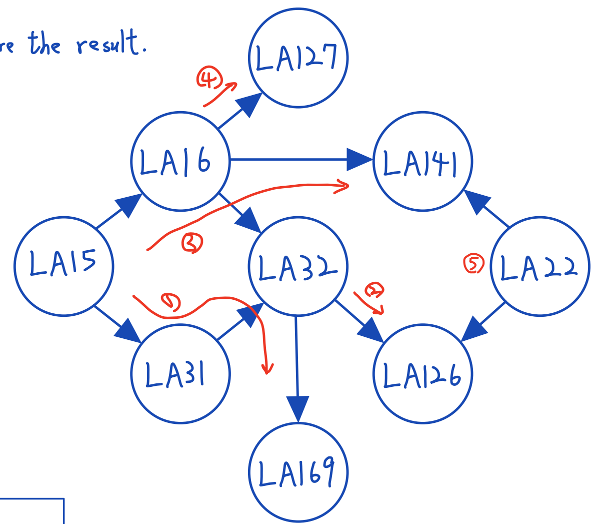
Module 9 Problem Set Hsuan-You(Shaun) Lin

1. My Answer:
O(E) is not necessarily O(V) because E = O(V^2), and when the graph complete, we will get the maximum edge number. Thus, we could get that O(log(E)) = O(log(V^2)) = O(2*log(V)) = O(log(V)).

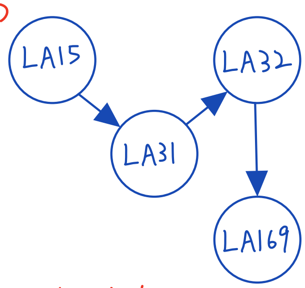
2. My Answer: LA22->LA15->LA16->LA127->LA141->LA31->LA32->LA126->LA169
My steps:

We could use topological sort + DFS for this question.
I traverse from left to right, bottom to top, and I use the stack in DFS to store the result.

The total time complexity of topological sort is O(M+N), where M is edges' number in the graph, N is the number of nodes, it will runs a simple for-loop to place all the nodes in the result array by checking the in-degrees to be zero which take O(N).



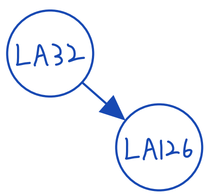
①



appended last one to stack

0	
1	
2	
3	
4	
5	
6	
7	
8	LA169

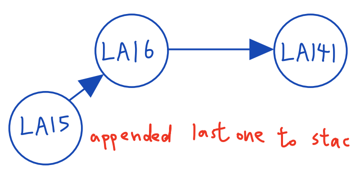
②



appended last one to stack, and backtrack to LA15, appended each course during backtrack.

0	
1	
2	
3	
4	
5	LA31
6	LA32
7	LA126
8	LA169

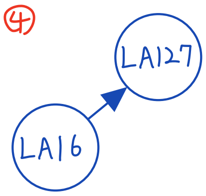
③



appended last one to stack

0	
1	
2	
3	
4	LA141
5	LA31
6	LA32
7	LA126
8	LA169

④



appended last one to stack, and backtrack to LA15, appended each course during backtrack.

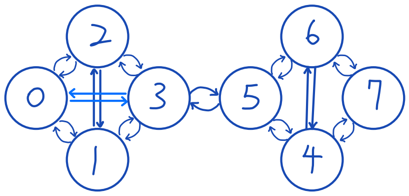
0	
1	LA15
2	LA16
3	LA127
4	LA141
5	LA31
6	LA32
7	LA126
8	LA169

⑤

appended last one to stack

0	LA22
1	LA15
2	LA16
3	LA127
4	LA141
5	LA31
6	LA32
7	LA126
8	LA169

3. My Answer: The DFS sequence is 0->1->2->3->5->4->6->7



← Visualized the adjacency list.

My Python code:

```python
# Hsuan-You Lin Module 9 Problem Set Question 4.
from collections import defaultdict as DefDict
from collections import deque

def BFS(adjacency_list, start_idx, results):
    queue = deque([start_idx])
    mark = set([start_idx])
    while queue:
        idx = queue.popleft()
        results.append(idx)
        for neighbor in adjacency_list[idx]:
            if neighbor not in mark:
                queue.append(neighbor)
                mark.add(neighbor)

if __name__ == "__main__" :
    adjacency_list = [[1, 2, 3],
                      [0, 2, 3],
                      [0, 1, 3],
                      [0, 1, 2, 5],
                      [5, 6, 7],
                      [3, 4, 6],
                      [4, 5, 7],
                      [4, 6]]
    results = []
    BFS(adjacency_list, 0, results)
    print(results)
```

```
Module9 — -bash —
(base) pisces:Module9 pisces$ python Q4.py
[0, 1, 2, 3, 5, 4, 6, 7]
(base) pisces:Module9 pisces$
```

