**Test 3**

Name: Brice Allard

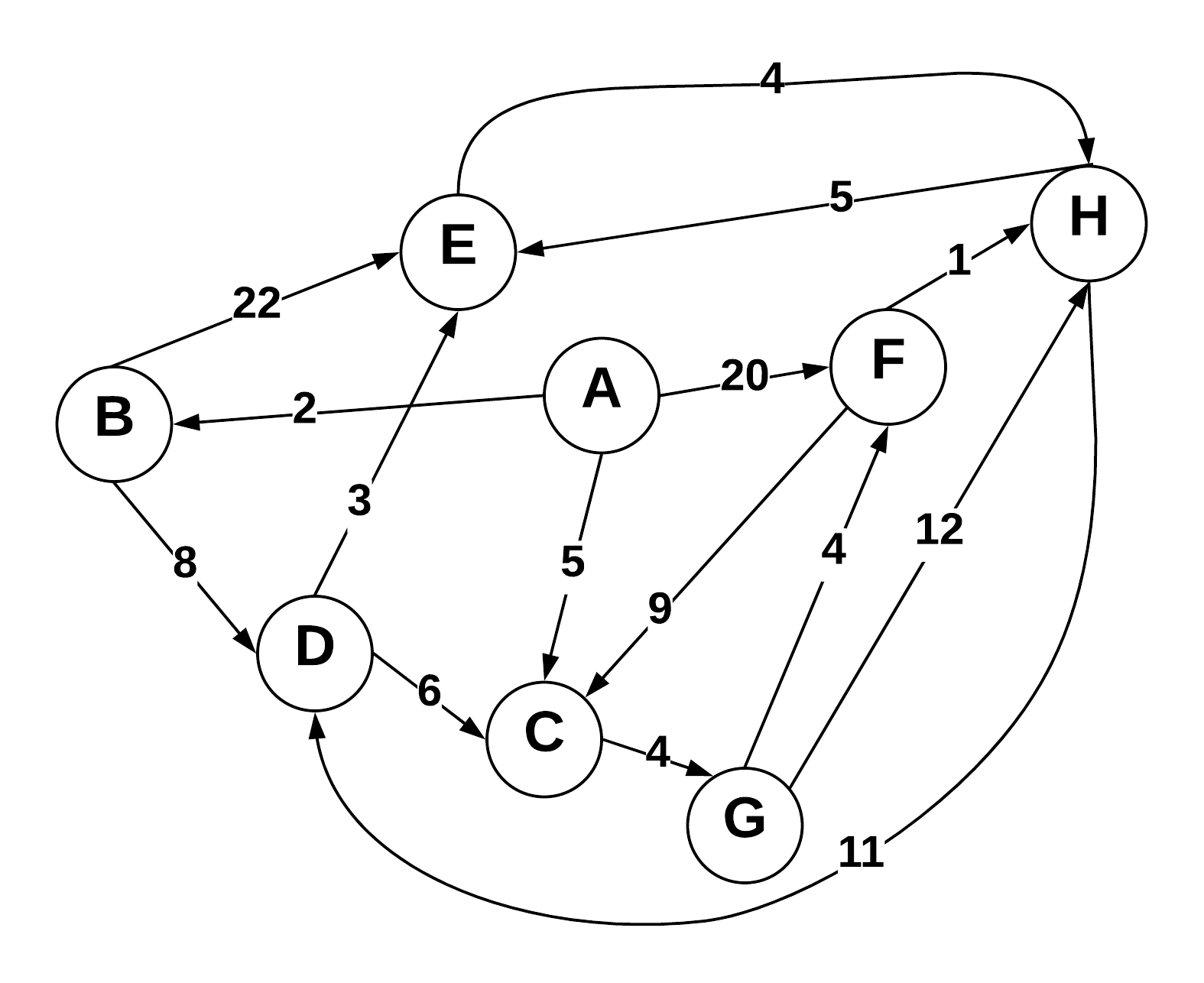
* Everything you turn in must be digitally created.
* No handwriting (except for signature below).
* You must work alone.
* Sharing of answers will result in a 0 on the exam, and possible F in the course.
* Send me your digitally created exam by Friday, May 4th by Midnight on a private slack message.
* 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**



