* Floyd算法运用 Matlab 软件编程计算，得到各社区间的最短距离D矩阵和Path路径矩阵，把两个矩阵保存下来，利用js进行矩阵操作，就能在断网的情况下查表得到任意两点之间的最短距离和路线规划方案，最短路线规划和距离总数。

~~吉林大学南岭校区校区地图最短路径实现，通过Flody算法实现，对数据进行计算，实现校内路径最短路巡航过程的最短路优化，实现路径最短最短路选择。~~

~~计划步骤：~~

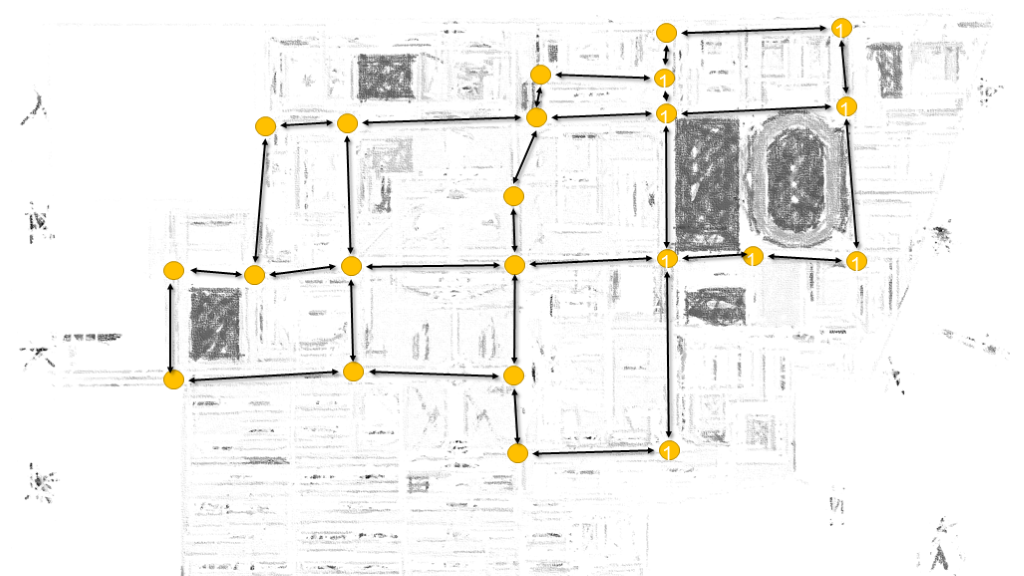
~~0 计算机图像识别，图像处理，卫星地图图像处理得到每个路口节点的坐标，自动通过卫星扫描地图实现图论模型的建立，选择地点的时候，则通过算法判断输入地点和道路交点的路径的最短距离右向车道行驶的情况下，哪个节点距离比较近就用哪个？~~

~~1 直接用Dijstra算法的javascript代码逻辑重写，实现路径判最小，但是因为js慢，每次都要重新算计算代价比较大~~

~~2 更改方案，选用Flody算法，根据校区地图最短路求解导航的项目，地图和路径，用matlab跑代码出矩阵最短路结果，然后js逻辑层就只需要查找已知地点最短路的矩阵了，或者代码重构直接计算~~

吉林大学南岭校区的图形分布，和某校区的路线，根据不同地区的学生活动区域，就近分组，对地图进行适当的简化，分组情况如下：

1，2，3公寓；4+11公寓；3餐厅+9公寓；4餐厅+8公寓；13公寓；5餐+7公寓+保卫办+快递站+交通楼；5，6公寓+1餐；大虾；图书馆；二教；致远超市；一教；三教前路口；逸夫楼+管理学院；路口1；路口2；体育馆；篮球场入口；体育入口；东门；西门；南门；北门；能源动力大楼+工训中心+工厂；工程小院；路口三+交通楼；计算机中心



首先，搜集资料，得到某校区的卫星地图的绿化地图，利用计算机软件Matlab等对原图像进行图像处理，如图1-1所示，分组完毕，得到真正的交通路线，就近选择交通路口作为节点，链接重要交通地形的节点。

