Mid Term Exam (Fall 2014)

- **Q(1) (2 points)** Decide if each of the following is true or false. Provide an brief explanation of your answer.
- 1- A system must think like a human in order to pass the Turing Test reliably.
- False, It just has to act humanly

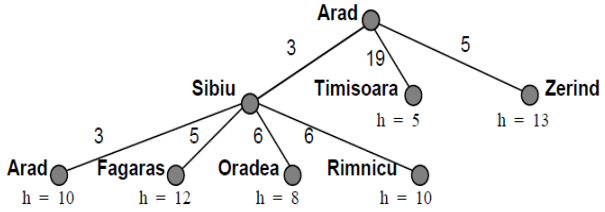
2- An agent that senses only partial information about the state cannot be perfectly rational.

- False, rationality and omniscience are two different concepts

3- Breadth-First search is complete if the state space has infinite depth but finite branching factor.

- True 4- Both Greedy and Depth-Limited Search are optimal.
- False: Greedy is not optimal

- **Q(2) (2 points)** The following diagram shows a partially expanded search tree. Each arc is labeled with the corresponding step cost, and the leaves are labeled with the h values.



1- Which leaf will be expanded next by a greedy search?

Timisora

2- Which leaf will be expanded by an A* search?

Arad

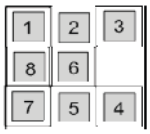
3- If the goal can be reached only from Fagaras at the fourth level. Which blind algorithm BFS or DFS will reach the goal first. Explain why?

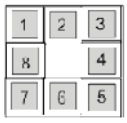
- BFS

- **Q(3) (2 points)** What is a rational agent? And what is bounded rational agent?
- Ans. A rational agent always selects an action based on the percept sequence it has received so as to maximize its (expected) performance measure given the percepts it has received and the knowledge possessed by it.
- . A rational agent that can use only bounded resources cannot exhibit the optimal behaviour. A bounded rational agent does the best possible job of selecting good actions given its goal, and given its bounded resources.
-

- **Q(4) (2 points)** n cars occupy squares (1, 1) through (1, n) (i.e., the top row) of an n x n grid. The cars are marked from 1 to n. The cars must be moved to the bottom row but in reverse order. On each time step, every one of the n cars can move one square up, down, left, or right.
- Give a complete problem formulation.(i.e. description of the state space, initial state, goal state(s), successor function)

- **Q(5) (2 points)** Consider the following 8-puzzel states, a solved one is on the left:





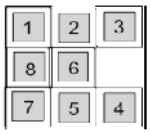
- Given a puzzle state like the one on the left, where the numbers are in the wrong places, we want to search for a series of moves which ends in the solution above, on the right. Consider the trivial heuristic function, h, which finds the number of tiles out of their position
- What is the value of h for the above board state?

the numbers 4, 5 and 6 are out of place. function would be 3

- In a greedy search, what move would be chosen next?
- Moving the 6 makes *h=3* the same Moving the 3 makes h=4 Moving the four up would reduce the value of *h* to just two

Therefore, a greedy searching agent would <u>move the four up.</u>

- **Q(5) (2 points)** Consider the following 8-puzzel states, a solved one is on the left:



- Consider the trivial heuristic function, h, which finds the number of tiles out of their position
- Is this heuristic admissible? Can you suggest a better heuristic measure?
- Each out of place number must be moved at least once to get it into its final position, but in reality it is to be moved more than once. Hence the number of out-of-place pieces underestimates the path cost, and hence is admissible. You may suggest the Manhattan one

Game Playing

A (pure) strategy:

a complete set of advance instructions that specifies a definite choice for every conceivable situation in which the player may be required to act.

In a two-player game, a strategy allows the player to have a response to every move of the opponent.

Game-playing programs implement a strategy as a software mechanism that supplies the right move on request.

Two-Person Perfect Information Deterministic Game

- Two players take turns making moves
- Call one Min and the other Max
- Deterministic moves: Board state fully known,
- One player wins by defeating the other (or else there is a tie)
- Want a strategy to win, assuming the other person plays rationally

A logic-based approach to games

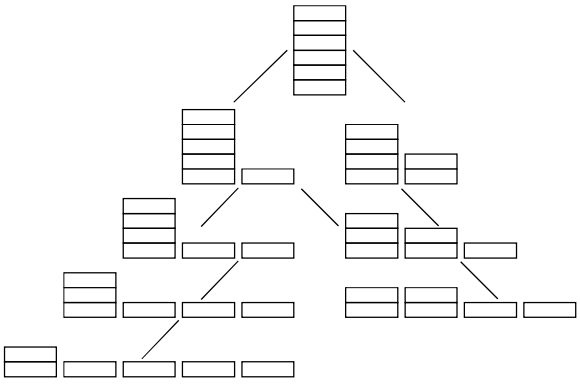
Find a winning strategy by *proving* that the game can be won -- use backward chaining.

