Name :- MRINAL PAUL
Reg :- RA2311033010054

HOTS DAA

(1) Given a weighted directed graph containing both positive and negative edge weights, but without any negative cycles, is Dykstra's algorithm appropriate for finding the single - source shortest path.

No, Dijkstra's algorithm is not suitable for graphs with negative edge weights, even if there are no negative cycles. This is because Dijkstra's algnorithm assumes that once the shortest path to a node ofound, it will not be improved later. However, a negative edge might lead to a shorter path after the node has already been processed, violating this assumption.

Correct Approach:
Use the Bellman-Ford algorithm instead. It works correctly with graphs containing negative weights (as long as there are no negative cycles) and runs in $O(V \times E)$ time.

(2) Given two sorted ~~ways~~ arrays, each combining n elements, devise an algorithm to efficiently compute the median of their combined 2n elements with a time complexity of $O(\log n)$.

2> Given two sorted arrays, each combining n elements, devise an algorithm to efficiently compute the median of their combined 2n elements with a time complexity of O(log n).

To compute the median of two sorted arrays of size n in O(log n) time, we can stay that we can use the Binary Search algorithm. The median of the combined 2n elements will be the average of the nth and (n+1)th smallest element.

Algorithm:—
We can find the K-th smallest element in the combined array using a recursion function. To find the median, we need to fixed the nth element and also the (n+1)th smallest element. Let's define a function findKth(arr1, arr 2, K) that finds the Kth smallest element in the combined sorted array arr1 and arr 2

<1> Best case:
- If arr1 is empty, return K-th element of arr2.
- If arr2 is empty, return K-th element of arr1
- If K=1, return the minimum of the first elements of arr1 and arr2

<2> Recursive Step

- Find the middle indices in both arrays relative to the current search space.

  Let, mid 1 = min(n/2 , size(arr1)) and

  mid 2 = min (n/2 , size (arr 2))

- Compare the elements arr1 [mid1-1] and arr2 (mid2-1]

- If arr1 [mid 1-1] < arr2 [mid 2-1] : This element in arr1 are upto mid-1 are smaller them arr2 [mid 2-1]. We recursively search for the (K-mid1)-th smallest element in the remaining part of arr1 (from index mid1) and the whole arr 2.

- If arr1 [mid 1-1] = = arr 2 [mid 2-1] : We have found the (mid1 + mid2) th smallest element if K = mid1 + mid 2, this is an element

<3> Given a weighted undirected graph, design an efficient algorithm to determine the existence of a second-best Minimum Spanning Tree and Calculate its total weight.

First, find a minimum Spanning Tree (MST) using a standard Algorithm like Kruskal's or Prim's

A second-best MST can be found by considering each edge (u,v) from the original graph that is not in the initial MST. Adding such an edge (u,v) to the MST creates a unique cycle. To find a potential second best MST, someone the edge with the max weight from this cycle. The second best MST is one among all three resulting spanning tree that has minimum total weight. The algorithm iterates through all edges not in the initial MST, calculates the weight of the resulting tree after adding the edge and removing the max weight edge on the cycle and keeps track of the minimum weight found.

⟨4⟩ You are given that Subset sum $\leq_p$ Problem X. What can you conclude about problem X. Provide a brief justification.

The notation subset sum $\leq_p$ Problem X means that there is a polynomial-time reduction from the subset problem to Problem X. Subset sum is a well known NP-complete problem, which implies it is also NP-hard. A polynomial time reduction means that an instance of student sum can be transformed into instance of Problem X in polynomial time such that solving the instance of Problem X guess the solution to the subset sum instance.

Because subset sum as NP hard. solving
Problem x in polynomial time (via reduction).
We can conclude that Problem x is NP-hard

<5> Consider a problem where: <a> Solution can be verified in $O(n^2)$ time, <b> No Known polynomial time algorithm exists to solve it. Which class does this problem belong to? Explain the reasoning.

This problem belongs to the complexity class
NP. The property that a proposed solution
can be verified in $O(n^2)$ time means the
verification process takes a polynomial amount
of time with repeat to the input size
n ($O(n^2)$) is a polynomial) This is the
defining characteristic of problems in the
class NP - solutions can be easily and
efficiently verified. The fact that there is no
polynomial time algorithm Known currently
to solve the problem suggest that it might
not be in class P (problems solvable in
polynomial time) and is consistent with
problems that are NP hard on NP-complete
However solely based on the criteria
provided, the verification property directly places
it within the class NP.