CONCORDIA UNIVERSITY

DEPARTMENT OF COMPUTER SCIENCE AND SOFTWARE ENGINEERING

COMP 6651: Algorithm Design Techniques

Winter 2022

Quiz # 5

First Name	**Last Name**	**ID#**

Question 1

Consider the flow network that is represented in Figure **??**. Consider the flow f that is represented in Figure **??**, with the notation $f(\ell)/c(\ell)$ at each link. If the flow optimal? Justify your answer. If not optimal, can you explain how to compute an optimal flow?

Solution:

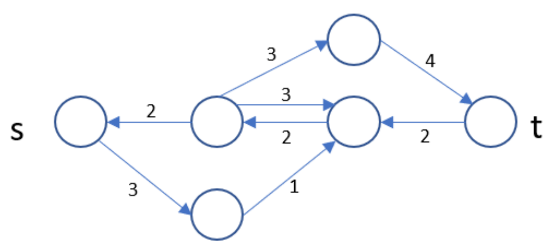

Figure 1: Residual graph

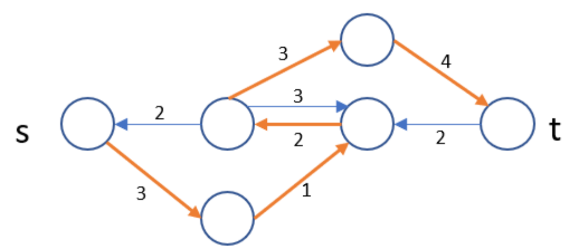

Figure 2: Augmented path

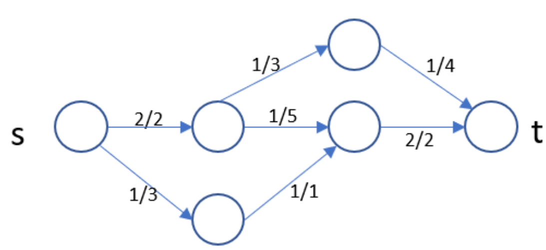

Figure 3: Optimal flow

From the residual graph of the current flow, we can find an augmented path, so the given flow is not optimal.

By adding the augmented path into the given flow, we obtain a new flow which is optimal (max flow value = 3)

Answering that flow is not optimal (1 point)
Residual graph (2 points)
Augmented path (1 points)
Optimal flow (1 points)

Question 2

1. Recall Dijkstra's algorithm
2. In breadth first search, recall at what time each vertex gets a 'visited' field: just before the vertex is removed from the queue or when the vertex is added to the queue? What happens if BFS instead uses the other rule? Does the algorithm still work? Does it run just as fast?
3. Dijkstra's algorithm. Suppose we want to find the shortest distance from s to some particular vertex (rather than to all vertices reachable from s), e.g., This is the problem that google maps solves. What would we do ?

Solution:

1. Check the slide

2. You should change the visited status of a vertex to TRUE at the time you enqueue it. If you do it at the time you removed it from the queue, although the algorithm still works, there would be multiple copy of that vertex (if the graph is not a tree) in the queue.

3. Consider all the edges have positive weight, then you can stop the exploration when the DISTANCE value of the node that you removed from the queue is bigger than or equal the DISTANCE values of all the destinations.

```
Dijkstra(G, s, d-array):
Q:= V[G]
for v ∈ V[G] do
    dist[v] := Infinity
    prev[v] := None
end for
dist[s] := 0
while not IsEmpty(Q) do
    u := ExtractMin(Q)
    if u ∈ d-array then
        Remove(d-array, u)
        if IsEmpty(d-array) then return dist, prev
        end if
    end if
    for v ∈ Adj[u] do
        t := dist[u] + w(u, v)
        if t < dist[v] then
            dist[v] := t
            prev[v] := u
        end if
    end for
end while
```

Question 1 (2 points)
Question 2 (1 points)
Question 3 (2 points)