

## **CptS 223 Homework #4 - Graphs**

Please complete the homework problems on the following page using a separate piece of paper. Note that this is an individual assignment and all work must be your own. Be sure to show your work when appropriate.

**NAME: BOXIANG LIN**

1. [13] Define these terms as they relate to graph and graph algorithms:  
Use mathematical terms where appropriate.

Graph: Graph, an ADT, is use to express the relation of one and another objects, in which objects are called the vertices and relation are expressed often in a line to arrow which called the edges. So, graph consist of vertices and edges.

Vertices: vertice in graph is an existence, an object or a node.

Edge: edge is the relation indicator between paired vertices(objects), in the graph, the shape of edges are often drawn as line and arrow.

Undirected Graph: A graph with bidirectional edges, where the shape of edges are often drawn as a line or bidirectional arrow.

Directed Graph: A graph with directional edges, where the edges pointing to one direction, shape of the edges is an single direction arrow.

Path: A series of edges needed from one starting vertex to another destinating vertex called path.

Loop: An edge that connecting a vertex to itself without visiting another vertex.

Cycle: A vertex has a cycle is said if there exist a path that start at a vertex and is able to end at the vertex itself, so loops are cycles but cycles not necessary loops.

Acyclic A graph with no cycle.

Connected: undirected graph is weak connected if there is a path from every(any) vertex to every other vertex, so no vertices are unreachable. While the directed graph is weak connected if there is no disconnection between pair of vertices, and strong connected if every vertex is reachable.

Sparse: if numbers of edges much less than numbers vertices  $|E| \ll |V|$ .

Weight: Graph is weighted if edges carry a "cost"(numerical value) to travel them.

2. [4] Under what circumstances would we want to use an adjacency matrix instead of an adjacency list to store our graph?

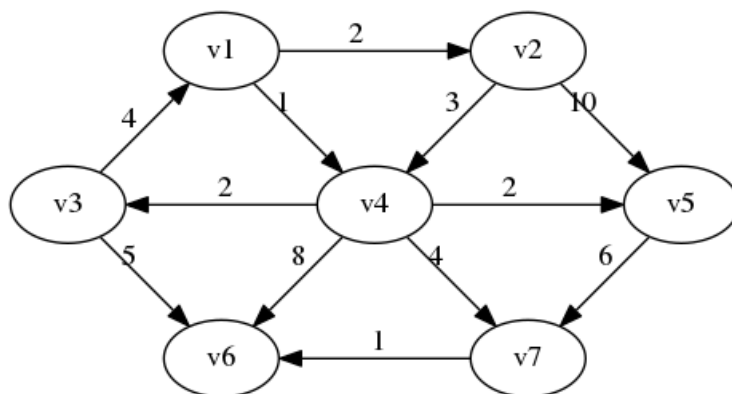
If  $|E| \approx V^2$ , then use adjacency matrix, because all matrix  $a[i,j]$  being filled with edge's weight. So, the space is fully use, and we can iterate over all edges with general runtime  $O(V^2)$  since not many unused  $a[i,j]$  need to be visited, and also adjacency matrix provides fast look ups.

If  $|E| \ll V$ , then use adjacency list. Since we only have little edges, we don't want to waste too much space, and we can iterate over all edges with runtime  $O(E)$  instead of  $O(V^2)$ , and adjacency list provide a not bad look ups in little edges case.

3. [6] Name three problems or situations where a graph would be a good data structure to use:

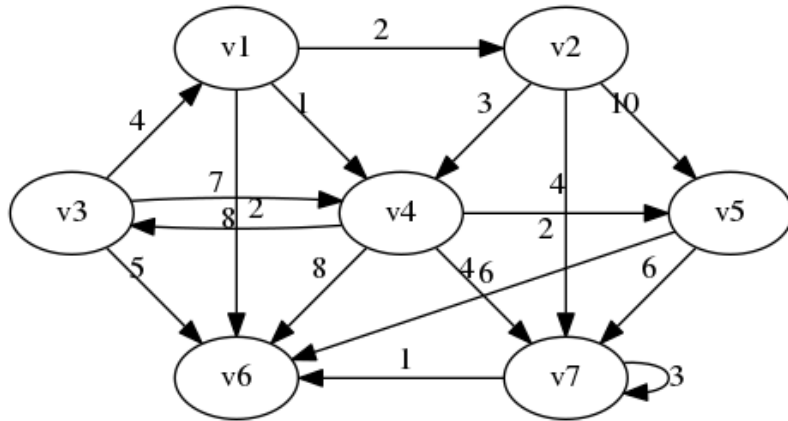
- Computational Biology, (studying the path of molecular activity, the interaction between proteins, etc.)
- Computer Network Optimization (studying the path of network node access, etc).
- Social Network Analysis (graph computing analysis)

4. [4] What kind of graph is this?



