CSCI-GA.1170-001 Homework 12

Ankit Sati

TOTAL POINTS

42 / 35

QUESTION 1

1 Zero-Weight Cycles and Shortest Paths

22 / 14

✓ **- 0 pts** *a,b,c Correct*

✓ **+ 3 pts** *12-1) e) extra credit correct*

✓ **+ 2 pts** *12-1) d) (extra credit) algo correct*

✓ **+ 3 pts** *12-1) d) (extra credit) correctness correct*

QUESTION 2

2 Modified Dijkstra - Too Slow? 8 / 10

✓ **- 0 pts** *Correct*

✓ **- 2 pts** *Partial points to proof in part (c)*

1 You are explaining how the algorithm works. Your proof needs to be more rigourous.

QUESTION 3

3 Testing the Tester! 12 / 11

(a)

✓ **- 0 pts** *Correct*

(b)

✓ **- 0 pts** *Correct Case-1*

✓ **+ 3 pts** *Correct Case-2 - Follows from Check 2,4*

✓ **+ 2 pts** *Partially correct Case-3 (proper reason for "why shorter path exists mathematically" not specified, etc.)*

✓ **+ 2 pts** *Partially correct Case-4 (reasoning, etc.)*

(c)

✓ **- 1.5 pts** *Incorrect/Missing $$w'_e$$ blank*

✓ **- 1.5 pts** *Incorrect/Missing Reasoning*

(d)

✓ **- 1 pts** *Incorrect/Missing $$w'(P)$$ blank*

✓ **- 1 pts** *Incorrect/Missing Derivation of the Equation (telescopic sum is main)*

✓ **- 1 pts** *Incorrect/Missing Reasoning behind statement "min-cost path in G is not affected by the new weights"*

(e)

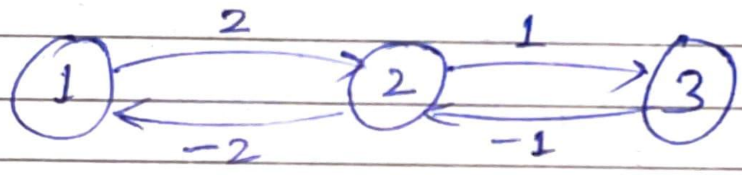
✓ **- 0 pts** *Incorrect/Missing solution*

Problem 12-1

(a)

Base Case:

Let us consider a graph with 2 vertices as shown:



A zero weight cycle can be $1 \to 2 \to 3 \to 2 \to 1$

Now, it can be further broken into two simple zero-weight cycles $1 \to 2 \to 1$ and $2 \to 3 \to 2$.

Induction Hypothesis:

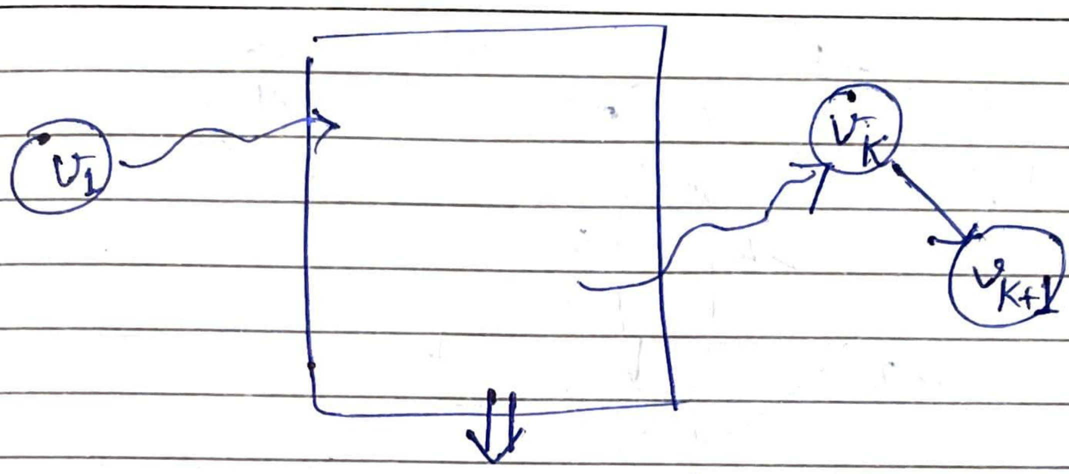
Let us consider a graph with K vertices and assume that it follows the given statement that a zero-weight cycle can be decomposed. into further simple zero-weight cycles.

