Overview

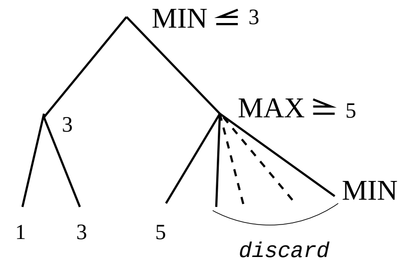
- formal $\alpha - \beta$ pruning algorithm
- $\alpha - \beta$ pruning properties
- games with an element of chance
- state-of-the-art game playing with AI
- more complex games
- **project #1: full description**
  - core routines for 8-puzzle

$\alpha - \beta$ Pruning: Initialization

Along the path from the beginning to the current **state**:

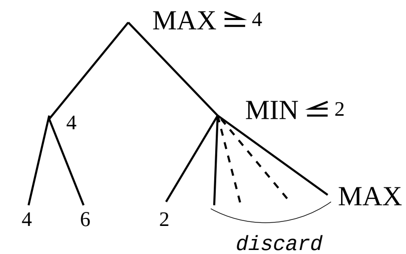
- α: best MAX value
  - initialize to $-\infty$
- β: best MIN value
  - initialize to ∞

$\alpha - \beta$ Pruning Algorithm: Max-Value



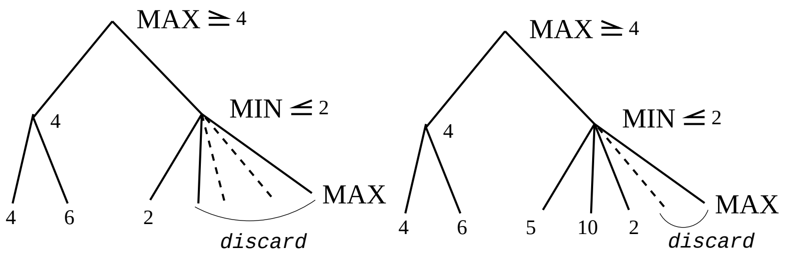
function Max-Value (state, game, α, β) **return** minmax(state)
α: best MAX on path to *state* ; β: best MIN on path to *state*
if Cutoff(state) **then return** Eval(state)
for each *s* in Successor(state) **do**
· $\alpha \leftarrow$ Max(α, Min-Value(s,game,α,β))
· **if** $\alpha \geq \beta$ **then return** β **/* CUT!! */**
end
return α

$\alpha - \beta$ Pruning Algorithm: Min-Value



function Min-Value (state, game, α, β) **return** minmax(state)
α: best MAX on path to *state* ; β: best MIN on path to *state*
if Cutoff(state) **then return** Eval(state)
for each *s* in Successor(state) **do**
· $\beta \leftarrow$ Min(β, Max-Value(s,game,α,β))
· **if** $\beta \leq \alpha$ **then return** α **/* CUT!! */**
end
return β

Ordering is Important for Good Pruning



- For MIN, sorting succesor's utility in an **increasing** order is better (shown above; left).
- For MAX, sorting in **decreasing** order is better.

$\alpha - \beta$ Pruning Properties

Cut off nodes that are known to be suboptimal.

Properties:

- pruning **does not** affect final result
- good move ordering improves effectivenes of pruning
- with **perfect ordering**, time complexity = $b^{m/2}$
  $\rightarrow$ **doubles** depth of search
  $\rightarrow$ can easily reach 8-ply in chess
- $b^{m/2} = (\sqrt{b})^m$, thus $b = 35$ in chess reduces to $b = \sqrt{35} \approx 6$!!!

** this slide is a copy from the last lecture*

Games With an Element of Chance

Rolling the dice, shuffling the deck of card and drawing, etc.

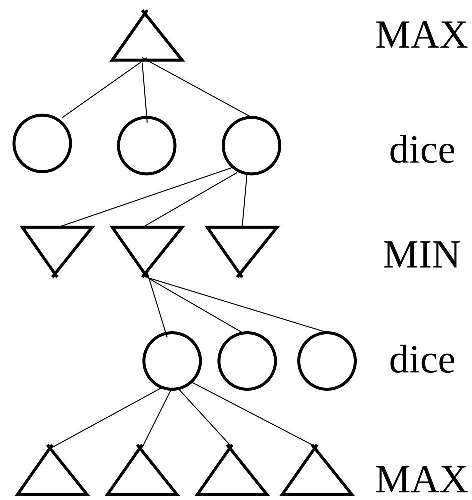
- **chance nodes** need to be included in the minimax tree
- try to make a move that maximizes the **expected value** $\rightarrow$ **expectimax**
- expected value of random variable X:

$$E(X) = \sum_x xP(x)$$

- expectimax

$$\text{expectimax}(C) = \sum_i P(d_i)\max_{s \in S(C,d_i)}(utility(s))$$

Game Tree With Chance Element



- chance element forms a new ply (e.g. dice, shown above)

