COL351 Major Exam

Viraj Agashe

TOTAL POINTS

22 / 35

QUESTION 1

1 Name / entry number 1 / 1

✓ **+ 1 pts Correct**

QUESTION 2

2 Short questions 6 / 6

+ 0 pts Incorrect / Not attempted

✓ **+ 2 pts (a) Correct**

+ 1 pts (a) Time complexity: partially correct

+ 0 pts Part a incorrect

✓ **+ 2 pts (b) Triangle counting: correct**

+ 1.5 pts (b) Triangle counting: over/under counting

+ 1 pts (b) Triangle counting: incomplete or partially correct

+ 0 pts Part(b) Incorrect

✓ **+ 2 pts (c) Vertex cover: correct**

+ 1 pts (c) Vertex cover: partially correct

+ 0 pts Part (c) incorrect

QUESTION 3

3 DP 3.5 / 5

+ 0 pts Incorrect / Not attempted

+ 3 pts Algorithm for finding the best route in O(n^2)

✓ **+ 2 pts Algorithm: Only calculated the maximum coins collected, didn't find the best route**

✓ **+ 1.5 pts Proof of correctness**

+ 1 pts Proof of correctness partial

+ 0.5 pts Time complexity analysis

+ 1 pts Point adjustment

- 0.5 pts Mistake in algorithm/incomplete

+ 0.5 pts Proof of correctness

+ 2 pts Implementation of algorithm is not given only the idea is given in paragraph

QUESTION 4

4 Max-flow 6 / 6

+ 0 pts Incorrect 4(a)

+ 0 pts Incorrect 4(b)

✓ **+ 3 pts Correct 4(a) Returns $$(S X T) / E$$**

✓ **+ 3 pts Correct 4(b), Checked for path from s to x and y to t in $$G_{f}$$ if (x, y) $$\in$$ (S X T) \ E**

+ 6 pts Correct

- 1 pts Missing claim : $$(S, S^c)$$ and $$(T^c, T)$$ are (s, t) min-cuts

- 1 pts Missing claim : if $$(x, y) \notin S X T$$ then (s, t) max flow is unchanged.

- 1 pts Missing claim: For (x, y) $$\notin (S X T) \ E $$, on the addition of edge (x, y) $$\in (S X T) \ E$$, there is the path from s to t in $$G_{f}$$, which implies the (s, t) max flow increases by 1.

QUESTION 5

5 NP completeness 2 / 7

✓ **+ 0 pts Incorrect / Not attempted**

+ 7 pts Correct

+ 2 Point adjustment

Some insight but not correct

1 vertices need to be join too (all pair)

QUESTION 6

6 Divide and Conquer 3.5 / 10

+ 0 pts Incorrect / Not attempted

+ 5 pts First part: correctly solved

+ 10 pts Second part: correctly solved

+ 2 pts First part: Overlapping matrices not correctly handled.

+ 2.5 pts First part: Reduced matrix problem to array problem, and identified that array problem can be solved in linear time. But algorithm/correctness for array not stated.

+ **2 pts** Second part: some intuition about divide and conquer provided, but solution passing through separators not handled.

✓ **+ 4 pts** **Second part: Some ideas using BFS traversal stated, but the the algorithm to find cycle (containing a vertex v) is incorrect/incomplete and correctness is also incomplete/incorrect.**

+ **7.5 pts** Second part: Some ideas using BFS traversal stated, the claim/algorithm to find cycle (containing a vertex v) also correct, but correctness is missing/incorrect.

- 0.5 Point adjustment

You cannot use DFS to find smallest cycle.

