Game Tree Search

Basic algorithm for deciding moves in a 2-player game is called the MiniMax Algorithm. The idea is to search the game tree down to a certain depth and then assign a numerical score to each board position that results in a positive score ⇒ position favorable to computer and negative score ⇒ position favorable to opponent.

Example:

```
   19
  /   \
19  -8
 /   /   \
13 9 6  -8 -15 3 5
```

How should we decide values of positions at this level? Since it is the computer's move and our alg. is operating on behalf of the computer, pick the best move (maximum value from the next level) down.
how do we decide value at next level up?

Since the opponent has the choice at that level, we assume we are playing an intelligent opponent who will pick the best move for them, so pick the minimum value from the next level down.

At top level, it is computer's choice again, so pick the maximum.

So, root node is assigned value '0' and the computer picks the move associated with the left branch.

This idea of maximizing at levels where the computer is moving and minimizing at levels where the opponent is moving is the Minimax Algorithm.

**Tic Tac Toe Example**

+1 = win for computer
0 = draw
-1 = win for opponent
we can modify basic Minimax to avoid searching some subtrees when they cannot change the outcome. This is called the Alpha-Beta Pruning Algorithm.

Examples where pruning can occur:

- In the first tree, if \( \gamma \geq 2 \), the computer can choose a move that leads to a loss, so we can prune the subtree to its left.
- In the second tree, if \( \gamma \leq 1 \), the opponent can make a move that leads to a loss, so we can prune the subtree to its right.
- In the third tree, if \( \gamma \leq -3 \), the opponent can make a move that leads to a loss, so we can prune the subtree to its left.

We had this situation in our tic-tac-toe example:
So, MiniMax Alg. w/ Alpha-Beta Pruning is a slightly more efficient than basic MiniMax (exhaustive search down to a certain level)

additional enhancements:

- timed moves ⇒ use progressive deepening

  1st search 2 levels
  then search 3 levels
  then search 4 levels

  when time expires, pick best move from last level finished

since tree grows exponentially, each level takes as much
time as all previous levels combined, so we don't lose
much in efficiency by doing this

- heuristic pruning can speed up search by a lot
  but does not guarantee best move as MiniMax w/ Alpha-Beta does ⇒ idea is to focus on
  a few moves at every level, danger is that in some
  very good moves (Queen sacrifice) will not be explored
  or very bad moves will be selected (because we didn't
  explore the opponent's move that made the choice really bad)

Note about tic-tac-toe: total no. of possible games < 9! = 362,880
(<, not =, because some games end before 9 moves are made)

w/ this no., a MiniMax Alg. can easily search the entire space of
pitches and play "perfect" tic-tac-toe, i.e. best opponent could hope for is a draw.
Circuit Partitioning

Given a circuit composed of multiple components (wires) with interconnections between them, partition circuit into 2 or more parts such that the wires going from one part to another are minimized (could model wires going between chips or between boards in a system).

Simplest case - 2 parts

Model circuit as a graph:

```
   a ---- 10 ---- b
   |      |      |
   v      |      v
  5      8     5
  |      |      |
  c ---- 7 ---- d
  |      |      |
  v      |      v
  f ---- 10 ---- g
  |      |      |
  v      |      v
  15     8     2
  |      |      |
  h ---- e
```

Find min-cut between A, B such that |A| = |B| (as close as possible), A \cap B = \emptyset, and A \cup B = V

Kernighan and Lin Algorithm

pick random initial partitioning
iteratively improve partitioning (reducing weight of cut)
by swapping a pair of vertices between A, B at each iteration (once swapped, vertices must stay where they are to prevent infinite loops)
swapping vertex pairs maintains |A| = |B| at each iteration
this is a heuristic "hill-climbing" procedure

optimization objective

iterative improvement

possible solutions

initial solution

find solution - local maximum, not necessarily global maximum

min-cut is an NP-hard problem, so heuristic is OK

K-L Alg. Ex.-

Suppose initial partition is: \( A = \{a, b, c, d, e\} \), \( B = \{f, g, h\} \)

