项目仓库地址：

gitee：https://gitee.com/li-_-zan/ds-design/tree/master

github：https://github.com/li-zan/ds-design

旅游区景点导游系统

1、数据格式

数据存储在文件 data.txt 中，数据存储格式如下

```
7
12
游客接待中心
本园入口和出口，欢迎大家!
鸣禽馆
展馆整体为庭院式风格，展馆与外廊连体,分前、后两部分
狮虎山
狮虎山建于1955年，是我园基本保持原貌的最古老展馆。
猴山
这里放养了以猕猴、长尾猴为主的猴类100多只，经过多年
熊猫馆
熊猫馆饲养有大熊猫。馆内模拟大熊猫栖息地生态环境，
水族馆
我园水族馆是目前世界上唯一一个设有中国展区，并对长
熊山
由黑熊和白熊两个下沉式露天馆舍组成，白熊山位于东侧
游客接待中心 猴山 80
游客接待中心 狮虎山 50
游客接待中心 鸣禽馆 30
鸣禽馆 熊猫馆 100
鸣禽馆 狮虎山 60
狮虎山 水族馆 90
狮虎山 猴山 150
狮虎山 熊猫馆 120
猴山 熊山 200
猴山 水族馆 40
熊猫馆 水族馆 180
水族馆 熊山 250
```

- 结点个数
- 边的条数
- 结点 1 名字
- 结点 1 信息
- 结点 2 名字
- 结点 2 信息
- ...
- ...
- ...
- ...
- ...
- ...
- 边端点 1　边端点 2　对应边的权值
- ...
- ...
- ...
- ...
- ...
- ...

2、数据结构（读文件创建图）

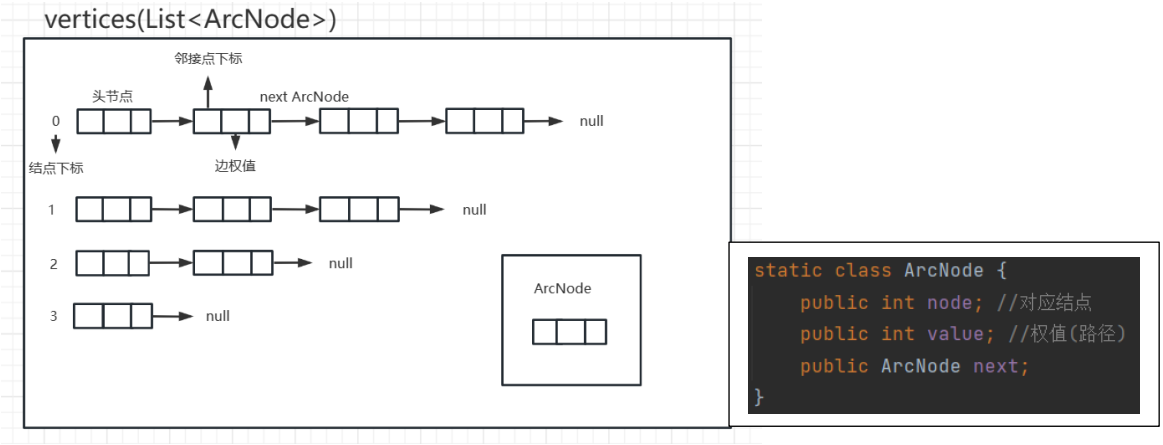
通过 Java I/O 操作将文件中的数据读入内存，选择邻接表作为存储结构，建立 图（数据结构）的模型。图的数据结构和邻接表的存储模型见下：

图的数据结构

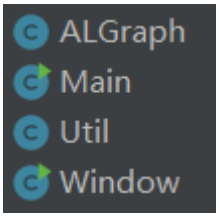
```java
public class ALGraph {
    public int vexNum; //结点数
    public int arcNum; //边数
    public List<String> vexes; //结点集（下标与结点名对应,以下将用下标操作）
    public List<String> info;  //结点信息集（下标与结点集的结点对应）
    public List<ArcNode> vertices; //邻接表

    public String path;

    static class ArcNode {
        public int node; //对应结点
        public int value; //权值(路径)
        public ArcNode next;
    }
```

