

ADSA 2025-26

Minor

(Answers and Marking Scheme)

Name:

Duration: 120 Minutes

Roll No:

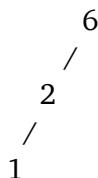
Max. Marks: 36

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 the answers in words as well. But make sure you are not missing necessary details.

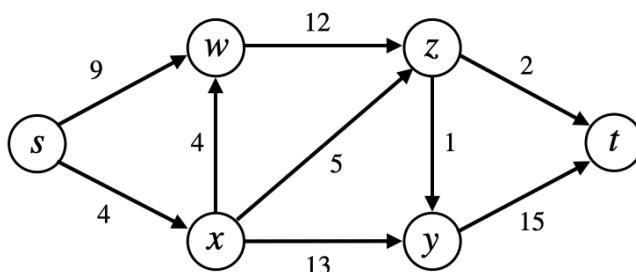
1. Let T be an empty BST. Show the final BST you will get after performing the following operations on T in the given order such that the tree remains BST at every point of time: **Insert**($T, 3$), **Insert**($T, 5$), **Insert**($T, 4$), **Insert**($T, 2$), **Insert**($T, 6$), **Delete**($T, 5$), **Delete**($T, 3$), **Insert**($T, 1$), **Delete**($T, 4$).

Solution: The final BST will be:



Marking Scheme: Full marks only when the final tree is shown as above. Partial marks will be provided if you have given the stepwise construction and the BSTs are constructed correctly till a few steps.

2. Give the order in which vertices will be removed from priority queue when you run Dijkstra's algorithm on the below graph with s as the source vertex. (Do not write anything else apart from the order.)



Solution: The right order is: s, x, w, z, y, t .

Marking Scheme: k marks, if the first k entries of the order are correct.

3. Suppose we have a flow network where anti-parallel edges are allowed, that is, it is possible to have both (u, v) and (v, u) edges. Suggest a modification in such a flow network so that we can use the max flow computing method for flow networks where anti-parallel edges are not allowed to compute the max flow for those flow networks where anti-parallel edges are allowed. Casually justify why the proposed modification works.

Solution: Suppose the flow network $G = (V, E)$ has some anti-parallel edges. We construct a corresponding network $G' = (V', E')$ which contains the same set of vertices, edges and capacities as G . Then, for every pair of anti-parallel edges (u, v) and (v, u) with capacities c_1 and c_2 , respectively, we break the edge (v, u) into two edges (v, w) and (w, u) with both having c_2 as the capacity. Now we can compute max flow f^* in G' . Due to flow conservations edges (v, w) and (w, u) must have the same flow, say f . We can easily get a corresponding flow in G , where every edge which was not broken in G' will get the same flow as in G' and any edge (v, u) which was broken into two edges (v, w) and (w, u) in G' will get the flow of (v, w) and (w, u) . This corresponding flow will also be a maximum flow in G .

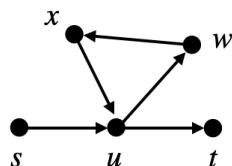
Marking Scheme: 4 marks for the modification. 2 marks for the casual justification.

4. Suppose we have a directed graph $G = (V, E)$ with a designated source vertex s and destination vertex t . We want to find out if there is a path from s to t which has exactly k edges. Recall that a path cannot have repeated vertices.

For any vertex v and for any number i , let us define a Boolean variable $\text{path}(v, i)$, which is supposed to be **True** if and only if there is a path from v to t with exactly i edges. As the base case we take $\text{path}(t, 0)$ to be **True**. We write the following procedure to compute $\text{path}(v, i)$ starting from $i = 1$ to k , for every vertex $v \in V$:

For each vertex u such that there is an edge from v to u , we check $\text{path}(u, i - 1)$. If any of them is **True** then we set $\text{path}(v, i)$ as **True**. Using this logic, we compute $\text{path}(v, i)$ for all pairs (v, i) . We finally output $\text{path}(s, k)$. Is the described algorithm correct? If yes, justify its correctness. If not, explain why?

Solution: The algorithm is incorrect. Consider the following graph as a counter-example.



