

“ Smart Metro Transportation of Chirasmriti “

Problem Statement of Smart Metro Transportation

The first phase of the Chirasmriti City project focuses on the initial development of a smart metro transportation system. **The challenges addressed under different modules are Efficient route network, Efficient ticketing system , Platform Crowd and ensuring smooth railway infrastructure for quick movement between platforms and issues related to passenger ticket allotment and platform management. These problems are explored as separate modules within this domain.** These problems were identified based on insights gained from a case study of the local train system in Mumbai, as outlined in the corresponding white paper and also based on the personal experience.

Module Description and Functionalty Analysis:

1. “ Connecting the City: Designing efficient Metro Route Networks”

The creation of an efficient route network in a metro system involves strategically planning the layout of lines and stations to maximize coverage while minimizing travel time for passengers.

a) Functionality Selection

Si. No.	Functionality Name	Known	Unknown	Principles applicable	Algorithms	Data Structures
	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?

1	The efficient design of the network. This allows the establishment of different routes	<p>(i)The design of metro lines in cities such as Tokyo, Hong Kong (MTR), Mumbai, New York, and Moscow has been studied through information available on their official websites that helped to analyse the efficiency and requirement of city.</p> <p>(ii)The case study of tokyo metro line: The research conducted by Hokaido University in 2010 on slime mould(Connecting to portfolio activity given at the beginning of semester) proved that they constructed a network of nutrient-channeling tubes that was strikingly similar to the layout of the Japanese rail system.</p> <p>(iii)Required knowledge on graph algorithms on finding mst and shortest path.</p>	<p>Since the locations were unknown, the points had to be laid under assumptions.</p> <p>Finding the shortest distances between all given pairs</p>	<p>(i) Design Principle: Kleene Closure. Design Technique: DP</p> <p>(ii) Design Principle: Edge Relaxation. Design Technique: Greedy.</p>	<p>(i) Warshall Algorithm.</p> <p>(ii) Kruskal's Algorithm .</p>	<p>(i) 2D Adjacency matrix.</p> <p>(ii) Union-Find</p>
2	To locate the metro stations that fall under a particular route (line).	The possible algorithms that would help to arrive at the result.		<p>Design Principle: Pruning Design Technique: Path Compression</p>	Disjoint-Set Algorithm	Union-Find by path Compression

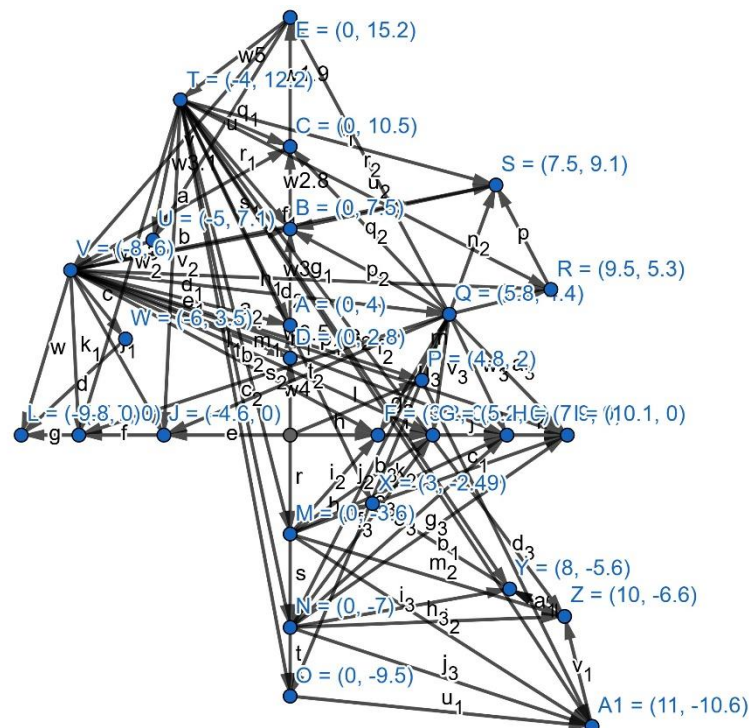
b) Functionality Analysis

(1) Functionality Analysis of creating efficient route network.

