Program Description

This lab is to use the Dijkstra's algorithm to find a shortest path in a directed graph from a vertex called "source" to all other vertices. The solution should provide a value of the shortest path along with all intermediate vertices and weights on the path.

Purpose of the Assignment

The purpose of this programming assignment is to understand and implement Graph and Dijkstra's algorithm with binary heap structure.

Data Structures Description

We used graph in this assignment. Also I recycled minimum priority with the binary heap from lab 5, I used a locator vector to the store the address of each element.

Algorithm Description

1. find_path()
   Print the shortest path from the source to the destination.
   This is a recursive function. The running time depends on the shortest route between the source and destination.

2. Djistras()
   Create a MPQ takes O(n*log n). Find the minimum element takes O(n). Remove the minimum element takes O(n*log(n)). Decrease the key takes O(n).
   Totally it is O(n*log n).

Program Organization and Description of Classes

Graph:

```
class Graph{


private:
   vector<Vertex*> vertList;
   vector<Edge*> edgeList;
public:
   Graph(string filename) {
   //
   //  The vertex with ID x is stored at vertList[x-1];
   //  for example, vertex 1 in vertList[0], vertex 2 in vertList[1] and so on.
   //
```

```
        ifstream ifs(filename.c_str());
        int vertnum;

        ifs>>vertnum;
        for(int i=0;i<vertnum;i++) // push all vertices into the vertex vector before
assigning the in and out list
        {
            Vertex *vert = new Vertex(i+1);
            vertList.push_back(vert);
        }
        int svert,evert,weit;
        ifs>>svert;
        while(svert!=-1){
            ifs>>evert;
            while(evert!=-1){
                ifs>>weit; // one weight means one edge
                Edge *tempedge = new Edge(vertList[svert-1],vertList[evert-1],weit); // new
edge
                edgeList.push_back(tempedge);
                vertList[svert-1]->outList.push_back(tempedge);
                vertList[evert-1]->inList.push_back(tempedge);
                ifs>>evert;
            }
            ifs>>svert;
        }
        ifs.close();
    }
  ~Graph() {
      for(int i = 0; i < vertList.size(); i++)
         delete vertList[i];
      for(int i = 0; i < edgeList.size(); i++)
         delete edgeList[i];
  }
  int getSize() {
      return vertList.size();
  }

  vector<Vertex*> getVertices() const {
      return vertList;
  }
```

```
    void PrintGraph() // print out all the edges and associated vertices in the entire
graph
    {
        cout<<endl<<vertList.size()<<endl;
        for(int i=0;i<vertList.size();i++){
            cout<<i+1<<" ";
            for (int j=0; j<getVertices()[i]->getOutEdges().size(); j++){
                cout<<getVertices()[i]->getOutEdges()[j]->geteVertP()->getID()<<" ";
                cout<<getVertices()[i]->getOutEdges()[j]->getWeight()<<" ";
            }
            cout<<endl;
        }
        cout<<endl;
    }

};
  #endif
```

Instructions to Compile and Run your Program

Please follow the commands below to run the program.

$ g++ GraphTest.cpp

$ ./a.out

Then input source and destination.

Input and Output Specifications

There is not input limit now.

Logical Exceptions

No logical error has been found in testing.

C++ object oriented or generic programming features

In this lab, object oriented programming features is used. The classes in this lab are shown in Program Organization and Description of Classes section of this lab report.

Analysis

Since each build up a MQP with binary heap and n elements takes O(n*log n). So even with the locator, the running time for Dijikstra's algorithm is still O(n*log n) for sparse graph. However, by using the locator, the findMin and decrease key

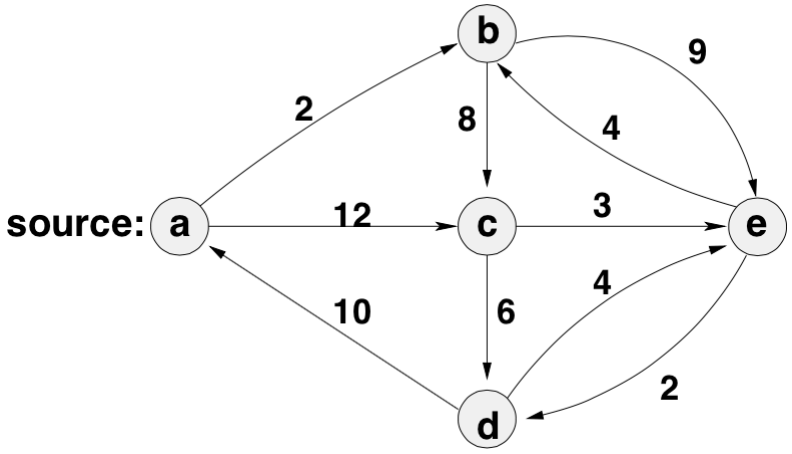
function's running time is constant. Thus the running time is O(1).

I recycled the Priority Queue ADT from the last lab. I used the MPQ implemented with binary heap and locator in this lab to initialize the priority queue to store the vertices. It takes less time to implement the Dijkstra's algorithm.

Applications

1. Walmart goods transportation plan. Find the shortest path.
2. Build the national highway network that connects big cities.
3. Air company flight schedule.

Bonus



iter	Pi[v]	d[a]	d[b]	d[c]	d[d]	d[e]
0	-	0	∞	∞	∞	∞
1	a	0	2(a)	12(a)	∞	∞
2	b	0	2(a)	10(b)	∞	11(b)
3	c	0	2(a)	10(b)	16(c)	11(b)
4	d	0	2(a)	10(b)	13(e)	11(b)
5	e	0	2(a)	10(b)	13(e)	11(b)