OPTIMIZED TAXI PRE-BOOKING SYSTEM

SLOT - C2 || GROUP - G12

Guided by

Prof Santanu Mandal

TEAM MEMBERS

18BCD7165	MADDINENI AASHIKA
18BCN7079	DEVI JAGANNADH KOTHA
18BCN7057	PADUCHURU ABHISHEK
18BCD7009	RISHABH SINGH BAIS
18BCE7236	SUNKARA YASWANTH REDDY

INTRODUCTION

Taxi is an important transportation mode between public and private transportations, delivering millions of passengers to different locations in urban areas. However, taxi demands are usually higher than the number of taxis in peak hours of major cities, resulting in many people spending a long time on roadsides before getting a taxi. Increasing the number of taxis seems an obvious solution .But it brings some negative effects, e.g., causing additional traffic on the road surface and more energy consumption, and decreasing taxi drivers income (considering that demands of taxis would be lower than number of taxis during off-peak hours). In this paper, A Taxi is a type of vehicle for hire with a driver, used by a single Customer or small group of Customers often for a non-shared ride. A taxi conveys Customers between locations of their choice. The service will run at real-time as or at a time designated by the client. This Taxi & Cab Management System is helpful for travel agencies for their cabs, taxis, and their vehicle maintenance.

OBJECTIVE

The main objective of this project is making more rides by less taxis by using taxi scheduling. Taxis play an important role in modern public transportation. In this project we are going to propose a new technique which is more economical for customers and very environmentally friendly called taxi scheduling. This project will help in getting more profits for the company.

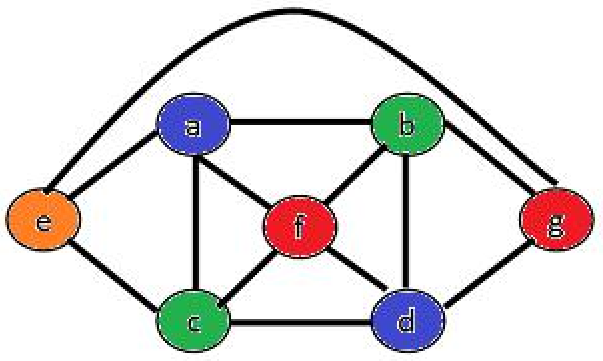
METHODOLOGY

In this mini-project we have used graph coloring concept of discrete mathematics and greedy algorithm to improvise the existing system of taxi pre-booking. We have implemented this using java programming.

GRAPH COLORING CONCEPT

Graph colouring definition :-

1. Graph coloring is nothing but a simple way of labelling graph components such as vertices, edges, and regions under some constraints. In a graph, no two adjacent vertices, adjacent edges, or adjacent regions are colored with minimum number of colors. This number is called the chromatic number and the graph is called a properly colored graph.
2. While graph coloring, the constraints that are set on the graph are colors, order of coloring, the way of assigning color, etc. A coloring is given to a vertex or a particular region. Thus, the vertices or regions having the same colors form independent sets.



Vertex 1: Symbol a
Vertex 2: Symbol b
Vertex 3: Symbol c
Vertex 4: Symbol d
Vertex 5: Symbol e
Vertex 6: Symbol f
Vertex 7: Symbol g

Use of graph colouring :-

1. Graphs are often used to model various real-world problems (and sometimes also made-up problems, because mathematicians and computer scientists still have to keep publishing papers after all), and we now have many algorithms that work on them efficiently (in terms of complexity), finding various attributes that may be interesting in order to solve the aforesaid problems.
2. Alternatively, we can sometimes prove that a problem can not be solved with any efficient algorithm. Graph coloring is one of these (or more accurately, the questions: can a graph be colored in up to k colors, or the question what is the minimal number of colors needed to color the graph), unless we're dealing with certain subtypes of graphs, such as planar graphs (an map of neighboring countries is a good example as it was used for some interesting graph coloring

proofs). Coloring here means attaching a "color" or a number to each vertice such that no two vertices with a connecting edge have the save value.

