Exercise 4.1

2 + 2 + 2 + 2 + 2 = 10pts

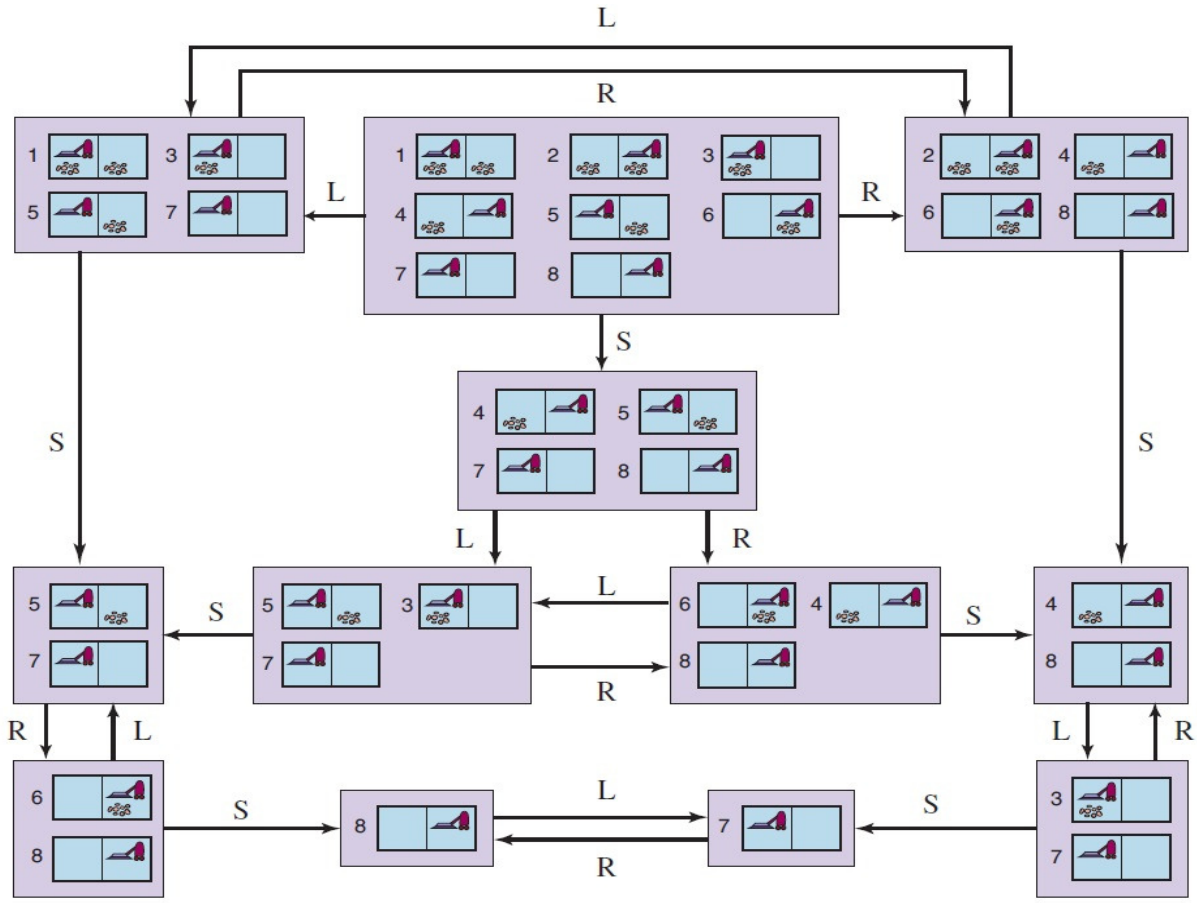
Give the name of the algorithm that results from each of the following special cases:

1. Local beam seach with $k = 1$
2. Local beam search with one initial state and no limit on the number of states retained.
3. Simulated annealing with $T = 0$ at all times (and omitting the termination test).
4. Simulated annealing with $T = \infty$ at all times.
5. Genetic alorithm with population size $N = 1$

Exercise 4.7

15pts

In Section conformant-section we introduced belief states to solve sensorless search problems. A sequence of actions solves a sensorless problem if it maps every physical state in the initial belief state b to a goal state. Suppose the agent knows $h^*(s)$, the true optimal cost of solving the physical state s in the fully observable problem, for every state s in b. Find an admissible heuristic $h(b)$ for the sensorless problem in terms of these costs, and prove its admissibilty. Comment on the accuracy of this heuristic on the sensorless vacuum problem of Figure **vacuum2-sets-figure**. How well does A* perform?



vacuum2-sets-figure

Exercise 4.8 [belief-state-superset-exercise]

5 + 5 = 10pts

This exercise explores subset–superset relations between belief states in sensorless or partially observable environments.

1. Prove that if an action sequence is a solution for a belief state b, it is also a solution for any subset of b. Can anything be said about supersets of b?
2. Explain in detail how to modify graph search for sensorless problems to take advantage of your answers in (a).

Exercise 4.10 [vacuum-solvable-exercise]

5pts

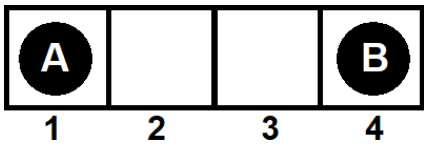
Consider the sensorless version of the erratic vacuum world. Draw the belief-state space reachable from the initial belief state $1, 2, 3, 4, 5, 6, 7, 8$, and explain why the problem is unsolvable.

Exercise 5.8

6 + 3 + 5 + 6 = 20pts

Consider the two-player game described in Figure **line-game4-figure**.

