CS218 Design and analysis of algorithms **Jan-Apr 2023**

Programming Assignment 2

Total Marks: 10

A game on graphs: Consider a two player game played on a complete directed graph with weights on its edges $\{w_{i,j} : 1 \leq i \neq j \leq n\}$. The game starts with a token being present on the vertex v_1. The two players alternately make a move in the game. A player, when it is their turn, moves the token from its current vertex i to a different vertex j and gets a reward of $w_{i,j}$. The total number of moves for allowed for each player is pre-determined, say m. Whichever player has the higher total reward at the end wins. In case the two players have equal total rewards, player 2 is declared as winner.

For example, consider the following game on a graph with 4 vertices. The edge weights are given by the following table.

	v_1	v_2	v_3	v_4
v_1	-	10	5	3
v_2	9	-	7	12
v_3	2	8	-	6
v_4	5	13	4	-

Suppose the number of allowed moves is $m = 2$ for each player. Consider the following play of the game:

- player 1 : $v_1 \to v_3$. Gets a reward of 5 points.
- player 2 : $v_3 \to v_4$. Gets a reward of 6 points.
- player 1 : $v_4 \to v_1$. Gets a reward of 5 points.
- player 2 : $v_1 \to v_2$. Gets a reward of 10 points.

The total reward for player 1 is 10 and for player 2 it is 16. Hence player 2 wins.

In fact, in this particular game, player 2 has a strategy to win irrespective of how player 1 plays. We demonstrate this in Figure 1.

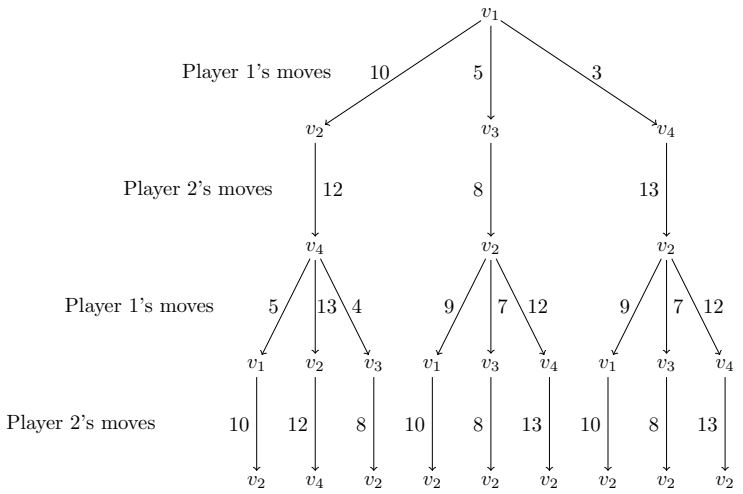

Figure 1: An example of a game, where irrespective of how Player 1 plays, Player 2 can find a way to have a higher total reward at the end.

Observe that in any given game, either player 1 or player 2 will have a winning strategy. To see this formally, suppose $m = 2$. Then player 2 has a winning strategy if

$$\forall v_i \; \exists v_j \; \forall v_k \; \exists v_\ell, \; w_{1,i} + w_{j,k} \leq w_{i,j} + w_{k,\ell}.$$

If player 2 does not have a winning strategy, then we get the negation of the above statement, which is equivalent to

$$\exists v_i \; \forall v_j \; \exists v_k \; \forall v_\ell, \; w_{1,i} + w_{j,k} > w_{i,j} + w_{k,\ell}.$$

This is nothing but saying that player 1 has a winning strategy.

For any given game, the task is to compute which player has a winning strategy.

Instructions

Input contains $n(n-1)+2$ lines.

Line 1: n (the number of vertices in the graph)
Line 2: m (the number of allowed moves for each player)
Line 3: $w_{1,2}$
Line 4: $w_{1,3}$
$\vdots$
Line $n+1$: $w_{1,n}$
Line $n+2$: $w_{2,1}$
Line $n+3$: $w_{2,3}$
$\vdots$
Line $(n(n-1)+2)$: $w_{n,n-1}$

Output :

1 or 2 (which player has a winning strategy)

- Programming Language: C++. We will compile your code with g++. Make sure that it works.
- Submission: put your code in a file named XXX.cpp where XXX is your roll number. Also, write a short explanation (a paragraph) of what your algorithm does, put this in XXX.pdf. The two files should be uploaded on Moodle (do not zip/compress).
- Given files: In the `GraphGame` folder, you will find: (i) helper.cpp (a c++ code showing expected input/output, feel free to use) (ii) Few sample input and output files, (iii) an executable file, which can be used to get the correct output on any input.
- Running time: we will test your code on some similar size instances as given in the sample input file. If your program runs in time polynomial in n and m, that should be good enough. An exponential running time like $(n-1)^m$ will be too slow to pass the inputs.
- Academic integrity: Mention all references if you have referred to any resources while working on this assignment in the pdf. You are supposed to do the assignment on your own and not discuss with anyone else. We will do a plagiarism check on your submission using MOSS. It's fairly sophisticated and can detect even when you have made modifications in someone else's code. Any cases found with significant overlap will be sent to DADAC. If DADAC finds it to be a case of plagiarism, then the penalty is zero in the assignment and final course grade reduced by 1 point.
- Grading: The test cases will be of varying sizes (will keep $n \leq 50$ and $m \leq 100$). Since the output is simply 1 or 2, one can try to generate the answer randomly and hope to be correct on half of the inputs. To catch this, we will run on each input multiple times and expect the same answer.