COL351 Fall 2021 Minor Exam

Tamajit Banerjee

TOTAL POINTS

34.5 / 42

QUESTION 1

Short Questions 15 pts

1.1 (a) 2 / 3

✓ **- 1 pts Incomplete calculations, not showing one sum is m-|LCS(A,b)| another sum is n-|LCS(A,b)|.**

1.2 (b) 3 / 3

✓ **- 0 pts For Correct**

1.3 (c) 3 / 3

✓ **- 0 pts Correct solution and proof**

1.4 (d) 3 / 3

✓ **- 0 pts Correct algorithm and proof**

1.5 (e) 3 / 3

+ **0 pts** Incorrect / Not attempted

✓ **+ 1 pts Correct dfs tree**

✓ **+ 1 pts correct start time, end time**

✓ **+ 1 pts correct topological ordering**

+ **0.5 pts** Reverse topological ordering

QUESTION 2

2 Gemstones 6 / 6

+ **0 pts** Incorrect / did not attempt

✓ **+ 4 pts Correct formation of optimal dp array**

✓ **+ 1 pts Forming correct list of indices**

✓ **+ 1 pts Proof of correctness**

+ **2 pts** Partially correct dp array / recursive method

+ **1 pts** Inefficient solution

+ **1 pts** Partial

- **6 pts** Cheating

QUESTION 3

Distances 12 pts

3.1 (a) 3 / 3

✓ **+ 3 pts Correct**

+ **1.5 pts** Only shown $$wt^*(x,y) \geq 0$$

+ **1.5 pts** Only shown "computable in $$O(mn)$$"

+ **0 pts** Incorrect / Unattempted

3.2 (b) 3 / 3

✓ **+ 3 pts Correct**

+ **0 pts** Incorrect / Unattempted

3.3 (c) 2 / 2

✓ **+ 2 pts Correct**

+ **1 pts** Only one direction proved

+ **0 pts** Click here to replace this description.

3.4 (d) 3 / 4

✓ **+ 4 pts Correct**

+ **2.5 pts** Proof Missing

+ **0 pts** Incorrect / Unattempted

- 1 Point adjustment

1 the shortest path is the same, but you needed to argue that the desired value i.e. dist between all pairs has been computed correctly by your algorithm

QUESTION 4

Circumference 9 pts

4.1 (a) 3.5 / 6

✓ **- 2.5 pts Currently identifying \gamma and sorting the sequences, however not computing Correct "trees" in auxiliary graph to compute global minima.**

4.2 (b) 0 / 3

✓ **- 3 pts Incorrect / Not attempted**

Tanu

9 (a)

$m+n-2(LCS(A,B))$

of $m+n-2|LCS(A,B)|$

we can at least match the strings at the LCS position and match the other with blank

∴ the length is bounded by

$m+n-2|LCS(A,B)|$

1.1 (a) **2 / 3**

✓ **- 1 pts** Incomplete calculations, not showing
one sum is m-|LCS(A,b)|
another sum is n-|LCS(A,b)|.

1(b)

Prove:

An edge is a bridge if and only if it does not lie on any simple cycle of G.

Prove: (1) Bridge implies does not lie on any simple cycle of G

Contrapositive statement: ; it lies on any simple cycle of G implies an edge is not a bridge.

Proof: Let $e = (x, y)$

: it lies on a simple cycle, there is also a path in $G \setminus \{e\}$ from x to y following the simple cycle

but by definition for a bridge, each path must pass through edge (x, y), so it is not a bridge.

* Hence, proved.

Prove: (2) It does not lie on any simple cycle of G implies it is a bridge.

Proof by contradiction: Let $e = (x, y)$

Suppose that it is not a bridge, then there would have been a path from x to y in $G \setminus \{e\}$

But, Let the path be P.

P begins at a and ends at j

and now if we add the edge e = (x, y)

we will get a simple cycle $\{e\} \cup P$

which is a contradiction.

Hence, proved.