Approaches to apply graph colouring:-

1. Graph Colouring Algorithm Graph Colouring Algorithm There is no efficient algorithm available for coloring a graph with minimum number of colours
2. Graph coloring problem is a known NPGraph coloring problem .
3. NP Complete ProblemNP Complete Problems are problems whose status is unknown.
4. No polynomial time algorithm has yet been discovered for any NP complete problem
5. It is not established that no polynomial-polynomial- time algorithm exists for any of them.
6. NP Complete Problem The interesting part is, if any one of the NP complete problems can be solved in polynomial time, then all of them can be polynomial time, then all of them can be solved.
7. Although Graph coloring problem is NP Complete problem there are some approximate algorithms to solve the graph coloring problem.
8. Some algorithms are :- Basic Greedy Algorithm, Welsh Power Algorithm etc.

Applications of graph colouring:-

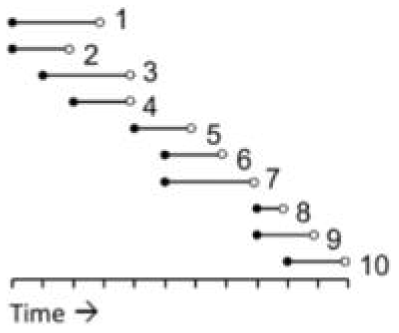
Graph coloring is one of the most important concepts in graph theory. It is used in many real-time applications of computer science such as −

1. Clustering
2. Data mining
3. Image capturing
4. Image segmentation
5. Networking
6. Resource allocation
7. Processes scheduling

THEORETICAL CONCEPT

Let's take an example on taxi scheduling:-

No. of rides



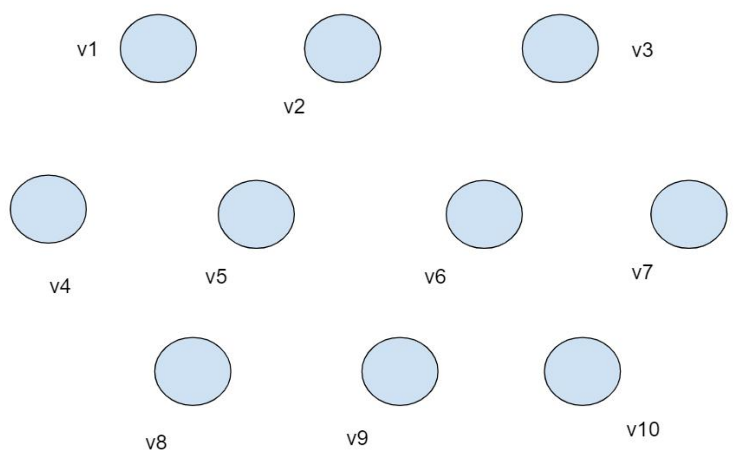
The above figure(a) shows an example problem where we have ten taxi bookings. For illustrative purposes these have been ordered from top to bottom according to their start times. It can be seen, for example, that bookings 1 overlaps with bookings 2, 3 and 4 hence any taxi carrying out booking 1 will not be able to serve bookings 2,3 and 4.

We can construct a graph from this information by using one vertex for booking and then adding edges between any vertex pair corresponding to overlapping bookings.

