Move Ordering & Nim

CMPUT 355: Games, Puzzles, and Algorithms

Lecture Outline

1. Logistics & Recap
2. Move ordering for pruning
3. Nim introduction

Logistics

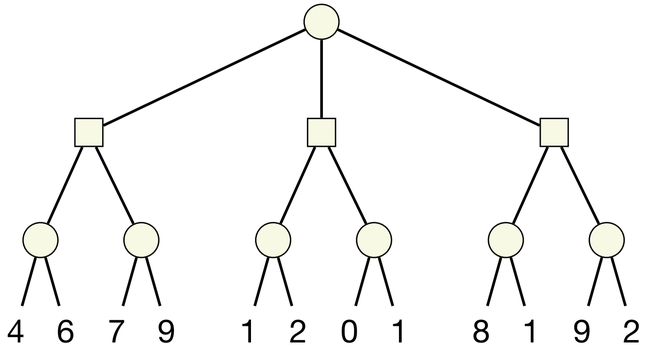
- **Quiz 2** is being marked
  - Scans of your exam will be attached to the Canvas "assignment"
- **Quiz 1:** deadline for **regrading requests** is **today** (Feb 13)
- **Practice questions #3** will be released **today** (Feb 13)
- **Reading week: No classes** next week (Feb 16 -- Feb 20)

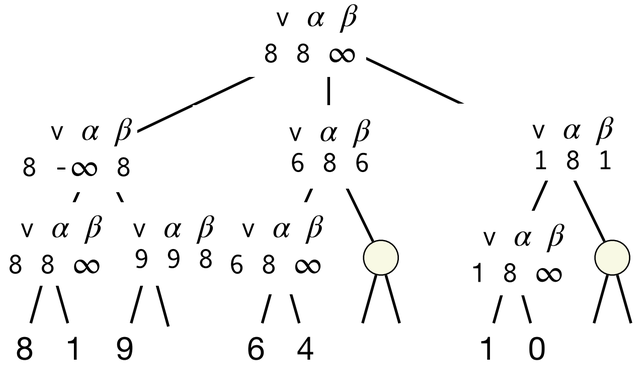
Pruning

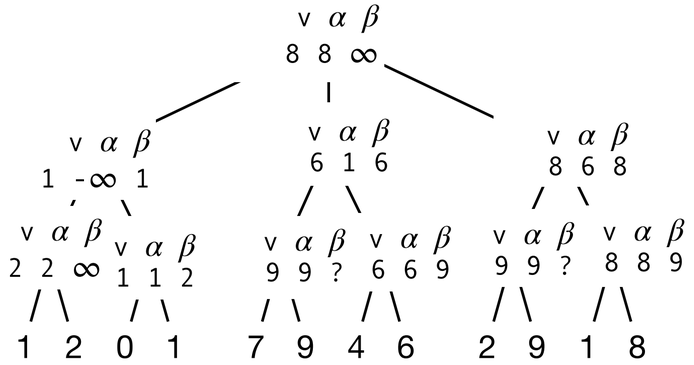
- **Transpositions:** Many positions can be reached by **multiple paths**
  - When we encounter one, we can simply **return the previously-computed value**
- **Isomorphisms:** Many positions are **structurally equivalent** (**isomorphic**)
  - E.g., playing in any corner is in some sense the same move
  - So we only need to evaluate **non-isomorphic children**
    - since any two isomorphic children will have exactly the same value
  - Convert each position to an isomorphic **canonical form**
  - Only explore the canonical forms
- **Alpha-beta pruning:** avoid exploring subtrees that will **provably** not be played in **optimal play**

Exploration Order Matters

- The **order** in which moves are explored changes how much pruning is possible
- There always exists an order that admits **no pruning at all!**
- Exploring the **best children first** (e.g., smallest minmax from a **min** node, largest minmax from a **max** node) yields the **most pruning** (**why?**)





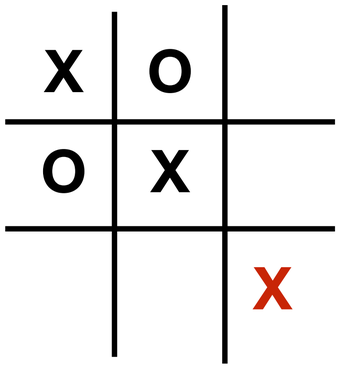


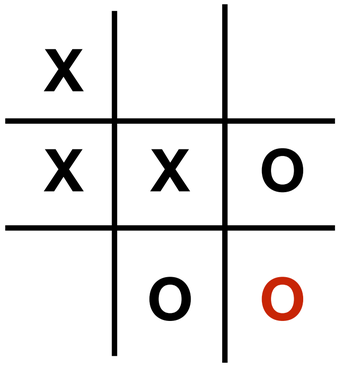
Heuristics: Exploiting Move Ordering

- So, an additional way to speed up alpha-beta search is by using heuristics to predict which children will have highest/lowest minmax value
- Explore children in **decreasing** order of **predicted minmax value** (from **max** node) or **increasing** order of predicted minmax value (from **min** node)
  - **Question:** What order to use in negamax?
- One class of extremely common heuristic: **Terminal detection**
  - i.e., explore children who are terminal nodes first
  - **Question:** Why is this useful in general?
  - **Question:** When is this **extra** useful?

