Documentation of Two Algorithms for Adversarial Searching Games

CSCE625 Programming Assignment 2

Bowen Wang, 223003296

October 08, 2014

Abstract

In a game which allows two player removes matches in turn, the one who is left with one match loses the game. One can remove one, two or three matches each time. Two algorithms are described and implemented to help the player make optimized strategy. They are called minimax algorithm and alpha-beta pruning algorithm.

1 Description of the Algorithm

1.1 The Minimax Algorithm

The minimax algorithm computes the minimax decision from the current state. It uses a simple recursive computation of alternating maximum or minimum to detect all the way through the tree path to calculate the utility values, and then make the optimized choice. It performs a depth-first search; for the depth m of the tree, b legal moves at each node, it has $O(b^m)$ time complexity and $O(bm)$ space complexity.

In this problem, the game result for the two players is either win or lose. Hence we can define the utility of the terminal point to be 1 for the win, and -1 for the lose. Considering for one player specifically, his goal for the game is to win, which is to get the maximum utility path from the root to the leaf. And then he can make the optimized decision.

Take a look at the following figure as an example.

When n=4, there are four levels. The player is at the root to choose maximum. He now has 3 choices. For the first branch 1, his maximum possible value is 1, which is the utility -1, because his opponent will has no choice but lose. For the second branch 2, his opponent will choose the minimum value from the next level, which we will see is the utility -1. For the third branch 3, his opponent

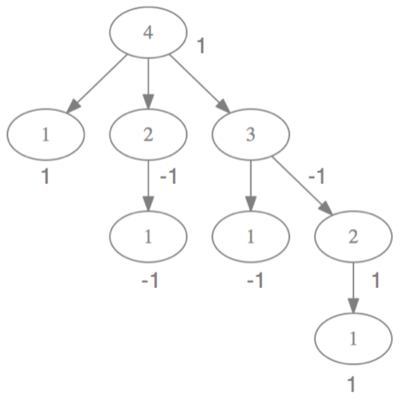

Figure 1: Minimax Tree for n=4

will do the same process to choose from the next level, which is 1 or the branch 2. Repeat this process, branch 3 will return -1. Hence we go back to the root, the player has three options: 1, -1, and -1. So his decision is to choose the first branch, because it has the maximum utility 1.

Below is how the algorithm is realized.

```
function MINIMAX-DECISION(state) returns an action
    return $argmax_{a \in ACTIONS(s)}$ MIN-VALUE(RESULT(state,a))

function MAX-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then returns UTILITY(state)
    $v \leftarrow -\infty$
    for each a in ACTIONS (state) do
        $v \leftarrow$ MAX(v, MIN-VALUE(RESULT(s,a)))
    return v

function MIN-VALUE(state) returns a utility value
    if TERMINAL-TEST(state) then returns UTILITY(state)
    $v \leftarrow \infty$
    for each a in ACTIONS (state) do
        $v \leftarrow$ MIN(v, MAX-VALUE(RESULT(s,a)))
    return v
```

1.2 Alpha-Beta Pruning Algorithm

Due to the high-volume searches of the Minimax Algorithm, we introduce an improved algorithm called Alpha-Beta Pruning, which eliminates large parts of the tree from consideration.

In this algorithm, the minimax decision are independent of the values of the pruned branches. Considering the player who takes the maximum decision, the possible choices are selected by his opponent by minimizing the utility values. For example, in the branch, there is a utility 3. Then if the player detects a path in the second branch with utility 2, the rest paths in the second branches are needless to search, which means the second branch is discarded. The reason is that once the second branch is chooses, his opponent will get the utility 2 or even smaller, which is bad decision for the player.

