# COL351 Minor Exam

Viraj Agashe

TOTAL POINTS

**26 / 30**

QUESTION 1

## Short Questions 4 pts

1.1 (a) **2 / 2**

+ **0 pts** Incorrect
✓ + **2 pts Correct**

1.2 (b) **2 / 2**

+ **0 pts** Incorrect
✓ + **2 pts Correct**

QUESTION 2

## 2 Matching in Forest 5 / 6

✓ + **0.5 pts Checking if 0 degree vertices exist**
+ **1.5 pts** Correct Polynomial(not Linear) Time Algorithm
+ **1.5 pts** Linear Time Algorithm Partial
✓ + **3 pts Correct Linear Time Algorithm**
+ **1 pts** Proof of Correctness Partial
✓ + **1.5 pts Proof of Correctness**
✓ + **1 pts Proof of Time Complexity**
+ **0.5 pts** Exponential Time Algorithm
+ **0 pts** Incorrect / did not attempt
- **6 pts** Cheating
+ **1 pts** Point Adjustment
- **1 Point adjustment**

1 Should add parent of x to set L only if deg(parent(x)) becomes 1.

QUESTION 3

## 3 Dynamic Programming 8 / 8

✓ + **8 pts Completely Correct**
+ **5 pts** Correct Algorithm
+ **2 pts** Slightly Correct Algorithm
+ **1 pts** Does not return minimum
+ **3 pts** Proof of Correctness
+ **2 pts** Slightly Incomplete/Almost Correct Proof of Correctness
+ **1 pts** Idea of proof correct
+ **0 pts** No/Incorrect proof of correctness
+ **0 pts** Incorrect/Unattempted

QUESTION 4

## Related pairs in DAG 12 pts

4.1 (a) **4 / 4**

✓ + **4 pts Correct**
+ **2 pts** Recursive Relation
+ **1 pts** Justification
+ **1 pts** How to compute N(x,y) for all pairs
+ **0 pts** Incorrect/Not attempted

4.2 (b) **2 / 2**

✓ + **2 pts Correct**
+ **0 pts** Incorrect / Not attempted
+ **1 pts** Part 1
+ **1 pts** Part 2

4.3 (c) **3 / 6**

+ **6 pts** Correct
+ **0 pts** Incorrect
✓ + **3 pts Partial Correct**
+ **1 pts** Partial Correct
+ **4 pts** Without correctness proof

No proof for probability calculation

### 1.1 (a) **2 / 2**

- \+ **0 pts** Incorrect
- ✓ + **2 pts** Correct

1.2 (b) **2 / 2**

+ **0 pts** Incorrect

✓ + **2 pts** Correct

## 2 Matching in Forest 5 / 6

- ✓ + 0.5 pts **Checking if 0 degree vertices exist**
- + 1.5 pts Correct Polynomial(not Linear) Time Algorithm
- + 1.5 pts Linear Time Algorithm Partial
- ✓ + 3 pts **Correct Linear Time Algorithm**
- + 1 pts Proof of Correctness Partial
- ✓ + 1.5 pts **Proof of Correctness**
- ✓ + 1 pts **Proof of Time Complexity**
- + 0.5 pts Exponential Time Algorithm
- + 0 pts Incorrect / did not attempt
- - 6 pts Cheating
- + 1 pts Point Adjustment

**- 1 Point adjustment**

1. Should add parent of x to set L only if deg(parent(x)) becomes 1.

## 3 Dynamic Programming 8 / 8

- ✓ + **8 pts** **Completely Correct**
- \+ **5 pts** Correct Algorithm
- \+ **2 pts** Slightly Correct Algorithm
- \+ **1 pts** Does not return minimum
- \+ **3 pts** Proof of Correctness
- \+ **2 pts** Slightly Incomplete/Almost Correct Proof of Correctness
- \+ **1 pts** Idea of proof correct
- \+ **0 pts** No/Incorrect proof of correctness
- \+ **0 pts** Incorrect/Unattempted

4.1 (a) **4 / 4**

- ✓ **+ 4 pts** **Correct**
- **+ 2 pts** Recursive Relation
- **+ 1 pts** Justification
- **+ 1 pts** How to compute N(x,y) for all pairs
- **+ 0 pts** Incorrect/Not attempted

