

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

```
1 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

```
1 private void dijkstra(int s, int d) {
2     int V = G.V();
3
4     // initialize
5     dist = new double[V];
6     pred = new int[V];
7     for (int v = 0; v < V; v++) dist[v] = INFINITY;
8     for (int v = 0; v < V; v++) pred[v] = -1;
9
10    // priority queue
11    IndexPQ pq = new IndexPQ(V);
12    for (int v = 0; v < V; v++) pq.insert(v, dist[v]);
13
14    // set distance of source
15    dist[s] = 0.0;
```

```

16     pred[s] = s;
17     pq.change(s, dist[s]);
18
19     // run Dijkstra's algorithm
20     while (!pq.isEmpty()) {
21         int v = pq.delMin();
22         //// System.out.println("process " + v + " " + dist[v]);
23
24         // v not reachable from s so stop
25         if (pred[v] == -1) break;
26
27         // scan through all nodes w adjacent to v
28         IntIterator i = G.neighbors(v);
29         while (i.hasNext()) {
30             int w = i.next();
31             if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
32                 dist[w] = dist[v] + G.distance(v, w);
33                 pq.change(w, dist[w]);
34                 pred[w] = v;
35                 //// System.out.println("    lower " + w + " to " + dist[w]);
36             }
37         }
38     }
39 }

```

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

```

Run: Distances x
131.02217547544702
from 76528 to 76540
128.4818387681604
from 73702 to 73713
1.0
from 78607 to 78619
82.20842255298763
=====
1000long Time: 50.399
5000short Time: 239.644

Process finished with exit code 0
Terminal 4: Run 5: Debug 6: TODO

```

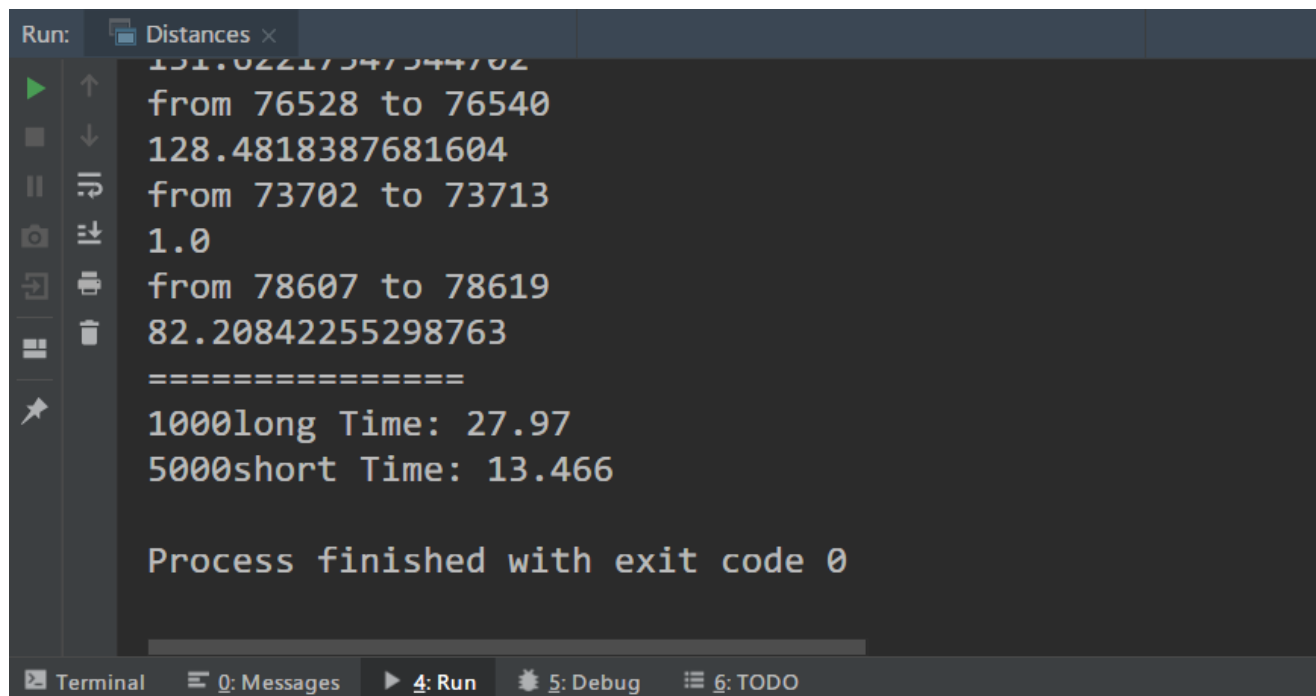
使用该算法完成 1000 条长路径和 5000 条短路径所用的总时间达到了 289s

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

```
1 while (!pq.isEmpty()) {
2     int v = pq.delMin();
3     if (v == d) break;
4     ...
5 }
```

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 `dijkstra` 对象的构造函数中执行：

```
1 public Dijkstra(EuclideanGraph G) {
2     this.G = G;
3     int v = G.V();
4     // initialize
5     dist = new double[v];
6     pred = new int[v];
7     for (int v = 0; v < V; v++) dist[v] = INFINITY;
8     for (int v = 0; v < V; v++) pred[v] = -1;
9
10    // priority queue
11    pq = new IndexPQ(V);
12    for (int v = 0; v < V; v++) pq.insert(v, dist[v]);
13 }
```

```
13  
14 }
```

修改 dijkstra 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 }
```

```
Run: Distances x
FROM 86989 TO 87088
131.62217547544702
from 76528 to 76540
128.4818387681604
from 73702 to 73713
1.0
from 78607 to 78619
82.20842255298763
=====
1000long Time: 25.858
5000short Time: 7.729

Process finished with exit code 0
```

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 dijkstra 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：

```
1 while (i.hasNext()) {
2     int w = i.next();
3     double dt = G.distance(v, w) + G.distance(w, d) - G.distance(v, d);
4     if (wt[v] + dt < wt[w] - EPSILON) {
5         //if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
6             dist[w] = dist[v] + G.distance(v, w);
7             wt[w] = wt[v] + dt;
8             //G.point(v).drawTo(G.point(w));
9             //pq.change(w, wt[w]);
10            mpq.changeKey(w, wt[w]);
11            pred[w] = v;
12            changed.add(w);
13            //Thread.sleep(2);
14            //// System.out.println("    lower " + w + " to " + dist[w]);
15        }
16    }
```

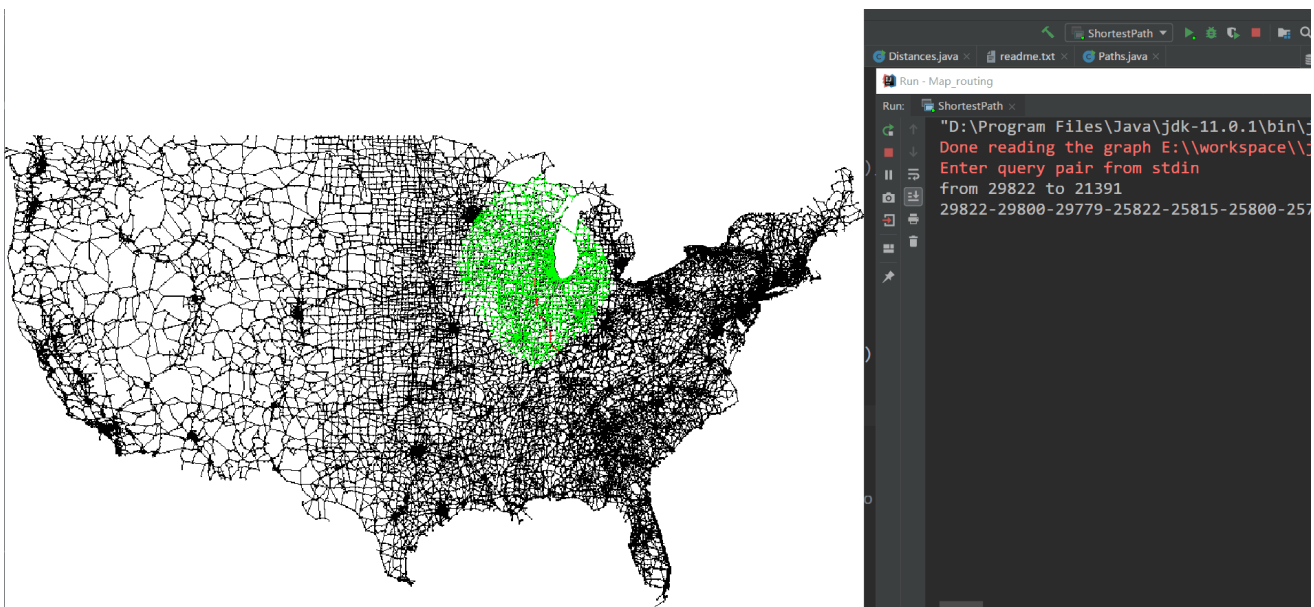
修改之后，重新计算完成 1000 条长路径和 5000 条短路径所用的总时间

```
Run - Map_routing
Distances x
532.1858141541555
from 4768 to 4813
355.467936053822
from 73918 to 73994
54.703838940173064
from 10419 to 10420
0.0
from 80074 to 80090
79.58139675849544
from 86989 to 87088
131.62217547544702
from 76528 to 76540
128.4818387681604
from 73702 to 73713
1.0
from 78607 to 78619
82.20842255298763
=====
1000long Time: 12.376
5000short Time: 2.445

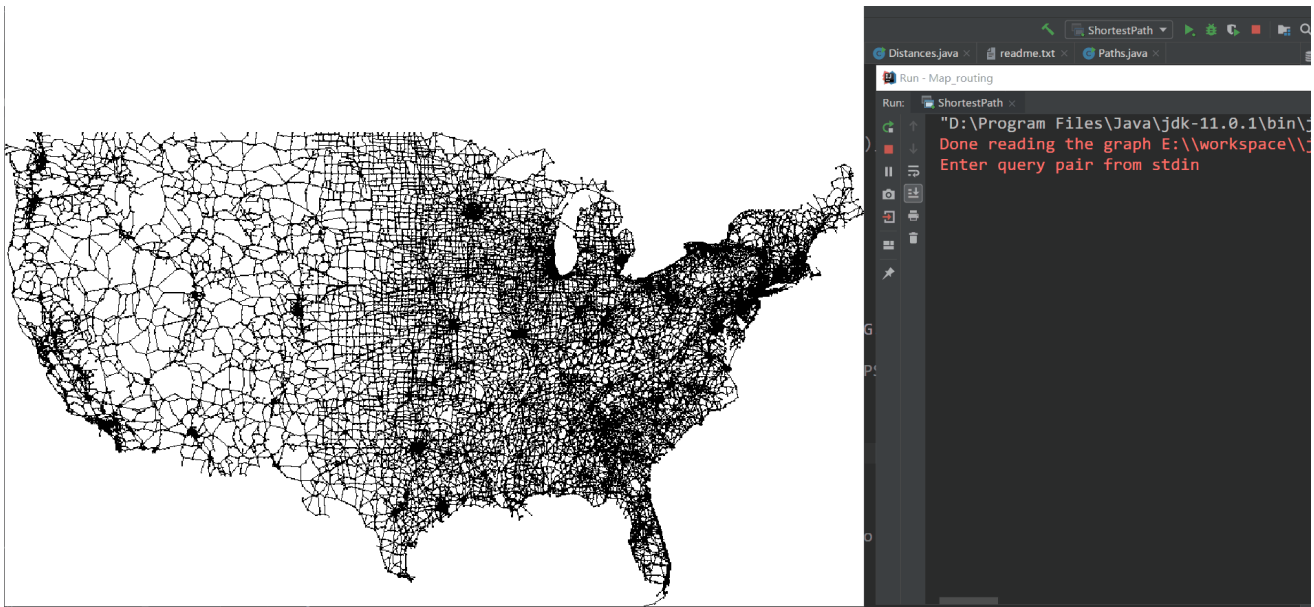
Process finished with exit code 0
```

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

```
Run - Map_routing
Run: Distances x
532.1858141541555
from 4768 to 4813
355.467936053822
from 73918 to 73994
54.703838940173064
from 10419 to 10420
0.0
from 80074 to 80090
79.58139675849544
from 86989 to 87088
131.62217547544702
from 76528 to 76540
128.4818387681604
from 73702 to 73713
1.0
from 78607 to 78619
82.20842255298763
=====
1000long Time: 12.376
5000short Time: 2.445

Process finished with exit code 0
```

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

```
1  /*****
2  *   Dijkstra's algorithm.
3  *
4  *****/
5
6  import edu.princeton.cs.algs4.StdOut;
7
8  import java.awt.*;
9  import java.util.LinkedList;
10 //import edu.princeton.cs.algs4.IndexMultiwayMinPQ;
11
12
13
14 public class Dijkstra {
15     private static double INFINITY = Double.MAX_VALUE;
16     private static double EPSILON = 0.000001;
17
18     private EuclideanGraph G;
19     private double[] dist;
20     private int[] pred;
21     private double[] wt;
22     IndexPQ pq;
23     IndexMultiwayMinPQ mpq;
24
25     private LinkedList<Integer> changed = new LinkedList<Integer>();
26
27     public Dijkstra(EuclideanGraph G) {
28         this.G = G;
29         int V = G.V();
30         // initialize
31         dist = new double[V];
32         pred = new int[V];
33         wt = new double[V];
34         for (int v = 0; v < V; v++) dist[v] = INFINITY;
35         for (int v = 0; v < V; v++) wt[v] = INFINITY;
36         for (int v = 0; v < V; v++) pred[v] = -1;
37
38         // priority queue
39         pq = new IndexPQ(V);
40         for (int v = 0; v < V; v++) pq.insert(v, dist[v]);
41         mpq = new IndexMultiwayMinPQ<Double>(V,6);
42         for (int v = 0; v < V; v++) mpq.insert(v, INFINITY);
43
44     }
45
46     // return shortest path distance from s to d
47     public double distance(int s, int d) throws InterruptedException {
48         dijkstra(s, d);
49         return dist[d];
50     }
51 }
```



```

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.setStroke(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.setStroke(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();

```

```

103         dist[i] = INFINITY;
104         wt[i] = INFINITY;
105         pred[i] = -1;
106         //StdOut.println(i);
107         mpq.changeKey(i, INFINITY);
108     }
109
110     //mpq = new IndexMultiwayMinPQ<Double>(V,4);
111     //for (int v = 0; v < V; v++) mpq.insert(v, INFINITY);
112
113     // set distance of source
114     dist[s] = 0.0;
115     wt[s] = 0.0;
116     pred[s] = s;
117     changed.add(s);
118     //pq.change(s, wt[s]);
119     mpq.changeKey(s, wt[s]);
120
121     // run Dijkstra's algorithm
122     while (!mpq.isEmpty()) {
123         //int v = pq.delMin();
124         int v = mpq.delMin();
125         if (v == d) break;
126         //// System.out.println("process " + v + " " + dist[v]);
127
128         // v not reachable from s so stop
129         if (pred[v] == -1) break;
130
131         // scan through all nodes w adjacent to v
132         IntIterator i = G.neighbors(v);
133         while (i.hasNext()) {
134             int w = i.next();
135             double dt = G.distance(v, w) + G.distance(w, d) - G.distance(v,
d);
136
137             if (wt[v] + dt < wt[w] - EPSILON) {
138                 //if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
139                 dist[w] = dist[v] + G.distance(v, w);
140                 wt[w] = wt[v] + dt;
141                 //G.point(v).drawTo(G.point(w));
142                 //pq.change(w, wt[w]);
143                 mpq.changeKey(w, wt[w]);
144                 pred[w] = v;
145                 changed.add(w);
146                 //Thread.sleep(2);
147                 //// System.out.println("    lower " + w + " to " + dist[w]);
148             }
149         }
150     }
151
152     private void dijkstra(int s, int d, boolean draw) throws InterruptedException {
153
154         int v = G.V();

```

```

155     mpq.n = v;
156     mpq.nmax = v;
157     while (!changed.isEmpty()){
158         int i = changed.removeFirst();
159         dist[i] = INFINITY;
160         wt[i] = INFINITY;
161         pred[i] = -1;
162         mpq.changeKey(i, INFINITY);
163     }
164
165     //pq = new IndexPQ(V);
166     //for (int v = 0; v < V; v++) pq.insert(v, INFINITY);
167
168     // set distance of source
169     dist[s] = 0.0;
170     wt[s] = 0.0;
171     pred[s] = s;
172     changed.add(s);
173     mpq.changeKey(s, wt[s]);
174
175     // run Dijkstra's algorithm
176     while (!mpq.isEmpty()) {
177         int v = mpq.delMin();
178         if (v == d) break;
179         //// System.out.println("process " + v + " " + dist[v]);
180
181         // v not reachable from s so stop
182         if (pred[v] == -1) break;
183
184         // scan through all nodes w adjacent to v
185         IntIterator i = G.neighbors(v);
186         while (i.hasNext()) {
187             int w = i.next();
188             double dt = G.distance(v, w) + G.distance(w, d) - G.distance(v,
d);
189
190             if (wt[v] + dt < wt[w] - EPSILON) {
191                 //if (dist[v] + G.distance(v, w) < dist[w] - EPSILON) {
192                 dist[w] = dist[v] + G.distance(v, w);
193                 wt[w] = wt[v] + dt;
194                 Turtle.setColor(Color.blue);
195                 //drawPath(s,v);
196                 //drawPath(s,d);
197                 Turtle.setColor(Color.green);
198                 G.point(v).drawTo(G.point(w));
199                 Turtle.render();
200                 mpq.changeKey(w, wt[w]);
201                 pred[w] = v;
202                 changed.add(w);
203
204                 //// System.out.println("    lower " + w + " to " + dist[w]);
205             }
206         }
207     }
208     //Thread.sleep(1);

```

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