CS2110 Computer Programming Lab, Monsoon 2017

Tasks for Week 1

Experiment Set 4: Graphs

Tasks for Day 1.

1.1 Graph representations

For most of the algorithms that we do in this set, we will represent graphs using adjacency matrices, because that's a very computer friendly format. But we would also like to see these graphs in a human way - as some one will draw it on paper. For this we will first convert the adjacency matrix to what is know as "DOT language file" and then view it using a simple tool called xdot.

You may want to read Section 10.3 of Rossen's book on adjacency matrices, and this tutorial on DOT language.

You can use the following five graphs (represented as adjacency matrices) as test cases for your tasks: Test Graphs (it contains 8 graphs. Use tar -xzvf TestGraphs.tgz to extract)

You can also see the dot file for Graph01, to serve as an example.

Task 1. (Making a DOT)

Write a C program to read the adjacency matrix of a simple undirected graph, stored in a text file and generate a dot file for the same. Run the program on the test cases given and view the output using xdot.

Upload the program to moodle as adj2dot.c. Do not forget to comment and align the code, you will reuse it yourself a lot.

Task 2. (Path finder)

1. Learn about Breadth First Search
2. Write a C program, which given a graph and two nodes in that graph tests whether these two nodes are connected.
   User inputs: Name of the file containing adjacency matrix of the graph (string), Name of two nodes (integers)
   You can reuse the functions you have written in earlier sessions for implementing the queue.
   Implement the BFS as a function (accepting an array and two indices as input) so that you can reuse it later.
   (Hints.
   1. Assume the node labels to be 0 to n-1.

    2. Make one we Boolean array "**visited**" of size n to store the visited-status of each node. That is, this array should be initialised to 0 and when a node i is visited by your algorithm for the first time, you should make visited[i] to be 1.
    3. Make one queue "**toExplore**" the visited nodes which are to be explored next. That is every time you visit and unvisited node, after you mark visited[j] = 1, you should enqueue j to the toExplore.
3. Modify the above program so that in addition to testing whether the given two nodes are connected, it also prints out one of the shortest paths between them (as a sequence of vertices) and reports the distance between the two given nodes.
4. Modify the above program so that in addition to printing out the sequence of vertices, it writes a DOT file containing the original graph, in which all the edges of this path are marked in red colour. For an example of how to do it see this file: Graph01WithARedPath.dot

Upload the program to moodle as path_finder.c

Task 3. (Counting pieces)

1. Write a C program which uses BFS to count the number of connected components in a graph.
   User inputs: Name of the file containing adjacency matrix of the graph (string)
2. [Optional] Modify the above program so that it generates a dot file for the graph, in which the edges of each connected component is shown in a different colour.
3. Modify the first program so that it generates a dot file for the graph, in which the edges of the **BFS tree** you trace is shown in a red colour. You include an edge into the BFS tree, only if that edge resulted in finding a previously unvisited vertex.

Upload the program to moodle as counting_pieces.c

Task 4. (How far?)

1. Write a C program to find the diameter of a given graph, if it is connected.
   User inputs: Name of the file containing adjacency matrix of the graph (string)
2. Modify the first program so that it generates a dot file for the graph, in which the edges of a **longest** path in the graph is shown in a red colour.
   [Update on 7-Oct-17: Actually I had in mind the "**longest shortest** path" - that is a diametrical path. Finding a longest path is a more difficult problem.]

Upload the program to moodle as farthest.c

Task 5 (Eulerian path)

1. Write a C program to find an Eulerian path in a given graph, if it has one.
   User inputs: Name of the file containing adjacency matrix of the graph (string)
   Output: Print the sequence of vertices in the Eulerian path

Upload the program to moodle as eulerian.c

Last modified: Saturday, 7 October 2017, 10:22 AM