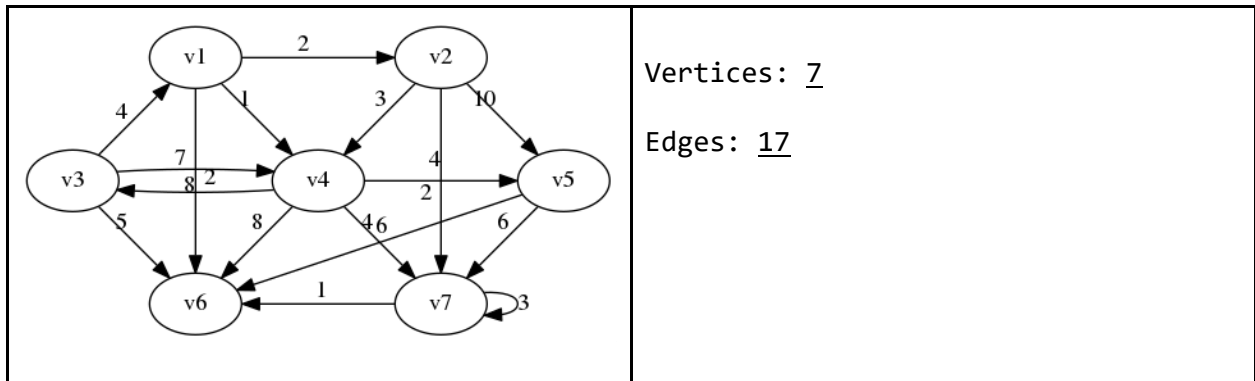
This is a directed cyclic graph (weak connected).

5. [4] Identify the loop in this graph:

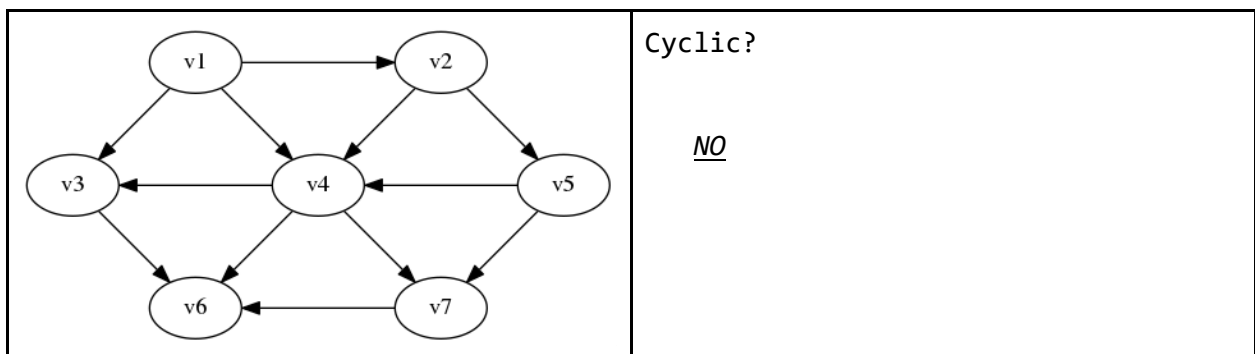


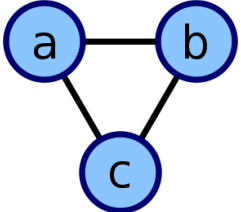
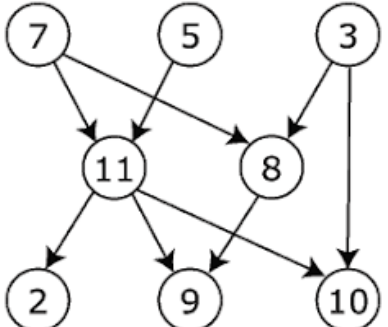
V7 -> V7, Edge weight:3

6. [4] How many vertices and edges are in this graph:

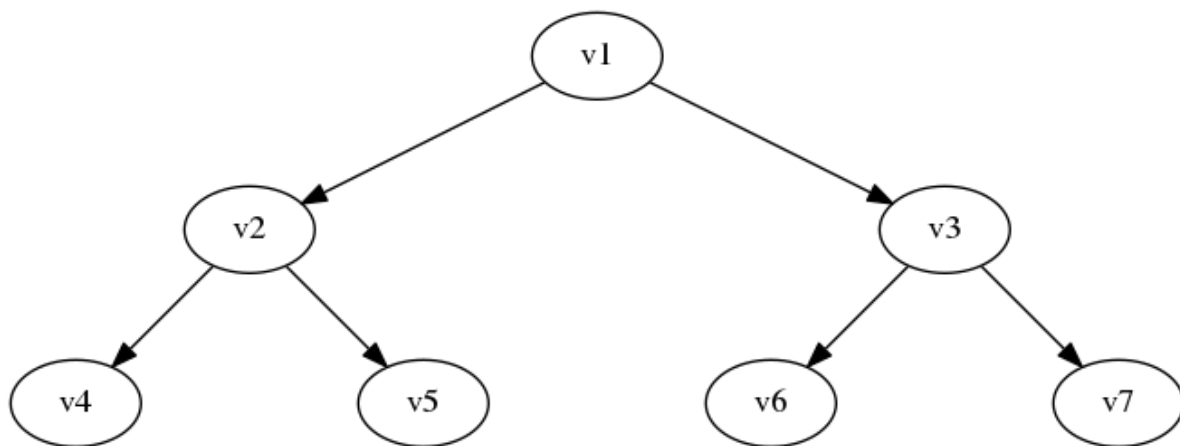


7. [6] Are these cyclic or acyclic graphs?



	Cyclic?  <u>Yes</u>
	Cyclic?  <u>No</u>

8. [5] A tree is a particular kind of graph. What kind of graph is that?



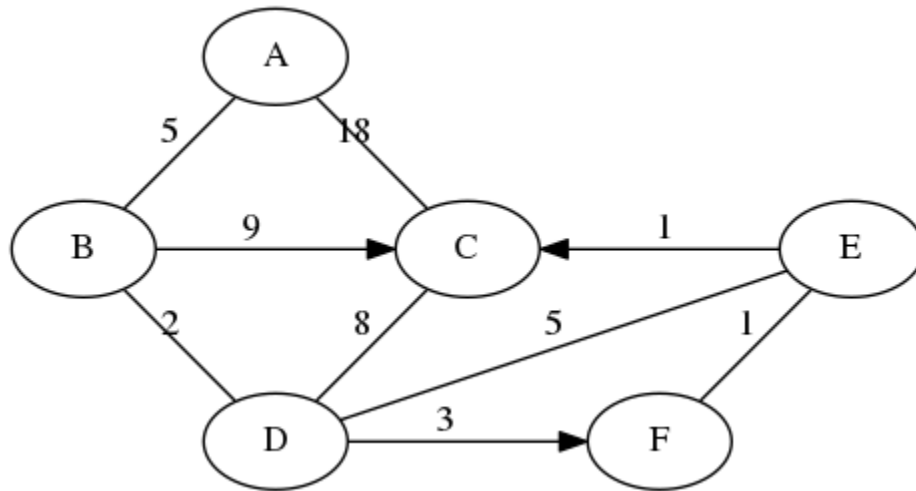
This is directed acyclic graph (weak connected)

9. [4] What is the difference between a breadth-first search and a depth first search?

- BFS start at a vertex and visiting its adjacency vertices level by level, can be implemented by queue.
- DFS start at a vertex and visiting one of its adjacency vertex all the way down then backtracking up and visiting the other possible previous adjacency vertex that has not been visited, can be implemented by stack.

**10. [10] Dijkstra's Algorithm.** Use Dijkstra's Algorithm to determine the shortest path starting at A. Note that edges without heads are bi-directional. To save time, you do not have to add items to the "priority queue" column after it has been discovered (listed in the "distance" column). Use the table below to show your work.

