|  |
| --- |
| **Data Structures and Algorithms** |
| Train Problem |
| **Course Project Report** |

|  |
| --- |
| **School of Computer Science and Engineering**  **2023-24** |

**Contents**

|  |  |
| --- | --- |
| **Si. No.** | **Topics** |
| 1. | Course and Team Details |
| 2. | Introduction |
| 3. | Problem Definition |
| 4. | Functionality Selection |
| 5. | Functionality Analysis |
| 6. | Conclusion |
| 7. | References |

**1. Course and Team Details**

**1.1 Course details**

|  |  |
| --- | --- |
| **Course Name** | Data Structures and Algorithms |
| **Course Code** | 23ECS205 |
| **Semester** | III |
| **Division** | A |
| **Year** | 2023-24 |
| **Instructor** | Mr. Prakash Hegade |

**1.2 Team Details**

|  |  |  |
| --- | --- | --- |
| **Si. No.** | **Roll No.** | **Name** |
| 1. | 101 | Anirudh R. Hanchinamani |
| 2. | 154 | Om Muddapur |
| 3. | 158 | Sumedh S. Kaulgud |
| 4. | 159 | B. R. Abhishek |

**1.3 Report Owner**

|  |  |
| --- | --- |
| **Roll No.** | **Name** |
| 101 | Anirudh R. Hanchinamani |

**2. Introduction**

The project began with the examination of a comprehensive document provided to our team—an insightful “Streets” paper that served as a repository of prevalent urban challenges. From the myriad of issues presented, our collective decision converged on a problem that resonated with both common experiences and personal observations. The focal point became addressing the challenges faced by individuals who are new to the city, particularly in the context of urban rail transportation—a multifaceted challenge.

Following are the Sub Modules that we have enlisted in this vast challenge of train problem,

* Information and Communication.
* Infrastructure and Facilities.
* Passenger comfort and assistance.
* Sustainability and Green Initiatives.

**3. Problem Statement**

The Train Problem: Seeking the most viable options to improve passenger mobility and provide better accommodations.

**3.1 Domain**

The problem statement revolves around seeking optimal solutions to enhance passenger mobility and provide improved accommodations in the realm of urban rail transportation. This initiative was motivated by a keen understanding of common experiences and personal observations outlined in the 'Streets' paper.

**3.2 Module Description**

**Module** **Name**: Passenger comfort and assistance.

This module focuses on providing assistance to passengers, the primary objective is to address challenges encountered by individuals who are new to the platform and unfamiliar with the city they are in. The module aims to provide comprehensive solutions to common queries faced by such individuals.

One notable feature is "City Promotions," allowing users to access information about the top cities in the state with detailed descriptions. This feature assists newcomers in discovering appealing destinations within the region. Additionally, the module encourages user engagement by incorporating a feedback mechanism, enabling continuous improvement and customization of the assistance provided.

To enhance user support within the railway station premises, the module offers features like navigation assistance. Users, especially those unfamiliar with the layout of the railway station, can rely on the system to guide them within the station, ensuring seamless navigation to their desired platforms. Furthermore, the module incorporates a search functionality, allowing users to find specific places within the station and receive directions to the corresponding platforms.

Recognizing the need for accommodations, the module extends its functionality to include information about dormitory options within or near the station premises. Users can conveniently view available dormitory facilities and proceed to book them through the system.

Furthermore, the module offers users the opportunity to take notes, view their notes, and mark them as completed. This functionality enhances the user experience by providing a convenient way to organize and track tasks or important information during their journey. Whether it's jotting down reminders, making to-do lists, or noting essential details, this feature adds a personalized and efficient dimension to the overall assistance provided by the module.

In summary, this module within Passenger Comfort and Assistance is dedicated to creating a user-friendly interface addressing the unique challenges faced by individuals new to a city and its railway station. Through features such as City Promotions, user feedback, navigation assistance, and dormitory options, the module significantly enhances the overall comfort and experience of passengers in unfamiliar surroundings.