Design Considerations for Probabilistic Games

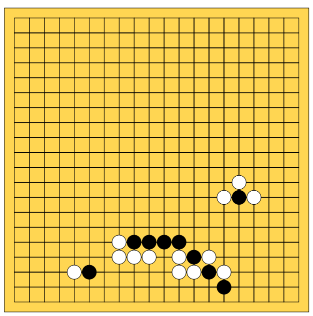
- the **value** of evaluation function, not just the **scale** matters now! (think of what expected value is)
- time complexity: $b^m n^m$, where n is the number of distinct dice rolls
- pruning can be done if we are careful

State of the Art in Gaming With AI

- Chess: IBM's Deep Blue beat Garry Kasparov (1997)
- Backgammon: Tesauro's Neural Network $\longrightarrow$ top three (1992)
- Othello: smaller search space $\longrightarrow$ superhuman performance
- Checkers: Samuel's Checker Program running on 10Kbyte (1952)

Genetic algorithms can perform very well on select domains.

Hard Games



The game of *Go*, popular in East Asia:

- $19 \times 19 = 361$ grid: branching factor is huge!
- search methods inevitably fail: need more structured rules
- the bet is high: $2,000,000 prize

Project 1: Due 3/22 Midnight

Solving eight-puzzle with various search methods:

- Input: a board configuration
  `'(1 3 4 8 6 2 7 0 5)`
- Output: sequence of moves
  `'(UP RIGHT UP LEFT DOWN)`
- Search methods to be used:
  Depth-First, Bounded Depth-First, Iterative Deepening, Breadth-First, Heuristic search with h_1 (tiles out-of-place), and h_2 (sum of manhattan distance)
- This is an **individual project**.

Project 1: Required Material

Use the exact filename as shown below (in **bold**).

- Program code (**eight.lsp**): put it in a single text file.
  – Ample indentation and documentation is required.
- Documentation (**README**): user manual
- Inputs and outputs (include in **README**)
  - Easy: `'(1 3 4 8 6 2 7 0 5)`
  - Medium: `'(2 8 1 0 4 3 7 6 5)`
  - Hard: `'( 5 6 7 4 0 8 3 2 1)`
  - Include 5 examples of your own

Project 1: Required Material (Cont'd)

Continued from the previous page

- For each run, report the **time** taken, and the **number of nodes expanded**. Compare the various search methods using the Easy, Medium, and Hard case examples. Explain why you think certain methods work better than others.
- Some search methods may fail to produce an answer. Analyse why it failed and report your findings.
- 10% Extra Credit for implementing IDA^*: this may not be hard!

Project 1 Tips (1)

Timing execution: use `(get-internal-run-time)` to get current time.

```
(defun loopit (x)
  (dotimes (i x res)
    (progn
      (setq res (+ i 1))
      (print (get-internal-run-time))
    )
  )
)
```


Project 1 Tips (2)

Checking for duplicate states

```
(defun dupe (state node-list)
  (dolist (node node-list nil)
     (if (equal state (first node))
         (return-from dupe T))))
```

A general expand function:

```
(defvar *expand-func*) ; name of expand function
(defun expand (node)
  (funcall *expand-func* node))
```

Project 1: State Representation

1	3	4
8	6	2
7		5

A node in the search tree has the following structure:

```
'((1 3 4 8 6 2 7 0 5);blank is stored as 0
   h                  ;heuristic function value
   depth              ;depth from the root
   path))             ;list of moves from
                      ;  the start
```


Project 1: Utility Routines

Source will be available on the course web page:

- `(apply-op <operator> <node>)`: return new node after applying operator on current node
- `(print-tile <state>)`: prints out the board
- `(print-answer <state> <path>)`: prints boards after each move in the path, starting from the state.

Project 1: Submission

- Send as email to the TA (attached text files): `ltapia@tamu.edu`, and also CC: `choe@tamu.edu`
- Submission deadline is 3/22/02 Friday midnight (23:59:59).
- Late policy: initial penalty -25%, and additional -25% per week thereafter.
- If more than half have problems meeting the deadline, I will consider an extension.
- Only send **plain ASCII text** files. **Do not send MS-Word documents or other formatted text.**

Key Points

- formal $\alpha - \beta$ pruning algorithm: know how to apply pruning
- $\alpha - \beta$ pruning properties: complexity
- games with an element of chance: what are the added elements? how does the minmax tree get augmented?

Next Week: Logic

- Propositional Logic: Chapter 6, 6.3–6.6

Today: AI Seminar

Title: An Artificial Life Approach to Computational Aesthetics

Speaker: Gary R. Greenfield (U. of Richmond)

- 3-4pm Today, HRBB 302 (space is limited)