

**ALGORITHMS AND COMPLEXITY HIT220**

**Assignment 3.3**

**Submitted by:**

**GROUP 46**

|  |  |
| --- | --- |
| **Student Name:** | **Student ID:** |
| Shahil Jha | s368427 |
| Walter Kruger | s344566 |
| Danny Nguyen | s344857 |
| Showkat Ara Khan | s343677 |

# Assumptions

1. Node number 1 is the destination node where all water flows towards:
   1. Reason for assumption: Node 1 is placed in the Timor Sea which is the largest notable near-by body of water.
   2. Rivers in-land are typically at a higher elevation than sea-level and flow toward a lower elevation.
2. The river can be represented with a directed tree; all nodes are connected by exactly one path.
   1. This assumption holds true for our map.
3. The range x and y coordinates can only be between the value of 0 to 650, inclusively.
   1. This assumption holds true in accordance to the provided map.
4. Assuming that the junctions seepage occurs only within the area formed by translation of 50 coordinate units area of the junction node, meaning the seeping can occur due to another headwater node within the diagonal area of :

Diagonal distance = sqrt(502 + 502) = sqrt(5000) units sqared

# Question 1

Performs a depth first search, starting at every vertex in the graph. During this traversal, we know that there is a cycle if a back-edge is found (a connection to a vertex already in the path).

Using naïve approach, it would have a as every possible combination of paths would be searched through. However, if we exclude vertices that have already been searched, as if it is part of a cycle we would have already found it, we get a .

# Question 2

The problem given is a form of the Traveling Salesman Problem (TPS). Since the drone can fly directly to any vertex in the graph, the “graph” is assumed to be complete.

First, an initial reasonable tour of the graph is found using the **Cheapest Insertion algorithm**. This is a greedy algorithm as the “cheapest insertion” is what is most locally optimal. It tends to produce more optimal tours than other greedy methods for graphs with a single cluster of vertices, like in our graph. Although Christofides–Serdyukov’s algorithm tends to produce better tours, it is more complex to implement and has a worse big-O; vs

Secondly, the reasonable tour is improved using **2-opt switching**. This is a local search in which every two edges are switched then kept if they shorten the tour. This will perform switches per cycle and will keep repeating these cycling until no more improvements are made. Thus, it has a however the number of cycles made is typically very small, in our graph’s search only two occur.

# Question 3

1. Check if the input *sequence* is all within the traversal from a headwater node to the destination, node 1 through iteration.
   1. Sort the given sequence in accordance with the *traversal path* and check the *concentration* of the chemical is decreasing as it travels downstream.
   2. If the first item of the sorted sequence is directly connected to the *headwater*, then add as possible contamination source.
   3. If the sequence is in subsequent sequence and connected, *concentration* is reducing directly as water flows downstream and if the first item of the sorted sequence is directly connected to the headwater source, then add as possible contamination source.
2. If all the input sequence is not within a single traversal path.
   1. Split the input sequence into nodes that are within the traversal path, i.e. *common list* and that not are not common, i.e. *difference list*.
   2. For *common list*,
      1. Skip loop if there is no element or only one element in *common list.*
      2. Check if in sequence. If not in sequence, check if any of the nodes is directly connected to a *headwater,* is true add to *possible source pool*.
      3. Recursively call the function on itself and add result list to *possible source pool*.
   3. For *difference list*,
      1. Skip loop if there is no element or only one element in *difference list.*
      2. If there is only one element in the *difference list,*
         1. Check if directly connected to *headwater*, if true then add to *possible source pool*.
         2. If not directly connected to any *headwater,* it might be a condition of **seeping.**
            1. Get all *headwater* node in the region.
            2. Calculate the shortest distance between the *junction* node and the possible *headwater* node.
            3. Add the *headwater* node with the shortest distance into *possible source pool.*
      3. If there is more than one element in the *difference list,* then recursively call the function on itself and add all the return list into *possible source pool.*
3. If there is any element in the *possible source pool,* 
   1. Iterate over every possible node from the *possible source pool,*
      1. Iterate over every observed node from the *input sequence,* calculate path distance between the *possible node* and *observed node* and save to a temporary variable.
   2. Get **sum of square** using the calculated distance.
   3. Assign the node with the least value of **sum of square** as the *possible headwater source.*
4. Return the *possible headwaters* list.

# Question 4

The Trie data structure is built to store the river and creek names. Multiple functions are used to encode the characters into their binary representation using their corresponding ASCII value. The words are encoded by traversing the Trie and are printed in a binary string.

The time and space complexity of creating the Trie is O(N \* M), with N being the number of words and M being the average length of the words.

The time and space complexity for encoding a word is O(M), where M is the length of the word.