Induction step:

Let us now consider the graph has K+1 vertices.

So, $V_{K+1} = V_K \cup v_{K+1}$

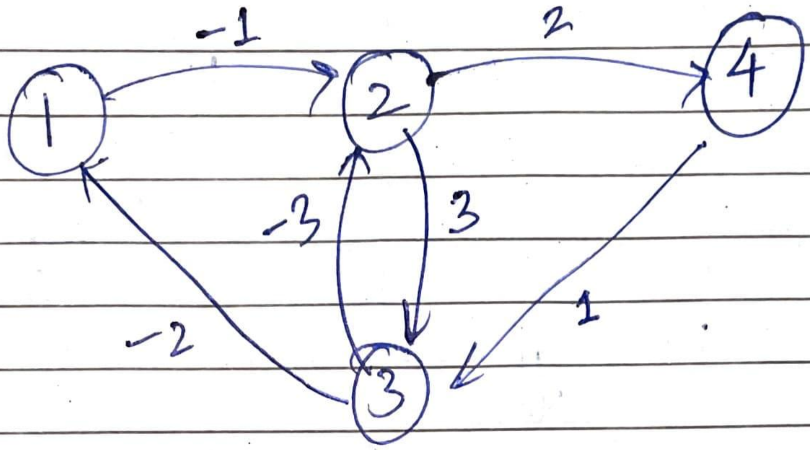
Basically, the cycles in graph with K vertices could be decomposed into simple cycles.



This is the region where some vertices are there which are present in more than one cycle

Now, Since the $(k+1)$th vertex is a new node added to the graph with K vertices. So, this new graph formed also can decomposed further into simple cycles.

eg:



The cycle $1 \to 2 \to 3 \to 2 \to 4 \to 3 \to 1$

can be decomposed into

$1 \to 2 \to 4 \to 3 \to 1$, $2 \to 3 \to 2$, $1 \to 2 \to 3 \to 1$

and $2 \to 4 \to 3 \to 2$

(b)

Given: R is relation such that $(u,v) \in R$ if both u and v lie on some cycle with zero weight.

Reflexive:

$(a,a) \in R \quad \forall a \in V$

The shortest weight from a node to itself is 0 which is found by moving to no other vertex from a.

So, R is reflexive.

Symmetric:

Let us consider $(a,b) \in R$

This means a and b lie

on some zero weight cycle of

graph.

So, we can say that b and a

also lie on the same zero

weight cycle.

So, $(b,a) \in R$

Since $(a,b) \in R$ and $(b,a) \in R$

So, R is symmetric

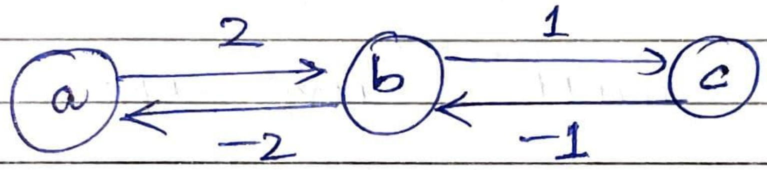
Transitive:

Let $(a,b) \in R$ and $(b,c) \in R$

So, a and b lie on some zero-weight cycle. Also, b and c lie on some zero weight cycle. So, a, b and c all lie on some zero weight cycle.

So, $(a, c) \in R$

eg.-



$(a, b) \in R$ for the cycle $a \to b \to a$

$(b, c) \in R$ for the cycle $b \to c \to b$

$(a, c) \in R$ for the cycle $a \to b \to c \to b \to a$

Therefore R is transitive.

