实验三：最短路（Map Routing）

张俊华 16030199025

一、实验内容

实现经典的 Dijkstra 最短路径算法，并对其进行优化。

地图。 本次实验对象是图 maps 或 graphs，其中顶点为平面上的点，这些点由权值为欧氏距离 的边相连成图。可将顶点视为城市，将边视为相连的道路。

二、实验环境

IntelliJ IDEA 2018.2.5 (Ultimate Edition)

JRE: 1.8.0_152-release-1248-b19 amd64

JVM: OpenJDK 64-Bit Server VM by JetBrains s.r.o

Windows 10 10.0

三、实验步骤

1. Dijkstra 基本算法

在 `dijkstra.java` 文件中，对 Dijkstra 算法进行了基本实现。

其核心函数为

```
private void dijkstra(int s, int d)
```

实现了以 s 为起点，到其余各点的最短路径

```
private void dijkstra(int s, int d) {
        int V = G.V();

        // initialize
        dist = new double[V];
        pred = new int[V];
        for (int v = 0; v < V; v++) dist[v] = INFINITY;
        for (int v = 0; v < V; v++) pred[v] = -1;

        // priority queue
        IndexPQ pq = new IndexPQ(V);
        for (int v = 0; v < V; v++) pq.insert(v, dist[v]);

        // set distance of source
        dist[s] = 0.0;
```

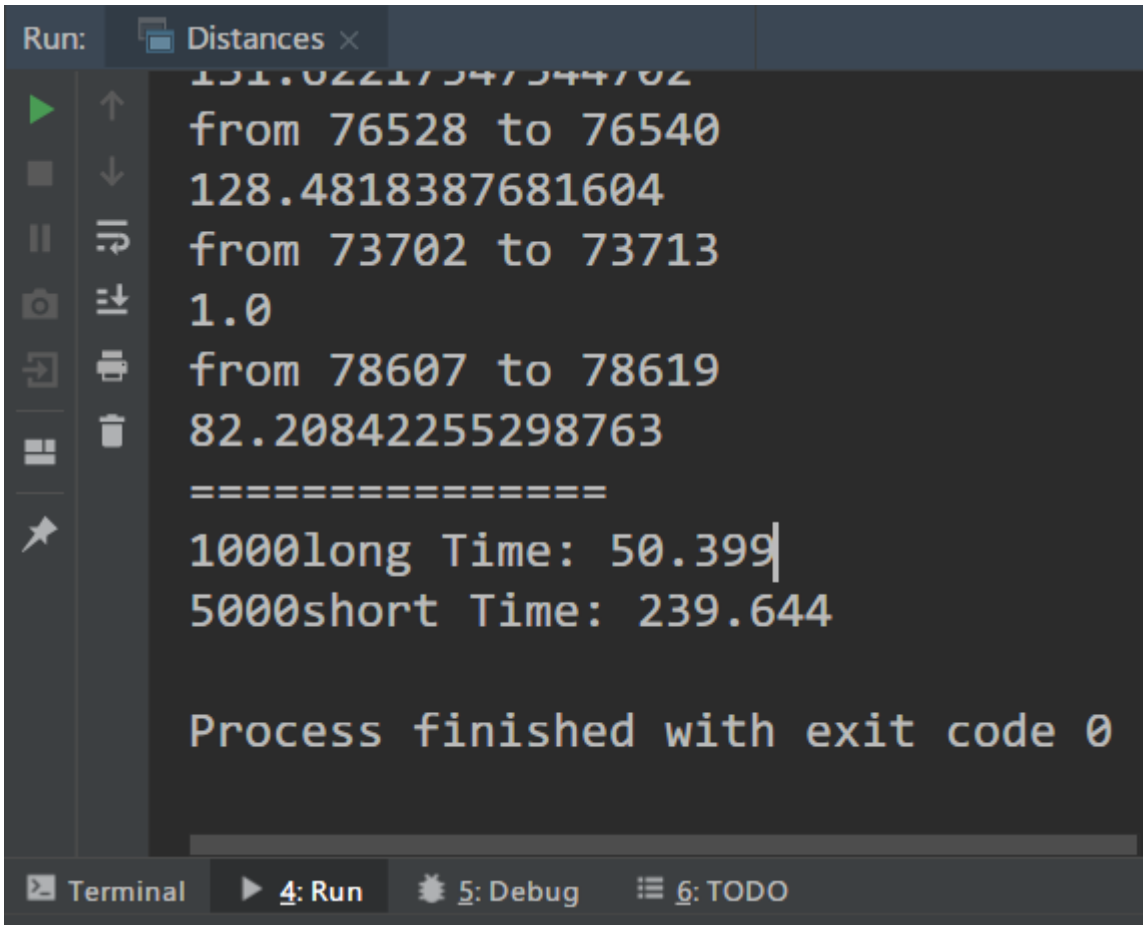
```
        pred[s] = s;
        pq.change(s, dist[s]);

        // run Dijkstra's algorithm
        while (!pq.isEmpty()) {
            int v = pq.delMin();
            //// System.out.println("process " + v + " " + dist[v]);

            // v not reachable from s so stop
            if (pred[v] == -1) break;

            // scan through all nodes w adjacent to v
            IntIterator i = G.neighbors(v);
            while (i.hasNext()) {
                int w = i.next();
                if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
                    dist[w] = dist[v] + G.distance(v, w);
                    pq.change(w, dist[w]);
                    pred[w] = v;
                    //// System.out.println("    lower " + w + " to " + dist[w]);
                }
            }
        }
    }
```

