EFFICIENT SCHEDULING OF MATCHES IN DOUBLE ROUND ROBIN TOURNAMENT USING DISCRETE MATHEMATICS

DISCRETE MATHEMATICS PROJECT

MAT1003

SLOT D+TD

PROF. SHAH PARTH

ANUJ GUPTA 20BCE7055

AARSH PATEL 20BCN7014

HEMANTH KATIKALA MUNIRAJ 20BCR7047

JATIN MALLARPU 20BCI7071

MANSING GHATAGE 20BCE7085

ABSTRACT

Discrete mathematics has many applications which can be used in our day-to-day life. We can use discrete mathematics to predict, how to design a game. One among them is Efficient Scheduling of Matches in Double Round Robin Tournament. In a tournament, assume that we have eight teams and each team plays with each other team twice. That concludes as fourteen matches. The winner is decided based on all matches. The considerable drawback of Double Round Robin is that the time duration of all the matches is too long when compared to the other tournaments that are held and hence leads to higher expenses of conducting the matches. In order to prevent those problems, we can use Discrete Mathematics. For this problem we can use different code languages which helps us in scheduling the matches in short period of time and also helps us manage the travelling salesman problems too.

INTRODUCTION

Sports League are big businesses around the world. One key to such an income level is the schedule team plays. No right holder wants to pay large sums only to get unattractive teams playing on a prime date. Teams do not want to see their large investment in players and infrastructure undermined by poor scheduling. Fans too wont be happy to see a poorly organised team. That too, the schedule should be more optimised in such a way that it is more exciting for fans so that these businesses earn a higher amount of profit.

Fans, who ultimately provide the income for the leagues, are also greatly affected by the schedules. In addition, travels become an important issue for the team. Teams are also concerned with the more traditional issues with regards to their home and away patterns. No teams like to be away from their home for a long period of time. The team members should also practice in order to perform well in the

upcoming matches. If the travel is reduced, they would less likely get tired so that they would have plenty of time to practice and relax for the game and also for planning their strategies.

From above, we can clearly see that the major problem is to schedule the matches in order to reduce travels for teams. This problem can be solved by the combination of Double round robin tournament and Travelling salesman problem.

OBJECTIVES

Here as an example, we have taken Pro Kabaddi League, as is going to be held in the upcoming days.

- There are many constraints such that every team plays with others twice, one on their own home ground and another game on the opponent’s home ground.
- As we all know that of double round robin in Pro Kabaddi League suffers from being too long to complete and hence required a lot of days to get over resulting in higher expenses of travelling, staying etc and hence we are trying to balance travel distance and breaks, while satisfying more requirements and making profit for the organization by higher attendance and TV viewership, lower costs and increased fairness.

We need to find an optimised solution so that the fans as well the organisers both are satisfied.

IMPLEMENTATION

1. SCHEDULING MATCHES

For scheduling matches for our tournaments we use Double Round Robin method. Generally, a Round Robin tournament is a tournament where every team plays with every other team exactly once. In each game a team is played with another team head-to-head and earns a point. At the end of the tournament, the overall winner is the team with the greatest number of points. Whereas, Double Round Robin is similar to that of Round Robin but every team plays with every other team twice instead of once. This is a convenient option if the organiser wants to provide the team with a choice of arena for the matches since each team can make an arena choice.

The general formula for the number of matches played by a team in Double Round Robin are (n teams):

Each team plays: 2*(n-1)

Total number of matches: n*(n-1)