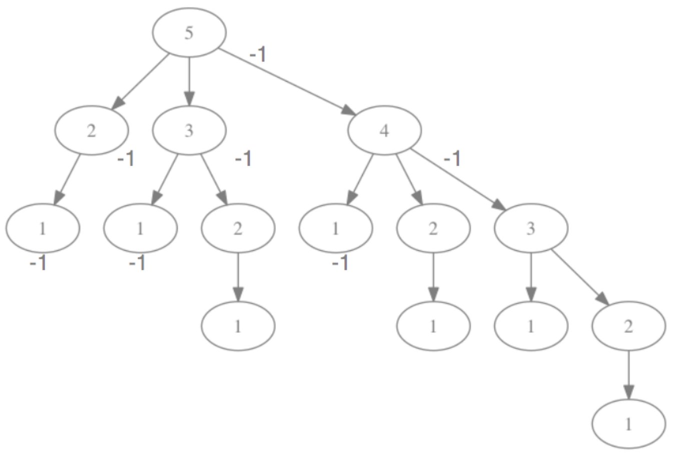

Figure 2: Minimax Tree pruned for n=5

In Figure 2, n=5 is considered for our match game. When the player searches the first branch 2, it gets the only choice of utility -1 from the next level. Then he searches the second branch 3. Below 3 the first path is 1 whose utility equals -1. Since it reaches the minimum possible utility, the rest paths of this branch are pruned. For the third branch 4 do the same process. He finds that the first path reaches utility -1 and hence all the three branches 2, 3, and 4 returns utility -1. The player has no option but to accept the utility -1 which leads him losing the game.

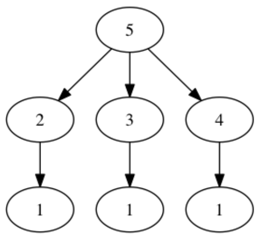

Figure 3: Alpha-Beta Tree for n=5

Then the alpha-beta pruning tree is formulated.

Below is how the algorithm is realized.

```
**function** ALPHA-BETA-SEARCH(*state*) **returns** *an action*
    $v \leftarrow$ MAX-VALUE(*state*, $-\infty, \infty$)
    **return** *the action* in ACTIONS(*state*) with value $v$

**function** MAX-VALUE(*state*) **returns** *a utility value*
    **if** TERMINAL-TEST(*state*) **then returns** UTILITY(*state*)
    $v \leftarrow -\infty$
    **for each** *a in* ACTIONS (*state*) **do**
        $v \leftarrow$ MAX($v$, MIN-VALUE(RESULT($s$,$a$), $\alpha, \beta$))
        **if** $v \geq \beta$ **then return** $v$
        $\alpha \leftarrow$ MAX($\alpha, v$)
    **return** $v$

**function** MIN-VALUE(*state*) **returns** *a utility value*
    **if** TERMINAL-TEST(*state*) **then returns** UTILITY(*state*)
    $v \leftarrow \infty$
    **for each** *a in* ACTIONS (*state*) **do**
        $v \leftarrow$ MIN($v$, MAX-VALUE(RESULT($s$,$a$), $\alpha, \beta$))
        **if** $v \leq \alpha$ **then return** $v$
        $\beta \leftarrow$ MIN($\beta, v$)
    **return** $v$
```

2 Visualization of the Trees

The trees are created by the python package graphviz 0.4.1 and visualized by the software Graphviz. The output are .gv files and can be translated into .png or .pdf files.

2.1 Minimax Tree n=7

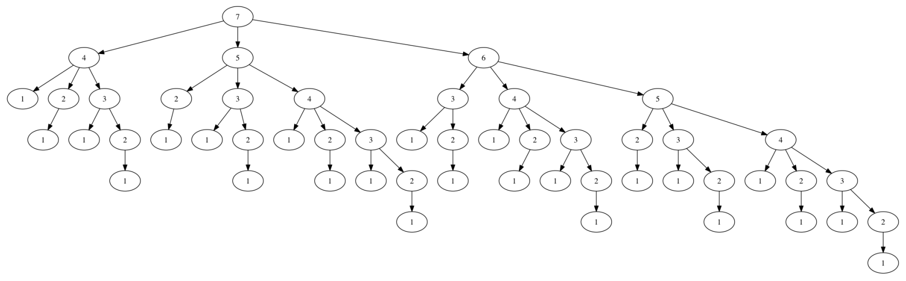

Figure 4: Minimax Tree for n=7

2.2 Minimax Tree n=15

The minimax tree for n=15 is too large to display in a nice look. So I captured part of the tree. The complete picture is included in the submission package.

