可停放区域的坐标（左上坐标集合、右下坐标数组，根据这两个数组可以获得P区的中心、长、宽）；
3. 最后进行图像可视化即可；

In [1]:
```
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import matplotlib.pyplot as plt

from matplotlib.ticker import MultipleLocator, FormatStrFormatter
#此模块包含用于配置记号定位和格式的类。提供了通用的记号定位器和格式化程序，以及特定于
import pandas as pd
import math
import numpy.linalg as la
```

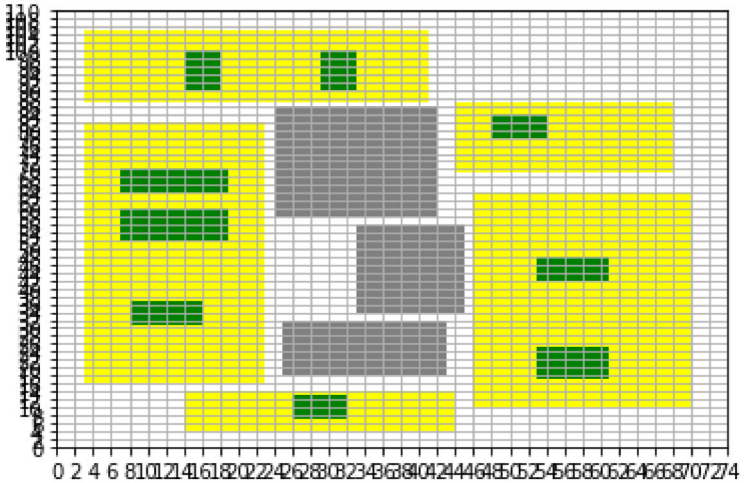
In [2]:
```
#首先，将P、M、G区的左上坐标、右下坐标分别记录在两个数组中，其中numP、numM、numG区分别
P_lu = np.array([[14,10],[29,10],[7,40],[7,50],[8,73],[26,97],[48,26],[53,62],[53,85]]
P_rd = np.array([[18,20],[33,20],[19,46],[19,58],[16,79],[32,103],[54,32],[61,68],[61,
P_lu[:,1] = 110 - P_lu[:,1]
P_rd[:,1] = 110 - P_rd[:,1]
M_lu = np.array([[3,5],[3,28],[14,96],[44,23],[46,46]])
M_rd = np.array([[41,23],[23,94],[44,106],[68,41],[70,100]])
M_lu[:,1] = 110 - M_lu[:,1]
M_rd[:,1] = 110 - M_rd[:,1]
G_lu = np.array([[24,24],[33,54],[25,78]])
G_rd = np.array([[42,52],[45,76],[43,92]])
G_lu[:,1] = 110 - G_lu[:,1]
G_rd[:,1] = 110 - G_rd[:,1]
```

In [3]:
```
#其次，对omega区域进行赋值，如下所示
width = 74
length = 110
numP = 9
numM = 5
numG = 3
Omega = np.zeros([width,length])
for gn in range(numG):
    for gx in range(G_lu[gn,0],G_rd[gn,0]):
        for gy in range(G_rd[gn,1],G_lu[gn,1]):
            Omega[gx,gy] = -1;
for mn in range(numM):
    for mx in range(M_lu[mn,0],M_rd[mn,0]):
        for my in range(M_rd[mn,1],M_lu[mn,1]):
            Omega[mx,my] = 1;
for pn in range(numP):
```

```
    for px in range(P_lu[pn,0],P_rd[pn,0]):
        for py in range(P_rd[pn,1],P_lu[pn,1]):
            Omega[px,py] = 2;
```

In [4]:
```
#为便于观看，将区域可视化（该步骤目的是为了更直观呈现图像）
def rectan(width,length,Omega):
    ax=plt.subplot(111)
    #可以使用三个整数，或者三个独立的整数来描述子图的位置信息。如果三个整数是行数、列数
    plt.xlim(0,width) #设定x坐标轴的范围
    plt.ylim(0,length) #设定y坐标轴的范围
    ax.xaxis.set_major_locator(MultipleLocator(2))#设置x主坐标间隔为 1
    ax.yaxis.set_major_locator(MultipleLocator(2))#设置y主坐标间隔 1
    ax.xaxis.grid(True,which='major')#major,color='black'
    ax.yaxis.grid(True,which='major')#major,color='black'
    for i in range(width):
        for j in range(length):
            if Omega[i,j] == 2:
                x=np.linspace(i,i+1)
                y1=j
                y2=j+1
                ax.fill_between(x,y1,y2,facecolor='green')
            elif Omega[i,j] == 1:
                x=np.linspace(i,i+1)
                y1=j
                y2=j+1
                ax.fill_between(x,y1,y2,facecolor='yellow')
            elif Omega[i,j] == -1:
                x=np.linspace(i,i+1)
                y1=j
                y2=j+1
                ax.fill_between(x,y1,y2,facecolor='gray')
    plt.show()
```

