Homework Assignment 8

CS 430 Introduction to Algorithms

Name : Karthik Mahesh

CWID : A20383027

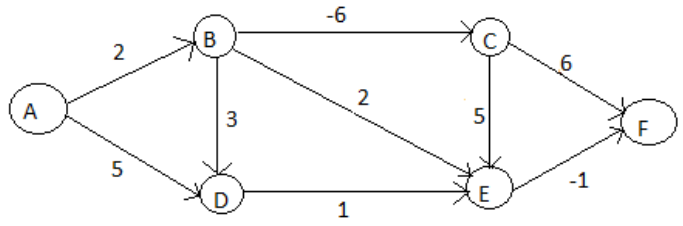
1.Solution:

A.

No. The shortest path between any pair of vertices, u and v, in the original graph does not remain the same after the modification.

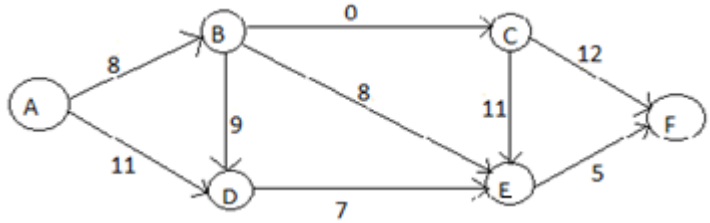
Proof :

Consider the below graph:



Here, shortest path from A to E is ABCE. Total cost is 2-6+5=1
And shortest path from A to F is ABCEF. The total cost is 2-6+5-1=0

Here the minimum weight w_e to make all the weights positive is 6.
When we add 6 to all the weights of all the edges, the graph becomes as below:



Now, shortest path from A to E is ABE. The total cost is 8+8=16
And shortest path from A to F is ABCF. The total cost is 8+0+12 = 20

The shortest path from A to E and A to F changed after adding min w_e(6) to all the weights.
Therefore we can conclude that the shortest path between any pair of vertices, u and v, in the original graph **does not** remain the same after the modification.

B:

We cannot determine a shortest path between 2 vertices when there is a negative weight cycle.
So we cannot determine shortest path in the original graph.
We can determine the shortest path after modification but it cannot be compared to the shortest path of the original graph.

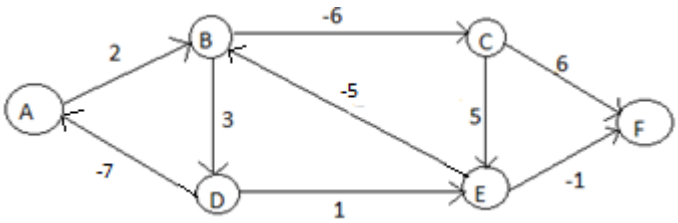
2.Solution :

A.

Source value will be changed only if there is a negative cycle
Proof by contradiction:
Lets assume there was no negative cycle
We want to reach source after starting from source
Initial value or label at source will be 0.
In bellman ford algorithm, the label changes only if there is a way we can reach that vertex in lesser cost.
So if 0 is changing, there has to be a way we can reach source with –ve cost.
Once we start from the vertex, we cannot reach to that vertex without a cycle.
So if the value at the source changes, it shows that there is a negative cycle in that graph.

Therefore we can conclude there is a –ve cycle if the label at the source changes.

B.



In the above graph, Values of all the vertices keep changing when we run bellman ford algorithm
v.d value will always be less than u.d + w(u,v)
The for loop never ends because of infinite edge relaxations.

3.Solution:

```
Shortest(w1, w2):
  w12 = a new V x V matrix
  for i in V
    for j in V
      w12(i, j) = w1(i, j)
      for k in V:
        if w12(i, j) > w1(i, k) + w2(k, j):
          w12(i, j) = w1(i, k) + w2(k, j)
  return w12
```

w2 = shortest(w, w)
w3 = shortest(w2, w)
.
.
W_k = shortest(w_{k-1},w)

Here W2 gives shortest path from i to j with 2 edges in between.
W_k gives shortest path from i to j with k edges in between.

