①

ⓐ Adjacency Matrix :-

vertices	0	1	2	3	4	5	6
0	0	6	5	5	0	0	0
1	0	0	0	0	-1	0	0
2	0	-2	0	0	1	0	0
3	0	0	-2	0	0	-1	0
4	0	0	0	0	0	0	3
5	0	0	0	0	0	0	3
6	0	0	0	0	0	0	0

1b) Bellman-Ford Algorithm:-

Round-k	$D^k[0]$	$D^k[1]$	$D^k[2]$	$D^k[3]$	$D^k[4]$	$D^k[5]$	$D^k[6]$
1	0	6	5	5	∞	4	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	5
4	0	1	3	5	0	4	3
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

The algorithm will update the distance which is optimal in each round.

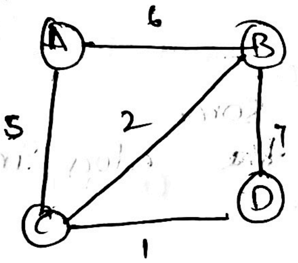
1c) Written seperately. & attached.

Minimum Spanning Tree:-

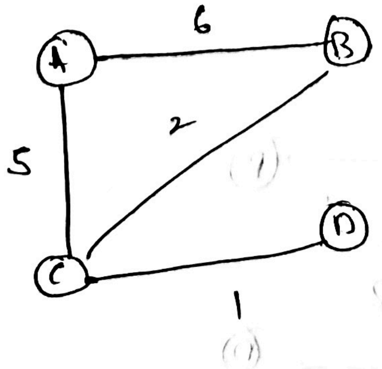
If all edges of a graph are positive then any subset of the edges that connects all vertices in the graph and has minimum weight in total should be a tree.

In a subgraph if it does not have loops (or) cycles and connects all vertices with minimum weight can be a tree.

Consider the following example



The minimal weight in total is obtained when all the nodes are connected as follows and has a cycle

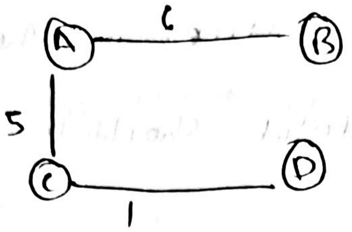


Total weight = 6+5+2+1

= 14

But it is not a tree.

Lets drop the edge that is forming a cycle.



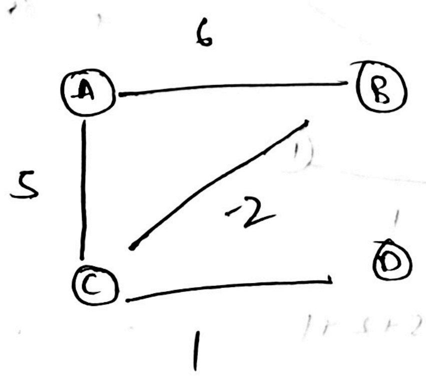
Total weight = 6+5+1

= ~~4~~ = 12 — ⓐ

We can see that these edges connelt all nodes with minimum possible total weights. So, the above graph is a minimum spanning tree.

But it won't work incase ~~the~~ some edges in the graph are negative.

For example in the same example if the edge making a cycle is negative then.



so, Total weight = 6+ 5-2+1 = 10.

= ~~Less~~ 10 .

But this is smoller than - ⓐ so it won't be minimum spanning tree if the edges are negative.

Q3)

Let's check whether the vertex pair v and u are realhable using depth first search algorithm. we will traverse graph (G). from vertex u and mark all that are reachable from u. If v is marked as visited then it is reachable from u.

Algorithm :-

```
ISREACHABLE (u, v, visited, G)
    if u==v
        return True
    visited[u] = True
    for adjacent in G[u]:
        if not visited[adjacent]
            if ISREACHABLE(adjacent, v, visited, G)
                return True
    return False
```

Complexity:-

As the procedure checks all adacent nodes from current node starting from u. In worst case scenario it should explore V vertices connected with E edges. So,

The complexity would be $O(V+E)$

(4) Given, some studenth need to participate in an athletic event which is represented as a graph where nodes are connected if the events have no time conflict.

Implementation:-

Let's Student-Enrolled-Events be an input which contains the information of the student enrolled events in the athletics. We are iterating through all the student events and for each event we will check whether the current event is in the list of events in the graph called Graph_AdJacency-List which represents nodes & its connected nodes with no time conflict.

So, if the student enrolled events has conflict we will return false or we will return true.

Algorithm :-

```
EVENTTIME CONFLICT ( Student-Enrolled-Events)

  Graph-Adjacency-List = { A:[C], B:[F], C:[A,D], D:[C,E], E:[D],
                           F:[B,D] }

  for each event in Student-Enrolled-Events:
      for each event-other in Student-Enrolled-Events:
          if event-Other == event:
              Continue
          if event-Other not in Graph-Adjacency-List[event]:
              return F
  return T.
```