Page | 2

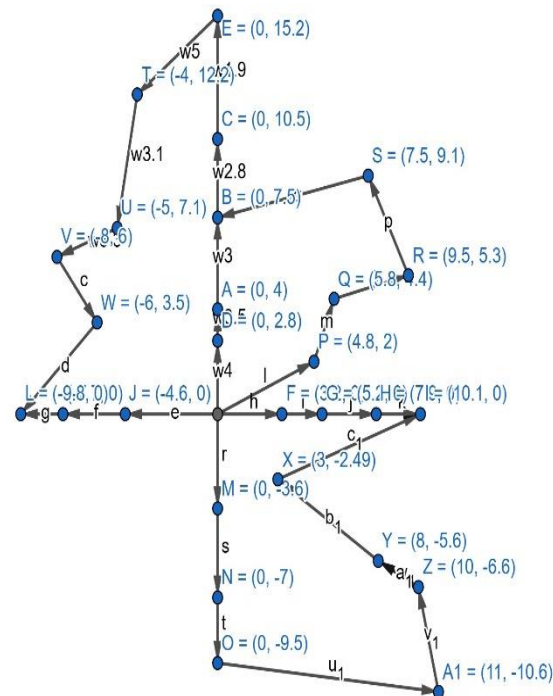
Assumptions:

- Since the latitudes and longitudes are unknown, the taken coordinates represents (x kms,y kms) of a place, from centre(origin) of the city.
 - { positive y-axis: some kms North; negative y-axis: some kms south ; positive x-axis: some kms some kms west; negative x-axis: some kms east }
1. Select the ideal locations for the 28 new metro stations that should be built.
 2. Find the pairwise distances between all 28 stations.



1. NOTE: Use the distance formula (take x and y as coordinates).
2. For a particular station, there are 27 options available. Select the minimum distance.
3. The above step can be achieved by using the Warshall algorithm.
4. Warshall Algorithm determines the shortest possible path (weight: distance) between any two stations by adopting dynamic programming technique.

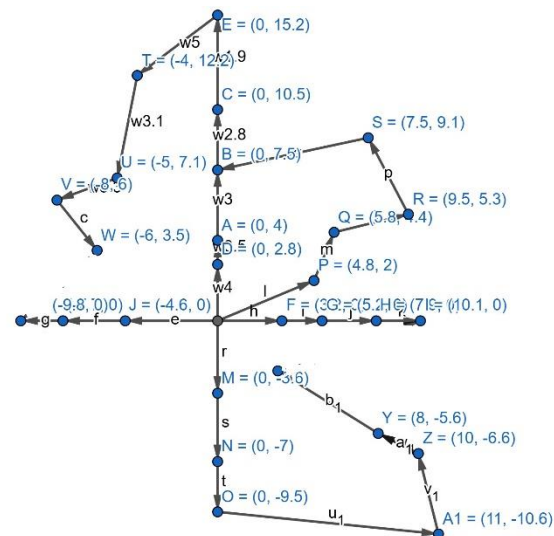
- Take the distances to be weights of the graph. Arrive at R^4 , the matrix that includes the shortest path between all pairs.
- Design principles: shortest path trees.
- Time complexity: $O(n^3)$.
- Data structure used: 2D matrix (adjacency matrix).



The graph obtained after applying Warshall's Algorithm

1. Convert R^4 into a graph and apply Kruskal's algorithm.
 - Sort the distances.
 - Each node represents a station.
 - Construct the minimum spanning tree that outlines the optimal networking path.
 - Design principle: Edge Relaxation.
 - Data structures used:
 - If using a Union-find, time complexity is $O(E \log V)$;

- If using union-find, time complexity is $O(\log E)$.



The graph obtained after applying Kruskal's Algorithm

```

E:\c++\optimized_kruskal.exe
11 -- 12 == 1.9
1 -- 2 == 2
6 -- 7 == 2.1
3 -- 4 == 2.2
18 -- 19 == 2.3
14 -- 15 == 2.5
25 -- 26 == 2.6
2 -- 3 == 2.7
10 -- 11 == 2.8
9 -- 10 == 3
5 -- 6 == 3.1
0 -- 1 == 3.2
22 -- 23 == 3.3
13 -- 14 == 3.4
8 -- 9 == 3.5
0 -- 13 == 3.6
19 -- 20 == 3.7
26 -- 27 == 3.8
13 -- 17 == 3.9
0 -- 8 == 4
0 -- 5 == 4.6
12 -- 21 == 5
0 -- 25 == 5.2
17 -- 18 == 5.3
23 -- 24 == 5.5
21 -- 22 == 6
Minimum Cost Spanning Tree: 91.2
Process returned 0 (0x0)   execution time : 428.623 s
Press any key to continue.

```

Conclusion: Based on the resulting MST, the metro network can cover a distance of 91.2 km, which supports the possibility of having 5 metro lines.



