

Graphs

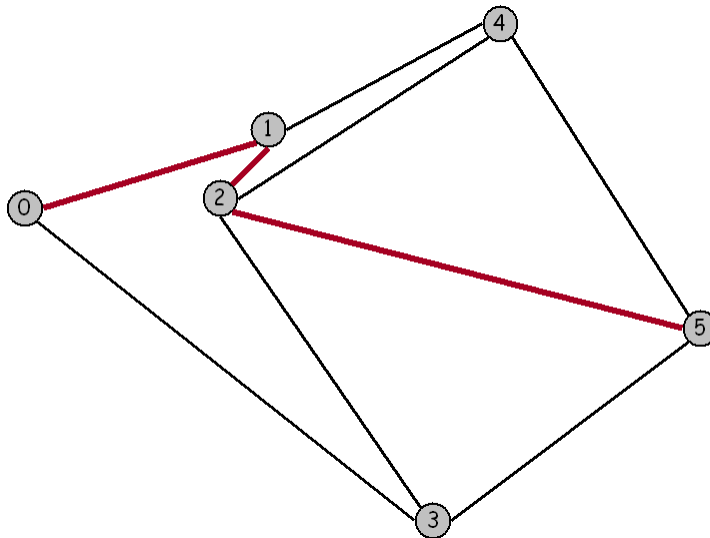
Background

For this problem, we need a priority queue. For simplicity, just use a linked list in which you search through the list to find the highest priority item. The code `OList.h` may be helpful. I have also included `TestOList.cpp` (that verifies that `OList` works) and `GraphStarter.h` (that gives you `Edge` and `Vertex` classes). These files are included in the zip file `OList7`.

Maps. In this assignment we work with *maps*, or graphs whose vertices are points in the plane and are connected by edges whose weights are Euclidean distances. Think of the vertices as cities and the edges as roads connected to them. To represent a map in a file, we list the number of vertices and edges, then list the vertices (index followed by its x and y coordinates), then list the edges (pairs of vertices), and **finally the source and sink vertices**. For example, represents the map below ((0,0) is the lower left hand corner):

```
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
```



The Problem:

For each of three undirected graphs (`prog7a.txt`, `prog7b.txt`, and `prog7c.txt`), perform each of the following algorithms:

1. Connectivity (10 points): Determine if the graph is connected by using a DFS (depth first search) from any node.

2. Compute an All Pairs Shortest Path Algorithm using Floyd Warshall (10 points)

```
procedure FloydWarshall ()
  for (k=0;k< n;k++)
    for (i=0;i< n;i++)
      if (path[i][k] !=INFINITY)
        for (j=0;j< n;j++)
          path[i][j] = min( path[i][j], path[i][k]+path[k][j] );
```

Print your path array after computation. You can assume there are no more than 12 vertices in the graph, so that you can statically allocated path.

3. Single Source Shortest Path using Dijkstra's algorithm (10 points).

Implement the classic Dijkstra's shortest path algorithm. Such algorithms are widely used in geographic information systems (GIS) including MapQuest and GPS-based car navigation systems. Dijkstra's algorithm is a classic solution to the shortest path problem. In this problem, you are not finding all pairs shortest path, but solving the problem for a given starting and ending position. You should get the same answer as one cell in Floyd Warshall – but don't cheat! We want to see that you have actually implemented Dijkstra's algorithm.

The basic idea is not difficult to understand. We maintain, for every vertex in the graph, the length of the shortest **known** path from the source to that vertex, and we maintain these lengths in a priority queue. Initially, we put the source on the queue with a length of zero. The algorithm proceeds by taking the lowest-distance vertex off the priority queue. **Use any priority queue implementation you want.** If the distance to that vertex is better than the best recorded, check all the vertices that can be reached from that vertex by one edge to see whether that edge gives a shorter path to the vertex from the source than the shortest previously known path. If so, put the new (vertex, length) pair on the PQ.

Each of the input files has the following format:

Number of Nodes, Number of Edges

For each node:

Node ID, x location, y location

Start Node, Goal Node.

For each undirected edge (between node1 and node2):

Node1 Node2

For this single source shortest path problem, assume you are only interested in going from Start Node to Goal Node

Below is a step-by-step description that shows how Dijkstra's algorithm finds the shortest path 0-1-2-5 from 0 to 5 in the example above. You need to **print out a similar trace** of your actions as proof that it is implemented correctly.

Single Source Shortest Path Between 0-5

Considering 0 BestWt=0 successors 3 1

Storing 3(3841)

Storing 1(1897)

Considering 1 BestWt=1897 successors 4 2 0

Storing 4(3775)

Storing 2(2537)

Considering 2 BestWt=2537 successors 5 3 4 1

Storing 5(6273)

Storing 3(5505)

Storing 4(5006)

// I could do slightly better if I knew I had already entered a smaller distance for 4 into the queue.

Considering 4 BestWt=3775 successors 5 2 1

Storing 5(6520)

Considering 3 BestWt=3841 successors 5 2 0

Storing 5(6341)

Considering 5 BestWt=6273 successors 4 3 2

Success 6273