|  |
| --- |
| **Data Structures and Algorithms** |
| Ola |
| **Course Project Report** |

|  |
| --- |
| **School of Computer Science and Engineering**  **2021-22** |

**Contents**

|  |  |
| --- | --- |
| **Si. No.** | **Topics** |
| 1. | Course and Team Details |
| 2. | Introduction |
| 3. | Problem Definition |
| 4. | Functionalities |
| 5. | Tools and Techniques |
| 6. | Learning and Takeaway |
| 7. | References |

**1. Course and Team Details**

**1.1 Course details**

|  |  |
| --- | --- |
| **Course Name** | Data Structures and Algorithms  (Theory and Lab) |
| **Course Code** | 20ECSC205 and 19ECSP201 |
| **Semester** | III |
| **Division** | D |
| **Year** | 2021-22 |
| **Instructor** | Prakash Hegade |

**1.2 Team Details**

|  |  |  |
| --- | --- | --- |
| **Si. No.** | **Roll No.** | **Name** |
| 1. | 417 | Michael Rohan Swaminathan |
| 2. | 429 | Nikhil Dupdal |
| 3. | 421 | Prasanna Roogi |
| 4. | 410 | Utkarsh Khot |

**2. Introduction**

The course Data structure and Algorithms (Theory and lab) with course code 20ECSC205 and 19ECSP201 introduces the beginner to world of algorithms. It helps in understanding the uses and application of Data structure. The course wants the beginner to have a look how algorithms make difference in real world. The complex problems from reality are broke down in modules, which are then solved by applying the algorithms. It only explains the inventor’s method which are the bases to infer derived algorithms and in what way the inventor thought of inventing the algorithm. Lab is totally based in the real-world application where we apply are theory knowledge and derive some sort of meaning full solution to some mini world.

This course is included in our syllabus to have knowledge of basic algorithms which implemented in some of our daily life problems by some or the other app or service. Course also encourages us to reckon still how many real-world problems are not solved by algorithm and how don’t have a solution. Example: Travelling salesman problem.

The project in which we are engaged propounds us to look for the algorithm which suits best for our topic.

**3. Problem Statement**

The problem we are automating is of a mock-up application of ola. OLA is a user-friendly cab service that employs shortest path algorithms to arrive at the user’s desired location and take him/her to their desired destination by means of the shortest route available. We are automating the process of ETA of the driver as well as the fare and time estimate from the source to the destination for the user to travel. Also, the user will have his/her ride history which they can use to keep track of their travel expenses.

**4. Functionalities**

Only the main functionalities are given below. Further can be accessed in the source code.

|  |  |  |  |
| --- | --- | --- | --- |
| **SI. No.** | **Function Name** | **Description** | **DS and Algorithm Used** |
|  | user signup | To register their account with the cab service. | Cases used to direct to other functions |
|  | User login | To login with mobile number or email id and password | Redirected to other functions using switch cases |
|  | welcome | First display of the application | None |
|  | Login menu | To go through login options | None |
|  | floyd | Shortest path for all sources from all sources. | Floyd’ Algorithm using 2d arrays |
|  | Ride\_journey | Shortest path from source to destination. | Using Dijkstra’s Algorithm and path, distance and visited arrays. |
|  | Search\_fare,  search\_coupon | Searching for appropriate  Data and returning the index of data found. | Linear search using arrays. |
|  | Tour\_journey | To traverse the minimum spanning tree. | Using Prim’s Algorithm and path, distance and visited arrays. |
|  | Path\_trace | To trace the shortest path of Dijkstra’s Algorithm recursively. | Recursion using path array and destination. |
|  | Min2 | To find the minimum for Dijsktra’s | Comparison and assignment |
|  | Tour\_cost | To find the total distance traversed and thereby calculating the cost. | Kruskal’s algorithm. |
|  | Minimum\_spanning  tree | Extension of Tour\_cost where the union find data structure is utilised | Union Find Data structure utilisation. |
|  | Union\_find | To compute union\_find data structure (disjoint set data structure) | Union Find Data structure implementation |
|  | Quicksort,  partition,swap | Sorting of edges for kruskal’s Algorithm | Quicksort  using structures. |
|  | Load\_driver\_eta | Compute the time taken by the driver to arrive to user’s location | Floyd’s  Algorithm Utilisation |
|  | Generate otp | Mock Application of generating  random string. | Arrays |
|  | Signup, login,  password and location validation | Checking  through constraints for approving the data | Brute force approach |
|  | Open\_log,  close\_log | Log data to check for run time errors | Files |
|  | Loading, dumping and printing | File operations | Files, linked lists and structures. |

**5. Tools and Techniques**

**5.1 Data Structures and Algorithms**

1. **Data structures:**

The data structure that we have implemented is a combination of singly linked lists and arrays whose memory have been dynamically allocated.

