18CSC204J
Design and Analysis of Algorithm

MINI PROJECT -Minimum platforms needed to avoid delay in the train arrival Using Greedy algo

Faculty Incharge -Revathi M

SEC -G2
Semester - IV

Year – II year

RA2011003011138	DIVYANSH JOSHI
RA2011003011142	KANKANA MAJUMDER
RA2011003011149	BHANU PRATAP SINGH

Problem Statement

Given the arrival and departure times of all trains that reach a railway station, the task is to find the minimum number of platforms required for the railway station so that no train waits.

INTRODUCTION

This is the challenge of finding the minimum platforms needed to avoid delay in the train arrival. We are going to find this using the Greedy Algorithm.

OBJECTIVE

This problem is an algorithmic problem tasked with finding minimum number of platforms required for the railway station so that no train waits.

We can find this using different algorithms , but we will be using the greedy algorithm for the solution of this problem.

GREEDY ALGORITHM :

A greedy algorithm is an algorithmic paradigm that follows the problem-solving heuristic of making the locally optimal choice at each stage with the hope of finding a global optimum.

Solution to the problem:

> If we observe the question, we need to find out how many trains will have an overlapping schedule.It means, if there is a train, Train A, and we need to find if there are any trains arriving before the arrival of a train.

So we need to sort both of the arrays combined. Then we need to check if there are any trains whose departures are after the arrival of a train.

After sorting, we will keep track of the platforms needed.

- Store the arrival time and departure time in array arr and sort this array based on arrival time.
- Declare a priority queue(min-heap) and store the departure time of the first train and also declare a counter cnt and initialize it with 1.
- Iterate over arr from 1 to n-1.
    - check if the arrival time of current train is less than or equals to the departure time of previous train which is kept on top of the priority queue.
        - If true, then push the new departure time and increment the counter cnt
        - otherwise, we pop() the departure time
        - push new departure time in the priority queue

Finally, return the cnt.

Algorithm:

- Sort the arrival and departure times of trains.
- Create two pointers i=0, and j=0, and a variable to store ans and current count plat
- Run a loop while i<n and j<n and compare the ith element of arrival array and jth element of departure array.
- If the arrival time is less than or equal to departure then one more platform is needed so increase the count, i.e., plat++ and increment i
- Else if the arrival time is greater than departure then one less platform is needed to decrease the count, i.e., plat– and increment j

- Update the ans, i.e. ans = max(ans, plat).

CODE

```
#include <algorithm>
#include <iostream>

using namespace std;
int findPlatform(int arr[], int dep[], int n)
{
   sort(arr, arr + n);
   sort(dep, dep + n);

   int plat_needed = 1, result = 1;
   int i = 1, j = 0;

   while (i < n && j < n) {
      if (arr[i] <= dep[j]) {
         plat_needed++;
         i++;
      }
      else if (arr[i] > dep[j]) {
         plat_needed--;
         j++;
      }
      if (plat_needed > result)
         result = plat_needed;
   }

   return result;
}
```

```
// Driver code
int main()
{
   int arr[] = { 900, 940, 950, 1100, 1500, 1800 };
   int dep[] = { 910, 1200, 1120, 1130, 1900, 2000 };
   int n = sizeof(arr) / sizeof(arr[0]);
   cout << findPlatform(arr, dep, n);
   return 0;
}
```

Input/Output:

Input: arr[] = {9:00, 9:40, 9:50, 11:00, 15:00, 18:00}

dep[] = {9:10, 12:00, 11:20, 11:30, 19:00, 20:00}

Output: 3

Explanation: There are at-most three trains at a time (time between 9:40 to 12:00)

Input: arr[] = {9:00, 9:40}

dep[] = {9:10, 12:00}

Output: 1

Explanation: Only one platform is needed.

Time Complexity:

O(N * log N)
{One traversal O(n) of both the array is needed after sorting O(N * log N).}

CONCLUSION :

For solving the minimum number of platforms required we use a greedy approach to find the optimal solution. The current choice is made such that it is the best choice at that moment, without worrying about the future consequences of the said choice. Once made, this choice is never altered.

Though intuitive, it is very hard to prove the correctness of the algorithm.

Greedy algorithm refers to a class of algorithms that address some optimization problem. It does not always lead to the global optimal solution.

A correct greedy algorithm is very fast and efficient as compared to other algorithms, such as dynamic programming.

References

1. geeksforgeeks.org/greedy-algorithms/?ref=shm
2. https://medium.com/techie-delight/top-7-greedy-algorithm-problems-3885feaf9430