A very simple game: nim.

- initially, there is one stack of chips;
- a move: select a stack and divide it in two unequal non-empty stacks;
- a player who cannot move loses the game.

(The player who moves first can win.)

Nim



Static evaluation

A **static evaluation function** returns the value of a move without trying to play (which would mean simulating the rest of the game but not playing it).

"Usually" a static evaluation function returns positive values for positions advantageous to **Player 1**, negative values for positions advantageous to **Player 2.**

If **Player 1** is rational, he will choose the maximal value of a leaf.
Then, **Player 2** will choose the minimal value.

Static evaluation

If we can have (guess or calculate) the value of an internal node **N**, we can treat it as if it were a leaf. This is the basis of the minimax procedure.

No tree would be necessary if we could evaluate the initial position statically. Normally we need a tree, and we need to look-ahead into it. Further positions can be evaluated more precisely, because there is more information, and a more focussed search.

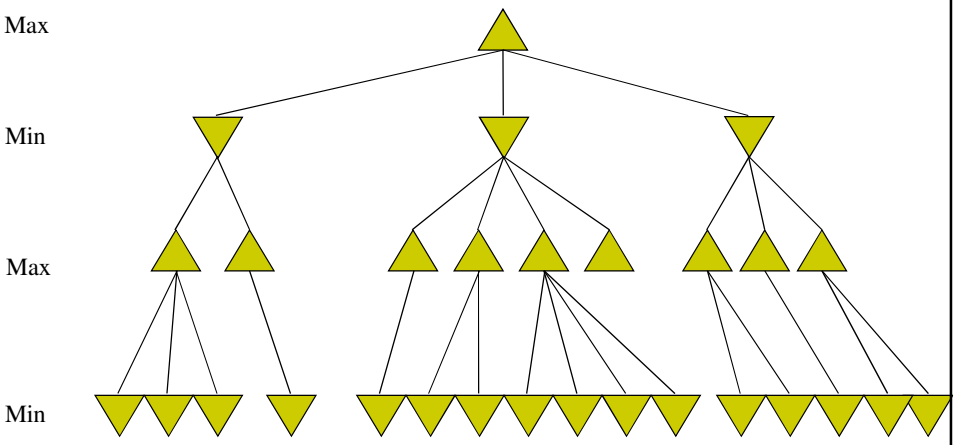
Minimax Tree

- Create a utility function
  - Evaluation of board/game state to determine how strong the position of each player.
  - Player 1 wants to maximize the utility function
  - Player 2 wants to minimize the utility function
- Minimax tree
  - Generate a new level for each move
  - Levels alternate between “max” (player 1 moves) and “min” (player 2 moves)

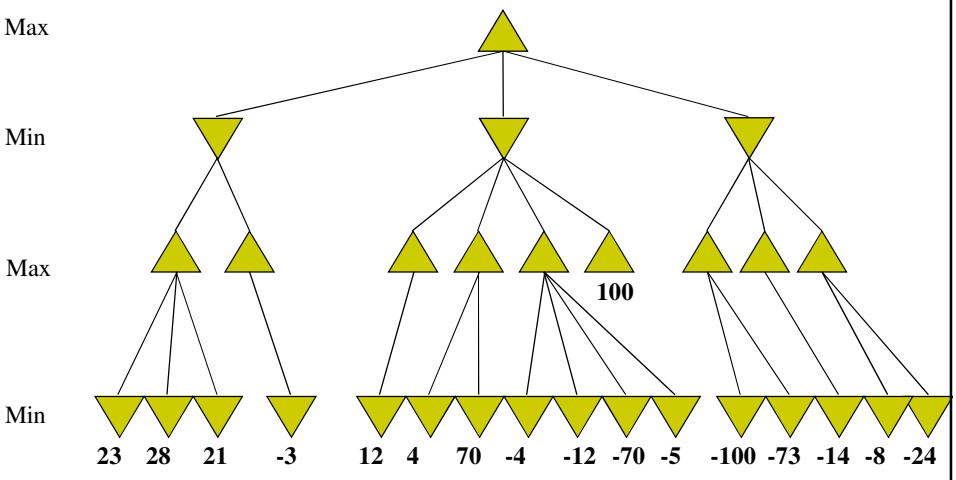
Minimax Tree Evaluation

- Assign utility values to leaves
  - If leaf is a “final” state, assign the maximum or minimum possible utility value (depending on who would win)
  - If leaf is not a “final” state, must use some other heuristic, specific to the game, to evaluate how good/bad the state is at that point

