Adversarial Search

• Basic minimax search
• Evaluation Functions (resource limited)
• Alpha beta pruning
• Techniques used in board playing
  – World champion of Checkers, Othello is a machine (http://www.cs.ualberta.ca/~chinook/)
  – Machines challenging in Chess, Backgammon
In Practice

Checkers: Chinook ended 40-year-reign of human world champion Marion Tinsley in 1994. Used an endgame database defining perfect play for all positions involving 8 or fewer pieces on the board, a total of 443,748,401,247 positions.


Othello: human champions refuse to compete against computers, who are too good.

Go: human champions refuse to compete against computers, who are too bad. In go, $b > 300$, so most programs use pattern knowledge bases to suggest plausible moves.
Games and Search

“Unpredictable” opponent ⇒ solution is a contingency plan

Time limits ⇒ unlikely to find goal, must approximate

Plan of attack:

• algorithm for perfect play (Von Neumann, 1944)
• finite horizon, approximate evaluation (Zuse, 1945; Shannon, 1950; Samuel, 1952–57)
• pruning to reduce costs (McCarthy, 1956)

<table>
<thead>
<tr>
<th>perfect information</th>
<th>deterministic</th>
<th>chance</th>
</tr>
</thead>
<tbody>
<tr>
<td>chess, checkers, go, othello</td>
<td>backgammon monopoly</td>
<td></td>
</tr>
</tbody>
</table>

| imperfect information |  | bridge, poker, scrabble, nuclear war |
|-----------------------| | *** |
Type of Games We Will Address From a Game Theory Perspective

• Turn taking
• Two player
• Zero-sum
• Perfect information
• Also
  – Player is rational
  – Game tree is too large to exhaustively search
    • i.e. Chess $b = 35$, $d = 50$
Items Required to Build a Game Tree

- Initial state
- Successor function
- Terminal test
- Utility function (…)
- Note, MAX player always goes first
- Consider Tic-Tac-Toe
Optimal Strategy

• No longer shortest path
• Rather the minimum maximum utility (minimax)
• Why?
Game Trees and Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
      = best achievable payoff against best play

E.g., 2-ply game:

```
function Minimax-Decision(game) returns an operator
    for each op in Operators[game] do
        Value[op] ← Minimax-Value(Apply(op, game), game)
    end
    return the op with the highest Value[op]
```

```
function Minimax-Value(state, game) returns a utility value
    if Terminal-Test[game](state) then
        return Utility[game](state)
    else if MAX is to move in state then
        return the highest Minimax-Value of Successors(state)
    else
        return the lowest Minimax-Value of Successors(state)
```
Minimax Example
Properties of Minimax Search

Complete

Optimal

Time complexity

Space complexity
Properties of Minimax Search

Complete?? Yes, if tree is finite (chess has specific rules for this)

Optimal?? Yes, against an optimal opponent. Otherwise??

Time complexity?? $O(b^m)$

Space complexity?? $O(bm)$ (depth-first exploration)

For chess, $b \approx 35$, $m \approx 100$ for “reasonable” games

$\Rightarrow$ exact solution completely infeasible
Evaluation Functions

• We can’t rely on using minimax to search to the bottom of the tree.
• Need to limit the number of moves to look ahead and then use an evaluation function to return a numerical value for each node at that level/ply.
• View the evaluation function as being an approximation to the EXPECTED utility.
• Evaluation function: Should order terminal nodes as the true utility would, quick, strongly correlated with chance of winning
Evaluation Functions

For chess, typically *linear* weighted sum of features

\[ \text{Eval}(s) = w_1 f_1(s) + w_2 f_2(s) + \ldots + w_n f_n(s) \]

e.g., \( w_1 = 9 \) with
\[ f_1(s) = (\text{number of white queens}) - (\text{number of black queens}) \]
etc.
Example – 3D Tic-Tac-Toe

 Depth of tree?
Number of nodes?
Evaluation function?
What are the components?
How should we score each component?
Close Enough is Good Enough

MAX

MIN

Behaviour is preserved under any monotonic transformation of Eval

Only the order matters:

payoff in deterministic games acts as an ordinal utility function
Three Player Games

How do we create game trees that incorporate chance?
Chance as a Bipartisan Third Player

In nondeterministic games, chance introduced by dice, card-shuffling.

