

CS263

ASSIGNMENT 7

NAME:

ARCHIT AGRAWAL

ROLL NO. :

202051213

SECTION:

A

1. Let us assume you got the assignments on different courses 1 to C. For each assignment, let d_i denote the number of days required to complete the assignments. For each day of delay before starting your i^{th} assignment, a penalty of p_i is imposed. We are required to find a sequence to complete the assignments so that the overall penalty is minimized. We can only work on one assignment at a time and if you start any assignment finish it then start another. Write an efficient algorithm for the given problem.

Algorithm

- Make a $c * 3$ matrix. In i^{th} row, store the number of days required to complete the i^{th} assignment at index 0, the penalty imposed per day at index 1, and the ratio $\text{penalty}[i]/\text{day}[i]$ at index 2.
- Sort this matrix in ascending order based on column 2, and move the other two columns accordingly.
- Make two variables waitTime and minimum to store the number of days to wait before starting an assignment and the penalty imposed and initialise them with 0.
- Now, we have the matrix sorted according to the ratio of penalty and days. For the minimum penalty, complete the assignments with greater penalty to days ratio first.
- Hence, traverse the matrix from index $c - 1$ to index 0. In i^{th} iteration, update 'minimum' with $(\text{minimum} + \text{waitTime} * (\text{matrix}[i][1]))$ i.e time before the starting of this assignment multiplied with penalty imposed per day for this assignment. And then update, the waitTime with the sum of previous waitTime plus the time taken to complete this assignment.
- Return minimum.

Since, we have a predefined process to reach to our solution, it is a Greedy Approach.

Code

```
import java.util.*;

public class Main{

    public static int minPenalty(int c, int[] daysRequired, int[]
penalty){

        //making a matrix
        double[][] matrix = new double[c][3];
        for(int i = 0; i < c; i++){
            matrix[i][0] = (double)daysRequired[i];
            matrix[i][1] = (double)penalty[i];
            matrix[i][2] = 1.0 * penalty[i]/daysRequired[i];
        }

        //sorting matrix based on penalty to day ratio
        Arrays.sort(matrix, new Comparator<double[]>(){
            //overriding the compare method
            public int compare(double[] a, double[] b){
                return Double.compare(a[2], b[2]);
            }
        });

        int waitTime = 0;
        int minimum = 0;

        //finding the penalty
        for(int i = c - 1; i >= 0; i--){
            minimum += (waitTime * (int) matrix[i][1]);
            waitTime += (int) matrix[i][0];
        }

        return minimum;
    }

    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of assignments : ");
        int c = sc.nextInt();
        System.out.println();

        int[] daysRequired = new int[c];
        System.out.println("Enter the number of days to complete each
assignment (separated by spaces) : ");
        System.out.println();
    }
}
```

```

        for(int i = 0; i < c; i++){
            daysRequired[i] = sc.nextInt();
        }

        int[] penalty = new int[c];
        System.out.println("Enter the penalty for each assignment
(separated by spaces)");
        for(int i = 0; i < c; i++){
            penalty[i] = sc.nextInt();
        }

        System.out.println("Minimum Penalty : " +minPenalty(c,
daysRequired, penalty));
    }
}

```

Output

```

PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { jav
Enter the number of assignments : 3

Enter the number of days to complete each assignment (separated by spaces) :
4 6 7
Enter the penalty for each assignment (separated by spaces)
2 1 4
Minimum Penalty : 25
PS C:\Users\Archit\Desktop\cprog> █

```

The assignment in this case should be done in the following order:

Assign3 -> Assign1 -> Assign2

```

PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { jav
Enter the number of assignments : 3

Enter the number of days to complete each assignment (separated by spaces) :
2 5 7
Enter the penalty for each assignment (separated by spaces)
8 10 3
Minimum Penalty : 41
PS C:\Users\Archit\Desktop\cprog> █

```

The assignment in this case should be done in the following order:

Assign1 -> Assign2 -> Assign3

```

PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) {
Enter the number of assignments : 4

Enter the number of days to complete each assignment (separated by spaces) :
2 1 4 3
Enter the penalty for each assignment (separated by spaces)
5 100 2 1
Minimum Penalty : 18
PS C:\Users\Archit\Desktop\cprog>

