Subject _____

Sat Sun Mon Tue Wed Thu Fri
O O O O O O O

Date : / /

Task-1a

I take a list adj_mat. In this list we have no. of vertex + 1 indexes and in each index I created a list which have no. of vertex + 1 indexes and initially values are 0. Then from the inputs, I updated the values which indicate weight.

Task-1b

For making adjacency list, I take a list adj_lst. In this list we have no. of vertex + 1 index and in each index I created a empty list. So, in the empty list, I stored the connected nodes and weight in a tuple.

Task-2

Firstly I represented the graph in adjacency matrix. Then took a list which tracked the nodes are visited or not initially values are 0. In BFS() I take a Q list first enqueue the source node and then dequee it and again enque

Subject

Date : / /

its neighbour nodes into the Q list untill the Q is [] -> empty. and every time update the value ~~0 in~~ 1 in visited list(G1) if the node visited.

Task-> 3

Here I represented the graph in dictionary. Here also I take a visited list which track the nodes are visite or not, initially value is 0. Here in DFS() if the node not visited I appended the node in output and update in ~~with~~ visited list and for i every neighbour I call the same DFS().

Task-> 4

In this task, I use topological sort algorithm which in Kahns algo. Since we know, topological sort only possible in DAG. so after the sort, if the sorted list length = no. vertex then there is no cycle. Otherwise there is a cycle.

Subject

Date: / /

I created a indegree array. and modified the BFS algorithm

Task -> 5

Here I take a list which counts the distance, initially -1 value and a ancestor list which track the ancetors of every nodes. Than I used BFS algorithm and every time I enquee the nodes I updated both the distance list (d) and ancester list.

Sat Sun Mon Tue Wed Thu Fri
O O O O O O O

Date : / /

Task->6

The code uses a flood-fill algorithm to explore a 2D grid representing a jungle with diamonds ('D') and obstacles ('#'). It finds the max number of diamonds that can be collected by starting from each empty cell ('.') and avoiding obstacle.