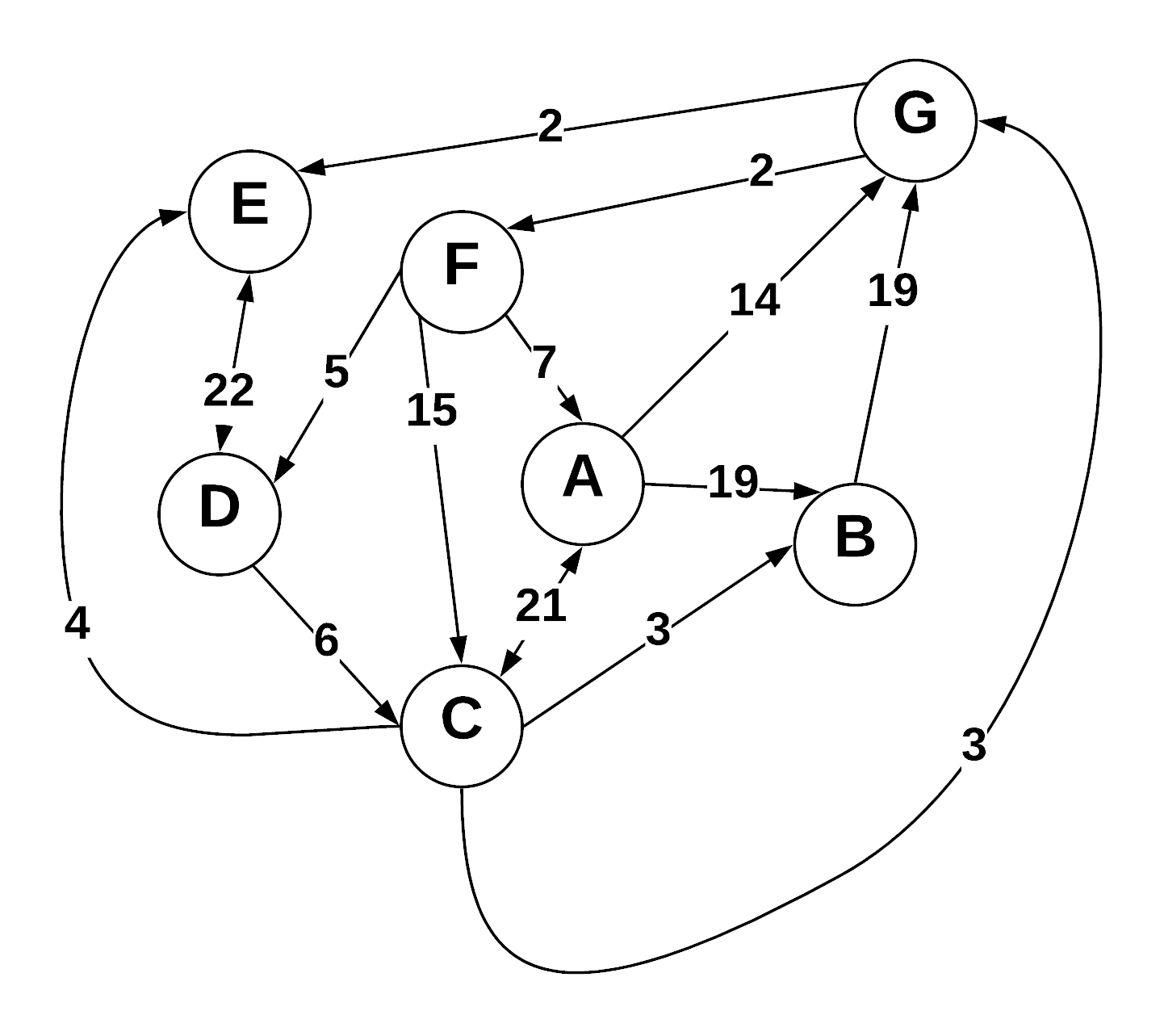
Use Dijkstra’s algorithm to compute the shortest paths from vertex A to every other vertex. Show your work in the space provided below. As the algorithm proceeds, cross out old values and write in new ones, from left to right in each cell. If during your algorithm two unvisited vertices have the same 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 | C | G | D | E | F | H |  |

|  |  |  |  |
| --- | --- | --- | --- |
| Vertex | Known | Cost | Previous |
| A | T | 0 |  |
| B | T | 2 | A |
| C | T | 5 | A |
| D | T | 10 | A B |
| E | T | ~~29~~ 13 | A B |
| F | T | ~~20~~ 13 | A C G |
| G | T | 9 | A C |
| H | T | ~~21 17~~ 14 | A F |