Alternate possible algorithms: (Analysis Slime Mould case study)

1. Breath-First-Search (BFS):

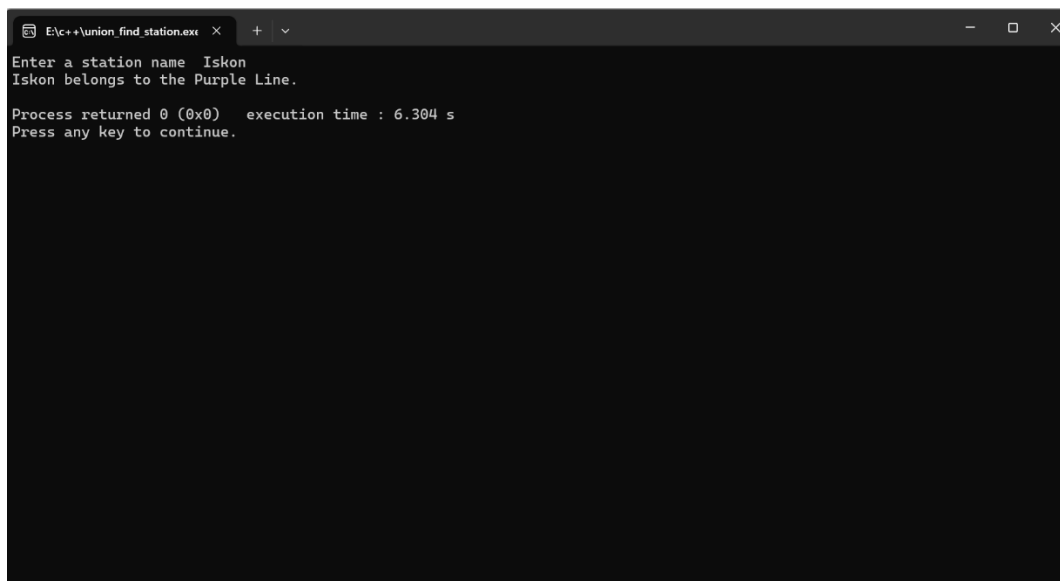
- Each station represents a node.
- From each station, other connected nearby stations must be explored. Backtrack and explore unvisited stations, thereby networking the entire city.
- Design Principle: Cautious, Level-order traversal.
- Data structure used: Queue. Time complexity: $\Omega(|V|^2)$ (V: vertex).

(2)Functionality Analysis to locate the metro stations that fall under a particular route.

To accurately determine which metro line a specific station belongs to, we can utilize the find() method, which scans through the various metro lines and identifies the one containing the station. This process is crucial for ensuring that stations are properly assigned to their respective lines, aiding in the efficient mapping of the metro system and optimizing route planning.

Page | 6

Efficiency Analysis: $O(1)$ (By path compression)



```
E:\c++\union_find_station.exe x + | v
Enter a station name Iskon
Iskon belongs to the Purple Line.
Process returned 0 (0x0)   execution time : 6.304 s
Press any key to continue.
```

c) Conclusion

- The project intrigued our thoughts and encouraged us to think critically about how algorithms can be applied to real-world challenges.
- It also helped to apply and feel the importances of learnt algorithms in real life.

2. Traffic Management: Smart Passenger Queue and Platform Allocation System

1. Allocating passengers to specific counters based on age and gender.
2. Managing metro platforms and schedules efficiently.

a) Functionality Selection

Si. No.	Functionality Name	Known	Unknown	Principles applicable	Algorithms	Data Structures
	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?
1	Sorting Passengers and allotting tickets.	Sorting by age, Counters allotted on the basis of age and gender.	Optimizing sorting for large data sets; handling edge cases like duplicate ages.	First in, first out (FIFO)	Queue	Vector, Queue
2	Train Platform Scheduling	Train schedule.	Real-time updates for train schedules; handling collisions in hash table efficiently	Hashing Principles	Hash	Hash Table

b) Functionality Analysis

5.1 Sorting Customers

- **Workflow:**
 - Passengers are assigned to different counters based on their age and gender.
 - Helps passengers get tickets easily without much delay and also avoids congestions.
 - Logic:
 - To sort passengers on the basis of their age:
 - 1) Senior citizens (60+): Assigned to dedicated counters
 - Counter 1.1 for males and 1.2 for females
 - 2) Adults (18-59): Assigned to separate counters
 - Counter 2.1 for males and 2.2 for females
 - 3) Minors (<18): Assigned to separate counters
 - Counter 3.1 for males and 3.2 for females

- **Efficiency:**

- Time Complexity: $O(n)$, where n is the number of passengers.
- Data Structure: Queue for assigning counter.

