ADSA 2025-26

Major

Name: **Roll No:**

Duration: 180 Minutes **Max. Marks: 48**

Instructions:

1. *All questions are of* 6 *marks, but they are not of equal difficulty level. So read all the question.*
2. *You can write algorithms in words as well. But you should not miss necessary details.*
3. *Be precise in writing. Write only what is being asked. DO NOT WRITE irrelevant things.*

1. Suppose we are using disjoint-set union data structure via tree implementation with rank heuristic and path-compression. Show final trees/tree (only final not intermediate) after running the following program where we are performing some **Union** operations after several **Make-Set** operations:

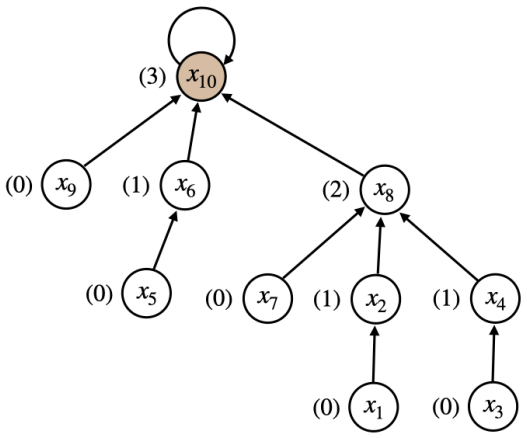
for $i = 1$ **to** 10

  Make-Set(x_i)

Union(x_1, x_2), **Union**(x_3, x_4), **Union**(x_5, x_6), **Union**(x_7, x_8), **Union**(x_9, x_{10}), **Union**(x_5, x_9), **Union**(x_3, x_7), **Union**(x_1, x_8), **Union**(x_2, x_{10})

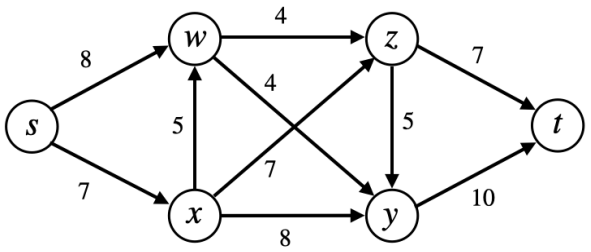
(Note: In **Union**(x, y)*, if the rank of the representative of x is the same as the rank of the representative of y, then the representative of y will become the parent of the representative of x.)*

Solution:



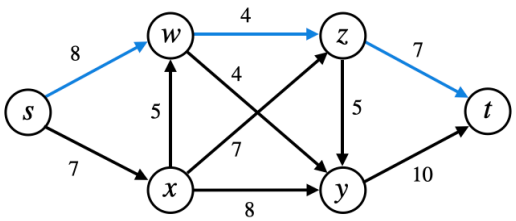
Marking Scheme: Full marks only when tree is entirely correct. Partial marks will be given based on how far from correct is your answer.

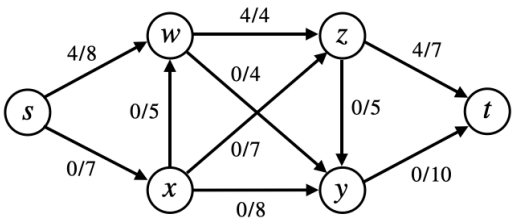
2. Run Ford-fulkerson algorithm on the below flow network in detail, i.e., find the residual network and indicate the augmenting path you are taking in every iteration. Calculate the max flow.



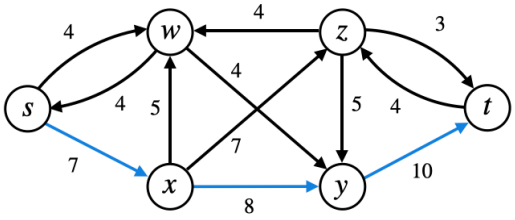
Solution:

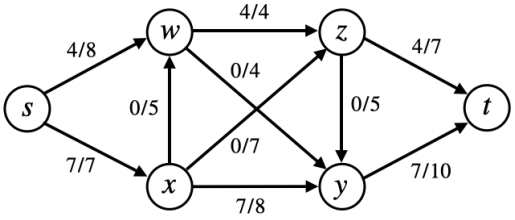
Iteration 1 (Residual Network and Updated Flow):



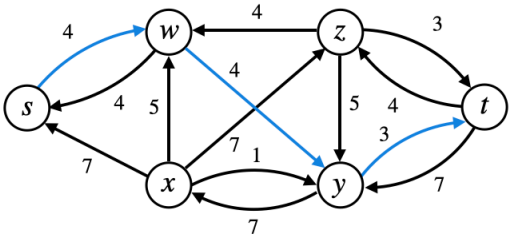


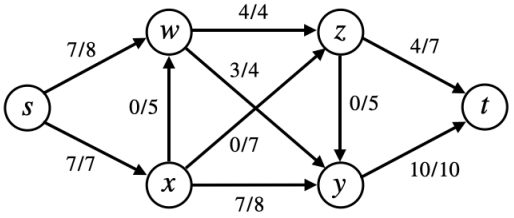
Iteration 2 (Residual Network and Updated Flow):