以上图为例子，本文可以根据原图像得到初步的地图分布，结合相关数据，不难得到某校区地形的基本路线图。

根据基本路线图，可以建立图论相关简化模型，然后

* 1. 建立校园地理特征的图论模型

针对最常见的学生活动路线进行分析，把最关键的交通路口作为结点，建立关键的工作节点，链接不同的节点，根据校区内复杂的交通路网，对真实问题中复杂的交通路网进行数学语言的简化描述，根据百度地图测量节点之间的实际距离，以米为单位标注为图中节点之间的权重，可以得到加权无向图，如下所示：

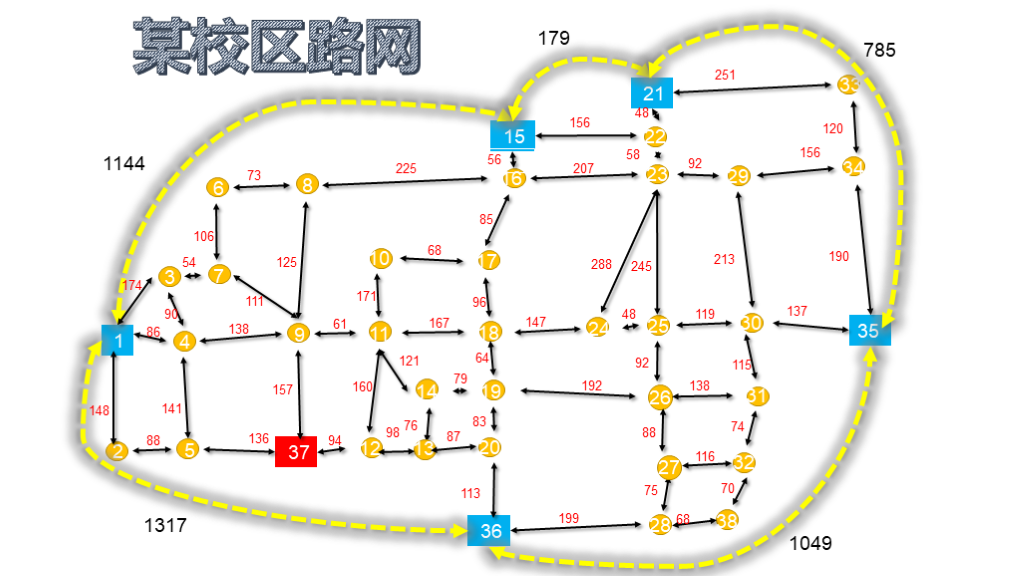


图1-3

路网说明，上面把某校区的实际区域分布抽象为37个没有空间面积的质点，质点之间的数字代表质点之间的距离（单位为米）。其中注释比较重要的几个地点，

1. 西门；
2. 大北门
3. 小北门

35 东门

36 南门

注：因为小南门一般不通车，或者学生很少路过，车流量，人流量非常小，故忽略此地。同样的道理，校内有一些区域的地形会予以忽略。

图1-3是一个加权有向图

其中



在图与网络分析的应用中，将面临一个问题——如何分析、计算一个较大型的网络，这当然需借助快速的计算工具——计算机。那么，如何将一个图表示在计算机中，或者，如何在计算机中存储一个图呢？现在已有很多存储的方法，但最基本的方法就是采用矩阵来表示一个图，图的矩阵表示也根据所关心的问题不同而有——邻接矩阵、关联矩阵、权矩阵等。

根据Matlab处理矩阵的特性，把某大学的图形存储为三种格式：

一 ,邻接矩阵

邻接矩阵——对于图

有n×n 阶 方矩阵

其中:

二 ,关联矩阵

关联矩阵——对于图

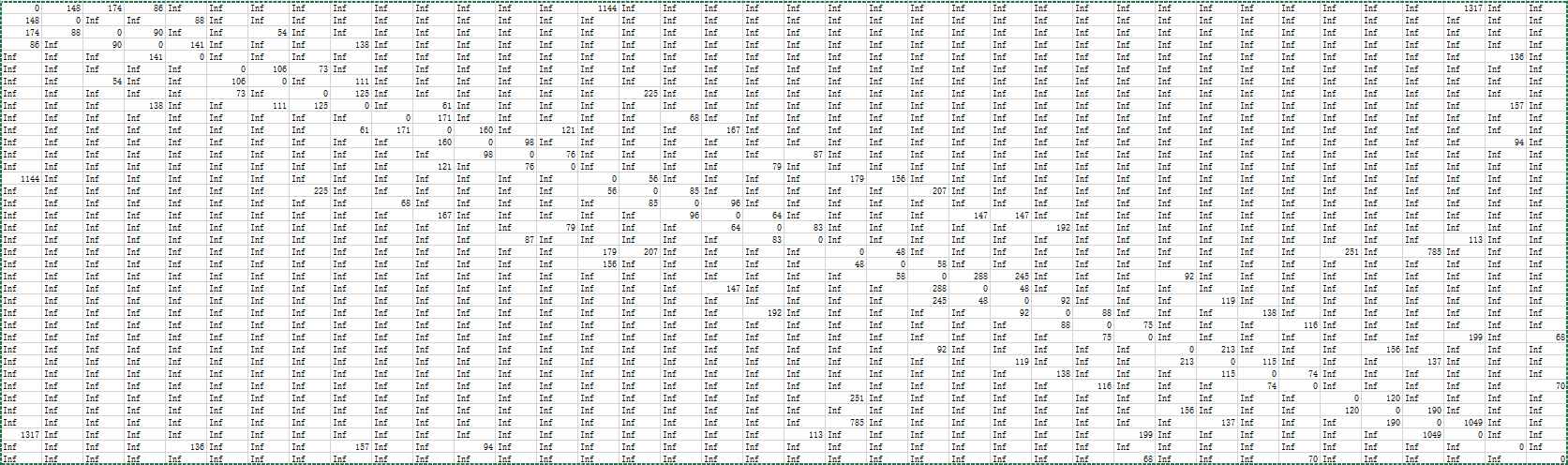
有m×n 阶 方矩阵

其中:

有m×n 阶矩阵M =（mij）m×n，

三 ,权矩阵

权矩阵（详见附录）



**假设**

把每个建筑物正门作为建筑的位置，以两个建筑物的前门的距离作为点之间的权重值。

不考虑突发性赛车或者校区内施工等意外或者事故导致的“道路不同的现象”。

暂时不考虑人口，上课路线，交通状况，人流分布，或者特殊的社会活动等因素，仅仅从校区内路网交通现况进行考虑。

基于Flody算法进行模型改进

Floyd算法Matlab

%调用方法为：[D,path]=floyd(w)。给出的结果D为路径的邻接矩阵，path为路径所经过的端点顺序。程序为：

function [D,path]=floyd(w)

%D R a

n=size(w,1);

%设初值

D=w;

path=zeros(n);

for i=1:n

for j=1:n

if D(i,j)~=inf

path(i,j)=j;

end

end

end

%迭代，更新D path

for k=1:n

for i=1:n

for j=1:n

if D(i,k)+D(k,j)<D(i,j)

D(i,j)=D(i,k)+D(k,j);

path(i,j)=path(i,k);