In [5]:
```
rectan(width,length,Omega)
```



2. 给定最近停车点函数NearestPark

获得分区矩阵后，我们可以直接得到每个点所对应的最近的P区格点，将格点坐标记录在停车点矩阵Park中。或者直接编写一个函数，不需要停车点矩阵，思路如下所示：

【主要思路】

- 如果该格点属于P区域，则输出endPoint；
- 如果该格点属于M区域，则运行NearestPark(2, endPoint)，得到最近距离以及P区停车点；
- 如果该格点属于F区域

- 运行NearestPark(2, endPoint)，得到最近距离minDist以及P区停车点
- 运行NearestPark(1, endPoint)，得到最近距离minDist以及M区停车点

1. 输入：目标地区的横坐标x、纵坐标y以及flagvalue（如果flag=1，则代表寻找最近的M区；如果flag=2，则代表寻找最近的P区）
2. switch 1：遍历全部格点，计算所有M区到每个格点到（x,y）的距离，输出最小值及其坐标
3. switch 2：遍历全部格点，计算所有P区到每个格点到（x,y）的距离，输出最小值及其坐标
4. over!

In [6]:
```
# 计算距离终点最近的P/M区的点的坐标及距离
## 输入：
## flagValue = 2：最近的P区；1：最近的M区
## endPoint = array([x坐标,y坐标])：随机事件的终点
##
def NearestPark(flagValue, endPoint):
    # 初始化距离矩阵Dist（即空间的每个格点到随机事件终点的距离形成一个与Omega大小相同的
    Dist = np.zeros([74,110])
    for i in range(74):
        for j in range(110):
            Dist[i,j]=la.norm(endPoint-np.array([i,j]))
    minDist=np.min(Dist[Omega==flagValue])
    nearestPoint=np.where(Dist == np.min(Dist[Omega==2]))
    FinalnearestPoint=np.array([nearestPoint[0][0],nearestPoint[1][0]])
    return minDist,FinalnearestPoint
```

In [7]:
```
a1,a2=NearestPark(2,[0,88])
a2
```

Out[7]: array([2, 74], dtype=int64)

3. 事件矩阵 Event

事件矩阵Event是一个shape为$m*n*2*2$的四维矩阵，第一维是用户数目，第二维是时刻数，第三维是起点&终点，第四维是横坐标&纵坐标。

理解：事件矩阵的行向量对应某用户在每个时刻所发生的事件，列向量对应某时刻每个用户所发生的事件的起、终点

下面讨论如何随机产生事件矩阵（二维区域随机投点）：

【主要思路】

1. 输入事件矩阵的行数m和列数n，即用户个数m和总时刻数n;
2. 再产生2mn个叠加后的正态分布的随机点;
3. 将这2mn个随机点赋给事件矩阵。

基本参数假设：

1. 不同p区的概率未给定，这里假设相同，即累积分布函数 $F = [1/9, 2/9, 3/9, 4/9, 5/9, 6/9, 7/9, 8/9, 9/9]$;
2. 在正态分布中，落在P区范围内服从~原则;

In [8]:
```
F = [1/9,2/9,3/9,4/9,5/9,6/9,7/9,8/9,9/9]
```

In [9]:
```
#首先，给定到达不同p区（共9个）的累积分布向量F、U(0,1)上的随机数eta、起点start、终点er
def bisection_search(F, eta, start, end):
    if (eta <= F[start]):
```

```
        return start
    n = end - start
    if (n <= 0):
        sys.exit()
    k = (start + end) // 2
    if (eta > F[k]):
        if (eta <= F[k + 1]):
            return k
        else:
            return bisection_search(F, eta, k + 1, end)
    else:
        return bisection_search(F, eta, start, k)
```

