CS 600 HW 6
SHUCHI PARAGBHAI MEHTA
CWID: 20009083

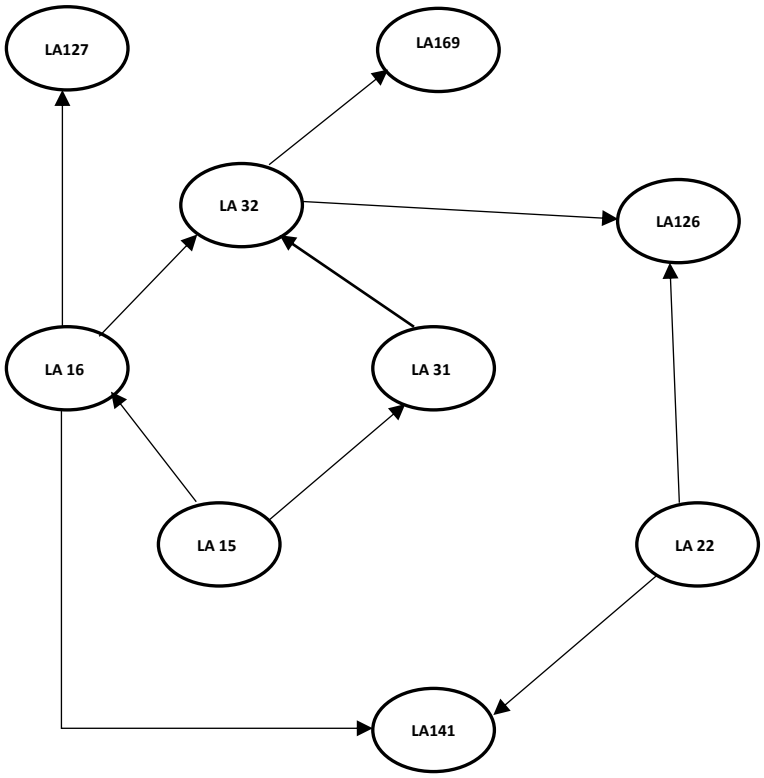
- 13.7.4

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.

By using directed graph, these sequences are possible that allows bob to satisfy all the prerequisites,



One possible sequence can be:
LA15, LA16, LA127, LA31, LA32, LA169, LA22, LA126, LA141

- 13.7.19

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.

Adjacency List is one way to represent G in $O(n + m)$ space. The degree of vertex v determines the overall time required to check all of its neighbors.

1. For each vertex *v* in adjacency list maintains a list which are reachable from *v*.
2. An adjacency list will have a maximum length of n lists for n vertices.
3. Sorting the list of neighbors that can be reached from vertex v.
4. Use binary search, which takes time proportional to the logarithm of the degree, to determine whether vertices v and w are adjacent.

The binary search will test whether any two vertices, v and w, are adjacent in $O(log\ n)$ time.

- 13.7.37

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?

A tree without root is called as free tree. This means that graph does not contain cycle.
Free trees are trees that have no roots. This indicates that the graph lacks a cycle.

As given, a free tree consists of node *v.* The longest route from v to any other node is eccentricity of v.

a) An Efficient algorithm which computes a center of T.

T does not include a cycle because it is a tree. Up until one or two nodes remain in the tree, which will be the heart of the tree T, we must remove all the leaf nodes (external nodes) from the tree.

1. Eliminating all a tree T's leaf nodes (external nodes), then renaming the new tree T.
2. Next, take out every leaf node from T` and give the remaining tree the name T``.
3. Continue the previous step until only one or two nodes remain.
4. If there is only one node left in the tree T^k, it will be in the center of the tree T, and its eccentricity will be k.
5. If not, the eccentricity of the center node for two nodes in tree T^k, will be $k + 1$.

Running time: The above method takes O(n) runtime. O(n) time will be required to traverse the tree, and the while loop only processes each node once.

b) No, the center cannot be unique.

- It is possible for a free tree to have 2 distinct centers.
- Let’s consider P as the longest path of tree T.
- The median of P is the path taken by the center of the tree T.
- The center of the tree will be one if P's length is odd. The center of the tree will be two if P's length is even.

- 14.7.11

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?

According to the above condition, alternative way to implement the Dijkstra algorithm is by keeping track of visited nodes in a set.

Three operations can be performed INSERT, EXTRACT-MIN, and DECREASE-KEY. Priority queue, however, does not support decrease-key when using a heap and priority queue to implement the Dijkstra algorithm.