1 Name / entry number **1 / 1**

✓ **+ 1 pts** **Correct**

2 Short questions 6 / 6

+ **0 pts** Incorrect / Not attempted

✓ **+ 2 pts** **(a) Correct**

+ **1 pts** (a) Time complexity: partially correct

+ **0 pts** Part a incorrect

✓ **+ 2 pts** **(b) Triangle counting: correct**

+ **1.5 pts** (b) Triangle counting: over/under counting

+ **1 pts** (b) Triangle counting: incomplete or partially correct

+ **0 pts** Part(b) Incorrect

✓ **+ 2 pts** **(c) Vertex cover: correct**

+ **1 pts** (c) Vertex cover: partially correct

+ **0 pts** Part (c) incorrect

3 DP 3.5 / 5

+ **0 pts** Incorrect / Not attempted
+ **3 pts** Algorithm for finding the best route in O(n^2)
✓ **+ 2 pts** **Algorithm: Only calculated the maximum coins collected, didn't find the best route**
✓ **+ 1.5 pts** **Proof of correctness**
+ **1 pts** Proof of correctness partial
+ **0.5 pts** Time complexity analysis
+ **1 pts** Point adjustment
- **0.5 pts** Mistake in algorithm/incomplete
+ **0.5 pts** Proof of correctness
+ **2 pts** Implementation of algorithm is not given only the idea is given in paragraph

4 Max-flow 6 / 6

+ **0 pts** Incorrect 4(a)

+ **0 pts** Incorrect 4(b)

✓ **+ 3 pts** **Correct 4(a) Returns $$(S X T) / E$$**

✓ **+ 3 pts** **Correct 4(b), Checked for path from s to x and y to t in $$G_{f}$$ if (x, y) $$\in$$ (S X T) \ E**

+ **6 pts** Correct

- **1 pts** Missing claim : $$(S, S^c)$$ and $$(T^c, T)$$ are (s, t) min-cuts

- **1 pts** Missing claim : if $$(x, y) \notin S X T$$ then (s, t) max flow is unchanged.

- **1 pts** Missing claim: For (x, y) $$\notin (S X T) \ E $$, on the addition of edge (x, y) $$\in (S X T) \ E$$, there is the path from s to t in $$G_{f}$$, which implies the (s, t) max flow increases by 1.

5 NP completeness 2 / 7

✓ + 0 pts Incorrect / Not attempted

+ 7 pts Correct

+ 2 Point adjustment

Some insight but not correct

1 vertices need to be join too (all pair)

6 Divide and Conquer 3.5 / 10

+ **0 pts** Incorrect / Not attempted

+ **5 pts** First part: correctly solved

+ **10 pts** Second part: correctly solved

+ **2 pts** First part: Overlapping matrices not correctly handled.

+ **2.5 pts** First part: Reduced matrix problem to array problem, and identified that array problem can be solved in linear time. But algorithm/correctness for array not stated.

+ **2 pts** Second part: some intuition about divide and conquer provided, but solution passing through separators not handled.

✓ **+ 4 pts Second part: Some ideas using BFS traversal stated, but the the algorithm to find cycle (containing a vertex v) is incorrect/incomplete and correctness is also incomplete/incorrect.**

+ **7.5 pts** Second part: Some ideas using BFS traversal stated, the claim/algorithm to find cycle (containing a vertex v) also correct, but correctness is missing/incorrect.

- 0.5 Point adjustment

You cannot use DFS to find smallest cycle.

COL 351 Major Exam

Maximum mark: 35
Timing: 2:00 PM - 4:00 PM

Important Guidelines:

1. Your answer to each question must be formal and have a **correctness proof**.
2. This is a closed book exam. You cannot look at your notes or browse the internet.
3. For each question, you must write your solution only in the space provided below it.
4. You **can not** directly reference a lemma / theorem proved in lecture/tutorial.

1 Write your name and entry number [1 marks]

Name: VIRAJ AGASHE

Entry number: 2020CS10567

2 Short Questions [2 + 2 + 2 = 6 marks]

(a) Let $f(n) = 2^{(\log^2 n)}$ and $g(n) = n^{1.5\sqrt{\log_2(n)}}$. Is $f(n) = O(g(n))$ or $g(n) = O(f(n))$? Justify.

(b) Design an $O(n^{2.4})$ time algorithm to count the number of triangles (cycles of length 3) in an undirected connected graph.

(c) Prove or disprove: The problem of verifying if there exists a vertex cover of size at most 10 in an n-vertex graph is NP-complete.

REDO.