注:path 非邻接图所必须，在此作为成员变量存储文件路径为了方便其他操作(后续存储)

邻接表存储模型



3、程序结构设计



ALGraph 图类定义图的结构和相关操作

Window 窗体类定义独立窗体样式接口，不只限于接收特定图的结构数据，从而实现松耦合。

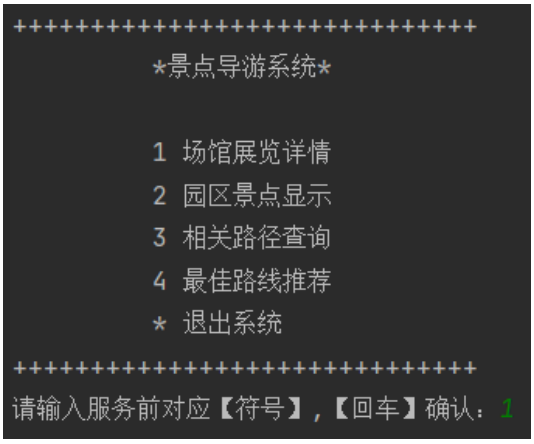
Util 工具类提供用户输入，校验用户输入，处理输入异常

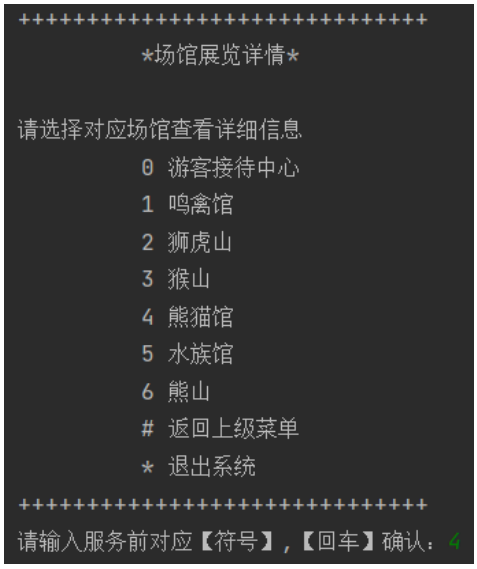
Main 类操作图和窗体类并配合工具类的相关校验 编写该项目的逻辑代码

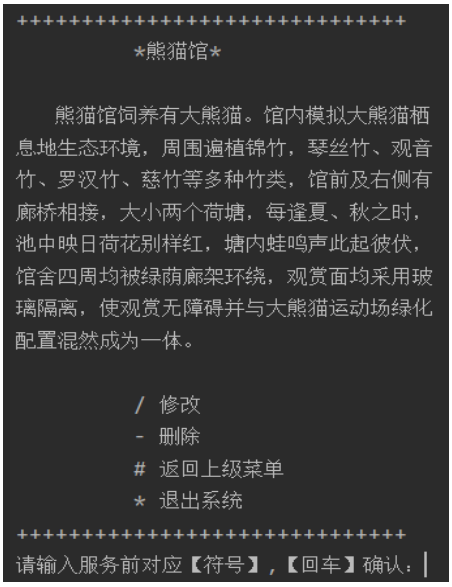
项目源码方法功能参考方法文档注释

4、查询、编辑景点信息

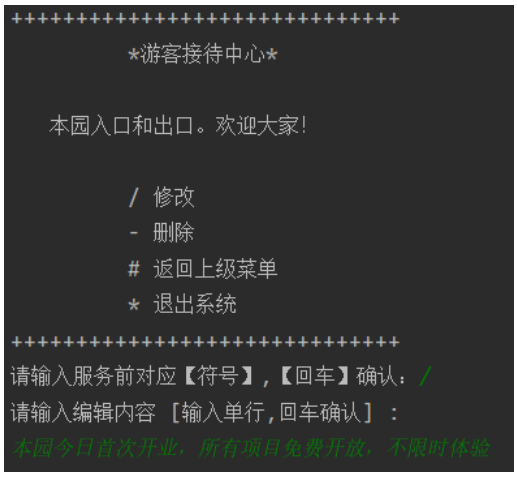
查询景点信息