Simplified example with coin-flipping:

```
MAX
   /
  / \ 3
CHANCE
/   \
/ 0.5 \ 0.5
MIN
/   \
2 4
\ 7
2 4
```

**EXPECTIMINIMAX gives perfect play**

Just like **MINIMAX**, except we must also handle chance nodes:

```
... if state is a MAX node then
    return the highest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a MIN node then
    return the lowest EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
if state is a chance node then
    return average of EXPECTIMINIMAX-VALUE of SUCCESSORS(state)
...```

Depth Limited Search

**MinimaxCutoff** is identical to **MinimaxValue** except
1. **Terminal?** is replaced by **Cutoff?**
2. **Utility** is replaced by **Eval**

Evaluation function is just a heuristic (not an admissible heuristic)
Break game state into m components
Eval(n) = w₁c₁(n)+w₂c₂(n)...wₘcₘ(n)
Note multiple states have the exact same component values
Eval function some estimate of the the expectation of
#wins/#games from this point (as described by the components)
Cutting of Search

MinimaxCutoff is identical to MinimaxValue except
1. Terminal? is replaced by Cutoff?
2. Utility is replaced by Eval

Does it work in practice?

\[ b^m = 10^6, \quad b = 35 \quad \Rightarrow \quad m = 4 \]

4-ply lookahead is a hopeless chess player!

4-ply \(\approx\) human novice
8-ply \(\approx\) typical PC, human master
12-ply \(\approx\) Deep Blue, Kasparov
Up To Here
Adversarial Search

• Typical Situation:
  – Two players, playing in turn, zero sum game (refers to payoff, think of it as M.E. goals), perfect information

• Aim:
  – From the start point, devise a strategy that will maximize the payoff, assuming our opponent plays rationally, by examining (if possible) to the complete game tree.

• Action:
  – Then make ONE move and wait for your opponent.

• Note: Not necessarily a MINIMAX solution
Counter-Intuitive Result!

Perfect play for deterministic, perfect-information games
Idea: choose move to position with highest \textit{minimax value} = best achievable payoff against best play
E.g., 2-ply game:

Optimal Strategy: Max plays A1, Min A1. Even if Min knows what Max is going to play and vice-versa, neither will change their strategy. Solution is an equilibrium in that it maxs Max’s payoff and mins Min payoff

In game theory, the \textbf{Nash equilibrium} (named after John Nash, who proposed it) is a kind of optimal collective strategy in a game involving two or more players, where no player has anything to gain by changing only his or her own strategy.
What About Equilibrium for Evaluation Functions
Game Trees and Minimax

Perfect play for deterministic, perfect-information games

Idea: choose move to position with highest minimax value
    = best achievable payoff against best play

E.g., 2-ply game:

```
function Minimax-Decision(game) returns an operator
    for each op in Operators[game] do
        Value[op] ← Minimax-Value(Apply(op, game), game)
    end
    return the op with the highest Value[op]

