cmput 355 2024   homework 4, with hints

For tic-tac-toe, use solver `ttt/tt24.py`. Explain each answer, show all work.

1. For each of the 3 non-isomorphic opening tic-tac-toe moves, find all best opponent responses.

2. Consider this 2-player alternate-turn win/loss/draw game, played on a 3x3 board. On a turn, a player colors any empty cell. Once the board is full, the win/loss/draw condition (which we are not telling you) is used to determine the winner.

   a) How does this game differ from tic-tac-toe? b) How many nodes are in the tree of all continuations of the game?

3. Use the solver: how many nodes are in the tic-tac-toe

   a) tree of all continuations of the game?

   b) directed acyclic graph (dag) of all continuations of the game?

   c) tree of all continuations of the game if we prune isomorphic positions?

   d) dag of all continuations of the game if we prune isomorphic positions?

4. In `tt24.py`, modify function `has_win()` so that it always returns false.

   a) From the empty board, does `info( )` still execute?

   b) If a) is yes, what is the first-player outcome (win/loss/draw)?

   c) What node counts does it return?

   d) What nodes counts does it return with transposition checking?

   e) What nodes counts does it return with isomorphism checking?

   f) What nodes counts does it return with transpotion and isomorphism checking?

5. a) In `tt24.py`, in function `negamax()`, uncomment this line

   ```
   if so_far == 1: break   # improvement: return once win found
   ```

   and solve from the empty board. How many nodes are in the search tree?

   b) Repeat a) for the search dag instead of search tree.

   c) Repeat b) if also checking for isomorphisms.

6. Modify `tt24.py` as in the previous question and also, in function `legal_moves()`, comment out preceding line `for j in range(Cell.n):`

   and uncomment line `#for j in (4, 0, 2, 6, 8, 1, 3, 5, 7):`.

   Give the number of nodes to solve from the empty board

   a) without transposition or isomorphism checking

   b) with transposition checking

   c) with both transposition and isomorphism checking.

7. Consider the tic-tac-toe variant *x-bias*:

   - x gets 3-in-a-row: game ends, x wins, o loses
   - o gets 3-in-a-row: game continues
   - board fills and x does not get 3-in-a-row: draw

   Modify `tt24.py` for these rules.

8. Consider a tic-tac-toe position p. Using `tt24.py`, we solve p and get a node count of x. Then we turn on the transposition table option and solve p and get a node count of y.

   a) What is the relationship between x and y? (equal, $x > y$, $x < y$, or it varies)

   b) Continue from a). Turn the transposition table option off and solve p for a third time. What is the node count z?

9. Convert decimal number 29 into binary.

10. Convert binary number `1 0 1 1 1 0 1 1` into decimal.

11. For a nim position with pile sizes 15, 27, 14, 25, 7, find all winning moves.

```
a    15      1 1 1 1
b    27      1 1 0 1 1
c    14      1 1 1 0
d    25      1 1 0 0 1
e     7      1 1 1
```

12. Find a 3-pile nim position with exactly 2 winning moves, or explain why there is no such position.

13. a) Draw the dag of all continuations of the game for nim(2 2 2). Group nodes by their multiset of non-zero pile sizes, e.g. group all 6 permutations of (1 0 2) as multiset {1, 2}, group all 3 permutations of (1 0 1) as multiset {1, 1}, etc. The root of your dag will be the node {2, 2, 2}, its children will be {1, 2, 2} and {2, 2}, and the lowest node in the dag will be {}, the position with all piles empty. Circle all losing nodes.

    b) Draw the top two levels (the root and all children) of the tree of all continuations (TOAC) of the game for nim(2 2 2). Also, for each multiset that appears as a node in the TOAC (so {}, {1}, {1,1}, {2}, {1,1,1}, {1,2}, {1,1,2}, {2,2}, {1,2,2}, {2,2,2}), give the number of nodes in the subtree of that multiset in the TOAC.

14. a) Explain the name of `nim/nim-memo-calls.py`.

    b) Show the output for nim(1 2 2)

    c) Show the output for nin(2 2 1)

    d) Uncomment line `psn = tuple(sorted(nim_psn, reverse=True))` and comment out line `psn = tuple(sorted(nim_psn))`. Show the output for nim(1 2 2).

15. a) In `go/tromp.c` and `go/tromp.py`, how would you change the move ordering so that the pass move is considered last instead of first?

    b) With this change, is 2x2 go solved faster or slower? Why?

    c) In `go/tromp.c`, why does Tromp write code that depends on bit manipulation, instead of writing code that is easier to understand?

    d) What changes would you have to make to `go/tromp.py` so that it solves 3x3 go instead of 2x2 go?

16. What is the minimax value of 1×2 go?

17. For 3×3 Go, prove that if a player gets a middle-3 shape (center line 3-in-a-row) then they can win by 9.

18. Solve the pinwheel problem for 3×3 go discussed in the following pdf. Assume positional superko rules. Give who wins and the principal variation.

