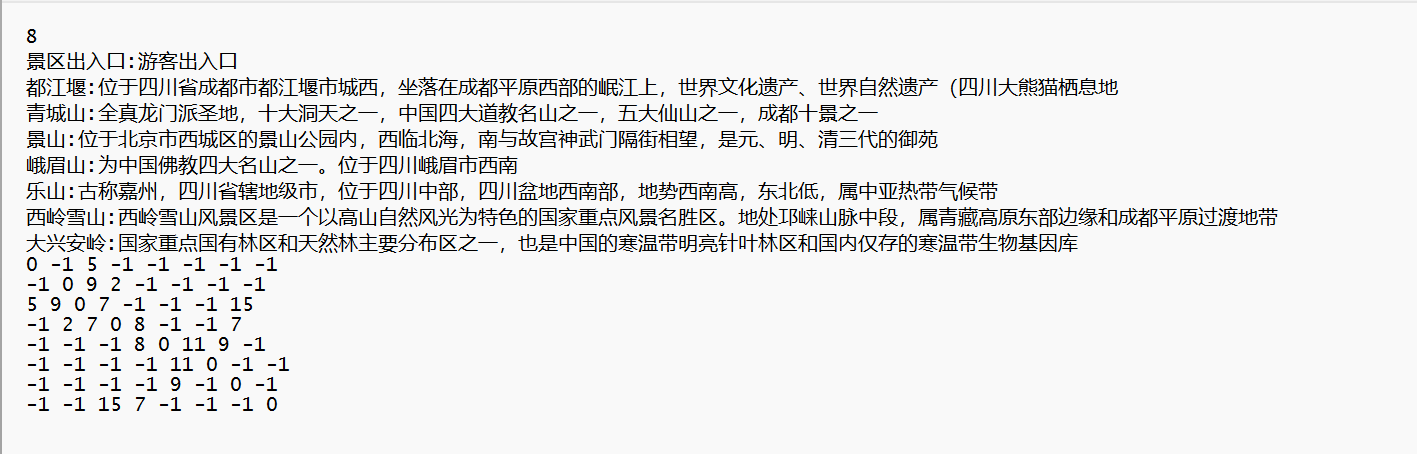
# 【题目】旅游区景点导游系统

1. **数据格式**

****

文件第一行数据表示景点数量；之后几行为景点信息和景点介绍，以冒号作为分割，每个景点占一行，最后附上矩阵作为每个景点的距离显示。

1. **数据结构（读文件创建图）**

该课程设计中采用的是邻接矩阵作为存储结构；

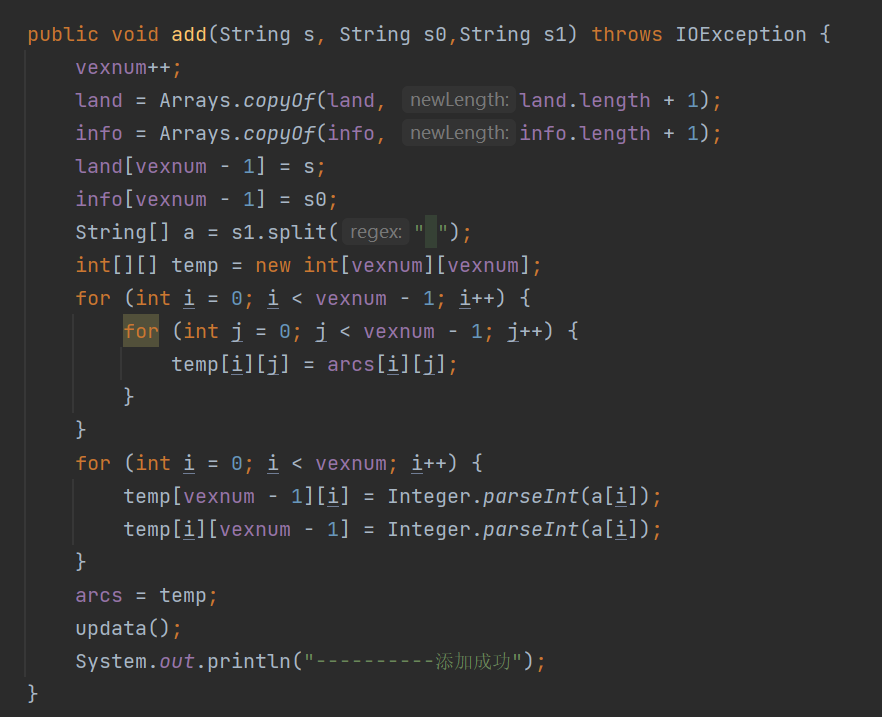
通过图类的构造函数读入文件数据，通过文件输入流读取数据，首先读得景点数量，然后通过字符串分割函数读得景点名称及介绍；最后读得邻接矩阵；