4. My Answer: The BFS sequence is 0->1->2->3->5->4->6->7
My Python code:

```python
# Hsuan-You Lin Module 9 Problem Set Question 3.
from collections import defaultdict as DefDict
from collections import deque

def DFS(adjacency_list, visited, idx, results):
    if idx in visited:
        return
    visited.add(idx)
    results.append(idx)
    for neighbor in adjacency_list[idx]:
        DFS(adjacency_list, visited, neighbor, results)

if __name__ == "__main__" :
    adjacency_list = [[1, 2, 3],
                      [0, 2, 3],
                      [0, 1, 3],
                      [0, 1, 2, 5],
                      [5, 6, 7],
                      [3, 4, 6],
                      [4, 5, 7],
                      [4, 6]]
    visited = set()
    results = []
    DFS(adjacency_list, visited, 0, results)
    print(results)
```

```
Module9 — -bas
(base) pisces:Module9 pisces$ python Q3.py
[0, 1, 2, 3, 5, 4, 6, 7]
(base) pisces:Module9 pisces$
```

5. My Python Code:

All the elements are in a priority queue and are maintained with the min-heap property, and the new_element is pushed into the priority queue preserving the min-heap property.

Here I changed the priority of B to D.

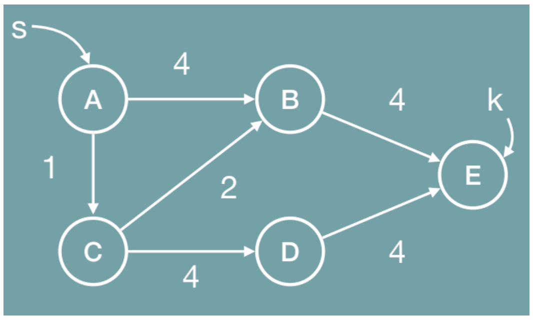
```python
# Hsuan-You Lin Module 9 Problem Set Question 5.
import time
import heapq as hq

def update_priority(arr, new_element):
    i, j = 0, 0
    hq.heapify(arr)
    print(arr, "\n")
    while len(arr) != 0:
        print("The ", arr[0][1], " with priority ",
              arr[0][0], " in progress", end="")

        for _ in range(0, 5):
            print(".", end="")
            time.sleep(0.5)
        hq.heappop(arr)

        if j < len(new_element):
            hq.heappush(arr, new_element[j])
            print("\n\nNew element uptate:", new_element[j])
            print()
            j = j+1

        print("\n New Queue:", arr)
        print("\n")

    print("\nUpdate Priority Queue completed.")

if __name__ == "__main__" :
    arr = [(2, 'A'), (5, 'B'), (1, 'D'),
           (4, 'E'), (3, 'C'), (6, 'F')]

    new_element = [(1, 'B')]
    update_priority(arr, new_element)
```

Module9 — -bash — 84×24

```
(base) pisces:Module9 pisces$ python Q5.py
[(1, 'D'), (3, 'C'), (2, 'A'), (4, 'E'), (5, 'B'), (6, 'F')]

The  D  with priority  1  in progress.....

New element uptate: (1, 'B')


 New Queue: [(1, 'B'), (3, 'C'), (2, 'A'), (4, 'E'), (5, 'B'), (6, 'F')]
```

6. My Python Code:

Here's the Dijkstra's Algorithm Python code which I modified from the notes, I also used the graph in the note as shown below. When a node gets extracted from the priority queue it means adding it to the set. And in the code "dist[u_id] + w_uv < dist[v_id]" can never be true, because dist[v_id] is the minimum distance for v once it has been added to set.




```python
def dijkstra(graph, start, end):
    # Assign to all nodes a distance value of 'inf', except for the start node which has a distance of zero, and initialize a predecessor dictionary.
    inf = sys.maxsize
    pred = {i:i for i in graph}
    dist = {i:inf for i in graph}
    dist[start] = 0
    # initialize a priority queue and insert the tuple
    PQ = []
    heapq.heappush(PQ, [dist[start], start])

    while(PQ):
        # find the next node with the smallest distance value.
        u = heapq.heappop(PQ)
        u_dist = u[0]
        u_id = u[1]
        if u_dist == dist[u_id]:
            for v in graph[u_id]:
                v_id = v[0]
                w_uv = v[1]
                if dist[u_id] +  w_uv < dist[v_id]:
                    dist[v_id] = dist[u_id] + w_uv
                    heapq.heappush(PQ, [dist[v_id], v_id])
                    pred[v_id] = u_id
    # reconstruct the shortest path with the help of the predecessor dictionary.
    else:
        st = []
        # follow the shortest path backwards from the target to the start
        node = end
        while(True):
            st.append(str(node))
            if(node == pred[node]):
                break
            node = pred[node]
        path = st[::-1]
        print("The distance from " + start + " to " + end + " is: " + str(dist[end]) + "\n")
        print("The shortest path is: " + " ".join(path))

if __name__ == "__main__":
    graph = {"A": [("B",4), ("C",1)],
             "B": [("E",4)],
             "C": [("B",2), ("D",4)],
             "D": [("E",4)],
             "E": []
            }
    start = "A"
    end = "E"
    dijkstra(graph, start, end)
```