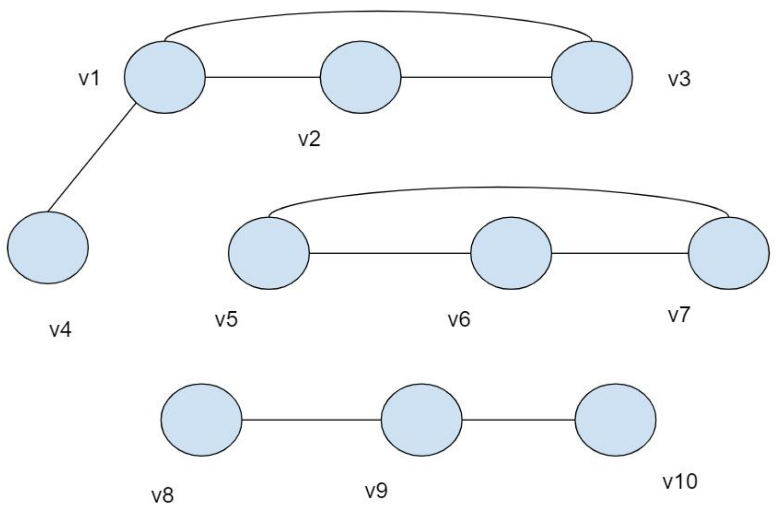
Graph coloring:-

Graph coloring and Scheduling

8. Convert problem into a graph coloring problem
9. Bookings are represented by vertices
10. Two vertices are connected by an edge if the corresponding Bookings have overlapping Bookings.



Connect the vertices according to overlapping of booking.



The Greedy Algorithm For Coloring Vertices:-

The algorithm is called greedy because it is a rather short-sighted way of trying to make a proper coloring with as few colors as possible. It does not always succeed in finding the minimum 2 number (the chromatic number), but at least provides some proper coloring. The procedure requires us to number consecutively the colors that we use, so each time we introduce a new color, we number it also. Here is the procedure.

Note: -

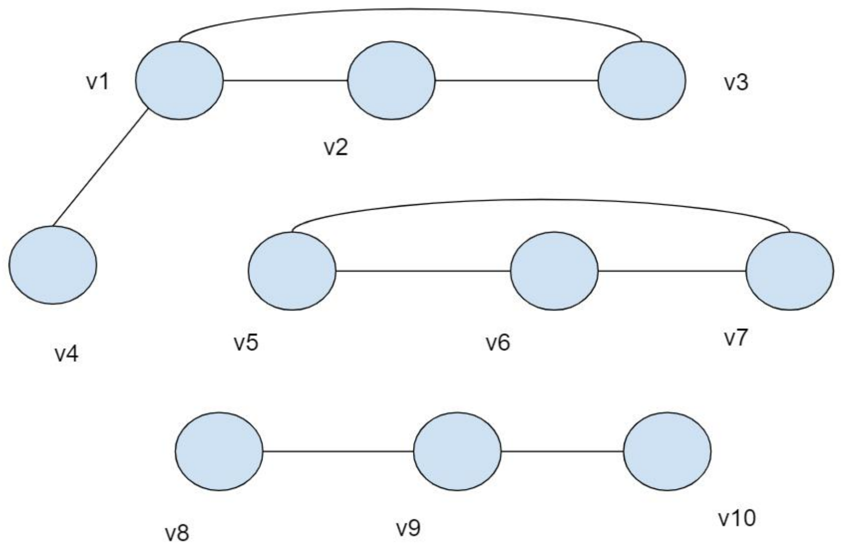
1. Color a vertex with color 1.

2. Pick an uncolored vertex v. Color it with the lowest-numbered color that has not been used on any previously colored vertices adjacent to v. (If all previously-used colors appear on vertices adjacent to v, this means that we must introduce a new color and number it.)

3. Repeat the previous step until all vertices are colored.

Clearly, this produces a proper coloring, since we are careful to avoid conflicts each time we color a new vertex. How many colors will be used? It is hard to say in advance, and it depends on what order we choose to color the vertices

Let's color the below graph using a greedy algorithm.



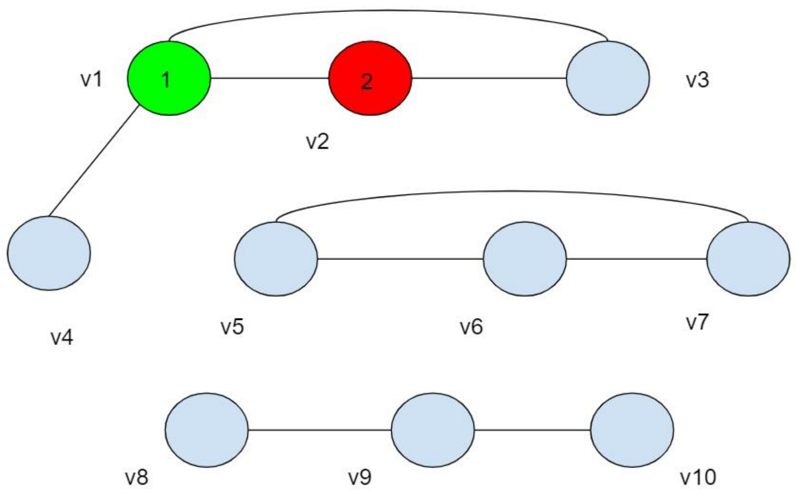
Clearly the graph is obviously not 1-colorable because there exist edges.