邻接矩阵定义：图的邻接矩阵是一种采用邻接矩阵数组表示顶点之间相邻关系，另一个数组存储数据元素信息的存储结构。

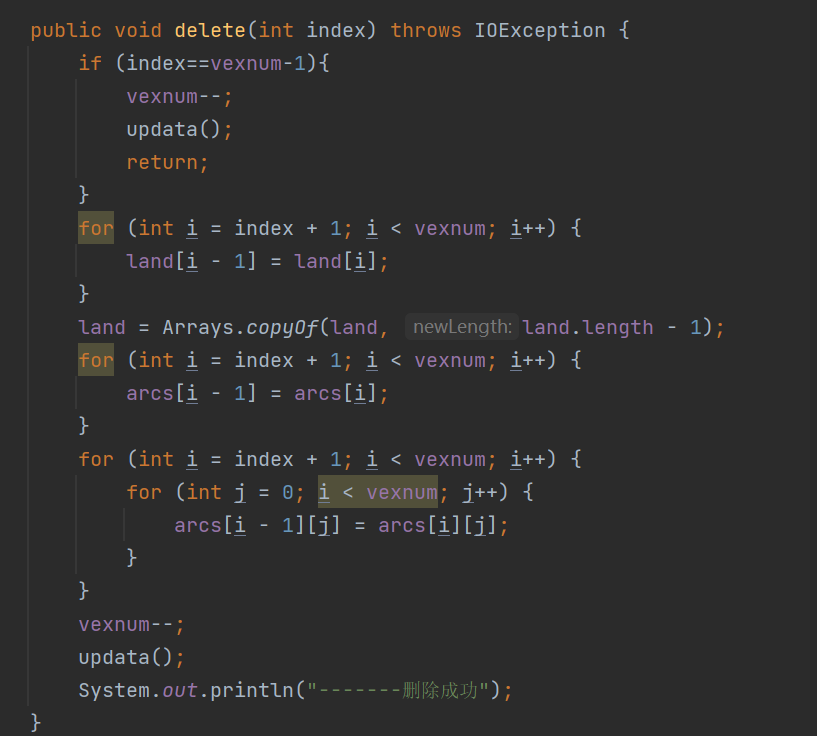
1. **查询、编辑景点信息**

首先写出数据更新后重写入文件的函数updata

因为每次打开一个文件流都会从头开始覆盖重写，所以只需要将更新后的图类数据写入文件即可实现数据的更新；其中只需注意写入景点和其介绍时要保持格式以冒号分隔。



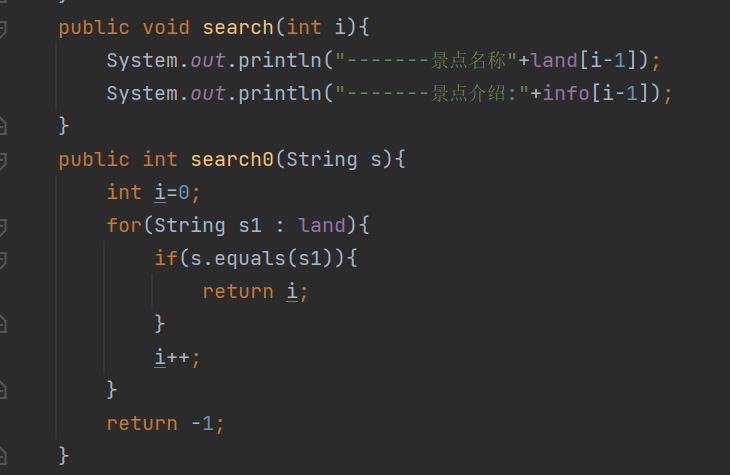
增加的方法注意的就是要更新表长，重新为数组分配空间，然后将新的数据放入增加的空间中；



