Alpha-Beta Search

References:

- D. E. Knuth and R. W. Moore. An analysis of alpha-beta pruning. Artificial Intelligence, 6:293–326, 1975.
- J. Pearl. The solution for the branching factor of the alpha-beta pruning algorithm and its optimality. Communications of ACM, 25(8):559–564, 1982.

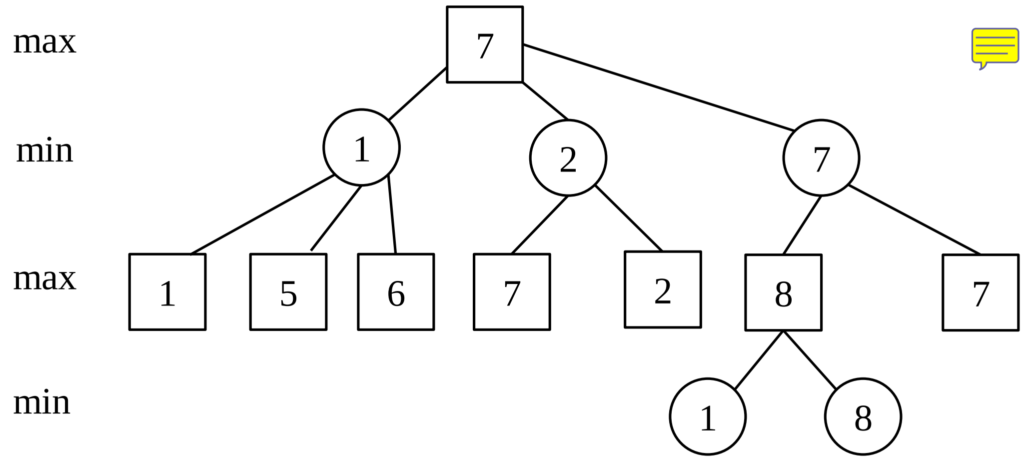
Acknowledgement:

- The slides of this chapter are modified from Prof. Hsu's teaching material, under the courtesy of Hsu.
  http://www.iis.sinica.edu.tw/~tshsu/tcg2007/index.html

Introduction

- Alpha-beta pruning is the standard searching procedure used for 2-person perfect-information zero sum games.
- Definitions:
  - A position p.
  - The value of a position p: $f(p)$, is a numerical value computed from evaluating p.
    - Value is computed from the root player's point of view.
    - Positive values mean in favor of the root player.
    - Negative values mean in favor of the opponent.
    - Since it is a zero sum game, thus from the opponent's point of view, the value can be assigned, $g(p) = -f(p)$.
  - A terminal position: a position whose value can be know.
    - Can be a position where win/loss/draw can be concluded.
    - Can be a position where some constraints are met.
  - A position p has d legal moves $p_1, p_2, \ldots, p_d$.

Mini-max formulation:



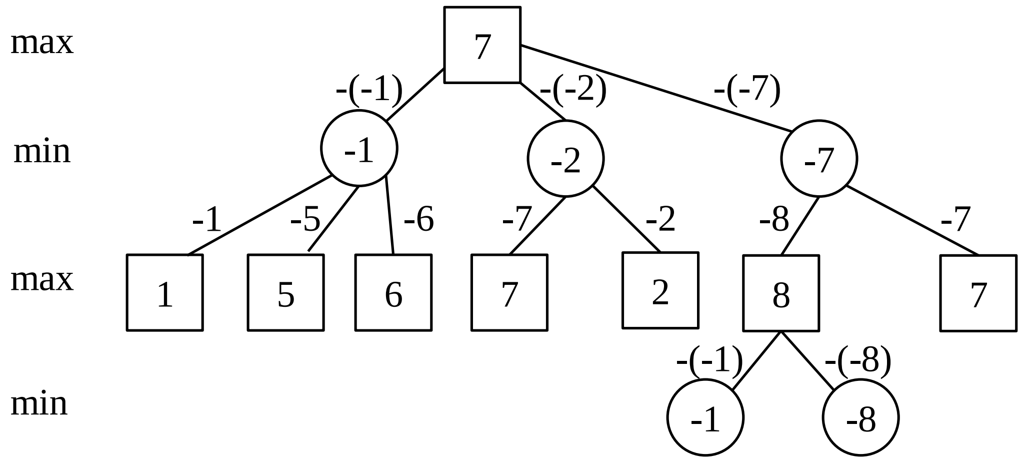
- Mini-max formulation

$$F(p) = \begin{cases} f(p) & \textbf{if } d = 0 \\ max\{G(p_1), \ldots, G(p_d)\} & \textbf{if } d > 0 \end{cases}$$

$$G(p) = \begin{cases} g(p) & \textbf{if } d = 0 \\ min\{F(p_1), \ldots, F(p_d)\} & \textbf{if } d > 0 \end{cases}$$

  – An indirect recursive formula!
  – Equivalent to AND-OR logic.

