# PJ3 实验文档

程茜　　19302010084

## 一. 最短路径算法实现

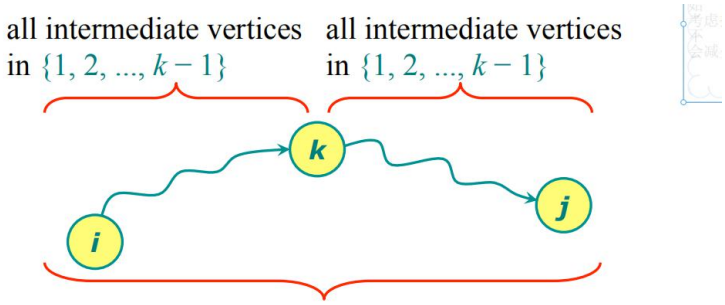
（1）采用的 **Floyd-Warshall** 算法，输入为一个邻接矩阵和一个保存各个节点的数组，用于路径的输出。算法用到了两个矩阵，一个矩阵（**matrix**）记录从 i 到 j 的最短路径长度，还有一个辅助矩阵（**path**）记录从 i 到 j 的最短路径中 j 的前一个节点。

（2）邻接矩阵 matrix 中若两点没有路径则值为无穷大，即 **Integer.MAX_VALUE**，定义 natrix[i][i]=0。然后首先对辅助矩阵 path 初始化为 path[i][j]=i。

（3）算法主体如下：

```
public static void floyd(int[][]matrix){
    for(int k = 0;k<matrix.length;k++){
        for(int i = 0;i<matrix.length;i++){
            for(int j = 0;j<matrix.length;j++){
                if((matrix[i][k]!=Integer.MAX_VALUE)&&(matrix[k][j]!=Integer.MAX_VALUE)&&matrix[i][j]>(matrix[i][k]+matrix[k][j])){
                    matrix[i][j] = matrix[i][k]+matrix[k][j];
                    path[i][j] = path[k][j];
                }
            }
        }
    }
}
```

算法的思路是在一条目前已知的从 i 到 j 的最短路径中考虑加不加入点 k，如果加入点 k 后路径变短则加入，反之不加入，算法的时间复杂度为 $O(n^3)$




$p$: all intermediate vertices in $\{1, 2, ..., k\}$

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{if } k = 0, \\ \min(d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}) & \text{if } k \geq 1. \end{cases}$$

$$d_{ij}^{(m)} = \min_k \{d_{ik}^{(m-1)} + a_{kj}\}.$$

## 二. 最短路径打印

最短路径的打印需要借助辅助矩阵，如下图所示。如果从 i 到 j 的最短路径值为无穷大，则说明没有相应路径，则输出相关内容。反之，要输出从 i 到 j 的最短路径，只需要从 path[i][j] 的值开始，一直到 path[i][m]=i，找到的即为最短路径，代码如下：


**Node 4**

$Path_{13} = 1 \rightarrow 2 \rightarrow 4 \rightarrow 3$

$d_{13}^{(4)} = 3 + 1 + -5 = -1$

$$\Pi^{(4)} = \begin{pmatrix} \varnothing & 1 & 4 & 2 & 1 \\ 4 & \varnothing & 4 & 2 & 4 \\ 4 & 3 & \varnothing & 2 & 4 \\ 4 & 3 & 4 & \varnothing & 1 \\ 4 & 4 & 4 & 5 & \varnothing \end{pmatrix}$$

```java
//打印路径
public static String printPath(int[][]matrix,int start,int end){
    floyd(matrix);

    if(matrix[start][end]==Integer.MAX_VALUE){
        return "没有相应的路径";
    }
    else {
        StringBuilder a = new StringBuilder();
        String b = "从" + locationName[start] + "到" + locationName[end] + "最短用时为" + matrix[start][end]+"min, 路径为: ";
        a.append(b);
        int i = start, j = end;
        a.append(locationName[j]+" <-- ");
        while (path[i][j] != i) {
            j = path[i][j];
            a.append(locationName[j]+" <-- ");
        }
        a.append(locationName[i]);

        String result = a.toString();
        return result;
    }
}
```