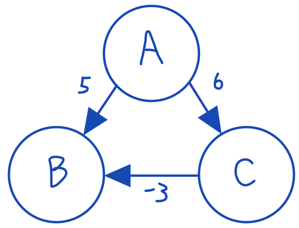
Module9 — -bas

```
(base) pisces:Module9 pisces$ python Q6.py
The distance from A to E is: 7

The shortest path is: A C B E
(base) pisces:Module9 pisces$
```

7. My Answer & reason:

When Dijkstra encounters negative edge weights, it won't be able to find the minimum distance, once a node is marked as visited it cannot be reconsidered even if there is another path with less cost or distance.

Here's an example for negative edge weights:

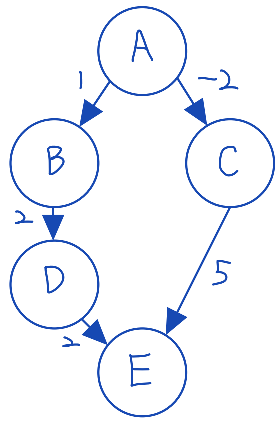


Consider node A as the source node, and we want to find the shortest distance from A to B, the shortest distance from A–>B = 5, but if we traveled as A–>C–>B the distance will be as 3 (A–>C = 6 and C–>B = -3 => 6 + (-3) = 3), but Dijkstra's Algorithm gives the incorrect answer as 5, which is not the shortest distance. If we want to solve this problem, we can use Bellman-Ford Algorithm to find the shortest distance in case of negative weights, as it stops the loop when it encounters a negative cycle.

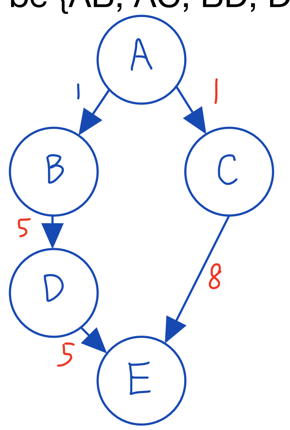
---

8. My Answer:

Here's an example that adding a positive offset to all negative edge weights.



The solution for above graph should be {AB, AC, BD, DE}, which cost of 0.



Above graph is after we adding a positive offset +3 to make all of the edge become positive. Since we add the positive offset to the graph, the solution will be {AB, AC, BD, CE}, which cost of 2, so this approach won't succeed in computing shortest paths.

9. My Answer & reason:

Universal sink is a vertex that has out degree zero, which corresponding row of that vertex in the adjacency matrix will be all zeros and the column of that vertex has all one's expected. The algorithm terminated when it find a row of all zeros, after we start to traverse the adjacency matrix, if we encounter a 0, so we increment j and next look at A[0][1]. Here we encounter a 1. So we have to increment i by 1. A[1][1] is 0, so we keep increasing j. Thus, we can find whether a universal sink exist or not from above step, so the overall time complexity is O(V).

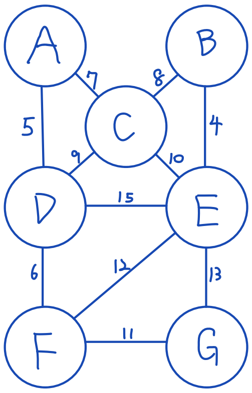
My Python Code:

```
# Hsuan-You Lin Module 9 Problem Set Question 9.
class Graph:
    def __init__(self, vertices):
        self.vertices = vertices
        self.adjacency_matrix = [[0 for i in range(vertices)]
                                    for j in range(vertices)]

    def insert(self, s, destination):
        self.adjacency_matrix[s - 1][destination - 1] = 1

    def issink(self, i):
        for j in range(self.vertices):
            if self.adjacency_matrix[i][j] == 1:
                return False
            if self.adjacency_matrix[j][i] == 0 and j != i:
                return False
        return True

    def eliminate(self):
        i, j = 0, 0
        while i < self.vertices and j < self.vertices:
            if self.adjacency_matrix[i][j] == 1:
                i += 1
            else:
                j += 1
        if i > self.vertices:
            return -1
        elif self.issink(i) is False:
            return -1
        else:
            return i

if __name__ == "__main__":

    number_of_vertices = 6
    number_of_edges = 5
    g = Graph(number_of_vertices)

    # input set 1
    g.insert(1, 6)
    g.insert(2, 6)
    g.insert(3, 6)
    g.insert(4, 6)
    g.insert(5, 6)
```

```
Module9 — -ba
(base) pisces:Module9 pisces$ python Q9.py
Sink found at vertex 6
(base) pisces:Module9 pisces$
```