Time Complexity = $O(n^3 \log k)$

4.Solution:

Associate a potential of k with the current state k of the algorithm
Φ = k where k represents current state of algorithm

```
COMPUTE-PREFIX-FUNCTION(P)
   m = P.length
   π(1) = 0
   k = 0   ("Initial Potential value is 0")
   for q = 2 to m
         do while k > 0 and P(k+1) != P(q)
                  do k = π(k)
("Here the potential k decreases since π(k)<k and k is never negative since
π(k) >= 0 for all k")
                  if P(k + 1) = P(q)
                           then k = k+1
("Potential increases by <=1 in each execution of for loop body")
                  π(q) = k
   return π
```

Amortized cost = actual cost + potential increase
Amortized cost of the loop body is in O(1)

There are O(m) iterations
Therefore time complexity for Compute Prefix function will be O(m)

A similar amortized analysis, using the value of q as the potential function, shows that the matching time of KMP-MATCHER is O(n).

Therefore total time complexity for KMP algorithm is O(n+m)

5.Solution:

In order to prove that finding a spanning tree where every node has a degree less than or equal to k is NP-Complete, we must show that given a solution S to the k-Spanning Tree problem, that we can check in polynomial time whether it is in fact a k-Spanning Tree.
This amounts to verifying that every node in the original graph G is used in solution S, that S has no cycles (i.e. it is a tree), and that every node in the tree has degree at most k. All of these can be checked efficiently in polynomial time, and therefore k-Spanning Tree is a search problem.
Therefore finding a spanning tree where every node has a degree less than or equal to k is NP-Complete.

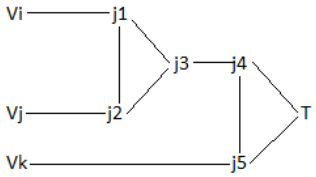
6.Solution :

Proof: (1) In NP: witness is a 3-coloring.
(2) Reduce 3-SAT to 3-COLORING.
(3) Given a 3-SAT formula of m clauses on n variables x1,x2,...,xn, we construct a graph G as follows. We have
(a) a vertex vi for each variable xi,
(b) a vertex vi' for the negation of each variable xi,
(c) 5 vertices j1-j5 for each clause j,
(d) 3 special vertices: T, F, R

We would like T, F, and R to be forced to different colors, so we will add edges between them to form a triangle. For the remaining nodes, and node that is colored the same color as T/F/R will be called colored TRUE/FALSE/RED, respectively.

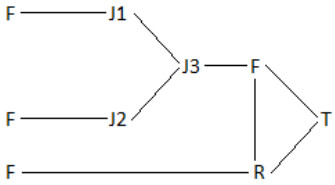
We would like the edges to enforce the constraints on satisfying assignments.
Constraint: For all i, exactly one of vi and vi' is colored TRUE and one is colored FALSE.
Edges: for each i, form a triangle between vi, vi', and R.

Constraint: For each clause j, at least one of the literals in the clause is colored TRUE.
Edges: for each clause j, say = (xi or not(xj) or xk), we have the following gadget

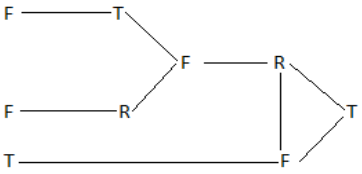


Claim: If each of vi, vj', and vk is colored TRUE or FALSE, then gadget is 3-colorable iff at least one of vi, vj', and vk is colored TRUE.

Proof: If vi, vj', and vk are all colored false, then we are forced to the following colors:



But then j1, j2, j3 all must be colored different colors and NONE can be colored F, so there is no legal coloring.
The remainder of the proof considers the 7 possible combinations of coloring vi, vj', and vk such that at least one is colored TRUE and the rest are colored FALSE, and shows that a 3 coloring exists in each case.
As an example, if vk is colored TRUE but vi and vj' are colored FALSE, we have the following legal 3-coloring:



The other cases are similar.
The construction takes polynomial time.

(4) Follows from the above arguments.

Thus 3-COLORING is NP-complete.

References Used:

CLRS2 Text book
https://piazza-resources.s3.amazonaws.com
http://www.cs.cmu.edu
stackoverflow.com