CSE 4081: Algorithms Spring 2024
Assignment 05: Joy with Network Flows
Due Date: Friday, April 12; 23:00 hrs
Total Points: 15

Network optimization is a field of computer science that solves optimization problems (problems that involve minimizing or maximizing an objective function) on graphs or networks. For example, computing shortest paths can be viewed as an optimization problem – among all paths from the source s to the sink t, what is the shortest?

The maximum (s, t) flow problem asks to compute the maximum amount of a commodity that can be sent from s to t. The famous result of Ford and Fulkerson in 1951 established the duality between maximum (s, t) flows and (s, t) minimum cuts, where an (s, t) cut is a partition of the nodes of the network into two components such that s is one of the components and t is on the other component. The cost of a cut is the sum of the edges that go between the two components. Maximum flows and minimum cuts have lots of applications including project scheduling, tuple selection, and bipartite matchings. In this assignment, we will use maximum flows to determine all teams that are already eliminated in a soccer tournament – specifically, in the English Premier League, from the 2018-2019 season.

1 Soccer Elimination

The English Premier League (EPL) is one of the most popular soccer tournaments in England (also includes a few teams from Wales). It is a 100 billion dollar industry enjoyed by fans around the world. There are 20 teams participating, and each team plays every other team twice, once in their home stadium and once away. Players from all around the world are contracted to play in these teams.

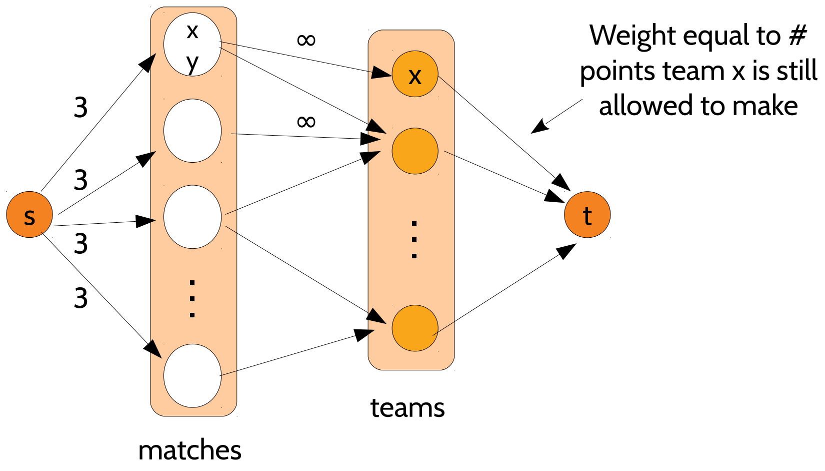
Each team plays 38 games, and there are a total of 380 games. Winning a game gives the team 3 points, and a draw gives each team 1 point. In this problem, however, we will only look at wins and loses (so draws don't count).

You are given the standings or scores of all the teams after certain number of games. Let the array $p_1, \ldots, p_n$ contain these scores, so p_i is the current score for team i. You are also given the remaining fixtures (or matches yet to be played) as an array m, where each match is a tuple (x, y) that indicates that a game is yet to be played between teams x and y. We would like to determine all teams that are already eliminated.

1.1 Modeling as a Maximum Flow Problem

In order to determine if team z is eliminated or not, we can model this problem as computing a maximum flow in a graph. Let the array $g_1, \ldots, g_n$ provide the number of remaining games yet to be played for each team, so g_i is the number of remaining games yet to be played for team i. We first assume that team z wins its remaining matches. Therefore the total points it can make is $p_z + 3g_z$. Note that a win constitutes 3 points.

We can construct a graph with the following nodes: a source node s, a sink node t, a set of nodes M for each match $m_i = (x, y)$ between teams x and y, and a set of nodes T for each team. We connect the source node to each of the match nodes with a weight of 3. We connect each match node between teams x and y to the respective team nodes x and y with infinite capacity. In this problem, we can set these capacities to be 4000. Finally, we connect each team node x to the sink node with capacity equal to the total number of points team x can still make so that team z is not eliminated, which is $p_z + 3g_z - p_x$ (see figure below). Note that team z and all of its matches are not included in this graph since we assume team z wins all of them.