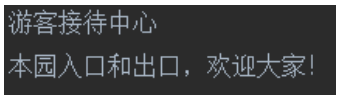


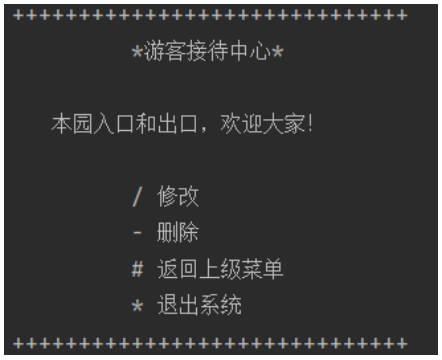


当景点信息内容不为空时，可键入“/”**修改**景点信息

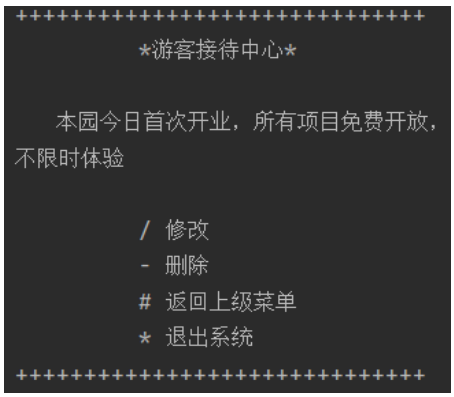


修改前 data 文件和查询显示如下

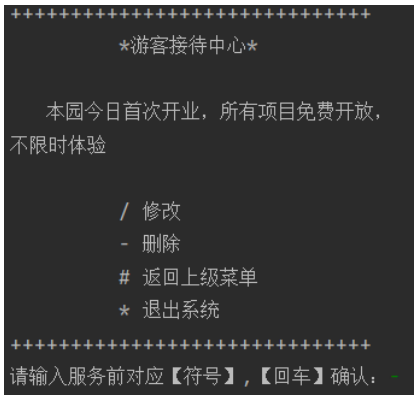


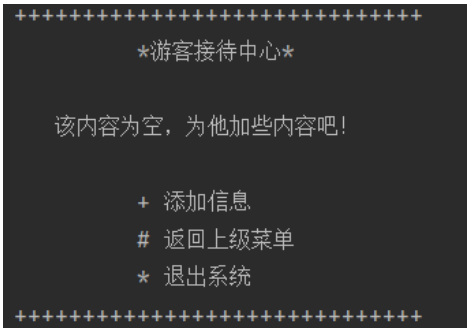


修改后 data 文件和查询显示如下

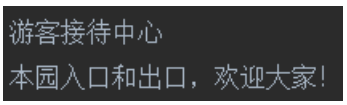


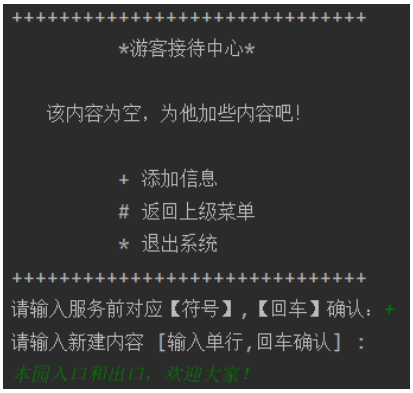
当景点信息**内容不为空**时，可键入“-”**删除**景点信息，同时对**空内容检验处理**
data 文件和查询显示变化如下

