Homework 5: Graphs, Minimum Spanning Trees, and Dijkstra Shortest-Path

1. (4 points) A graph is Hamiltonian if there is a cycle in the graph visiting each vertex exactly once. Give an example of an Eulerian graph that is not Hamiltonian, and a Hamiltonian graph that is not Eulerian.

Answer: A Hamiltonian graph that is not Eulerian is K2, the complete graph on n vertices, for any n ≥ 2, since each vertex has odd degree. An Eulerian graph that is not Hamiltonian is, for
example two simple cycles C[n], C[m] sharing a single vertex.

2. (2 points) For an unweighted graph, DFS traversal of the graph produces:
a. minimum spanning tree
b. longest spanning tree
c. all pair shortest path tree
*d. both a and c

Explanation: Minimum spanning tree is actually all pair shortest path. It visits all the nodes once.

3.(4 points) We want to assign classrooms for each class. There are n classes that meet in a day. For each class, we know the start and end-time of the class. No two classes whose meeting times overlap can be assigned the same classroom. Further assume that each classroom is large enough, so that any class can be assigned any classroom. Suppose we want to fewest number of classrooms required to accommodate all classes. How can we model this problem as a graph problem? Give an example of a graph and how you'll use a graph algorithm to solve this problem.

Explanation: Model the classes as vertices of a graph, and an edge between two vertices if the corresponding classes overlap. Let the colors correspond to the classrooms. Then, the fewest number of colors required to color the vertices so that no adjacent vertices share the same color is the fewest number of classrooms required so that no two overlapping classes have to share the same classroom.

5. (4 points) Write pseudo-code to detect a cycle in a graph.

Answers:

```
def isCyclicUtil(self, v, visited, recStack):

        # Mark current node as visited and
        # adds to recursion stack
        visited[v] = True
        recStack[v] = True

        # Recur for all neighbours
        # if any neighbour is visited and in
        # recStack then graph is cyclic
        for neighbour in self.graph[v]:
            if visited[neighbour] == False:
                if self.isCyclicUtil(neighbour, visited,
recStack) == True:
                    return True
            elif recStack[neighbour] == True:
                return True

        # The node needs to be popped from
        # recursion stack before function ends
        recStack[v] = False
        return False

    # Returns true if graph is cyclic else false
    def isCyclic(self):
        visited = [False] * self.V
        recStack = [False] * self.V
        for node in range(self.V):
            if visited[node] == False:
                if self.isCyclicUtil(node,visited,recStack) ==
True:
                    return True
        return False
```

6. Why can't Prim's or Kruskal's algorithms be used on a directed graph?

Prim's and Kruskal's algorithm output a minimum spanning tree for connected and "undirected" graph. If it is not connected, we can tweak them to output minimum spanning forests.

In Prim's algorithm, we divide the graph in two sets of vertices. One set of the explored vertices which have already formed MST (Set1) and another set of unexplored vertices which will eventually join the first set to complete "spanning"(Set2). At each instant, we select a minimum weighted edge in the cut joining the two disjoint sets. If there is no directed edge from explored nodes of MST to remaining unexplored, the algorithm gets stuck even though there are edges from unexplored nodes to explored nodes in MST.

In Kruskal's algorithm, the idea is to sort the edges in ascending order by their weight and pick them up in order and include them in MST explored nodes/edges if they do not already form a cycle with explored nodes. This is done by Union-Find DS. But detection of cycle for directed graphs fails with this method. For ex: Graph containing edges [1->2] [2->3] [1->3] will be reported to contain a cycle with the Union-Find method.

So Prim's fails because it assumes, every node is reachable from every node which though valid for undirected graphs may not be true for digraphs. Kruskal fails because of failure to detect cycles and sometimes it is essential to add edges making cycles to satisfy "minimum" weighted property of MST.

Also, in case of digraphs, MST doesn't make complete sense. Its equivalent for digraphs is "minimum spanning arborescence" which will produce a tree where every vertex can be reached from a single vertex.

Answer from:
https://stackoverflow.com/questions/22649416/why-cant-prims-or-kruskals-algorithms-be-used-on-a-directed-graph/35685918#35685918

7. (2 points) If my code represents a graph by {a: (b, c, d), b: (a), c: (a, d), d: (a, c)} then I am using a _____ to represent it?

