Assignment Four

Objectives

- Understand how to represent graphs using adjacency lists
- Understand how to traverse graphs
- Understand how to find a shortest path

Admin

Marks 10 marks. Marking is based on the correctness and efficiency of your code. Your code must be well commented.
Group? This assignment is completed individually.
Due Time 23:59:59pm Wed 1 May 2019.
Late Submissions Late submissions will not be accepted!

In this assignment, you will implement a graph ADT (Abstract Data Type) with several functions. A graph G considered in this assignment consists of a set V of vertices and a set E of edges, where each vertex in V is a point on a Cartesian plane, and each edge is a line segment between the two points in V. Consequently, G is an undirected graph. Each point has a x-coordinate and a y-coordinate. We assume that the x-coordinate and the y-coordinate of each point are integers.

The distance of between two points $v_1=(x_1, y_1)$ and $v_2=(x_2, y_2)$ is $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. The length of an edge is the distance between its two end points. Therefore, each edge has an implicit weight which is its edge length.

Given a path in a graph, its path length is the sum of lengths of all its edges. A path between two vertices is a shortest path if it has the minimum path length among all the paths between the two vertices.

Given two vertices u and v in a graph, u is reachable from v if there is a path from u to v.

Basic types are provided as follows:

```
// A vertex is a 2D point
typedef struct Vertex {
  int x;   // x-coordinate
  int y;  // y-coordinate
} Vertex;

// each edge is a pair of vertices (end-points)
typedef struct Edge {
  Vertex *p1;   // first end point
  Vertex *p2;  // second end point
} Edge;

// A vertex node stores a vertex and other information, and you need to expand this type
typedef struct VertexNode {
 Vertex *v;
```

```
} VertexNode;

typedef struct GraphRep { // graph header
  VertexNode *vertices;  // an array of vertices or a linked list of vertices
  int   nV;    // #vertices
  int   nE;    // #edges
} GraphRep;

typedef struct GraphRep *Graph;
```

The above types serve as a starting point only. You can revise them and add more types.

You need to implement the following functions:

- Graph CreateEmptyGraph(). This function creates an empty graph and returns it.
- int InsertEdge(Graph g, Edge *e). This function does the following task. Check if the edge e is in the graph g. If e is not in g, insert e into g and return 1. Otherwise, return 0.
- void DeleteEdge(Graph g, Edge *e). This function deletes the edge e from the graph g. If e is not in g, it does nothing. After deleting the edge e, this function also deletes any isolated end vertex of e. An isolated vertex is a vertex without any edge between this vertex and any other vertex.
- void ReachableVertices(Graph g, Vertex *v). This function finds all the vertices reachable from the vertex v in g and prints them in any order on the screen. A vertex u is reachable from a vertex v if there is a path from v to u. In the output, each vertex is displayed as a pair (x', y'), where x' and y' are its x-coordinate and y-coordinate, and two adjacent vertices are separated by a comma (,). If no vertex is reachable from v, nothing will be printed. If v is not a vertex of g, this function does nothing.
- void ShortestPath(Graph g, Vertex *u, Vertex *v). This function finds the shortest path between the vertex u and the vertex v, and print all the vertices of the shortest path in their order on the shortest path from the vertex u to the vertex v in the form of (x1, y1), …, (x2, y2), where the first and second element of each pair are the x-coordinate and y-coordinate of the corresponding vertex. If either u or v is not a vertex of g, this function does nothing.
- void FreeGraph(Graph g). This function frees the heap space occupied by the graph g.
- void ShowGraph(Graph g). This function prints each edge of g once in breadth-first order. Your breadth-first search algorithm can pick any vertex as the first vertex to be visited in the breadth-first search. In the output, each vertex is displayed as a pair (x', y'), where x' and y' are its x-coordinate and y-coordinate, and each edge (x1,y1)-(x2,y2) is displayed as (x1,y1),(x2,y2), and two adjacent edges are separated by a white space.

Time complexity analysis

You need to include the time complexity analysis of each function as comments in your program. Try your best to make each function time-efficient. Any time complexity that goes against the best algorithm you have learned in this course will receive some penalty. For example, the time complexity of your ShortestPath() function should not be higher than O((m+n)log n), where m and n are the number of edges and the number of vertices of graph g. There is no specific requirement on the space complexity. However, you need to make your functions as space efficient as possible.

How to submit your code?

a. Go to the Assignment 4 section
b. Click on Assignment Specifications
c. Click on Make Submission
d. Submit your MyGraph.c file that contains all the code.

Plagiarism

This is an individual assignment. Each student will have to develop their own solution without help from other people. In particular, it is not permitted to exchange code or pseudocode. You are not allowed to use code developed by persons other than yourself. All work submitted for assessment must be entirely your own work. We regard unacknowledged copying of material, in whole or part, as an extremely serious offence. For further information, see the Course Information.