CS109B PROJECT: A HARD GRAPH PROBLEM

Due Friday, March 8

The purpose of this lab is to get practice using graph algorithms and to appreciate that not all problems for which we can write a program can be solved in a reasonable amount of time. You are asked to develop a program to play a game of "cut-vs.-short"; in particular, we shall use the lower 48 states as the game board. We start with a pair of states as the endpoints, say Maine and California.

The players take turns choosing states. You cannot choose a state already chosen by the other player, and you cannot choose one of the endpoints. The goal of "short" is to choose a set of states that form a path between the endpoints (i.e., a short circuit), and the goal of "cut" is to choose a set of states whose removal separates the two endpoints.

For example, with Maine and California as the endpoints, cut can seize either New Hampshire or New York, because both are on every path from CA to ME. In fact, even if short goes first, one of NH and NY must be left when cut goes, so cut is the winner of the CA-ME game no matter who goes first.

Strategies

It is probably best to represent positions by modifying the graph. A state seized by cut is simply removed from the graph. A state seized by short is removed by an operation similar to pivoting in Floyd's algorithm. That is, to remove v, we introduce edges between each of v's neighbors. More formally, we find all edges $\{u_1, v\}$, $\{u_2, v\}, \ldots$, and add edges $\{u_i, u_j\}$ for all $i \neq j$, if they are not already there. Then we can remove v and all edges containing v, sure that we can get between any two neighbors of v.

Suppose your program is playing short (playing cut is similar). In principle, we could write a recursive program that explores the entire set of possible positions. If there is now an edge between the two endpoints, then you win, and no recursion is necessary. If the two endpoints are in separate connected components, then you lose, and again no recursion is necessary. In all other cases, you consider each possible choice of node, modify the graph (temporarily) as if you had chosen that node, and call yourself recursively, but with the understanding that it is now cut's move. If you find a move that always leads to a win, take it. Otherwise, choose a move arbitrarily, since you must lose.

If you try this, you'll find that there is just not enough time to follow the process to completion, and you must try a *heuristic*, that is, a strategy that is not guaranteed to find a winning move, even if one exists, but often finds the right move. With luck and skill, you will find a heuristic that works well and also spends no more than a fraction of a minute selecting a move.

- One approach is to start by checking for a forced or obvious move. For example, an *articulation point* is a node that, when removed, breaks one connected component into two or more. NY and NH are the only articulation points in the full map of the states, but as the game proceeds, you may find others. If you are cut, you win by seizing an articulation point, provided it separates the endpoints. If you are short, you had better seize such an articulation point if there is one, or you will lose on the next move.

- Thinking more deeply, if there are two nodes such that any path between the endpoints goes through one of them, it might be a good idea to seize one of them.
- In more ambiguous situations, there might be a tendency to think that seizing a state next to an endpoint is a good idea, but probably something in the middle leaves you with more flexibility.
- You might be able to rule out some states as irrelevant, because the winning path or cut couldn't possibly use them (e.g., states in the corners tend to be irrelevant if they are not endpoints).
- Another possible approach is for short to find a shortest path (counting one for each edge) between the endpoints, and pick one of the nodes on that path. That is sure to reduce the minimum distance between the endpoints, and when it gets down to 1, short wins. Conversely, cut might try to find a node on all the shortest paths, or on most of them, and seize that node, thus trying to increase the distance between the endpoints.
- As a final suggestion, when the graph gets small enough, you may be able to go for an exact solution, that is, search all possible sequences of moves or the graphs that they engender.

The point of the above paragraphs is not to establish what you *must* do, but to give you some sense of the variety of possibilities. Often, when an instructor poses a complex problem such as this one, the most effective solutions have little to do with what the instructor imagines is the right idea.

Requirements

At a minimum, you should write a program that will play one of cut and short, accept two states as endpoints, and alternate with an external player. Your program may go first. Preferably, your program should also accept input telling it whether to play cut or short, and whether to go first or second.

This project is open-ended; there is no limit to how much effort you could put into perfecting the strategy. However, we shall accept a program that implements only a few simple techniques such as the ones we've discussed above. You should hand in

1. Your code.
2. A clear description of the strategies you've implemented.

We shall try the programs out, either playing against a person or another program. In order that we not get bored doing so, we shall require that programs take *no more than a half-minute per move*. However, since that limit is hard to enforce exactly, and may depend on load, we'll be flexible if you've tried to keep the running time down.

Note: Winning or losing in the trials will not affect your grade for the project. You can get full marks for sensibly implementing some sensible strategies and describing them well.

Language

The official languages are Pascal and ML. You may find that ML loses its charm when the problem gets complex, and it is hard to implement some efficient algorithms in ML. We'll consider other languages if there is a TA willing to grade the work.

Teams

It is strongly suggested that you form teams of 2 or 3 for this project. There is rather more work to do than in the CS109A project. It would also be wise to work on the project continuously, rather than start on March 1.