5.2 Train Platform Management

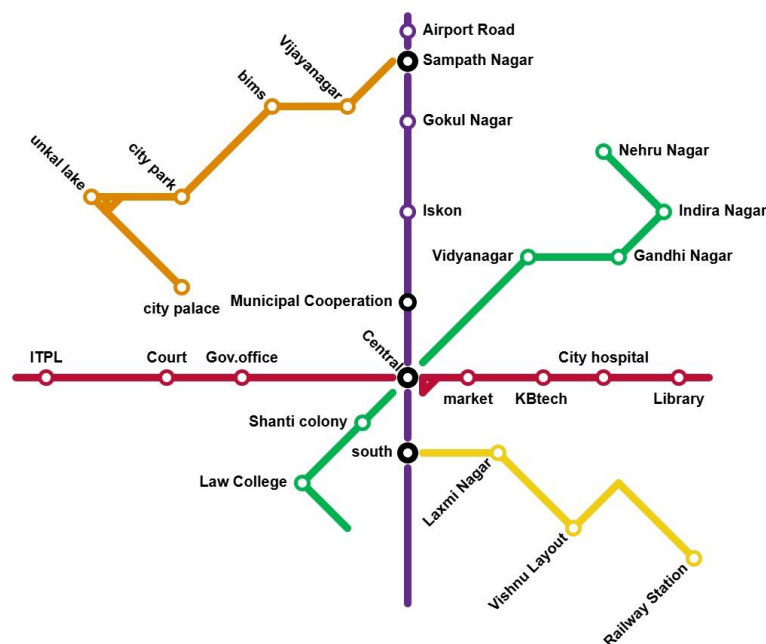
Page | 8

- **Workflow:**

1. Train platforms and schedules are stored in a hash table.
2. This would help people locate the platform easily and also avoids Unnecessary Crowd on the Platforms.
3. Operations:
 - Insertion: Adding new schedules.
 - Searching: Finding schedules for a specific platform.
 - Delete: Deleting outdated schedules.
 - Update: Updating the time or platforms information.
 -

- **Efficiency:**

- Time Complexity: $O(n)$ for insertion, search, and deletion using hash tables.
- Data Structure: Hash Table.



Sample Outputs of Implementation:

1) Sort:

```
_____ALLOTED COUNTERS_____
Ana (20 years, F): Counter 2.2 (Adults - Female)
Adi (24 years, M): Counter 2.1 (Adults - Male)
Ani (27 years, M): Counter 2.1 (Adults - Male)
Vij (71 years, F): Counter 1.2 (Senior Citizens - Female)
Vii (21 years, F): Counter 2.2 (Adults - Female)
Aru (12 years, F): Counter 3.2 (Minors - Female)
```

Page | 9

2) Queue:

```
_____QUEUE_____
Counter 2.2 (Adults - Female): Ana Vii
Counter 2.1 (Adults - Male): Adi Ani
Counter 1.2 (Senior Citizens - Female): Vij
Counter 3.2 (Minors - Female): Aru
```

3)Issuing Tickets

```
_____Issuing tickets_____
Ticket issued to Ana from Counter 2.2 (Adults - Female).
Ticket issued to Adi from Counter 2.1 (Adults - Male).
Ticket issued to Vij from Counter 1.2 (Senior Citizens - Female).
Ticket issued to Aru from Counter 3.2 (Minors - Female).
Ticket issued to Vii from Counter 2.2 (Adults - Female).
Ticket issued to Ani from Counter 2.1 (Adults - Male).
```

4) Hash Map:

```
_____Train Platform Details_____
Central-- Platform 1: 8:30 AM
Market-- Platform 2: 9:00 AM
Library-- Platform 3: 10:15 AM
Updated platform at 'Market'

_____Train Platform Details_____
Central-- Platform 1: 8:30 AM
Market-- Platform 2: 9:30 AM
Library-- Platform 3: 10:15 AM
```

c)Conclusion

This project has highlighted the practical application of data structures and algorithms in solving real-world problems. By integrating quick sort for passenger sorting, hash tables for platform management, and rule-based logic for counter assignment, the project demonstrates how computational principles can improve urban transportation systems. The learning outcomes include the effective use of dynamic programming, greedy algorithms, and hashing techniques to address complex problems.

Page | 10

3.Long bridges and subways in the railway station hinders quick movement of people between platforms.

WHY THIS TOPIC?