4.2 (b) **2 / 2**

✓ **+ 2 pts** Correct

**+ 0 pts** Incorrect / Not attempted

**+ 1 pts** Part 1

**+ 1 pts** Part 2

4.3 (c) **3 / 6**

+ **6 pts** Correct
+ **0 pts** Incorrect
+ ✓ **+ 3 pts Partial Correct**
+ **1 pts** Partial Correct
+ **4 pts** Without correctness proof

No proof for probability calculation

COL 351 Minor Exam

Maximum mark: 30

Name: VIRAJ AGASHE

Entry number: 2020CS10567

**Important Guidelines:**

1. For each question, you must write your solution only in the space provided below it.
2. You **cannot** directly reference a lemma / theorem proved in lecture/tutorial.
3. Your answer to each question must be formal and have a **correctness proof**.

# 1 Short Questions [$2 + 2 = 4$ marks]

2

(a) Design an algorithm that verifies whether a given undirected graph (not necessarily connected) with $n$ vertices and $m$ edges is acyclic or not in $O(n)$ time.

Ans. We will perform DFS on the graph $G = (V, E)$ & mark edges as 'visited'. If we visit any visited node again, then the graph has a cycle. Else if DFS concludes without any such visited being visited again, graph is acyclic.

Note that we will in the worst case, expand all edges in a DFS tree which is bounded by $|V| = n$. So algo is $O(n)$.

Pf: If some visited node is visited again, there is a cycle (2 paths from root to that vertex). Otherwise, if no visited again, then it is a forest (no cycles)

2

(b) Let $G = K_n$ be a complete graph on $n$ vertices where for any distinct $i, j \in [1, n]$, $(i, j)$ is an edge in $G$ with weight $|i - j|$. Compute an MST of $G$.

Ans. Consider the tree with edges $e = (i, i+1)$ only, $\forall\, i \in [1, n]$.

① Spanning Tree: (i) Covers all vertices i.e. $|V_T| = n$, $|E| = n-1$

(ii) Acyclic, since for any $(i, j)$, $\exists$ only one path $i - i+1 - \dots\ j-1 - j$


② MST : By exchange argument, no other edges have $|i-j| \leq 1$. ~~except [illegible]~~ and we have $|E| = n-1$.

So weight of ~~MST~~ spanning tree $= (n-1)\cdot 1$

weight of MST $\geq$ (min edge wt) $|E| \geq n-1$.

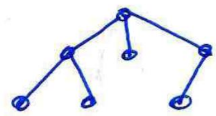
So, T is a MST.

## 2 Matching in Forest [6 marks]

Give a linear-time algorithm that takes as input a forest $F$ on $n$ vertices and determines whether it has a perfect matching: a set of edges that touches each node exactly once.

Ans. We give the algorithm for a tree. We can apply the same algo on each tree of a forest.

TREE ALGO



We can apply the following greedy algorithm:

0. ROOT THE TREE at some r.

~~1/ Consider the leaves L of the graph G.~~

1. Keep a set (list) of vertices to consider, say L. Initially add all leaves of T to L.
2. For every node (vertex) $\in L$: (Say $l$)
   - For $\forall$ edge $(l, x)$ incident on $l$:
     - If $x$ is not marked, mark $x$. Add ~~a neighbour~~ parent of $x$ to the set L. Remove $l$ from the set — $O(\deg V)$ time.
   - If we cannot find unmarked $x$, return FALSE
3. If the set $L = \emptyset$:
   Return TRUE.

TC analysis: Since we start from leaves, we are simply traversing tree in a bottom-up manner.

$$T = O\left(\sum \deg(|V|) + n\right) = O(m+n) \text{ time.}$$

(Linear for each tree)

$\therefore$ For a forest, $T = \sum O(m_i + n_i)$, for each forest $F_i$

$$= O(|E| + |V|) = O(M+N).$$

Proof: We must cover leaves with an edge, so we add the edge with the parent (which we must add to any sol) & mark the parent ($x$). Further, the parent of $x$ must be processed, so we add it to the "end" of the list/set to be processed.