function Minimax-Value(state, game) returns a utility value
    if Terminal-Test(game)(state) then
        return Utility(game)(state)
    else if MAX is to move in state then
        return the highest Minimax-Value of Successors(state)
    else
        return the lowest Minimax-Value of Successors(state)
```
Example – 3D Tic-Tac-Toe

Number of states: Let's just consider 3x3 Tic-Tac-Toe, $3^9 = 19683$ overstates. Why?

The solution I proposed:
Board is full: $^9C_5 +$
One empty space $^9C_1 x ^8C_4 +$
Two empty spaces $^9C_2 x ^7C_4 +$
Three empty spaces $^9C_3 x ^6C_3 +$ …
9 empty spaces $^9C_9 x ^0C_1$
=6046 still overstates. Why?
Properties of Minimax Search

**Complete**? Yes, if tree is finite (chess has specific rules for this)

**Optimal**? Yes, against an optimal opponent. Otherwise?

**Time complexity**? \( O(b^m) \)

**Space complexity**? \( O(bm) \) (depth-first exploration)

For chess, \( b \approx 35, m \approx 100 \) for “reasonable” games

\[ \Rightarrow \text{exact solution completely infeasible} \]
Three Player and Non-Zero Sum Games

A

/  \\
B        (?? ?? ??)

/  \\
C       (?? ?? ??)  (?? ?? ??)  (?? ?? ??)

/  \\
A (+1 +2 +3) (+4 +2 +1) (+6 +1 +2) (+7 +4 -1) (+5 -1 -1) (-1 +5 +2) (+7 +7 -1) (+5 +4 +5)

How do we create game trees that incorporate chance?
Chance as a Bipartisan Third Player

In nondeterministic games, chance introduced by dice, card-shuffling.

Simplified example with coin-flipping:

\[
\begin{array}{c}
\text{MAX} \\
\text{CHANCE} \\
\text{MIN}
\end{array}
\]

\[
\begin{array}{cccccc}
& 3 & & & & \\
\downarrow & & \downarrow & & \downarrow & \\
0.5 & 0.5 & 0.5 & 0.5 & \\
\downarrow & & \downarrow & & \downarrow & \\
2 & 4 & 0 & -2 & \\
\downarrow & & \downarrow & & \downarrow & \\
2 & 4 & 6 & 5 & -2
\end{array}
\]

Before each link implicitly Had Pr=1 on it.

**EXPECTIMINIMAX** gives perfect play

Just like **MINIMAX**, except we must also handle chance nodes:

\[
\begin{align*}
\text{if } state \text{ is a MAX node then} & \quad \text{return the highest } \text{EXPECTIMINIMAX-VALUE of SUCCESSORS}(state) \\
\text{if } state \text{ is a MIN node then} & \quad \text{return the lowest } \text{EXPECTIMINIMAX-VALUE of SUCCESSORS}(state) \\
\text{if } state \text{ is a chance node then} & \quad \text{return average of } \text{EXPECTIMINIMAX-VALUE of SUCCESSORS}(state)
\end{align*}
\]

\[
\ldots
\]
Alpha Beta Pruning

- Some branches will never be played by rational players since they include sub-optimal decisions (for either player).

```
    MAX
   /   /
  MIN  MAX
 /  /  /  /  /  /  /  /  /  /  /
 4 3 6 2 2 1 9 5 3 1 5 4 7 5
```

Write down as a function ...
What is vital to the success of this approach?
Write down as a function …
**Alpha-Beta pruning**

```plaintext
function MAX-VALUE(state, game, α, β) returns the minimax value of state
    inputs: state, current state in game
game, game description
    α, the best score for MAX along the path to state
    β, the best score for MIN along the path to state

    if GOAL-TEST(state) then return EVAL(state)
    for each s in SUCCESSORS(state) do
        α ← MAX(α, MIN-VALUE(s, game, α, β))
        if α ≥ β then return β
    end
    return α

