Hw4 Yufi Zu

(a) True.

Bellman-Ford Alg. is for finding shortest path, but use Bellman-Ford to find longest path need to change some initial values.

(b) True

Each iteration updates the distances along the paths. If there is a negative weight cycle, you can keep decreasing the dis without limit as you traverse the cycle repeatedly.

(c) True, Floyd's algorithm for finding all-pairs shortest paths. Each iteration of the major outer loop., the algorithm explores the possibility of improving the shortest paths by considering additional intermediate nodes. After the completion of the algorithm, the 'D' array contains the final shortest distances between all pairs

(d) False, the correct form should be $d(j) = MIN[d(j), d(i) + w(i,j)]$

(e) True, in this special case, each arc has the same weight. Even though, Dijkstra's and BFS have different objectives and operating for finding shortest path, whatever Dijkstra's account the edges and aims min cost, the answer should be the same with BFS, Because all weights are the same. For BFS, it will not account the weight, and explore all nodes at current depth before moving on to the nodes at the next depth. BFS is good for solving unweighted or same weight graph

(f). True This is the key for Dijkstra's algorithm. As the algorithm progresses, it selects the vertex with the smallest temporary distance, finalizes its distance. and updates the temporary distances of its neighbors

(f) False, when the E is at most V^2, so the total time is $O(N^2 \log N)$

(h) True, the graph is undirected. and every pair of nodes i, j with a unique simple path between them. Connected. undirected and Acylic is tree.
If there are some multiple simple paths between some pairs of nodes, there was a cycle.

(i) True. It states that in an MST with distinct edge weights, each node is connected to the tree by its minimum weight edge

(j) True, they are all examples of greedy algorithm. The greedy choice made at each step. The algorithm makes the locally optimal choice at each step with the hope to lead a globally optimal solution.

2.

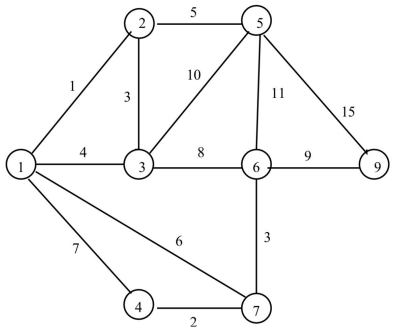
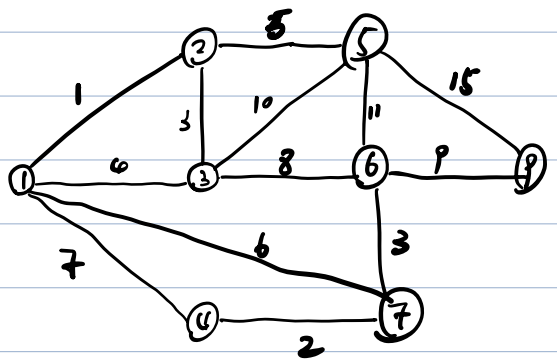


Figure 1:

2. (15 pts) Consider the undirected weighted graph in Figure 1. Show the distance estimates computed by each step in the outer loop of the Bellman-Ford algorithm for finding a shortest path tree in the graph, starting from node 1 to all other nodes. **Note the other loop in the Bellman-Ford algorithm adds an arc so you should give a table recording each new distance $d(j)$ and new predecessor $p(j)$ for each node.**

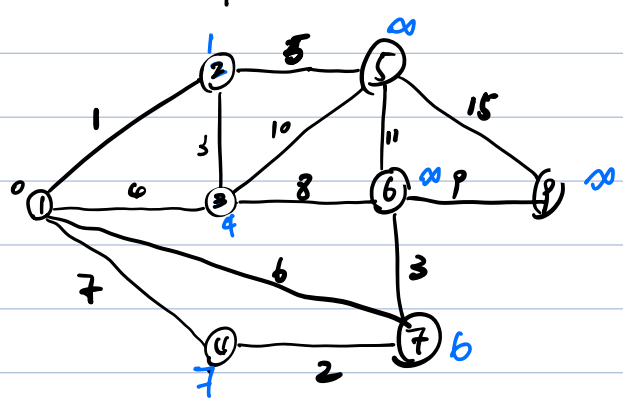


Initial : $d(1)=0$, $d(2)=int$, $d(3)=int$, $d(4)=int$, $d(5)=int$, $d(6)=int$, $d(7)=int$, $d(9)=int$.