```

The assignment in this case should be done in the following order:

Assign2 -> Assign1 -> Assign3 -> Assign4

```

PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { ja
Enter the number of assignments : 4

Enter the number of days to complete each assignment (separated by spaces) :
2 4 3 5
Enter the penalty for each assignment (separated by spaces)
2 4 3 5
Minimum Penalty : 71
PS C:\Users\Archit\Desktop\cprog>

```

The assignment in this case can be done in any order because the penalty to day ratio is equal.

```

PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) {
Enter the number of assignments : 10

Enter the number of days to complete each assignment (separated by spaces) :
2 5 3 1 6 3 2 4 8 8
Enter the penalty for each assignment (separated by spaces)
2 4 1 5 2 5 6 3 2 1
Minimum Penalty : 247
PS C:\Users\Archit\Desktop\cprog>

```

Analysis

The algorithm first creates a matrix from the given data which takes $O(3n) = O(n)$. After that it sorts the matrix based on one index, sorting has a complexity of $O(n \log n)$. After that, we traverse the matrix once again in $O(n)$ time. Therefore, the time complexity is:

$$T(n) = 2 O(n) + O(n \log n)$$

$$T(n) = O(n \log n)$$

2. Event Organiser team has decided to take few classroom on permission from the Government Engineering college to schedule the 1-day Tech event of IIIT Vadodara. They got the start and end timing of all the events, the task is to find the minimum number of classroom required for the events so that no event will wait or get cancelled. We are given two arrays that represent the start and end times of events that need to be conducted.

Note: As no information was given about how the timings are given, they are assumed to be given as (a, b) i.e from hour 'a' to hour 'b'. 'a' being inclusive and 'b' being exclusive.

Algorithm

- Since, it is a 1-day event, make an array 'hours' of size 24 which represents each hour in the day. Initialise each hour with 0.
- Traverse the given start and end timings table and for each event, increment the index at starting hour and decrement the index at ending hour.
- Initialise a variable 'rooms' with hours[0].
- Traverse through the 'hours' array and update hours[i] with hours[i] + hours[i - 1]. Update 'rooms' with maximum of 'rooms' and hours[i].
- Return 'rooms'.

Code

```
import java.util.*;

public class Main{

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the number of events : ");
        System.out.println();
```

```

        int n = sc.nextInt();

        int[][] eventTimings = new int[n][2];

        System.out.println("Enter the start timing and end timing for n
events, in new line for each event..");
        System.out.println("input start time, then end time separated
by space in each line");

        for(int i = 0; i < n; i++){
            eventTimings[i][0] = sc.nextInt();
            eventTimings[i][1] = sc.nextInt();
        }

        System.out.println("Minimum Rooms Required are : "
+minClassrooms(eventTimings));
    }

    public static int minClassrooms(int[][] timings){

        int[] hours = new int[25];
        for(int i = 0; i < timings.length; i++){
            hours[timings[i][0]]++;
            hours[timings[i][1]]--;
        }

        int rooms = hours[0];

        for(int i = 1; i < 25; i++){
            hours[i] += hours[i - 1];
            rooms = Math.max(rooms, hours[i]);
        }

        return rooms;
    }
}

```

Output

```
PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { java -jar c:\Users\Archit\Desktop\cprog\RoomAllocation.jar
Enter the number of events :
5
Enter the start timing and end timing for n events, in new line for each event..
input start time, then end time separated by space in each line
1 5
2 5
3 6
5 6
6 8
Minimum Rooms Required are : 3
PS C:\Users\Archit\Desktop\cprog>
fwd-i-search: _
```

```
PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { java -jar c:\Users\Archit\Desktop\cprog\RoomAllocation.jar
Enter the number of events :
10
Enter the start timing and end timing for n events, in new line for each event..
input start time, then end time separated by space in each line
1 5
2 3
3 5
4 9
10 22
22 23
11 15
12 18
9 10
5 6
Minimum Rooms Required are : 3
PS C:\Users\Archit\Desktop\cprog>
```

```
PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { java -jar c:\Users\Archit\Desktop\cprog\RoomAllocation.jar
Enter the number of events :
3
Enter the start timing and end timing for n events, in new line for each event..
input start time, then end time separated by space in each line
1 2
2 3
3 5
Minimum Rooms Required are : 1
PS C:\Users\Archit\Desktop\cprog>
```

```
PS C:\Users\Archit\Desktop\cprog> cd "c:\Users\Archit\Desktop\cprog\" ; if ($?) { java -jar c:\Users\Archit\Desktop\cprog\RoomAllocation.jar
Enter the number of events :
3
Enter the start timing and end timing for n events, in new line for each event..
input start time, then end time separated by space in each line
1 5
2 5
3 5
Minimum Rooms Required are : 3
PS C:\Users\Archit\Desktop\cprog>
```