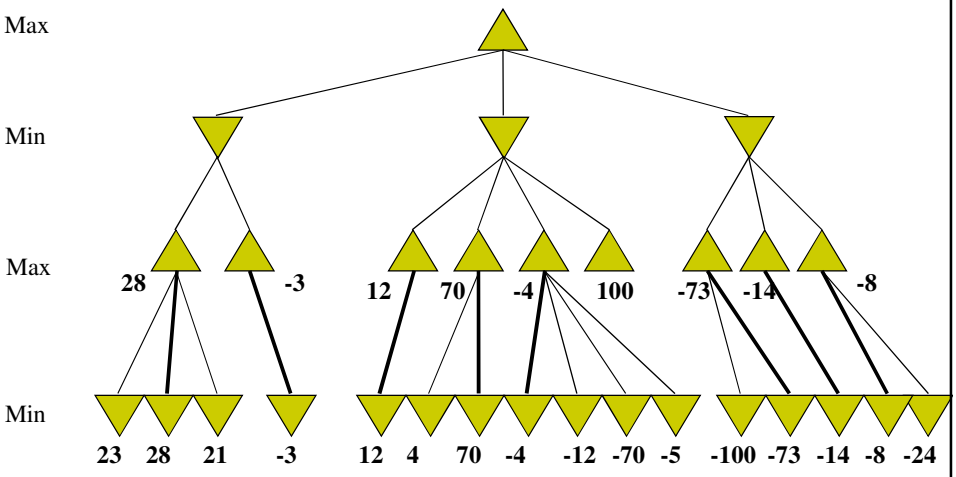
Minimax tree



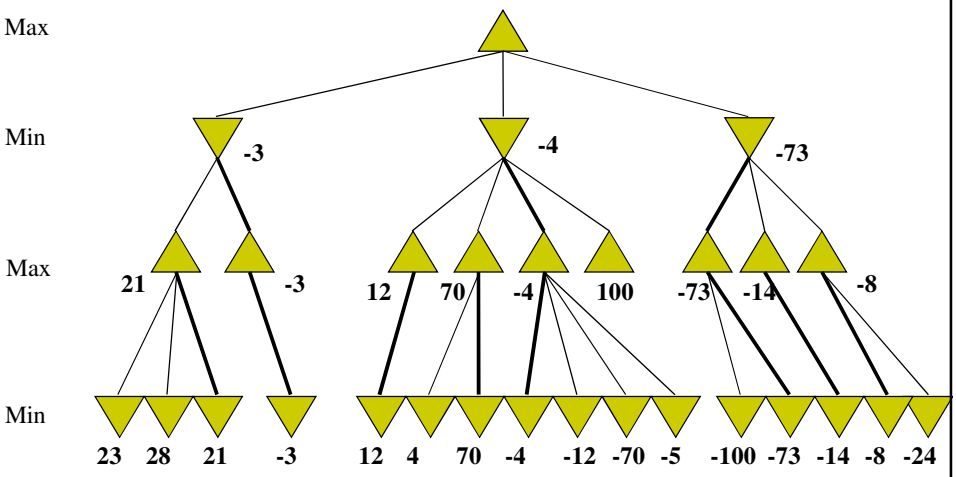
Minimax tree



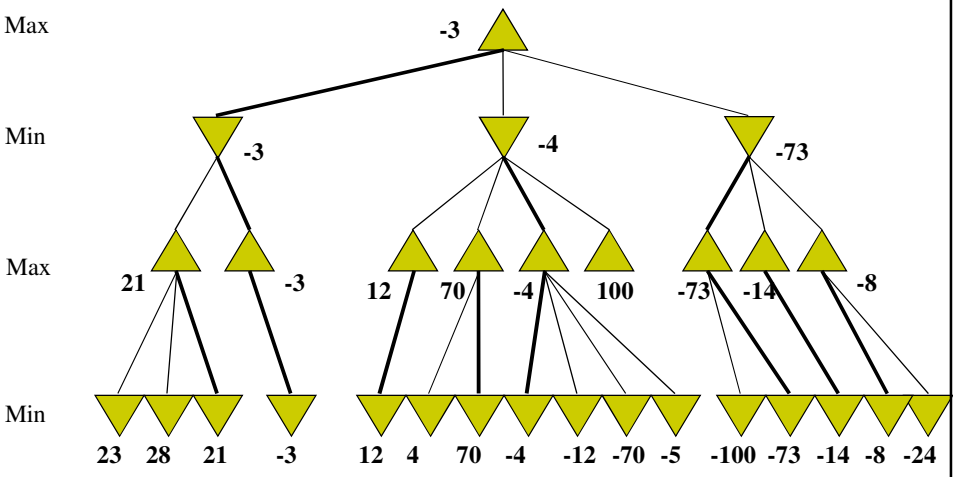
Minimax tree



Minimax tree



Minimax tree



Tic-Tac-Toe

Let player **A** be x and let open(x), open(o) mean the number of lines open to x and o. There are 8 lines. An evaluation function for position P:

f(P) = -∞ if o wins

f(P) = +∞ if x wins, otherwise

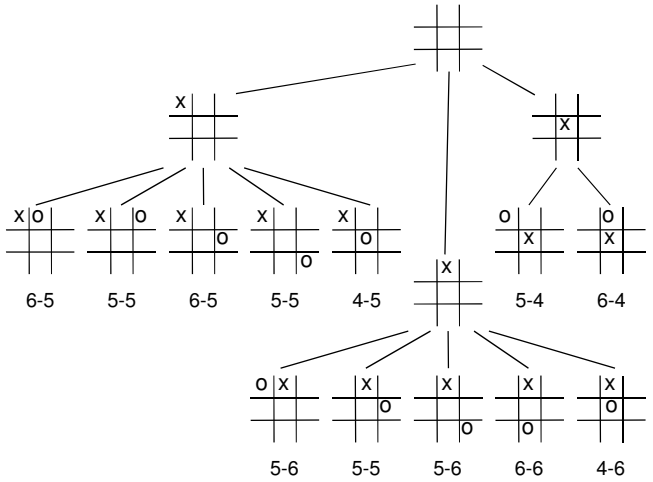
f(P) = open(x) - open(o)

Example:

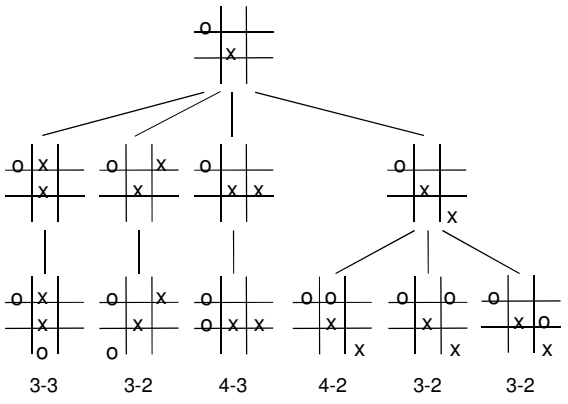
open(x) - open(o) = 4 - 6

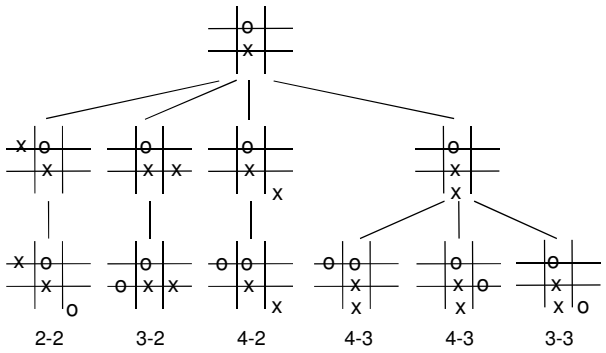
Assumptions:

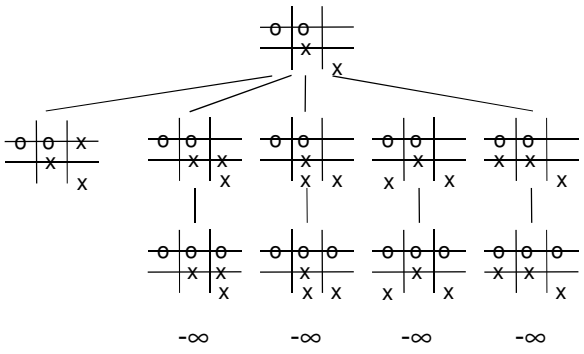
only one of symmetrical positions is generated;



Player **B** chooses the minimal backed-up value among level 1 nodes.
Player **A** chooses the maximal value, and makes the move.
Player **B**, as a rational agent, selects the optimal response.







Building complete piles is usually not necessary. If we evaluate a position when it is generated, we may save a lot.

Assume that we are at a minimizing level. If the evaluation function returns -∞, we do not need to consider other positions:
-∞ will be the minimum.

The same applies to +∞ at a maximizing level.