We use the claim inductively on the structure of the tree to prove the algorithm,

Claim: After all leaves initially in L have been processed, parent($x$) is equivalent to a leaf.

→ This follows as edges below it have been covered & are essentially "no longer in the tree", i.e. $G/\{v : \text{covered}(v)\}$.

→ Also, if we cannot find a unmarked node, that means a node must compulsarily be covered (touched) twice, so we return FALSE.

FAR RADECER
RECEDAR

AD CE

## 3 Dynamic Programming [8 marks]

AED C E EC DA
AED C EA

You are given a sequence $X = (x_1, \ldots, x_n)$. Design an $O(n^2)$ time algorithm to find minimum number of characters that needs to be inserted in $X$ so that the resultant sequence is a palindrome.

Example: If $X = (LABEL)$, then the answer is 2, as we need to add $E, A$ at appropriate positions.

Ans. Consider the strings

$$X_1 = x_1 x_2 \ldots x_n$$
$$X_1' = x_n x_{n-1} \ldots x_1$$

L A E B E A L

Note that the "LCS" of $X_1$ & $X_1'$ $\in$ PALINDROME(X), where P(X) (P(X)) is the palindrome constructed by adding min. chars to X.

Further all characters not in the LCS must be inserted in one another at the appropriate positions in order to make the string a palindrome.

ALGORITHM:

1. Compute LCS(X, Rev(X)) ← Length of LCS

{ This can be done as discussed in class using dynamic programming. Make a 2D array of size n×n, fill each L[i][j] as the LCS ending at i in X & j in Rev(X). So the relation used is:

Rev(X) = Y.

$$L[i][j] = \begin{cases} \text{if } X_i = Y_j & \text{then } L[i-1][j-1]+1 \\ \text{else} & L[i-1][j-1] \end{cases}$$

Finally, L[n][n] is the answer (Length of LCS) }

2. The no. of insertions needed is simply

len(X) − LCS(X, rev(X))

Pf: Note that for the LCS of X & Y(= rev(X)), we do not ~~need to insert any characters (as they are same in both~~

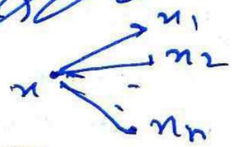
DONE ON LAST PAGE

# 4 Related Pairs in DAG $[(4 + 2 + 6) = 12 \text{ marks}]$

Let $G$ be a DAG on $n$ vertices with vertex-set $\{1, 2, \ldots, n\}$ and topological ordering $(1, 2, \ldots, n)$. For each pair $(x, y)$, let $N(x, y)$ denote the number of distinct paths from $x$ to $y$ in $G$. Let $\cong$ be an equivalence relation on vertex-pairs of $G$, such that $(a, b) \cong (c, d)$ if and only if $N(a, b) = N(c, d)$.

(a) Provide a recursive relation to compute $N(x, y)$ from $N(x_1, y)$, $N(x_2, y)$, …, $N(x_t, y)$, where $x_1, \ldots, x_t$ are out-neighbours of $x$ in $G$.
Justify your answer, and show how it can be used to compute $N(x, y)$, for each pair $(x, y)$.

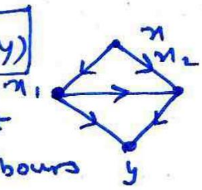
Ans. ~~To immediate neighbours, there can be only one path. (acyclic)~~



We have that,

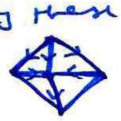
$\cancel{N(x,y) = N(x,x_1)\cdot N(x_1,y) + N(x,x_2)\cdot N(x_2,y)}$

Ans. $\boxed{N(x,y) = N(x_1,y) + N(x_2,y) + \ldots + N(x_t,y)}$



$= 3$

Proof: ~~There are two~~ The paths to reach $x$ from $y$ must pass through one of the out neighbours of $x$. Further, the first edge on each of these paths is diff. so they are all disjoint.



5.

Algo: We can start ~~with the vertex x~~ from $y$ in a bottom-up manner, (i.e., see its immediate in neighbours. For them, $N(y_i, y) = 1$. Then proceed inductively upwards, storing the results using DP. This can be done for each vertex $y \in |V|$.