*a. adjacency list
b. adjacency matrix
c. incidence matrix
d. adjacency incident

8. (2 points) Consider the following adjacency matrix:

	a	b	c	d	e	f	g
a	0	0	1	0	1	0	1
b	0	1	0	0	0	1	0
c	1	0	0	0	0	1	1
d	0	0	0	0	1	0	0
e	1	0	0	1	0	0	0
f	0	1	1	0	0	0	1
g	1	0	1	0	0	1	0

In the graph so described, there is a loop at vertex ___?
a. a
*b. b
c. d
d. f

9. (2 points) Consider the graph with following adjacency matrix:

	a	b	c	d
a	0	1	1	0
b	1	0	0	1
c	1	0	0	1
d	0	1	1	0

Which of the following is an Eulerian circuit through that graph?
a. a-c, c-b, b-d, d-a
b. a-d, d-b, b-c, c-a
*c. a-b, b-d, d-c, c-a
d. a-b, b-c, c-d, d-a

10. (2 points) Consider the graph described by the adjacency list {a: (b, c), b: (a, e), c: (b), d: (f, g) f: (d), g: (d)}. What are the connected components of that graph?

a. a-b-c-d and e-f-g
*b. a-b-c-e and d-f-g
c. a-b-c-g and d-f-g
d. the whole graph is connected

11. (2 points) Consider the graph described by the adjacency list {a: (b, c, d), b: (a, c), c: (a, b, d), d: (a, c)}. Does it contain an Eulerian path?
*a. Yes
b. No
c. Not enough information

12. (2 points) Consider the following matrix for a weighted graph; in what order will Kruskal's add edges to the MST?

	a	b	c	d	e	f	g	h
a	0	8	0	0	0	0	0	0
b	8	0	4	2	0	0	0	0
c	0	4	0	11	0	3	0	0
d	0	2	11	0	5	6	10	0
e	0	0	0	5	0	0	0	0
f	0	0	3	6	0	0	0	7
g	0	0	0	10	0	0	0	9
h	0	0	0	0	0	7	9	0

a. b-d, b-c, d-e, c-f, f-h, a-b, g-h
b. c-f, b-d, b-c, d-e, f-h, a-b, g-h
*c. b-d, c-f, b-c, d-e, f-h, a-b, g-h
d. b-d, c-f, b-c, d-e, g-h, f-h, a-b

13. (2 points) Consider the following matrix for a weighted graph; in what order will Prim's add edges to the MST, if it starts at vertex a?

	a	b	c	d	e	f	g	h
a	0	8	0	0	0	0	0	0
b	8	0	4	2	0	0	0	0
c	0	4	0	11	0	3	0	0
d	0	2	11	0	5	6	10	0

e	0	0	0	5	0	0	0	0
f	0	0	3	6	0	0	0	7
g	0	0	0	10	0	0	0	9
h	0	0	0	0	0	7	9	0

a. a-b, c-f, b-c, d-e, f-h, b-d, g-h

*b. a-b, b-d, b-c, c-f, d-e, f-h, h-g

c. a-b, b-c, c-f, d-e, b-d, f-h, h-g

d. a-b, c-f, b-d, b-c, d-e, f-h, h-g

14. (2 points) Consider the graph described by the adjacency list {a: (b, c, d), b: (a, c), c: (a, b, d), d: (a, c)}. Does it contain an Eulerian circuit?

a. Yes

*b. No

c. Not enough information

15. (2 points) Consider the following matrix for a weighted graph; at what point will Dijkstra's shortest-path algorithm, looking for the shortest path from a to h, overwrite the previously recorded shortest path to an intermediate node?

	a	b	c	d	e	f	g	h
a	0	8	0	0	0	0	0	0
b	8	0	4	2	0	0	0	0
c	0	4	0	11	0	3	0	0
d	0	2	11	0	5	6	10	0
e	0	0	0	5	0	0	0	0
f	0	0	3	6	0	0	0	7
g	0	0	0	10	0	0	0	9
h	0	0	0	0	0	7	9	0

a. when it finds a new path to b

b. when it finds a new path to e

c. when it finds a new path to d

*d. when it finds a new path to f