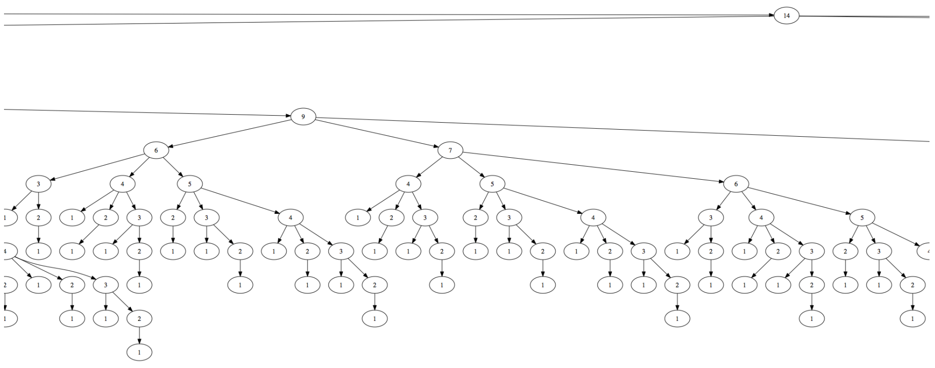

Figure 5: Minimax Tree for n=7

2.3 Minimax Tree n=21

The minimax tree for n=21 is generated successful but is too large to visualize. It causes the software stuck.

2.4 Alpha-Beta Tree n=7

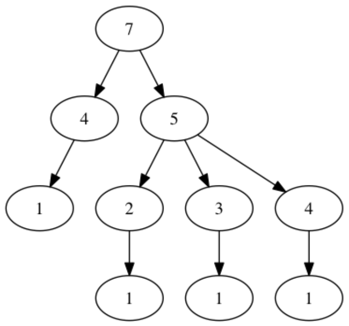

Figure 6: Minimax Tree for n=7

2.5 Alpha-Beta Tree n=15

Due to its large expansion, the picture is zoomed out. The original look of the picture is included in the submission package.

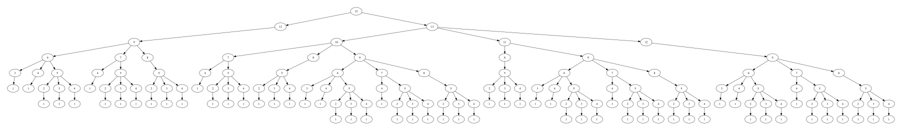

Figure 7: Minimax Tree for n=7

2.6 Alpha-Beta Tree n=21

The alpha-beta tree for n=21 is too large to display in a nice look. So I captured part of the tree. The complete picture is included in the submission package.

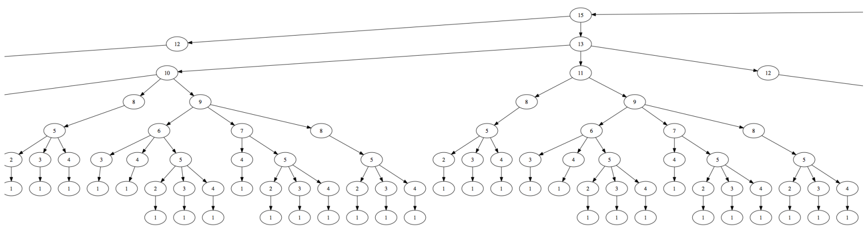

Figure 8: Minimax Tree for n=7

3 Analysis of the Games

3.1 Relative Size of the Trees

From the program, there is an variable "index" that records the size of the trees. For the Minimax Tree:

- $n = 7$, size=15
- $n = 15$, size=6872
- $n = 21$, size=266079

For the Alpha-Beta Tree:

- $n = 7$, size=10
- $n = 15$, size=156
- $n = 21$, size=1839

3.2 Analysis of Opting to Go First or Second

We can conclude from the game that one is doomed to win or lose the game from the very beginning, given that his opponent takes the same strategy. The state when $n = 4k+1$, (where k is any non-negative integer) is the *KEY* state. Whenever given the state n=4k+1 (i.e. 1, 5, 9, 13...), the player is on the path to lose.