1.2 (b) 3 / 3

✓ - 0 pts For Correct

1.)(c)

Algorithm

1. Select any random vertex . Suppose A.

2. Run BFS on the directed graph from A

3. See if all the vertices are reachable
   else return (NOT SCC)

4. Run BFS on the directed graph but now with the edges reversed from A.

5. See if all the vertices are reachable
   else return (NOT SCC)

6. Return, single SCC

Time Complexity of steps 1.) O(1)

2.) O(m+n) ← BFS

3.) O(1)

4.) O(m+n) ← BFS

5.) O(1)

1.1(c) Correctness

(a) The 1st BFS ensures that all vertices are reachable or not

i.e. there exists a path from A to B $\forall B \in V$.

(b) The 2nd BFS ensures that all vertices can reach A or not.

(c) ∵ all vertices are reachable from A and all vertices have a path to A implies that all the vertices are in a single SCC as SCC property is transitive

1.3 (c) **3 / 3**

✓ **- 0 pts** Correct solution and proof

1(d)

(a) Algorithm:

(1) Perform DFS using st and end time ← list stores the order

(2) $L =$ & Produce an Topological sorting of the DFS

(3) ~~Iterate from 1 to n~~

for i ← 1 to n-1

if (there does not exist an edge from L[i] to L[i+1])

return FALSE (NOT UNIQUE)

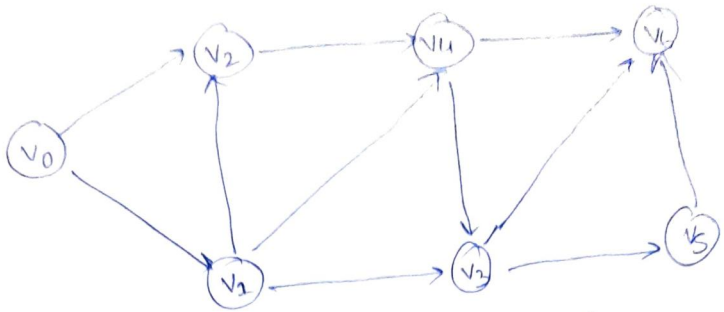
(4) return TRUE

Proof: The above condition shows that if there is no edge, we can reorder them and so not unique else we always have to put i before i+1

1.4 (d) 3 / 3

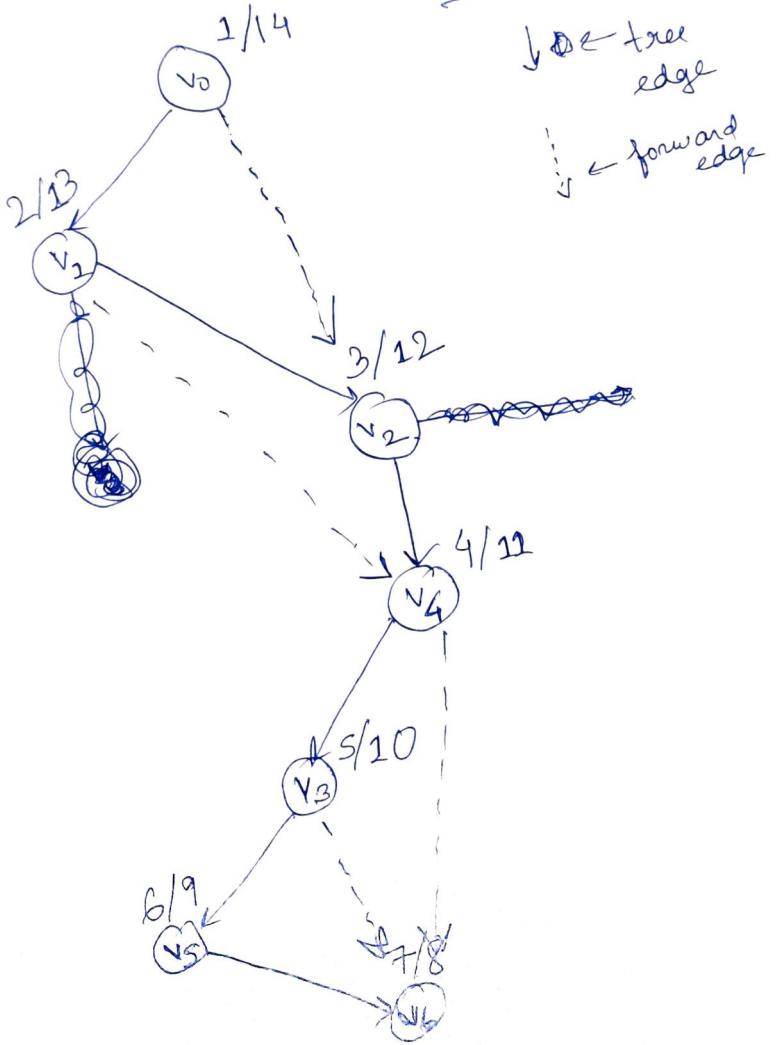
✓ **- 0 pts** Correct algorithm and proof