In [10]:
```
#其次，尝试生成叠加后的二维正态分布的随机点
#step1：生成一个服从U(0,1)的随机数，并带入bisection_search函数，从而得到随机点所属P区；
#step2：计算该P区的中心（（右下横坐标+左上横坐标）/2，（右下纵坐标+左上纵坐标）/2）、
#step3：获得随机点
def rand_PDF():
    #首先产生服从累积分布F的随机数
    U = np.random.rand()
    X = bisection_search(F, U, 0, numP)
    #再产生对应P区的二维正态分布随机数
    mu_x = (P_rd[X,0]+P_lu[X,0])/2
    mu_y = (P_lu[X,1]+P_rd[X,1])/2
    sigma_x = (P_rd[X,0]-P_lu[X,0])/6
    sigma_y = (P_lu[X,1]-P_rd[X,1])/6
    mu = np.array([mu_x,mu_y])
    Sigma = np.array([[sigma_x**2, 0], [0,sigma_y**2]])
    s = np.random.multivariate_normal(mu,Sigma,1)
    s = s[0]
    if 0<=s[0]<=width and 0<=s[1]<=length and Omega[int(s[0]),int(s[1])]!=-1:
        return s
    else:
        s = rand_PDF()
```

In [11]:
```
#若干个随机产生事件矩阵，其中m为行数，即用户个数；n为列数，即总时刻数。
def geEvent(m,n):
    #用shape为m*n*2的Event矩阵存储每个用户在每个时刻下发生的事件
    Event=np.zeros([m,n,2,2])
    #首先，随机生成所有事件的横、纵坐标
    E = np.array([rand_PDF() for i in range(2*m*n)])
    Event = E.reshape(m,n,2,2)
    return Event
```

In [12]:
```
m = 100
n = 100
```

In [13]:
```
Event = geEvent(m,n)
```

4. 给定方格矩阵 Cell

方格矩阵代表了各个小方格上的自行车数量。

- 方格矩阵初始化；
- 方格矩阵的更新：在每个时刻事件发生过程中都要更新一次；

【主要思路】

1. 方格矩阵的初始化：所有P区的方格赋值为2（比如说），其他区域全部为0。
2. 方格矩阵的变化：（自上至下，自左至右）每个时刻 t_i 下，每个人从起点前往终点，只有在确定该人是否骑车以及停车位置后才能改变方格矩阵。

- 利用方格矩阵，首先判断是否该地区是否有单车；
- 再考虑距离因素，过大或者过小都不适合骑车；
- 其次，根据停车点矩阵和内部判定条件确定是否更改终点；
- 若此人骑车，则改变方格矩阵；否则方格矩阵不变。

In [14]:
```
# 判定是否有车
## 作用：判定方格矩阵中某方格周围共9个方格内是否有车，没车就直接判定为不骑车，有车再骑
## 输入：某个点
## 输出：周围方格是否有车以及新的起点
## 查阅可得，紫金港占地面积5740000平方米，我们的矩阵是74*110，相当于缩小了700倍，依据这
def judgeCell(point):
    px = int(point[0])
    py = int(point[1])
    if Cell[px,py]!=0:
        return [px,py],1   #起点不变，且该方格内有车
    else:
        for i in range(px-1,px+2):
            for j in range(py-1,py+2):
                if Cell[i,j]!=0:
                    return [i,j],1   #起点改变，周围方格内有车
                else:
                    continue
        return [px,py],0   #起点不变，该方格和周围方格都无车。判定为不骑车！

# 初始化方格矩阵
## 作用：每次搬车后，车子回到初始状态——P区(2)1辆车，M区(1)和F区(0)都是0辆车
def initialCell():
    Cell = np.zeros([74,110])
    Cell[Omega==2] = 1
    Cell[Omega==1] = 0
    Cell[Omega==0] = 0
    return Cell

#更新该方格的车
# 作用：每一件事件发生后都需要重新更新方格矩阵
# 输入：起点和终点
# 无输出，仅对全局变量Cell做更改
def updateCell(startpoint,endpoint):
    sx = int(startpoint[0])
    sy = int(startpoint[1])
    Cell[sx,sy] = Cell[sx,sy] + 1
    ex = int(endpoint[0])
    ey = int(endpoint[1])
    Cell[ex,ey] = Cell[ex,ey] + 1
```

In [15]:
```
#给定各个参数
park=np.zeros([m,n,2])
distance=np.zeros([m,n])
a=4
#定义a<Cp<C<Cm<Cf<Df<Dm<D<Dp<b
#定义p和m
C=20
Cp=12
Cm=40
Cf=60
Df=800
Dm=800
Dp=800
D=800
b=800
limitp=12
limitm=4
```

