Winter 2024
May 4, 2024

ADA Endsem (Sec-A) Solution and Rubrics

Max: 80 Marks
180 minutes

Roll No: _____ Name: _____

Question	1	2	3	4	5	6	7	8	9	Total
Marks										

Instructions: *Please write solutions independent of each other. This is a closed book test. So, you can not use books or lecture notes. Please note that your solution must fit in the space provided. The extra sheet will be provided* only *for roughwork. So try to be precise and brief. Meaningless blabber fetches negative credit.*

All The Best!

---

1. **(8 Marks)** For each of the statements, just write whether the statements are True or False.

   (a) Let A be an array of n numbers such that every element in A are from the set $\{1, 2, \ldots, 3n\}$. Then, there does not exist any algorithm without using hashmap/hashtable that can find if there are two distinct indices i and j such that $A[i] = A[j]$ in $O(n)$-time.

   Answer: False.

   (b) Let L_1 be a problem that is in NP and L_2 be a problem such that there is a valid polynomial-time reduction that transforms an instance of L_1 into an instance of L_2. If L_2 can be solved in polynomial-time, then any problem that is in NP can be solved in polynomial-time.

   Answer: False.

   (c) Given a directed acyclic graph with positive edge weights and two vertices s and t. If a longest path (in terms of total edge weights) P from s to t passes through u, then the subpath of P that starts from u and ends at t is also a longest path betwen u and t.

   Answer: True.

   (d) Let G be an undirected graph with n vertices and m edges. Then, there is an algorithm that can check whether there exists a cycle in G in $O(n)$-time and the running time is independent of m.

   Answer: True

2. **(7 Marks)** Given a graph G, a vertex subset $S \subseteq V(G)$ is called *independent set* if for every $u, v \in S$, $(u, v) \notin E(G)$. Consider the problem where the input is a path graph $G = (v_1, \ldots, v_n)$ with each vertex given positive weights. The edges are (v_i, v_{i+1}) for every $1 \leq i \leq n-1$. The objective is to find the maximum possible weight of an independent set in G.

Here is a partial algorithm using divide and conquer approach.

Function MAXIND(G, n).

(i) If $n \leq 5$, then use brute-force approach (looking at all possible subsets) and return the total weight of a maximum weight independent set.

(ii) $r \leftarrow \lfloor n/2 \rfloor$.

(ii) $G_A \leftarrow$ the path with the first $r-1$ vertices, i.e. $(v_1, \ldots, v_{r-1})$.

(iii) $G_B \leftarrow$ the path $(v_{r+2}, \ldots, v_n)$.

(iv) $G_C \leftarrow$ the path $(v_{r+1}, \ldots, v_n)$.

(v) $G_D \leftarrow$ the path $(v_1, \ldots, v_r)$.

(vi) $I_A \leftarrow$ MAXIND$(G_A, r-1)$.

(vii) $I_B \leftarrow$ MAXIND$(G_B, n-r-1)$.

(viii) $I_C \leftarrow$ MAXIND$(G_C, n-r)$.

(ix) (fill up here)

$I_D \leftarrow$ MAXIND(G_D, r).

$$M \leftarrow \max \begin{cases} I_A + I_B \\ I_A + I_C \\ I_D + I_B \end{cases}$$

Return M.

(**3 Marks**. Deduct 1 mark for every incorrect instruction.)

(a) What is the above algorithm's recurrence and time complexity? Give a proper explanation of your answer.

Since this algorithm is recursive and solves 4 subproblems each of size $n/2$, and then combines these solutions in $O(1)$-time, the recurrence relation is $T(n) = \begin{cases} 4T(n/2) + O(1) & \text{if } n \geq 5 \\ c & \text{otherwise} \end{cases}$

By using Master's Theorem, this recurrence solves to $O(n^2)$.