Analysis

As we are traversing the timings array once for making the 'hours' array, it takes $O(n)$ time (n being the number of events). After that we are traversing only the 'hours' array which is of 24 size, therefore, it takes $O(1)$ time. Therefore, overall time complexity of the algorithm is:

$$T(n) = O(n) + O(1)$$

$$T(n) = O(n)$$

3. There are N stations on the route of a train Gandhinagar to Jaipur. If there exists any direct train between any two stations i and j , then the ticket cost for all pairs of stations (i, j) is given where j is greater than i . For example: There are Five stations A, B, C, D. A is the source and D is the destination.

Cost of route A-B - Rs100 by Train X,

Cost of route A-C- 200 by Train Y,

Cost of route A-D- 300 by Train Z

Cost of route B-C-No train

Cost of route B-D- 100 by train W

Cost of route C-D- 250 by train V

Find the minimum cost route that will help you to reach the destination at minimum cost.

Algorithm

The idea in below code is to first calculate min cost for station 1, then for station 2, and so on. These costs are stored in an array `price[0...n-1]`.

- 1) The min cost for station 0 is 0, i.e., `price[0] = 0`
- 2) The min cost for station 1 is `cost[0][1]`, i.e., `price[1] = cost[0][1]`
- 3) The min cost for station 2 is minimum of following two.
 - a) `price[0] + cost[0][2]`
 - b) `price[1] + cost[1][2]`
- 3) The min cost for station 3 is minimum of following three.
 - a) `price[0] + cost[0][3]`
 - b) `price[1] + cost[1][3]`

c) price[2] + cost[2][3]

Similarly, price[4], price[5], ... price[n-1] are calculated.

Code

```
import java.util.*;
public class Station {

    static int max = Integer.MAX_VALUE;
    // This function returns the smallest possible cost to
    // reach station n-1 from station 0.
    public static int minCost(int[][] cost, int n)
    {
        // dist[i] stores minimum cost to reach station i
        // from station 0.
        int price[] = new int[n];
        int i, j;
        for (i=0; i<n; i++)
            price[i] = max;
        price[0] = 0;

        // Pass through every station and check if using it
        // as an intermediate station gives better path
        for (i=0; i<n; i++)
            for (j=i+1; j<n; j++)
                if (price[j] > (price[i] + cost[i][j]))
                    price[j] = price[i] + cost[i][j];

        return price[n-1];
    }

    public static void main(String[] args)
    {
        Scanner Sc = new Scanner(System.in);
        //System.out.println("Enter the number of stations");
        int n = 3;
        int i,j;
        int cost[][] = { {0, 15, 30, 56, 87},
                        {max, 0, 40, 49, 53},
                        {max, max, 0, 22, 54},
                        {max, max, max, 0, 23}
                      };
        System.out.println("The Minimum cost to reach station "+ n+ "
is "+minCost(cost, n));
    }
}
```