Here is the explanation:

Given $n = 4k + 1$, whatever the player chooses to take, his opponent will take the corresponding move to keep the state in $4k - 3$ ($1 \rightarrow 4k - 3$, $2 \rightarrow 4k - 3$, $2 \rightarrow 4k - 3$). This will finally lead to the state 1 that kills the player. Similarly, given $n \neq 4k + 1$, the player's strategy should be keep the next state to $4k + 1$.

And hence, whether the player opts to go first or second is determined by the initial state, which is how many matches are at the beginning. If there are $4k + 1$ matches, the player should choose to go second. Otherwise the player chooses to go first. In this game, when $n = 7$ or $n = 15$, the player should go first. When $n = 21$, the player should go second.

The value of the game is the possible gains, which is the utility 1 times the win chances plus the utility -1 times the lose chances. It is represented as:

$$E[n = i] = 1 * Pr(i)_{[win]} + (-1) * Pr(i)_{[lose]}$$

3.3 Analysis of Playing Against a Random Player

When the player is a optimal player who makes the right decision, and the opponent is a random player, who takes the decision with uniform probability, the game will not be determined at the very beginning. That is, the player has a much larger chance to win. If he falls in the $4k + 1$ state, he can choose to take 1 match put the state to 4k so as to get the most possibility for the opponent to make the wrong decision. Once the opponent goes wrong, the player wins. If he does not fall in the $4k + 1$ state, he is sure to win. The figures in the next section will show this comparison clearly.

Next we analyze this problem in a mathematical way.

3.3.1 Go First

If the player goes first, we calculate the expected probability for $n = 1, 2, 3, ..., \infty$ by following the equation:

$$E[n = i] = 1 * Pr(i)_{[win]} + (-1) * Pr(i)_{[lose]}$$

where when $k = 0, 1, 2, ...$

- $n = 4k + 1, Pr(i)_{[win]} = (\frac{1}{3})^k$
- $n \neq 4k + 1, Pr(i)_{[win]} = 1$

Then we obtain:

- $E[n = 1] = -1 = 1 - (\frac{1}{3})^0 - (\frac{1}{3})^0 = 1 - 2 * (\frac{1}{3})^0$

- $E[n=2]=1$
- $E[n=3]=1$
- $E[n=4]=1$
- $E[n=5]=\frac{1}{3}=1-(\frac{1}{3})^1-(\frac{1}{3})^1=1-2*(\frac{1}{3})^1$
- $E[n=6]=1$
- $E[n=7]=1$
- $E[n=8]=1$
- $E[n=9]=\frac{7}{9}=1-(\frac{1}{3})^2-(\frac{1}{3})^2=1-2*(\frac{1}{3})^2$
- ...

We summarized and get the expected value as:

$$\begin{aligned}
E &= \frac{1}{n}*(-1+1+1+1+\frac{1}{3}+1+1+1+\frac{7}{9}+...) \\
&= \frac{1}{n}*((1-2*(\frac{1}{3})^0)+1+1+1+(1-2*(\frac{1}{3})^1)+1+1+1+(1-2*(\frac{1}{3})^2)+...) \\
&= \lim_{n\to\infty}(\frac{1}{n}*\sum_{i=1}^{n}1)-\lim_{n\to\infty}(\frac{2}{n}*\sum_{i=0}^{n}(\frac{1}{3})^i) \\
&= 1-\lim_{n\to\infty}(\frac{3}{n}*(1-(\frac{1}{3})^n)) \\
&\to 1
\end{aligned}$$

As the number of levels n increases, the probability of the optimal player wins increases and closer to 1. The overall value expectation is converged to 1.