So try it with 2-coloring by using color 1 (green) and color 2 (red) .

Colors using :

1. Green
2. Red

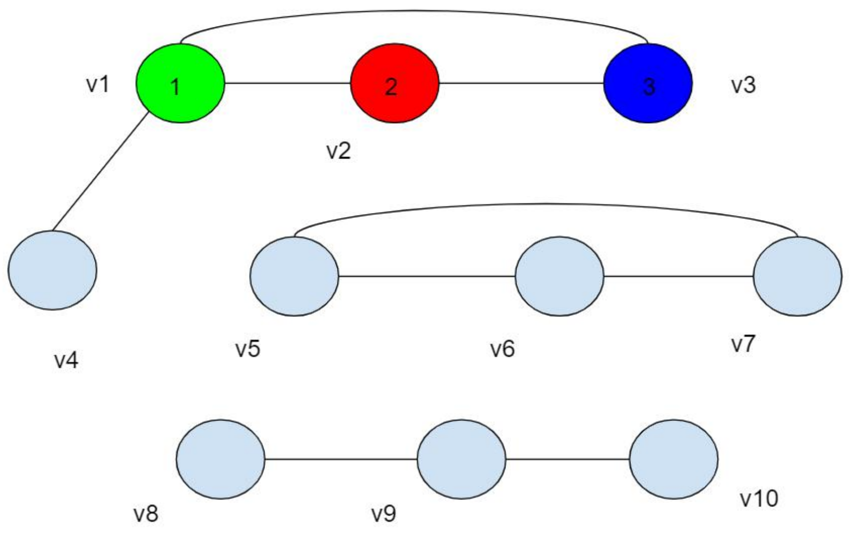
Now, color vertex v1 with color 1(green) and v2 with color 2(red)



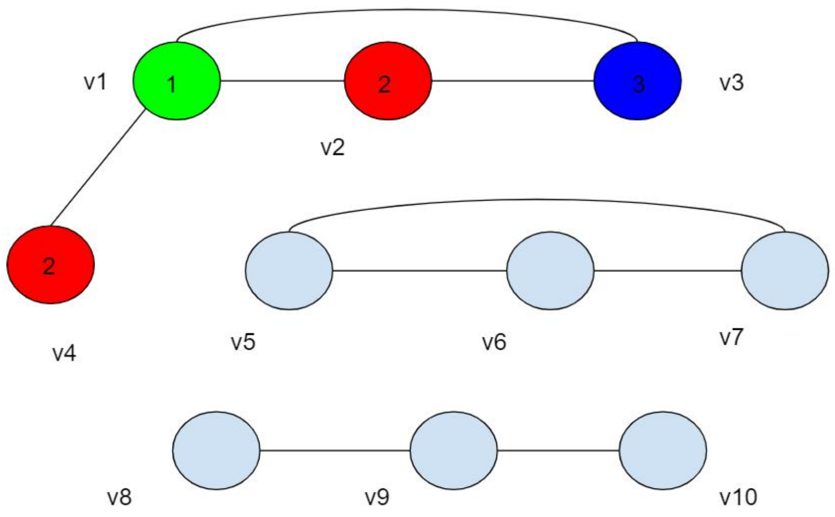
In the above figure we see that v3 is connected with both v1 and v2, we can't give color 1(green) and color 2(red) colors. So we introduce a new color, color 3(Blue) to color v3 and the graph results in 3-coloring.