10. My Steps:
The basic idea of Kruskal's Algorithm is to maintain a forest, in every iteration it will find the smallest edge, when all nodes visited the algorithm will stop, and the eventually there is only one tree.

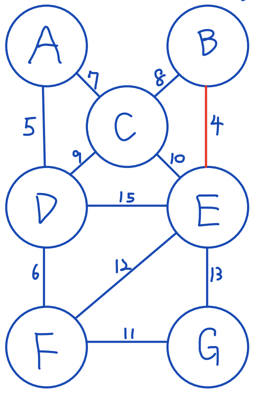
Build a queue of edges, and sort the elements in ascending order.



edge	weight
BE	4
AD	5
DF	6
AC	7
BC	8
CD	9
CE	10
GF	11
EF	12
GE	13
DE	15

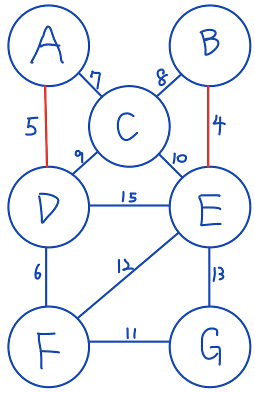
$\Rightarrow$

Perform dequeue, if the edge is not in the same tree, mark the edge as red.



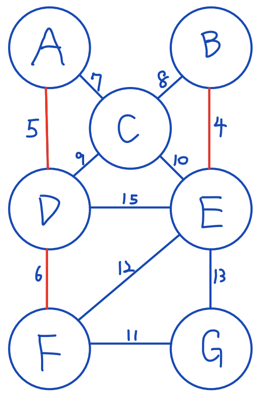
edge	weight
BE	4
AD	5
DF	6
AC	7
BC	8
CD	9
CE	10
GF	11
EF	12
GE	13
DE	15

$\Rightarrow$



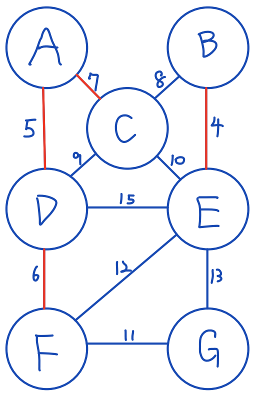
edge	weight
BE	4
AD	5
DF	6
AC	7
BC	8
CD	9
CE	10
GF	11
EF	12
GE	13
DE	15

$\Downarrow$



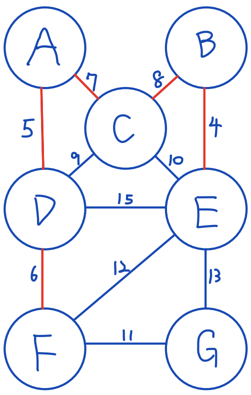
edge	weight
BE	4
AD	5
DF	6
AC	7
BC	8
CD	9
CE	10
GF	11
EF	12
GE	13
DE	15

$\Leftarrow$



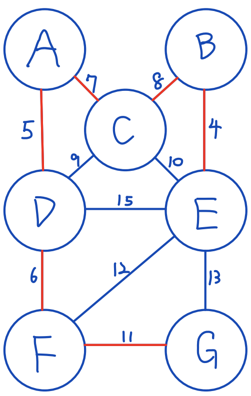
edge	weight
BE	4
AD	5
DF	6
AC	7
BC	8
CD	9
CE	10
GF	11
EF	12
GE	13
DE	15

$\Leftarrow$



edge	weight
BE	4
AD	5
DF	6
AC	7
BC	8
CD	9
CE	10
GF	11
EF	12
GE	13
DE	15

$\Downarrow$



edge	weight
BE	4
AD	5
DF	6
AC	7
BC	8
CD	9
CE	10
GF	11
EF	12
GE	13
DE	15

those edge are in the same tree, so don't mark as red.

The final minimum spanning tree