(**2 Marks** for correctly explaining the recurrence and **2 Marks** for correct use of master's theorem or any other method, e.g. substitution method. Mentioning of Master's theorem of any specific recurrence solving method is important)

3. **(10 Marks)** Given an undirected graph $G = (V, E)$ with n vertices, m edges, and two specified vertices s and t. A shortest path between s and t is a path with minimum number of edges. Design an $O(n + m)$-time algorithm that computes the number of shortest paths between s and t. Explain the running time of your algorithm. Your algorithm does not have to output all the shortest paths between s and t, just the number of shortest paths between s and t suffices.

(if your algorithm description goes past this page, then also it is okay, but please make sure to properly mention where you write the time complexity of your algorithm)

- **A brief description of your algorithm:**

  The idea is a combination of BFS and Dynamic Programming.

  Subproblem: For every vertex $u \in V(G)$, define $Path(u)$ the number of shortest paths from s to u in G.

  Recurrence: The recurrence of the dynamic programming is as follows.

  Base Case: $\text{PATH}(s) = 0$.

  For every $u \in V(G) \setminus \{s\}$, $\text{PATH}(u) = \sum_{v \in \text{Adj}(u): Level(v)+1=Level(u)} \text{PATH}(v)$.

  Subproblem that solves the actual problem: $\text{PATH}(t)$.

  Algorithm Description: The main steps of the algorithm are as follows.

  (i) Invoke BFS on G starting from s.

  (ii) Let $s, v_1, v_2, \ldots, v_k, t, v_{k+1}, \ldots, v_{n-2}$ be the order of vertices obtained by the BFS traversal of the graph.

  (iii) Let $\text{Level}(u)$ denotes the distance of u from s as obtained from BFS traversal.

  (iv) Initialize $\text{NumPath}[s] = 0$.

  (v) For every $v_1, \ldots, v_k$ (in this order as appears in BFS traversal) set

  $NumPath[v_i] = \sum_{v_j \in Adj[v_i]: \text{Level}(v_i)=\text{Level}(v_j)+1} \text{NumPath}[v_j]$.

  (vi) $\text{NumPath}[t] = \sum_{v_j \in \text{Adj}[t]: \text{Level}(t)=\text{Level}(v_j)+1} \text{NumPath}[v_j]$.

  (vii) Return $\text{NumPath}[t]$.

  (If used dynamic programming, then **3 marks** for stating the recurrences, subproblem definitions, and subproblem that solves the actual problem and **5 Marks** for algorithm description)

  (If not used recurrence, but directly used the algorithm, then he should explain why his algorithm should work. **8 Marks** for the algorithm description and explanation.)

- **Explanation of running time of your algorithm.**

  Observe that BFS traversal takes $O(n + m)$-time for graphs with n vertices and m edges. The subsequent steps need to scan the adjacency list for every vertex constant number times. This procedure also takes $O(n + m)$-time.

  Hence, the total time taken is $O(n + m)$.

  (2 Marks)

4. **(15 Marks)** Let G be a directed acyclic graph (DAG) whose vertices have some labels from some fixed alphabet $\Sigma = \{a, b\}$, and let $S[1, \ldots, r]$ be a string of length r over Σ. The number of vertices of G is n and the number of edges is m. Design a poly$(n + m + r)$-time algorithm to find the length of a longest path whose label is a subsequence of S. Explain the time complexity of your algorithm as well.

(**Comment:** You can assume that you are given an array Label$[1, \ldots, n]$ for every vertex. If your algorithm description goes past this page, then also it is okay, but please make sure to properly mention where you write the time complexity of your algorithm)

- **A brief description of your algorithm.**

  Preprocessing: Perform topological sort of G. Let $u_1, u_2, \ldots, u_n$ be the topological order and Label$(u_i) \in \Sigma$ for every $i \in [n]$. **(1 Mark)**

  Let G_k denotes the subgraph induced by the vertex set $G[\{u_1, \ldots, u_k\}]$ of the input DAG and S_k denotes the substring $S[1, \ldots, k]$.