Updated Colors using :

1. Green
2. Red
3. Blue



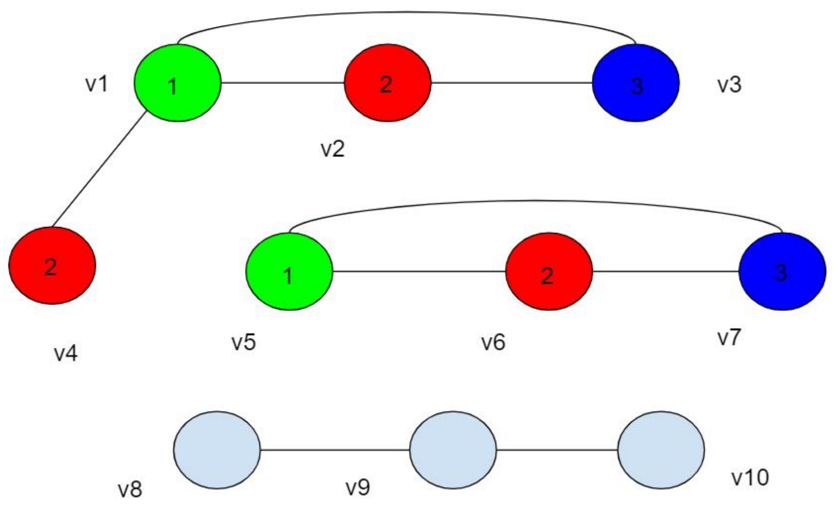
Next consider vertex v4 as it is connected with v1 it prevents the usage of color 1(green) so we choose Red(2) because we always need to check from lowest numbered color.



Similarly, we start again with vertex v5 color with color 1(green).

Vertex v6 can't be color 1(green)so we give next color, color 2(red).

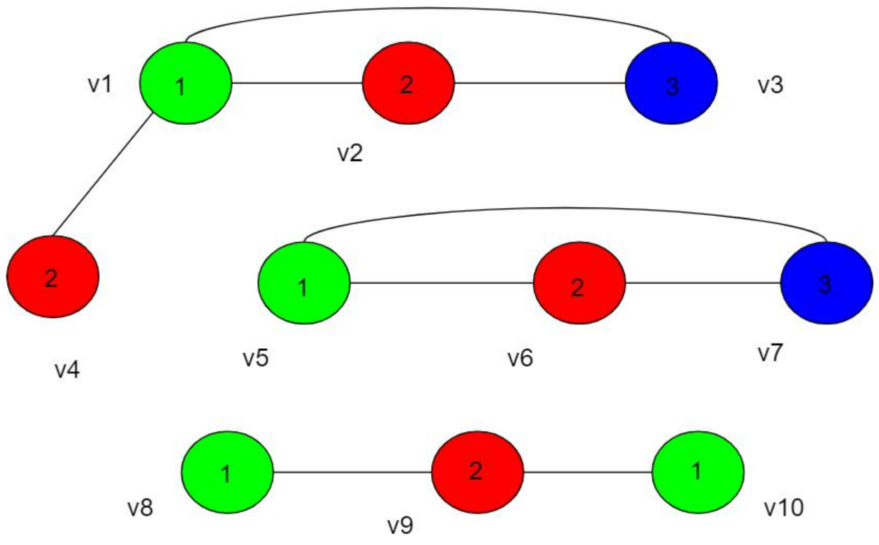
Vertex v7 can't be color 1(green) and color 2(red) so we give it color 3(Blue).



Consider vertex v8 and color it with color 1(green).

Vertex v9 can't be color 1(green) so we give next color, color 2(red).

Vertex v10 can't be color 2(red) so we go to the lowest number color and give it Green.



We see that the quality of our coloring (with fewer colors being considered better) from the Greedy Coloring Algorithm is dependent on the order in which we color the vertices.

