UCSC Silicon Valley Extension

Project

Java Programming, Comprehensive

Radhika Grover

Total points: 40

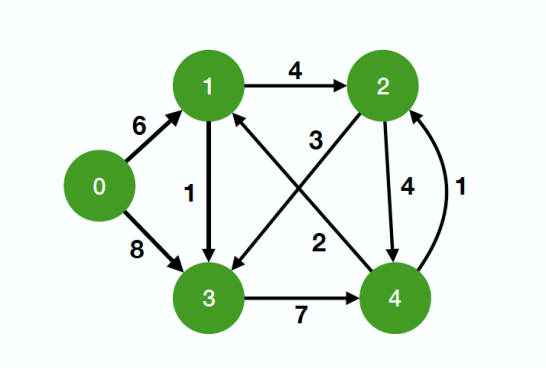
You have been hired to develop *Top Flight* a new flight scheduling application for an airline

company that will help users with their travel itinerary.
The application will be used for each of the following queries:

(a) Find the path with the lowest cost to the destination airport.
(b) Find the path to the destination airport that passes through a given airport and has the lowest cost.
(c) Find the path to destination city with the lowest cost and the smallest number of layovers.

A graph will be used to model the airline network. Each vertex represents an airport location and each edge represents a connection between two airports. An edge has a weight that represents the cost of flying that connection. The total cost of the journey is the sum of the costs of all connections.

For example, suppose that the following graph represents an airline network, each node is an airport, and the weight of each edge is the cost of traveling on that connection in hundreds of dollars. For example, to travel from airport #0 to airport #1, the cost is $600. Suppose that we want to find the path with lowest cost from airport #0 to airport #4. For part (a), one solution is {0, 1, 3, 4} with a cost of $1400. Suppose the user would like to travel through node 2. Then for part (b), the solution is {0, 1, 2, 4} for a total cost of $1400. In part (c), a new constraint is added in which the lowest cost path with the smallest number of edges has to be selected. Here both paths from 0 to 4 have 3 edges, so either one can be selected; otherwise, select the one with fewest edges.



An airline network where each node represents an airport and an edge represents a connection

Implementation details

(a) Develop a data structure called `Graph` to store information about cities. You can implement the graph using an adjacency list or an adjacency matrix. This contains the following methods:

`Graph *createGraph(int numVertices, string graph_type)` creates a graph with the specified number of vertices (numVertices) and the given type (graph_type is directed or undirected).

`void addNewEdge(Graph *graph, int u, int v, int _weight)` adds a new edge connecting vertices u and v to the graph

`void print(Graph *graph)` prints out the graph

`ListNode* getAdjListHead(Graph *graph, int index)` /* if using an adjacency list */

`int getNumberOfVertices(Graph *graph)` returns the number of vertices in the graph

Test file format

```
10    # number of test cases
directed  # directed graph
5         # starting vertex
0         # destination vertex
9         # number of edges
0 1 6     # edge connects vertices 0 and 1 with weight 6
0 3 8     # edge connects vertices 0 and 3 with weight 8
1 2 4     # edge connects vertices 1 and 2 with weight 4
1 3 1     # edge connects vertices 1 and 3 with weight 1
```

Deliverables:

Your program will be graded for correctness and efficiency. The source code should be documented and indented correctly.

1. [5 points] Submit the source code for the Graph data structure.

2. [20 points] Submit the source for each of the programs in parts (a) to (c) and any additional data structures in **separate text files**.

3. [10 points] Determine the recurrence relation for your algorithm in part (c). Solve the recurrence relation and obtain the worst-case running time of your algorithm. Show your work.

4. [5 points] Obtain the execution time of your program.