该算法是未经优化过的最简实现，虽然能完成最短路径的计算，但是，求得每条最短路径的复杂度为 N^2



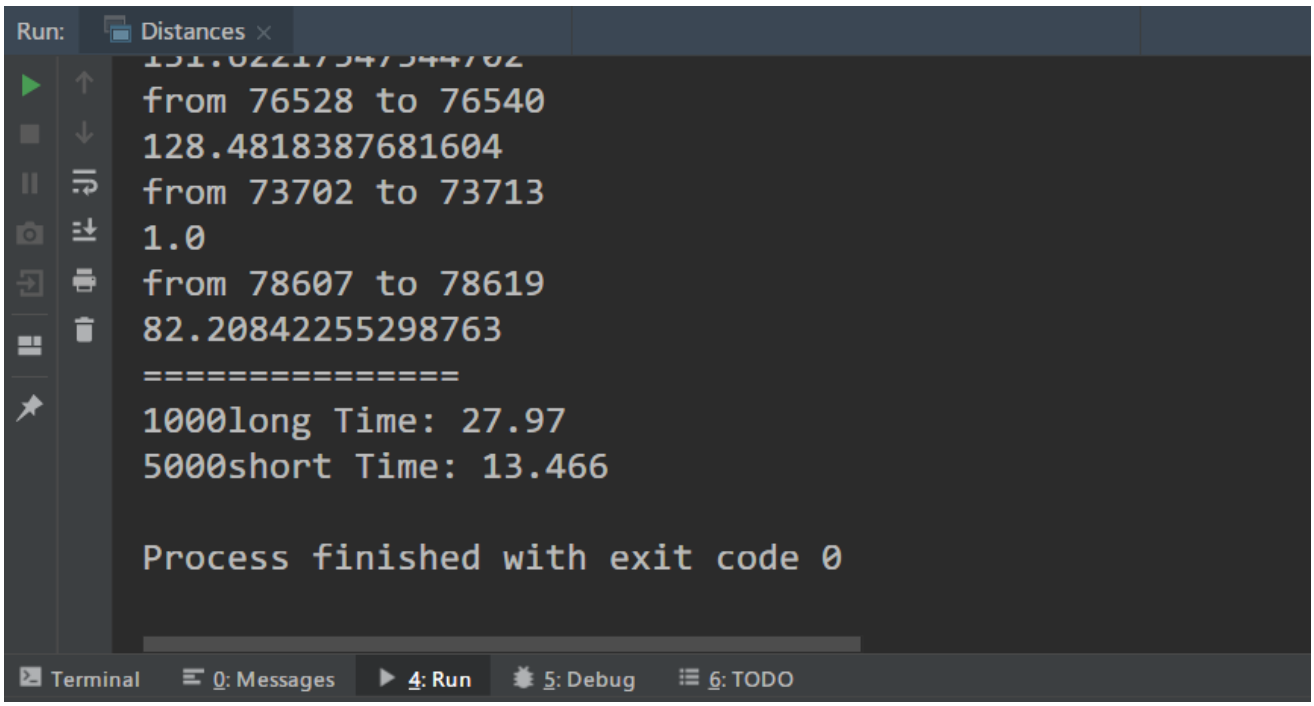
使用该算法完成 1000条长路径和 5000条短路径所用的总时间达到了 289s

2. 优化，发现最短路之后就停止搜索

由 Dijkstra 算法可知，从优先队列里面取出的最小顶点，其最短路一定已经确定，因此，可以直接停止搜索

```
while (!pq.isEmpty()) {
            int v = pq.delMin();
            if (v == d) break;
            ...
       }
```

提前停止搜索后，算法的执行时间有了显著的减少：



还是不够快?

因为每次查询最短路，整个 `dist` 数组，整个 `pred` 数组，全部都需要重新初始化，而这个初始化时间是与 V 成正比的，因此，如果能每次只初始化上次修改过的元素，还可以进一步提高速度。

可以将 `dist` `pred` 两个数组的初始化，以及优先队列的初始化操作在 `dijstra` 对象的构造函数中执行：

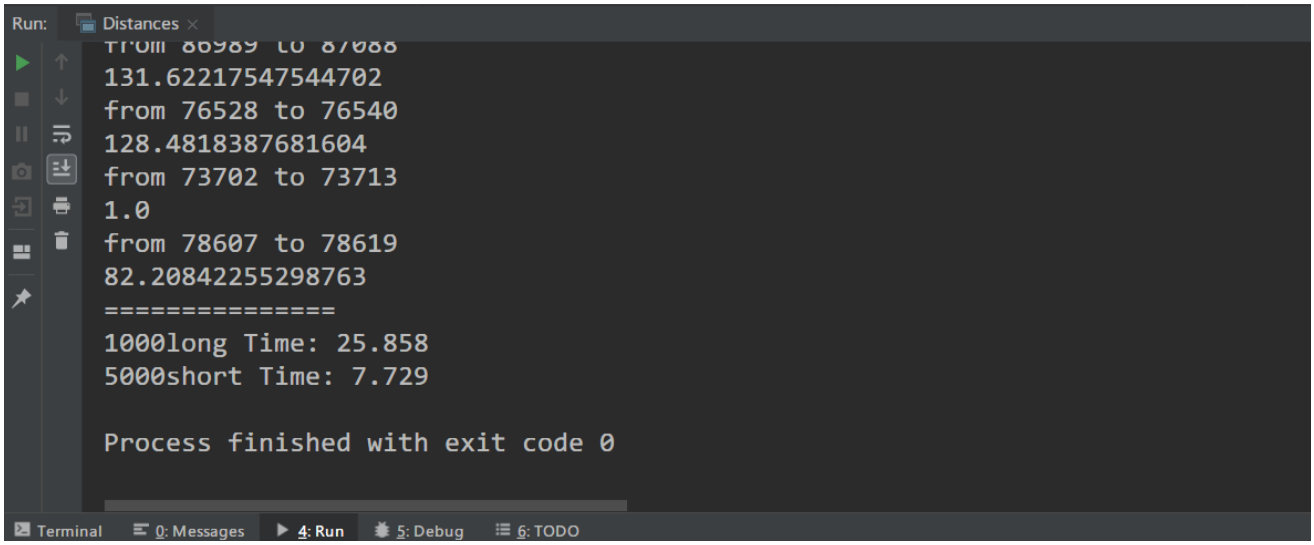
```
public Dijkstra(EuclideanGraph G) {
        this.G = G;
        int V = G.V();
        // initialize
        dist = new double[V];
        pred = new int[V];
        for (int v = 0; v < V; v++) dist[v] = INFINITY;
        for (int v = 0; v < V; v++) pred[v] = -1;

        // priority queue
        pq = new IndexPQ(V);
        for (int v = 0; v < V; v++) pq.insert(v, dist[v]);
```

```
13
14      }
```

修改 dijstra 函数，使用 `LinkedList` 类型变量 `changed` 记录每次求最短路过程中发生了变化的顶点元素，之后，在每次求最短路之前，只对这些顶点初始化：

```
1  private void dijkstra(int s, int d) {
2          int V = G.V();
3          pq.N = V;
4          while (!changed.isEmpty()){
5              int i = changed.removeFirst();
6              dist[i] = INFINITY;
7              pred[i] = -1;
8              pq.change(i,INFINITY);
9          }
10
11         //pq = new IndexPQ(V);
12         //for (int v = 0; v < V; v++) pq.insert(v, INFINITY);
13
14         // set distance of source
15         dist[s] = 0.0;
16         pred[s] = s;
17         changed.add(s);
18         pq.change(s, dist[s]);
19
20         // run Dijkstra's algorithm
21         while (!pq.isEmpty()) {
22             int v = pq.delMin();
23             if (v == d) break;
24             //// System.out.println("process " + v + " " + dist[v]);
25
26             // v not reachable from s so stop
27             if (pred[v] == -1) break;
28
29             // scan through all nodes w adjacent to v
30             IntIterator i = G.neighbors(v);
31             while (i.hasNext()) {
32                 int w = i.next();
33                 if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
34                     dist[w] = dist[v] + G.distance(v, w);
35                     //G.point(v).drawTo(G.point(w));
36                     pq.change(w, dist[w]);
37                     pred[w] = v;
38                     changed.add(w);
39                     //// System.out.println("    lower " + w + " to " + dist[w]);
40                 }
41             }
42         }
43     }
```



可以看到，在完成上述修改后，对短路径的执行效率提升效果最大，所用时间缩短为一半

3. 使用 A* 算法，减少搜索范围

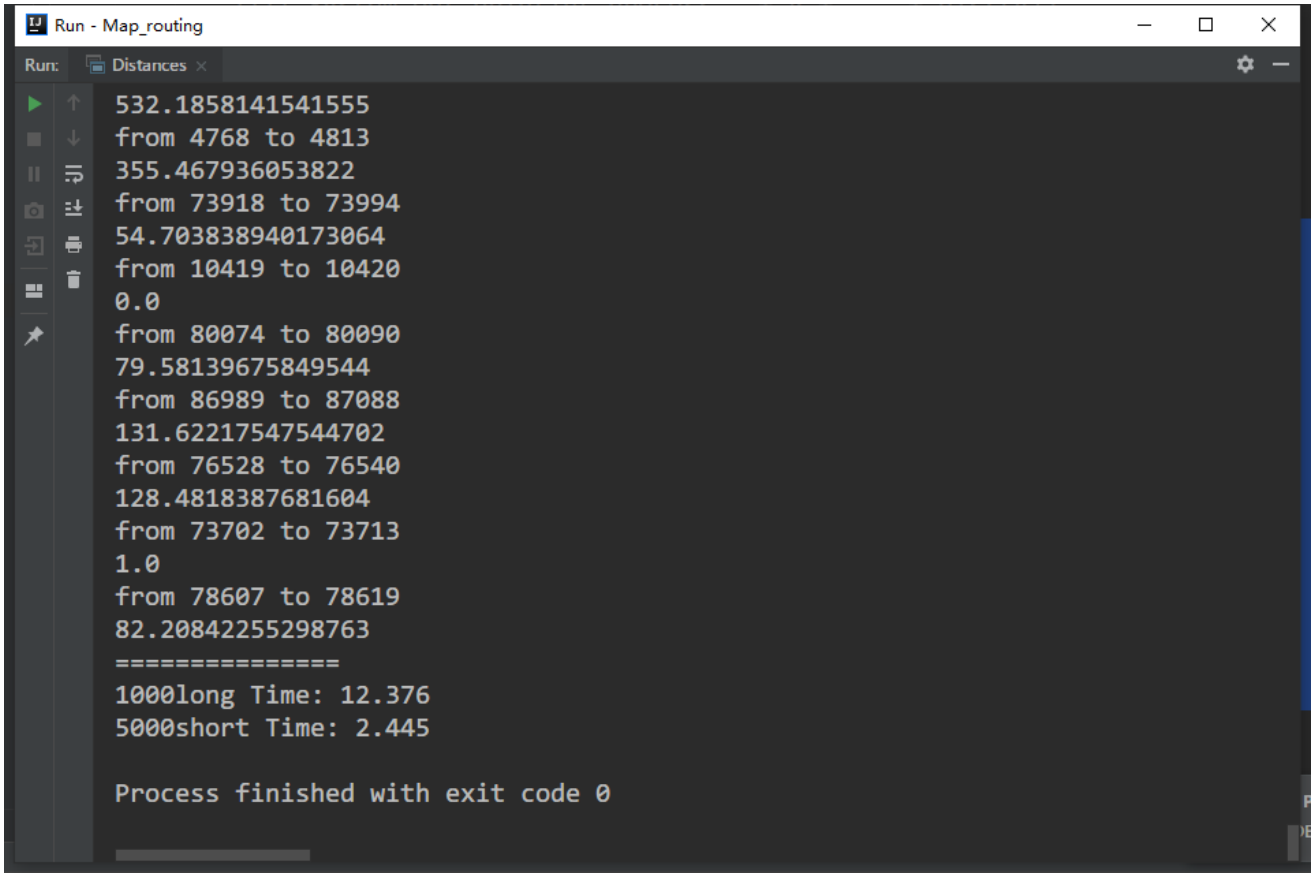
由于我们在地图中寻找最短路径，两点间路径的权重即为这两点之间的欧氏距离。因此，可以给 dijstra 加入一个启发式函数，构成 A* 算法，能更快的到达目的地。

按照题目给出的方案，对于一般图，Dijkstra通过将d[w]更新为d[v]+从v到w的距离来松弛边v-w。对于地图，则将dw]更新为d[v]+从v到w的距离+从w到d的欧式距离-从v到d的欧式距离。

因此，将 Dijkstra 算法部分修改为如下：

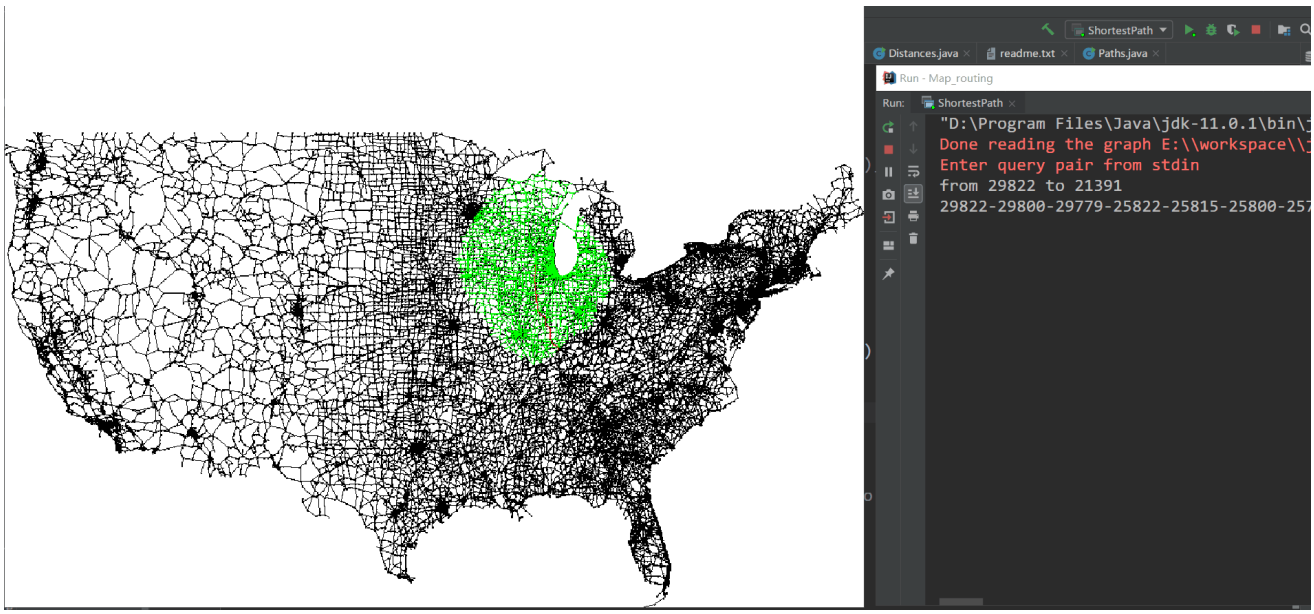
```
while (i.hasNext()) {
                int w = i.next();
                double dt = G.distance(v, w) + G.distance(w, d) - G.distance(v, d);
                if (wt[v] + dt< wt[w] - EPSILON) {
                //if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
                    dist[w] = dist[v] + G.distance(v, w);
                    wt[w]=wt[v] + dt;
                    //G.point(v).drawTo(G.point(w));
                    //pq.change(w, wt[w]);
                    mpq.changeKey(w,wt[w]);
                    pred[w] = v;
                    changed.add(w);
                    //Thread.sleep(2);
                    //// System.out.println("    lower " + w + " to " + dist[w]);
                }
            }
```

修改之后，重新计算完成 1000条长路径和 5000条短路径所用的总时间

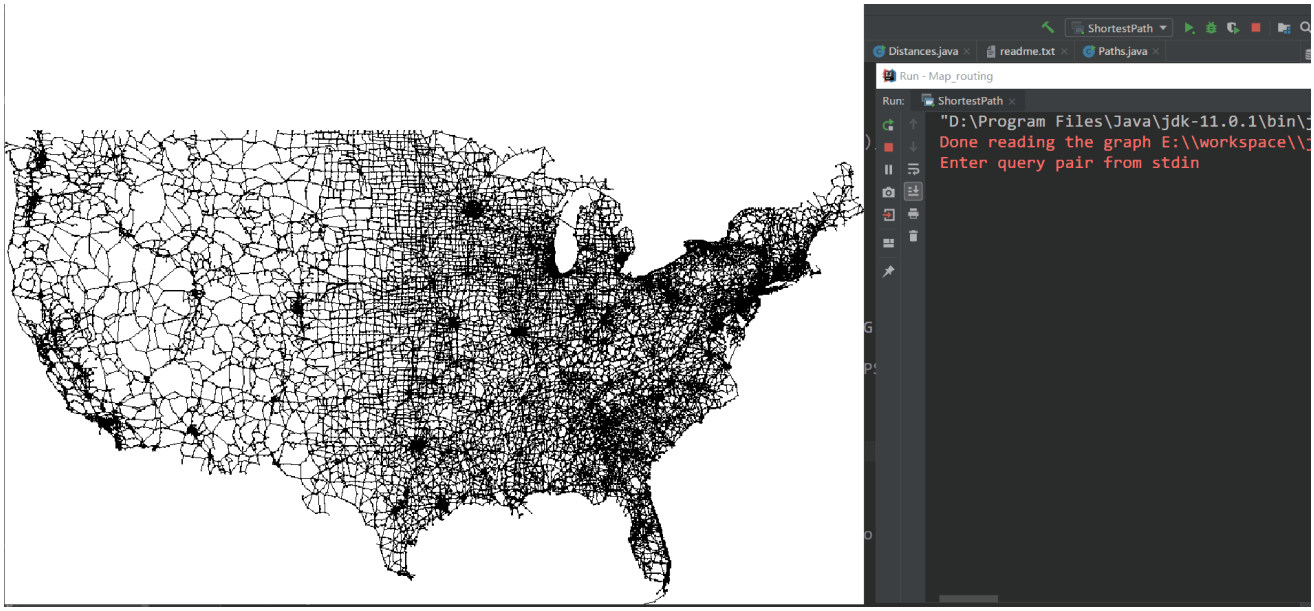


可以看到，完成上述修改后，算法的运行时间得到大幅减少，特别是执行短搜索的时间，从7s缩短到2s。

为算法执行可视化，可以明显看到，使用 A* 算法之前，整个搜索区域从起点开始以近似圆形展开，使用 A* 算法后，搜索区域有明显的指向性。大幅缩小了搜索范围

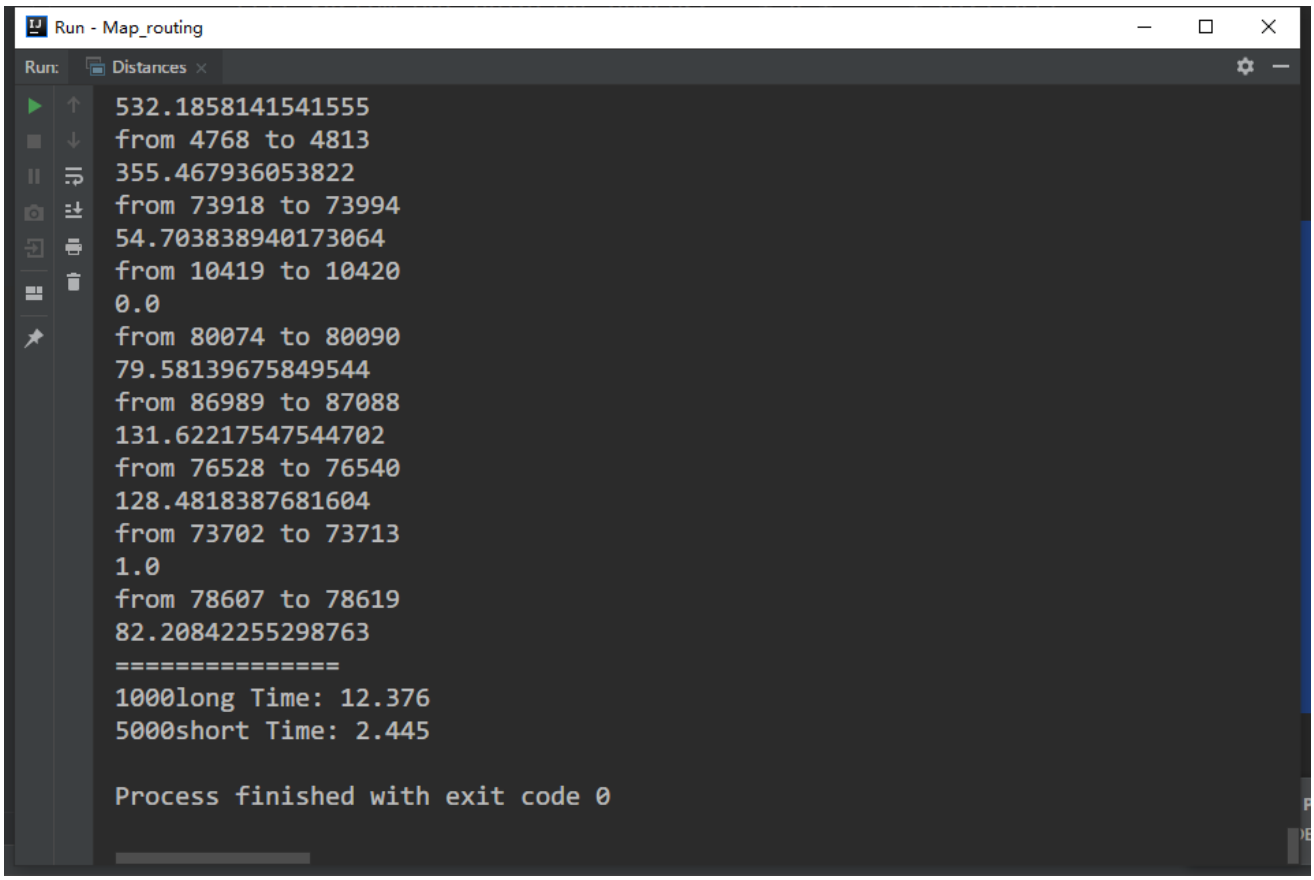


http://media.sumblog.cn/%E6%9C%AA%E4%BC%98%E5%8C%961.gif



四、实验结果

完成上述修改之后，计算完成 1000条长路径和 5000条短路径所用的总时间



1000 条长路径用时 12.376s

5000 条短路径用时 2.445 s

附：改进后的 Dijkstra 算法

```java
/******************************************************************************
 *  Dijkstra's algorithm.
 *
 ******************************************************************************/

import edu.princeton.cs.algs4.StdOut;

import java.awt.*;
import java.util.LinkedList;
//import edu.princeton.cs.algs4.IndexMultiwayMinPQ;



public class Dijkstra {
    private static double INFINITY = Double.MAX_VALUE;
    private static double EPSILON  = 0.000001;

    private EuclideanGraph G;
    private double[] dist;
    private int[] pred;
    private double[] wt;
    IndexPQ pq;
    IndexMultiwayMinPQ mpq;

    private LinkedList<Integer> changed = new LinkedList<Integer>();

    public Dijkstra(EuclideanGraph G) {
        this.G = G;
        int V = G.V();
        // initialize
        dist = new double[V];
        pred = new int[V];
        wt = new double[V];
        for (int v = 0; v < V; v++) dist[v] = INFINITY;
        for (int v = 0; v < V; v++) wt[v] = INFINITY;
        for (int v = 0; v < V; v++) pred[v] = -1;

        // priority queue
        pq = new IndexPQ(V);
        for (int v = 0; v < V; v++) pq.insert(v, dist[v]);
        mpq = new IndexMultiwayMinPQ<Double>(V,6);
        for (int v = 0; v< V; v++) mpq.insert(v, INFINITY);

    }

    // return shortest path distance from s to d
    public double distance(int s, int d) throws InterruptedException {
        dijkstra(s, d);
        return dist[d];
    }
```

```
51
52      // print shortest path from s to d  (interchange s and d to print in right
    order)
53      public void showPath(int d, int s,boolean draw) throws InterruptedException {
54          if (!draw)
55              dijkstra(s, d);
56          else
57              dijkstra(s,d,draw);
58          if (pred[d] == -1) {
59              System.out.println(d + " is unreachable from " + s);
60              return;
61          }
62          for (int v = d; v != s; v = pred[v])
63              System.out.print(v + "-");
64          System.out.println(s);
65      }
66
67      public void showPath(int d, int s) throws InterruptedException {
68          dijkstra(s, d);
69          if (pred[d] == -1) {
70              System.out.println(d + " is unreachable from " + s);
71              return;
72          }
73          for (int v = d; v != s; v = pred[v])
74              System.out.print(v + "-");
75          System.out.println(s);
76      }
77
78      // plot shortest path from s to d
79      public void drawPath(int s, int d) throws InterruptedException {
80          dijkstra(s, d);
81          if (pred[d] == -1) return;
82          //Turtle.setColor(Color.red);
83          //Turtle.set_Stroke(2.0f);
84          for (int v = d; v != s; v = pred[v]){
85              G.point(v).drawTo(G.point(pred[v]));
86              Turtle.render();
87          }
88
89          //Turtle.set_Stroke(1.0f);
90
91
92      }
93
94      // Dijkstra's algorithm to find shortest path from s to d
95      private void dijkstra(int s, int d) throws InterruptedException {
96          int V = G.V();
97          //pq.N = V;
98          mpq.n = V;
99          mpq.nmax = V;
100
101         while (!changed.isEmpty()){
102             int i = changed.removeFirst();
```

```
            dist[i] = INFINITY;
            wt[i] = INFINITY;
            pred[i] = -1;
            //StdOut.println(i);
            mpq.changeKey(i,INFINITY);
        }

        //mpq = new IndexMultiwayMinPQ<Double>(V,4);
        //for (int v = 0; v< V; v++) mpq.insert(v, INFINITY);

        // set distance of source
        dist[s] = 0.0;
        wt[s] = 0.0;
        pred[s] = s;
        changed.add(s);
        //pq.change(s, wt[s]);
        mpq.changeKey(s,wt[s]);

        // run Dijkstra's algorithm
        while (!mpq.isEmpty()) {
            //int v = pq.delMin();
            int v = mpq.delMin();
            if (v == d) break;
            //// System.out.println("process " + v + " " + dist[v]);

            // v not reachable from s so stop
            if (pred[v] == -1) break;

            // scan through all nodes w adjacent to v
            IntIterator i = G.neighbors(v);
            while (i.hasNext()) {
                int w = i.next();
                double dt = G.distance(v, w) + G.distance(w, d) - G.distance(v,
d);
                if (wt[v] + dt< wt[w] - EPSILON) {
                //if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
                    dist[w] = dist[v] + G.distance(v, w);
                    wt[w]=wt[v] + dt;
                    //G.point(v).drawTo(G.point(w));
                    //pq.change(w, wt[w]);
                    mpq.changeKey(w,wt[w]);
                    pred[w] = v;
                    changed.add(w);
                    //Thread.sleep(2);
                    //// System.out.println("    lower " + w + " to " + dist[w]);
                }
            }
        }
    }

    private void dijkstra(int s, int d,boolean draw) throws InterruptedException {

        int V = G.V();
```

```
        mpq.n = V;
        mpq.nmax = V;
        while (!changed.isEmpty()){
            int i = changed.removeFirst();
            dist[i] = INFINITY;
            wt[i] = INFINITY;
            pred[i] = -1;
            mpq.changeKey(i,INFINITY);
        }

        //pq = new IndexPQ(V);
        //for (int v = 0; v < V; v++) pq.insert(v, INFINITY);

        // set distance of source
        dist[s] = 0.0;
        wt[s] = 0.0;
        pred[s] = s;
        changed.add(s);
        mpq.changeKey(s, wt[s]);

        // run Dijkstra's algorithm
        while (!mpq.isEmpty()) {
            int v = mpq.delMin();
            if (v == d) break;
            //// System.out.println("process " + v + " " + dist[v]);

            // v not reachable from s so stop
            if (pred[v] == -1) break;

            // scan through all nodes w adjacent to v
            IntIterator i = G.neighbors(v);
            while (i.hasNext()) {
                int w = i.next();
                double dt = G.distance(v, w) + G.distance(w, d) - G.distance(v,
d);
                if (wt[v] + dt< wt[w] - EPSILON) {
                    //if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
                    dist[w] = dist[v] + G.distance(v, w);
                    wt[w]=wt[v] + dt;
                    Turtle.setColor(Color.blue);
                    //drawPath(s,v);
                    //drawPath(s,d);
                    Turtle.setColor(Color.green);
                    G.point(v).drawTo(G.point(w));
                    Turtle.render();
                    mpq.changeKey(w, wt[w]);
                    pred[w] = v;
                    changed.add(w);

                    //// System.out.println("    lower " + w + " to " + dist[w]);
                }
            }
            //Thread.sleep(1);
```

```
207             }
208         }
209
210
211 }
```