
Question 1:
Given two strings s and t, determine whether some anagram of t is a sub string of s

Solution: The anagram test helper function takes 2 input strings makes a letter-count dictionary for each. If the dictionaries are equal to each other, they are anagrams. The question1 function iterates through sub strings of s with length len(t) and performs the anagram test until an anagram is found, in which case the function returns true. If the function reaches the end of s without finding an anagram, it returns false.Because the algorithm must loop through all the characters of s, the time complexity is O(len(s)). Because we are only storing s and t, the space complexity is O(1)

Question 2:
Given a string a, find the longest palindromic sub string contained in a.

Solution: The question2 function iterates every sub string of a, checks if each sub string is a palindrome. Palindromic sub strings are stored in an lps variable, which is updated every time a longer palindrome is found. The algorithm generates every sub string of a with a nested loop which loops through the whole string for each character. Therefore, the time complexity is O(n^2). Because we only store the longest visited palindromic sub string as we go, the storage complexity is O(1).

Question 3:
Find the minimum spanning tree of a weighted undirected graph

Solution: The question3 function is an implementation of Kruskal's algorithm. This sorts the edges by weight, and from lowest to highest weight adds edges to the mst if the edges do not create a cycle. Cycles are detected using the disjoint set/union find algorithm. This algorithm stores each node's value in a set, the sets being stored in a disjoint super set. The algorithm then iterates through edges and checks if the nodes are in the same set. If not, the sets containing the two nodes are combined into a single set with a union operation. If the two nodes are in the same set, then a cycle is detected.

Complexity analysis:
Step 1: Generating sets of edges and takes O(E) time and O(E) space.
Step 2: Sorting the edges takes O(Elog(E)) time and O(E) space
Step 3: Looping through the edges and adding them to the mst_edges list takes O(E*V) time in the worst case, and O(V) space.
Step 4: Converting the mst_edges list into a properly formatted adjacency list dictionary takes O(E) time and O(E) space.

Overall the function takes O(E*V) time and O(E) space in the worst case scenarios.

Question 4:
Find the least common ancestor between two nodes on a binary search tree.

Solution: This solution represents the bst as a series of nodes with values, and left/right nodes which are initialized to None. A bst is then created by stringing together node objects and assigning left and right values. Starting at the root, the question4 function checks each node to see if the value is higher or lower than both input nodes. If value is higher than both, the function proceeds to the left node, if lower the right. If the node's value is between the values of node1 and node2, then that node is the lca of node1 and node2

Because this algorithm only needs to visit one node per level of the tree, the run time complexity is proportional to the height of the tree, i.e, O(log(n)). Because we have to store the tree, the space complexity is O(n).

Question 5:
Find the element in a singly linked list that's m elements from the end.

Solution: The findLength helper function iterates through the linked list, increment a counter, until the current node has no "next" value, returning the counter. The question5 function iterates through the linked list, increment a counter, until the counter value reaches lengthList - m - 1, returning the current node.

Because the function iterates through the linked list, the time complexity is O(n). Because we only have to store the current node and the counter value at any given time, the space complexity is O(1).