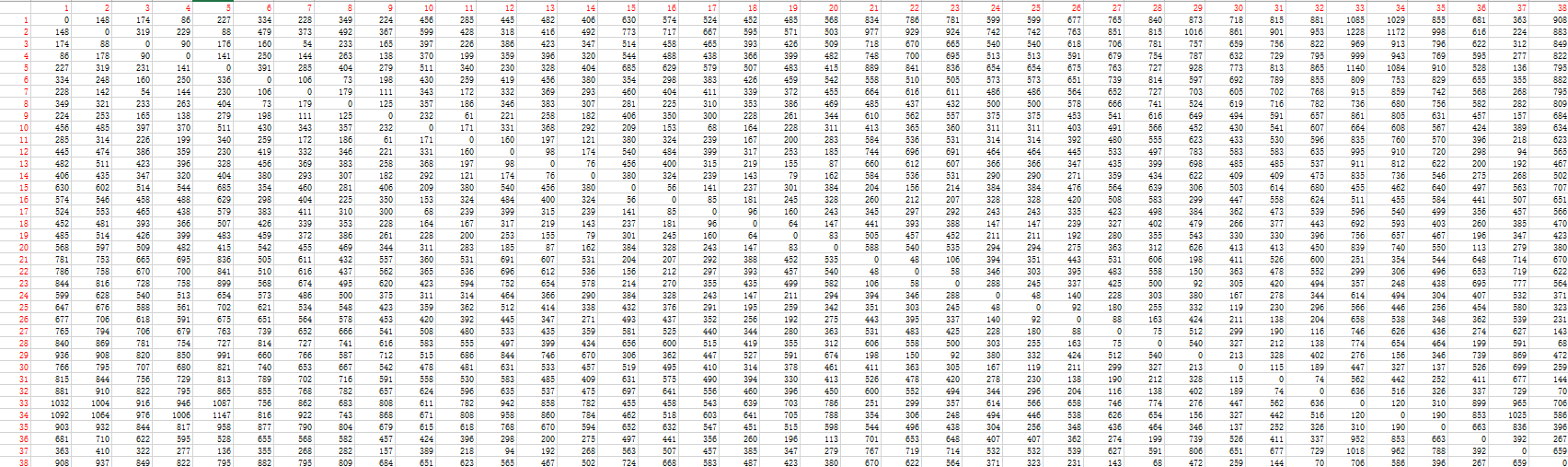
end

end

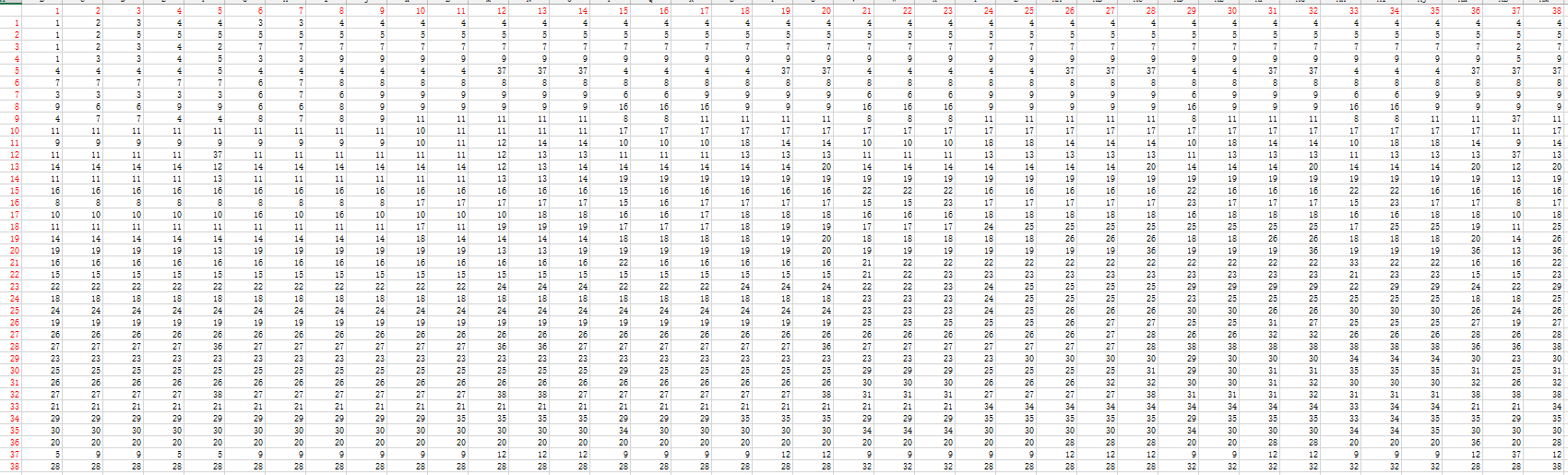
end

end

3.2校区内运用Floyd算法得到的图论模型求解



得到D矩阵



Path矩阵

**根据上述矩阵，我们很容易得到任意两点之间的最短距离和对应的路径。**

举个例子,

同样假设目标路径从37保卫办到21小北门，

查看D[37][21]=767,即两点之间发现最短距离和Dijkstra算法计算结果相同，均为767米。

再观察Path矩阵，

首先，

Path[37][21]=9,

Path[9][21]=8,

Path[8][21]=16,

Path[16][21]=15,

Path[15][21]=22,

Path[22][21]=21

也就是说，从37保卫办到21小北门的最短路径是

7->9->8->16->15->22->21

首先根据校区的地形图，对校内的区域路网进行规划，设计，建模，建立描述某大学校区的加权无向图，根据相关的理论，通过计算机编程计算。不需要每次都用Matlab等编程语言进行计算，可以在小程序中输入输入输出地点，然后计算和输入距离最近的输入和输出地点，查表得到最优路线，免去重复计算，通过一次性的计算的结果得到某大学校区的任意两点之间的最短距离和路线规划。