同理删除也只需注意表长即可

最后调用更新函数，将新数据写入文件中；

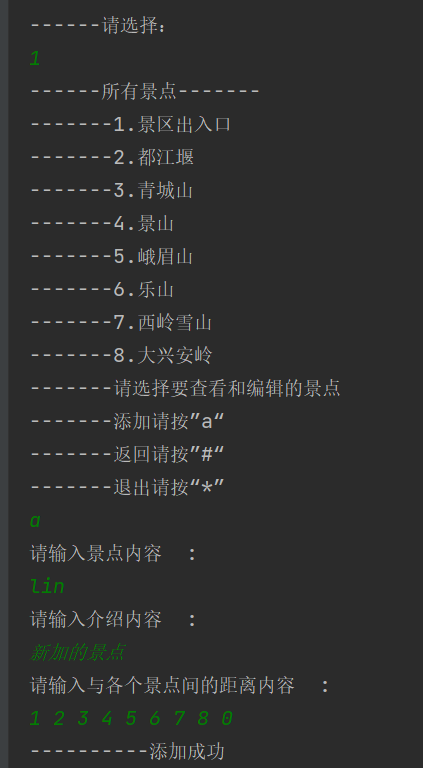
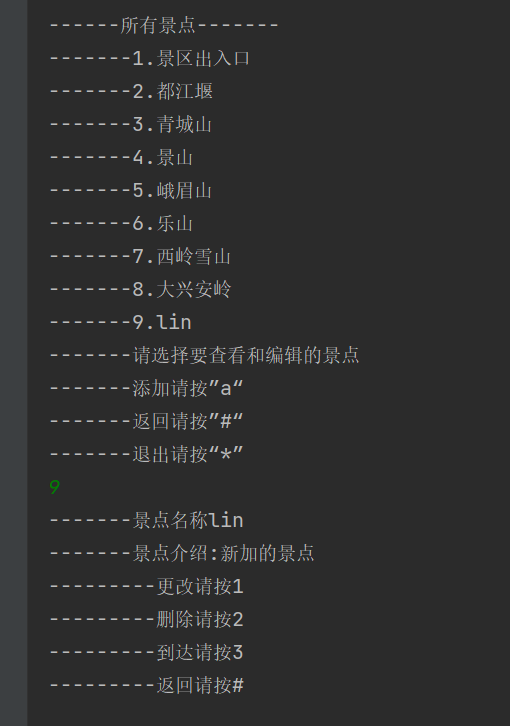
更改则直接更改对应信息数组位置的信息，调用更新函数；