![Graph diagram]

Initial cut weight = \( 2 + 7 + 10 + 5 + 8 = 42 \)

Consider every possible \( x, y \) swap where \( x \in A \), \( y \in B \)

How many possible swaps are there? \( 4 \times 4 = 16 \) (in general, \( \frac{n}{2} \times \frac{n}{2} = \frac{n^2}{4} \) )
1st iteration:

<table>
<thead>
<tr>
<th>x,y</th>
<th>net change to cut weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>a,e</td>
<td>+5+10-2 +2-8 = +7</td>
</tr>
<tr>
<td>a,f</td>
<td>+5+10 +6 = +21</td>
</tr>
<tr>
<td>a,g</td>
<td>+5+10-2 +6+8 -7-10 = +10</td>
</tr>
<tr>
<td>a,h</td>
<td>+5+10-2 +2+8-15 = +8</td>
</tr>
<tr>
<td>b,e</td>
<td>+10+8 +2-8 = +12</td>
</tr>
<tr>
<td>b,f</td>
<td>+10+8 +6-2 = +22</td>
</tr>
<tr>
<td>b,g</td>
<td>+10+8 +6+8 -7-10 = +15</td>
</tr>
<tr>
<td>b,h</td>
<td>+10+8 +2+8-15 = +13</td>
</tr>
<tr>
<td>c,e</td>
<td>+5+5-7 +2-8 = -3</td>
</tr>
<tr>
<td>c,f</td>
<td>+5+5-7 +6-2 = +7</td>
</tr>
<tr>
<td>c,g</td>
<td>+5+5 +6+8 -10 = +14</td>
</tr>
<tr>
<td>c,h</td>
<td>+5+5-7 +2+8-15 = -2</td>
</tr>
<tr>
<td>d,e</td>
<td>+8+5 -15-10 +2 = -10</td>
</tr>
<tr>
<td>d,f</td>
<td>+8+5 -8-15-10 +6-2 = -16</td>
</tr>
<tr>
<td>d,g</td>
<td>+8+5 -8-15 +8+6-7 = -3</td>
</tr>
<tr>
<td>d,h</td>
<td>+8+5 -8-10 +2+8 = +5</td>
</tr>
</tbody>
</table>

swap d & f:

Cut weight = 6+7+5+8 = 26
2nd iteration: d and f are fixed (only 3x3 = 9 possible swaps left)

<table>
<thead>
<tr>
<th>x, y</th>
<th>not change</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, e</td>
<td>+10 + 5 + 2 + 8 + 2 = +27</td>
</tr>
<tr>
<td>a, g</td>
<td>+10 + 5 + 2 + 8 - 10 - 7 - 6 = +22</td>
</tr>
<tr>
<td>a, h</td>
<td>+10 + 5 + 2 + 2 + 15 + 8 = +42</td>
</tr>
<tr>
<td>b, e</td>
<td>+10 - 8 + 8 + 2 = +12</td>
</tr>
<tr>
<td>b, g</td>
<td>+10 - 8 + 8 - 10 - 7 - 6 = +7</td>
</tr>
<tr>
<td>b, h</td>
<td>+10 - 8 + 2 + 15 + 8 = +27</td>
</tr>
<tr>
<td>c, e</td>
<td>+5 - 5 - 7 + 8 + 2 = +3</td>
</tr>
<tr>
<td>c, g</td>
<td>+5 - 5 + 8 - 10 - 6 = +12</td>
</tr>
<tr>
<td>c, h</td>
<td>+5 - 5 - 7 + 2 + 15 + 8 = +18</td>
</tr>
</tbody>
</table>

no more beneficial swaps,

final partition = \{ a, b, c, f \} \cup \{ d, e, g, h \}

if we want to partition into 2^k parts, i.e. 4, 8, 16, ... then we can hierarchically apply K-L Alg.

1st, partition into 2
then, partition each of the 2 \( \Rightarrow \) 4 parts
then, partition each \( \Rightarrow \) 8 parts

if we want \( p > 2 \) parts, where \( p \) is not a power of 2, we need a different algo.