In [16]:
```
##计算距离
Cell=initialCell()
for i in range(m):
    for j in range(n):
        distance[i,j] = np.linalg.norm(Event[i,j,1]-Event[i,j,0])#事件矩阵起点和终点当

##判定是否骑车及其起点和终点
for i in range(m):
    for j in range(n):
        if a<distance[i,j] and distance[i,j]<b:#（排除极端情况）#if 这里有车才能进行:
            vector0 = Event[i,j,0]
            vector1,flag = judgeCell(vector0)
            vector2 = Event[i,j,1]
            if flag == 1:
                if Omega[int(vector2[0]),int(vector2[1])] == 2:  #（终点在p区内）
                    distance[i,j] = np.linalg.norm(vector2 - vector1)
                    park[i,j] = vector2 #（停车点距离相应的就等于终点的位置，录入到矩阵
                    if Cp>=distance[i,j] or distance[i,j]>=Dp:
                        distance[i,j] = 0
                    else:
                        updateCell(vector1,vector2)
                elif Omega[int(vector2[0]),int(vector2[1])] == 1:#（终点在m区）
                    Kp,vector3 = NearestPark(2, vector2)#此处用xll的方法找到最近的p区的
                    if Kp >= limitp:
                        distance[i, j] = np.linalg.norm(vector2 - vector1)
                        park[i,j] = vector2 #（停车点距离相应的就等于终点的位置，录入到
                        if Cm>=distance[i,j] or distance[i,j]>=Dm:
                            distance[i, j] = 0
                        else:
                            updateCell(vector1,vector2)
                    elif Kp < limitp:
                        distance[i, j] = np.linalg.norm(vector3-vector1)# 就是那个p点
                        park[i,j] = vector3
                        if Cp>=distance[i,j] or distance[i,j]>=Dp:
                            distance[i, j]=0
                        else:
                            updateCell(vector1,vector3)
                elif Omega[int(vector2[0]),int(vector2[1])] == 0:  # （终点在f区）
                    Kp,vector3 = NearestPark(2, vector2)#此处用xll的方法找到最近的p区的
                    Km,vector4 = NearestPark(1, vector2)#此处用xll的方法找到最近的m区的
                    if Kp<limitp:
                        distance[i,j] = np.linalg.norm(vector3-vector1)  # 就np.linal
                        park[i,j] = vector3
                        if Cp>=distance[i,j] or distance[i,j]>=Dp:
                            distance[i, j] = 0
                        else:
                            updateCell(vector1,vector3)
                    elif Kp>=limitp and Km<limitm:
                        distance[i,j] = np.linalg.norm(vector4-vector1) # 就是那个m点
                        park[i,j] = vector4
                        if Cm>=distance[i,j] or distance[i,j]>=Dm:
                            distance[i, j] = 0
                        else:
                            updateCell(vector1,vector4)
                    else:#就是此时Kp比p大，而且Km比m大
                        distance[i,j] = np.linalg.norm(vector2-vector1)# 就是终点减去
                        park[i,j] = vector2
                        if Cf>=distance[i,j] or distance[i,j]>=Df:
                            distance[i, j] = 0
                        else:
                            updateCell(vector1,vector2)
            else:
                distance[i,j] = 0
        else:
```

```
            distance[i,j] = 0
#这一部分实际上就是为了计算相应的费用，需要结合5，如下所示：
```

5. 计算盈利情况

- 成本：搬运费
- 收入：单车使用费、广告费

In [17]:
```
# 利用方格矩阵计算搬运费
updatetime = 50
p1 = 1
p2 = 1.5
p3 = 2
cost = np.zeros(int(n/updatetime))  #初始化搬运费用，共有n/updatetime次
def countcost(Cell,t):  #其中t代表当前时刻，即t = i + 1
    if t%updatetime == 0:
        countF = np.sum(Cell[Omega==0])
        countM = np.sum(Cell[Omega==1])
        countP = np.sum(Cell[Omega==2])  #由于P区中部分车无需搬运，这部分要如何计算？
        cost[t//updatetime - 1] = p1*countF + p2*countM + p3*countP  #搬运费是不同地
        initialCell()    #搬运车辆，即初始化
```