As of my personal experience :

When I was once travelling from Hubli to Hospet through , took a ticket in platform 1 but had that train departure on platform no 6 which was too far. Me and my friend couldn't make it till platform no 6 and get onto it even we asked a quick transport but it was still not possible.

Because:

- 1.The entrance of railway station always starts from platform no 1 and we have to take the tickets in platform no 1 itself and then run towards the train to the other platform.
- 2.To move between platforms for eg: from P1 to P5 you had to take a long path move through the stage cross a long bridge which would take a lot of time.
- 3.There is no short paths (shortcuts) between platforms for the people departure and arrival to the exit and entry points of the railway station respectively.

a) Functionality Selection

Si. No.	Functionality Name	Known	Unknown	Principles applicable	Algorithms	Data Structures
	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?
1	Platform Navigation System	Long path, bridge, platform layout	Real-time crowd density, traversal time	User experience design, Path optimization	Dijkstra Algorithm	Graph, Map
2	Quick Transport Between Platforms	No short paths, need for quick transport	Feasibility, potential routes	Efficiency, Accessibility	Pathfinding algorithms	Priority Queue, Linked List

b) Functionality Analysis

For each module you have implemented, describe your workflow and write its efficiency analysis. Create as many sub headings as necessary. It is compulsory to do efficiency analysis for each module.

Quick Transport Between Platforms

To create a Direct Movable Bridge above the Track itself for the easiest movement between one platform to another especially helpful for old age people and children.

Constraints (The solution is only applicable if :)

1. Ticket counters are on the first, middle, and last platforms so that people do not have to much travel to take the ticket and then run .
2. There are separate entry and exit points in the first and last platforms.
3. Train arrival and departure time of the adjacent platforms should be same.

