**Test 3**

Name: Marissa Norris

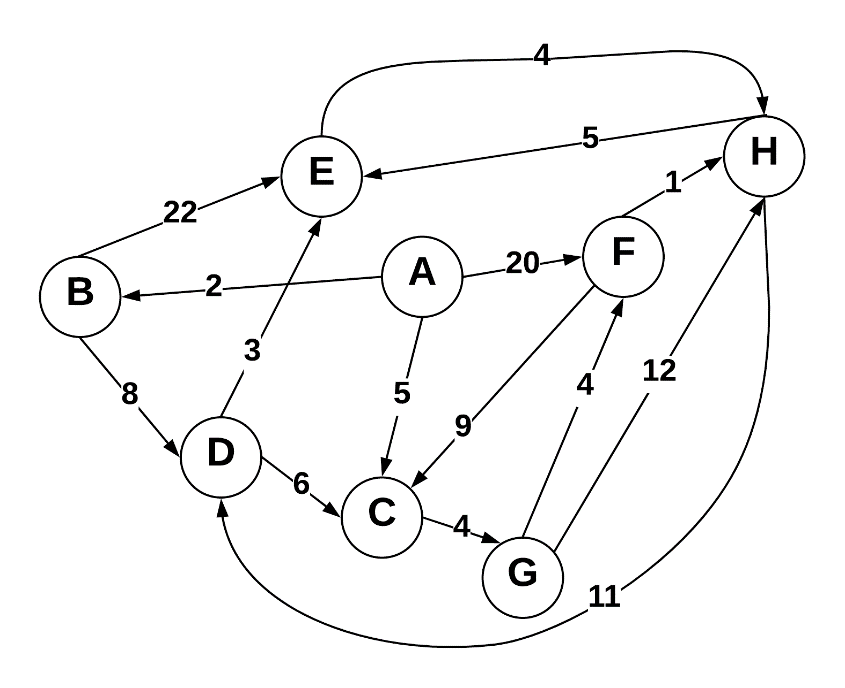
* Everything you turn in must be digitally created.
* No handwriting (except for signature below).
* You must work alone.
* Sharing of answers will result in a 0 on the exam, and possible F in the course.
* Send me your digitally created exam by Friday, May 4th by Midnight on a private slack message.
* Bring your printed signed copy by Monday Morning 10:00 am to my office.

|  |  |  |
| --- | --- | --- |
| Question | Possible | Score |
| 1 | 10 |  |
| 2 | 10 |  |
| 3 | 10 |  |
| 4 | 10 |  |
| 5 | 10 |  |
| 6 | 10 |  |
| 7 | 10 |  |
| 8 | 10 |  |
| 9 | 10 |  |
| 10 | Bonus |  |
| Total: |  |  |

By signing this, your saying “I worked alone and did not plagiarize”:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1) Dijkstra’s Algorithm**



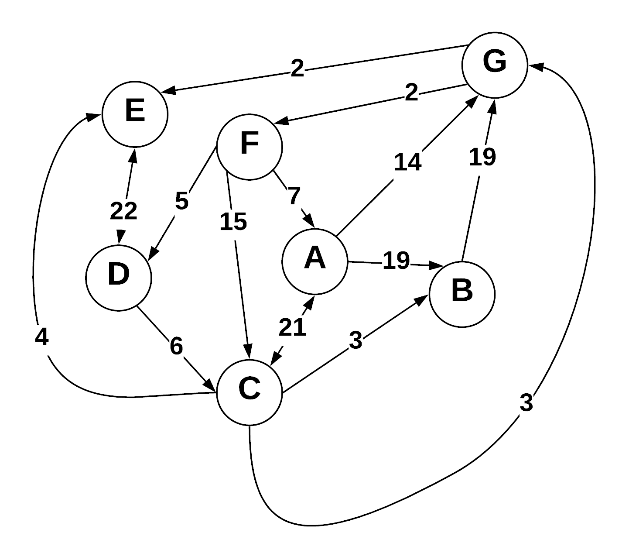
Use Dijkstra’s algorithm to compute the shortest paths from vertex A to every other vertex. Show your work in the space provided below. As the algorithm proceeds, cross out old values and write in new ones, from left to right in each cell. If during your algorithm two unvisited vertices have the same distance, use alphabetical order to determine which one is selected first. Also list the vertices in the order which Dijkstra's algorithm marks them as discovered.

Vertices in Order of Discovery:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | D | E | H |  |  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Vertex | Known | Cost | Previous |
| A |  | 0 |  |
| B |  | 2 | A |
| C |  |  |  |
| D |  | 10 | B |
| E |  | 13 | D |
| F |  |  |  |
| G |  |  |  |
| H |  | 17 | E |

**2) Prims Algorithm**



Step through Prim’s algorithm to calculate a minimum spanning tree starting from vertex *G.* Show your steps in the table below. As the algorithm proceeds, cross out old values and write in new ones, from left to right in each cell. If during your algorithm two unvisited vertices have the same distance, use alphabetical order to determine which one is selected first. Also list the vertices in the order which Prims algorithm discovers them.