CODE

```
import java.util.*;
public class Main{
public void main(String[] args) {
    //obtain the number of teams from user input
 Scanner input = new Scanner(System.in);
 System.out.print(“How many teams should the fixture table have?”);
 int teams = input.nextInt();
 // Generate the schedule using round robin algorithm.
 int totalRounds = (teams – 1) * 2;
 int matchesPerRound = teams / 2;
 String[][] rounds = new String[totalRounds][matchesPerRound];
 for (int round = 0; round < totalRounds; round++) {
    for (int match = 0; match < matchesPerRound; match++) {
        int home = (round + match) % (teams – 1);
        int away = (teams – 1 – match + round) % (teams – 1);
        // Last team stays in the same place
 // while the others rotate around it.
 if (match == 0) {
    away = teams – 1;
 }
```

```
 // Add one so teams are number 1 to teams
 // not 0 to teams – 1 upon display.
 Rounds[round][match] = (“team “ + (home + 1) + “ plays against team “
+ (away + 1));
 }
 }
// Display the rounds
 for (int I = 0; I < rounds.length; i++) {
     System.out.println(“Round “ + (I + 1));
     System.out.println(Arrays.asList(rounds[i]));
     System.out.println();
     }
}
}
```

OUTPUT

```
How many teams should the fixture table have?4
Round 1
[team 1 plays against team 4, team 2 plays against team 3]

Round 2
[team 2 plays against team 4, team 3 plays against team 1]

Round 3
[team 3 plays against team 4, team 1 plays against team 2]

Round 4
[team 1 plays against team 4, team 2 plays against team 3]

Round 5
[team 2 plays against team 4, team 3 plays against team 1]

Round 6
[team 3 plays against team 4, team 1 plays against team 2]
```

2. MINIMIZING TRAVEL DISTANCE

The minimization of travel distance is important as the teams should travel from one away game to the next without returning home. If we solve this problem then we can obtain huge savings when long trips are applied and teams located close together are visited on the same trip.

So we are going to solve this in such a manner that we will be displaying the closes way of travelling which in return minimises the travelling distance. As the distance travelled is minimised the cost and expense will also be reduced.

CODE:

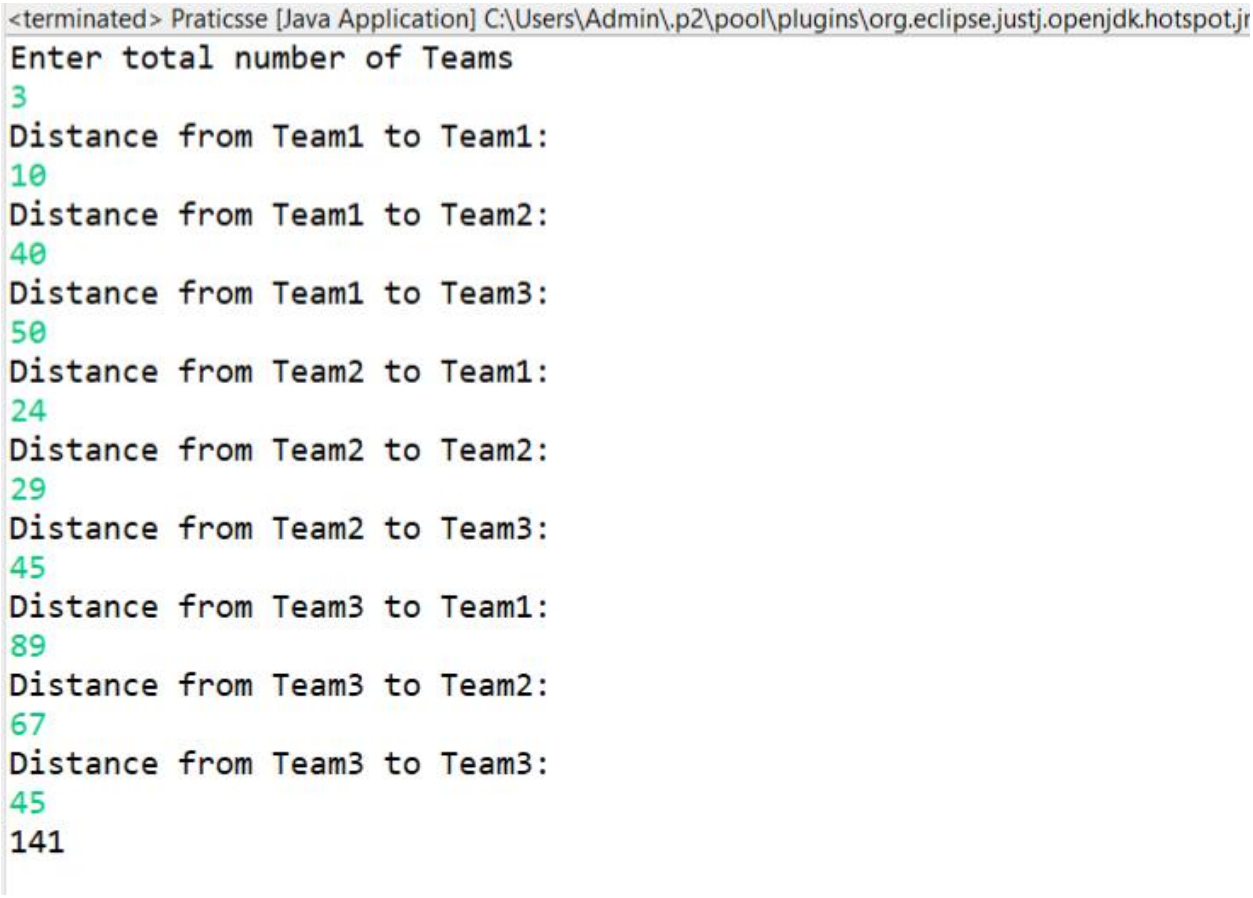
```
import java.util.ArrayList;
import Java.util.*;
   import java.io.*;
   import java.util.Scanner;

   // create TSPExample class to implement TSP code in Java
   public class Praticsse {
      // create findHamiltonianCycle() method to get minimum weighted
cycle
      static int findHamiltonianCycle(int[][] distance, boolean[] visitCity,
int currPos, int cities, int count, int cost, int hamiltonianCycle)
      {

         if (count == cities && distance[currPos][0] > 0)
```

```
        {
            hamiltonianCycle = Math.min(hamiltonianCycle, cost +
distance[currPos][0]);
            return hamiltonianCycle;
        }

        // BACKTRACKING STEP
        for (int i = 0; i < cities; i++)
        {
            if (visitCity[i] == false && distance[currPos][i] > 0)
            {

                // Mark as visited
                visitCity[i] = true;
                hamiltonianCycle = findHamiltonianCycle(distance, visitCity,
i, cities, count + 1, cost + distance[currPos][i], hamiltonianCycle);

                // Mark ith node as unvisited
                visitCity[i] = false;
            }
        }
        return hamiltonianCycle;
    }

    // main() method start
    public static void main(String[] args)
    {
        int cities;

        //create scanner class object to get input from user
```

```
        Scanner sc = new Scanner(System.in);

        // get total number of cities from the user
        System.out.println("Enter total number of Teams ");
        cities = sc.nextInt();



        //get distance of cities from the user
        int distance[][] = new int[cities][cities];
        for( int i = 0; i < cities; i++){
           for( int j = 0; j < cities; j++){
              System.out.println("Distance from Team"+ (i+1) +" to
Team"+ (j+1) +": ");
              distance[i][j] = sc.nextInt();
           }
        }

        // create an array of type boolean to check if a node has been
visited or not
        boolean[] visitCity = new boolean[cities];

        // by default, we make the first city visited
        visitCity[0] = true;



        int hamiltonianCycle = Integer.MAX_VALUE;

        // call findHamiltonianCycle() method that returns the minimum
weight Hamiltonian Cycle
        hamiltonianCycle = findHamiltonianCycle(distance, visitCity, 0,
```

```
cities, 1, 0, hamiltonianCycle);

        // print the minimum weighted Hamiltonian Cycle
        System.out.println(hamiltonianCycle);
    }
}
```

OUTPUT:



REFERENCE

http://www.oxfordcroquet.com/manage/doubleroundrobin/index.asp

https://en.m.wikipedia.org/wiki/Travelling_salesman_problem