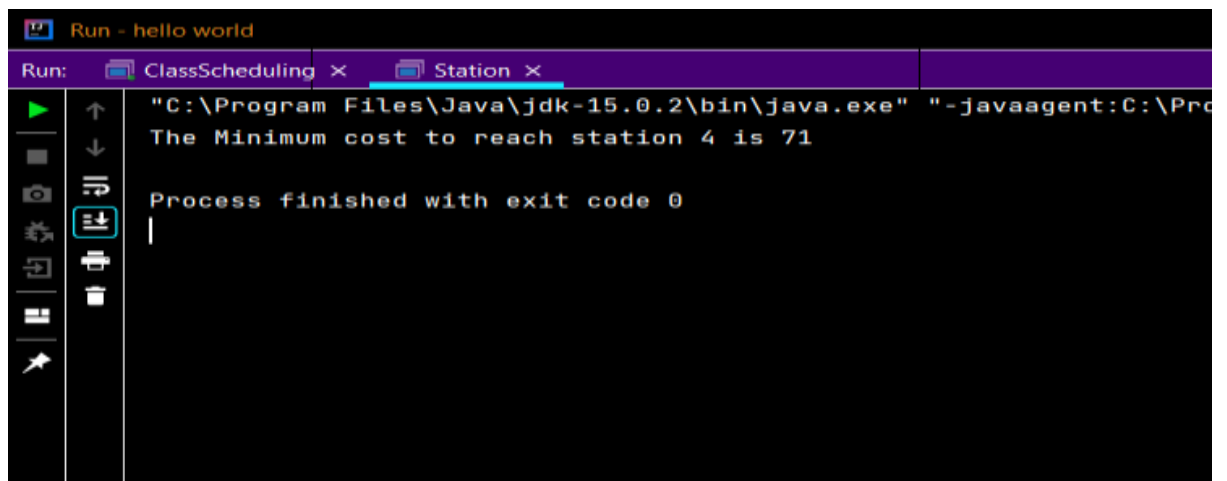
Output

```
int cost[][] = { {0, 15, 80, 90},
                 {max, 0, 40, 50},
                 {max, max, 0, 70},
                 {max, max, max, 0}
};
```



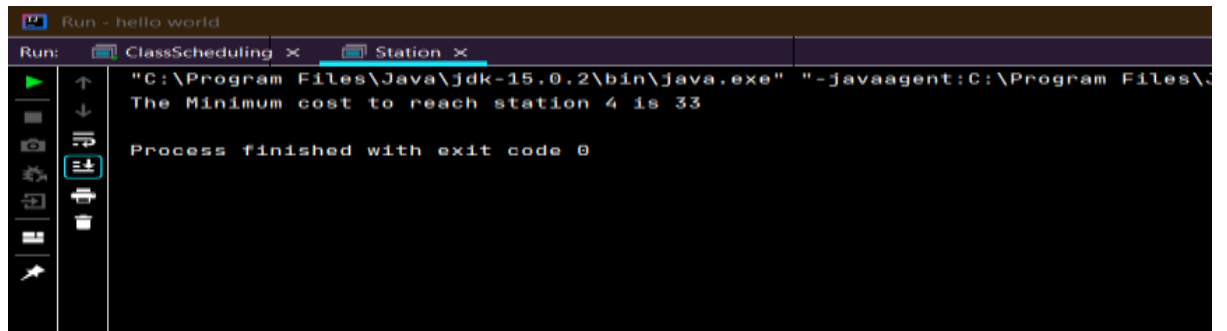
```
Run - hello world
Run: ClassScheduling x Station x
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ
The Minimum cost to reach station 4 is 65
Process finished with exit code 0
```

```
int cost[][] = { {0, 15, 80, 85},
                 {max, 0, 40, 56},
                 {max, max, 0, 33},
                 {max, max, max, 11}
};
```



