Department of Computer Engineering

BLG 336E
Analysis of Algorithms II
Project 3 Report

Spring 2019

Muhammed Raşit EROL

150150023

1 INTRODUCTION

In this project, it is required to implement flow algorithm in order to solve the soccer league problem. The detailed league constrains are given in the project pdf. In order to solve this soccer league problem, the maximum flow of a network approach is used in this project; moreover, Ford Fulkerson Algorithm is used when implementing the given problem. The detailed implementation of project is given following parts.

2 A SIMPLER ELIMINATION RULE

Let the current team with the highest points in the league T, and its points P. Instead of using a maximum-flow based method, we could not just use a simple criterion such that "If current points of a team plus the number of remaining matches a team has >= P, that team still has a chance to win the league". Because even a team has points PT >= P, current team with P points still can win other matches with other teams and consequently mentioned team may not win the league even if its points PT >= P. Example scenario is given below:

Input file:	Output file:
4 6 1 1 1 0 2 7 5 2 0 0 3 7 0 0 7 5 3 7 0	1 0 1 1

As in shown in this example, second team has potentially 6 points which equals to team one with point 6. In the simpler elimination rule, second team has chance to win; however, output of this league T is given above as well and second team has not chance to win the league T. Hence, simpler elimination rule cannot be applied in this problem.

3 IMPLEMENTATION

In this project, it is used three class which are Edge, Node and Graph. With that classes graph representation of the maximum flow of network problem is handled and Ford Fulkerson Algorithm is applied on that graph.

In the program workflow, firstly input file is read and it is stored in a variable in the Graph class. After that with *play_matches()* function, for each team graph is created with residual property and Ford Fulkerson Algorithm is executed.

The graph creation is performed for each team as mentioned above. In each iteration among each team, all matches except matches which chosen team not involved are put the graph. After that, all team which match each other is edged to that nodes put before. Also, graphs have s and t nodes as in the network flow problem.

In these graphs, all edges between nodes are bidirectional. Because, in the Ford Fulkerson Algorithm undo operations are required. Shorty, all edge connections are made with *create_graph()* and *create_residual_graph()* functions. Also, in my implementation, firstly all matches between teams are placed to graph as a node and after *create_residual_graph()* function only needed nodes are kept as active for Ford Fulkerson Algorithm.

In the *ford_fulkerson()* function, Ford Fulkerson Algorithm is implemented. In that algorithm, firstly, it is needed to find a path from s node to t node. This job is handled with *dfs()* function in this project. If *dfs()* function returns available path, than a flow value which is found with determined path, is flowed on the returned path. These operations are handled with *augmenting_path()* function. The *dfs()* function is executed until there is no path between s and t nodes. After that, all edges are checked in order to check criteria given formulation of the problem part below. This routine is performed for all teams.

The code compiled at ITU SSH server successfully like that:

g++ *.cpp –o algo -std=c++11

./algo input1.txt *or* ./algo input1.txt output.txt

4 FORMULATION OF THE PROBLEM

The soccer league problem can be solved like as network flow problems. In order to solve this problem, Ford Fulkerson Algorithm is used. The Ford Fulkerson Algorithm is executed on the graphs and for that reason, graph structure is used in this project.

In the graph, s and t nodes are placed two very ends. There are 4 level with s and t nodes. In the second level, there are nodes which represent matches and in the level 3, there are each team nodes. All nodes are connected each other with bidirectional edges. If we look for first team win chance, all matches and team 1 node are deactivated in this implementation.

The edges between s node and level 2 nodes have capacity as number of matches of corresponding teams. These values are given with input file. The edges between level 2 and level 3 have capacity as number of matches of corresponding teams same as capacity values of edges between s node and level 2. Also, in the level 3 each team has edge with level 2 nodes for each match (see Figure 1). Finally, the edges between level 2 and t node have capacity as a team point which is checking win chance mines current point of the edge which is coming from level 2. Figure 1 explains graph structure more clearly.

For example, we can construct graph as like in Figure 1 for given input below:

Input file:

```
4
2 3 1 1
0 4 0 1
4 0 6 3
0 6 0 3
1 3 3 0
```

When we look for team zero (counting starts from zero for teams) for win chance, graph can be constructed like in the Figure 1. Edge values between level 1 and level 2 are all matches of each team (can be seen in the input matrix). The edge values between level 2 and level 3 are match numbers of each team (this also comes from input matrix). The edge values between level 3 and level 4 are found team zero potential point (current point + remaining maximum point which can get) 7 – current point of each team. From these formulation, for team 1, edge value can be found like that 7 – 3 = 4.