因为交通路网建立的过程中，没有考虑到每个公寓的居住人口，可以加入居住人口和日常出行，精确规划交通。同时，因为每一个公寓的学生都是同一批年纪，可以加入他们的课程表信息，得到每日的人流量数据，得到学生交通流量的时间高峰和人流量高峰地段------校园宣传或者小广告派发能做到更加精准高效。

可以通过统计的方法，得到校区内的“事故高发区域”，认为设置权重，进一步对问题进行更精确的仿真，更符合真实情况的算法结论。

关于算法方面的优化，可以尝试运用神经网等现代算法对问题进行路线优化，模拟，从而得到对现实情况更加精准的仿真。

基于原问题模型假设的简化程度过高，还可以对问题进行进一步放宽假设，加强对现实情况的刻画程度，使预估效果更好，比如不把建筑区域抽象为质点，而是把建筑的面积和建筑之间的关系都考虑在一起。

除了考虑现有的汽车巡航路线，还可以考虑行人路线的道路规划，因为行人走的小路的最佳路线规划和汽车不一样，理应把校区内的路网分布更加复杂地设置为人行道排布的路网，从而得到针对行人的最短路线规划。

还能考虑校区内餐厅，快递，购物场所运动场所的的位置，考虑多目标规划下的最优路线规划，道路巡航规划。

**推广意义**

上述想法，只要在校内的路网能够加以实现，说明其他的路网全部都能采用相同的方法。

当进行城市交通路网规划的时候，可以根据车主的导航和日常统计大数据，实时反馈正常城市交通的轨道途径，为以后的交通出行规划做出更科学，更加合理的规划，建立更加合理的科学的城市交通规划实时调控机制。

通过社会调查和统计的方法，可以得到路网交通每日的人流量数据，得到学生交通流量的时间高峰和人流量高峰地段------校园宣传或者小广告派发能做到更加精准高效。

本问题的算法，很容易能够转移到其他编程语言去实现，比如在服务器上调用百度地图的API,根据Javascript改写的Dijkstra算法进行计算，不难得到任何地区想要求解的两点之间的道路规划。再有甚至，可以挂接在网站服务器上面，建立一个实时的Web应用，方便任何访问此网站的人员使用。

Floyd算法就更有推广价值了，只要以此计算得到D矩阵和Path矩阵，就能把矩阵打印出来，每次需要得到路线的时候就查表分析，短时间内计算出结果，摆脱对计算机和网络的依赖，轻松得到想要的结果。

0 计算机图像识别，图像处理，卫星地图图像处理得到每个路口节点的坐标，自动通过卫星扫描地图实现图论模型的建立，选择地点的时候，则通过算法判断输入地点和道路交点的路径的最短距离右向车道行驶的情况下，哪个节点距离比较近就用哪个？

1 直接用Dijstra算法的javascript代码逻辑重写，实现路径判最小，但是因为js慢，每次都要重新算计算代价比较大

2 更改方案，选用Flody算法，根据校区地图最短路求解导航的项目，地图和路径，用matlab跑代码出矩阵最短路结果，然后js逻辑层就只需要查找已知地点最短路的矩阵了，或者代码重构直接计算。怎么样？

];/\*\*

\* Dijkstra算法：单源最短路径

\* 思路：

\* 1. 将顶点分为两部分：已经知道当前最短路径的顶点集合Q和无法到达顶点集合R。

\* 2. 定义一个距离数组（distance）记录源点到各顶点的距离，下标表示顶点，元素值为距离。源点（start）到自身的距离为0，源点无法到达的顶点的距离就是一个大数（比如Infinity）。

\* 3. 以距离数组中值为非Infinity的顶点V为中转跳点，假设V跳转至顶点W的距离加上顶点V至源点的距离还小于顶点W至源点的距离，那么就可以更新顶点W至源点的距离。即下面distance[V] + matrix[V][W] < distance[W]，那么distance[W] =?distance[V] + matrix[V][W]。

\* 4. 重复上一步骤，即遍历距离数组，同时无法到达顶点集合R为空。

\*

\* @param matrix 邻接矩阵，表示图

\* @param start 起点

\*

\*

\*

\* 如果求全图各顶点作为源点的全部最短路径，则遍历使用Dijkstra算法即可，不过时间复杂度就变成O(n^3)了

\* \*/

function Dijkstra(matrix, start) {

const rows = matrix.length,//rows和cols一样，其实就是顶点个数

cols = matrix.length;

if(rows !== cols || start >= rows) return new Error("邻接矩阵错误或者源点错误");