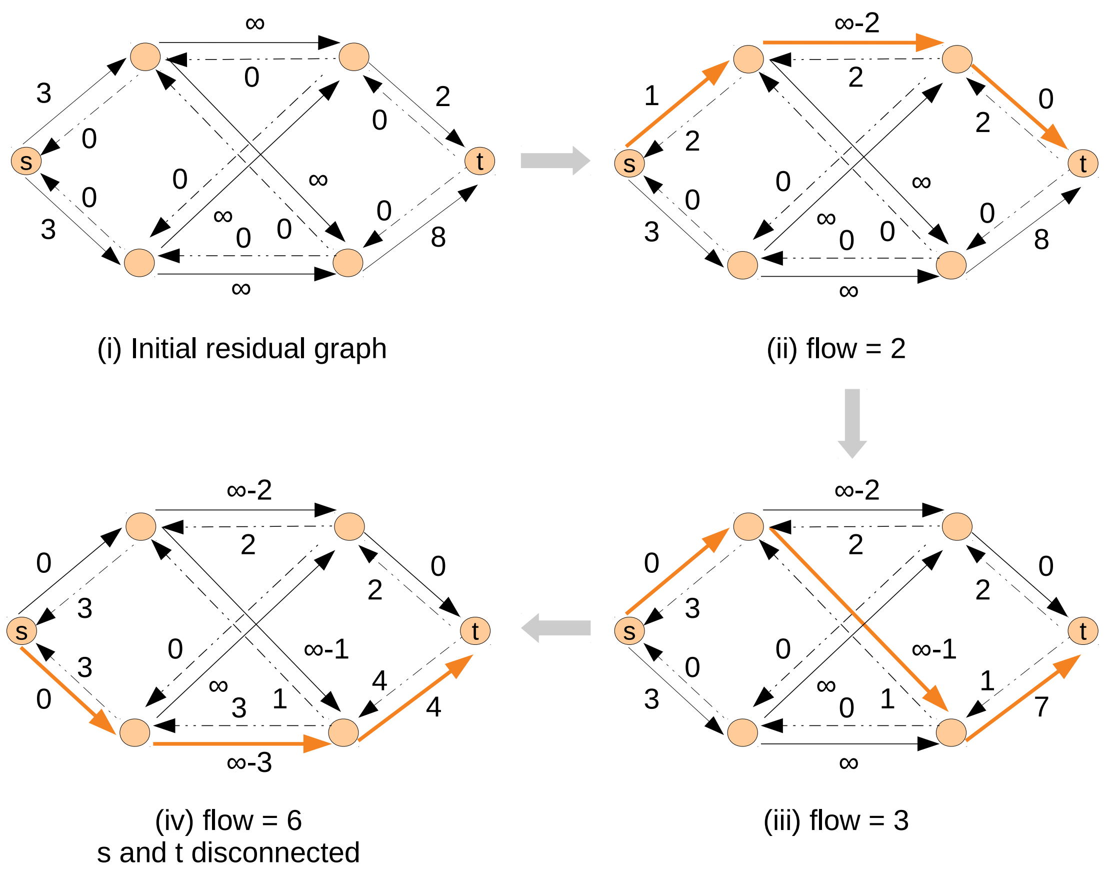
A maximum flow in this graph from source s to sink t can determine whether team z is eliminated or not. If each of the edges that go out of the source node is saturated (i.e., we are able to send a flow equal to the capacity of each of these edges), then it means that all of the matches can play out in such a way that there is still a chance for team z to win the title. If the total number of matches not including team z is $m_z = |M| - g_z$, then a maximum flow value of $3m_z$ indicates that team z is not eliminated. Otherwise, the maximum flow will be smaller.

1.2 Ford-Fulkerson Augmenting Path Algorithm

To compute a maximum flow from a source node s to a sink node t, we can use the famous Ford-Fulkerson algorithm. Let $G = \{V, E\}$ be the original graph with node set V and edge set E. We first construct a *residual graph* G' as follows. G' contains the same set of nodes as G along with the set of edges E. In addition, for each edge $u \to v$, we also add an edge $v \to u$ with a capacity of 0. The edges in the residual graph are called *residual edges* and their capacities are called *residual capacities*, which indicate how much more flow can be sent through that edge.