Ans(a) $f(n) = 2^{(\log n)^2} = (2^{\log n})^{\log n}$. Assuming

$= \left(2^{\frac{\log_2 n}{\log_2 e}}\right)^{\frac{\log n}{\log_2 e}}$

$= n^{\frac{\log n}{\log_2 e}}$ & $= n^{\frac{\log_2 n}{(\log_2 e)^2}}$

$g(n) = n^{(\log_2 n)^{3/2}}$

Now, note that $(\log_2 n)^{3/2} > \log_2 n$ $\forall n > 1$

$\therefore$ i.e. $\log_2(n) = o\left((\log_2 n)^{3/2}\right)$. $\therefore \exists c$ s.t.

$\Rightarrow n^{(\log_2 n)^{3/2}} \geq c \cdot n^{\log_2 n / (\log_2 e)^2}$ $\forall n > 1$

$\therefore$ $f(n) = O(g(n))$

(b) For a graph G, we can construct its adjacency matrix A in $O(n^2)$ time. Now, we can simply compute

$$M = A^3$$

Then, the entries $M[i][i]$ will give the no. of paths of length 3 from a vertex i to i (i.e. no. of Δ / cycles of length 3)

$$\therefore \boxed{\text{No. of } \Delta = \sum_{i=1}^{N} M_{ii}}$$

(If we want unique Δ, each Δ will be counted 6 times. So we can divide above by $3! = 6$, i.e.

$$\boxed{\text{No of } \Delta = \frac{\sum_{i=1}^{N} M_{ii}}{3!}}$$

)

Pf:
By induction, say $A^2[i][j]$ gives paths of length 2 no. of then $\sum A^2[i][k] \cdot A[k][j]$ is no. of paths of length 3. Base case: Obviously true for $n=1$

(c) FALSE. We can check all subsets of vertices of size 10.

There are ${}^nC_{10}$ such sets of vertices. We know that we can verify whether a set of vertices is a vertex cover in $O(m)$ time.

$\therefore$ The above algorithm is $O({}^nC_{10} \cdot m)$

$$= \underbrace{O(n^{10} \cdot m)}_{\text{Polynomial}}$$

$\therefore$ Since polynomial time algorithm exists, the problem cannot be NP-complete (else P=NP).

(a) $f(n) = 2^{(\log n)^2}$ $\quad g(n) = n^{1.5\sqrt{\log n}}$

$$= \left(2^{\log_2 n}\right)^{\log_2 n}$$

$$= n^{\log_2 n}$$

Since $\log_2 n > \sqrt{\log_2 n}$ $\forall n > 1$

$\therefore n^{\log_2 n} \geq c\, n^{1.5\sqrt{\log_2 n}}$, for some $c, n > 0$.

$$\Rightarrow \boxed{g(n) = O(f(n))}$$

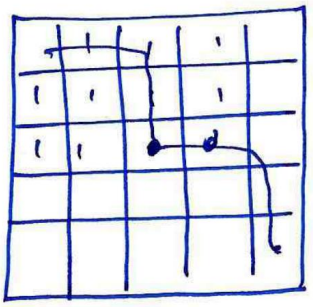
3 Dynamic Programming [5 marks]

Several coins are placed in cells of an $n \times n$ matrix M with no more than one coin per cell.

A robot, located in the upper left cell (i.e. $M[1,1]$), needs to collect as many coins as possible and bring them to the bottom right cell (i.e. $M[n,n]$). On each step, the robot can move either one cell to the right or one cell down from its current location.

Design an $O(n^2)$ time algorithm to find the best route.

Ans. We will solve using DP.



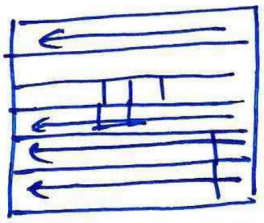
Algorithm:

Let C[i][j] represent the no. of coins we can collect starting from (i,j). Since we can move only right or down, we have the recursive relation:

$$C[i][j] = \max\left(C[i+1][j],\ C[i][j+1]\right) + M[i][j]$$

Then acc. to the above recurrence, we can solve as follows:

1. Initialize an $n \times n$ matrix C with $C[i][j] = 0\ \forall i,j$.
2. Start filling C[i][j] row wise starting from the bottom right (bottom up DP). We have $C[n][n] = M[n][n]$, and we will calculate C[i][j] as per the above equation.
   (In the corners we will only take the squares where we can move, i.e.



