Data Structures and Algorithms

Smart City

Course Project Report

School of Computer Science and Engineering
2023-24

Contents

1. Course and Team Details

1.1 Course details

Course Name	Data Structures and Algorithms
Course Code	23ECAC203
Semester	III
Division	F
Year	2023-24
Instructor	Prof. KMM Rajashekariah

1.2 Team Details

Si. No.	**Roll No.**	**Name**
1.	139	Saakshi V
2.	140	Nidhi M
3.	141	Dhanya R
4	165	Chinmay J S

1.3 Report Owner

Roll No.	**Name**
165	Chinmay J S

2. Introduction

A smart municipal corporation is crucial for the success of a smart city. It serves as the backbone, managing and implementing intelligent systems that enhance city living. From efficient waste management and traffic control to digital governance and data analytics, a smart municipal corporation plays a pivotal role in integrating technology for the benefit of residents.

In essence, a smart municipal corporation is instrumental in transforming a city into a connected, responsive, and efficient urban landscape, aligning with the goals of a smart city.

3. Problem Statement

3.1 Domain

Our goal is to imagine a smart city whose all system work harmoniously with out defacing the cause of working of another function. Automation is an essential part of smart city, without which a smart city wouldn't even exit.

Every smart city requires an smart municipal corporation, which can imagine scenarios using real time data. These new ideas of implementation will help the city by allowing for test scenarios to test its working.

3.2 Module Description

My code implements function that fall under the domain the minicamp corporation of a city. It imagines the duties of the corporation and intends to mimic there function by using data structures, It implements functions like BFS, Dijkstra, AVL Trees, Hashing, and Prims algorithm to be competent to help the corporation to the fullest.

4. Functionality Selection

Si. No.	Functionality Name	Known	Unknown	Principles applicable	Algorithms	Data Structures	Efficiency
	Name the functionality within the module	What information do you already know about the module? What kind of data you already have? How much of process information is known?	What are the pain points? What information needs to be explored and understood? What are challenges?	What are the supporting principles and design techniques?	List all the algorithms you will use	What are the supporting data structures?	O()
1	View_Matrix ()	2D Matrix, size of the matrix	-	Brute Force	-	2D Arrays	O(n^2)
2	BuildRoad(), BuildPipe(), BuildDrain()	Respective Adjacency Matrix and size of the matrix	Builds infrastructure if it doesn't exist.	-	-	2D Arrays	O(1)
3	Displayareas()	Respective array of structure and its the attributes	-	-	-	Array of structure	O(n)
4	addNewArea ()	Reads data of the new node	-	Brute Force	-	Array of Structure and 2D array	O(n)
5	deletelastArea()	Deletes the last node	-	-	-	Array of Structure and 2D array	O(1)
6	CheckDeficiencies()	Checks the facilities of an area	-	-	-	Array of structure	O(log n)
	bfs()	Implements Breadth first Traversal	-	-	-	Queue, 2D Array	O(V + E) V : Vertices E: Edges
	EdgeunderConstruction( )	The edge is disabled	If the edge exists it is disabled	-	-	2D Array	O(1)
	AllEdgeConstructed()	Constructs back all edges	-	Brute Force	-	2D array	O(n^2)

	dijkstra()	Finds alternate shortest path between two edges if two are connected	Adjacent edge of source and destination should be ignored	Greedy Algorithm	Dijstras	2D array	O(V + E) V : Vertices E: Edges
	CheckPipelineHazards()	Respective array of structure and adjacency matrix.	Checks if pipes and drains are next to each other	Brute Force	-	2D array	O(n^2)
	insert()	Root node of the AVL Tree	Inserts nodes into AVL Tree	-	-	Array of Structure	O(log n)
	treeCreation()	Details of all nodes	-	Brute Force	-	AVL Tree	O(n log n)
	inOrderTraversal()	AVL Tree	Visits all nodes inorder traversal.	Recursive	-	AVL Tree	O(n)
	printMaxnode()	AVL Tree	Prints Max node	Recursive	-	AVL Tree	O(h)
	printMinNode()	AVL Tree	Prints Minimum node	Recursive	-	AVL Tree	O(h)
	intializeTreasury()	Hash Tables	Initialises Hash Tables to random values	Brute Force	-	Arrays	O(1)
	HashFuntion()	-	Gives out a hash values for any input	Hashing	Hashing	Arrays	O(1)
	Treasury()	Hash Tables	Gives out the final value	Hashing	Hashing	Arrays	O(1)
	Hashkey()	Hash Tables	Gives possible keys to the Hash function	Hashing, Brute Force	-	Arrays	O(1)
	primsAlgorithm()	Adjacency Matrix of the Graph	Finds the a MST with all the nodes	Greedy Algorithm	Prims	2D arrays	O(n^2)

5. Functionality Analysis

CheckCityFacilities(): Breadth-First Search (BFS) is a graph traversal algorithm that explores a graph level by level. Starting from a specified source vertex, BFS systematically explores all the vertices at the current level before moving on to the vertices at the next level. This algorithm employs a queue data structure to keep track of the vertices to be visited, ensuring that vertices are processed in the order they were discovered.Then during the traversal it calculates the total cost to develop the area.

CheckPipeDrainHazard(): Dijkstra's algorithm is a popular algorithm used to find the shortest path between a source vertex and every other vertex in a weighted graph with non-negative weights. It works well for both directed and undirected graphs. Node first finds a hazard i.e pipe and drain are next to each other for the same vertices. Then that edge is destroyed and an alternate path is suggested.

TreeCreation(): An AVL tree (Adelson-Velsky and Landis tree) is a self-balancing binary search tree. In an AVL tree, the heights of the two child subtrees of any node differ by at most one, ensuring that the tree remains balanced. This balancing property helps maintain efficient search, insert, and delete operations. The function depending on the facilities in a location assign a point. Then this point is used as a key through which a tree is built.
Treasury():

primsAlgorithm(): Prim's algorithm ensures that the constructed tree spans all vertices with the minimum possible total edge weight. It is a greedy algorithm, and its time complexity is generally O(E log V), where E is the number of edges and V is the number of vertices in the graph. The structure of the MST will the path through which cable will be laid as it will also contain the most minimum weights.

6. Conclusion

Graph Representation: Understanding how to represent a real-world railway network as a graph and efficiently store and manage connections between stations.

Dijkstra's Algorithm: Implementing Dijkstra's algorithm for finding the shortest path between two stations. Learning how to use priority queues for efficient traversal of the graph.

User Input Handling: Incorporating user input for source and destination stations or landmarks, demonstrating the importance of robust input handling in real-world applications.

Application of Graphs: Recognising how graph algorithms play a crucial role in solving real-world problems, such as simplifying complex railway networks for optimal route planning.

7. References

Cite your references. Examples are given below:

[1] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2001. Introduction to Algorithms (2nd ed.). McGraw-Hill Higher Education.
[2] lbackstrom. The Importance of Algorithms. Link: https://www.topcoder.com/community/data-science/data-science-tutorials/the-importance-of-algorithms/. Site last accessed on: 23 October 2017.

Notes:

- www.google.com or www.wikipedia.com etc cannot be a reference.
- References can be in IEEE, APA or any other standard format. But put all of them in one single format.
- Remove all the template data before submitting. Example: These notes should not be part of your report.
- Do not deviate from the template provided at any case.
- Submit report in .pdf format.
- Report submissions will be along with code and in soft copy only. NO HARD COPIES.
- Code and report copy has to be emailed to concerned faculty soon after the exam scheduled on 23 / 24th Jan 2024

~*~*~*~*~*~*~*~