Nega-max formulation



- Nega-max formulation:
  - Let *F(p)* be the greatest possible value achievable from position *p* against the optimal defensive strategy.

$$F(p) = \begin{cases} f(p) & \textbf{if } d = 0 \\ max\{-F(p_1), \ldots, -F(p_d)\} & \textbf{if } d > 0 \end{cases}$$

  - Equivalent to NOR or NAND logic.

Algorithm: Nega-max

- Algorithm F(position p)

  determine the successor positions $p_1, \ldots, p_d$

  if $d = 0$, then return $f(p)$ else

  begin

  $m := -\infty$

  for $i := 1$ to d do

  $t = -F(p_i)$

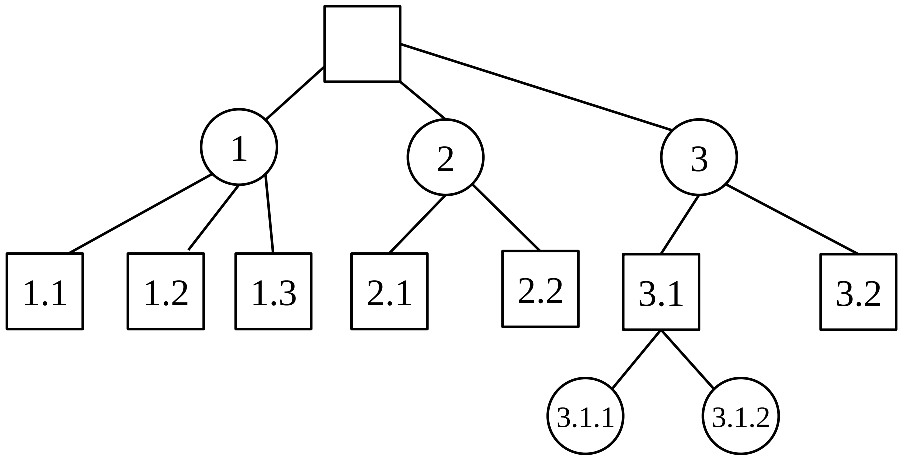
  if $t > m$ then $m = t$

  end

  return m

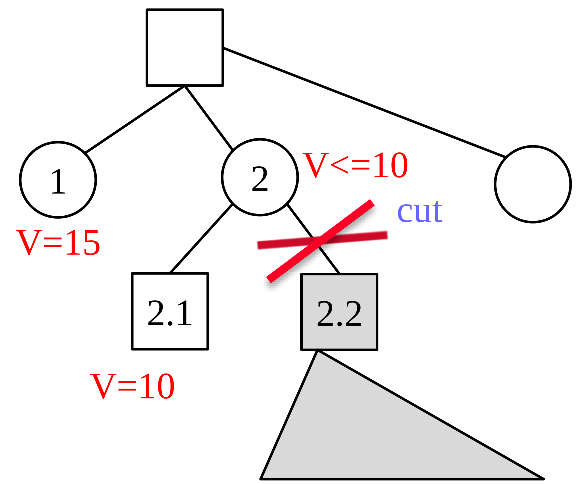
- A brute-force method to try all possibilities!