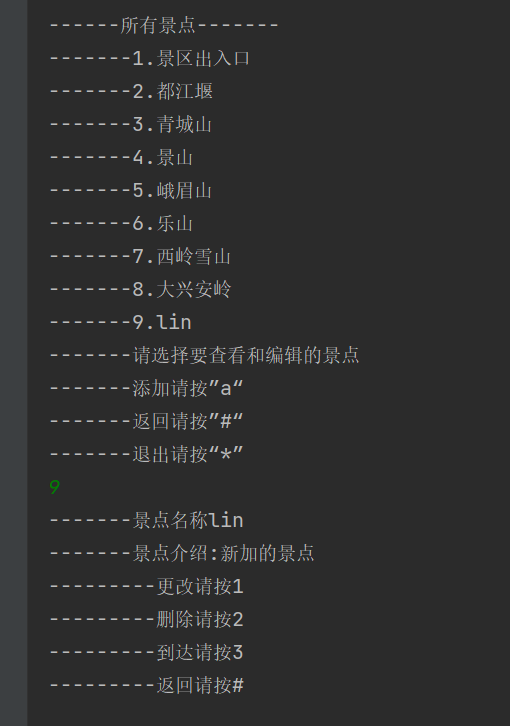


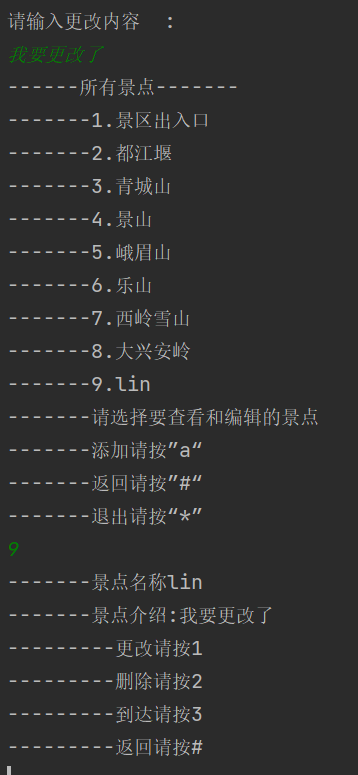
设置两个查找函数，前者通过位置给出景点及其介绍，后者则是通过景点名称返回景点位置。