$p(1)=-1$, $p(2)=-1$, $p(3)=-1$, $p(4)=-1$, $p(5)=-1$, $p(6)=-1$, $p(7)=-1$, $p(9)=-1$

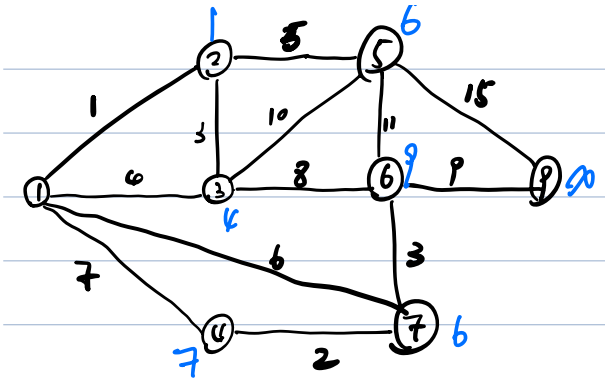
1st round: $d(1)=0$, $d(2)=1$, $d(3)=4$, $d(4)=7$, $d(5)=int$, $d(6)=int$, $d(7)=6$, $d(9)=int$

$p(1)=-1$, $p(2)=1$, $p(3)=1$, $p(4)=1$, $p(5)=-1$, $p(6)=-1$, $p(7)=1$, $p(9)=-1$



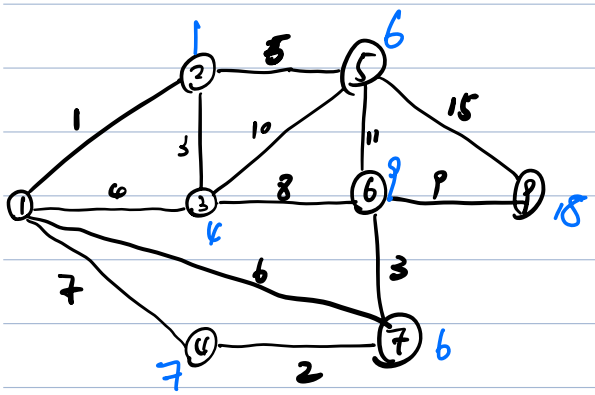
2nd Rounds: $d(1)=0$, $d(2)=1$, $d(3)=4$, $d(4)=7$, $d(5)=6$, $d(6)=9$, $d(7)=6$, $d(9)=\infty$

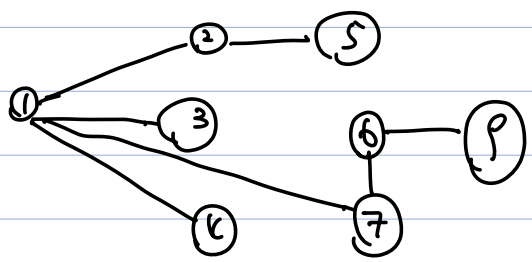
$p(1)=-1$, $p(2)=1$, $p(3)=1$, $p(4)=1$, $p(5)=2$, $p(6)=7$, $p(7)=1$, $p(9)=-1$



3rd Round: $d(1)=0$, $d(2)=1$, $d(3)=4$, $d(4)=7$, $d(5)=6$, $d(6)=9$, $d(7)=6$, $d(8)=18$

$p(1)=-1$, $p(2)=1$, $p(3)=1$, $p(4)=1$, $p(5)=2$, $p(6)=7$, $p(7)=1$, $p(8)=6$





3.