**4. Functionality Selection**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Si. No.** | **Functionality Name** | **Known** | **Unknown** | **Principles applicable** | **Algorithms** | **Data Structures** |
| 1 | Platform Assistance | Platforms count and connections, Distance between the platforms. | Managing dynamic user data and ensuring secure storage of user notes. | Dijkstra's Algorithm (Edge Relaxation), User Data Encryption. | Dijkstra's Algorithm, Knuth Morris Pratt Algorithm | Array of Structures or Trees |
| 2 | **Providing User to Make Notes/To-Do’s** | User-generated notes and to-do item | need to include both Recommendation Engine as well as Search Engine in this functionality | User Interface Design Principles, Data Synchronization. | Quick Sort | File Handling |
| 3 | City Promotion | City's heritage sites, a short description about it, rating etc | Effective promotion techniques, one thing which can be done is based on their search the machine can recommend. | Greedy Approach | Recommendation System / Seach Engine | Array of Structures |

**5. Functionality Analysis**

The following are the various functionalities that are included in the module:

* City Promotions.
* Get Platform Assistance.
* Providing user to make Notes/ To-Do’s.

In this module there are some infinite loops which are used to provide Console application feel and they don’t contribute to the calculation of Time Complexity

* 1. **City Promotions**

In this functionality, the workflow begins with the user being prompted with four different assistance options:

* Option To View Top City
* Option To View Top N Cities
* Option To Search for a City
* Option to Add Feedback
  + 1. **To View Top City**

To implement the aforementioned assistance, the raw data was available in the file. To identify the top city from the raw data, the utilization of the standard Quick Sort function is evident here. The Quick Sort function exhibits diverse time complexities based on different scenarios: O(n\*log(n)) for its best-case performance, O(n2) for the worst-case scenario, and O(n\*log(n)) on average. Regardless of the case, the space complexity consistently stands at O(n) (Patel, 2023).

* + 1. **To View Top N Cities**

To aid the user in identifying the top N cities from the file, we've implemented the standard Insertion Sort algorithm. The Best-Case Time Complexity occurs when the array is already sorted, resulting in O(n). However, the Worst-Case and Average-Case Time Complexities are O(n^2). In terms of space complexity, it remains consistent across all cases at O(n) (Gupta, 2023)

* + 1. **To Search for City**

To search for a city name in the text file, which is then loaded into an array of structs, the space complexity has been maintained at O(n), and the time complexity at O(n). However, the adoption of a Brute Force string search algorithm (Pedamkar, Brute Force Algorithm, 2023)on each line of the loaded data has increased the time complexity to O(n\*m), where 'm' is the length of the string. This algorithm exhibits both best and worst-case time complexities, with the former being O(k\*m), where 'k' represents the string found within the first 'k' iterations, and the latter being O(n\*m).

* + 1. **To Add Feedback**

This functionality is designed to gather user feedback about a city, incorporating a robust locking system encrypted with a password. Upon the user entering the username and password, a background data check ensues, searching for duplicate accounts with the same username. This check is implemented using the standard Rabin Karp String Search Algorithm (Cox, 2021). Its best-case time complexity is O(m+n), where 'm' and 'n' represent the lengths of the pattern and the text, respectively. In the worst-case scenario, when brute force comparison is necessary, the time complexity becomes O(n\*m), particularly when coupled with a poor hash function. Additionally, it requires O(m) time to obtain the hash value needed for the search.

* 1. **Get Platform Assistance**

In this functionality, the workflow begins with the user being prompted with four different assistance options

* Assistance to Inter-Platform Commutation
* Assistance to Find Tourist spot near Platform
* Search for Dormitories of Required type
  + 1. **Assistance to Inter-Platform Commutation**

In the implementation of this functionality, Dijkstra’s Algorithm (baeldung, 2022)is employed to determine the shortest path between two distinct platforms. The standard Dijkstra’s Algorithm, utilizing Fibonacci heap, exhibits a much faster time complexity. However, within the project, a method with a time complexity of O(n) is employed to obtain the minimum element inside a loop that iterates n times. Consequently, the overall time complexity of this functionality is O(n2).