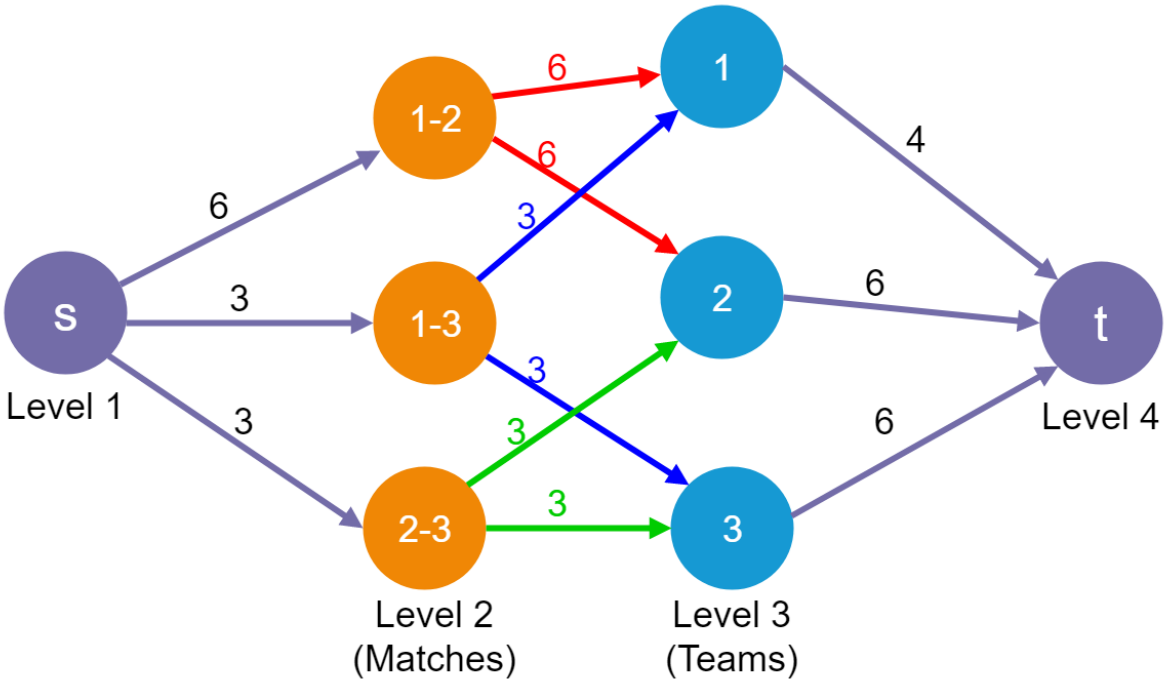

Figure 1: Graph structure of the implementation

In the implementation, after creating these graph, with *create_residual_graph()* function, all edges converted bidirectional edges for undo operation of Ford Fulkerson Algorithm.

The description of method with this implementation in order to find a team has chance to win, is explained like that:

- After Ford Fulkerson Algorithm flow values of edges between level 1 and level 2 are calculated
- If all capacities are full, then it can be said that team which is looking for win chance, can win the league; because, other teams are played their games and still cannot exceed the looking team point. Meanly, maximum flow is flowed in the network and still there is no chance to other teams exceed the looking team.
- If at least one edge capacity is zero, then looking team has not chance to win. If all edge capacities are full, then looking team has chance to win. Because, all matches are played; however, still other teams cannot exceed looking team point.

For Figure 1, Ford Fulkerson Algorithm is executed. After that, it can be seen that, all edge capacities between level 1 and level 2 are full, which shows us team zero has chance to win.

5 ANALYSIS

In this problem, Ford Fulkerson algorithm is used. The pseudo code of that algorithm can be seen in the Figure 2.

```
Augment(f, c, P) {
   b ← bottleneck(P)
   foreach e ∈ P {
      if (e ∈ E) f(e) ← f(e) + b      forward edge
      else       f(e^R) ← f(e) - b    reverse edge
   }
   return f
}
```

```
Ford-Fulkerson(G, s, t, c) {
   foreach e ∈ E  f(e) ← 0
   G_f ← residual graph

   while (there exists augmenting path P) {
      f ← Augment(f, c, P)
      update G_f
   }
   return f
}
```

Figure 2: Pseudo code of Ford Fulkerson Algorithm

The analysis of Ford-Fulkerson depends heavily on how the augmenting paths are found. In my implementation, it used depth-first search for finding augmenting path. With this method, Ford-Fulkerson runs in polynomial time.

In my implementation, all flows are integers; hence, the while loop of Ford-Fulkerson is run at most C times, where C is the maximum flow. This is because the flow is increased, at worst, by 1 in each iteration.

Finding the augmenting path inside the while loop takes $O(V + E)$ where E is the set of edges in the residual graph. This can be simplified as $O(E)$. So, the runtime of Ford-Fulkerson is $O(EC)$.

Because of graph structure, the space complexity of the Ford Fulkerson Algorithm in my implementation is $O(V + E)$ where V is number of vertices and E is number of edges.