If we apply the above algorithm to find whether there is a path from s to t which has exactly 5 edges, then the algorithm will answer **True**, while there is no path from s to t containing exactly 5 edges. To see why algorithm will answer **True**, notice that in the 1st iteration $\text{path}(u, 1)$ will become true because of the edge (u, t) , in the 2nd iteration $\text{path}(x, 2)$ will become true because of the edge (x, u) , in the 3rd iteration $\text{path}(w, 3)$ will become true because of the edge (w, x) , in the 4th iteration $\text{path}(u, 4)$ will become true because of the edge (u, w) , in the 5th iteration $\text{path}(s, 5)$ will become true because of the edge (s, u) . Therefore, algorithm will finally output **True**.

Marking Scheme: 1 mark for only saying algorithm is incorrect. 2 for the right counter-example and 3 for the explanation. 0 marks for saying the algorithm is correct.

5. Let $G = (V, E)$ be a flow network with source s , sink t , and integer capacities. Suppose that we are given a maximum flow in G . Suppose that we increase the capacity of a single edge $(u, v) \in E$ by 1. Give an $O(|V| + |E|)$ time algorithm to update the maximum flow. Give a formal justification of correctness and runtime of your algorithm.

Solution: After increasing the capacity of the edge $(u, v) \in E$ by 1, to compute the max flow in the updated network we first check whether there exists an augmenting path in the residual network of updated network. If there does not exist an augmenting path, then the current flow is still a max flow and nothing needs to be done. If there exists an augmenting path, then we run one more iteration of Ford-Fulkerson. Notice that max-flow cannot increase by more than 1 because capacity of the minimum cut can also increase by only 1. Hence, running only one iteration of Ford-Fulkerson is enough as the flow will increase by at least 1 in a flow network with integer capacities.

Constructing the residual network, checking for an augmenting path in the residual network, and running one iteration of Ford-Fulkerson clearly requires only $O(|V| + |E|)$ time.

Marking Scheme: 2.5 marks for saying that we need to run one iteration of Ford-Fulkerson, 2.5 marks for the justification why only one iteration would suffice. 1 mark for explaining the runtime.

6. Show how to maintain a dynamic set Q of numbers that supports the operation **Min-Gap**, which gives the absolute value of the difference of the two closest numbers in Q . For example, if we have $Q = \{1, 5, 9, 15, 18, 22\}$, then **Min-Gap**(Q) returns 3, since 15 and 18 are the two closest numbers in Q . Implement the operations **Insert**, **Delete**, **Min-Gap** such that the cost of **Insert** and **Delete** is $O(\log n)$ and of **Min-Gap** is $O(1)$. Formally prove the correctness and runtime of your implementation. (In case you are using tree to maintain such a set, you are allowed to store constantly many extra values or pointers per node.)

Solution Sketch: We can perform these operations in required time by storing the integers in nodes of an augmented RB tree as *keys*. We augment the RB tree so that (1) every node contains two extra pointers, one to its predecessor and other to its successor, and (2) every node stores the absolute value of the difference of the two nodes in the subtree rooted at that node (for a node x , this value is referred to as $x.val$). Maintaining these information during insertion and deletion is doable in $O(\log n)$ time. We explain here how to maintain them during insertion and leave deletion to the reader. After inserting a new node, say x , in the same way we do in a normal BST, we find the predecessor and successor of x and change the right set of pointers of the x , its predecessor, and its successor as they are the only nodes whose successor and predecessor will get affected after the insertion. We also set the $x.val = \infty$. Observe that val of only those nodes can get affected which are ancestors or x . To compute new val of ancestors of x , we need to travel from parent of x to root. In this process, for any node y , we already know $y.left.val$ and $y.right.val$. Now, $y.val = \min(y.left.val, y.right.val, y.key - y.predecessor.key, y.successor.key - y.key)$. Hence, $y.val$ can be computed in constant time. So far everything can be done in $O(\log n)$ time.

After this during fix-up of RB tree properties we do not need to do anything for successor and predecessor pointers as fix-ups require only recolouring and rotations, neither of which affects

successor or predecessor of any node. While val will get affected during rotations, we can maintain them similar to how we maintained $size$ in augmented trees seen in the class.

Now to answer **Min-Gap** is $O(1)$, we only need to return $T.root.key$.

Marking Scheme: Partial marks will be given based on close your answer is to the right solution.