3.3.2 Go Second

If the player goes second, we also calculate the expected probability for $n = 1,2,3,...,\infty$ by following the equation:

$$E[n=i]=1*Pr(i)_{[win]}+(-1)*Pr(i)_{[lose]}$$

where when $k=1,2,3,...$

- $n=4k+1, Pr(i)_{[win]}=1$
- $n=4k+1, Pr(i)_{[win]}=1-(\frac{1}{3})^k*\frac{1}{2}$

Then we obtain:

- $E[n=1]=1$

- $E[n=2] = -1 = 1 * 0 + (-1) * 1$
- $E[n=3] = 0 = 1 * \frac{1}{2} + (-1) * \frac{1}{2}$
- $E[n=4] = \frac{1}{3} = 1 * (1 - \frac{1}{3}) + (-1) * \frac{1}{3} = 1 - 2 * \frac{1}{3}$
- $E[n=5] = 1$
- $E[n=6] = \frac{2}{3} = 1 * (1 - \frac{1}{3} * \frac{1}{2}) + (-1) * \frac{1}{3} * \frac{1}{2} = 1 - 2 * \frac{1}{3} * \frac{1}{2} = 1 - \frac{1}{3}$
- $E[n=7] = \frac{2}{3} = 1 * (1 - \frac{1}{3} * \frac{1}{2}) + (-1) * \frac{1}{3} * \frac{1}{2} = 1 - 2 * \frac{1}{3} * \frac{1}{2} = 1 - \frac{1}{3}$
- $E[n=8] = \frac{2}{3} = 1 * (1 - \frac{1}{3} * \frac{1}{2}) + (-1) * \frac{1}{3} * \frac{1}{2} = 1 - 2 * \frac{1}{3} * \frac{1}{2} = 1 - \frac{1}{3}$
- $E[n=9] = 1$
- $E[n=10] = \frac{2}{3} = 1 - (\frac{1}{3})^2$
- ...

We summarized and get the expected value as:

$$\begin{aligned}
E &= \frac{1}{n} * (1 + (-1) + 0 + \frac{1}{3} + 1 + (1 - \frac{1}{3}) + (1 - \frac{1}{3}) + (1 - \frac{1}{3}) + 1 + (1 - \frac{1}{3})...) \\
&= \lim_{n\to\infty} (\frac{1}{n} * (\frac{1}{3} + \sum_{i=5}^{n} 1 - 3 * \sum_{i=1}^{n} (\frac{1}{3})^i)) \\
&= \lim_{n\to\infty} (1 - \frac{11}{3n} - \frac{3}{2n} * (1 - (\frac{1}{3}))^n) \\
&\to 1
\end{aligned}$$

As the number of levels n increases, the probability of the optimal player wins increases and closer to 1. The overall value expectation is converged to 1.

4 Graphs of Moore Machine

In the following graphs, the numbers in the circle represents the number of matches left. The tuple on the arrow represents: (input, output). If the player goes first, the input represents the player's move and the output represents the opponent's move; and vice versa.