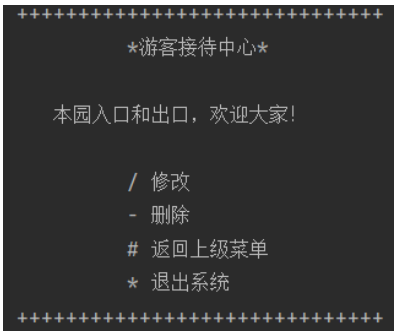




当景点信息**内容为空**时，可键入“+”**添加**景点信息。
data 文件和查询显示变化如下

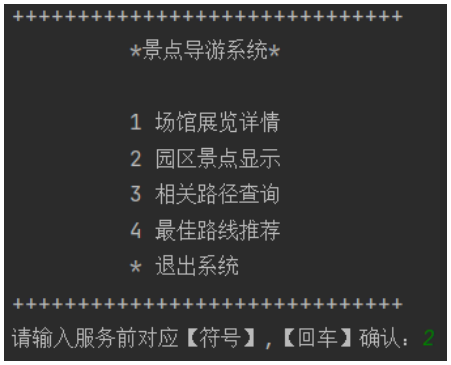


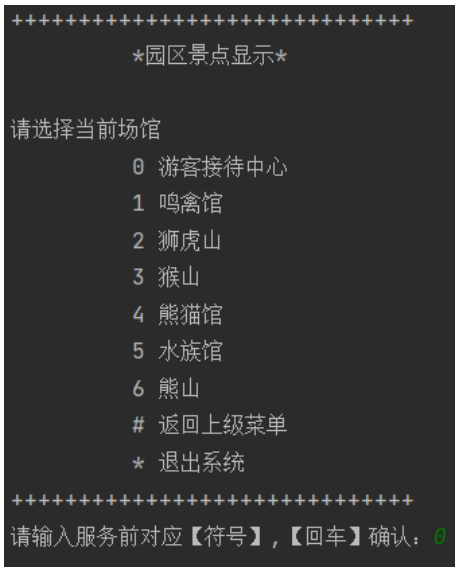


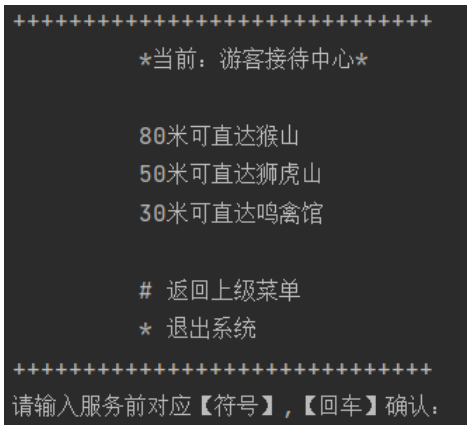


5、旅游区景点显示

显示游客当前所在景点及所有与游客所在景点相邻景点信息。







6、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

用户输入任一景点，输出旅游区每个景点到该景点的最短简单路径及距离。

使用 Floyd 算法求**多源最短路径**

1) 建立一张表格，记录每个顶点直达其各个顶点的权值
2) 在该表的基础上，将顶点 1 作为 "中间顶点"，计算从各个顶点出发途径顶点 1 再到达其它顶点的权值，如果比表中记录的权值更小，证明两个顶点之间存在更短的路径，对表进行更新。
3) 在表的基础上，再以顶点 2 作为 "中间顶点"，计算从各个顶点出发途径顶点 2 再到达其它顶点的权值。
4) 通过将所有的顶点分别作为“中间顶点”，最终得到的表就记录了各个顶点之间的最短路径。

```
// 遍历每个顶点，将其作为其它顶点之间的中间顶点，更新 graph 数组
for (k = 0; k < V; k++) {
    for (i = 0; i < V; i++) {
        for (j = 0; j < V; j++) {
            // 如果新的路径比之前记录的更短，则更新 graph 数组
            if (graph[i][k] + graph[k][j] < graph[i][j]) {
                graph[i][j] = graph[i][k] + graph[k][j];
```

总结：Floyd 算法采用动态规划，状态转移方程 dp[i][j]＝min{ dp[i][k]+dp[k][j], dp[i][j]}

关于最短**路径的记录**：

1) 初始化二维数组，如果任何一对节点的最短路径发生变化，它将跟踪中转结点，下一个要指向的节点。最初，任何两个节点 i 和 j 之间的最短路径是 j（即从 i -> j 的直接边）。

