三、实验内容及基本要求

（三）地图路由（Map Routing）

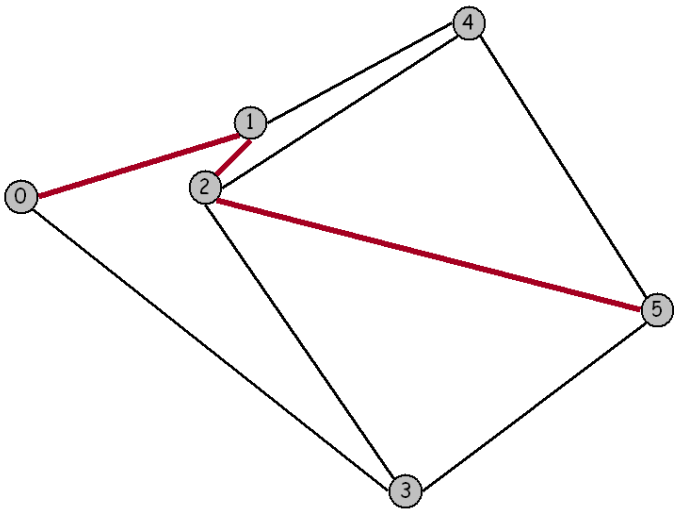
题目来源： [COS 226 Programming Assignment Map Routing]

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。

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，边视为相连的道路。输入数据文件中列出了表示地图的顶点数和边数，然后列出顶点（后跟其 *x* 和 *y* 坐标），然后列出边（顶点对），最后列出源点和汇点。 例如，如下左图信息表示右图：

```
6 9
0 1000 2400
1 2800 3000
2 2400 2500
3 4000    0
4 4500 3800
5 6000 1500

0 1
0 3
1 2
1 4
2 4
2 3
2 5
3 5
4 5
0 5
```



Dijkstra 算法。 Dijkstra 算法是最短路径问题的经典解决方案。教科书 4.4 节描述了该算法。 基本思路不难理解。 对于图中的每个顶点，我们维护从源点到该顶点的最短已知的路径长度，并且将这些长度保持在优先队列（*priority queue*, *PQ*）中。 初始时，把所有顶点放在这个队列中，并设置高优先级，然后将源点的优先级设为 0.0。 算法通过从 *PQ* 中取出最低优先级的顶点，然后检查可从该顶点经由一条边可达的所有顶点，以查看这条边是否提供了从源点到那个顶点较之之前已知最短路径的更短路径。 如果是这样，它会降低优先级来反映这种新的信息。

这里给出了 Dijkstra 算法计算从 0 到 5 的最短路径 0-1-2-5 的详细过程。

process 0 (0.0)
    lower 3 to 3841.9
    lower 1 to 1897.4
process 1 (1897.4)
    lower 4 to 3776.2
    lower 2 to 2537.7
process 2 (2537.7)
    lower 5 to 6274.0
process 4 (3776.2)
process 3 (3841.9)

process 5 (6274.0)

该方法计算最短路径的长度。 为了记录路径，我们还保持每个顶点的源点到该顶点最短路径上的前驱。 文件 Euclidean Graph.java，Point.java，IndexPQ.java，IntIterator.java 和 Dijkstra.java 提供了针对 map 的 Dijkstra 算法的基本框架实现，你应该以此作为起点。客户端程序 ShortestPath.java 求解一个单源点最短路径问题，并使用图形绘制了结果。客户端程序 Paths.java 求解了多个最短路径问题，并将最短路径打印到标准输出。客户端程序 Distances.java 求解了许多最短路径问题，仅将距离打印到标准输出。

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（预处理可选），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。

Your goal. Optimize Dijkstra's algorithm so that it can process thousands of shortest path queries for a given map. Once you read in (and optionally preprocess) the map, your program should solve shortest path problems in *sublinear* time. One method would be to precompute the shortest path for all pairs of vertices; however you cannot afford the quadratic space required to store all of this information. Your goal is to reduce the amount of work involved per shortest path computation, without using excessive space. We suggest a number of potential ideas below which you may choose to implement. Or you can develop and implement your own ideas.

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。

Idea 1. The naive implementation of Dijkstra's algorithm examines all V vertices in the graph. An obvious strategy to reduce the number of vertices examined is to stop the search as soon as you discover the shortest path to the destination. With this approach, you can make the running time per shortest path query proportional to E' log V' where E' and V' are the number of edges and vertices examined by Dijkstra's algorithm. However, this requires some care because just re-initializing all of the distances to ∞ would take time proportional to V. Since you are doing repeated queries, you can speed things up dramatically by only re-initializing those values that changed in the previous query.

想法 2. 你可以利用问题的欧式几何来进一步降低搜索时间，这在算法书的第 21.5 节描述过。对于一般图，Dijkstra 通过将 wt[w]更新为 wt[v] + 从 v 到 w 的距离来松弛边 v-w。 对于地图，则将 wt[w]更新为 wt[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。参考 pdf 文件 A* algorithm

Idea 2. You can cut down on the search time further by exploiting the Euclidean geometry of the problem, as described in Sedgewick 21.5. For general graphs, Dijkstra's relaxes edge v-w by updating wt[w] to the sum of wt[v] plus the distance from v to w. For maps, we instead update wt[w] to be the sum of wt[v] plus the distance from v to w *plus* the Euclidean distance

from w to d *minus* the Euclidean distance from v to d. This is known as the *A* algorithm*. This heuristics affects performance, but not correctness. (See Sedgewick 21.5 for a proof of correctness.)

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）。

Idea 3. Use a faster priority queue. There is some room for optimization in the supplied priority queue. You could also consider using a multiway heap as in Sedgewick Program 20.10.

测试。美国大陆文件 usa.txt 包含 87,575 个交叉口和 121,961 条道路。图形非常稀疏 - 平均的度为 2.8。你的主要目标应该是快速回答这个网络上的顶点对的最短路径查询。你的算法可能会有不同执行时间，这取决于两个顶点是否在附近或相距较远。我们提供测试这两种情况的输入文件。你可以假设所有的 *x* 和 *y* 坐标都是 0 到 10,000 之间的整数。

Testing. The file usa.txt contains 87,575 intersections and 121,961 roads in the continental United States. The graph is very sparse - the average degree is 2.8. Your main goal should be to answer shortest path queries quickly for pairs of vertices on this network. Your algorithm will likely perform differently depending on whether the two vertices are nearby or far apart. We provide input files that test both cases. You may assume that all of the x and y coordinates are integers between 0 and 10,000.

注：这个问题的实验由 Bob Sedgewick 和 Kevin Wayne 设计开发（Copyright © 2004）。更多信息可参考 http://algs4.cs.princeton.edu/。