17(c)



DFS tree

start time x/y finish time



the topological ordering

V_0	V_1	V_2	V_4
14	13	12	11

(Ans)

1.5 (e) **3 / 3**

+ **0 pts** Incorrect / Not attempted

✓ **+ 1 pts** **Correct dfs tree**

✓ **+ 1 pts** **correct start time, end time**

✓ **+ 1 pts** **correct topological ordering**

+ **0.5 pts** Reverse topological ordering

2.) Algorithm

1.) Initialise an array Dp of length k+1 with 0

2.) Initalise an array P of length k+1 containing the number which gave the maximum

3.) for i ← 1 to k:
    for j ← 1 to i:
        if (F(j) + Dp[i-j] > Dp[i]):
            Dp[i] ← F(j) + Dp[i-j]
            P[i] ← j

4.) List L ← []

5.) cur ← k

6.) while (cur != 0):
        L.append(P[cur])
        cur ← cur - P[cur]

7.) Output L.

Time complexity:

1) Initialise ← $O(k)$

2) Intialise ← $O(k)$

3) Inner loop runs for i time for $i = 1$ to k

⇒ the total time : $O(k^2)$

4) while loop cur decreases at least by 1 each time so $O(k)$

Proof of correctness

① $Dp[i]$ is the maximum overall profit by partitioning i gemstons.

Recursion :

$$Dp[i] = \max(F(j) + Dp[i-j]) \quad j \in \{1, 2, \ldots, i\}$$

Base case:

$Dp[0] = 0$ which is true

For a value of i gemstons let j gemstones be one of the partition values $j \in [1, i]$

$\therefore Profit_i = F(j) + Profit_{i-j}$

which is exactly what $Dp[i-j]$ stores

hence proved

Time complexity:

1) Initialise ← O(k)

2) Intrelise ← O(k)

3) Inner loop runs for i time for i = 1 to k

⇒ the total time = $O(k^2)$

4) while loop var decreases at least by 1 each time so O(k)

Proof of correctness

① Dp[i] is the maximum overall profit by partitioning i gemstons.

Recursion:

$$Dp[i] = \max(F(j) + Dp[i-j]) \quad j \in \{1, 2, \dots, i\}$$

Base case:

$$Dp[0] = 0 \text{ which is true}$$

For a value of i gemstons let j gemstones be one of the partition values $j \in [1, i)$

$\therefore$ $Profit_i = F(j) + Profit_{i-j}$

which is exactly what Dp[i-j] stores

hence proved

2 Gemstones 6 / 6

+ **0 pts** Incorrect / did not attempt

✓ **+ 4 pts** **Correct formation of optimal dp array**

✓ **+ 1 pts** **Forming correct list of indices**

✓ **+ 1 pts** **Proof of correctness**

+ **2 pts** Partially correct dp array / recursive method

+ **1 pts** Inefficient solution

+ **1 pts** Partial

- **6 pts** Cheating

3(a) Ist, for each edge $(x, y) \in E$,

$wt^*(x, y) \geq 0$

~~we know~~ $dist_G(s, x)$ = shortest distance of x from s

$dist_G(s, y)$ = shortest distance of y from s

By ~~triangle~~ triangle property, we know,

$dist_G(s, y) \leq dist_G(s, x) + wt(x, y)$

$\Rightarrow 0 \leq wt(x, y) + dist_G(s, x) - dist_G(s, y)$

$\Rightarrow \underline{0 \leq wt^*(x, y)}$

Also, $wt^*(x, y)$ is computable in $O(mn)$.

this is because by Bellman Ford Algorithm we can find the shortest path from a source to all vertices in $O(mn)$ time even if the edges are negative.