Chromatic Number:-

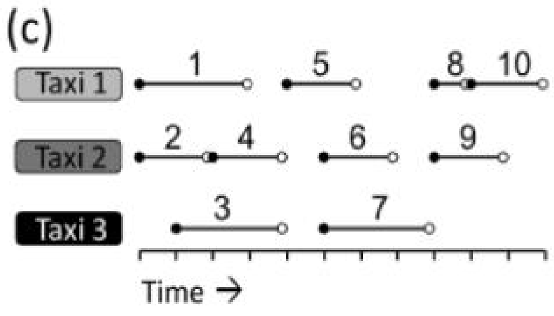
The chromatic number of a graph is the minimum number of colors in a proper coloring of that graph.

There is a certain minimum quality we get, which we can determine by the following theoretical argument: Suppose that d is the largest degree of any vertex in our graph, i.e., all vertices have d or fewer edges attached, and at least one vertex has precisely d edges attached. As we go about coloring, when we color any particular vertex v, it is attached to at most d other vertices, of which some may already be colored. Then there are at most d colors that we must avoid using. We use the lowest-numbered color not prohibited. That means that we use some color numbered d + 1 or lower, because at least one of the colors 1, 2,...., d + 1 is NOT prohibited. So we never need to use any color numbered higher than d + 1. This gives us the following theorem:

Greedy Coloring Theorem:-

If d is the largest of the degrees of the vertices in a graph G, then G has a proper coloring with d + 1 or fewer colors, i.e., the chromatic number of G is at most d + 1.

In this particular case we see that our example problem has resulted in a graph made of three smaller graphs(components), comprising vertices v1 to v4, v5 to v7 and v8 to v10 respectively. However, this will not always be the case and will depend on the nature of the bookings received.



A graph constructed from time-dependent tasks such as this is usually referred to as an interval graph. There are many simple inexpensive algorithms that exist for interval graphs that will produce an optimal solution (that is a solution using the fewest number of colors possible).