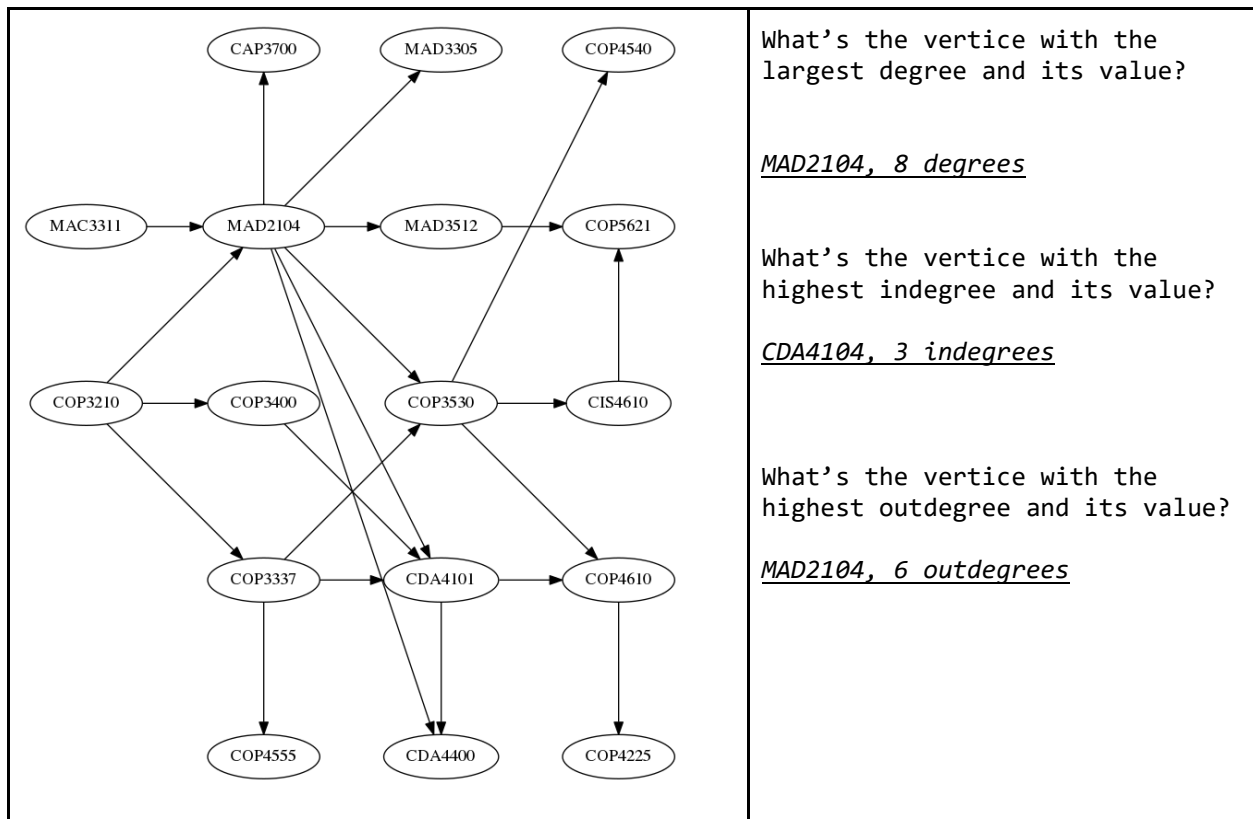
What's the shortest route (by weight) from A to C? A->B->D->F->E->C Weight: 12



Node: Distance	Priority Queue
	A:0
A:0	B:5, C:18
B:5	D:7, C:14, C:18
D:7	F:10, E:12, C:14, C:15, C:18
F:10	E:11, E:12, C:14, C:15, C:18
E:11	C:12, E:12, C:14, C:15, C:18
C:12	E:12, C:14, C:15, C:18
	C:14, C:15, C:18 poll(E:14) out from priority queue, and not do anything since E:11 already in path.
	C:15, C:18 --- poll(C:14) out from priority queue, and not do anything since C:12 already in path.

	C:18 poll(C:15) out from priority queue, and not do anything since C:12 already in path.
	Poll(C:18) and not do anything to the path. NOW Priority Queue Empty, completed!

11. [10] **Topo sort.** Show the final output of running Topo Sort on this graph:



Topo sort output:

MAC3311, COP3210,

MAD2104, COP3400, COP3337,

CAP3700, MAD3305, MAD3512, COP4555, COP3530, CDA4101, CDA4400

COP4540, CIS4610, COP4610,

COP5621, COP4225