Algo, we can compute $wt^*(x, y)$ with the knowledge of $dist_G(s, v)$. : After applying Bellman-Ford

3.1 (a) **3 / 3**

✓ **+ 3 pts Correct**

+ 1.5 pts Only shown $$wt^*(x,y) \geq0$$

+ 1.5 pts Only shown "computable in $$O(mn)$$"

+ 0 pts Incorrect / Unattempted

3(b)

For a path $P(v_0, \ldots, v_t)$, ~~$wt^*(P) + dist_G$~~

$wt^*(P) = wt(P) + dist_G(s, v_0) - dist_G(s, v_t)$

$\therefore wt^*(P) = \sum_{e \in \{v_i v_{i+1}\}} wt^*(v_i, v_{i+1})$

$= \sum_{e \in \{v_i v_{i+1}\}} \left(wt(v_i, v_{i+1}) + dist_G(s, v_i) - dist_G(s, v_{i+1}) \right)$

$= \sum_{e \in \{v_i, v_{i+1}\}} \left(wt(v_i, v_{i+1}) \right) + 0 \; dist_G(s, v_0) - dist_G(s, v_t)$

[as rest gets cancelled alternating]

$= wt(P) + dist_G(s, v_0) + dist_G(s, v_t)$

(Ans)

3.2 (b) **3 / 3**

✓ **+ 3 pts** **Correct**

+ 0 pts Incorrect / Unattempted

(c) Proof (i)

(i) wt(P) shortest $\Rightarrow$ $wt^*(P)$ shortest

Let the shortest path from x to y according to the weight function wt be wt(P)

$\therefore$ shortest

$$wt(P) \leq wt(P') \quad \forall P' \neq P$$

$$\Rightarrow wt(P) + dist_G(s,x) - dist_G(s,y)$$

$$\leq wt(P') + dist_G(s,x) - dist_G(s,y)$$

[Adding constant]

$$\Rightarrow wt^*(P) \leq wt^*(P') \quad [\text{from (b)}]$$

Proof (ii) $wt^*(P)$ shortest $\Rightarrow$ $wt^*(P)$ shortest

$\therefore$ shortest

$$wt^*(P) \leq wt^*(P') \quad \forall P' \neq P$$

$$\Rightarrow wt^*(P) + dist_G(s,y) - dist_G(s,x) \leq wt(P') + dist_G(s,y) - dist_G(s,x)$$

$$\Rightarrow wt(P) \leq wt(P')$$

Hence proved

3.3 (c) **2 / 2**

✓ **+ 2 pts** **Correct**

+ 1 pts Only one direction proved

+ 0 pts Click here to replace this description.

3(d)

Algorithm:

Assuming we have the G input

1.) Compute G', where G' = (V, e, wt)

2.) Append a new vertex s to G, G ∪ {s}
and add directed edges from s to all the vertices and take w(s, v) = 0.

Let $G_1 = G'$

3.) Compute BELLMAN-FORD : (G', w, s)

4.) We will get all the edge weight
$wt^*(x, y)$ ∀ edges (x, y)
$wt^*(x, y) = wt(x, y) + dist_G(s, x) - dist_G(s, y)$

5.) Let D = $wt^*(x, y)$ be a new $n \times n$ matrix.

6.) For each vertex u in the graph G_1,
run DIJKSTRA (G', wt^*, u) to compute
the $dist'_{G'}(u, v)$ ∀ $v \in G'$

3(d) To compute the final distance,

recompute

$$dist_{uv} = dist'_G(u,v) + dist_G(s,v) - dist_G(s,u)$$

8) Output D matrix.

Time complexity is $O(mn \log n)$

② we run DIJKSTRA Algorithm which takes $O((m+n)\log n)$ n times

$\therefore$ total = $O(mn \log n)$

Proof: (i) the edge weights are non negative Dijkstra outputs the correct shortest path distance for $\forall u$.