Filling the DP Table.

$$C[i][j] = \begin{cases} \max(C_{i+1,j},\ C_{i,j+1}) + M_{i,j} & \text{if } i+1, j+1 < n \\ C_{i+1,j} + M_{i+1,j} & \text{if } j+1 > n \\ C_{i,j+1} + M_{i,j+1} & \text{if } i+1 > n \\ M_{n,n} & \text{if } i,j = n \end{cases}$$

3. Finally, the answer is contained in C[1][1].

Proof: By induction.

Base case: $C[n][n] = M[n][n]$ as required.

I.H. Suppose that C[i][j] contains the correct ans for all $i' > i$ and $j' > j$, all rows $i' > i$ and columns $j' > j$.

Proof of correctness:

Base case: $C[n][n] = M[n][n]$ which is clearly as required.

I.H. Suppose we have a given column j and ~~row~~ i. Then we suppose for all $i' > i$, ~~and~~ j, $C[i'][j']$ stores the correct result and for all $j' > j$ in $i' = i$, $C[i][j']$ stores the correct result (that is how we will the DP table).

I.S. By above recurrence,

~~$C[i][j] = \max(C[i+1][j] + M[i+1][i]$~~

$$C[i][j] = \max(C[i+1][j], C[i][j+1]) + M[i][j]$$

Since we can move only R or D, and $C[i+1][j]$ and $C[i][j+1]$ are correct by I.H,

$\therefore$ $C[i][j]$ is also correct

$\therefore$ Algorithm is correct.

4 Max Flows [3 + 3 = 6 marks]

Let $G = (V, E)$ be a directed flow graph on n vertices and m edges, where each edge has capacity one. Further, let s be the source vertex and t be the destination.

(a) Design an $O(mn)$ time algorithm to compute all pairs $(x, y) \in (V \times V) \setminus E$ such that addition of edge (x, y) to G increases the max-flow in G.

(b) Let f be an (s, t)-max-flow in G, and (x, y) be a pair in $(V \times V) \setminus E$. Design an $O(m + n)$ time algorithm to compute max-flow in graph $G + (x, y)$ using f.

Ans. (a) We note that by max flow - min cut theorem, the max flow in a graph G = capacity of min-cut.

∴ Only edges which increase capacity of min-cut can increase the flow.

First, we apply Ford-Fulkerson's algorithm to compute the max-flow of the graph. In the residual graph G_f we then run DFS from the source s to find the set of vertices S which are reachable from s. This forms the min cut of G.

Note that the T.C. of Ford-Fulkerson is $O((m+n)|F|)$.

Since all capacities ← 1.

The max flow $|F|$ is limited by the degree of the source s which can be atmost $O(n)$. ∴ Finding the residual graph & flow is $O((m+n)\cdot n) = O(mn)$. DFS from source s to find S is $O(m+n)$.

Next, we now note that to increase the capacity of this min-cut, we must connect a vertex $v \in S$ to some vertex $w \in T$ s.t. T is the set of vertices which can reach the sink t. This creates a path from s to t and increases the max flow.

So, we reverse the residual graph G_f and find the set T by running DFS from the sink t. This also takes $O(m+n)$ time.

Now, we simply iterate over all vertices in S and T. If $(u,v) \notin E$, connecting them increases the max flow. This step takes $O(|S||T|) = O(n^2)$ time.

∴ The algorithm is $O(mn + n^2) = \underline{O(mn)}$.

Proof of correctness: The correctness is argued above. In brief, since $f_{max} = C(S, \bar{S})$, the flow can only increase by increasing capacity of cut $(S, \bar{S})$. This is possible only if some vertex u is connected to a vertex from which the sink is reachable. Then, we can find another augmenting path in Ford-Fulkerson & increase the flow, else not. Hence, the algorithm is correct.

