CS 6301.502. Implementation of advanced data structures and algorithms

Fall 2017

Short Project 8: Shortest paths, Enumeration

Fri, Oct 20, 2017

Version 1.0: Initial description (Fri, Oct 20).

Due: 11:59 PM, Sun, Oct 29.

Solve as many problems as you wish. Maximum score: 50.

Create a ShortestPath class for all the shortest path algorithms. The

following constructor is common to all algorithms:

ShortestPath(Graph g, Vertex s)

It initializes the class fields, with s as the source node for

shortest paths. The following methods in Q1-6 are the public methods

of the ShortestPath class.

1. [20 points] Implement BFS:

public void bfs() { ... }

2. [20 points] Implement DAG shortest paths:

public void dagShortestPaths() { ... }

Implement this algorithm without duplicating the DFS code for

finding topological order into the ShortestPath class.

Reuse your DFS code from previous projects.

3. [20 points] Implement Dijkstra's algorithm, reusing your code for Indexed heaps:

public void dijkstra() { ... }

4. [20 points] Implement Bellman-Ford algorithm:

public boolean bellmanFord() { ... }

5. [30 points]

Given a graph, apply the fastest algorithm for shortest paths on

the given graph. If all edges of G have the same positive weight,

apply BFS. Otherwise, if G is a DAG, apply DAG-shortest-paths

algorithm. Otherwise, if G has no negative-weight edges, then

apply Dijkstra's algorithm. Otherwise, call Bellman-Ford

algorithm. If the problem is not solvable because the graph has a

negative cycle, return false.

public boolean fastestShortestPaths() { ... }

6. [30 points]

Use the BFS implementation in Q1 to find an odd-length cycle in a

given non-bipartite graph. Given a graph, find an odd-length cycle

and return it. If the graph is bipartite, return null.

Do not merge this algorithm into BFS code. Write code that uses

output of BFS and finds an odd-length cycle. Make sure that the

edges of the cycle are in proper order.

public List<Edge> findOddCycle() { ... }

7. [20 points]

Implement permutation and combination algorithms nPk and nCk.

Use a VERBOSE flag to decide if the output is just the number

of permutations or combinations visited (VERBOSE = 0), or, a

complete listing.

8. [20 points]

Implement Knuth's L algorithm.

9. [20 points]

Implement non-recursive version of Heap's algorithm for generating

all n! permutations. See Wikipedia page for the algorithm.

https://en.wikipedia.org/wiki/Heap%27s\_algorithm