程序结果展示：

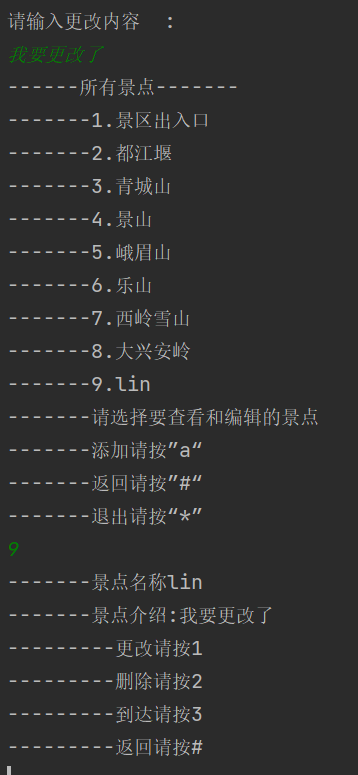
增加功能：

增加前： 增加后

更改前： 更改后：

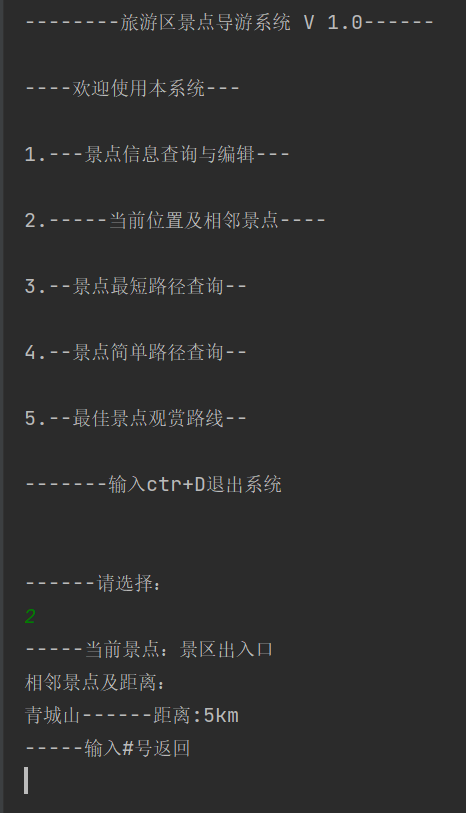


删除前： 删除后：

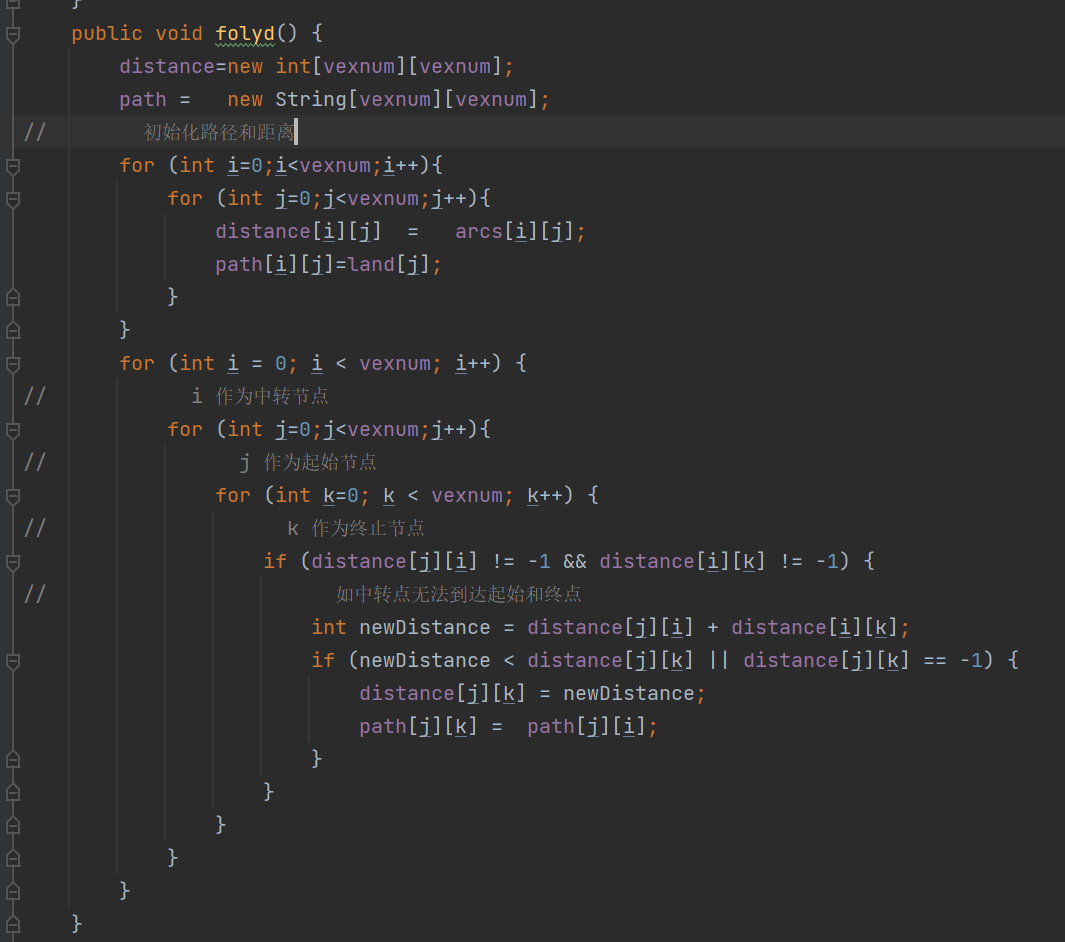


查询则是在增删前选择的一步，上述结果中有体现。

1. **旅游景点显示**



同时也可通过第一个景区信息查询更改游客当前的位置

1. **查询从每个景点出发到其他任一景点的最短简单路径及距离**

利用弗洛伊德算法更新出所有节点的互达的最短距离与路径；

Floyd算法是经典的动态规划算法，基本思想是递推产生一个矩阵序列A1,A2,.....,Ak,...,An（图有n个节点），Ak=(ak(i,j))nxn。其中矩阵Ak第i行第j列表示从顶点vi到顶点vj的路径上经过的顶点序号不大于k的最短路径长度。