Since R is reflexive, symmetric, and transitive. So, R is an equivalence relation.

(c)

For the adjacency matrix of the graph, we will put $Adj[u,u] = \infty$ which means the distance with node u to itself is ∞ initially. Now, we will run the Floyd - Warshall Algorithm and check if $Adj[u,u]$ is 0 for some $u \in V$

Algorithm →

FLOYD - WARSHALL(W)

$n \leftarrow rows[W]$
$D^0 \leftarrow W$

for $K \leftarrow 1$ to n
{ do for $i \leftarrow 1$ to n
{ do for $j \leftarrow 1$ to n
{
if $d_{iK}^{K-1} + d_{Kj}^{K-1} < d_{ij}^{K}$:
{ $d_{ij}^{K} = d_{iK}^{K-1} + d_{Kj}^{K-1}$
$\Pi_{ij} = K$ }
}
}
}
for $i = 1$ to n
{
if $d_{ii}^{n} == 0$:
{ Take a vector v
$v =$ NODES_IN-CYCLE(i, Π)
}
}

NODES-IN-CYCLE (i, Π)

· Backtrack from node i over parent matrix Π until you again get i

Running Time:

This algorithm is mainly Floyd-Warshall algorithm with slight changes. So, runtime is $O(|V|^3)$.

Correctness -

We did a run of Floyd-Warshall-Algorithm to know if distance between a node u to itself is becoming 0. In this case, we call the function NODES-IN-CYCLE which backtracks from current node until it gets back to that node again. So, in this way we are keeping track of all the vertices found in the zero weight cycle. And hence the algorithm is justified.

(d)

Algorithm →

FLOYD - WARSHALL(W)

n ← rows[W]
D^0 ← W

for K = 1 to n
    for i = 1 to n
        for j = 1 to n

if $d_{ik}^{k-1} + d_{kj}^{k-1} < d_{ij}^{k}$:

$d_{ij}^{k} = d_{ik}^{k-1} + d_{kj}^{k-1}$

$\Pi_{ij} = k$

for i = 1 to n

if $d_{ii}^{n} == 0$

vector<int> v = NODES_IN-CYCLE (i, Π)

m = INT-MAM
for i = 0 to v.size()
m = min (m, v[i])

i . equivalence = m

Approach →

Our algorithm is similar to the

algorithm used in previous part.
Here after finding the vector of vertices which are involved in the cycle, we find the minimum of all those and store in equivalence property of every vertex.

In this way, u. equivalence will store the minimum node v value for any node v which is present in the same zero weight cycle in which u is also present.

In this way the algorithm is justified.

(2)

1(e)

Since there is some zero-weight cycle which includes vertices h and i. Also, there is one another zero-weight cycle which includes vertices j and K. Since there would be some balancing edges whose weights must be cancelling each other's weight, so it will be optimal.

to consider the shortest path from h to i and shortest path from j to k.

So, for the shortest path from h to k, we can go via shortest path from h to i, then via shortest path from i to j and then via the shortest path from j to k.

This will provide us the shortest path between h and k.

So, $\delta(h,k) = \delta(h,i) + \delta(i,j) + \delta(j,k)$.

1 Zero-Weight Cycles and Shortest Paths **22 / 14**

✓ **- 0 pts** *a,b,c Correct*

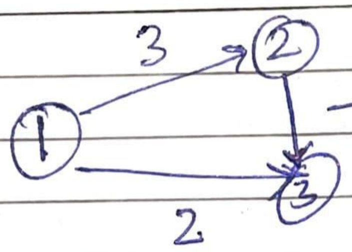
✓ **+ 3 pts** *12-1) e) extra credit correct*

✓ **+ 2 pts** *12-1) d) (extra credit) algo correct*

✓ **+ 3 pts** *12-1) d) (extra credit) correctness correct*

Problem 12-2

(a) The required example is:



If we remove step 12, then the shortest path from 1 to 3 will come 2, which is wrong as the path is -1 which comes on going $1 \rightarrow 2 \rightarrow 3$.