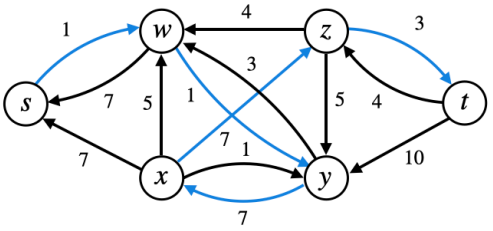


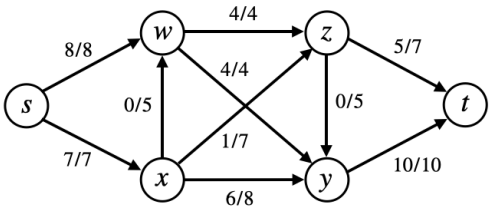
Iteration 3 (Residual Network and Updated Flow):



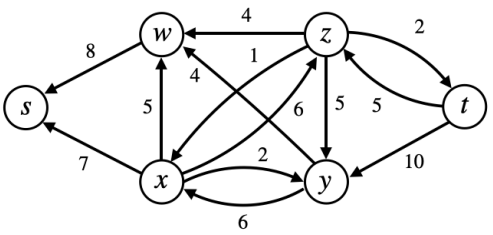


Iteration 4 (Residual Network and Updated Flow):





Iteration 5 (Residual Network and Updated Flow):



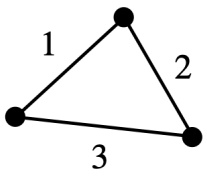
No augmenting path present now.

Hence, max flow is 15.

Marking Scheme: 5 marks for iterations. 1 mark for max flow. Partial marks will be provided based on how many iterations are right.

3. If the following claim is true prove it, else give a counterexample. Let G be an undirected, weighted, and connected graph such that all the edges in the graph have distinct edge weights. Let e be the edge with the least weight in G. If $G - e$ is connected, then MST of $G - e$ will always be a second smallest minimum spanning tree.

Solution: The claim is wrong. Following is a counterexample:



In the above graph the second smallest MST is of weight 4. But if we remove the edge with weight 1, the MST of the remaining graph has weight 5.

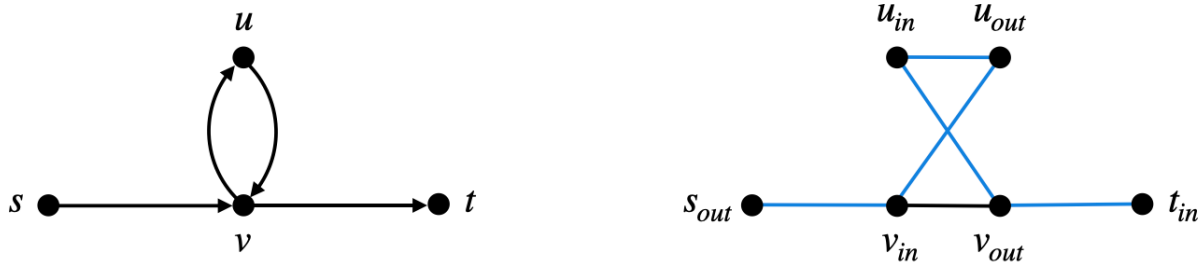
Marking Scheme: 0 marks for saying the claim is right. 4 marks for the counterexample, 2 marks for the explanation.

4. Recall the reduction from *DirHamPath* to *HamPath*, where we added three vertices u_{in}, u_{mid}, and u_{out} in the reduced graph for every vertex apart from s and t in the original graph. We also discussed the importance of u_{mid} in the reduction. In this problem, you need to give and explain a counterexample for the following flawed reduction from *DirHamPath* to *HamPath*, where we have not used u_{mid}:

Reduction from $\langle G, s, t\rangle$ to $\langle G', s', t'\rangle$:

- Vertices of G':
  - For s add s_{out}, for t add t_{in}.
  - For any other vertex u in G add 2 vertices u_{in}, u_{out}.
- Edges of G':
  - Ignore incoming edges to s and outgoing edges from t.
  - For every edge (u, v) in G add an edge $\{u_{out}, v_{in}\}$.
  - Edges of the type $\{u_{in}, u_{out}\}$.
- $s' = s_{out}$, $t' = t_{in}$.

Solution: In the below reduction, there is a hamiltonian path from s_{out} to t_{in} but there is no hamiltonian path from s to t.



Marking Scheme: 4 marks for the counterexample, 2 marks for the explanation.

5. Suppose you are given a set of positive integers $A = \{a_1, a_2, \ldots, a_n\}$ and a positive integer k. A subset $S \subseteq A$ is called **feasible** if the sum of the numbers in S does not exceed k, that is,

$$\sum_{a_i \in S} a_i \leq k$$

You would like to select a feasible subset S of A whose total sum is as large as possible. Design an $O(n \log n)$ time approximation algorithm that returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set S ⊆ A. Casually justify the correctness of your algorithm.