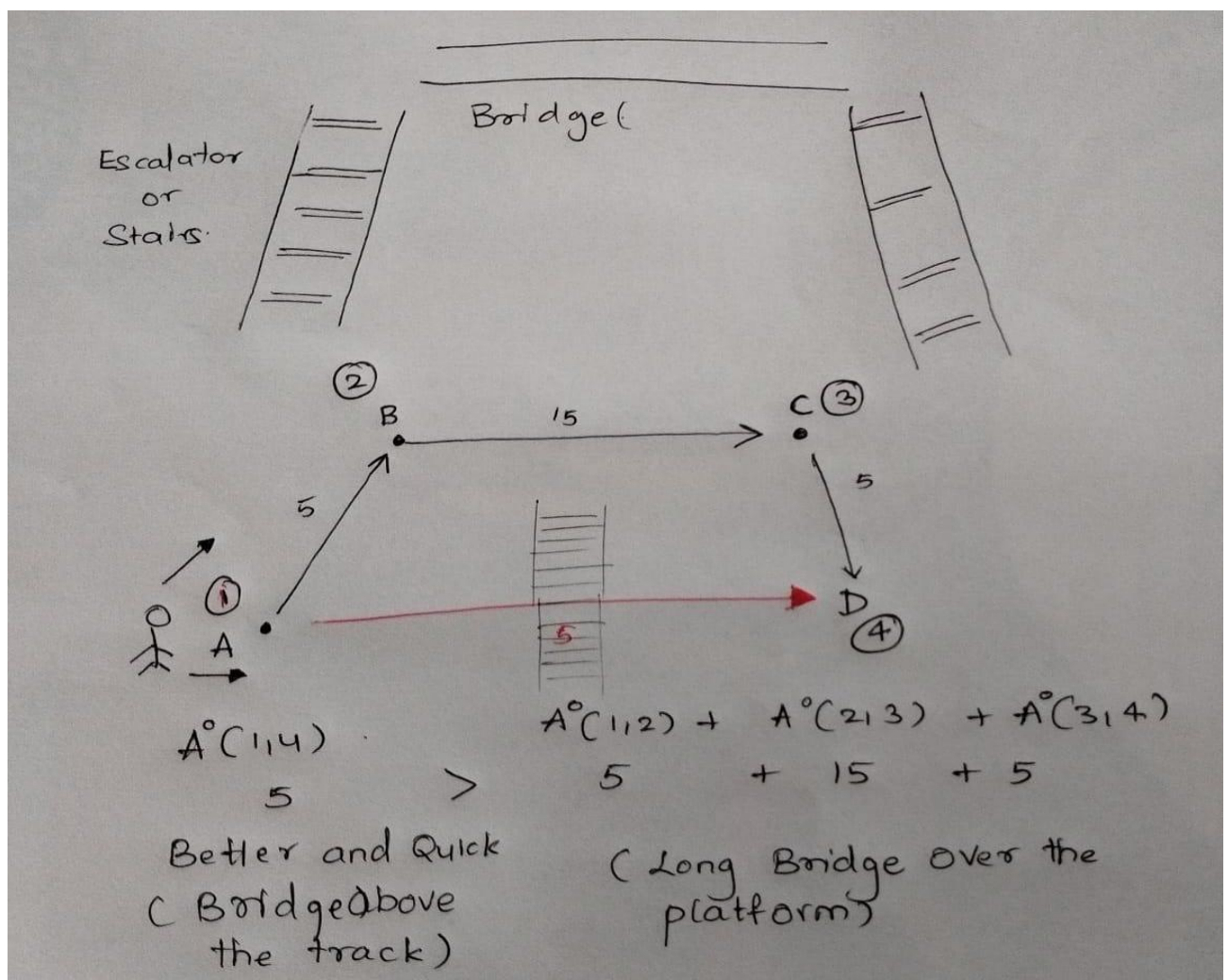
Page | 12

Assumptions

1. Access to a train schedule or a train detection system.
2. The bridge is represented by a state variable (bridge_state) that toggles between "deployed" and "retracted."

The algorithm used to find the shortest path as a bridge – Floyd Algorithm

A simple illustration of How the shortest path is found as a solution of quick movement Using Floyd algorithm.



Explanation :

Assume 3 pairs A to B and B to C for the stairs and B to C for the pavement.

The passenger wants to move to point D assuming the train is on the other side.

So to move from platform one to platform 2 He can directly have a path from A to D

A direct path can be proved through the Floyd algorithm.

Page | 13

And the movement of the bridge can be through this pseudo code-

```
If(train_arrival <=5) //If the train arrival is in 5 minutes
```

```
{
```

```
    Print("Retract the Bridge") // Remove the Bridge
```

```
}
```

Else

```
{
```

```
    Print ("Deployed the bridge") //Keep the bridge
```

```
}
```

The real application of this problem includes a lot more concepts as below:

Train Schedule: This code uses a simulated train schedule. In a real-world scenario, this could be replaced with live data from sensors or APIs.

Threshold Logic: The bridge retracts when a train is within 10 minutes of arrival.

Simulation: The simulate function runs a loop to periodically check the train schedule and update the bridge state.

Extensibility: You can enhance this code by integrating real-time APIs or hardware control (e.g., using GPIO pins on a Raspberry Pi).

c)Conclusion

During the project working

1. Knowledge of the algorithms using it in real scenarios explores a lot of understanding of the subject back to the mind
2. We cant use direct algorithms as a solution to any real problem in the world. We need to modify it and apply it accordingly.

4) Managing Crowd Density: Designing Smart Solutions for Platform Crowd Management:

Platform crowd management uses real-time data from sensors, cameras, and ticketing systems to monitor passenger movements and crowd density. This helps to predict congestion and adjust train schedules to optimize the passenger flow and enhance safety.

Page | 14

a) Functionality Selection

Si. no.	Functionality Name	Known	Unknown	Principles applicable	Algorithms	Data Structures
	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?
1	Detecting overcrowded platforms.	(i) Using real-time data, efficient algorithms, and advanced technologies, cities like London(UK), Singapore, Tokyo(Japan) and New York City(USA) predict the	Algorithms that predict crowd behavior and train schedules may rely on historical data that do not accurately predict the current conditions.	(i) Design Principle: Rotations for balancing the tree (ii) Design Principle: Binary Search Tree (BST) Properties	AVL tree	Nodes Balance Factor

		<p>congestion, and improve transit efficiency.</p> <p>(ii) TfL(London) uses real-time data to monitor passenger flow across the underground network. Advanced algorithms predict crowding, while dynamic signage(digital displays) helps guide passengers to less congested areas.</p> <p>(iii) Singapore's MRT system monitors the crowd density to adjust train schedules and send real-time alerts to about crowded</p>	<p>Integrating multiple systems- train schedules, ticketing, crowd sensors, platform monitoring for accurate crowd management.</p>			Parent node pointers
--	--	--	--	--	--	----------------------

		platforms or delays, ensuring smoother operations.				
2	Rescheduling the train arrivals and departure by prioritizing highly crowded platforms.	The possible algorithms that would help to arrive at the result.		<p>(i) Design Principle: Heap Property</p> <p>(ii) Design Principle: Array-Based Representation</p>	Max heap Priority Queue	<p>Structure of the Heap Node</p> <p>Array</p>