function MIN-VALUE(state, game, α, β) returns the minimax value of state

    if GOAL-TEST(state) then return EVAL(state)
    for each s in SUCCESSORS(state) do
        β ← MIN(β, MAX-VALUE(s, game, α, β))
        if β ≤ α then return α
    end
    return β
```

Passed by value
Alpha-beta pruning example: Step 1

\[ \alpha \] is the maximum lower bound of possible solutions (MAX plays)

\[ \beta \] is the minimum upper bound of possible solutions (MIN plays)

So what does Alpha and beta Represent?

Max fills left hand alpha value
Min fills in right beta value
Parent can set current best Estimates for alpha and beta
For this path, worst min can do is 5.

Even without seeing The rest of the tree If Min plays rationally, They will allow a pay-off No greater than 5.

For this path, worst max can do is 5.
Alpha-beta pruning example: Step 4
\[\alpha\] is the maximum lower bound of possible solutions (MAX plays)
\[\beta\] is the minimum upper bound of possible solutions (MIN plays)

Irrational for MAX to ever Play this path

Alpha-beta pruning example: Step 5
\[\alpha\] is the maximum lower bound of possible solutions (MAX plays)
\[\beta\] is the minimum upper bound of possible solutions (MIN plays)
Alpha-beta pruning example: Step 6
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 7
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)
Alpha-beta pruning example: Step 8

$\alpha$ is the maximum lower bound of possible solutions (MAX plays)
$\beta$ is the minimum upper bound of possible solutions (MIN plays)

All branches “complete”
Alpha-beta pruning example: Step 10
α is the maximum lower bound of possible solutions (MAX plays)
β is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 11
α is the maximum lower bound of possible solutions (MAX plays)
β is the minimum upper bound of possible solutions (MIN plays)
Expand the node next to 3
Since it is RATIONAL that
Min could redefine MAX’s lower bound

But it doesn’t!
If 7 was say 4 then
It would have!
Alpha-beta pruning example: Step 14
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

\[
\begin{array}{c}
\quad \\
5 \leq \infty \\
\quad \\
5 \leq 5 \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\quad \\
5 \leq \infty \\
\end{array}
\]

Alpha-beta pruning example: Step 15
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)
Alpha-beta pruning example: Step 16

- $\alpha$ is the maximum lower bound of possible solutions (MAX plays)
- $\beta$ is the minimum upper bound of possible solutions (MIN plays)

---

Alpha-beta pruning example: Step 17

---

Alpha-beta pruning example: Step 18

---
Alpha Beta Pruning

- Produces exact same results as minimax algorithm
- Reduces search by not expanding sub-optimal paths
- Alpha and Beta terms are the floor and ceiling of the range of values, the player is interested in.
Initial call is to $\text{MAX\_VALUE}(\text{root\_node}, -\infty, \infty)$

$\text{Alpha} \leq \text{Beta}$, otherwise cut-off search

**Alpha-Beta pruning**

```plaintext
function \text{MAX\_VALUE}(\text{state}, \text{game}, \alpha, \beta) \text{ returns} \text{ the minimax value of state}
inputs: \text{state}, \text{current state in game}
\text{game}, \text{game description}
\alpha, \text{the best score for MAX along the path to state}
\beta, \text{the best score for MIN along the path to state}

if \ \text{GOAL\_TEST}(\text{state}) \text{ then return } \text{EVAL}(\text{state})
for each \text{s} \text{ in } \text{SUCCESSORS}(\text{state}) \text{ do}
    \alpha \leftarrow \text{MAX}(\alpha, \text{MIN\_VALUE}(s, \text{game}, \alpha, \beta))
    if \alpha \geq \beta \text{ then return } \beta
end
return \alpha

function \text{MIN\_VALUE}(\text{state}, \text{game}, \alpha, \beta) \text{ returns} \text{ the minimax value of state}