  Subproblem Definition: SEQ(i, j) denotes the length of a longest path in G_i that is a subsequence of S_j. **(2 Marks)**

  Subproblem Recurrence:

  If $i = 0$ or $j = 0$, then SEQ$(i, j) = 0$ - this is for the base case. (**2 Marks**. There are two base cases. Deduct 1 marks for one incorrect base case.)

  If $i \geq 1$ or $j \geq 1$, then we have

  $$\text{SEQ}(i,j) = \begin{cases} 1 + \max\limits_{k<i:(u_k,u_i)\in E(G)} \{\text{SEQ}(k, j-1)\} \text{ if Label}(u_i) = S[j] \\ \max\{\text{SEQ}(i-1, j), \text{SEQ}(i, j-1)\} \text{ otherwise} \end{cases}$$

  The first part of the recurrence holds when the label of u_i matches with the j-th character of the string S.

  (**3 Marks** for the recurrence other than the base case.)

  Subproblem that solves the final problem: SEQ(n, r) **(1 Marks)**

  Algorithm Description: both text description and pseudocode are accepted.

  - Construct the reverse graph G_R where the graph is the same but all arcs are reversed.
  - Initialize $X[0, 0] = 0$.
  - For every $i = 1, \ldots, n$, initialize $X[i, 0] = 0$.
  - For every $j = 1, \ldots, r$, initialize $X[0, j] = 0$.
  - For $i = 1$ to n and for $j = 1$ to r
    If Label$[u_i] = S[j]$, then
    - find temp $= \max\limits_{k<i:u_k \in Adj_{G_R}(u_i)} X[k, j-1]$.
    - $X[i, j] = temp + 1$.

    Otherwise Label$[u_i] \neq S[j]$. Then
    - $temp_1 = X[i-1, j]$ and $temp_2 = X[i, j-1]$.
    - If $temp_1 > temp_2$, then set $X[i, j] = temp_1$.
    - Otherwise, set $X[i, j] = temp_2$.
  - Return $X[m, r]$.

  (4 Marks)

- **Explanation of the running time of your algorithm.**
  Observe that the construction of G_R takes $O(m+n)$-time.
  The for loop has $n \cdot r$ values computation and computation of every value needs to scan the adjacency list of every vertex. Hence, the total running time is $O(mnr)$.
  (**3 Marks**)
  2

5. **(8 Marks)** A graph H is said to be a clique if every pair of vertices of H are adjacent to each other. The CLUSTER VERTEX DELETION problem is defined as follows.

   - **Input:** An undirected graph $G = (V, E)$ and an integer k.
   - **Question:** Is there $S \subseteq V(G)$ such that $|S| \leq k$ and every connected component of $G - S$ is a clique?

   Prove that CLUSTER VERTEX DELETION is in NP.

   - **Certificate or Proposed Solution of the Problem**
     A vertex subset $X \subseteq V(G)$ of the graph. (**3 Marks**).
     Deduct one mark if written a vertex subset of size at most k.
     0 Marks if written that a vertex cover or something else that is not a vertex subset of the graph.
   - **A text descrition of the algorithm.** (**6 marks** total out of which **4 Marks** for algorithm description)
     – Given a proposed solution $X \subseteq V(G)$, the first step is to delete every vertex of X from G. This can be done by creating a new adjacency matrix where the rows and columns represent the vertices of G that are not in X. Precisely, this new adjacency matrix is the adjacency matrix of the original graph G restricted to the rows and columns corresponding to the vertices of G that are not in X.
     – Compute the connected components of $G - X$ using BFS.
     – For every connected component C, if every pair of vertices of C are adjacent to each other, then return "X is a cluster vertex deletion set (or a valid certificate) of G". Otherwise, return "X is not a valid certificate of G".
     The running time of this algorithm involves constructing $G - X$ that takes $O(n^2)$-time. After that, it involves executing BFS with the adjacency matrix representation that also takes $O(n^2)$-time. Finally, it involves checking every pair of vertices of every connected component of $G - X$ that is also $O(n^2)$-time.
     Hence, total running time is $O(n^2)$.