**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.

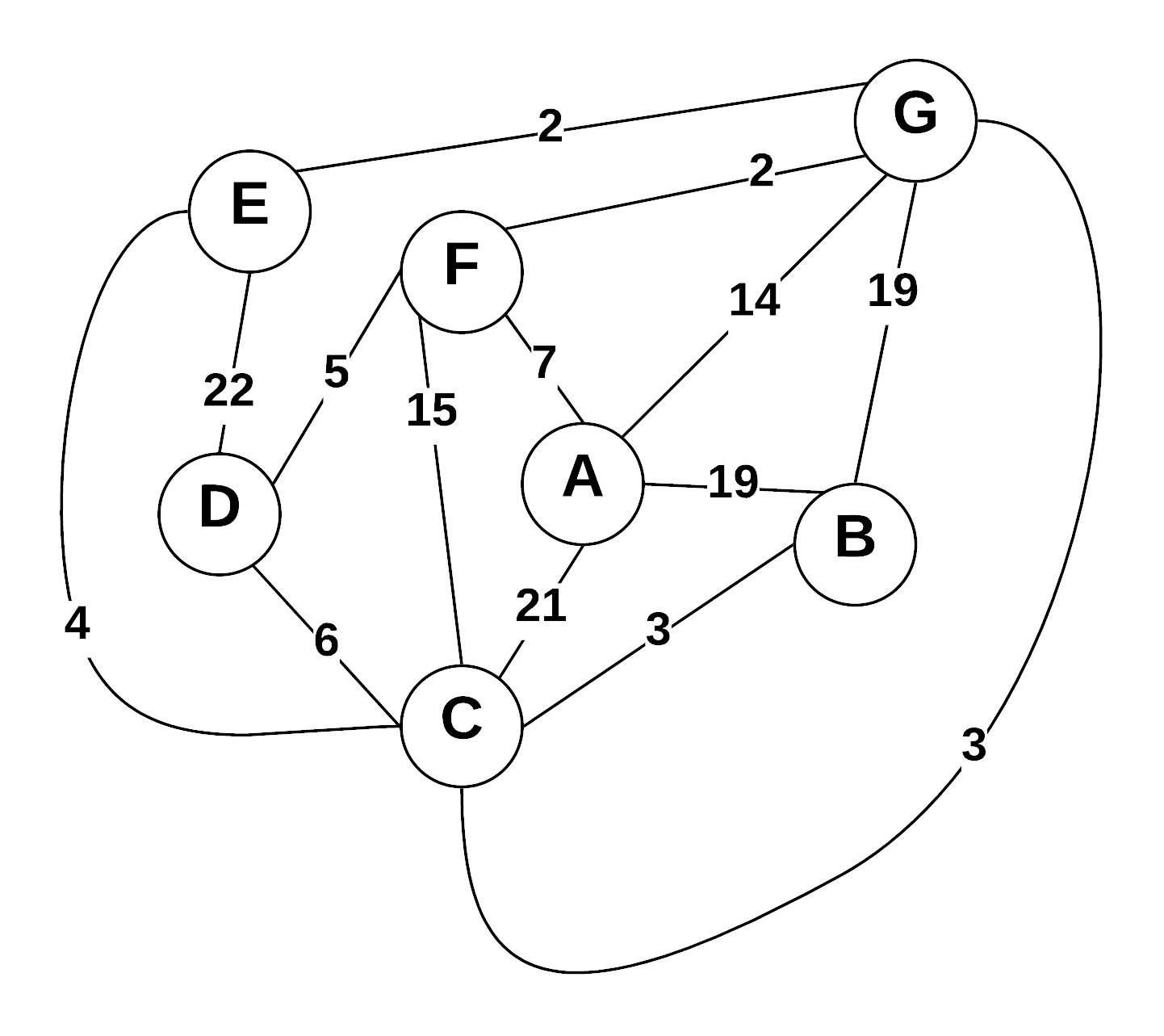
Vertices in Order of Discovery:

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

* 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 | A B C D E F | (E G) (F G) |
| G E | A B C D F | (D E) (F G) |
| G E F | A B C D | (D E) (A F) (C F) (D F) |
| G E F D | A B C | (C D) (A F) (C F) |
| G E F D C | A B | (A C) (B C) (A F) |
| G E F D C B | A | (A C) (A F) |
| G E F D C B A |  |  |
|  |  |  |

**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 |
| (F G) | 2 | 2 | (A) (B) (C) (D) (E) (F G) |
| (E G) | 2 | 4 | (A) (B) (C) (D) (E F G) |
| (C G) | 3 | 7 | (A) (B) (D) (C E F G) |
| (B C) | 3 | 10 | (A) (D) (B C E F G) |
| (C E) | 4 | X – Cycle |  |
| (D F) | 5 | 15 | (A) (B C D E F G) |
| (C D) | 6 | X – Cycle |  |
| (A F) | 7 | 22 | (A B C D E F G) |