To solve this problem, instead of updating the key, we can insert a copy of it. This enables the same vertex to exist in numerous instances.

Every vertex will only execute the INSERT operation once. And maintaining set S for visited vertices. Each edge in the adjacency list is examined only once.

Running time:
When using a heap to implement the priority queue Q, the above algorithm will execute in $O(m\ log\ n)$ time, where n is the number of edges and m is the number of vertices.

- 14.7.17

In a ***side-scrolling video game***, a character moves through an environment from, say, left-to-right, while encountering obstacles, attackers, and prizes. The goal is to avoid or destroy the obstacles, defeat or avoid the attackers, and collect as many prizes as possible while moving from a starting position to an ending position. We can model such a game with a graph, G, where each vertex is a game position, given as an (x, y) point in the plane, and two such vertices, v and w, are connected by an edge, given as a straight line segment, if there is a single movement that connects v and w. Furthermore, we can define the cost, $c(e)$, of an edge to be a combination of the time, health points, prizes, etc., that it costs our character to move along the edge e (where earning a prize on this edge would be modeled as a negative term in this cost). A path, P, in G is ***monotone*** if traversing P involves a continuous sequence of left-to-right movements, with no right-to-left moves. Thus, we can model an optimal solution to such a side-scrolling computer game in terms of finding a minimum-cost monotone path in the graph, G, that represents this game. Describe and analyze an efficient algorithm for finding a minimum-cost monotone path in such a graph, G.

As stated, a path P in a graph G exists if and only if it can be traversed entirely by moving left to right.

The resulting graph will be a Directed Acyclic Graph (DAG), and to determine the single source shortest distance, we can compute the topological ordering of n vertices (game positions) in a m edge DAG.

Initialize the distance to the source to 0 and the number of vertices to infinite.

Finding topological ordering of the graph represent as linear ordering. updating distances of its adjacent vertex using the distance of the current vertex while processing each vertex in topological order.

Running Time:
The algorithm's execution time is $O(n + m)$, where n are the DAG G's vertices and m are its edges. The algorithm processes all the vertices and executes a loop for each one after determining topological order. Graphing adjacent vertices and treating them as $O(m)$. Time taken by the inner loop is $O(n + m)$.

- 14.7.20

Suppose you are given a *timetable*, which consists of the following:

- A set $\mathcal{A}$ of n airports, and for each airport $a \in \mathcal{A}$, a minimum connecting time $c(a)$
- A set $\mathcal{F}$ of m flights, and the following, for each flight $f \in \mathcal{F}$:
  - Origin airport $a_1(f) \in \mathcal{A}$
  - Destination airport $a_2(f) \in \mathcal{A}$
  - Departure time $t_1(f)$
  - Arrival time $t_2(f)$.

Describe an efficient algorithm for the flight scheduling problem. In this problem, we are given airports a and b, and a time t, and we wish to compute a sequence of flights that allows one to arrive at the earliest possible time in b when departing from a at or after time t. Minimum connecting times at intermediate airports should be observed. What is the running time of your algorithm as a function of n and m?

As, given:

If a flight *f* origin from the airport $a_1(f)$,
Destination airport $a_2(f)$
Departure time $t_1(f)$
Arrival time $t_2(f)$

Naming n airports as $a_1, a_2, a_3 \ldots\ldots\ldots a_n$, and flights as $f_1, f_2, f_3 \ldots\ldots\ldots f_n$

1. When analyzing the above case as a di-graph, use airports as the vertices(n) and flights as the edges (m). Edges are flights with two weights.
2. Where the edges' associated weights, c(a), can be used to express the minimum connecting time for each airport as c(a).
3. Assume that start and s are the origin airports. The commencement time is s. T is the earliest arrival time.
4. Set the starting time T[s], starting with the vertex s, to 0. For now, with an arrival time of T[a], check for all vertices (a).
5. For vertices keyed by T, create the priority queue Q. If Q is not empty, delete the minimum value from the priority queue.
6. Now consider each and every vertex, let's say w, that is both present in the Q and adjacent to a. choosing on the next flight by giving a different priority For flights f with departure times keyed by t1(f), there is a line (Q1) (f)
7. There is no need to delete the minimum value based on the arrival time (t2(f)) if Q1 is not empty.
8. Relaxing the edges while taking into account how long the nearby edges T[w] take to complete. Update w in the priority queue Q if T[w] is smaller than the time, and then return the earliest arrival time T.

Running time:
The Dijkstra algorithm will execute in $O(m \log n)$ time, where m is the number of edges (flights) in the graph and n is the number of vertices (airports).