CS 310 Programming Assignment 3 (pa3)

Algorithm techniques: Games and dynamic programming

Betty O'Neil, Spring 2021

Due date: Monday, April 26 in your cs310/pa3 directory

Purpose

This assignment aims to help you:

- Learn about two-player games
- Try out recursive search, and dynamic programming to speed it up
- Design an interface to describe an activity

Reading

See the slides of Apr. 1 on Games, and Apr. 6 on PA3. Study the slides on dynamic programming. Read Wikipedia Dynamic Programing intro, and Sec. 1.3 and 2.2. Also there: a fascinating story on how DP got its name. Also see Geeks for Geeks Dynamic Programming Basic Concepts. Weiss Chap. 7, esp. 7.6., and 10.2.

See Suggested Steps for more information.

Description

In this assignment we continue our study of algorithms and algorithm patterns. We'll develop another interface, in this case a Games API. Here we will work with recursive search and dynamic programming. Dynamic programming is of great use in serious programming efforts, as we saw in Prof. Haspel's slides.

Parts

0. Download the pa3setup project. We will start from Weiss's TicTacToe program covered in class, provided in TicTacToe.java, and Weiss's Best.java, in a version using backtracking with dynamic programming (but not using alpha-beta pruning). I've changed a few details in it for better software engineering. PlayTicTacToe is a provided client for it. Try it out as provided. Create a new package cs310.games and relocate all these sources to is by adding a package statement to each. Make sure PlayTicTacToe still runs in its new home. Do the rest of the project in the cs310.games package.

1. Copy TicTacToe.java and PlayTicTacToe.java to TicTacToe1.java and PlayTicTacToe1.java (in cs310.games) for editing. First, get rid of TicTacToe1.getBoard(). This method copies out the internal data of the game and gives it to the client, violating the encapsulation of the class. The client only uses it to print out the board, a legitimate action for TicTacToe itself to do, so move this printBoard method to TicTacToe1. Also make Position into a private inner class, tucking it inside the game class TicTacToe1. Leave this version as TicTacToe1.java and PlayTicTacToe1.java.

2. Try out the provided Nim.java by executing its test code in main.Then write PlayNim2.java, by starting from PlayTicTacToe1 and replacing calls as necessary, i.e., change it as little as possible. In this version, to replace the call to chooseMove by asking the user for a move for the computer. You'll end up with a PlayNim2 where the user is playing for both human and computer.

3. In Nim3.java and PlayNim3.java, provide Nim3 with chooseMove (recursive search without dynamic programming) so that the computer plays a good game. Note that making a trial move in chooseMove requires changing one of the heaps *and* the nextPlayer value. Create a class BestMove.java for the best move that has generic names for the two numbers, i and j, as well as val.. The computer, once optimal, should choose move

"take 3 stars from row 0", i.e. i=0, j=3, to start the game. Convert TicTacToe to use BestMove, resultiing in TicTacToe3.and PlayTicTacToe3.

Do either 4a or 4b, but study the solutions to both parts.

4a. **Game Interface.** Write an interface Game.java that covers the actions needed for the clients for both games PlayTicTacToe3 and PlayNim3, and change the method names in one or the other game sources for similar actions, ending up with TicTacToe4.java and Nim4.java that both implement Game. Put the four constants used for values of the side parameter and which side won in the interface too. Merge the client codes into one code PlayGame that uses the interface type as much as possible (everything except creating needed objects). For example, clearBoard() in TicTacToe has the same generic action as init() in Nim, namely, to bring the game to its initial state. Clearly init() is the more generic name, so rename clearBoard in TicTacToe to init and put init in the interface. Change the prompts to "i" and "j", a little cryptic, but simple. Usage: PlayGame uses TicTacToe4 by default (no arguments case) or Nim4 if it is run by "java games.PlayGame Nim".

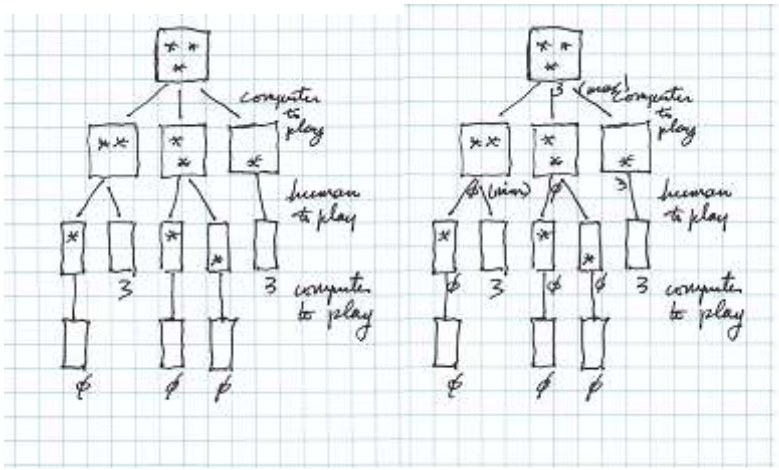
4b. **DP for Nim**. In Nim4DP, add an inner class Position and the big Map, following the setup for TicTacToe. Change chooseMove to use it to speed up Nim. Edit PlayNim3 to PlayNim4DP. It should only need a change to the game-creation step.

memo.txt In the file memo.txt, answer the following questions, in one to two pages (50-100 lines) of text. Use complete sentences, as you would for an English class
1. Discuss your experiences in writing these programs. What was the hardest part for you? Are you now a recursion expert?
2. It is well known that the first player in 5-3-1 Nim has a winning strategy by picking 3 starts from row 1. What does this mean for the value of the initial position in Nim? What is the initial position value for TicTacToe? What does that mean?
3. How many game states are there in Nim (upper bound, like 3^9 for tic tac toe)? Don't forget that the full game state includes whose turn it is. What is the height of the full game tree? (Don't try to draw the whole tree! Just think about the longest possible game.)
4. The simple game Sticks starts with 7 sticks. Each player removes 1 or 2 sticks on a turn. Players alternate turns. The last player to remove a stick wins. How should we represent the game state for this game? Specifically, what Java instance variables are needed in the game class, corresponding to "private int[][] board = new int[3][3];" in TicTacToe or the two instance variables in Nim?

Deliverables: in your cs310/pa3/src/cs310/games directory on Linux:

- `TicTacToe1.java, PlayTicTacToe1.java, Best.java (Best is unchanged)`
- `Nim.java (unchanged), PlayNim2.java`
- `BestMove.java, Nim3.java, PlayNim3.java, TicTacToe3.java, PlayTicTacToe.java`
- `Game.java, TicTacToe4.java, Nim4.java, PlayGame.java`
    - `To run PlayGame: "java cs310.games.PlayGame" plays TicTacToe, "java cs310.games.PlayGame Nim" plays Nim`
- **`or instead of Game, TicTacToe4, etc:`** `Nim4DP.java, PlayNim4DP.java`
- `memo.txt`

For help in debugging chooseMove in Nim. Partial Game tree for Nim. To get this in execution, edit the Nim.java source to have SZ_ROW0 = 0, SZ_ROW1 = 2 in Nim.java. This shows that the computer should take 2 stars from the current top row on the next move after doing max over (0,0,3). That leaves one star on one row, which the human has to take, so loses.



Nim tree for simplified setup, before and after propagation of values up the tree. Note that the computer has a sure win here.