6. **(7 Marks)** Two problems 3-COLORABILITY and 5-COLORABILITY are defined as follows.

3-COLORABILITY

- **Input:** An undirected graph $G = (V, E)$.
- Is there a coloring function $\lambda : V(G) \to \{1, 2, 3\}$ such that for every edge $(u, v) \in E(G)$, $\lambda(u) \neq \lambda(v)$?

5-COLORABILITY

- **Input:** An undirected graph $G = (V, E)$.
- Is there a coloring function $\lambda : V(G) \to \{1, 2, 3, 4, 5\}$ such that for every edge $(u, v) \in E(G)$, $\lambda(u) \neq \lambda(v)$?

Consider the following algorithm to suggest a **polynomial-time reduction** from 3-COLORABILITY to 5-COLORABILITY.

Reduction: The main steps are as follows.

(i) Let G be an instance of 3-COLORABILITY and $V(G) = \{u_1, \ldots, u_n\}$.

(ii) For every $i \in \{1, 2, \ldots, n\}$, create a vertex v_i.

(iii) For every pair of vertices (u_i, u_j), if $(u_i, u_j) \in E(G)$, then make $(v_i, v_j) \in E(G)$.

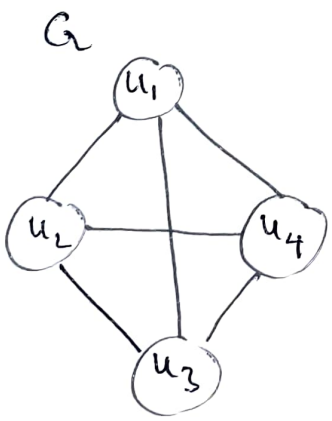
(iv) For every $i \in \{1, \ldots, n-1\}$, add edges (u_i, v_i) and (u_i, v_{i+1}).

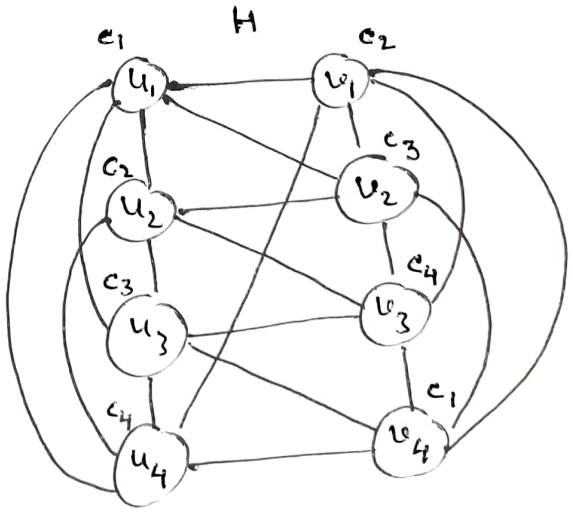
(v) Then, add edges (u_n, v_n) and (u_n, v_1) (*a typo corrected from the question. Clarified in the exam hall also.*)

(vi) The output graph is H.

Either prove that this algorithm is a polynomial-time reduction from 3-COLORABILITY to 5-COLORABILITY, or give a counter example to disprove that this is not a valid reduction.