(b) Argue that for any $(x, y)$, $N(x, y) \leqslant 2^n$. Use this to show that for any two pairs $(a, b)$, $(c, d)$ the number of prime factors of $|N(a, b) - N(c, d)|$ is at most $n$.

Ans. Since $|V| = n$, for each vertex, we can have a "choice", i.e. either it lies on path, or not.

$x$ —— —$y$

There are $n-2$ vertices, so $N(x,y) \le 2^{n-2} \le 2^n$.

Further, we know that

$$N(a,b) \le 2^n, \quad N(c,d) \le 2^n.$$

$$\therefore |\underbrace{N(a,b) - N(c,d)}_{k}| \le 2^n. \text{ And,}$$

Suppose $k = p_1^{\alpha_1} p_2^{\alpha_2} \ldots p_k^{\alpha_k} \ge 2^{\alpha_1 + \ldots \alpha_k}$

$2^{\Sigma \alpha_k} \le k \le 2^n$

$\therefore n \le \Sigma \alpha_k \le n \quad \therefore$ Each $\alpha_k = 1$. $\Rightarrow$ At most $n$ prime factors of $|N(a,b) - N(c,d)|$

(c) Design an $O(|V| \cdot |E|)$ time algorithm that computes with probability $(1 - \frac{1}{n})$ the equivalence classes under relation $\cong$. Also present the correctness of your algorithm.

Ans:- We can compute the ~~distances~~ no. of paths from $(x,y)$ using the algorithm in part (a).

This takes $O(|E|)$ time for each vertex $v \in V$.

$\therefore$ $O(|V||E|)$ time.

Further, we can store the computed $N(x,y)$ $\forall x,y$ in a hash table structure, i.e. we can hash the values using hash fn,

$$H(z) = (az \bmod p)$$

where $p$ is a prime.

Now, probability that equivalence classes have no "collision" $= 1 - 1/n$.

**You can use the following results as black-box:**

**1. Prime Number Theorem:** For any $L \geqslant 2$, number of primes in range $[2, L]$ is $O(\frac{L}{\log L})$.

**2. AKS Primality Testing:** For any $p \geqslant 2$, one can check if $p$ is prime in $O(\log^{O(1)} p)$ time.

Q3

Ans. Consider

$$X_1 = x_1 x_2 \dots x_n$$

$$X_2 = x_n x_{n-1} \dots x_1$$

Note that LCS of both strings will $\in P(X)$, where $P(X)$ is the palindrome constructed by $X$ with min additions. So,

ALGORITHM

1. Compute LCS(X,Y), where $Y = rev(X)$.
   - Create 2D array of size $n \times n$. $\Rightarrow L$, $L[i][j] \rightarrow$ len of LCS ending at i in X & j in Y.
   - $L[i][j] = \begin{cases} L[i-1][j-1], & \text{if } X[i] \neq Y[j] \\ L[i-1][j-1]+1 & \text{otherwise.} \end{cases} \Bigg\} O(n^2)$

2. No. of insertions = $Len(X) - LCS(X, rev(X))$ } $O(1)$.

$TC = O(n^2)$

Proof: Note that for string X, all $c \in LCS(X, rev(X))$ can be added to $P(X)$ without any insertions needed.

For all characters $c \notin LCS(X, rev(X))$

Let such chars be

$y_1 \ y_2 \ \dots \ y_k$ ~~k must be even~~

~~To ensure that~~

So we have,

$$\left.\begin{matrix} y_1 \ y_2 \ \dots \ y_k \\ y_k \ \dots \ y_1 \end{matrix}\right\} \text{ in } X \ \& \ rev(X).$$

To make the strings same, we must insert $y_k$ before $y_1$ in X & $y_1$ after $y_k$ in rev(X). Proceeding this way, we see that we can make both strings same. So no. of insertions is $len(X) - LCS(X, rev(X))$. Further we can show at least $len(X) - LCS(X, rev(X))$ insertions are needed, to make the strings same, as these are no. of diff chars in X & rev(X) (in order), so we need atleast these many changes. So algorithm is optimal & correct.