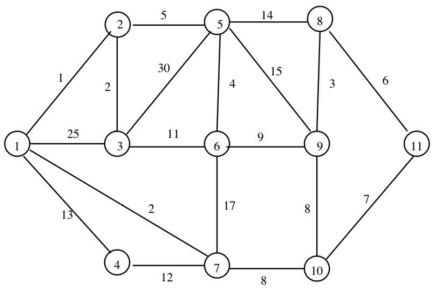
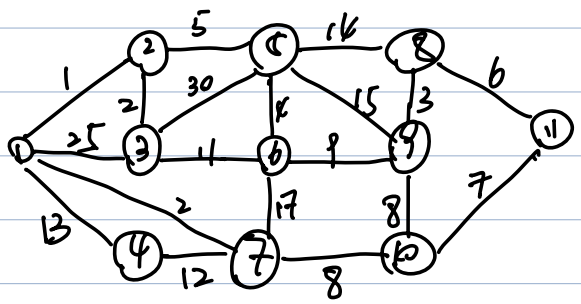
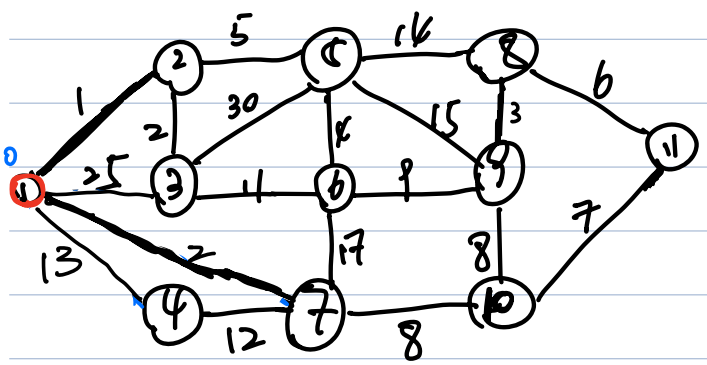


Figure 2:

3. (15 pts) Consider the weighted, undirected graph in Figure 2. Assume that the arcs can be traveled in both directions. Illustrate the steps of Dijkstra's algorithm for finding a shortest paths from node 1 to all others.



arcs can travel ~~both~~ Directions

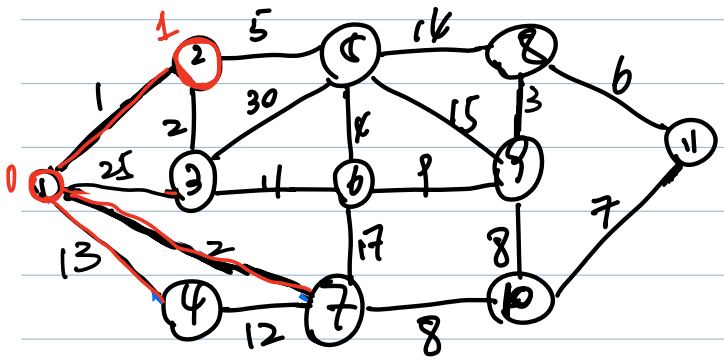


Unvisited nodes:

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 }

Distance:

1: 0	5: ∞	9: ∞
2: ∞	6: ∞	10: ∞
3: ∞	7: ∞	11: ∞
4: ∞	8: ∞	

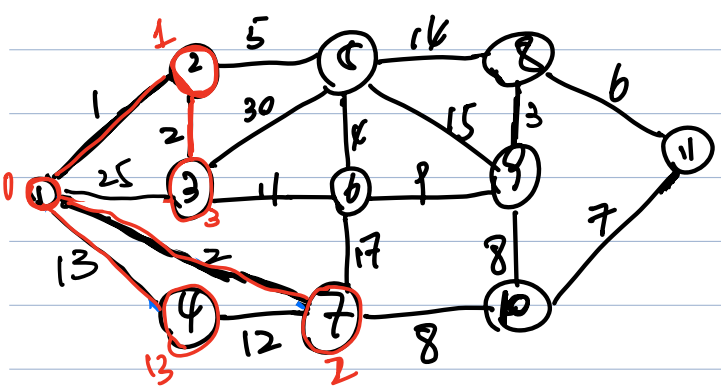


Unvisited nodes:

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 }

Distance:

1: 0	5: ∞	9: ∞
2: 1	6: ∞	10: ∞
3: ∞	7: ∞	11: ∞
4: ∞	8: ∞	

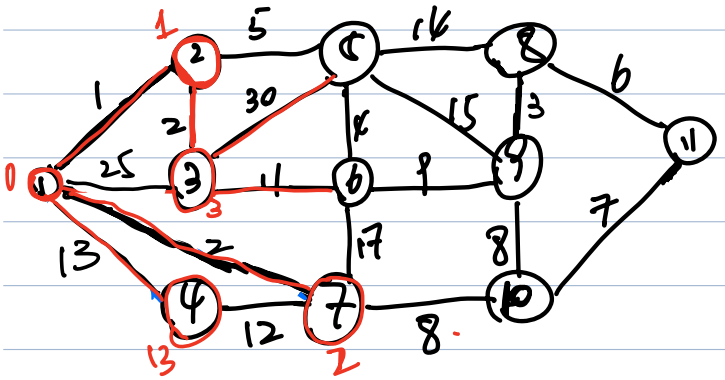


Unvisited nodes:

{ 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11 }

Distance:

1: 0	5: ∞	9: ∞
2: 1	6: ∞	10: ∞



3: 3 7: 2 11: ∞
4: 13 8: ∞

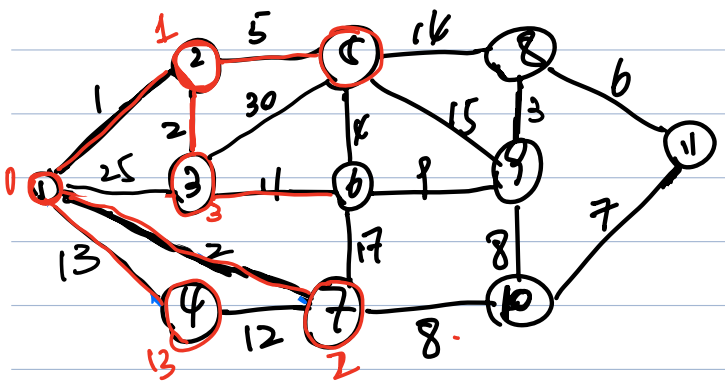
Unvisited nodes:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Distance:
1: 0 5: ∞ 9: ∞
2: 1 6: ∞ 10: ∞

3: 3 7: 2 11: ∞
4: 13 8: ∞

From node 1 → Node 3 to Node 5, cost 1+2+30 = 33.

From Node 1 to Node 2 to Node 5, cost 1+5=6, so this is a closest path



Unvisited nodes:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Distance:
1: 0 5: 6 9: ∞
2: 1 6: ∞ 10: ∞

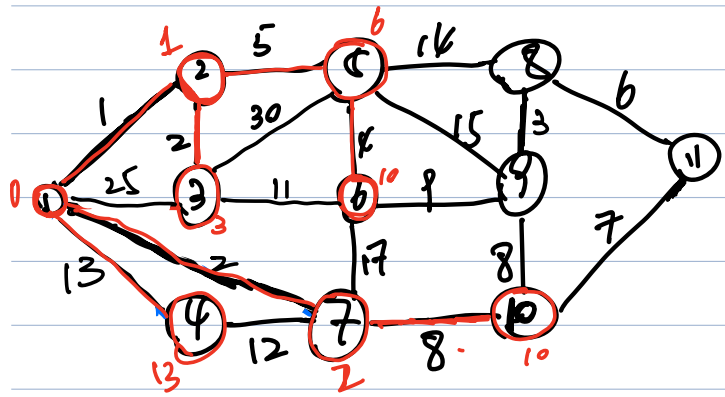
3: 3 7: 2 11: ∞
4: 13 8: ∞

From Node 1 → Node 2 → Node 5 → Node 6

1 + 5 + 4 = 10, the closest

From Node 1 → Node 2 → Node 3 → Node 6, From node 1 → node 7 → node 6 = 2+17=19

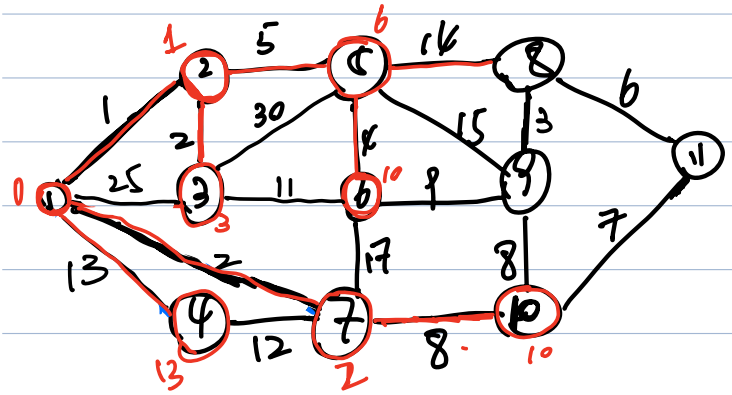
1+2+11 = 14



Unvisited nodes:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Distance:
1: 0 5: 6 9: ∞
2: 1 6: 10 10: ∞