利用了 StringBuider 类来逐个几率路径上经过的点，最后返回一个 String 字符串。

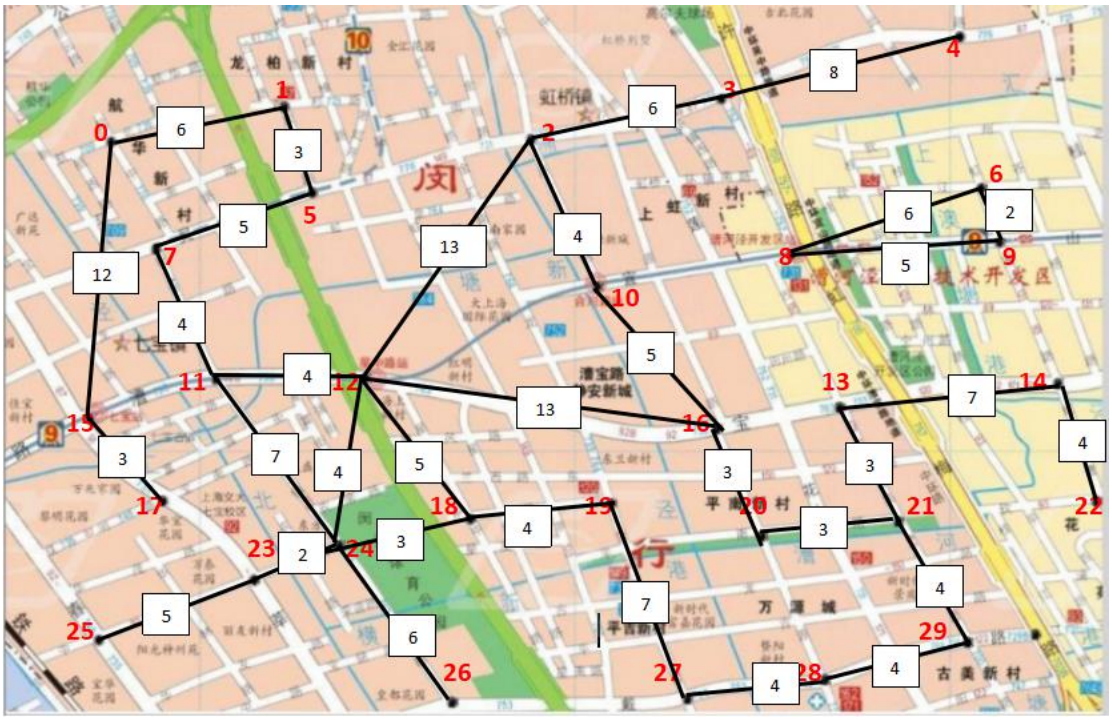
输出示例：

从0到5最短用时为9min, 路径为: 5 <-- 1 <-- 0

从0到29最短用时为42min, 路径为: 29 <-- 27 <-- 19 <-- 18 <-- 12 <-- 11 <-- 7 <-- 5 <-- 1 <-- 0