4.1 n=7, Go First

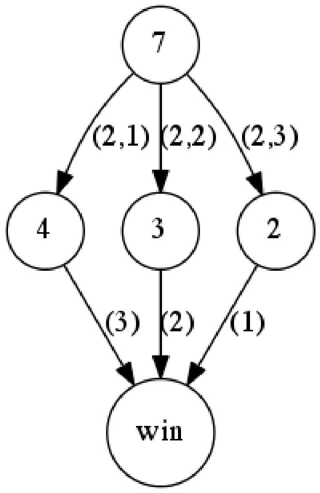

Figure 9: Moore machine for n=7, go first

4.2 n=7, Go Second

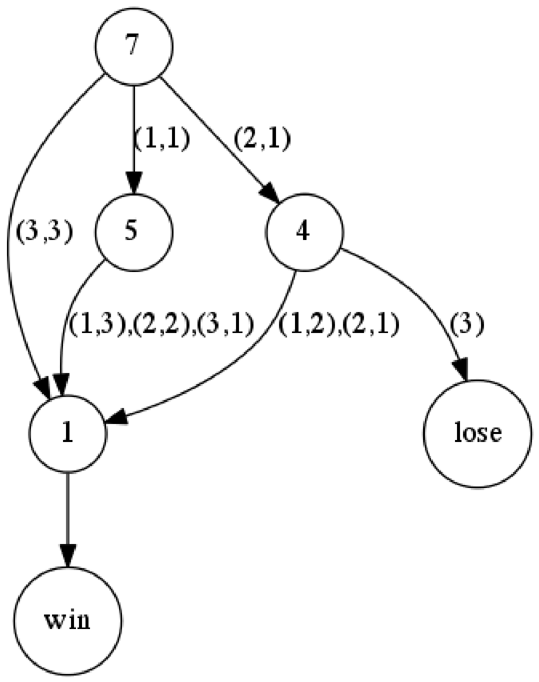

Figure 10: Moore machine for n=7, go second

4.3 n=15, Go First

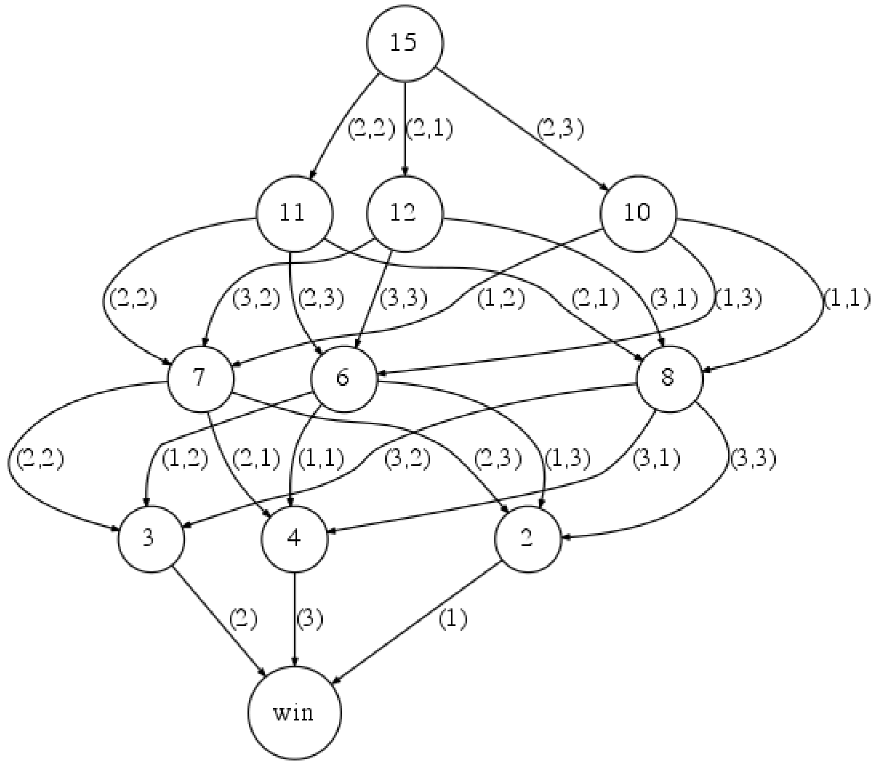

Figure 11: Moore machine for n=15, go first