Solution: The below is an $O(n \log n)$ time approximation algorithm that returns a feasible set $S \subseteq A$ whose total sum is at least half as large as the maximum total sum of any feasible set S ⊆ A.

FeasibleSum$(A[1\ldots n], k)$**:**

1. Sort A in non-increasing order.

2. $S = \emptyset, curr_{sum} = 0$
3. **for** $i = 1$ **to** n
4. **if** $curr_{sum} + A[i] \leq k$
5. $S = S \cup A[i]$
6. $curr_{sum} = curr_{sum} + A[i]$
7. **return** S

In the above algorithm we are first sorting the numbers in A in a non-increasing order. After that we start adding elements to S from A as long as sum of the elements added to S is less than or equal to k. If the first element we picked is $\geq k/2$ then we are done. If it is $\leq k/2$, then the last element we add to S must also be $\leq k/2$. Now, either we will add all the elements after picking the first element or we will stop at some point because sum of the elements is becoming more than k. If we add all the elements, then we are finding optimal feasible set as we added the first element which was $\leq k$ and then the remaining elements as well. If we stop at some point because sum of the elements is becoming more than k, then also we are fine because if adding a number which is $\leq k/2$ is making sum more than k, then sum must be $\geq k/2$.

Marking Scheme: 3 **marks for the algorithm,** 3 **marks for justification.**

6. The shortest common supersequence (SCS) of X and Y is the smallest sequence Z such that both X and Y are a subsequence of Z. Let $SCS(i, j) =$ length of the SCS of $X[1 \ldots i]$ (that is, the first i characters of X) and $Y[1 \ldots j]$ (that is, the first j characters of Y). Define the recurrence for $SCS(i, j)$ that can be used to design an $O(|X| \cdot |Y|)$ time dynamic programming algorithm to find the length of SCS of X and Y. Casually justify the correctness of your recurrence.
Solution: The value of $SCS(i, j)$ will depend on the following cases:
• $S(i, j) = i$, if $j = 0$.
• $S(i, j) = j$, if $i = 0$.
• $S(i, j) = 1 + S(i - 1, j - 1)$, if $X[i] = Y[j]$.
• $S(i, j) = min(S(i - 1, j), S(i, j - 1)) + 1$, if $X[i] \neq Y[j]$.

If $j = 0$, then the shortest common supersequence should contain all the i characters of X. Hence, when $j = 0$, $S(i, j) = i$. Same reasoning applies for $S(i, j) = j$, when $i = 0$. When the last characters, that is, $X[i]$ and $Y[j]$, are equal we can have a common character for them in SCS. Hence, $SCS(i, j) = 1 + SCS(i - 1, j - 1)$. Finally, when the last characters are not the same, we have a choice to make either $X[i]$ or $Y[j]$ the last character of SCS of $X[1 \ldots i]$ and $Y[1 \ldots j]$ and pick the one which gives the lower value. Hence, $S(i, j) = min(S(i - 1, j), S(i, j - 1)) + 1$, if $X[i] \neq Y[j]$.

Marking Scheme: 4 marks for the recurrence, 2 marks for the casual justification.

7. Recall that while learning the Bellman–Ford algorithm, I mentioned that a polynomial-time algorithm for computing shortest paths in directed weighted graphs containing negative-weight cycles is unlikely to exist and in future we will see the reason for it. Unfortunately, I forgot to discuss that reason in the class. I will now let you discover it on your own. Prove that if there exists a polynomial-time algorithm that correctly computes shortest paths in general weighted graphs (even in the presence of negative-weight cycles), then it would imply that $\mathsf{P} = \mathsf{NP}$.

Solution: We will reduce the *DirHamPath* problem to the *ShortestPath* problem. Consider an instance of *DirHamPath*, say $\langle G, s, t \rangle$. Let's give -1 weight to every edge of G and call this weighted graph G'. Clearly, G' may contain a negative weight cycle now. Now, it is easy to see that G contains an st-hamiltonian path if and only if weight of a shortest path from s' (copy of s in G') to t' (copy of t in G') in G' is $-(n-1)$, where n is the number of vertices in G' (or G). I am leaving the proof to you. Now, if we have a polynomial time algorithm to solve *ShortestPath*, we can also solve *DirHamPath* in polynomial time. Since *DirHamPath* is **NP-complete**, this proves that $\mathsf{P} = \mathsf{NP}$.

Marking Scheme: 3 marks for the reduction and 3 marks for the proof.

8. You are given a tree $T = (V, E)$ and weight function $w : V \to \mathbb{Z}^+$ (every node has a positive integer as its weight). Design an $O(|V|)$ time algorithm that finds the weight of an independent set S in T that maximises $\sum_{v \in S} w(v)$. Prove the correctness of your algorithm.

Solution: You can get the solution here.

Marking Scheme: 4 marks for the algorithm and 2 marks for the correctness proof.