A path in the residual graph is called an *augmenting path*. To *augment a flow* along an augmenting path from s to t means to find the minimum edge e (based on their residual capacities) on the path and subtracting the residual capacities of each forward edge $u \to v$ by e and adding the residual capacities of each reverse edge $v \to u$ by e on the path. The Ford-Fulkerson algorithm works as follows on the residual graph.

1. Find an augmenting path P from node s to t using a depth-first search.
2. Augment flow along path P. Note that this updates the residual capacities along edges and their reverse edges on P.
3. Repeat steps 1 and 2 until s is disconnected from t.



The figure above illustrates this algorithm on a small sample graph. For this problem, we will set $\infty = 4000$. See the initial residual graph in Step (i). Steps (ii, iii and iv) shows three different augmenting paths through which we augment the flow. In Step (iv), notice that the source and the sink nodes are disconnected, and therefore the Ford-Fulkerson algorithm terminates. In the worst-case, this algorithm could send only one unit of flow along an augmenting path, and therefore runs for $O(f)$ iterations, where f is the maximum flow value. Since each time, depth-first search is run to find an augmenting path, the total running time of the algorithm is $O(fm)$. This is not a polynomial running time, although faster maximum flow algorithms exist.

Also note that when the Ford-Fulkerson algorithm terminates, we can run depth-first search another time from the source node to identify all nodes that are on the source side of cut, and all the unvisited nodes are on the sink side of the cut. The sum of the capacities of the edges that cross the cut is equal to the maximum flow value based on the max-flow-min-cut theorem.

1.3 Input and Output

The input file contains the following format. On the first line is one integer n that represents the total number of teams in the tournament. The next $2n$ lines describes each team, where the first of these lines represents a team and the following line represents its score. Following the descriptions of all teams, we have one line with a single integer m that contains the number of remaining matches. The next $2m$ lines each describes a match, where the first team is listed on the first line, and the second team is listed on the second line. Please see the input file `matches300.in`, which is the input after 300 matches in the 2019-20 season.

The output file lists teams that are still challenging for the title, and teams that are already eliminated. The first line lists a message "Teams still challenging are...". Then each team that is still competing for the title are listed one line at a time, sorted based on their current scores. Each team is described on their own line, with their scores and the maximum flow value in parenthesis. Then we have an empty line followed by a line carrying the message "Teams that been eliminated are...". This is followed by the descriptions of teams that have been eliminated, sorted by their scores. Each team name, its score and a message as to how it is eliminated – which is either "as it cannot catch up", or "due to smaller max flow" are displayed in parenthesis. Please see the output file `matches300.out`, which is also shown below.

```
Sample Output: (file matches300.out)

Teams still challenging are...
Man City 74 (flow = 216)
Liverpool 73 (flow = 216)
Tottenham 61 (flow = 216)
Arsenal 60 (flow = 216)
Man United 58 (flow = 216)
Chelsea 57 (flow = 213)

Teams that have been eliminated are...
Wolves 44 (as it cannot catch up)
Watford 43 (as it cannot catch up)
West Ham 39 (as it cannot catch up)
Leicester 38 (as it cannot catch up)
Bournemouth 38 (as it cannot catch up)
Everton 37 (as it cannot catch up)
Newcastle 35 (as it cannot catch up)
Crystal Palace 33 (as it cannot catch up)
Brighton 33 (as it cannot catch up)
Southampton 30 (as it cannot catch up)
Burnley 30 (as it cannot catch up)
Cardiff 28 (as it cannot catch up)
Fulham 17 (as it cannot catch up)
Huddersfield 14 (as it cannot catch up)
```

For teams that are still challenging, the flow value indicates how much units of flow is successfully sent through the residual graph. For teams that are eliminated, we display a small message as to why they are eliminated. They are eliminated two possible ways – either this team can never catch up to the leader – so the total number of points from its remaining games is still not enough, or the max flow value is smaller than the total points it can make from its remaining games. Please check your output for `matches300.in` using the `diff` Unix utility with the expected output in `matches300.out`. In addition, please also thoroughly test your program against other inputs.