Hence, proved

3.4 (d) **3 / 4**

✓ **+ 4 pts** **Correct**

+ **2.5 pts** Proof Missing

+ **0 pts** Incorrect / Unattempted

- 1 Point adjustment

1 the shortest path is the same, but you needed to argue that the desired value i.e. dist between all pairs has been computed correctly by your algorithm

4) (a)

Algorithm to

Order

1) compute (a_i, b_i) all the (a_i, b_i) with respect to (a_{i_0}, b_{i_0})

2) Subtract a_{i_0} from all the arc coordinates and if the coordinates become < 0 add 2π to it and the 0 the coordinate

3) Sort the arcs in the circle in the increasing order of b_i / anticlockwise angle w.r.t. a_{i_0}

4) Initialise a loop through the array covered $\leftarrow b_{i_0}$

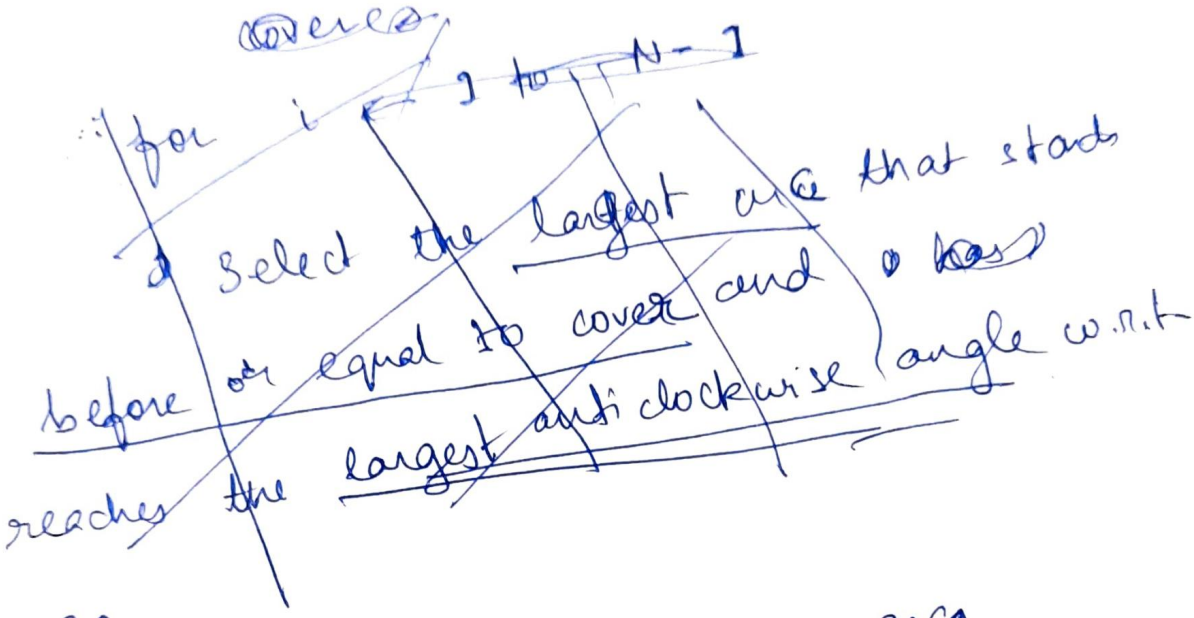
while covered

5) Run a loop from $i \leftarrow 1$ to $N-1$ and find the arc that start before and select

3) cover ← b_{i_0} and index ← 0

cur_cover ← a_{i_0}

4.) while cover < a_{i_0} as it has not covered the full circumference:



update: cover ← end of the arc

S ← S ∪ { arc }

5.) return S.

Time complexity: $O(n \log n)$ we will need to use priority queue or set.

one for loop, no. of iteration $O(n)$

and each loop $O(n)$

for i ← 1 to N-1

if cover < bi

cover = bi

cover = max (cover, bi)

and S V are only if

bi > cover

priority queue or set

used

and S V are only if

bi > cover

priority queue or set used

The current cover needs to updated every time and are later cleary

4.1 (a) 3.5 / 6

✓ **- 2.5 pts** Currently identifying \gamma and sorting the sequences, however not computing Correct "trees" in auxiliary graph to compute global minima.

4.2 (b) 0 / 3

✓ - 3 pts Incorrect / Not attempted