(b)

In the worst case, the shortest path from a source node s to a node v will pass through all the edges. So, the number of iterations will be bounded by the number of edges m. Hence the algorithm terminates in a finite number of steps.

(c)

In the usual Dijkstra algorithm, we donot visit a vertex more than once. This was so because all the weights were considered to be positive in that case So, as soon as the vertex is visited, its distance from source node is the minimum.

But, when we consider negative weights also, then we may have to revisit some node to find even more shorter path. This thing is maintained by line 12, where we are.

inserting the node v back to the priority-queue Q so that it may be visited again in the coming iterations to find more shorter paths.

In this way MODIFIED DIJKSTRA well works for the negative weight edge case such that there are no negative weight cycles.

Hence the ~~above~~ above algorithm is justified.

2 Modified Dijkstra - Too Slow? **8 / 10**

✓ **- 0 pts** *Correct*

✓ **- 2 pts** *Partial points to proof in part (c)*

1 You are explaining how the algorithm works. Your proof needs to be more rigourous.

Problem 3

(a) Runtime for each step of Algo.

1. $O(1)$
2. $O(|V|+|E|) + O(|E|) + O(|V|+|E|)$
3. $O(|V|)$
4. $O(|V|+|E|)$ {Single pass of Bellman ford}
5. $O(1)$

$\therefore$ Runtime $= O(|V|+|E|)$

The algorithm will accept this, if this answer is complete.

Note ⇒ Any right answer will be a single source of shortest paths and it should have:

1. $s.d = 0$, $s.\pi = NIL$ for source vertex.
2. The given shortest path is reachable from s.
3. v.d is correctly calculated.
4. This graph cannot be relaxed any further.

If all the above arguments are met & verified then the algorithm will return correct.

Hence, all the right verification are made by the algorithm & will return correct for a right answer.

(4) $\delta(s, v) = \infty$ but $v.d < \infty$

→ v is actually not reachable but $v.d < \infty$ indicates it is still reachable.
Hence this will fail in:
(i) 2.a → There will be no edge $(v.\pi, v)$ as v is unreachable
(ii) 2.c → On running DFS, we fail to reach this node with $v.d < \infty$

→ $\delta(s, v) < \infty$ but $v.d = \infty$

v is actually reachable but $v.d = \infty$ indicates it is not.
It will be caught in:
→ 2.c → On running DFS, it will reach node v with $v.d = \infty$
→ 4. → On the Bellman-Ford pass, graph will relax further, as $v.d$ shouldn't be ∞.

→ $\delta(s, v) < v.d < \infty$

As $\delta(s, v) < v.d$, it means we can further relax v.
Hence it will fail in:
→ 4. - On the bellman ford pass, we will observe relaxations.

→ $v.d < \delta(s, v) < \infty$

This case indicates wrong calculation of $v.d$.

To catch this specific case, we need to find the v where $v.d$ is not matching with $v.\pi.d + w(v.\pi, v)$.

Hence this case will be caught in 3.

3 Testing the Tester! **12 / 11**

(a)

✓ **- 0 pts** *Correct*

(b)

✓ **- 0 pts** *Correct Case-1*

✓ **+ 3 pts** *Correct Case-2 - Follows from Check 2,4*

✓ **+ 2 pts** *Partially correct Case-3 (proper reason for "why shorter path exists mathematically" not specified, etc.)*

✓ **+ 2 pts** *Partially correct Case-4 (reasoning, etc.)*

(c)

✓ **- 1.5 pts** *Incorrect/Missing $$w'_e$$ blank*

✓ **- 1.5 pts** *Incorrect/Missing Reasoning*

(d)

✓ **- 1 pts** *Incorrect/Missing $$w'(P)$$ blank*

✓ **- 1 pts** *Incorrect/Missing Derivation of the Equation (telescopic sum is main)*

✓ **- 1 pts** *Incorrect/Missing Reasoning behind statement "min-cost path in G is not affected by the new weights"*

(e)

✓ **- 0 pts** *Incorrect/Missing solution*