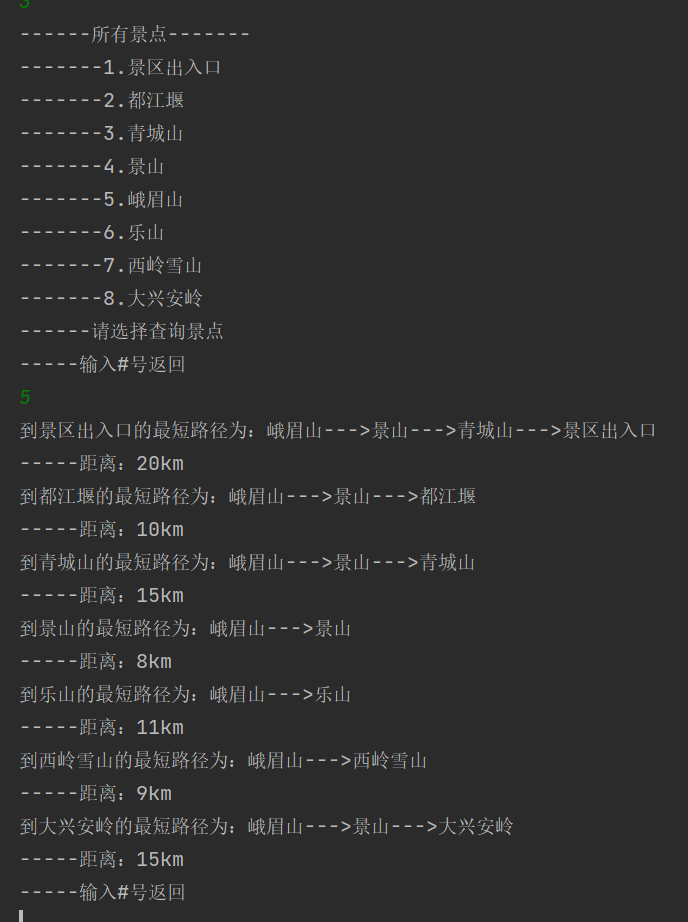
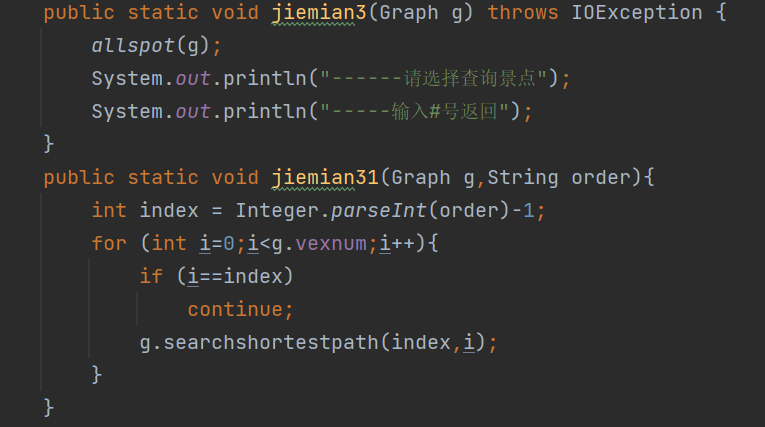
迭代公式：a_{k}(i,j)=min(a_{k-1}(i,j),a_{k-1}(i,k)+a_{k-1}(k,j))

k是迭代次数，i，j，k=1,2......n。

当最后k=n是时，An矩阵就是各个顶点之间的最短距离值了。

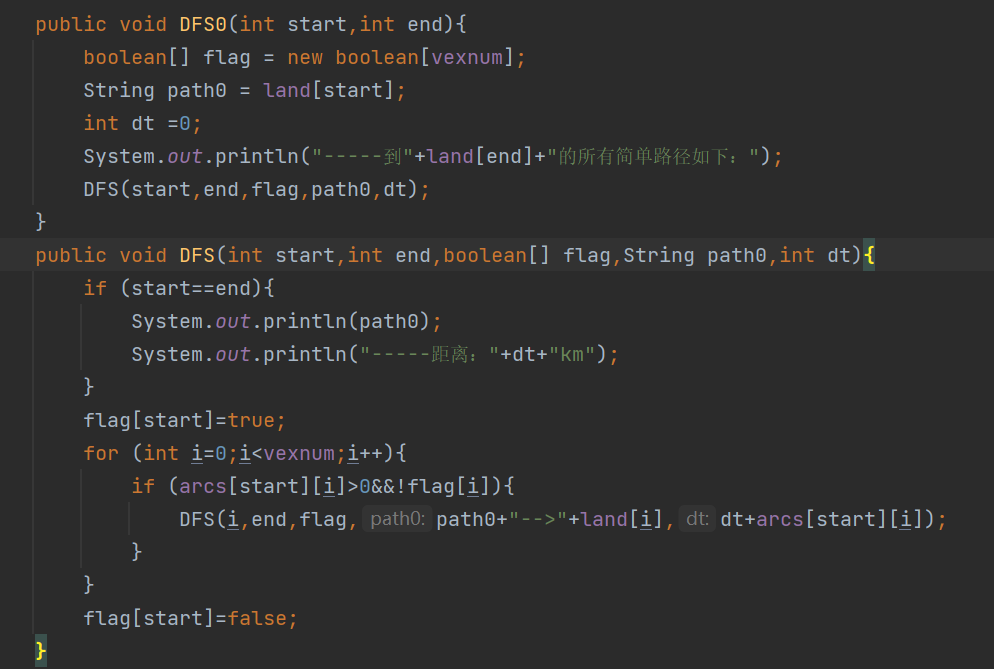
记录路径则引入路径矩阵。路径矩阵Rk=(rk(i,j))nxn,用来记录两点之间路径的前驱后继的关系，其中rk(i,j)表示从顶点vi到顶点vj的路径经过编号为rk(i,j)的顶点。所以路径矩阵里面不是数值，而是结点。由于中转节点不止一个所以每次存入路径矩阵的应该是起始点与中转点的路径矩阵值，这样就可以做到将起点到终点所有路径记录下来；