//初始化distance

const distance = new Array(rows).fill(Infinity);

distance[start] = 0;

for(let i = 0; i < rows; i++) {

//达到不了的顶点不能作为中转跳点

if(distance[i] < Infinity) {

for(let j = 0; j < cols; j++) {

//比如通过比较distance[i] + matrix[i][j]和distance[j]的大小来决定是否更新distance[j]。

if(matrix[i][j] + distance[i] < distance[j]) {

distance[j] = matrix[i][j] + distance[i];

}

}

console.log(distance);

}

}

return distance;

}

/\*\*

\* 邻接矩阵

\* 值为顶点与顶点之间边的权值，0表示无自环，一个大数表示无边(比如10000)

\* \*/

const Inf = Infinity;//没有边或者有向图中无法到达

const matrix=

W=[

[0,148,174,86,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,1144,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,1317,Inf,Inf],

[148,0,Inf,Inf,88,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[174,88,0,90,Inf,Inf,54,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[86,Inf,90,0,141,Inf,Inf,Inf,138,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,141,0,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,136,Inf],

[Inf,Inf,Inf,Inf,Inf,0,106,73,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,54,Inf,Inf,106,0,Inf,111,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,73,Inf,0,125,Inf,Inf,Inf,Inf,Inf,Inf,225,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,138,Inf,Inf,111,125,0,Inf,61,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,157,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,0,171,Inf,Inf,Inf,Inf,Inf,68,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,61,171,0,160,Inf,121,Inf,Inf,Inf,167,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,160,0,98,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,94,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,98,0,76,Inf,Inf,Inf,Inf,Inf,87,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,121,Inf,76,0,Inf,Inf,Inf,Inf,79,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[1144,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,0,56,Inf,Inf,Inf,Inf,Inf,156,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,225,Inf,Inf,Inf,Inf,Inf,Inf,56,0,85,Inf,Inf,Inf,Inf,Inf,207,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,68,Inf,Inf,Inf,Inf,Inf,85,0,96,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,167,Inf,Inf,Inf,Inf,Inf,96,0,64,Inf,Inf,Inf,Inf,147,147,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,79,Inf,Inf,Inf,64,0,83,Inf,Inf,Inf,Inf,Inf,192,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,87,Inf,Inf,Inf,Inf,Inf,83,0,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,113,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,207,Inf,Inf,Inf,Inf,0,48,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,251,Inf,785,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,156,Inf,Inf,Inf,Inf,Inf,48,0,58,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,58,0,288,245,Inf,Inf,Inf,92,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,147,Inf,Inf,Inf,Inf,288,0,48,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,245,48,0,92,Inf,Inf,Inf,119,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,192,Inf,Inf,Inf,Inf,Inf,92,0,88,Inf,Inf,Inf,138,Inf,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,88,0,75,Inf,Inf,Inf,116,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,75,0,Inf,Inf,Inf,Inf,Inf,Inf,Inf,199,Inf,68],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,92,Inf,Inf,Inf,Inf,Inf,0,213,Inf,Inf,Inf,156,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,119,Inf,Inf,Inf,213,0,115,Inf,Inf,Inf,137,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,138,Inf,Inf,Inf,115,0,74,Inf,Inf,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,116,Inf,Inf,Inf,74,0,Inf,Inf,Inf,Inf,Inf,70],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,251,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,0,120,Inf,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,156,Inf,Inf,Inf,120,0,190,Inf,Inf,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,785,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,137,Inf,Inf,Inf,190,0,1049,Inf,Inf],

[1317,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,113,Inf,Inf,Inf,Inf,Inf,Inf,Inf,199,Inf,Inf,Inf,Inf,Inf,Inf,1049,0,Inf,Inf],

[Inf,Inf,Inf,Inf,136,Inf,Inf,Inf,157,Inf,Inf,94,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,0,Inf],

[Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,Inf,68,Inf,Inf,Inf,70,Inf,Inf,Inf,Inf,Inf,0]

];

console.log(Dijkstra(matrix, 0));//[ 0, 5, 2, 7, 6 ]

alert(Dijkstra(matrix, 37));//分别到每一个点的距离