1. Draw the complete game tree, using the following conventions:
   - Write each state as (s_A, s_B), where s_A and s_B denote the token locations.
   - Put each terminal state in a square box and write its game value in a circle.
   - Put *loop states* (states that already appear on the path to the root) in double square boxes. Since their value is unclear, annotate each with a "?" in a circle.
2. Now mark each node with its backed-up minimax value (also in a circle). Explain how you handled the "?" values and why.
3. Explain why the standard minimax algorithm would fail on this game tree and briefly sketch how you might fix it, drawing on your answer to (b). Does your modified algorithm give optimal decisions for all games with loops?
4. This 4-square game can be generalized to n squares for any $n > 2$. Prove that A wins if n is even and loses if n is odd.



line-game4-figure: The starting position of a simple game. Player A moves first. The two players take turns moving, and each player must move his token to an open adjacent space in either direction. If the opponent occupies and adjacent space, then a player may jump over the opponent to the next open space if any. (For example, if A is on 3 and B is on 2, then A may move back to 1.) The game ends when one player reached the opposite end of the board. If player A reaches space 4 first, then the value of the game to A is $+1$; if player B reaches space 1 first, then the value of the game to A is -1.

Exercise 5.9

3 + 4 + 3 + 4 + 4 = 18pts

This problem exercises the basic concepts of game playing, using tic-tac-toe (noughts and crosses) as an example. We define X_n as the number of rows, columns, or diagonals with exactly n X's and no O's. Similarly, O_n is the number of rows, columns, or diagonals with just n O's. The utility function assigns $+1$ to any position with $X_3 = 1$ and -1 to any position with $O_3 = 1$. All other terminal positions have utility 0. For nonterminal positions, we use a linear evaluation function defined as $Eval(s) = 3X_2(s) + X_1(s) - (3O_2(s) + O_1(s))$.

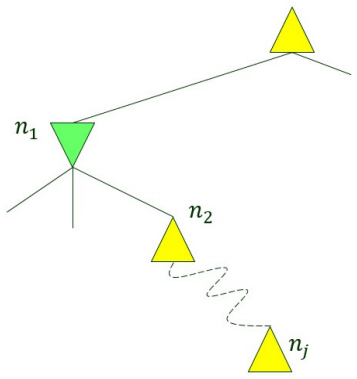
1. Approximately how many possible games of tic-tac-toe are there?
2. Show the whole game tree starting from an empty board down to depth 2 (i.e., one X and one O on the board), taking symmetry into account.
3. Mark on your tree the evaluations of all the positions at depth 2.
4. Using the minimax algorithm, mark on your tree the backed-up values for the positions at depths 1 and 0, and use those values to choose the best starting move.
5. Circle the nodes at depth 2 that would *not* be evaluated if alpha–beta pruning were applied, assuming the nodes are generated in the optimal order for alpha–beta pruning.

Exercise 5.14

4 + 4 + 4 + 5 = 17pts

Develop a formal proof of correctness for alpha–beta pruning. To do this, consider the situation shown in Figure **alpha-beta-proof-figure**. The question is whether to prune node n_j, which is a max-node and a descendant of node n_1. The basic idea is to prune it if and only if the minimax value of n_1 can be shown to be independent of the value of n_j.

1. Mode n_1 takes on the minimum value among its children:
   $n_1 = min(n_2, n_{21}, ..., n_{2b_2})$. Find a similar expression for n_2 and hence an expression for n_1 in terms of n_j.
2. Let l_i be the minimum (or maximum) value of the nodes to the *left* of node n_i at depth i, whose minimax value is already known. Similarly, let r_i be the minimum (or maximum) value of the unexplored nodes to the right of n_i at depth i. Rewrite your expression for n_1 in terms of the l_i and r_i values.
3. Now reformulate the expression to show that in order to affect n_1, n_j must not exceed a certain bound derived from the l_i values.
4. Repeat the process for the case where n_j is a min-node.



alpha-beta-proof-figure: Situation when considering whether to prune node n_j.

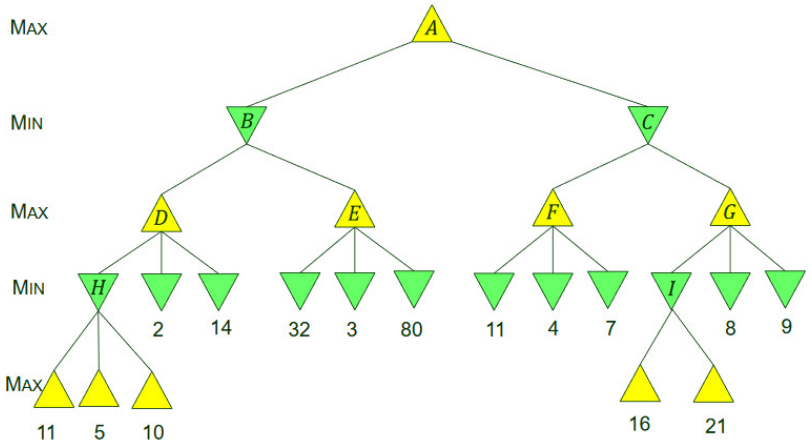
Extra Problem

5 + 7 + 3 = 15pts

You are given a **minimax search tree** as shown on the next page. The tree has nine internal nodes $A, B, ..., I$. Not all terminal states (leaves) are at the same depth.

Execute the alpha-beta pruning algorithm (use the version from the 3rd edition of the textbook in the reposted lecture notes on September 11)

a. Mark all the subtrees (including leaves) that have been pruned. You may, for instance, simply put double slashes // across the edge entering the root of such a subtree from the above.
b. Next to each internal node, write down the two values $[\alpha, \beta]$ just before the return from the call `MAX-VALUE` or `MIN-VALUE` invoked on the state represented by the node.
c. What is the final value for `MAX` at the root?



minmax search tree