SSZG519 Data Structures & Algorithms Design

EC-1 Assignment - 2

Weightage 7% Max marks : 28M

IMPORTANT:

(a) You must only upload the source files for the 2 problems.
(b) All code related to problem 1 must be inside a folder named q1
(c) All code related to problem 2 must be inside a folder named q2
(d) You must provide a text file (.txt) named **README** clearly indicating how to compile and execute your code inside the 2 directories.
(e) The names of the ADTs and functions suggested must be maintained as it is.

Note: C-like function prototypes are given here for your convenience. End Note.

Problem Definition:

Given an arrangement of balls on 2-D Euclidean plane (i.e a flat surface), you have to assign a color to each ball such that no two adjacent balls are of the same color. A greedy approach can be used to reduce the number of colors required.

Question:

Model this as a graph problem. [Hint: Balls become vertices, adjacency relation is modeled by edges, and each vertex has a unique identification number and a color.]. <u>Write a program that finds the number of colors required and outputs the balls (unique ids) along with their colors.</u>. Note that to solve this problem, all balls and their neighbors must be inspected.

Use adjacency lists to represent the graph.

The input to this program is a file containing the number of balls in the first line followed by the list of adjacencies – one per line: e.g. an input line containing

x,y

denotes that balls **x** and **y** are neighbors. Here **x** and **y** denote the unique ids of the two balls.

A simple greedy algorithm for this color assignment problem is as follows:

I. Sort all the vertices in the graph on the basis of their degrees [This sorting should be done in-place on the array of adjacency lists.]. Assume colors are ordered c1, c2, …

II. Let u be the un-colored vertex with the smallest degree. [Break ties in favor of the vertex with the smaller id]

   a. Assign first color c_i in the list of colors to u such that

      color(u) $\leftarrow$ c_i where c_i != color(v_j) for any vertex v_j in the adjacency list of u.

III. Repeat step II until all vertices are colored.

Implement your solution using a Graph ADT that supports the following interfaces:

a) Graph createGraph() : Creates an empty graph.

b) Graph addEdge(Graph g, Vertex v1, Vertex v2): adds an edge from vertex v1 to vertex v2 to the graph g. If a new vertex is found, then an entry has to be added in the adjacency list

c) Iterator getNeighbors(Graph, Vertex): gets a list of neighbors of the vertex.

d) Graph sortGraphbyDegree(Graph) : sorts the adjacency list based on degree of the vertices. The vertex with smallest degree will appear first, and the vertex with the largest degree will appear last. If two vertices are having the same degree, then their order of appearance will not change.

e) Color chooseColor(Graph, Vertex): returns the first color in the list of colors that satisfies the condition mentioned in step (2) above.

f) int assignColors(Graph) : invokes chooseColor vertex by vertex and stores the chosen color in the corresponding vertex. This function returns the number of colors used.

g) printGraph(Graph , num_colors_used, file) : prints the number of colors used, num_colors_used, in the first line of the output file. It then prints the graph into the file using the following format:

`(vertexid,color):v1,v2,v3,vn`

where **`vertexid`** is the unique id of the vertex, **`color`** is the color assigned to the vertex, and **`v1,v2,…,vn`** correspond the unique identification numbers of the vertices that are adjacent to the current vertex.

Data structures Used:

`Graph`: This is a dynamic array, such that each entry in the array contains a pointer to vertex v_i, the degree of v_i and a list of neighbors of v_i.

`Vertex`: Each vertex contains the unique identification number of the vertex, its color.

`List`: This is for the list of neighbors, such that each entry in the list corresponding to vertex v_i consists of a pointer to vertex v_j, $\forall$ $v_j \in$ neighborhood(v_i)

Steps to perform:

1. Write the relevant Header files for an adjacency list representation of Graph
2. Write a driver file that reads an input file containing the edges in the graph and prints the graph after coloring the balls. This driver uses the adjacency list for graph representation.
    a. The driver takes the name of the input file and output file as command line parameters.
    b. From the file, find number of nodes involved.
    c. For each line in the input file corresponding to an edge
        i. add the edge using call to addEdge().
    d. The driver must then invoke function sortGraphbyDegree() to sort the vertices in the increasing order of their degrees.
    e. Invoke assignColors to assign colours to each vertex.
    f. Print the number of colours used and the resultant graph into the output file using the call to the function printGraph(). If no output file is mentioned in the command-line, then print the graph into the screen.
3. Write the code for the functions (a) - (g)