* + 1. **Assistance to Find Tourist spot near Platform**

In this functionality, the raw data associating each place with its corresponding station is copied to a struct, resulting in time and space complexities of O(n). However, the inclusion of the Knuth Morris Pratt Algorithm (Pedamkar, 2023) introduces an additional time complexity of O(m+n) for each line of data. This holds true for both the best and worst cases.

* + 1. **Search for Dormitories of Required Type**

To facilitate passengers in choosing the dormitories of their preferred type, an implementation has been developed. The process involves loading data from a file, which takes O(n) time to iterate through all the details. Additionally, inserting nodes into a Binary Search Tree (BST) (Sharma, 2023) is part of the process, and this operation takes O(log(n)) time. Consequently, the overall time complexity for this functionality is O(n\*log(n)).

Furthermore, the user-provided dormitory type is identified using the Breadth-First Search (BFS) algorithm (Sauce, 2023), which has a complexity of O(v+e), where 'v' is the number of vertices and 'e' is the number of edges. Within the BFS algorithm, the Knuth Morris Pratt (KMP) algorithm is called for a number of times equal to the sum of vertices and edges, resulting in a time complexity of O((v+e) \* (m+n)) for both the best and worst-case scenarios

* 1. **Providing user to make Notes/ To-Do’s**

This functionality follows a similar approach, involving the retrieval of data from a file, incorporation into a structure, and the execution of a Knuth Morris Pratt (KMP) search operation. The time and space complexities for this process remain consistent with the aforementioned functionality.

**6. Conclusion**

Concluding the Data Structures and Algorithms (DSA) course marks the culmination of an extensive exploration into the fundamental building blocks of computer science. The course has provided a thorough understanding of various data structures and algorithms, establishing a solid groundwork for solving complex computational problems.

Various algorithmic strategies, such as divide-and-conquer, dynamic programming, and greedy algorithms, have been thoroughly examined throughout the course. The acquired skill of analysing the time and space complexity from basic sorting to difficult graph related algorithms serves as a crucial asset. The ability to evaluate algorithmic efficiency and make informed decisions about their applicability in different scenarios is a key takeaway.

Algorithmic problem-solving is highlighted by the practice of breaking down complex problems into more manageable subproblems, a theme evident throughout the entire module written here. Engaging with these challenges not only deepened understanding but also fostered confidence in approaching and resolving complex computational problems.

# **7. References**

baeldung. (2022, November 9). *Understanding Time Complexity Calculation for Dijkstra Algorithm*. Retrieved from https://www.baeldung.com/cs/dijkstra-time-complexity

Cox, J. (2021, April 4). *The Rabin-Karp String Searching Algorithm*. Retrieved from https://medium.com/smucs/the-rabin-karp-string-searching-algorithm-21c2b50905f4

Gupta, V. (2023, March 14). *Visualizing, Designing, and Analyzing the Insertion Sort Algorithm.* Retrieved from https://medium.com/javarevisited/visualizing-designing-and-analyzing-the-insertion-sort-algorithm-latest-4bb963fd82f7

Patel, H. (2023, December 19). *QuickSort Algorithm: An Overview*. Retrieved from https://builtin.com/articles/quicksort

Pedamkar, P. (2023, February 28). Retrieved from https://www.educba.com/kmp-algorithm/

Pedamkar, P. (2023, June 7). *Brute Force Algorithm*. Retrieved from https://www.educba.com/brute-force-algorithm/

Sauce, T. (2023, June 6). *Time Complexity and Space Complexity of DFS and BFS Algorithms*. Retrieved from 2023

Sharma, R. (2023, December 7). *Binary Search Algorithm: Function, Benefits, Time & Space Complexity*. Retrieved from https://www.upgrad.com/blog/binary-search-algorithm/

Woltmann, S. (2020, August 19). *Heapsort – Algorithm, Source Code, Time Complexity*. Retrieved from https://www.happycoders.eu/algorithms/heapsort/

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