The Prim’s algorithm cannot work on directed graphs. It fails because this algorithm assumes that all vertices are reachable by each other. Therefore, I had to answer the question as if the graph was undirected. For instance, if I was to work through the graph as directed, I would covered few vertices and encounter errors. Starting at G, the only 2 options would be E or F, eliminating the 3 other possible edges. If E was taken then that would lead to a dead end because there are no edges leaving E. Furthermore, if F was taken then it would result in a cycle (G, F, D, C, B, and G) which cannot happen in Prim’s Algorithm. When the graph is undirected, I am able to traverse through all the vertices, accomplishing Prim’s Algorithm.

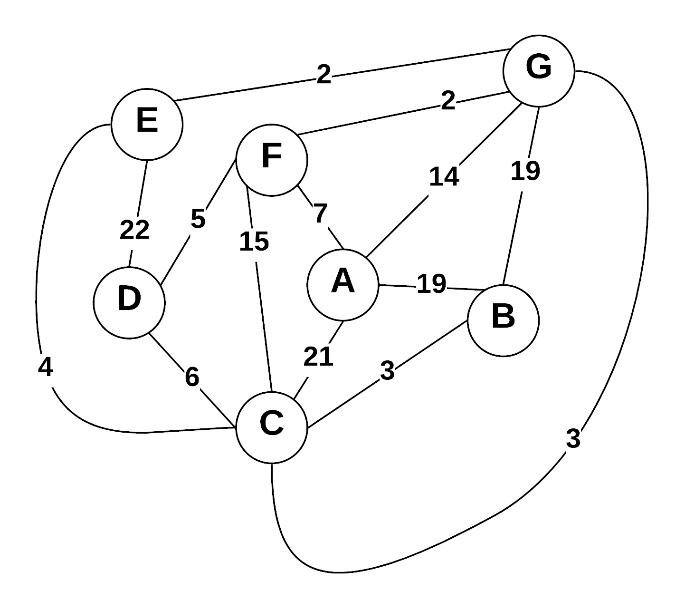
Vertices in Order of Discovery:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| G | E | C | B | A | F | D |  |  |

* S = Vertices in spanning tree
* U = ! S (vertices not in S)
* Cut = edges going across cut listed alphabetically: (A B) , (C D) , etc.

|  |  |  |
| --- | --- | --- |
| *S (spanning tree)* | *U* | Cut (alphabetize) |
| G, E, F, A, B, C | D | (G,E), (G,F), (G,A), (G,B), (G,C) |
| E,G,D, C | F,A,B | (E,G), (E,D), (E,C) |
| C, E, D, F, A, B, G |  | (C,E), (C,D), (C,F), (C,A), (C,B), (C,G) |
| B, C, A, G | E,D,F | (B,C), (B,A), (B,G) |
| A, C, B, G, F | E,D | (A,C), (A,B), (A,G), (A,F) |
| F, G, A, C, D | E,C,B | (F,G), (F,A), (F,C), (F,D) |
| D, E, F, C | A,B,G | (D,C), (D,F),(D,E) |

**3) Kruskel’s Algorithm**



Use Kruskal’s algorithm to calculate a minimum spanning tree of the graph. Show your steps in the table below, including the disjoint sets at each iteration. If you can select two edges with the same weight, select the edge that would come alphabetically last (e.g., select E—F before B—C. Also, select A—F before A—B).

* Edge Added: put edges added to MST marked as (A B), (E G), etc.
* Edge Cost: weight of edge added
* Running cost is total weight of spanning tree at the point another edge is added.
* Disjoint sets starts as: (A) (B) (C) (D) (E) (F) (G) , and as edges are added => (A) (B C) (D) (E) (F) (G)

|  |  |  |  |
| --- | --- | --- | --- |
| Edge Added | Edge Cost | Running Cost | Disjoint Sets |
| (G F) | 2 | 2 | (A)(B)(C)(D)(E)(F G) |
| (G E) | 2 | 4 | (A)(B)(C)(D)(F G E) |
| (G C) | 3 | 7 | (A)(B)(D)(F G E C) |
| (C B) | 3 | 10 | (A)(D)( F G E C B) |
| (F D) | 5 | 15 | (A)( F G E C B D) |
| (F A) | 7 | 22 | ( F G E C B D A) |
|  |  |  |  |
|  |  |  |  |

**4) Prims Vs Kruskels**

Explain why Prim’s algorithm is better for dense graphs, while Kruskal’s algorithm is better for sparse graphs. What data structures are used to represent each?