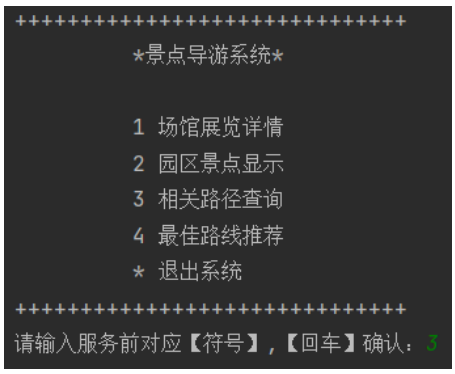
2) 如果路径存在于两个结点之间，则 path[i][j] = j 否则我们设置 path[i][j] = -1

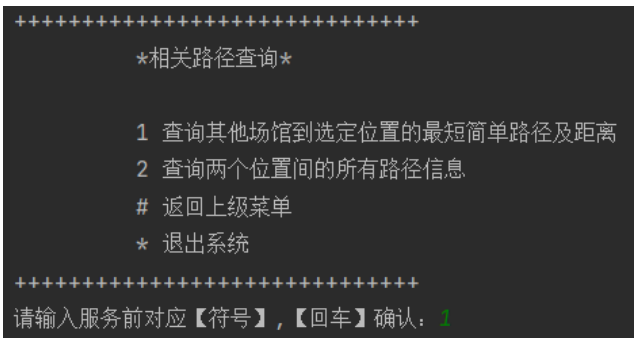
3) 在 Floyd 算法的 if 条件中，我们将添加一个语句 path[i][j] = path[i][k]（这意味着我们找到了 i， j 之间通过中间节点 k 的最短路径，需要在 i，j 记录 i 到中转结点 k 的最短路径即记录 path[i][k]）。

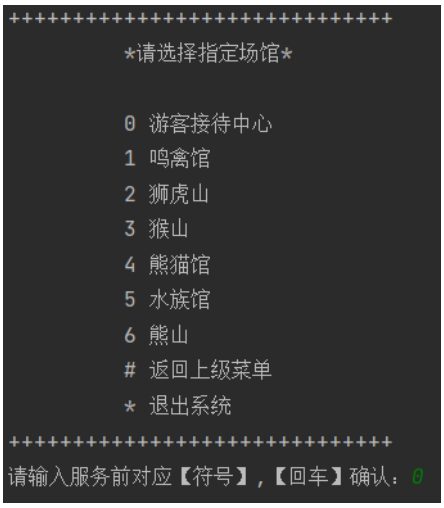
4) 输出最短路径 i->j 我们只需开始循环遍历节点 i，同时将其值更新为 path[i][j]，直到到达节点 j。

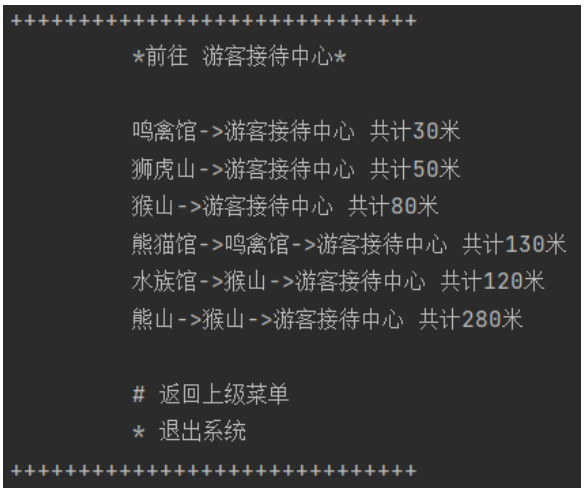
结合本题目的具体代码细节请查阅项目源码

程序运行效果截图：









7、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

用户输入任意两个景点，输出两个景点间所有简单路径及距离、最短简单路径及距离。

dfs 遍历结点记录所有简单路径：

首先 dfs 结束条件是当前结点已经到达了用户指定的目标结点，此时，我们要将当前的这条路径和权值记录在答案集合中，同时结束结点继续 dfs。

```
if (begin == end) {
    simplePaths.add(path);
    values.add(value);
    return;
}
```