算法实现如上图所示：首先初始化距离与路径数组以方便之后记录，然后建立三重循环，最外层循环变量为中转节点，第二次为起点，第三层为终点，因为当中转节点不可达时，矩阵必不可能更新所以设置判断条件直接进入下一层循环，矩阵的更新条件则为起点到终点距离大于起点到中转点加中转点到终点距离。

当矩阵更新完成，则开始显示

1. **查询任意两个景点之间所有简单路径及距离、最短简单路径及距离**

最短简单路径则利用上一个功能解决

简单路径可知是没有重复节点的路径，可以利用DFS算法遍历图的内容，则可以得到两节点全部的简单路径。

DFS算法使用递归实现，给出起始与终止节点，终止条件为开始点与终点重合，如不终止递归则先将此次递归起点状态置为已访问，再通过循环找到起点与哪个点之间有路并且未被访问过，再进入下一层的递归；当深层递归退出时，要将该层的起点重新置为未访问，使DFS下一层的选择不会出错。另外，传入的路径与距离变量不能直接更改他的值，只能在传入下一层递归参数时加上，以保证每一层递归的路径与距离是他该有的值。

DFS算法从概念上就是先将起点压入栈中，然后找与起点相连的点再压入直到找不到可压入的点再将节点退栈，然后找其他与栈顶相连但是未访问的节点，以此类推，这样就能访问到所有节点，而在该设计中只需加入一个终点节点加以限制，就能得到两点间所有的简单路径

1. **最佳游览路线推荐**

根据贪心算法可得，当所有的局部最优解结合起来，对于大部分问题来说就是全局最优解，所以写出一组景点游览顺序后，求两两景点间的最短路径，最后组合起来，但是由于节点的可重复性，所以无法确定是从该节点的最短游览路径，因此要对从该景点出发的所有景点游览循序排列出来，再去求其最短路径及距离相互之间比较，得出最短的路径，而实现则需要用到全排列的算法得到所有的可能景点游览排列。

全排列算法可以说是一个回溯算法：这个问题可以看作有 n 个排列成一行的空格，我们需要从左往右依此填入给定的 n 个景点，每个景点只能使用一次。那么很直接的可以想到一种穷举的算法，即从左往右每一个位置都依此尝试填入一个景点，看能不能填完这 n 个空格，在程序中可以用回溯法来模拟这个过程。定义递归函数表示从左往右填到start个位置，并记录当前排列为 那么整个递归函数分为两个情况：