`https://webdocs.cs.ualberta.ca/~hayward/355/ssgo.pdf`

19. In the pdf for the preceding question, for each of the three $1{\times}5$ linear go principal variations shown, prove that the minimax score for that variation is correct.

hints

1. (from `tt24.py`) corner: center center: any corner a2: a1, b2, c2, a3

2. a) in ttt, game can end before board is full
   b) $1 + 9 + 9*8 + 9*8*7 + \ldots + 9*8*7*6*5*4*3*2*1 = 986410$

3. a) 549946 b) 16168 c) 58524 d) 8307

4. a) yes b) draw c) number of nodes in tree of all continuations, with wins never detected (so continues until board is full and returns loss). 986410, confirms our answer from the previous question d) 19108 e) 101648 f) 10344

5. a) 94978 b) 9973 c) 2740

6. a) 67182 b) 8866 c) 2458

7. see `ttt/dev/tt24bias.py`

8. a) less b) 1

9.
```
29   1        answer  1 1 1 0 1
14   0
 7   1        check  16 + 8 + 4 + 0 + 1 = 29 :)
 3   1                  or use method below
 1   1
```

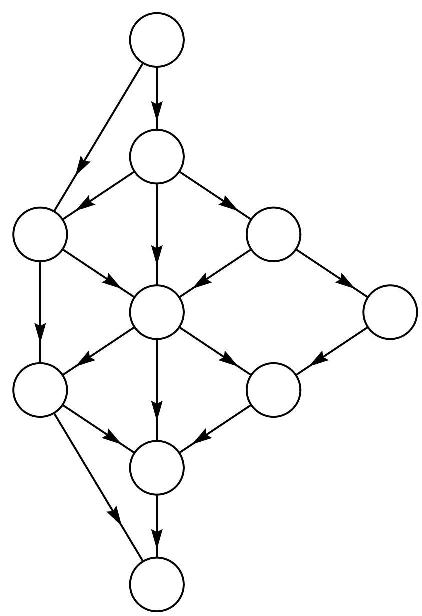
10.

```
1   1       check ->  187 1
0   2                  93 1
1   5                  46 0
1  11                  23 1
1  23                  11 1
0  46                   5 1
1  93                   2 0
1 187  <- answer        1 1   10111011 checked :)
```

11. check your answer with `nimbig.py`

12. There is a winning move from a pile if the binary representation of the pile size has a 1 in the same bit position as the left-most 1 in the binary representation of the xorsum of all the piles. For there to be exactly two winning moves, there would have to be exactly two piles with a 1 in this bit position, but that would make the xorsum of the piles 0 for that bit position, which contradicts the fact that the xorsum there is 1. Therefore, it's not possible to have exactly 2 winning moves.

13. a) see also `https://webdocs.cs.ualberta.ca/~hayward/355/jem/nim.html#sol`, where the dag is drawn sideways



b) {} 1, {1} 2, {1,1} 3, {2} 4, {1,1,1} 4, {1,2} 10, {1,1,2} 18, {2,2} 15, {1,2,2} 44, {2,2,2} 60

14. a) nim game, recursive algorithm with memoization, show the calls

```
b) nim game pile sizes (eg. 3 5 7)   1 2 2
 (1, 2, 2)
   (0, 2, 2)
     (0, 0, 2)
       (0, 0, 0)
       (0, 0, 0) lose
     (0, 0, 2) win
     (0, 1, 2)
       (0, 0, 2)
       (0, 0, 1)
         (0, 0, 0)
       (0, 0, 1) win
       (0, 1, 1)
         (0, 0, 1)
       (0, 1, 1) lose
     (0, 1, 2) win
   (0, 2, 2) lose
 (1, 2, 2) win
winning move to  (0, 2, 2)   10 calls
```

c) pile sizes are sorted at each step, so same as a)

```
d) nim game pile sizes (eg. 3 5 7)   1 2 2
 (1, 2, 2)
   (0, 1, 2)
     (0, 0, 1)
       (0, 0, 0)
       (0, 0, 0) lose
     (0, 0, 1) win
     (0, 1, 1)
       (0, 0, 1)
     (0, 1, 1) lose
```

```
    (0, 1, 2) win
    (1, 1, 2)
      (0, 1, 1)
    (1, 1, 2) win
  (1, 2, 2) lose
losing, 8 calls
```

15. originally, each computer had its own instruction set. the introduction of assembly language programming (applicable to multiple machines) was a big step forward. the next big step forward was to the more human-readable language FORTRAN (at the time programmers didn't believe that such a language's compiled code would run within a factor of 2 of hand-written machine code: they were wrong). progressively more human-readable languages came along (yay python :), but, especially for small programs, some programmers enjoy writing code in a language close to machine level such as C. John Tromp participated in competitions for writing deliberately unreadable (obfuscated) code. he is obviously an expert programmer: maybe he thinks that his 2x2 C code *is* human-readable :)

16. see `https://webdocs.cs.ualberta.ca/~hayward/355/ssgo.pdf`

17. see `https://webdocs.cs.ualberta.ca/~hayward/355/ssgo.pdf`

18. left for you

19. left for you