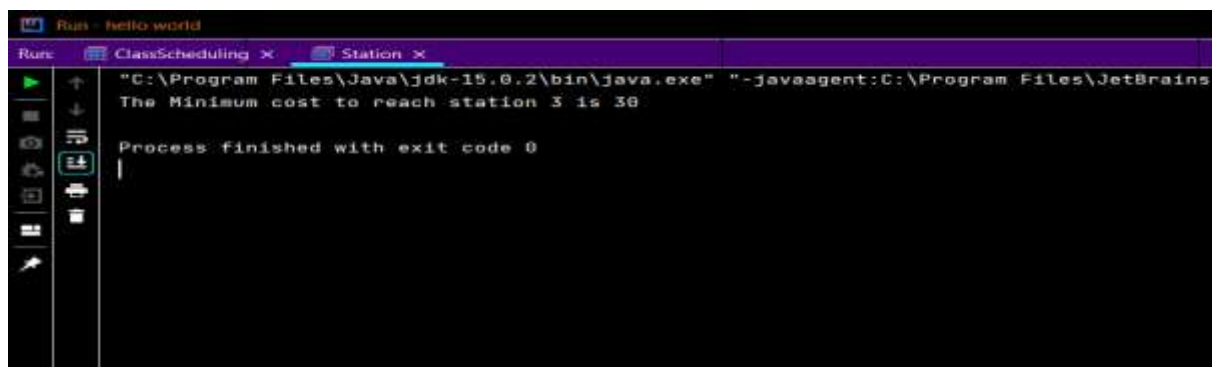
```
Run - hello world
Run: ClassScheduling x Station x
"C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Pro
The Minimum cost to reach station 4 is 71
Process finished with exit code 0
|
```

```
int cost[][] = { {0, 15, 30, 33},
                 {max, 0, 40, 54},
                 {max, max, 0, 37},
                 {max, max, max, 15}
};
```



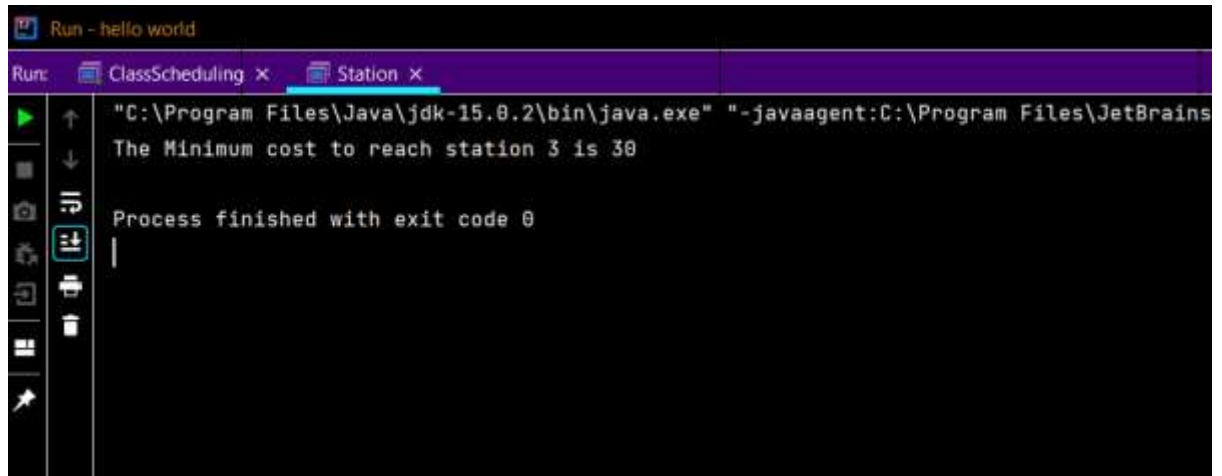
```
Run: "C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\
The Minimum cost to reach station 4 is 33
Process finished with exit code 0
```

```
int cost[][] = { {0, 15, 30},
                 {max, 0, 40},
                 {max, max, 0},
};
```



```
Run: "C:\Program Files\Java\jdk-15.0.2\bin\java.exe" "-javaagent:C:\Program Files\JetBrains
The Minimum cost to reach station 3 is 30
Process finished with exit code 0
```

```
int cost[][] = { {0, 15, 30, 56, 87},  
                 {max, 0, 40, 49, 53},  
                 {max, max, 0, 22, 54},  
                 {max, max, max, 0, 23}  
};
```



The screenshot shows a Java IDE's run console. At the top, it says "Run - hello world". Below that, the command executed is "C:\Program Files\Java\jdk-15.0.2\bin\java.exe" with various JVM arguments. The output of the program is "The Minimum cost to reach station 3 is 30". At the bottom, it states "Process finished with exit code 0".

Analysis

The extra space required in this solution would be $O(n)$ as we use an array of size n to store the answers of subproblems.

Time Complexity : $O(n^2)$

Space Complexity : $O(n)$