(if you give a counter example, please clearly explain in the example why this reduction is not a valid reduction in that example)





Example: 5 marks

~~No~~

(G is a No-instance if and only if H is a yes-instance.) → 2 marks

0 → if incorrect counter-example or a proof which is not formal

7. **(8 Marks)** If there is a set S of n distinct elements, then given an integer k, design a polynomial-time algorithm that counts the number of possible tuples (A, x) such that $A \subseteq S$, $x \in A$, and $|A| = k$.
(If your algorithm uses subroutines that computes $r!$ for any r, then your solution will be awarded zero marks)

- **A brief description of your algorithm**:
There are two main steps in this algorithm. The first step is to design an algorithm that counts the number of subsets of size k of a set of n elements. Then, for every set of k elements, there are k many choices, hence k has to be multiplied with that.
The idea is to build a DP table as follows.
Count-Subset(n, k)
(i) If $k = 0$ or $n \leq k$, then return 1.
(ii) For $i = 1, \ldots, n$
- - - Initialize $TAB[i, 0] \leftarrow 1$.
- - - Initialize $TAB[i, i] \leftarrow 1$.
(iii) For $i = 1, \ldots, n$ and for $j = 1, \ldots, k$ in this order.
- - - $TAB[i, j] = TAB[i-1, j] + TAB[i-1, j-1]$.
(iv) Return $TAB[n, k]$.
The idea above was to exploit the recurrence $\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$.
Main-Algo(n, k)
(i) $x \leftarrow$ *Count-Subset*(n, k).
(ii) Return $x * k$.
The idea is that the final answer is $k\binom{n}{k}$.
Rubric: 3 marks for the Count-Subset algorithm and 3 marks for the Main-Algo. Deduct marks if there is/are major mistakes in either of the algorithm.
Similar credit to be awarded if the Main-Algo first executes $x \leftarrow$ Count-Subset$(n-1, k-1)$ and then then the final instruction is $x * n$.
The idea here is that $k\binom{n}{k} = n\binom{n-1}{k-1}$.
0 Marks if used algorithms computing $k! \cdot \frac{n!}{k!(n-k)!}$.

- **Explanation of the running time of your algorithm**.
Observe that building the table and computing all the values take $O(nk)$-time. Finally, the running time is $O(nk)$.
(2 Marks)

8. **(10 Marks)** The CS department of a university has a flexible curriculum with a complicated set of graduation requirements. The department offers n courses and there are m requirements. A student has to satisfy all of the m requirements to graudate. Each requirement specifies a subset A of courses and the number of courses that must be taken from the subset. The subsets for different requirements may overlap but each course can be used to satisfy at most one requirement.
Design a poly$(n+m)$-time algorithm that determines whether a student can graduate.
Example: There are 5 courses x_1, x_2, x_3, x_4, x_5 and $m = 2$. The requirements are **(i)** at least two courses must be taken from $\{x_1, x_2, x_3\}$, and **(ii)** at least two courses must be taken from $\{x_3, x_4, x_5\}$. A student can graudate with courses $\{x_1, x_2, x_3, x_4\}$ but cannot graduate with courses $\{x_2, x_3, x_4\}$.

- **Formulation as Flow Network (4 Marks)**
  For every course, create a vertex. For every requirement create a vertex. Create two additional vertices s and t.
  In summary, the vertices $\{x_1, \ldots, x_n\}$ represent the courses and the vertices $\{y_1, \ldots, y_m\}$ represent the requirements.
  Suppose that the j-th requirement contains a subset of courses D_j says that at least r_j courses must be taken from D_j.
  For every $i \in \{1, \ldots, n\}$, create a directed edge $s \to x_i$ with capacity 1.
  For every $j \in \{1, \ldots, m\}$, create a directed edge $y_j \to t$ such that the capacity of the edge $y_j \to t$ is r_j.
  Finally, for every $i \in \{1, \ldots, n\}$, if the course corresponding to the vertex x_i appears in D_j, then put a directed edge $x_i \to y_j$ with capacity 1.
- **Designing the algorithm with this flow network and justification.**
  (i) Construct the flow network as described above.
  (ii) Compute a maximum s-t-flow using Ford Fulkerson's Algorithm.
  (iii) If the value of the maximum flow is $\sum_{j=1}^{m} r_j$, then output "yes, a student can graduate". Otherwise, output that a student cannot graduate.
  Justification: Note that in order to graduate, there must be a set of courses that must be taken so that each of those chosen courses satisfies exactly one requirement, and for every requirement D_j, exactly r_j courses are taken.
  The capacity of $x_i \to y_j$ being 1 is justified since one chosen course can satisfy exactly one requirement, no more than that.
  (**4 Marks** total out of which 3 marks for the algorithm description and 1 mark for the justification.)