JAVA IMPLEMENTATION

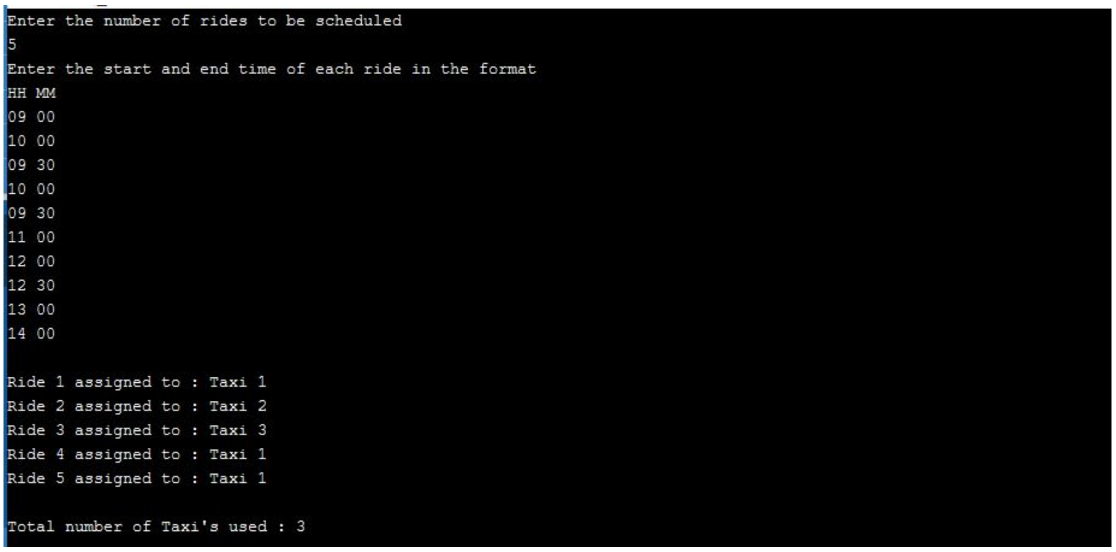
```
import java.util.*;
import java.io.*;
public class K_Coloring {
static ArrayList<Integer> adj_lst[];     //To store the edges for each vertex of the graph
static boolean vis[];
//To maintain the list of currently already visited vertices while DFS to a avoid cycles
static int color[];          //To store the color of each vertex
public static void main(String args[]){
Scanner input=new Scanner(System.in);
//Start and End time of rides
System.out.println("Enter the number of rides to be scheduled");
int n=input.nextInt();                //No. of rides
float strt[]=new float[n];            //Start time if ride
float end[]=new float[n];              //End time of ride
color=new int[n];
vis=new boolean[n];
System.out.println("Enter the start and end time of each ride in the format\nHH
MM");
for(int i=0;i<n;i++) {
float start_hh=input.nextInt();
float strt_mm=input.nextInt();
strt[i]=start_hh+(strt_mm/60);
float end_hh=input.nextInt();
float end_mm=input.nextInt();
end[i]=end_hh+(end_mm/60);
}
create_graph(n,strt,end);     //To create the graph using the ride times
                              //To perform DFS for all vertices[in case of disconnected graph]
for(int i=0;i<n;i++) {
if(!vis[i]) {
DFS(i);
}
```

```
}
System.out.println();
int total_taxi_used=Integer.MIN_VALUE;
                                        //To find the total number of taxis used[1 indexed]
for(int i=0;i<n;i++) {
System.out.println("Ride "+(i+1)+" assigned to : Taxi "+color[i]);
total_taxi_used=Math.max( total_taxi_used,color[i]);
}
System.out.println("\nTotal number of Taxis used : "+ total_taxi_used);
}

                                                            //Depth First Search
public static void DFS(int root) {
System.out.println(root);
vis[root]=true;
boolean used_color[]=new boolean[color.length+1];
                        //To find the color used in the adjacent vertices of the current vertex
for(int i=0;i<adj_lst[root].size();i++) {
used_color[color[adj_lst[root].get(i)]]=true;
}
int least_number_unused_color=-1;
                           //To find the least number color we can use[currently unused]
for(int i=1;i<used_color.length;i++) {
if(!used_color[i]) {
least_number_unused_color=i;
break;
}
}
color[root]=least_number_unused_color;
                                           //Assigning that color the the current vertex
for(int i=0;i<adj_lst[root].size();i++) {
if(!vis[adj_lst[root].get(i)]) {
DFS(adj_lst[root].get(i));
}
```

```
}
}
public static void create_graph(int n,float strt[],float end[]) {
                                        //Initializing each ArrayList[representing each vertex]
adj_lst=new ArrayList[n];
for(int i=0;i<n;i++) {
adj_lst[i]=new ArrayList<>();
}
                 //To find overlapping rides and creating a edge between those vertices
for(int i=0;i<n;i++) {
for(int j=i+1;j<n;j++) {
if(!(end[i]<=strt[j] || strt[i]>=end[j])) {
adj_lst[i].add(j);
adj_lst[j].add(i);
}
}
}
}
}
```

OUTPUT



```
Enter the number of rides to be scheduled
5
Enter the start and end time of each ride in the format
HH MM
09 00
10 00
09 30
10 00
09 30
11 00
12 00
12 30
13 00
14 00

Ride 1 assigned to : Taxi 1
Ride 2 assigned to : Taxi 2
Ride 3 assigned to : Taxi 3
Ride 4 assigned to : Taxi 1
Ride 5 assigned to : Taxi 1

Total number of Taxi's used : 3
```

CONCLUSION

We can conclude from our project that it will increase efficiency of our existing system, which will decrease company's cost and result in less use of taxis and result in less environmental pollution. More economical, very environmentally friendly.

REFERENCES

1. https://en.wikipedia.org/wiki/Graph_coloring#References
2. https://webdocs.cs.ualberta.ca/~joe/Coloring/index.html
3. https://www.researchgate.net/publication/260026333_A_novel_approach_to_inde endent_taxi_scheduling_problem_based_on_stable_matching
4. https://www.researchgate.net/publication/234163991_Multi-agent_real_time_sche duling_system_for_taxi_companies