(b) We will use the idea of the part (a).

Given an (s,t) max flow, we can simply calculate the sets S and T in $O(m+n)$ time using DFS as done above.

(Reminder: $S \rightarrow$ vertices reachable from S
$T \rightarrow$ vertices which can reach t)

Now, note that on the addition of such an edge e the max flow can increases by 1 (since the capacity of min cut increases by 1). In all other cases the max flow remains the same.

Now, given an edge $e = (x,y)$, we can

(b) We can reuse the old flow to solve the problem in $O(m+n)$ time.

Given the flow, through each of the edges, we can compute the residual graph G_f in $O(m+n)$ time, by introducing reverse edges / removing forward edges on comparing the flows with capacities of each edge.

Now, in the residual graph G_f, create the given edge $e=(x,y)$ of capacity 1.

Now, run DFS starting from the source $\Rightarrow O(m+n)$ time

① → If we do not find an augmenting path from $s \rightarrow t$, the max flow is the old flow f.

② → If $\exists$ an augmenting path from $s \rightarrow t$, then increase the flow using 1 iteration of Ford-Fulkerson algorithm, to get f_{new}.

Since the capacity of min cut can only increase by 1, f_{new} is optimal & the algorithm is correct (we need not run further iterations of Ford-Fulkerson). Note that in ①, max flow cannot increase & this follows from proof of part (a) [since dest not reachable from source, min cut capacity same]

5 NP completeness [7 marks]

Let G be an undirected connected graph, S be a subset of vertices, and k be an integer. Prove that the problem of deciding if there exists a subtree of G of size at most k that contains set S is NP-complete.

Hint: Reduce an instance $H = (V, E)$ of vertex-cover to the above problem by computing a layered graph whose one layer corresponds to V and another layer corresponds to E.

Sol: We first prove that subtree of size k problem is in NP.

Given a subtree T

- First check if $|T| \leq k$. Else return False.
- We can run DFS on T to find the set of vertices contained in T, say V. We also check if T is a tree (~~if any back edge in DFS~~ if visit any visited vertex then FALSE).
- If $S \subseteq V$ and T is a tree, then T is a YES instance, else FALSE.

Since DFS is $O(m+n)$ and checking $S \subseteq V$ is $O(n)$ (set membership)

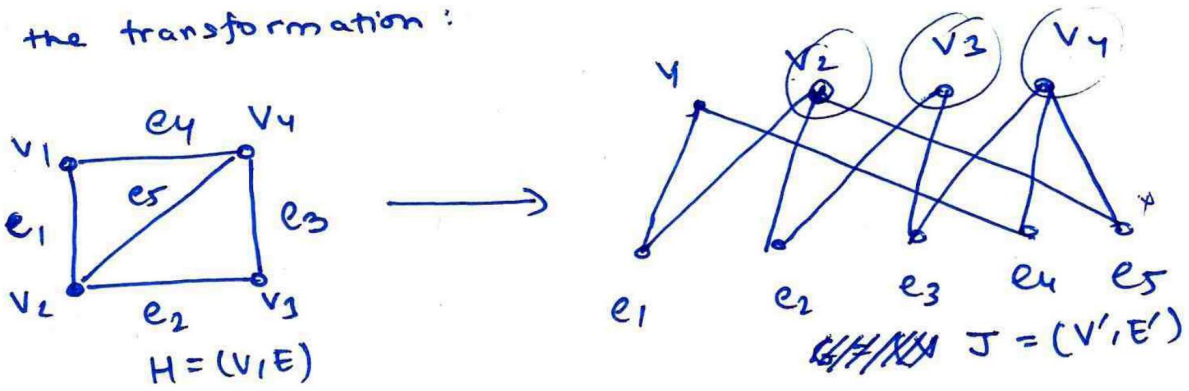
$\therefore$ ∃ polynomial time verifier for the problem.

(For conciseness, let us call the problem K-SUBTREE).

$\therefore$ K-SUBTREE $\in$ NP.