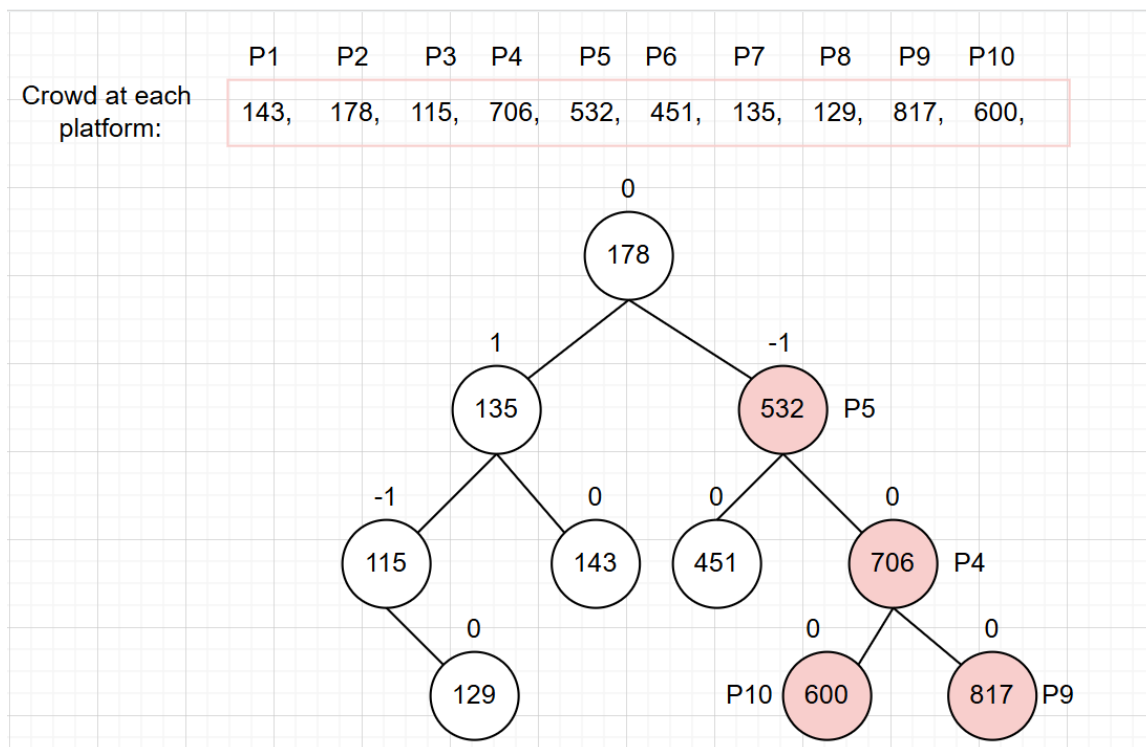
b) Functionality Analysis

(1)Functionality Analysis of detecting overcrowded platforms:

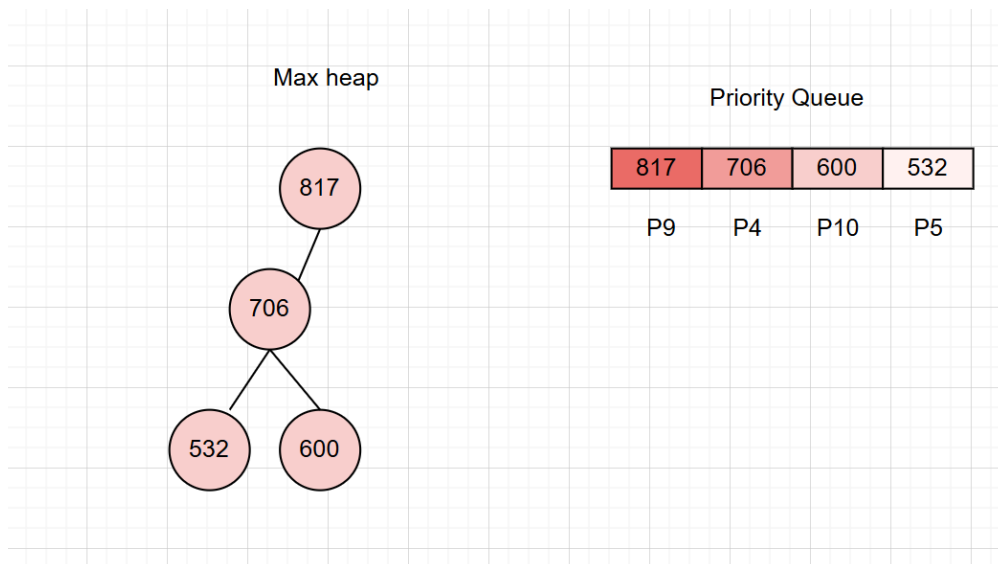
Assumptions:

- Real-time data is continuously available from platforms through sensors, cameras, or ticketing systems.
 - A pre-defined crowd density threshold of 500 passengers exists that determines whether a platform is overcrowded (assume the number of platforms to be 10).
3. Consider crowd density at each platform at a particular time as the key in the nodes of the AVL Tree.
 4. As passenger data arrives in real-time (from sensors, cameras or ticketing systems) the AVL tree allows dynamic insertion of platform nodes with crowd densities.
- ➔ Insert each node as per binary search tree properties

→ Balance the AVL tree after every insertion.



5. Scan all the platforms to detect the overcrowded platforms (with crowd density 500 or more)
 - delete those nodes from the AVL Tree. and insert them into priority queue
6. Using Max heap priority Queue the platforms are prioritized based on their crowd density, ensuring that highly overcrowded platforms are addressed first.
7. The priority queue works alongside the AVL tree to efficiently reschedule trains.
8. Overcrowding in certain platforms can also occur during special events or festivities .
9. Peak hours can be managed by sending empty trains to the overcrowded platforms.

**Time Complexity of an AVL Tree:**

-> Search: $O(\log n)$

-> Insertion: $O(\log n)$

-> Deletion: $O(\log n)$

-> Traversal: $O(n)$

The picture below is the result after implementing the code using AVL Tree for the information from 3 platforms:

Inputs:

Platform Crowd Density: Real-time number of passengers waiting.

Output:

Platform ID of overcrowded platforms.

```
C:\Users\sanma\OneDrive\Do  X  +  v
Enter the number of platforms: 3
Enter platform ID and crowd density for platform 1: 1
345
Enter platform ID and crowd density for platform 2: 2
678
Enter platform ID and crowd density for platform 3: 3
455

Platforms in AVL Tree (sorted by crowd density):
Platform 1 with crowd density 345
Platform 3 with crowd density 455
Platform 2 with crowd density 678

Detecting and moving overcrowded platforms to priority queue:
Overcrowded Platform ID: 2, Crowd Density: 678

Overcrowded platforms in priority queue (Max Heap):
Platform 2 with crowd density 678

Process returned 0 (0x0)    execution time : 12.968 s
Press any key to continue.
```

Alternate possible algorithms:

1. Red Black Tree

-> Useful for scenarios that require efficient, real-time insertion, deletion, and overcrowd data based on factors like priority, time, density, and urgency.

-> Ideal for systems where quick adjustments are needed to maintain safety and efficiency.

-> Time Complexity : $O(\log n)$

c)Conclusion:

Based on the result, we can detect the highly crowded platforms which require immediate attention and necessary changes in the train schedules are made for better passenger flow.

- The project encouraged us to think critically about how algorithms can be applied to address real-world challenges and design efficient and smart solutions.

- It helped us to enhance our logical thinking and the ability to think algorithmically when approaching new problems when given a real world scenario.

References

Page | 20

- <https://www.tokyometro.jp/en/subwaymap/index.html>
- <https://nammametro.bmrc.co.in/>
- Introduction to the Design and Analysis of Algorithms(3rd Edition) by Michael Levitin
- The research paper on mapping of Tokyo's Rail network using slime mould : <https://www.researchgate.net/publication/>
- [1] Introduction to the Design and Analysis of Algorithms (3rd Edition) by Michael Levitin.
-
- [2] <https://www.networkrailmediacentre.co.uk/news/a-better-experience-for-passengers-at-liverpool-street-station>
-
- [3] <https://www.hindustantimes.com/cities/noida-news/nmrc-launches-cashless-ticket-vending-machines-at-stations-101730839322291.html>
- <https://www.quora.com/How-can-the-service-of-Indian-Railways-practically-be-improved>
- https://youtu.be/o_Vs2NejfgU?si=wUn2t2dsRq4JQjll
- Introduction to the Design and Analysis of Algorithms(3rd Edition) by Michael Levitin
- https://www.reddit.com/r/bangalore/comments/1bo1u7b/bangalore_metro_over_crowding/
- <https://www.london.gov.uk/who-we-are/what-london-assembly-does/questions-mayor/find-an-answer/crowd-management-london-underground/>

Tools used:

- <https://www.geogebra.org/>
- <https://metromapmaker.com/>
- <https://app.diagrams.net/>

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