**三．地图选择：**

从图中选取了 30 个点，标号为 0-29，用红色标出，黑色直线代表两点可直接到达，上面的数字是步行用时。

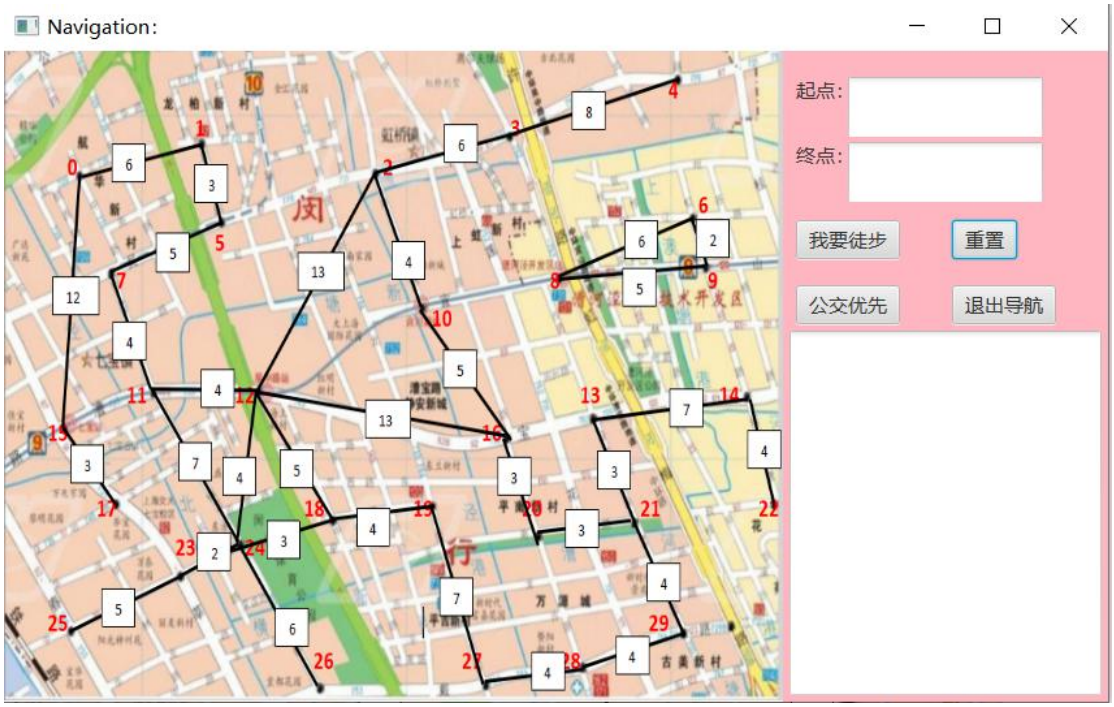


根据地图得到的邻接矩阵为：

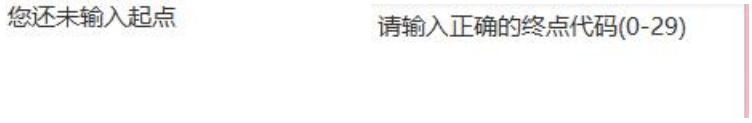
```
final int X = Integer.MAX_VALUE;
int [][] matrixDefalt = {
        {0,6,X,X,X,X,X,X,X,X,X,X,X,12,X,X,X,X,X,X,X,X,X,X,X,X,X},  //0
        {6,0,X,X,X,3,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X},   //1
        {X,X,0,6,X,X,X,X,X,X,4,X,13,X,X,X,X,X,X,X,X,X,X,X,X,X,X},  //2
        {X,X,6,0,8,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X},//3
        {X,X,X,8,0,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X},//4
        {X,3,X,X,X,0,X,5,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X},//5
        {X,X,X,X,X,X,0,X,6,8,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X},//6
        {X,X,X,X,X,5,X,0,X,X,X,4,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X},//7
        {X,X,X,X,X,X,6,X,0,5,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X},//8
        {X,X,X,X,X,X,2,X,6,0,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X},//9
        {X,X,4,X,X,X,X,X,X,X,0,X,X,X,X,X,5,X,X,X,X,X,X,X,X,X,X},//10
        {X,X,X,X,X,X,X,4,X,X,X,0,4,X,X,X,X,X,X,X,X,X,X,7,X,X,X},//11
        {X,X,13,X,X,X,X,X,X,X,X,4,0,X,X,X,13,X,5,X,X,X,X,X,4,X,X,X,X},//12
        {X,X,X,X,X,X,X,X,X,X,X,X,X,0,7,X,X,X,X,X,X,3,X,X,X,X,X,X,X},//13
        {X,X,X,X,X,X,X,X,X,X,X,X,X,7,0,X,X,X,X,X,X,4,X,X,X,X,X,X,X},//14
        {12,X,X,X,X,X,X,X,X,X,X,X,X,X,0,X,3,X,X,X,X,X,X,X,X,X,X,X},//15
        {X,X,X,X,X,X,X,X,X,X,5,X,13,X,X,X,0,X,X,X,3,X,X,X,X,X,X,X,X},//16
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,3,X,0,X,X,X,X,X,X,X,X,X,X,X},//17
        {X,X,X,X,X,X,X,X,X,X,X,X,5,X,X,X,X,X,0,4,X,X,X,X,3,X,X,X,X},//18
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,4,0,X,X,X,X,X,X,7,X,X},//19
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,3,X,X,X,0,3,X,X,X,X,X,X,X},//20
        {X,X,X,X,X,X,X,X,X,X,X,X,3,X,X,X,X,X,X,3,0,X,X,X,X,X,X,4},//21
        {X,X,X,X,X,X,X,X,X,X,X,X,X,4,X,X,X,X,X,X,X,0,X,X,X,X,X,X},//22
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,0,2,5,X,X,X},//23
        {X,X,X,X,X,X,X,X,X,X,X,7,4,X,X,X,X,X,3,X,X,X,X,2,0,X,6,X,X},//24
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,5,X,0,X,X,X},//25
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,6,X,0,X,X},//26
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,7,X,X,X,X,X,X,X,0,4,4},//27
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,7,X,X,X,X,X,X,X,4,0,4},//28
        {X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,X,4,X,X,X,X,X,X,4,X} //29
```

## 四．界面实现

利用 javafx 实现了简单的 UI 界面，左边是地图，右边是按钮和输入输出框，用户可以输入起点和终点的对应代码，然后点击我要徒步按钮，就会在下方的输出框中输出最短路径。




如果用户没有输入起点终点或者输入的代码不对，都会在下方的输出框里提示相应内容：



当用户点击重置时，会清空所有输入框里面的内容，点击退出导航则可以退出程序。
示例：