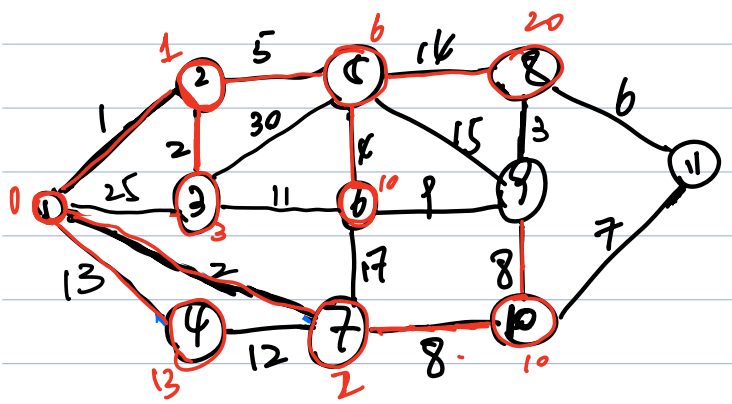
3: 3 7: 2 11: ∞
4: 13 8: ∞



Unvisited nodes:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Distance:

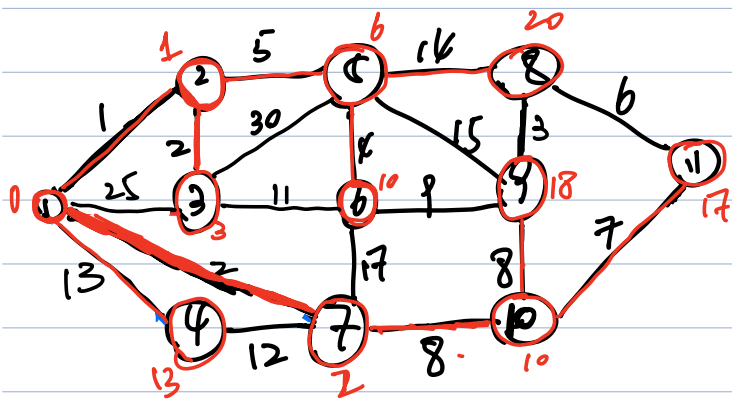
1: 0	5: 6	9: ∞
2: 1	6: 10	10: 10
3: 3	7: 2	11: ∞
4: 13	8: ∞	



Unvisited nodes:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Distance:

1: 0	5: 6	9: ∞
2: 1	6: 10	10: 10
3: 3	7: 2	11: ∞
4: 13	8: 20	



Unvisited nodes:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11}

Distance:

1: 0	5: 6	9: 18
2: 1	6: 10	10: 10
3: 3	7: 2	11: 17
4: 13	8: 20	

4.

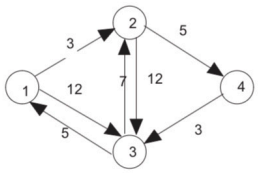
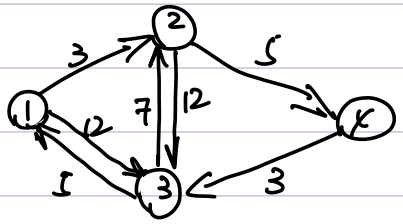


Figure 3:

4. (15 pts) Consider the graph in Figure 3. This is a directed graph. Use the Floyd-Warshall algorithm to find the shortest distance for all pairs of nodes. Show your work as a sequence of 4 by 4 tables.



i \ j	1	2	3	4
1	0	3	12	∞
2	∞	0	12	5
3	5	7	0	∞
4	∞	∞	3	0

Treat node 1 as intermediate node, calculate the Distance[i][j] for every [i,j] node pair the formula.

Dis[i][j] = min(Dis[i][j], Distance[i][1] + Distance[1][j])

i \ j	1	2	3	K
1	0	3	12	∞
2	∞	0	12	5
3	5	7	0	∞
4	∞	∞	3	0

Then Treat 2 as intermediate node

Dis[i][j] = min(Dis[i][j], Dis[i][2] + Dis[2][j])

i \ j	1	2	3	K
1	0	3	12	8
2	∞	0	12	5
3	5	7	0	12
4	∞	∞	3	0

Treat node 3 as intermediate node

i \ j	1	2	3	4
1	0	3	12	8
2	12	0	12	5
3	5	7	0	12
4	8	10	3	0

$Dis[i][j] = \min(Dis[i][j], Dis[i][3] + Dis[3][j])$

Treat node 4 as intermediate node

i \ j	1	2	3	4
1	0	3	11	8
2	5	0	8	5
3	5	7	0	12
4	8	10	3	0

$Dis[i][j] = \min(Dis[i][j], Dis[i][4] + Dis[4][j])$

final Answer

i \ j	1	2	3	4
1	0	3	11	8
2	5	0	8	5
3	5	7	0	12
4	8	10	3	0

5.

