CS 3510: Algorithms

Assignment 3

Assignment

Problems identified by x.y(z) denote the problem “y”, in chapter “x” of the textbook, with part “z”. If “z” is not noted, then the entire problem is required.

Assignment 3a

- 3.28(a,b) Show all satisfying truth assignments, and show why your formula is unsatisfiable.
- 3.11 Give an algorithm, argue correctness, analyze runtime.
- Complete the tasks for Programming Assignment `Graph` Class.

Assignment 3b

- 3.2 (b) DFS with pre/post numbers and edge type identification.
- 3.12 Prove means give a convincing argument. Counter example means a single concrete example that proves the statement is false.
- 3.22 The algorithm is a simple extension of one in the chapter.
- 3.28 (part c) This means draw the graphs that result from the example and your example.
- Complete the tasks for Programming Assignment `dfs`.

Assignment 3c

- 3.4 (i)(a,b,c,d) (i) is the first graph in the problem.
- 3.28 (d) The hint should give it away.
- Complete the tasks for Programming Assignment `pre/post-visit`.

Assignment 3d

- 3.15 (a) Describe a graph problem that represents this problem, and the graph algorithm that will answer the question in linear time.
- 3.15 (b) Describe a graph problem that represents this problem, and the graph algorithm that will answer the question in linear time.
- 3.28(e,f) For f, prove that the runtime is linear.
- Complete the tasks for Programming Assignment Search.

Assignment 3z, Due Never (optional)

- 3.18 Give the preprocessing algorithm, and the query algorithm. Analyze their complexity.
- 3.24 The algorithm is a simple extension of one in the chapter.

Programming Assignment `Graph` Class

- Create and work in the directory `03-graph`.
- Implement a `Graph` class.
- You choose whether to use an adjacency-matrix or an adjacency-list.
- Nodes identifiers are unsigned integers from `1` to `n`, inclusive.
- `Graph()` constructor
- `void setNodeCount(unsigned int count);` sets the number of nodes in the graph, updating the size of the adjacency matrix/list, and removing all edges.
- `void addUEdge(unsigned int u, unsigned int v);` adds an unweighted, undirected edge between `u` and `v`.
- `void addDEdge(unsigned int u, unsigned int v);` adds an unweighted, directed edge from `u` to `v`.
- `void addUEdge(unsigned int u, unsigned int v, double w);` adds a weighted, undirected edge between `u` and `v`.
- `void addDEdge(unsigned int u, unsigned int v, double w);` adds a weighted, directed edge from `u` to `v`.
- `unsigned int getNodeCount() const;` returns the number of nodes in the graph.
- `double getEdge(unsigned int u, unsigned int v) const;` returns the weight of the edge from `u` to `v`. If no edge exists, returns `0.0`.
- Test the code by writing sample code that stores the graphs from Figures 3.1 and 3.2, and then displays a table of the edges by fetching them from the graph.

Programming Assignment `dfs`

- Add to the `Graph` class.
- Note that you will likely need to add data members for visited data.

- `void explore(unsigned int u);` Does the explore algorithm described in Figure 3.3. (except pre/post visit calls)
- `void dfs();` Does the depth-first search algorithm described in Figure 3.5.
- `void markAllUnvisited();` marks all nodes as unvisited.
- `int getVisited(unsigned int u) const;` returns `1` if `u` has been visited, `0` if `u` has not been visited.
- Be sure that the visited data is resized correctly when necessary.
- Add to the test programs for each of the 2 sample graphs.
- Test the code by running explore on various vertices.
- Test the code by running dfs on the sample graphs.

Programming Assignment `pre/post-visit`

- Add to the `Graph` class.
- Note you will need to add data members to support the bookkeeping.
- `void previsit(unsigned int u);` marks the connected component number, and previsit number.
- `void postvisit(unsigned int u);` marks postvisit number.
- `unsigned int getConnectedComponent(unsigned int u) const;` returns the node's connected component number.
- `unsigned int getPrevisit(unsigned int u) const;` returns the node's previsit number.
- `unsigned int getPostvisit(unsigned int u) const;` returns the node's postvisit number.
- Be sure that the data are resized correctly when necessary.
- Add to `explore` to correctly use these methods.
- Add to `dfs` to correctly initialize the bookkeeping data.
- Test the code by running dfs on the sample graphs, and displaying the pre/post/cc data for each node.

Programming Assignment Search

- Use your `Graph` class to evaluate the graphs available in `/usr/citlocal/cs3510/connectivity-graphs` on the department computers.
- The graph filenames have the format `cgraph-n-s.txt`, where `n` is the number of vertices and `s` is a graph number. For example, `cgraph-20000-2.txt` is the second graph with `20000` vertices.
- Each file has a first line with the number of vertices, followed one line per edge. An edge description line has the format `u v w`, where `u` and `v` are integers, and `w` is a floating point number. `u` and `v` are vertex numbers. `w` is the edge weight.
- All edges are undirected.
- Any edges not listed in the file do not exist.
- The vertices are numbered 1 - `n`.
- For each of the graphs with 10000 vertices, compute the connected component numbers using `dfs`. For each connected component, sum the node numbers. Display the sums from smallest to largest sum, on a single line, with a space between each number.
- In your repository, store the output for each graph in a file with the same name as the graph, except, replace the `.txt` with `.out`.
- For example, `cgraph-10-3.txt` would be store a line with `13 17 25` into the file `cgraph-10-3.out`.

Submission

- Submit you solutions by the due date and time. For written problems, your work and answers as a PDF to Canvas. For code, submit the source code to the class git repository. For tables and graphs, submit a PDF to Canvas.