if \ \text{GOAL\_TEST}(\text{state}) \text{ then return } \text{EVAL}(\text{state})
for each \text{s} \text{ in } \text{SUCCESSORS}(\text{state}) \text{ do}
    \beta \leftarrow \text{MIN}(\beta, \text{MAX\_VALUE}(s, \text{game}, \alpha, \beta))
    if \beta \leq \alpha \text{ then return } \alpha
end
return \beta
```

Max of min values

Min of max values
Analysis of Alpha Beta Pruning

  - Most results for perfectly ordered trees
  - Some for randomly ordered trees
Performance analysis of Alpha-Beta Pruning

• Since alpha-beta pruning performs a minimax search while pruning much of the tree, its effect is to allow a deeper search with the same amount of computation.

• The question: how much does alpha-beta improve performance?
Example of alpha-beta *best* case

- Evaluation from left to right. Best case analysis $O(b^{d/2})$. What does this mean in practice?

Note when Min is playing last the best order is monotonically increasing.
The Proof (kind of)

For $b = 3$ and $d = 3$ we need to expand 11 nodes.
What type of state values do we have in this tree?
What information do we need to know an exact value?
What information do we need to know a bounded value?
The Proof - 2

- We want to know how many expansions we have to do to get an exact value
- Let $S(k)$ be the min # of states $k$-ply away to expand to get an exact value
- Let $R(k)$ be the min # of states $k$-ply away to expand to get a bounded value
- $b$ is the branching factor
- $S(k+1) = ?$
- $R(k+1) = ?$
The Proof - 3

\[ S(k+1) = S(k) + (b-1)R(k) \]
\[ R(k+1) = S(k) \]
\[ S(0)=R(0)=1 \]
\[ S(3) = S(2) + (b-1)R(2) \]
\[ = S(1) + (b-1)R(1) + (b-1)S(1) \]
\[ = 1 + (b-1) + (b-1) + (b-1)(1+b-1) \]
\[ = 1 + 2b - 2 + b + b^2 - b - 1 - b + 1 \]
\[ = b^2 + b - 1 \]
\[ = ??? \]
The Proof - 4

We said we can look twice as far ahead with alpha-beta pruning.

How can we express this with respect to S(k)?
The Proof - 5

We said we can look twice as far ahead with alpha-beta pruning.

Without alpha beta pruning:

\[ S(k+2) = b^2 S(k) \]

With alpha beta pruning

\[ S(k+2) < \frac{b^2}{2} S(k) < 2b S(k) \]
The Proof - 6

\[ S(k+2) = S(k+1) +(b-1)R(k+1) \]

What do we want to do …

\[ = S(k) + (b-1)R(k) + (b-1)S(k) \]

\[ = bS(k) + (b-1)S(k-1) \]

\[ < (2b-1)S(k) \text{ why?} \]

\[ < 2bS(k) \text{ why?} \]
Minimax value of game trees

- The most natural definition for the average case is that the leaf nodes are randomly ordered.
- Heuristic node ordering would violate this assumption.
- Average case performance is not a prediction of its performance in practice.
Game Trees

• Alpha-Beta Pruning
• Best Case Analysis
• Ordering of Nodes to Get Best Case Analysis
Initial call is to \( \text{MAX\_VALUE}(\text{root\_node}, -\infty, \infty) \)
Alpha \( \leq \) Beta, otherwise cut-off search

**Alpha-Beta pruning**

```
function MAX-VALUE(state, game, \( \alpha \), \( \beta \)) returns the minimax value of state
    inputs: state, current state in game
            game, game description
            \( \alpha \), the best score for MAX along the path to state
            \( \beta \), the best score for MIN along the path to state
        
    if GOAL-TEST(state) then return Eval(state)
    for each s in SUCCESSORS(state) do
        \( \alpha \leftarrow \max(\alpha, \text{MIN\_VALUE}(s, game, \alpha, \beta)) \)
        if \( \alpha \geq \beta \) then return \( \beta \)
    end
    return \( \alpha \)