Tic-tac-toe: Win-Threat Detection

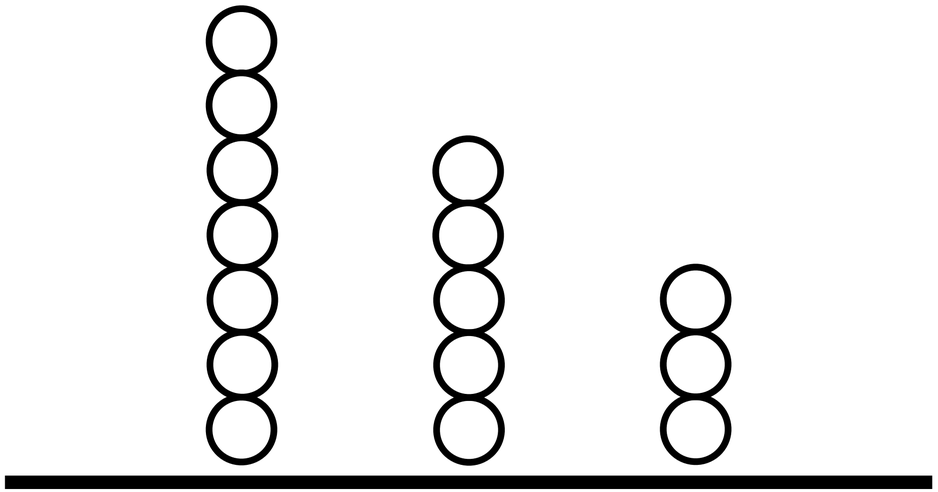
- In tic-tac-toe, it's easy to detect **win-threats** from a position
  - i.e., moves that will enable an immediate win
  - Explore those **first,** because other moves won't matter (**why?**)
- If there are no win-threats, next look for **lose-threats**
  - i.e., moves that will allow the opponent an immediate win
  - Explore the blocking move first (i.e., a move that will prevent the opponent from making their winning move)
  - Other moves don't matter when a lose-threat exists (**why?**), so explore it first





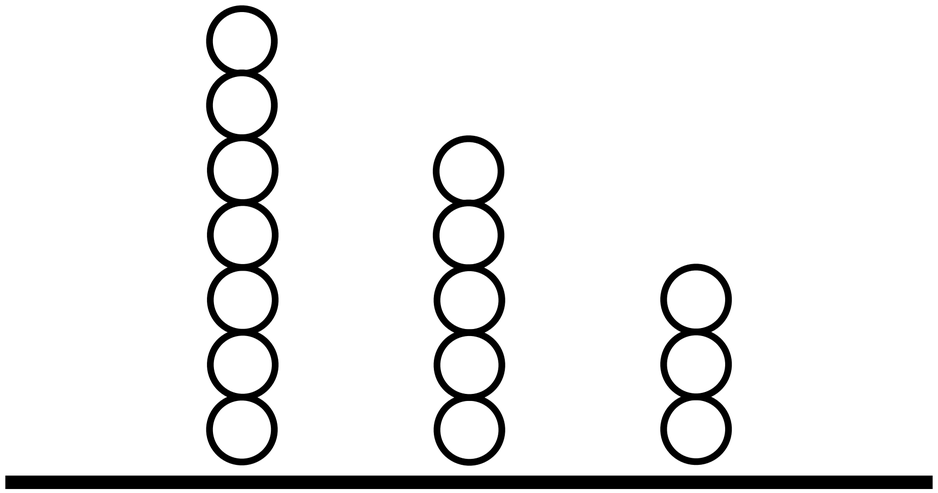
Nim

- There are k piles of stones
- A move is removing some **positive** number of stones from a **single pile**
- Players alternate moves
- Last player to move wins
  - i.e., if there are **no stones left** and you are player-to-move, you lose
- Example at right is position $7,5,1$
- **Question:** How many **positions** are reachable from a position with k piles of n stones each?
- **Question:** Does Nim have **transpositions?**



Nim Isomorphisms

- **Question:** What are some equivalent positions to $7,5,3$?
- Any ordering of piles is equivalent
  - If I was going to take 3 stones from the 7-pile in $7,5,3$, I could just take 3 stones from the 7-pile in $5,3,7$
- **Question:** How many positions are equivalent to an arbitrary k-pile position?
- **Question:** What would be a good **canonical form** for Nim positions?



Nim Implementation: **nim/nimnega_v2.py**
Negamax + Transpositions + Isomorphisms

```python
def winning(nim_psn, sd, depth, verbose):
# tuple, dictionary, recursion depth, verbose mode (True/False)
  if nim_psn in sd:
    return sd[nim_psn]
  # nim_psn not in dictionary, so update before we return
  psn = tuple(sorted(nim_psn))
  for j in range(len(psn)): # each pile
    for k in range(psn[j]): # number of stones that will remain in that pile
      child = tuple(sorted(psn[:j] + (k,) + psn[j+1:]))
      if not winning(child, sd, depth+1, verbose):
        sd.update({ nim_psn: True })   # update before return
        return True
  sd.update({ nim_psn: False })  # update before return
  return False
```

Demo:

% time python3 nimnega_v2.py < 3-5-7.in
% time python3 nimnega_v2.py < 10-10s.in
% time python3 nimnega_v2.py < 20-20s.in

Nim: What About Move Ordering?

- Transposition tables and isomorphism pruning get us a long way in Nim
- But it's easy to construct large games that are still intractable
- Could we apply move ordering to speed things up?
- **Question:** what would be a good move ordering to use?
- *Next time:* A move ordering that is good enough to prove a win for **Nim(20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20, 20)**

Summary

- **Alpha-beta** prunes more or less depending on the **order** it explores children
  - Worst-case: **no pruning at all**
  - Ideally: explore **promising children first** before unpromising ones
- **Tic-tac-toe:**
  - **Win-threat** and **loss-threat** detection for searching important subtrees first
- **Nim:** a game with a very large search space
  - Factorially reduced by focusing on canonical form: sort piles by height
- Isomorphisms and transposition tables are powerful for Nim
  - But **not powerful enough** for every possible input