Congratulations! You passed!

Grade received 100% **Latest Submission Grade** 100% **To pass** 80% or higher

Go to next item

1. Which of these algorithms can be used to count the number of connected components in a graph? **1 / 1 point**

- ○ Count the number of times a breadth first traversal is started on every vertex of a graph that has not been visited by a previous breadth first traversal.
- ○ Count the number of times a depth first traversal is started on every vertex of a graph that has not been visited by a previous breadth first traversal.
- [✓ All of the above]
- ○ None of the above

✓ **Correct**
Both breadth first and depth first searches visit every vertex connected to a starting vertex by a path of edges, and thus visit every vertex in a connected component of the graph.

2. Which elements encountered by a breadth first search can be used to detect a cycle in the graph? **1 / 1 point**

- ○ Unexplored edges to unexplored vertices that remain so after completion of the breadth first search.
- ○ Unexplored vertices that have been encountered by the traversal of a previously unexplored edge.
- [✓ Previously visited vertices that have been encountered again via a previously unexplored edge.]

- ○ Discovered edges that were previously unexplored by the traversal have been added to the breadth-first traversal.

✓ **Correct**
A breadth first traversal returns a spanning tree of each connected component of the graph. Any edge that is not part of the breadth first search (e.g. not marked discovered) will connect one portion of the tree to another forming a cycle. Thus all unexplored edges, including ones ignored because they reach a previously visited vertex will create a cycle if added to the breadth first search.

3. A breadth first traversal starting at vertex v1 of a graph can be used to find which ones of the following? **1 / 1 point**

- ● The shortest path (in terms of # of edges) between vertex v1 and any other vertex in the graph.
- ○ The shortest path (in terms of # of edges) between any two vertices in the graph.
- ○ All of the above.
- ○ None of the above.

✓ **Correct**
A breadth first search adds vertices connected to the current frontier of vertices using a queue, so that all vertices a given number of edges away from the start vertex must have been processed through the queue before the next wave of vertices can be processed. Since vertices are added in this edge-length order, the breadth first traversal finds the shortest path from the start vertex to any vertex, but this is only true for the start vertex.

4. Which traversal method has a better run time complexity to visit every vertex in a graph? **1 / 1 point**

- ○ Breadth First Traversal
- ○ Depth First Traversal
- ● Both have the same run time complexity.
- ○ Neither traversal method will necessarily visit every vertex in a graph.

✓ **Correct**
Both options run in time O(n + m) for n vertices and m edges.

5. The breadth first traversal of a connected graph returns a spanning tree for that graph that contains every vertex. If the graph has weighted edges, which of the following modifications is the simplest that produces a minimum spanning tree for the graph of weighted edges. **1 / 1 point**

- ○ An ordinary breadth first traversal is run from each vertex (as its start vertex) and the resulting spanning tree with the least total weight is the minimum spanning tree.
- ○ The queue is replaced by a priority queue that keeps track of the total weight encountered by the current traversal plus each of the edges that connects a vertex to the current breadth first traversal.
- [✓ The queue is replaced by a priority queue that keeps track of the least-weight edge that connects a vertex to the current breadth first traversal.]
- ○ No modification is necessary because a breadth first traversal always returns a minimum spanning tree.

✓ **Correct**
A minimum spanning tree for a weighted graph can be found through a greedy breadth-first algorithm that simply chooses from the entire queue the least weight edge to add.

6. True of false: a connected directed graph with no cycles is a tree. **1 / 1 point**

- ○ True
- [✓ False]

✓ **Correct**
For example, a directed graph such as A -> B, A --> C, B --> D and C --> D is connected and has no cycle, but is not a tree because there are multiple paths from vertex A to vertex D.

7. For which situation described here can Dijkstra's algorithm sometimes fail to produce a shortest path? You would want to avoid using Dijkstra's algorithm in this situation. **1 / 1 point**

- [✓ A connected graph where some of the edge weights are negative and some have weight zero.]
- A connected graph where some of the edge weights are zero and the rest are positive.
- A connected graph where there are multiple paths that have the same overall path cost (distance), and all of the edge weights are non-negative.
- A connected graph where all of the edges have the same positive weight.

✓ **Correct**

There is nothing wrong with the edge weights of zero, but the negative weights are a problem. Dijkstra's algorithm, without modifications, achieves its fast running time by making certain assumptions about which paths are best. If it encounters an edge with negative weight, the assumptions fail, and it may not correctly identify the shortest path.

Some people modify Dijkstra's algorithm to iterate when negative edge weights are encountered, to make corrections. However, this causes the algorithm to run very slowly in the worst case, and it's not part of the classical algorithm.

(As a separate note, if there is any graph where a cycle has weights that sum to a negative value overall, then other shortest path algorithms can also fail to find a shortest path even if they are able to handle negative edge weights in some cases. That's because the graph may have paths with infinitely negative weight.)

8. Which of the following is a true statement about Dijkstra's algorithm? Assume edge weights (if any) are non-negative. **1 / 1 point**

- Dijkstra's algorithm finds the shortest unweighted path, if it exists, between a start vertex and any other vertex, but only for an undirected graph.
- Dijkstra's algorithm finds the shortest weighted path, if it exists, between a start vertex and any other vertices, but only for an undirected graph.

- [✓ Dijkstra's algorithm finds the shortest weighted path, if it exists, between a start vertex and any other vertices in a directed graph.]
- Dijkstra's algorithm finds the shortest weighted path, if it exists, between all pairs of vertices in a directed connected graph.

Correct
Dijkstra's algorithm indeed works with directed edges, only following an edge along its direction from a start vertex to an end vertex.

9. Which of the following is the optimal run time complexity to find the shortest path, if it exists, from a vertex to all of the other vertices in a weighted, directed graph of n vertices and m edges. **1 / 1 point**

- [✓ O(m + n lg n)]
- O(m + n)
- O(n)
- O(m + lg n)

Correct
This is the running time for Dijkstra's algorithm which is optimal.

10. Suppose you are given an undirected simple graph with unweighted edges, and for a particular specification of three vertices u, v, and w, you want to find the shortest path from u to w that goes through v as a landmark. What is the most efficient method that can find this? **1 / 1 point**

- Two runs of Dijkstra's algorithm, first from u and then from v.
- A single run of Dijkstra's algorithm from u.
- [✓ A single run of breadth-first search from v.]
- Three runs of breadth-first search: once each from u, v, and w.

Correct

A single breadth-first search from the landmark vertex finds the shortest paths from it to the start vertex and the end vertex, and since the edges are undirected, their combination is the shortest path from start to end that also visits the landmark.

It's not necessary to use Dijkstra's algorithm in this case since the edges are unweighted.