function MIN-VALUE(state, game, \( \alpha \), \( \beta \)) returns the minimax value of state
    if GOAL-TEST(state) then return Eval(state)
    for each s in SUCCESSORS(state) do
        \( \beta \leftarrow \min(\beta, \text{MAX\_VALUE}(s, game, \alpha, \beta)) \)
        if \( \beta \leq \alpha \) then return \( \alpha \)
    end
    return \( \beta \)
```

Max of min values

Min of max values
Alpha-beta pruning example: Step 1

\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)

\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

So what does Alpha and beta Represent?

Max fills left hand alpha value
Min fills in right beta value
Parent can set current best Estimates for alpha and beta
What is vital to the success of this approach?
Write down as a function …
Alpha-beta pruning example: Step 1

$\alpha$ is the maximum lower bound of possible solutions (MAX plays)
$\beta$ is the minimum upper bound of possible solutions (MIN plays)

So what does Alpha and beta Represent?

Max fills left hand alpha value
Min fills in right beta value
Parent can set current best Estimates for alpha and beta

Alpha and Beta are the Bounds of what’s interesting!
For this path, worst min can do is 5.

Even without seeing The rest of the tree If Min plays rationally, They will allow a pay-off No greater than 5.
Alpha-beta pruning example: Step 4
\(\alpha\) is the maximum lower bound of possible solutions (MAX plays)
\(\beta\) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 5
\(\alpha\) is the maximum lower bound of possible solutions (MAX plays)
\(\beta\) is the minimum upper bound of possible solutions (MIN plays)

Irrational for MAX to ever Play this path
Alpha-beta pruning example: Step 6
\(\alpha\) is the maximum lower bound of possible solutions (MAX plays)
\(\beta\) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 7
\(\alpha\) is the maximum lower bound of possible solutions (MAX plays)
\(\beta\) is the minimum upper bound of possible solutions (MIN plays)
Alpha-beta pruning example: Step 8
$\alpha$ is the maximum lower bound of possible solutions (MAX plays)
$\beta$ is the minimum upper bound of possible solutions (MIN plays)

All branches “complete”
Alpha-beta pruning example: Step 10
\(\alpha\) is the maximum lower bound of possible solutions (MAX plays)
\(\beta\) is the minimum upper bound of possible solutions (MIN plays)

---

Alpha-beta pruning example: Step 11
\(\alpha\) is the maximum lower bound of possible solutions (MAX plays)
\(\beta\) is the minimum upper bound of possible solutions (MIN plays)
Expand the node next to 3
Since it is RATIONAL that
Min could redefine MAX’s lower
bound

But it doesn’t!
If 7 was say 4 then
It would have!
Alpha-beta pruning example: Step 14
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 15
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)
Alpha-beta pruning example: Step 16
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 17
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)

Alpha-beta pruning example: Step 18
\( \alpha \) is the maximum lower bound of possible solutions (MAX plays)
\( \beta \) is the minimum upper bound of possible solutions (MIN plays)
For \( b = 3 \) and \( d = 3 \) we need to expand 11 nodes.

What type of state values do we have in this tree?

What information do we need to know an exact value?

What information do we need to know a bounded value?
The Proof - 2

• We want to know how many expansions we have to do to get an exact value
• Let $S(k)$ be the min # of states $k$-ply away to expand to get an exact value
• Let $R(k)$ be the min # of states $k$-ply away to expand to get a bounded value
• $b$ is the branching factor
• $S(k+1) = ?$
• $R(k+1) = ?$
The Proof - 3

$S(k+1) = S(k) + (b-1)R(k)$
$R(k+1) = S(k)$
$S(0)=R(0)=1$
$S(3) = S(2) + (b-1)R(2)$

$= S(1) + (b-1)R(1) + (b-1)S(1)$
$= 1 + (b-1) + (b-1) + (b-1)(1+b-1)$
$= 1 + 2b - 2 + b + b^2 - b - 1 - b + 1$
$= b^2 + b - 1$
$= ???$
The Proof - 4

We said we can look twice as far ahead with alpha-beta pruning.

How can we express this with respect to S(k)?
We said we can look twice as far ahead with alpha-beta pruning.

Without alpha beta pruning:
\[ S(k+2) = b^2 S(k) \]

With alpha beta pruning
\[ S(k+2) < \frac{b^2}{2} S(k) < \frac{2}{2} b S(k) \]
The Proof - 6

\[ S(k+2) = S(k+1) + (b-1)R(k+1) \]

What do we want to do …

\[ = S(k) + (b-1)R(k) + (b-1)S(k) \]

\[ = bS(k) + (b-1)S(k-1) \]

\[ < (2b-1)S(k) \text{ why?} \]

\[ < 2bS(k) \text{ why?} \]
How To Get This Best Case Result?

- We build the tree after all, how do we get this monotonically increasing/decreasing requirement?
- Naïve way: sorting (defeats the purpose)
- For loss/win games (payoff is –1, 0 or +1)
  - Generate states we know will win first (or last)
- When using an evaluation function
  - Generate nodes with highest/lowest costs first