In [18]:
```
##计算距离
Cell=initialCell()
for i in range(m):
    for j in range(n):
        distance[i,j] = np.linalg.norm(Event[i,j,1]-Event[i,j,0])#事件矩阵起点和终点坐

##判定是否骑车及其起点和终点
for i in range(m):
    for j in range(n):
        if a<distance[i,j] and distance[i,j]<b:#（排除极端情况）#if 这里有车才能进行:
            vector0 = Event[i,j,0]
            vector1,flag = judgeCell(vector0)
            vector2 = Event[i,j,1]
            if flag == 1:
                if Omega[int(vector2[0]),int(vector2[1])] == 2:  #（终点在p区内）
                    distance[i,j] = np.linalg.norm(vector2 - vector1)
                    park[i,j] = vector2 #（停车点距离相应的就等于终点的位置，录入到矩
                    if Cp>=distance[i,j] or distance[i,j]>=Dp:
                        distance[i,j] = 0
                    else:
                        updateCell(vector1,vector2)
                elif Omega[int(vector2[0]),int(vector2[1])] == 1:#（终点在m区）
                    Kp,vector3 = NearestPark(2, vector2)#此处用xll的方法找到最近的p区
                    if Kp >= limitp:
                        distance[i, j] = np.linalg.norm(vector2 - vector1)
                        park[i,j] = vector2 #（停车点距离相应的就等于终点的位置，录入
                        if Cm>=distance[i,j] or distance[i,j]>=Dm:
                            distance[i, j] = 0
                        else:
                            updateCell(vector1,vector2)
                    elif Kp < limitp:
                        distance[i, j] = np.linalg.norm(vector3-vector1)# 就是那个p点
                        park[i,j] = vector3
                        if Cp>=distance[i,j] or distance[i,j]>=Dp:
                            distance[i, j]=0
                        else:
                            updateCell(vector1,vector3)
                elif Omega[int(vector2[0]),int(vector2[1])] == 0:  # （终点在f区）
                    Kp,vector3 = NearestPark(2, vector2)#此处用xll的方法找到最近的p区
```

```
                    Km,vector4 = NearestPark(1, vector2)#此处用xll的方法找到最近的m区
                    if Kp<limitp:
                        distance[i,j] = np.linalg.norm(vector3-vector1)  # 就np.linal
                        park[i,j] = vector3
                        if Cp>=distance[i,j] or distance[i,j]>=Dp:
                            distance[i, j] = 0
                        else:
                            updateCell(vector1,vector3)
                    elif Kp>=limitp and Km<limitm:
                        distance[i,j] = np.linalg.norm(vector4-vector1) # 就是那个m点
                        park[i,j] = vector4
                        if Cm>=distance[i,j] or distance[i,j]>=Dm:
                            distance[i, j] = 0
                        else:
                            updateCell(vector1,vector4)
                    else:#就是此时Kp比p大，而且Km比m大
                        distance[i,j] = np.linalg.norm(vector2-vector1)# 就是终点减去
                        park[i,j] = vector2
                        if Cf>=distance[i,j] or distance[i,j]>=Df:
                            distance[i, j] = 0
                        else:
                            updateCell(vector1,vector2)
                else:
                    distance[i,j] = 0
            else:
                distance[i,j] = 0
            countcost(Cell,i+1)  #计算是否需要重新搬运，对cell矩阵进行初始化
```

In [19]:
```
sumcost = sum(cost)
```

In [20]:
```
# 单车骑行收费和广告费用
k1=0.5    #一个参数p区收费和距离的参数
k2=1.5   #一个参数m区收费和距离的参数
k3=2.5    #一个参数f区收费和距离的参数
g1=50    #一个参数广告和人数的参数
def countincome():
    income1 = 0
    income2 = 0
    for i in range(m):
        for j in range(n):
            feetype = Omega[int(park[i,j,0]),int(park[i,j,1])]
            if feetype == 2:
                income1 = income1 + (int(distance[i,j]/120)+1)*k1
            elif feetype == 1:
                income1 = income1 + (int(distance[i,j]/120)+1)*k2
            elif feetype == 0:
                income1 = income1 + (int(distance[i,j]/120)+1)*k3
        income2 = np.sum(distance!=0)*g1 #广告费
    return income1,income2
```

