

**Work Integrated Learning Programmes Division M.Tech (Data Science and Engineering)**

**(S2-18\_DSECFZG519)  
(Data Structures and Algorithms Design) Academic Year 2018-2019**

**Assignment 2 – PS3 - [Shipment Problem]**

**AS2\_BLR\_B2\_G03**

1. **Group Information**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 2018ab04635 | Sathish Kumar M | 2018ab04635@wilp.bits-pilani.ac.in | AS1\_BLR\_B2\_G03 | PS3 |
| 2018ab04662 | Aditya | 2018ab04662@wilp.bits-pilani.ac.in | AS1\_BLR\_B2\_G03 | PS3 |
| 2018ab04714 | Samarth | 2018ab04714@wilp.bits-pilani.ac.in | AS1\_BLR\_B2\_G03 | PS3 |

1. **Problem Statement**

Warehouse in Bangalore was expecting a freight shipment at 10 am on a Monday. However, the company responsible for the shipment mistook the destination as Amazon Development Centre, Bangalore. In order to save the extra cost of transportation back to the warehouse, the warehouse manager wants to help them find the fastest route from the DC to warehouse. He has the location map available which is given below (the weightage given is the distance in km between each location). However, as he did not take DSAD course he turns to you for help.

1. **Requirements**:

* Formulate an efficient algorithm to perform the above task.
* Provide a description about the design strategy used.
* Analyse the time complexity of the algorithm and show that it is an “efficient” one.
* Implement the above problem statement using Python

1. **Assumptions**

* The input files entries are strictly of the given format “node1 / node2 / distance”.
* All distances in the above entries are non negative numbers
* All file location used (to print the output and read the inputs) are relative locations to the current path and are hard coded as these were clearly stated in the problem.
* The start time from the development centre is assumed to be 10am as specified in the problem.
* Below are some of the values hard coded according to problem statement
  + ﻿ ﻿inputFileName = "inputPS3.txt"
  + outputFileName = 'outputPS3.txt'
  + start\_time = 10

1. **Tools and Versions**
   * Python version: 3.7
   * Spyder Interface
2. **Choice of Algorithm**

The following were the algorithms considered before choosing the final option

* **Dijkstra algorithm**
* Bellman-Ford algorithm
* Floyd Warshall Algorithm

Floyd Warshall Vs Dijkstras

Since the source node is known we need not go for Flyod Warshall Algorithm as that is optimum only when our problem is to find the distance between all pair of nodes. Also there is no negative edges/cycles possible as our assumption is distance cannot be negative. So this rules out Floyd Warshall Algorithm.

Bellman-Ford Vs Dijkstras

When comparing Dijkstras with Bellma-Ford again we take into consider the two points. One is there is no need to detect negative cycle as this is not possible. So again the complexity of Bellman\_Ford is more so we need not sacrifice on complexity for finding negative cycles which is not possible in our problem

Modification of Dijktras

So now we have narrowed down to Dijkstras algorithm. But even Dijkstras is used to find the shortest path from a particular source to all other node. But our problem here is to find the distance between a particular source and destination.

So we have modified the logic slightly to stop the iteration once the destination node is moved to the discovered list which denotes that there cannot be a distance shorter than that to the destination from the source. This will prevent and save unnecessary iterations in finding the shortest to the other nodes.

1. **Design**

* **def initializeHash (self):** 
  + This function creates an empty hash table
  + **Complexity = O(1)**
* **def insertHash (driverhash, lic):**
* Compute Hash function to determin the index
* Move to the particular index and iterate through the list whether the element is already present.
* Exit if the end of the list is reached or if the license entry is found. In the first case append the entry to the list and in the second case increase the no of violation case by 1
* Worst case all the entries compute the same hash and gets appened in the same index and so for inserting one entry into the table the **complexity is O(n)**
* **def printViolators (driverhash):**
  + Iterate through the entire hasindex from 1 to the size of the hask
  + For each of the hash index iterate through the list of entries
  + If an entry has number of violations greater than the number specified then make an entry in the output file
  + In all cases we have to read through all the entries and in worst case license number may be unique for all entries in the input file and hence we end up reading n entries. **So complexity is O(n)**
* **def destroyHash (driverhash):** 
  + This function destroys all the entries inside the hash table. This is a clean-up code.
  + Complexity is O(size of hast table)
* **def insertByPoliceId (policeRoot, policeId, amount):**.
* Standard binary tree insertion has been used where we traverse to the left if the element is smallerelse move to right if the element is larger
* Recursive implementation used
* Recursive relation
  + T(n) = T(n-1) + c for all n>0
  + T(n) = c for n=0
* So the order of **complexity for inseting one entry is O(n)**
* **def reorderByFineAmount (policeRoot)**:
  + **here we have to delete nonde by node from one tree and insert it to another node**
  + here inorder to reduce the complexity we move to the lower most leaf and start removing one by one from the leaf. So the complexity is O(n) for removing
  + The we perform standard BST insertion for all the nodes. The complexity for one insertion is O(n) and so for n insertions **complexity is O(n2)**
* **def printBonusPolicemen (policeRoot):** 
  + For this we have to do traverse all the elements in the tree.
  + We can use the BST property and again start from the right most leaf(right child, root, left child) and print into file if the threshold limit is satisfied
  + We can stop the traversal once the fine threshold is not satisfied for a node as all the remaining nodes will definitely not satisfy the criteria
  + In worst case all the entries may satisfy the bonus criteria and so the worst case **complexity is still O(n)**
* **def destroyPoliceTree (policeRoot):** 
  + This just deletes all the nodes and removes the link to the corresponding parent node if a parent node exist
* **def printPoliceTree (policeRoot):** 
  + This will do a recursive inorder traversal of the tree and print all the elements
  + So the **complexity is O(n)**

1. **Complexity**

* For **calculating the Violators**
  + Initializing hash + inserting all entries from the file to hast table + calculating the violators and printing to a file
  + O(1) + n\*O(n) + O(n)
  + So **the total complexity is O(n2)**
* For calcu**lating the Bonus Policeman**
  + Intialising the tree + Inserting all the entries in the file to a tree sorted by PoliceId + Removing the elements from the tree and inserting into another tree sorted by fine amount + Printing all police man who cross the threshold fine amount
  + O(c) + nO(n) + O(n2) + O(n)
  + So the **total complexity is O(n2)**

1. **Testing**

The following scenarios was covered during our testing

* Testing with smaller file to test the basic functionality
* Generated an input file with 1Lakh entries and ensured the successful run
* Negative cases
  + Input file not present
  + Input file present but no entries
  + Input file present with invalid entries
    1. Negative value for fine amount/PoliceId/License Number
    2. Non numbers in the place of fine amount/PoliceId and License number

1. **Scope for improvement**

* As of now the hash table size was kept fixed. We can come up with a solution based on the use case the hash table size can be decided based on the input size automatically