如果当前结点不是目标结点，应该在标志访问数组中标记当前结点已经访问，防止重复。

```
isVisit[begin] = true;
```

接下来更新路径 结点可以从当前结点走到其所有邻接结点，同时为了防止重复经过结点，所以访问的结点要保证在该路径没有被访问过[主要涉及链表的遍历操作]

```
while (p.next != null) { //遍历链表
    if (! isVisit[p.next.node]) {
        String strSeg = "->" + vexes.get(p.next.node);
        int num = p.next.value;
        dfs1(p.next.node, end,  path: path+strSeg,  value: value+num, simplePaths, values, isVisit);
    }
    p = p.next;
}
```

最后在 dfs 回溯时，即完成了从回溯结点开始的所有路径的尝试，要将该结点标志重置，开始新的路径

```
isVisit[begin] = false;
```

参数说明已在相应方法文档注释，具体细节请参阅项目源码

最短路径：

这里求最短路径只需在当下已经求出的所有简单路径中比较查询最小权值的路径。

程序运行效果截图：

```
++++++++++++++++++++++++++++++++
          *景点导游系统*

          1 场馆展览详情
          2 园区景点显示
          3 相关路径查询
          4 最佳路线推荐
          * 退出系统
++++++++++++++++++++++++++++++++
请输入服务前对应【符号】,【回车】确认：3
```

```
++++++++++++++++++++++++++++++++
          *相关路径查询*

          1 查询其他场馆到选定位置的最短简单路径及距离
          2 查询两个位置间的所有路径信息
          # 返回上级菜单
          * 退出系统
++++++++++++++++++++++++++++++++
请输入服务前对应【符号】,【回车】确认：2
```

```
++++++++++++++++++++++++++++++++
          *请选择指定场馆*

          0 游客接待中心
          1 鸣禽馆
          2 狮虎山
          3 猴山
          4 熊猫馆
          5 水族馆
          6 熊山
          # 返回上级菜单
          * 退出系统
++++++++++++++++++++++++++++++++
          选择第一个位置
请输入服务前对应【符号】,【回车】确认：1
          选择第二个位置
请输入服务前对应【符号】,【回车】确认：2
```

```
++++++++++++++++++++++++++++++++
        *鸣禽馆 - 狮虎山*

        所有简单路径信息：
        鸣禽馆->熊猫馆->水族馆->熊山->猴山->狮虎山  共计880米
        鸣禽馆->熊猫馆->水族馆->熊山->猴山->游客接待中心->狮虎山  共计860米
        鸣禽馆->熊猫馆->水族馆->猴山->狮虎山  共计470米
        鸣禽馆->熊猫馆->水族馆->猴山->游客接待中心->狮虎山  共计450米
        鸣禽馆->熊猫馆->水族馆->狮虎山  共计370米
        鸣禽馆->熊猫馆->狮虎山  共计220米
        鸣禽馆->狮虎山  共计60米
        鸣禽馆->游客接待中心->猴山->熊山->水族馆->熊猫馆->狮虎山  共计860米
        鸣禽馆->游客接待中心->猴山->熊山->水族馆->狮虎山  共计650米
        鸣禽馆->游客接待中心->猴山->水族馆->熊猫馆->狮虎山  共计450米
        鸣禽馆->游客接待中心->猴山->水族馆->狮虎山  共计240米
        鸣禽馆->游客接待中心->猴山->狮虎山  共计260米
        鸣禽馆->游客接待中心->狮虎山  共计80米

        最短简单路径：
        鸣禽馆->狮虎山  共计60米

        # 返回上级菜单
        * 退出系统
++++++++++++++++++++++++++++++++
```