4.4 n=15, Go Second

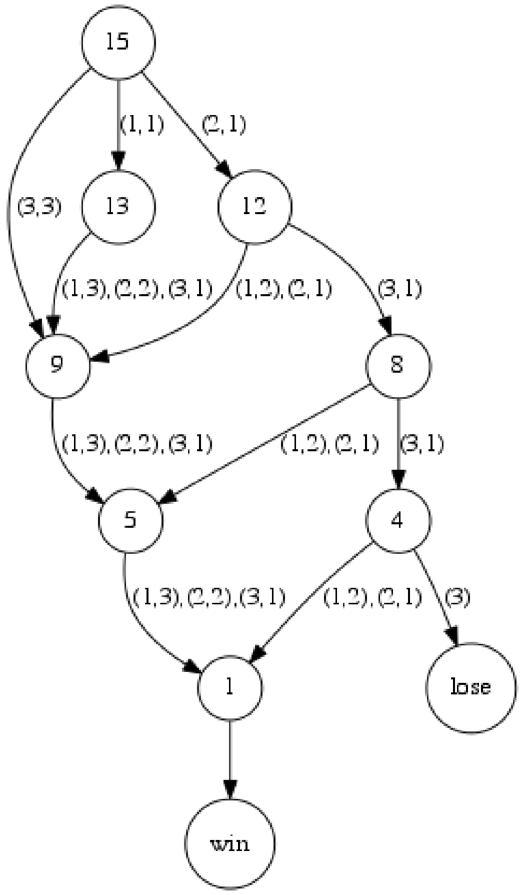

Figure 12: Moore machine for n=15, go second

4.5 n=21, Go First

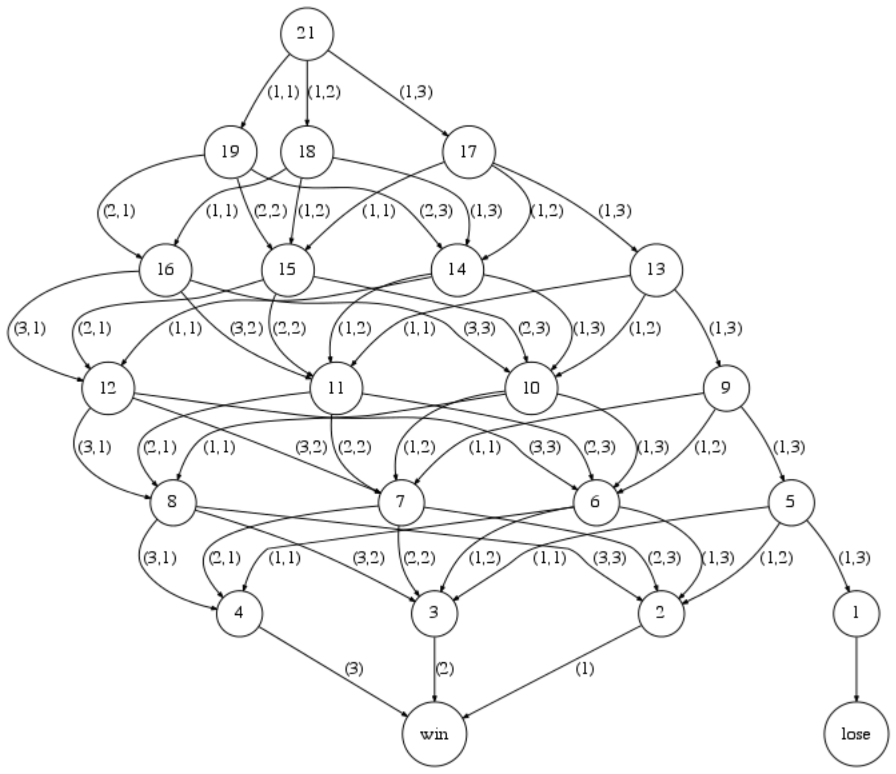

Figure 13: Moore machine for n=21, go first

4.6 n=21, Go Second

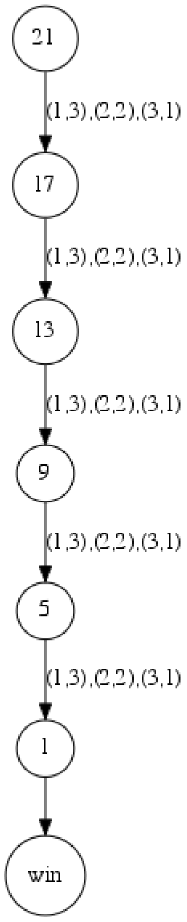

Figure 14: Moore machine for n=21, go second