Prim’s algorithm is better for dense graphs or in other words, graphs with many edges. It can be made to run in linear time to make it faster for processing. First must consider the time complexity. Prim’s algorithm has time complexity O(m+nlog(n), where we represent the number of edges by m and the number vertices by n. If the data structure of a Fibonacci heap is used then Prim’s algorithm would be O(m) which is the desired time complexity.

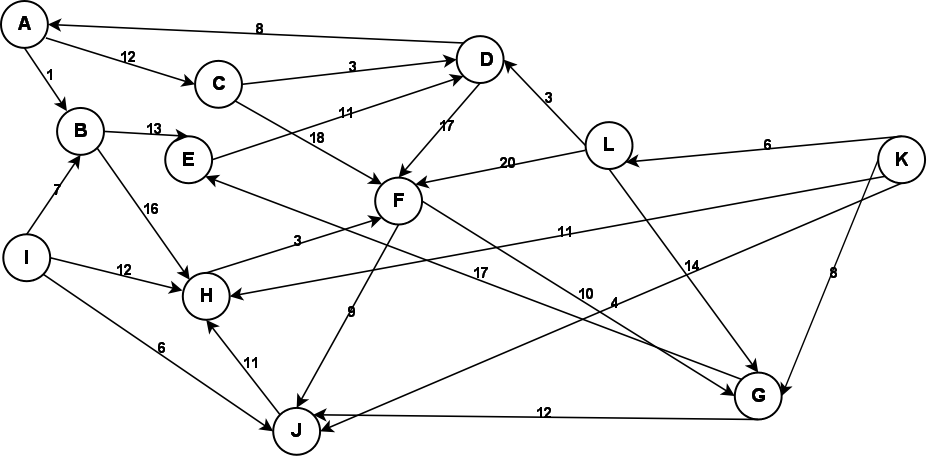
Kruskal’s algorithm is better for sparse graphs. Kruskals are better than Prims because they use simpler datat structures like arrays which take significantly less time than complex data structures in prims. For instance, in a sparse graph, it is assumed that m= O(n), kruskal runs at O(nlog(n)), which is a desired time complexity.

**5) Greedy Algorithms**

1. Define “Greedy Algorithm”
2. Give an example of a greedy algorithm with explanation of its greediness and performance.
3. Can greedy algorithms produce “optimal” solutions? Short explanation.

Greedy Algorithm is a mathematical method for finding the most simple and easy to apply solution for a complex problem. These algorithms concentrate on the solution of a smaller instance of the bigger problem. The function recursively by constructing a set of objects from the smallest basic parts. It is an optimal short term solution to a problem. An example of a greedy algorithm is prim’s algorithm. This finds the minimum spanning tree in a weighted undirected graph. The greediness is shown by how the algorithm creates the spanning tree one sub tree at a time breaking down the problem as a greedy algorithm should do. It is high performance and time complexity is at its best when it comes to dense trees. Greedy algorithms can produce optimal solutions to an extent, but this is only for the short term. Greedy algorithms are not the best solutions when it comes to the long term because their primary concern themselves with a small portion of the overall problem.

**6) Graph Traversals**



Given the above graph, provide the output of a breadth first and a depth first search. Make choices based on smallest edge weight, then alphabetical to break ties. Start at node A for both.

**Depth First:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | E | D | F | J | H | G | C |  |  |  |

**Breadth First:**

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | C | H | E | D | F | J | G |  |  |  |

**7) Graph Storage / Manipulation**

Given that a weighted directed graph is represented as an adjacency matrix called *adjM*, write a method that reverses all the edges of the graph. That is, for every edge ( A , B ) in the original graph, there will be an edge  ( B , A ) in the reversed graph with the same weight. Your function should be called  *reverse* .

**8) Graph Traversal**

Write a method that returns whether a graph is a tree. Your method takes a graph *G=(V,E)* as the input and outputs a boolean value. Your method should be called *isTree*().

bool Graph::isTree(int V, int E)

{

bool \*visited = new bool[E];

for (int i = 0; i < E; i++)

visited[i] = false;

if (visited[v] = true)

return false;

for (int u = 0; u < V; u++)

if (!visited[u])

return false;

return true;

}

**9) Huffman Coding**

**(A)** What is an optimal Huffman code for the following set of frequencies, based on the first 8 Fibonacci numbers:

a:1 b:1 c:2 d:3 e:5 f:8 g:13 h:21.

Show your answer as a tree. *Note:* assume that the ordering on the nodes is first by the frequency, and then by the alphabetic order of the node label, so that ab:2 precedes c:2; the node labels are alphabetized too, so that we have a node ab:2 but not ba:2.

**(B)** Use the code from part (a) to decode the string 11111111111001111101. (As a check: the result should be the name of something that is often yellow.)

**10) Bellman Ford (Optional)**

Using the graph from question 3, show a Bellman Ford solution.