1.
Bob loves foreign languages and wants to plan his course schedule to take the following nine language courses: LA15, LA16, LA22, LA31, LA32, LA126, LA127, LA141, and LA169. The course prerequisites are:

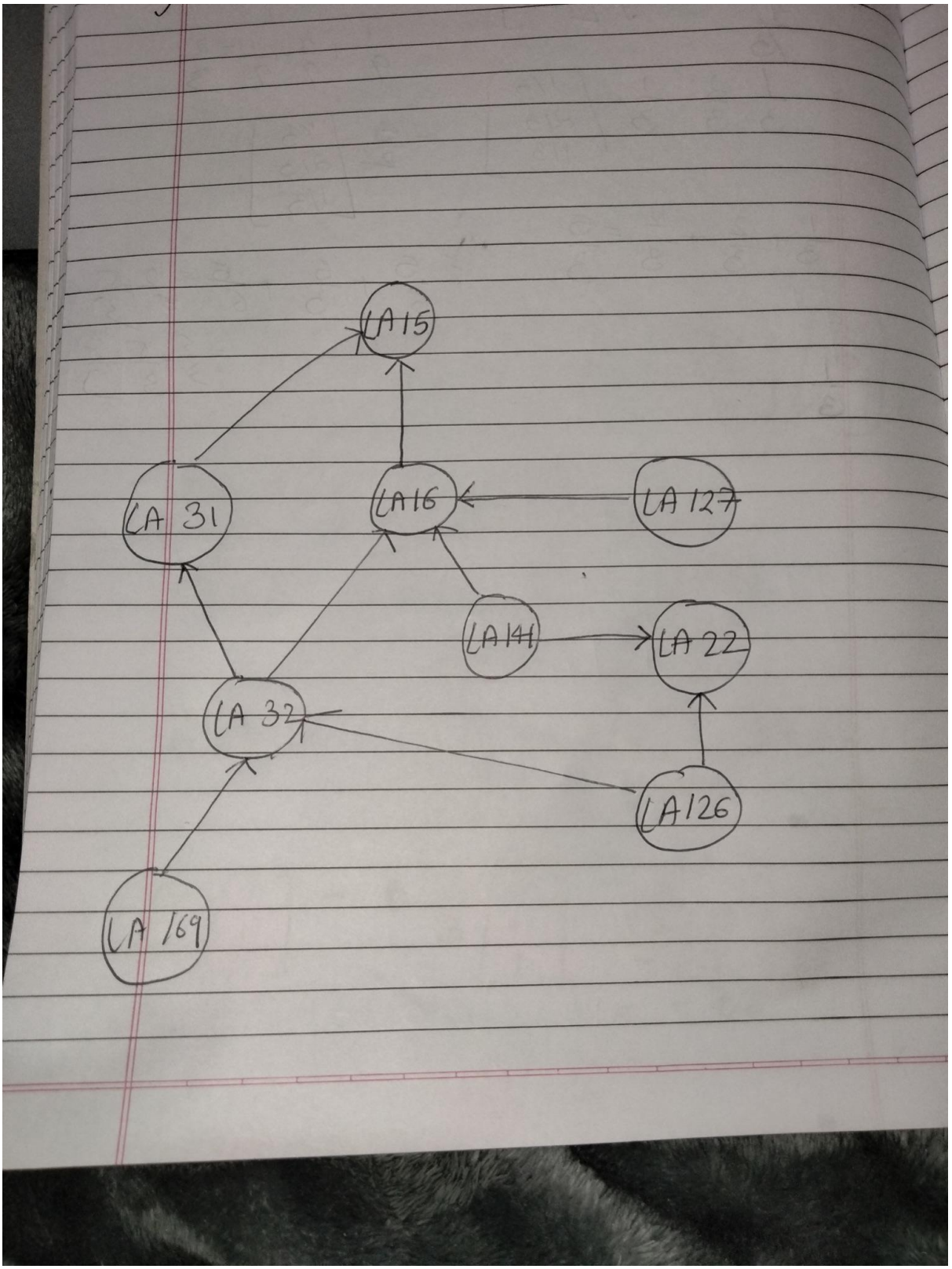
- LA15: (none)
- LA16: LA15
- LA22: (none)
- LA31: LA15
- LA32: LA16, LA31
- LA126: LA22, LA32
- LA127: LA16
- LA141: LA22, LA16
- LA169: LA32.

Find a sequence of courses that allows Bob to satisfy all the prerequisites.

Solution:

We can use the graph data structure to represent the subjects. Nodes will represent the subjects and the incoming edges to the node represent the prerequisites for the adjacent courses.