如果start=n，说明已经填完了，则记录当前排列顺序结束递归

如果 start<n，要考虑这第 start 个位置我们要填哪个数。根据要求肯定不能

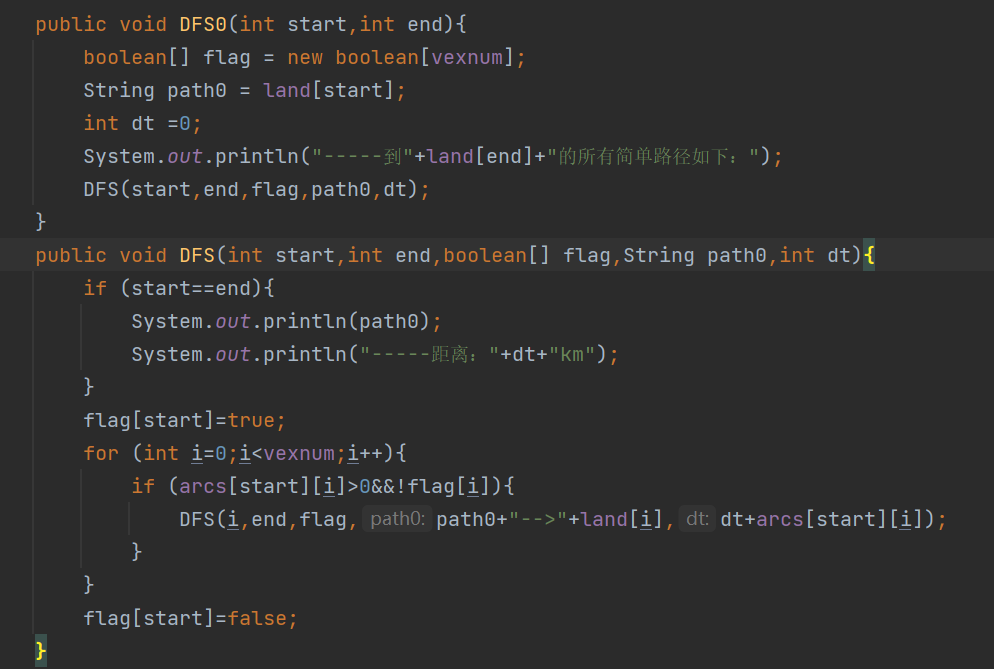
填已经填过的数，具体来说，假设我们已经填到第start个位置，那么给出的

排序数组从0-start-1是已填过的数的集合，而start-n-1是是待填的数，设待

填的数下标为i，那么填完以后要将第i个与第start个数交换，这样在填第

Start+1个数的时候，能做到0-start部分为已填的数，之后为未填的数，而回

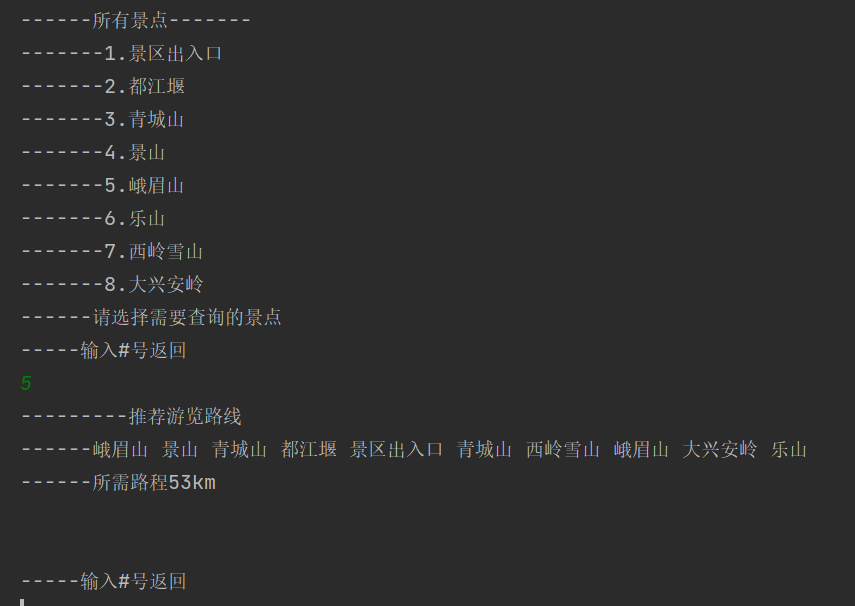
溯时交换回来即能完成撤销操作。

算法实现如下图所示：

这样可以得到所有节点的全排列，然后根据需要的节点查找相应的以该起点

开头的排列情况，通过比较得出最佳游览路线

运行结果如下：



1. **设计总结**

首先在设计之初卡死在界面上，不知道界面应该设计才能一级菜单套一级菜单，后来发现标签循环可以解决这个问题，而且将输入，界面，结构，main分开来写实在是很好用，第二次栽跟头在递归函数的传参数上，忘记了要在每层递归上保持当前参数的统一性，不能退出上一层递归后，参数后还是上一层的，不会恢复原样，，还有在记录全排列结果时，对数组传递不太清楚，直接赋予了地址，结果最后一堆加入arrylist的数组全是一种排列结果。总体设计分为了四个类，主要功能实现写在了图类，界面写在home类，同时也用了个专门判断输入正确性的类。