Game Analysis and Project

This is extra material, connected with Depth First Search in Chapter 22. An *optional* project is described after the material.

1 The Game

We consider a two person game, call the Players Paul and Carole. There is a set of positions V. For each $v \in V$, $MOVE[v]$ is either Paul or Carole and tells you whose turn it is. There is a designated starting position s. We have a directed graph G on V. For each $v \in V$ there is an adjacency list $Adj[v]$ of those positions w that can be reached in one move. We assume that V is finite and that G has no cycles, that is, a DAG. We call v an end position if $Adj[v]$ is empty, that is, a leaf in the DAG. For each end position v there is a value $VALUE[v]$.

Here is the game. Start at s. Players move (remember that $MOVE[v]$ is part of the data so you know who must move) until reaching a leaf v. The game is then over and Carole pays Paul $VALUE[v]$ dollars.

Naturally, Paul wants to maximize his payoff and Carole wants to minimize it. But if the game has a million positions how can they analyze it.

DFS provides the key. We apply $DFS - VISIT[G, s]$. (Any position not reachable from s is clearly irrelevant to the analysis. So lets assume that all of V is reachable from s and that G has V vertices and E edges. Now $E \geq V - 1$ as every position except s came from somewhere. So the time $\Theta(V + E)$ for DFS is actually $\Theta(E)$.) Everytime a vertex v becomes Black we define $VALUE[v]$. For the final positions v, $VALUE[v]$ is given in the data. So assume v is not a final position. Note, critically, that when v becomes black all $w \in Adj[v]$ have already become blace so that, recursively, their $VALUE[w]$ have already been determined. There are two cases:
$MOVE[v]$ is Paul. Paul wants to make the move that will maximize his value. So set

$$VALUE[v] = \max_{w \in Adj[v]} VALUE[w]$$

$MOVE[v]$ is Carole. Carole wants to make the move that will maximize his value. So set

$$VALUE[v] = \min_{w \in Adj[v]} VALUE[w]$$

For the extra time, beyond DFS itself, observe that for each v we examine the $w \in Adj[v]$ and so that is a total of E pairs v, w so the additional time

is $\Theta(E)$. This is the same order of time as DFS itself. So the game can be analyzed in time $\Theta(E)$. This works pretty nicely for Tic Tac Toe.

2 The Optional Project

`Special Rules:` You may work in a group with at most 3 students and you may (unlike the assignments) hand in a single report for all of you. The report will *not* get a grade. A bad report cannot hurt your final grade, a good report can help. No specific percentages are given to the effect of the report on your grade. But please note that this is not considered a major project and you should limit the amount of time you devote to it.

This optional project is due on May 1. Electronic submission directly to Prof. Spencer is prefered, but hard copy submission is also acceptible.

The project is to implement this algorithm for a variety of games and to analyze the date. *It will be fine if you analyze one of the two games below.*
`Nim`: There are piles of matches of different sizes and the state is given by the size of each pile *and* whose move it is. The classic example is the start state being $(3, 5, 7)$ meaning there are piles of size $3, 5, 7$ and it is Paul's move. At each round the player takes any amount of matches (at least one) from any one of the piles. For example, from $(3, 5, 7)$ there is an arc to $(1, 5, 7)$ and it is now Carole's move. When there are no matches left the player whose turn it is loses. We make the values $+1$ for a Paul win and -1 for a Carole win. There are two leaves.$(0, 0, 0)$ has value -1 when it is Paul's move and value $+1$ when it is Carole's move. Applying the algorithm will give the value of the start state, meaning who will win with perfect play. Run it for several initial states. Nim has a mathematical solution so, if you know it, check against that solution.
`Sequence`: There are n moves, alternating between Paul (first) and Carole. At each move the player selects a bit, zero or one. The starting node, denoted e, is the empty string. After u moves the intermediate node will be a binary string of length u. At the end of the game, the leaves are the 2^n strings of length n. The values $VALUE(x)$ for the leaves x are set in advance. The algorithm should then compute $VALUE(e)$, the value of the game to Paul.

Analyze, through simulation, what happens when the 2^n values $VALUE(x)$ are chosen as random real numbers in the interval $[-1, +1]$. What is $VALUE(e)$?? Of course, it will depend on the random numbers. For each n do some number of simulations and get data on the different $VALUE(s)$. (The number of nodes is $2^{n+1} - 1$ so that even for $n = 20$ the algorithm should be pretty quick.)

Now – put on your Data Scientist hat! Display this data in some meaningful way and come up with a reasoned conjecture (from the data) about what $VALUE(s)$ will usually be as a function of n. Your instructor does *not* know what will happen! Here are some things to think about.

1. Is there an advantage to playing last? If so the parity of n would affect the data.
2. Is there an advantage to playing first? By putting values for leaves random in $[-1, +1]$ if there was no advantage the value would be zero.
3. Is there much variance in the results. (Take some n and do many runs.)
4. Is there a distribution to the results? (Again, take some n and do many runs.)
5. What does the limiting behavior as $n \to \infty$ appear to be?
6. Surprise your instructor! Come up with some other analysis.

Most of all, have fun – explore – take to heart the words of the founder of Theoretical Computer Science, Don Knuth:

> ...pleasure has probably been the main goal all along. But I hesitate to admit it, because computer scientists want to mantain their image as hard-working individuals who deserve high salaries. Sooner or later society will realise that certain kinds of hard work are in fact admirable even though they are more fun than just about anything else.