**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?

**It has to do with the algorithm and complexity for both. Given edges E and vertices V, Kruskal’s runs in O(E log V) time while Prim’s is O(E + V log V) while using a heap (preferably a Fibonacci) or in O(V2) with an adjacency matrix. Therefore, Prim’s is better for graphs with more edges although the graph must be connected while Kruskel’s will work on a disconnected graph as well.**

**Prim’s is typically used in conjunction with a heap (preferably Fibonacci) and Kruskel’s uses a simpler data set such as an array or vector.**

**5) Greedy Algorithms**

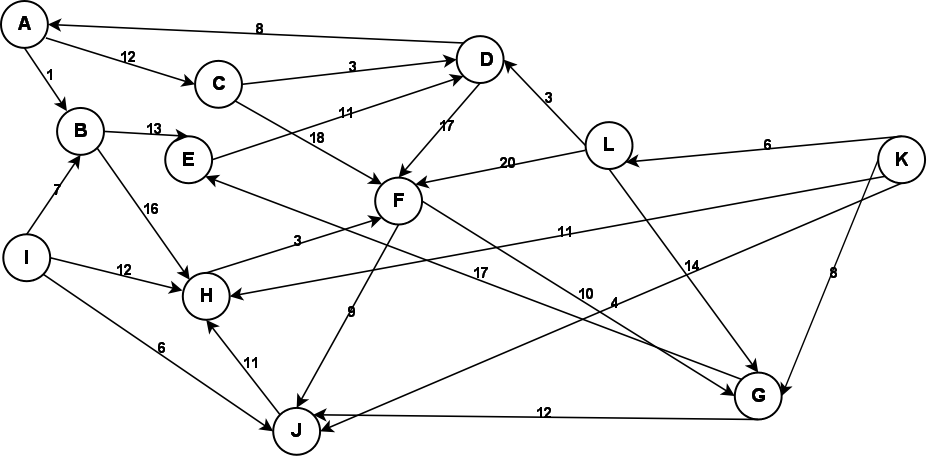
1. Define “Greedy Algorithm”
2. Give an example of a greedy algorithm with explanation of its greediness and performance.
3. Can greedy algorithms produce “optimal” solutions? Short explanation.

**A) Greedy Algorithm is an algorithm that makes decisions based on what is best at the current step of a finite solution without adherence to the general outcome of the entire process.**

**B) Prim’s is one example of a greedy algorithm. For example, the traveling salesman problem where we look at individual “stages” of the problem to find the nearest unvisited city to our current location. While finding the closest location may result in a locally optimal decision, this may not result in the finding the global optimal decision for the entire process, making this algorithm greedy, in terms of it only thinks for it’s current stage. In this example the process of finding locations would be in O(n2) because you look at all the local solutions, pick one, then repeat the process for the next location.**

**C) They can produce optimal solutions however are not always guaranteed to do so since the combination of locally optimal choices do not always result in finding the global optimum.**

**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 | E | H | 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* .

int\*\* reverse(int\*\* adjM) {

int\*\* transposed;

for (int u = 0; u < N; u++) {

for (int v = 0; v < M; v++) {

transposed[u][v] = adjM[v][u];

}

}

return transposed;

}

**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 isTree(Graph &G, int v, vector<bool> &found, int parent) {

found[v] = true;

for (int i = 0; i <G.adjM[v].size()) {

if (!found[i]) {

if (!isTree(G, i, found, v))

return false;

else if (i != parent)

return false;

}

}

return true;

}

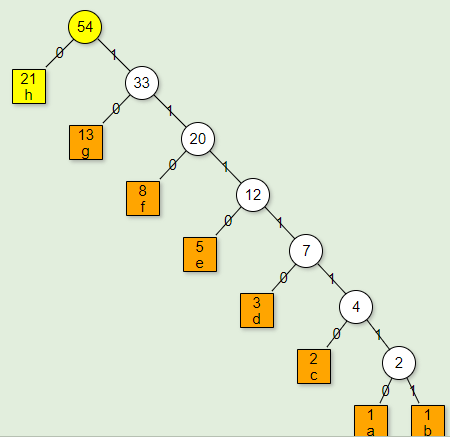
**9) Huffman Coding**

**(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.

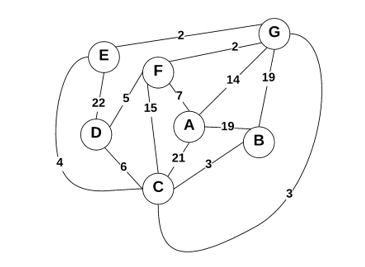
**(B)** Use the code from part (a) to decode the string 11111111111001111101. (As a check: the result should be the name of something that is often yellow.)



B) C A B

**10) Bellman Ford (Optional)**

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



Vertex Order of Discovery

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