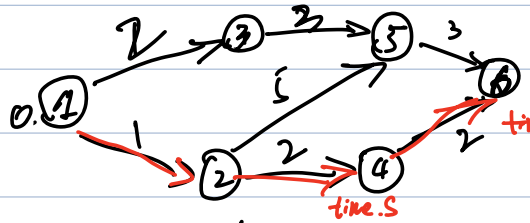
5. (15 pts)In city streets, the length of an arc often depends on the time of day. Suppose you have a directed graph of streets connecting nodes that represent intermediate destinations, and you are given the travel time on the arc as a function of the time at which you start to travel that arc. Thus, for arc e, you are given $d_e(t)$, the time it takes to travel arc e if you start at time t. These travel times must satisfy an interesting ordering property: You can't arrive earlier if you started later. That is, if $s < t$, then $s + d_e(s) <= t + d_e(t)$. Suppose that you start at time 0 at the origin node 1. Describe an algorithm for computing the minimum time path to all nodes when travel times on arcs are time dependent and satisfy the ordering property.

By given a directed weighted graph, there is no negative weights.

So the algorithm for computing the min-time path to all nodes is Dijkstra's Algorithm.

By given if $s<t$, then $s + de(s) <= t+de(t)$

for example,



Start from Node1 with 0, choose adjacent nodes with min distance, since $S<t$, for instance $S=3$, $t=5$, $s+ d_s(e) \leq t+d_t(e)$, and when $t=5$, the node will reaches the destination node 6., because choose min distance to reach next node can achieve the min time. The time s, only reaches the node before destination node.

. So during the implementation, build a minHeap to keep track of the minimum time to reach each node. If arrival time is less than the current min time for the destination node, update the min time. And add the destination node to the Heap.

6. (20 pts) This is a forward star representation for a directed graph with $|V| = 11$ vertices and $|E| = 16$ edges.

```
                0  1  2  3  4  5  6  7  8  9  10  11  12
Vertex Number:  1  2  3  4  5  6  7  8  9  10 11  12
Array First:   { 1, 3, 4, 5, 7, 8, 12, 12, 14, 14, 15, 17 }

                0  1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16
Edge Number:    1  2  3  4  5  6  7  8  9  10 11  12  13  14  15  16  17
Array Edge:    { 2, 6, 6, 7, 3, 7, 8, 5, 8, 9, 10, 9, 11, 9, 9, 10, -1 }
```

(a) Draw the graph on the template in Fig. 6. (HINT: You may want to do part b first.)

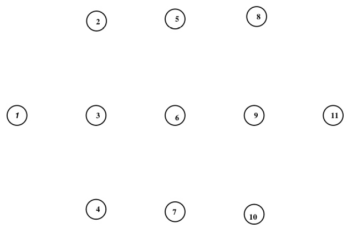


Figure 4:

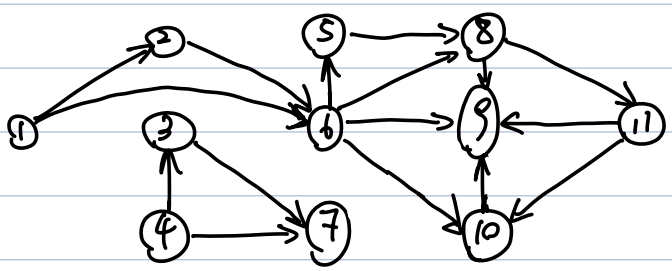
(b) Represent this graph as an adjacency list.

(c) Is this graph a DAG?

Array	first[i]
i=1	1
i=2	3
i=3	4
i=4	5
5	7
6	8
7	12
8	12
9	14
10	14
11	15
12	17

Edge[j]	
2	j=1
6	j=2
6	j=3
7	j=4
3	j=5
7	6
8	7
5	8
8	9
9	10
10	11
9	12
11	13
9	14
9	15
10	16
-1	17

(a)



(b) Vertex

1 → 2 → 6

2 → 6

3 → 7

4 → 3 → 7

5 → 8

6 → 5 → 8 → 9 → 10

7 →

8 → 9 → 11

9 →

10 → 9

11 → 9 → 10

(c) The above graph is DAG, there is no cycles
DAG means no cycles, and directed.