LA15, LA16, LA22, LA31, LA32, LA126, LA127, LA141, LA169


LA 15
LA 31
LA 16
LA 127
LA 141
LA 22
LA 32
LA 126
LA 169

2. Suppose G is a graph with n vertices and m edges. Describe a way to represent

G using O(n+m) space so as to support in O(log n)time an operation that can test, for any two vertices v and w, whether v and w are adjacent.

Solution:

We can design a graph using a balanced binary search tree data structure. We are using this data structure because the search and the insert operation takes O(log n). We will be using an array to represent the balanced binary search tree.

Size of the array will be equal to the total number of vertices of the graph. Each index of the array will store the adjacent vertices.

Because of this technique the space complexity of the above algorithm will become O(n + m)

In order to check if the vertices v and w are adjacent, we can use any traversal tree algorithm to check if the w is present in the vertex index v.

The time complexity of the searching algorithm will be logn.

3.
Tamarindo University and many other schools worldwide are doing a joint project on multimedia. A computer network is built to connect these schools using communication links that form a free tree. The schools decide to install a file server at one of the schools to share data among all the schools. Since the transmission time on a link is dominated by the link setup and synchronization, the cost of a data transfer is proportional to the number of links used. Hence, it is desirable to choose a "central" location for the file server. Given a free tree T and a node v of T, the *eccentricity* of v is the length of a longest path from v to any other node of T. A node of T with minimum eccentricity is called a *center* of
T.
a) Design an efficient algorithm that, given an n -node free tree T, computes a center of T.

(b)Is the center unique? If not, how many distinct centers can a free tree have?

Solution:

A free tree is a tree which does not have any root node. Since we are removing the nodes from the tree, the tree is acyclic. Also, it is mentioned that the cost of a data transfer is proportional to the number of links used, hence we have an equal edge weight.

We will be using a linked list data structure to represent the graphs.

Steps:

1. Run a while loop until the number of vertices are not equal to one or not.

2. Traverse through each of the vertices of the graphs. Check if the degree of the vertex is equal to one. If yes, remove the vertex from the graph and update its adjacency list. If not then go to other vertices.

Since we are iterating over all the vertices of the graph, the time complexity of the above algorithm is O(n) and the space complexity of the above algorithm is O(1) as we are not using any extra space.

b) The center is not unique. We can have at most two centers in the graph.

4.

There is an alternative way of implementing Dijkstra's algorithm that avoids use of the locator pattern but increases the space used for the priority queue, Q, from O(n) to O(m)for a weighted graph, G, with n vertices and m edges. The main idea of this approach is simply to insert a new key-value pair, D[v],v, each time the D[v]value for a vertex, v, changes, without ever removing the old key-value pair for v. This approach still works, even with multiple copies of each vertex being stored in Q, since the first copy of a vertex that is removed from Q is the copy with the smallest key. Describe the other changes that would be needed to the description of Dijsktra's algorithm for this approach to work. Also, what is the running time of Dijkstra's algorithm in this approach if we implement the priority queue, Q, with a heap?

Solution:
We will be using a priority queue implemented with a min heap property to solve the problem. We will be sorting based on the distance parameter. In this algorithm, we are not using set to check if the node is visited or not.

Since we are using priority queue, the node with the minimum distance will always be on top of the queue which will guarantee us the minimum distance from source to destination.

The time complexity of the above algorithm will be O(N +E)logn which is equivalent to O(Nlogn) and the space complexity is O(N)

Resource:

https://www.youtube.com/watch?v=jbhuqIASjoM

5.

Solution:

We will be using the Dijikstra’s algorithm to solve this problem.

```
function dijkstra(G, S)

    for each vertex V in G

        distance[V] <- infinite
```

```
        previous[V] <- NULL
        If V != S, add V to Priority Queue Q
    distance[S] <- 0


    while Q IS NOT EMPTY
        U <- Extract MIN from Q
        for each unvisited neighbour V of U
            tempDistance <- distance[U] + edge_weight(U, V)
            if tempDistance < distance[V]
                distance[V] <- tempDistance
                previous[V] <- U
    return distance[], previous[]
```

The Time complexity of the above algorithm will be O(NlogN) and the space complexity will be O(N).

6.

Solution:

The following traits may be used to transform this issue into a graph traversal issue:

On the graph, each flight represents an edge. The weight of the edge and the flying time are correlated.In the graph, every airport represents a node. Any note must have a discrepancy between the leaving time of the subsequent flight and the prior flight's arrival time that is more than c.The aforementioned condition will reduce the number of edges we must go over in order to locate the best solution.

This problem can be solved using depth first search and basically boils down to a graph traversal problem with optimization (DFT).The journey is complete when we have identified a set of routes that connect the source and destination airports and have reported the cost of each route. By keeping track of whether a node has previously been visited, cycles need to be found and removed.

The execution time will be O(n+m), similar to classical DFS, where n is the number of nodes and m is the number of flights. We must also keep an eye out for the quickest route.