Now, we will reduce vertex cover to K-SUBTREE.

Consider the transformation:



We claim that

YES instance of VC of size k' $\iff$ YES instance of K-SUBTREE with $S = E'$, $k = k' + |E|$

$S = \{e_1, e_2, \ldots e_5\}$

— PTO —

We define a new graph $J = (V', E')$, where

$V' \equiv$ ~~Vertices of H~~ $\{ V_i \mid V_i \in V_H \} \cup \{ e_i \mid e_i \in E_H \}$

$E' =$ Create an edge (V_i, e_j) iff e_j incident on V_i.

Now, the proof:

[$\Rightarrow$] If $\exists$ a VC of size k in H

Consider the subgraph of J consisting of the vertices in the VC and the all the edges $e_i \; \forall i$. Call this subgraph T.

Then, note that $|T| = k + |E|$ ——— (i)

$S = E' \subseteq T$ (since we select all edges of H in T) ——— (ii)

~~Also, T is a tree since if it is not a tree~~

Also, T is a tree since it is connected and has $|T| - 1$ edges. ($\sum deg(v)$)

$\therefore$ J is a YES instance of k-SUBTREE

[$\Leftarrow$] If $\exists$ a k-subtree of size $k + |E|$

- If all edge-vertices e_i are in the subtree T, then we are done (the top layer vertices form a vertex cover in H)

~~Else, notice that for each e_i in the set E, we can~~

- Else, notice that we can replace a vertex in the top layer with a edge vertex e_i in the bottom layer while keeping the size of $|T| = k + |E|$.

$\therefore$ Modify T s.t. all "edges" of bottom layer are in the set.

Since all edges are covered, the vertices in top layer can be considered as vertex cover for H.

$\Rightarrow$ H is a YES-INSTANCE of vertex cover

$\therefore$ k-SUBTREE $\in$ NP and VC $\leq_R$ k SUBTREE. Since VC $\in$ NP complete,

k-SUBTREE $\in$ NP-Complete.

6 Divide and Conquer

This question has two choices. You must attempt **exactly one** part.

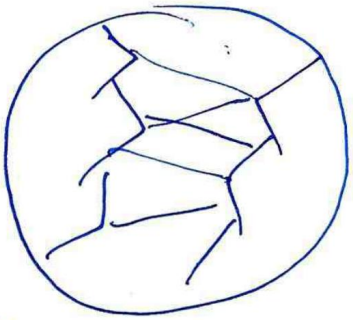
Let M be a 2D matrix of size $n \times n$ with integer entries. Design an $O(n^3 \log n)$ time algorithm find a sub-matrix of M of largest sum. **[5 marks]**

OR

The *Planar Separator theorem* states that every n vertex planar graph G contains a cycle of length at most $O(\sqrt{n})$ whose removal partitions G into two disjoint subgraphs each of which has at most $2n/3$ vertices. Such separators are computable in linear time.

Use Planar Separator theorem to design an $O(n^{1.5} \log n)$ algorithm to find a cycle of smallest length in planar graphs. **[10 marks]**

Ans. We will use the planar separator theorem as a black box.
Note that given a problem of size n ($|V|=n$) we can compute the planar separator in linear time.
Since $|E| = O(n)$, let us ~~say/use~~ compute the separator in $O(n)$ time.



So, given a graph G, first let us compute the cycle C (separator) in $O(n)$ time.

→ Now, suppose we remove this cycle. Then we have 2 subproblems of size $\leq \frac{2n}{3}$. We can recursively solve these in time $T(\frac{2n}{3})$. (DIVIDE STEP)

→ CONQUER STEP
However, now we must check if there is any ~~smaller~~ smaller cycle containing one of the edges in the cycle C. We can perform a DFS from each vertex of this cycle to check if the edge is contained in some other cycle or not and compare its sizes with the other cycles obtained, and finally return the min. $\Rightarrow T = O(n^{3/2})$

$\therefore T(n) = 2T(2n/3) + O(n^{3/2})$
Solve recursively to get $O(n^{3/2} \log n)$