We maintained the users list with all their details, the drivers list with all the driver’s details, the various cabs and their basic fares, the ride history of a particular individual and the coupon codes that can be availed for discount. Also, we store the details of the logged in user in a separate structure which we copy from the main users list and also the ride history of the user is noted down in a separate file bearing the user id uniquely given to a user. Also, the time and distance matrix are formulated dynamically with edge list in a sperate file. The shortest path algorithms are implemented using arrays i.e., the distance, path and visited arrays. Also, appropriate structures for the utilization of the shortest path algorithms have been accommodated. Also, union-find data structure has been used to implement the Kruskal’s MST. And the adjacent matrix is maintained with respect to the

1. **Algorithms:**

We have utilised various algorithms for the implementation of our mock up application. First, the shortest path algorithm:

**i) Dijkstra’s Algorithm:**

The go-to algorithm in terms of the shortest path from source to

Destination. As, we aren’t dealing with negative edges, this algorithm

Definitely fits the bill. We implemented using the distance, path and

Visited arrayand keeping track of the minimum weight while traversing

Through the data set. As we implemented using arrays, the time

Complexity of its O(V^2), which the square of the number of vertices or

Nodes or in our case the locations we can traverse to. Also, for source to

destination traversals, we will calculate the fare per min travelled and

therefore, show the path traversed recursively.

**ii) Floyd’s Shortest Path Algorithm:**

The Floyd algorithm solves the all-pairs shortest path problem in O (V^3) time. even though there may be up to {\displaystyle \Omega (|V|^{2})} O(V^2) edges in the graph, and every combination of edges is tested. It does so by incrementally improving an estimate on the shortest path between two vertices, until the estimate is optimal. The Floyd algorithm uses dynamic programming and divides the given problem into subproblem. The Estimated time of arrival (ETA) of the driver to one’s location could be easily calculated and compared and then brought forward with the nearest driver to the user’s location.

**ii) Prim’s Algorithm:**

Prim’s algorithm runs in O(V^2) when implemented using arrays and varies with other forms of data structures. Then the effective runtime of the algorithm varies with the data structures used to implement the algorithm. Prim’s algorithm depends on a method of determining which greedy choices are safe. The method is to continually enlarge a single connected component by adjoining edges emanating from isolated vertices. As the tour function has to travel through all the vertices resulting in a minimum spanning tree and gives the traversal of the user through the cab.

**iii) Kruskal’s Algorithm:**

Kruskal’s algorithm Uses the disjoint-set data structure. For a graph with *E* edges and *V* vertices, Kruskal's algorithm can be shown to run in O (E log E) time, or equivalently (*E* log *V*) time. Kruskal’s will be used to gather the distance traversed during the tour and therefore calculate the cost by means of the basic fare and distance traversed. Quick Sort is implemented to sort the edges which will be talked about in the next section.

**iv) Quicksort:**

Quicksort is a type of divide and conquer algorithm for sorting an array, based on a partitioning routine; the partitioning we have considered is the usual using the pivot element. It is Applied to a range of at least two elements, partitioning produces a division into two consecutive non empty sub-ranges, in such a way that no element of the first sub-range is greater than any element of the second sub-range. After applying this partition,

the quicksort then recursively sorts the sub-ranges, possibly after excluding from them an element at the point of division that is at this point known to be already in its final location. Due to its recursive nature, quicksort (like the partition routine) has to be formulated so as to be callable for a range within a larger array, even if the ultimate goal is to sort a complete array. We use this algorithm to sort the edge lists to implement Kruskal’s and is according to the algorithm specified by Levitin in his book.

The average case of Quicksort is O(nlogn) but the worst-case O(n^2), so the implementation needs to be done carefully.

**v) Brute force string search:**

We do a brute force string search when finding the locations for

Pickup and destination, as this was the only substring search algorithm we could implement. It has a running time of O(m\*n).

**vi) Linear search of data:**

O(n) running time as the implementation of binary search proved to

Be a tough piece of puzzle to solve as the sorting of the various lists had to

Be taken into account.

**5.2 Project Statistics**

|  |  |  |
| --- | --- | --- |
| **Si. No.** | **Measure** | **Value** |
|  | Total Functions in Project | 60 |
|  | Total number of lines of code  (Including comments, newlines etc.) | 2325 |
|  | Number of Errors | 0 |
|  | Number of Warnings | 0 |
|  | Team Satisfaction about Project | 95 |

**6. Learning and Takeaway**

1. How to deal with the files in the secondary storage.

2.The bridge between theory and practical application was brought out while implementing the project.

3. The patience and tolerance required to implement the project.

4. The brain-storming sessions of the “why, how and where” the implementations should be done and lead to.

5. Understanding the real-time scenario and how it can be best show-cased through the console application.

6. Debugging of the code was the most tiresome and mentally challenging issue we faced!!

7. Learning to utilize the algorithms and data structures in a systematic and appropriate manner.

8. The art of programming and making the computer understand our vision and implement it.

9. The beauty of shortest path algorithms and the perspective required to harness it was truly inspirational and reminds us of the importance of it in our daily lives.

10. Maintenance of a log file to note the run time errors in the program is a handy tool we have learnt.

11. The respect gained for the developers who have developed various applications and also motivating us to move forward with the same attitude.

12. Last but not least, the fun that we had during the process of learning and understanding this course and lab will truly be memorable.

**7. References**

[1] Thomas H. Cormen, Clifford Stein, Ronald L. Rivest, and Charles E. Leiserson. 2001. Introduction to Algorithms (2nd ed.). McGraw-Hill Higher Education.

[2] Anany Levitin.2012 Introduction to Design and Analysis of Algorithms (3rd Edition). Pearson Publications.

[3]OLA.com

**~\*~\*~\*~\*~\*~\*~\*~**