- **Running time of the algorithm.**
  The ford-fulkerson's algorithm runs in time $O((|V|+|E|)|f^*|)$ such that $|f^*|$ is the value of maximum flow. Observe in this graph that the value of maximum flow is $\sum_{j=1}^{m} r_j$ and for every $j = 1, \ldots, m$, $r_j \leq n$. Therefore, value of the maximum flow is $\sum_{j=1}^{m} r_j \leq mn$. The number of vertices is $n+m+2$ and the number of edges is $O(mn)$. Hence, it holds that the running time of the Ford-Fulkerson's algorithm is $O((mn+n+m)mn)$ which is $O(m^2n^2)$. Finally, the construction of the graph takes $O(mn)$-time.
  Hence, the running time of the algorithm is $O(m^2n^2)$.

9. **(7 Marks)** Consider the following flow network. The numbers denote the capacities of the edges. Illustrate the execution of Ford-Fulkerson's Algorithm to compute a maximum flow from s to t in this network. Your illustration must show step by step execution of the Ford-Fulkerson's Algorithm, particularly the changes in the residual graph, and the flow value in every edge at each step.

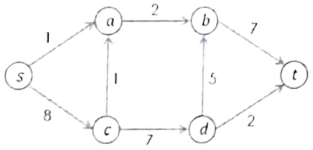
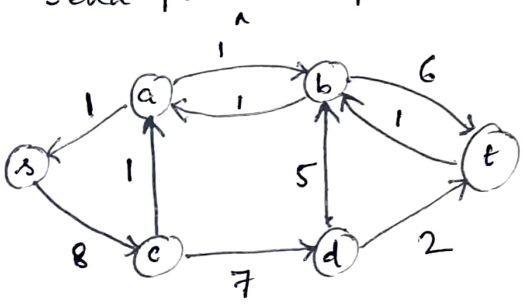


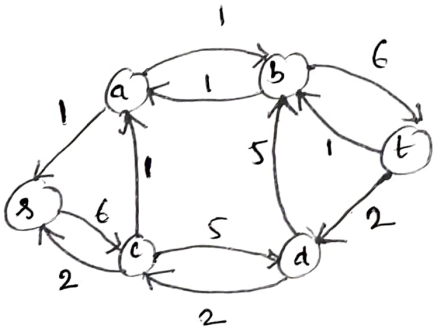
Figure 1: Flow network

(1 unit)
Send flow via path $s \to a \to b \to t$



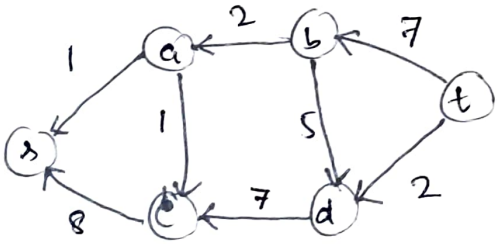
Send 2 units flow via $s \to c \to d \to t$

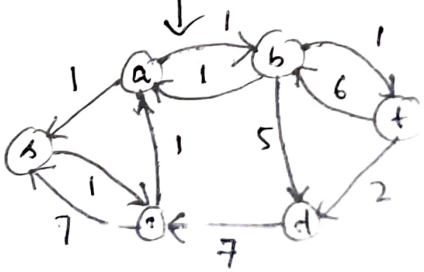
All edges fully saturated.
Flow value = capacity for all edge at the end



Send 5 units flow via $s \to c \to d \to b \to t$

Send 1 unit through $s \to c \to a \to b \to t$





Value of flow 9

5 marks for showing stage by stage residual graph.

2 marks for mentioning flow value (max flow) and final value of flow for every edge