In [21]:
```
inc1,inc2 = countincome()
sumincome = inc1 + inc2
```

In [22]:
```
# 综上所述，共享单车公司的总利润为：收入 - 成本
sumbenifit = sumincome - sumcost
```

In [23]:
```
sumbenifit1=sumbenifit
```

In [24]:
```
sumbenifit1
```

Out[24]: 377838.0

Part2：统一型定价

In [25]:
```
park=np.zeros([m,n,2])
distance=np.zeros([m,n])
a=4
#定义a<Cp<C<Cm<Cf<Df<Dm<D<Dp<b
#定义p和m
C=20
Cp=12
Cm=40
Cf=60
Df=800
Dm=800
Dp=800
D=800
b=800
limitp=12
limitm=4

##计算距离
for i in range(m):
    for j in range(n):
        distance[i,j] = np.linalg.norm(Event[i,j,1]-Event[i,j,0])#事件矩阵起点和终点当

##判定是否骑车及其起点和终点
for i in range(m):
    for j in range(n):
        if a<distance[i,j] and distance[i,j]<b:#（排除极端情况）#if 这里有车才能进行:
            vector0 = Event[i,j,0]
            vector1,flag = judgeCell(vector0)
            vector2 = Event[i,j,1]
            if flag == 1:
                if Omega[int(vector2[0]),int(vector2[1])] <= 2 and Omega[int(vector2[
                    distance[i,j] = np.linalg.norm(vector2 - vector1)
                    park[i,j] = vector2 #（停车点距离相应的就等于终点的位置，录入到矩阵
                    if C>=distance[i,j] or distance[i,j]>=D:
                        distance[i,j] = 0
                    else:
                        updateCell(vector1,vector2)
            else:
                distance[i,j] = 0
        else:
            distance[i,j] = 0
```

In [26]:
```
# 利用方格矩阵计算搬运费
updatetime = 50
p1 = 1
p2 = 1.5
p3 = 2
cost = np.zeros(int(n/updatetime))  #初始化搬运费用，共有n/updatetime次
def countcost(Cell,t):  #其中t代表当前时刻，即t = i + 1
    if t%updatetime == 0:
        countF = np.sum(Cell[Omega==0])
        countM = np.sum(Cell[Omega==1])
        countP = np.sum(Cell[Omega==2])  #由于P区中部分车无需搬运，这部分要如何计算？
        cost[t//updatetime - 1] = p1*countF + p2*countM + p3*countP  #搬运费是不同地
        initialCell()    #搬运车辆，即初始化
```

In [27]:
```
##计算距离
Cell=initialCell()
for i in range(m):
    for j in range(n):
        distance[i,j] = np.linalg.norm(Event[i,j,1]-Event[i,j,0])#事件矩阵起点和终点当
```

```
##判定是否骑车及其起点和终点
for i in range(m):
    for j in range(n):
        if a<distance[i,j] and distance[i,j]<b:#（排除极端情况）#if 这里有车才能进行:
            vector0 = Event[i,j,0]
            vector1,flag = judgeCell(vector0)
            vector2 = Event[i,j,1]
            if flag == 1:
                if Omega[int(vector2[0]),int(vector2[1])] <= 2 and Omega[int(vector2[
                    distance[i,j] = np.linalg.norm(vector2 - vector1)
                    park[i,j] = vector2 #（停车点距离相应的就等于终点的位置，录入到矩
                    if C>=distance[i,j] or distance[i,j]>=D:
                        distance[i,j] = 0
                    else:
                        updateCell(vector1,vector2)
                else:
                    distance[i,j] = 0
        else:
            distance[i,j] = 0
        countcost(Cell,i+1)  #计算是否需要重新搬运，对cell矩阵进行初始化
```

In [28]:
```
sumcost = sum(cost)
```

In [29]:
```
# 单车骑行收费和广告费用
k1=0.5    #一个参数p区收费和距离的参数
k2=1.5   #一个参数m区收费和距离的参数
k3=2.5    #一个参数f区收费和距离的参数
g1=50    #一个参数广告和人数的参数
def countincome():
    income1 = 0
    income2 = 0
    for i in range(m):
        for j in range(n):
            feetype = Omega[int(park[i,j,0]),int(park[i,j,1])]
            if feetype == 2:
                income1 = income1 + (int(distance[i,j]/120)+1)*k1
            elif feetype == 1:
                income1 = income1 + (int(distance[i,j]/120)+1)*k2
            elif feetype == 0:
                income1 = income1 + (int(distance[i,j]/120)+1)*k3
        income2 = np.sum(distance!=0)*g1 #广告费
    return income1,income2
```

In [30]:
```
inc1,inc2 = countincome()
sumincome = inc1 + inc2
```

In [31]:
```
# 综上所述，共享单车公司的总利润为：收入 - 成本
sumbenifit = sumincome - sumcost
```

In [32]:
```
sumbenifit2=sumbenifit
```

In [33]:
```
sumbenifit2
```

Out[33]: 351600.5

Part3：二者比较

在m=100，n=100下，阶梯型和统一型定价结果如下所示：

- benifit1：377838.0
- benifit2：351600.5

在人工参数下，阶梯型比统一型更佳。

In [ ]: