地图路由（Map Routing）

1 问题描述

实现经典的 Dijkstra 最短路径算法，并对其进行优化。这种算法广泛应用于地理信息系统（GIS），包括 MapQuest 和基于 GPS 的汽车导航系统。

目标　优化 Dijkstra 算法，使其可以处理给定图的数千条最短路径查询。一旦你读取图（并可选地预处理），你的程序应该在亚线性时间内解决最短路径问题。一种方法是预先计算出所有顶点对的最短路径；然而，你无法承受存储所有这些信息所需的二次空间。你的目标是减少每次最短路径计算所涉及的工作量，而不会占用过多的空间。建议你选择下面的一些潜在想法来实现，或者你可以开发和实现自己的想法。

2 解决思路

2.1 想法 1

Dijkstra 算法的朴素实现检查图中的所有 V 个顶点。减少检查的顶点数量的一种策略是一旦发现目的地的最短路径就停止搜索。通过这种方法，可以使每个最短路径查询的运行时间与 E' log V' 成比例，其中 E' 和 V' 是 Dijkstra 算法检查的边和顶点数。然而，这需要一些小心，因为只是重新初始化所有距离为 ∞ 就需要与 V 成正比的时间。由于你在不断执行查询，因而只需重新初始化在先前查询中改变的那些值来大大加速查询。

2.2 想法 2

你可以利用问题的欧式几何来进一步减少搜索时间，这在算法书的第 4.4 节描述过。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 d[w] 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。这种方法称之为 A* 算法。这种启发式方法会有性能上的影响，但不会影响正确性。

2.3 想法 3

使用更快的优先队列。在提供的优先队列中有一些优化空间。你也可以考虑使用 Sedgewick 程序中的多路堆（Multiway heaps, Section 2.4）。

3 程序实现

3.1 DijstraUndirectedSP

```
public class DijkstraUndirectedSP {
    private double[] distTo;
    private Edge[] edgeTo;
    private IndexMultiwayMinPQ<Double> pq; // 想法3 使用多路堆
    private EdgeWeightedGraph G;
    private int s, d;   // 起点和终点
    private Point[] points;
    Stack<Edge> path;   // 路径
    Draw draw;

    public DijkstraUndirectedSP(EdgeWeightedGraph G, Point[] points, Draw draw) {
        this.G = G;
        this.points = points;
        distTo = new double[G.V()];
        edgeTo = new Edge[G.V()];
        pq = new IndexMultiwayMinPQ<>(G.V(), 3);
        for (int v = 0; v < G.V(); v++)
            distTo[v] = Double.POSITIVE_INFINITY;
        this.draw = draw;
        this.draw.setPenColor(Color.RED);
    }

    // 恢复初始状态
    public void initDijkstra(int s, int d) {
        draw.clear();
        this.s = s;
        this.d = d;
        draw.setPenRadius(0.01);
        draw.setPenColor(Color.RED);
        draw.point(points[s].x/9000.0, points[s].y/9000.0+0.3);
        draw.point(points[d].x/9000.0, points[d].y/9000.0+0.3);
        draw.setPenRadius();
        for (int v = 0; v < G.V(); v++) {
```

```
                if (pq.contains(v)) pq.delete(v);
                if (edgeTo[v] != null) edgeTo[v] = null;
                if (distTo[v] != Double.POSITIVE_INFINITY) distTo[v] = Double.POSITIVE_INFINITY;
            }
            distTo[s] = 0.0;
            pq.insert(s, distTo[s]);
        }

        public boolean hasPathTo(int v) {
            draw.setPenColor(Color.BLACK);
            while (!pq.isEmpty()) {
                int x = pq.delMin();
                // 想法1: 一旦找到终点就退出
                if (x == v) {
                    return true;
                }
                for (Edge e : G.adj(x)) {
                    int a = e.either();
                    int b = e.other(a);
                    // 画出当前要松弛的边
                    draw.line(points[a].x / 9000.0, points[a].y / 9000.0 + 0.3,
                    points[b].x / 9000.0, points[b].y / 9000.0 + 0.3);
                    relax(e, x);
                }
            }
            return distTo[v] < Double.POSITIVE_INFINITY;
        }

        private void relax(Edge e, int v) {
            int w = e.other(v);
            // 想法2: 采用启发式优化
            double weight = distTo[v] + e.weight() +
            points[w].getDistTo(points[d]) - points[v].getDistTo(points[d]);
    //        double weight = distTo[v] + e.weight();
            if (distTo[w] > weight && e.weight() != 0.0) {
                distTo[w] = weight;
                edgeTo[w] = e;
                if (pq.contains(w)) pq.decreaseKey(w, distTo[w]);
                else pq.insert(w, distTo[w]);
            }
        }
```

```
    public double distTo(int v) {
        return distTo[v];
    }

    public Iterable<Edge> pathTo(int v) {
        draw.setPenColor(Color.RED);
        path = new Stack<>();
        int x = v;
        for (Edge e = edgeTo[v]; e != null; e = edgeTo[x]) {
            path.push(e);
            StdOut.println(e);
            x = e.other(x);
        }
        return path;
    }
}
```

3.2 MapRouting

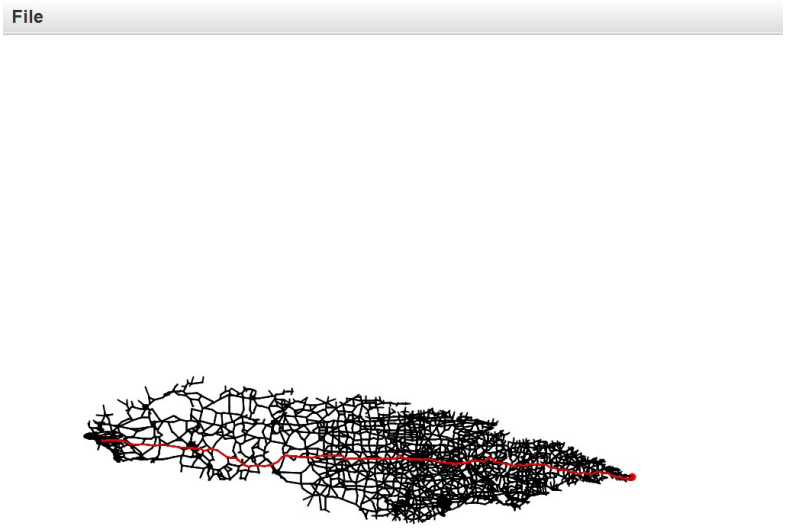
```
public static void main(String[] args) {
        In in = new In("usa.txt");
        int V = in.readInt();
        int E = in.readInt();
        EdgeWeightedGraph G = new EdgeWeightedGraph(V);
        Draw draw = new Draw();
        Point[] points = new Point[V];
        for (int i = 0; i < V; i++)
            points[in.readInt()] = new Point(in.readInt(), in.readInt());
        for (int i = 0; i < E; i++) {
            int v = in.readInt(), w = in.readInt();
            G.addEdge(new Edge(v, w, points[v].getDistTo(points[w])));
        }
        DijkstraUndirectedSP sp = new DijkstraUndirectedSP(G, points, draw);
        while (true) {
            System.out.println("请输入起点和终点: ");
            int s = StdIn.readInt(), d = StdIn.readInt();
            if (s == -1) break;
            if (s >= V || s < 0 || d >= V || d < 0)
                throw new IllegalArgumentException("s and d must be non-negative and less V.");
            sp.initDijkstra(s, d);
            double weight = 0.0;
```

```
                Stopwatch stopwatch = new Stopwatch();
                if (sp.hasPathTo(d)) {
                    System.out.println("路径: ");
                    for (var e : sp.pathTo(d)) {
                        StdOut.println(e);
                        weight += e.weight();
                        int a = e.either();
                        int b = e.other(a);
                        // 画出最短路径上的边
                        draw.line(points[a].x / 9000.0, points[a].y / 9000.0+0.3,
                        points[b].x / 9000.0, points[b].y / 9000.0+0.3);
                    }
                    System.out.printf("最短路长度为: %f\n", weight);
                } else System.out.println("不存在路径");
                System.out.println("花费时间: " + stopwatch.elapsedTime());
            }
        }
    }

    class Point {
        int x, y;

        public Point(int x, int y) {
            this.x = x;
            this.y = y;
        }

        public double getDistTo(Point p) {
            double dx = x - p.x;
            double dy = y - p.y;
            return Math.sqrt(dx * dx + dy * dy);
        }
    }
```

4 实验结果

4.1 输出示例

```
请输入起点和终点：
10000 20000
路径：
20000-20016 2.00000
20016-20022 3.60555
20022-20021 2.00000
20021-20035 7.81025
20062-20035 10.63015
20090-20062 21.63331
20089-20090 3.16228
20084-20089 9.48683
20072-20084 21.58703
20066-20072 2.23607
20052-20066 23.34524
20066-20072 2.23607
20072-20084 21.58703
20084-20089 9.48683
20089-20090 3.16228
20090-20062 21.63331
20062-20035 10.63015
20021-20035 7.81025
20022-20021 2.00000
20016-20022 3.60555
20000-20016 2.00000
最短路长度为：6570.331928
花费时间：88.138
```



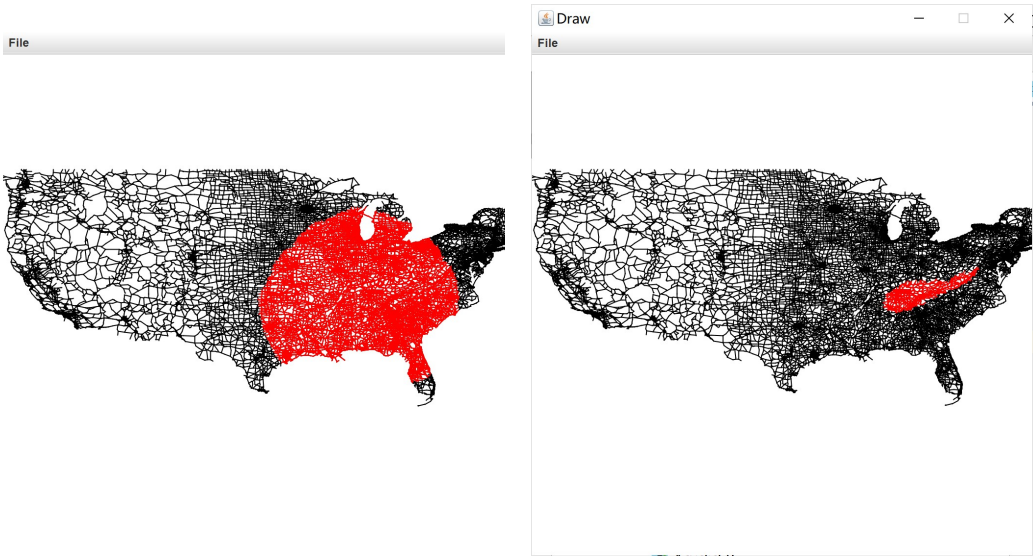
4.2 多路堆优化

	使用二叉堆	使用三叉堆
run time	4.687s	2.223s

4.3 启发式 A^* 优化

4.3.1 时间对比

	未用启发式 A^* 优化	使用启发式 A^* 优化
run time	28.203s	4.236s



(a) 未用启发式 A^* 优化　　(b) 使用启发式 A^* 优化

4.4

5 实验心得

通过本次实验我学习到了 $A*$ 的启发式优化，该优化大大提升了迪杰斯特拉算法的效率，在实际表现中非常的出色。除此之外我还学习了使用不同的优先队列来对迪杰斯特拉算法进行优化，它们同样很有效的提升了算法的表现。