1.4 Note on Computation

Note that this problem asks you to find if each of the teams in the input are eliminated or not. This means that you need to run your Ford-Fulkerson algorithm n times, one for each team. Each time you run your algorithm to compute the maximum flow, you are to create the adjacency matrix (or adjacency list) of the initial residual graph. Moreover, care must be taken to initialize all other data structures, for example, the visited and predecessor arrays while running depth-first search. Therefore consider putting the Ford-Fulkerson algorithm as a function that takes in the appropriate residual graph, source and sink nodes as arguments, and that returns the value of the maximum flow from the source to the sink. It will also be beneficial to have a function that builds the residual graph from the input data (which is read once and stored in appropriate lists, sets or maps). At a very high level, the program should follow these steps.

1. Read the input and store them in appropriate data structures (lists, sets or maps). Note that you might need separate data structures for teams and matches.
2. For each team t:
    (a) See if team t can catch up to the leader. If not, this team is eliminated and added to the eliminated list.
    (b) Build a residual graph G_t as described in Section 1.1. Note that this graph contains all teams except t, and all matches except between those played against team t. The Ford-Fulkerson algorithm is easier to implement if the graph is represented using adjacency matrix (although you are welcome to use adjacency list for your implementation). This means that there needs to be integer labeling of all the nodes. In my implementation, I used numbers $0, \ldots n-1$ for the team nodes, n for the source, $n+1, \ldots n+m$ for the matches and $n+m+1$ for the sink. Any other valid integer labeling can be used. This means that it may also be beneficial to have a hash table that maps team names and matches to these integer node labels.
    (c) Run the Ford-Fulkerson on G_t to compute the maximum flow f from the source node to the sink node. If f saturates all edges out of the source node, or in other words, f is equal to the total number of points team t can still make, then there is a way for all the games to play out such that team t can still compete for the title. Add team t to the list containing teams not eliminated. Otherwise, add team t to the eliminated list.
3. Sort both eliminated and not eliminated lists based on team scores.
4. Go through each team in the not eliminated list and print them in the right format.
5. Go through each team in the eliminated list and print them in the right format.

2 Postamble

2.1 Data Credits

The data for this assignment was taken from here.

Many thanks to Daniel Jeffries, as always, in collecting the data for this assignment and formatting it in a way that can be used in this problem.

2.2 Makefile

As part of the submission, please submit a makefile that includes the following targets with their actions as described.

`copyfiles`	Copy all the test files from the directory specified by the argument `filepath`.
`compile`	Compiles all the files of your solution.
`run`	Runs your solution with two arguments `input` that takes the input file, and `output` that writes the output file.
`list`	Lists the files of your solution.
`show`	Prints your solution, one file at a time.

A sample makefile for my solution is in `makefile`. Please use this to build your own makefile.

2.3 Submission

Please submit all the files as a single zip archive `h05.zip` through Canvas. The zip archive should only contain the individual files of the assignment, and these files should not be inside folders. You can use the `zip` command in Unix for this.

```
$ zip h05.zip <space separated list of files>
```

Please only include files that represent your solution. Please do not include other extraneous files, including pre-compiled class or object files. We will compile your solutions before executing them so these files are not necessary.

2.4 Proper Use of Libraries

For this assignment, you are allowed to use the following data structure utility libraries. In general, you are allowed to use libraries for input and output, math and random functions.

C++	STL libraries `pair`, `vector`, `list`, `deque`, `queue`, `priority_queue`, `stack`, `set`, `multiset`, `map`, `multimap`, `unordered_set`, `unordered_multiset`, `unordered_map`, `unordered_multimap`
Java	`Arrays`, `Stack`, `ArrayList`, `LinkedList`, `TreeSet`, `TreeMap`, `HashSet`, `HashMap`, `Queue`, `Deque`, `PriorityQueue`
Python	tuples, lists, sets and dicts as defined in the Python language. Also allowed are `deque` and `heapq` from `collections`.

You may use these `Geeks for Geeks` resources for more information, or please check out the official language documentations.

C++ STL: `https://www.geeksforgeeks.org/the-c-standard-template-library-stl/`

Java Collection: `https://www.geeksforgeeks.org/collections-in-java-2/`

Python: `https://www.geeksforgeeks.org/python-programming-language/`