8、最佳游览路线推荐

输入某一景点，输出从该景点出发经过景区所有景点（景点可以重复）且距离最短的游览路线。

主要思想是生成所有可能的路径并获得权值最小的路径。

首先，将生成节点的所有可能的排列。每个排列将表示图中访问节点的顺序。路径的权值将等于每两个连续节点之间所有最短路径的总和。

这里使用在上面实现 Floyd 算法的方法获得所有结点间的最短路径信息

最后，访问图中所有结点的最短路径就是所有可能的最短顺序路径中具有最小权值的路径。

对于全排列可采用字典排序的策略：

1 2 3 的全排列如下：1 2 3 ， 1 3 2 ， 2 1 3 ， 2 3 1 ， 3 1 2 ， 3 2 1

我们这里是通过字典序法找出来的。

字典排序：

1. **从右至左找第一个本身比其右临小的数(即为需要改变的位置也是下次排列的固定位置)，记下位置 i，值 value_a**
2. **从右边往左找第一个大于 value_a 的数，记下位置 j，value_b**
3. **交换 value_a 和 value_b 的值**
4. **将 i 以后的元素重新按从小到大的顺序排列[或者相当于把之后的逆序]**

主要代码如下：

permutationValue 方法可求出 permutation 这个排列的最小访问路径权值

在循环比较之后，最小路径的排列和权值会分别存储在 shortPermu 和 value 中

```
int value = permutationValue(permutation, valueFloyd);
int[] shortPermu = Arrays.copyOf(permutation, permutation.length);
while (true) {  //枚举全排列
    int i, j, valueTemp;
    for (i = vexNum-2; i > 0 && permutation[i] >= permutation[i+1]; --i) ;
    if (i <= 0) break;
    for (j = vexNum-1; permutation[j] <= permutation[i]; --j) ;
    permutation[i] = permutation[i] ^ permutation[j];
    permutation[j] = permutation[i] ^ permutation[j];
    permutation[i] = permutation[i] ^ permutation[j];
    Arrays.sort(permutation,  fromIndex: i+1, vexNum);
    valueTemp = permutationValue(permutation, valueFloyd);
    if (valueTemp < value) {
        value = valueTemp;
        shortPermu = Arrays.copyOf(permutation, permutation.length);
    }
}
```

关于最短路径的输出即为 shortPermu 排列中两两之间的最小路径组合，两两之间的最小路径即为上述 Floyd 路径记录的输出

具体细节请查阅程序源码

程序运行效果截图：

```
++++++++++++++++++++++++++++++++
          *景点导游系统*

          1 场馆展览详情
          2 园区景点显示
          3 相关路径查询
          4 最佳路线推荐
          * 退出系统
++++++++++++++++++++++++++++++++
请输入服务前对应【符号】,【回车】确认：4
```

```
++++++++++++++++++++++++++++++++
          *最佳路线推荐*

请选择指定场馆
将为您匹配最佳游览路线
          0 游客接待中心
          1 鸣禽馆
          2 狮虎山
          3 猴山
          4 熊猫馆
          5 水族馆
          6 熊山
          # 返回上级菜单
          * 退出系统
++++++++++++++++++++++++++++++++
请输入服务前对应【符号】,【回车】确认：3
```

```
++++++++++++++++++++++++++++++++
          *最佳路线推荐*

          猴山->游客接待中心->鸣禽馆->熊猫馆->狮虎山->水族馆->猴山->熊山
          全长660米
          带你花最少的时间体验最多的欢乐

          # 返回上级菜单
          * 退出系统
++++++++++++++++++++++++++++++++
```

9、设计总结

本次程序设计本着高内聚低耦合的设计理念，抽象出图类，窗体类，工具类，且设计彼此数据交互不依赖特定数据结构。本程序设计对于用户输入部分和校验处理做了单独封装，并重载了多种校验输入方法，以提高程序代码的健壮性。输入校验策略举例：对用户 dos 命令行输入文档结束信号(Windows Ctrl+Z; Unix Ctrl+D)的处理，若不对输入流开启情况校验而是直接读入数据的话程序会抛出异常，所以在每次输入前先对输入流校验，检测到输入流关闭后不再执行读取操作转而保存数据正常结束程序。