Representation



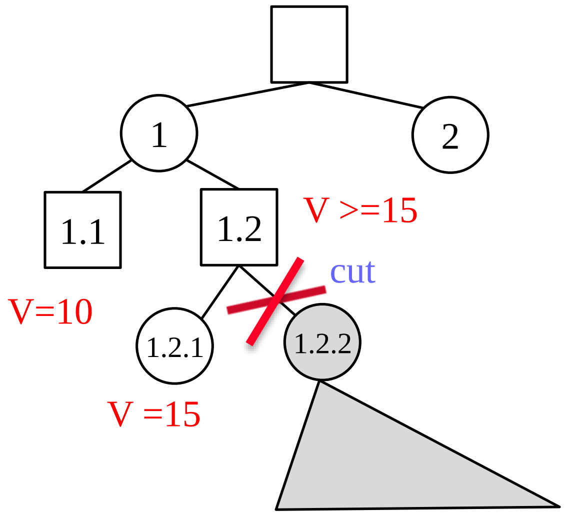
- From the root, number the node in a search tree by a sequence of integers a.b.c.d · · ·
  - Meaning from the root, you first take the *a*th branch, then the *b*th branch, and then the *c*th branch, and then the *d*th branch · · ·
  - The root is specified as an empty sequence.
  - This is called “Dewey decimal system”.

Alpha Cut-off



- On a max node
  - Assume you have finished exploring the first branch 1 and obtain the best value from that branch as bound.
  - You now search the second branch 2.
  - Assume branch at 2.1 returns a value of < bound.
    - Then no need to evaluate 2.2, 2.3, …, at all.
  - Since the best possible value for 2 is <= bound (10),
    - the root should choose the first branch as the current best solution.

Beta Cut-off

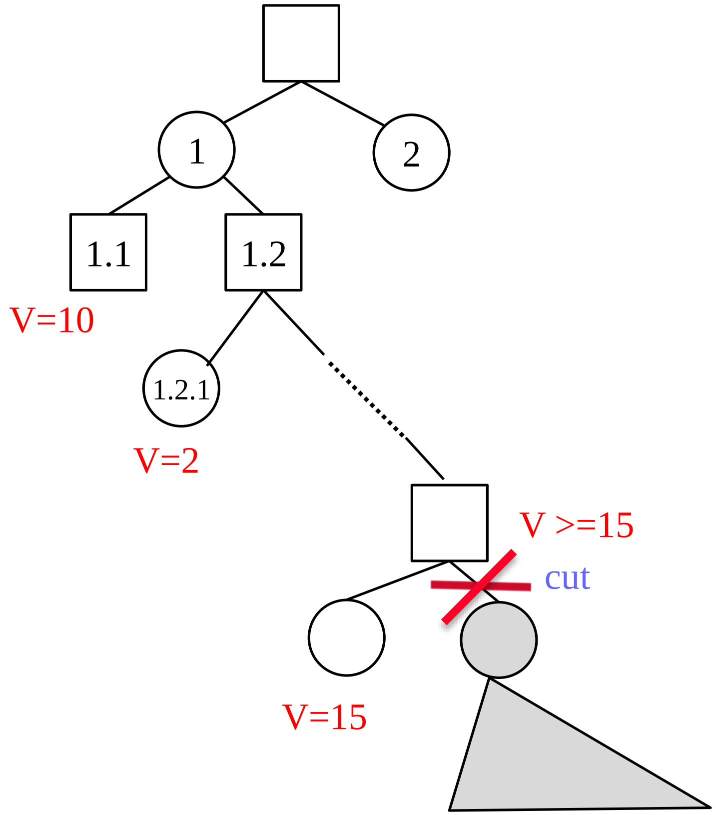


- On a min node
  - Assume you have finished exploring the first branch 1.1 at 1 and obtain the best value from that branch as bound (10).
  - You now search the second branch 1.2.
  - Assume branch at 1.2.1 returns a value (15) of > bound.
    - Then no need to evaluate 1.2.2, 1.2.3, …, at all.
  - Since the best possible value for 1.2 is > bound,
    - choose the first branch as the current best solution.

Deep Cut-off

- Shallow (alpha) cut-off (See slides before)
  - For a min node *u*, the branch of its elder brother produces a lower bound V_l for its parent.
  - The first branch of *u* produces an upper bound V_u for *v*.
  - If $V_l > V_u$, then there is no need to evaluate the second branch of *u*.
- Deep (alpha) cut-off:
  - Def: For a node *u* in a tree and a positive integer *g*, *Ancestor*(*g*, *u*) is the direct ancestor of *u* by tracing the parent's link *g* times.
  - When the lower bound V_l is produced at and propagated from *u*'s great grand parent, i.e., *Ancestor*(3,*u*), or any *Ancestor*($2i + 1$, *u*).
- Similar properties for beta cut-off.

Deep Cut-off



Illustration

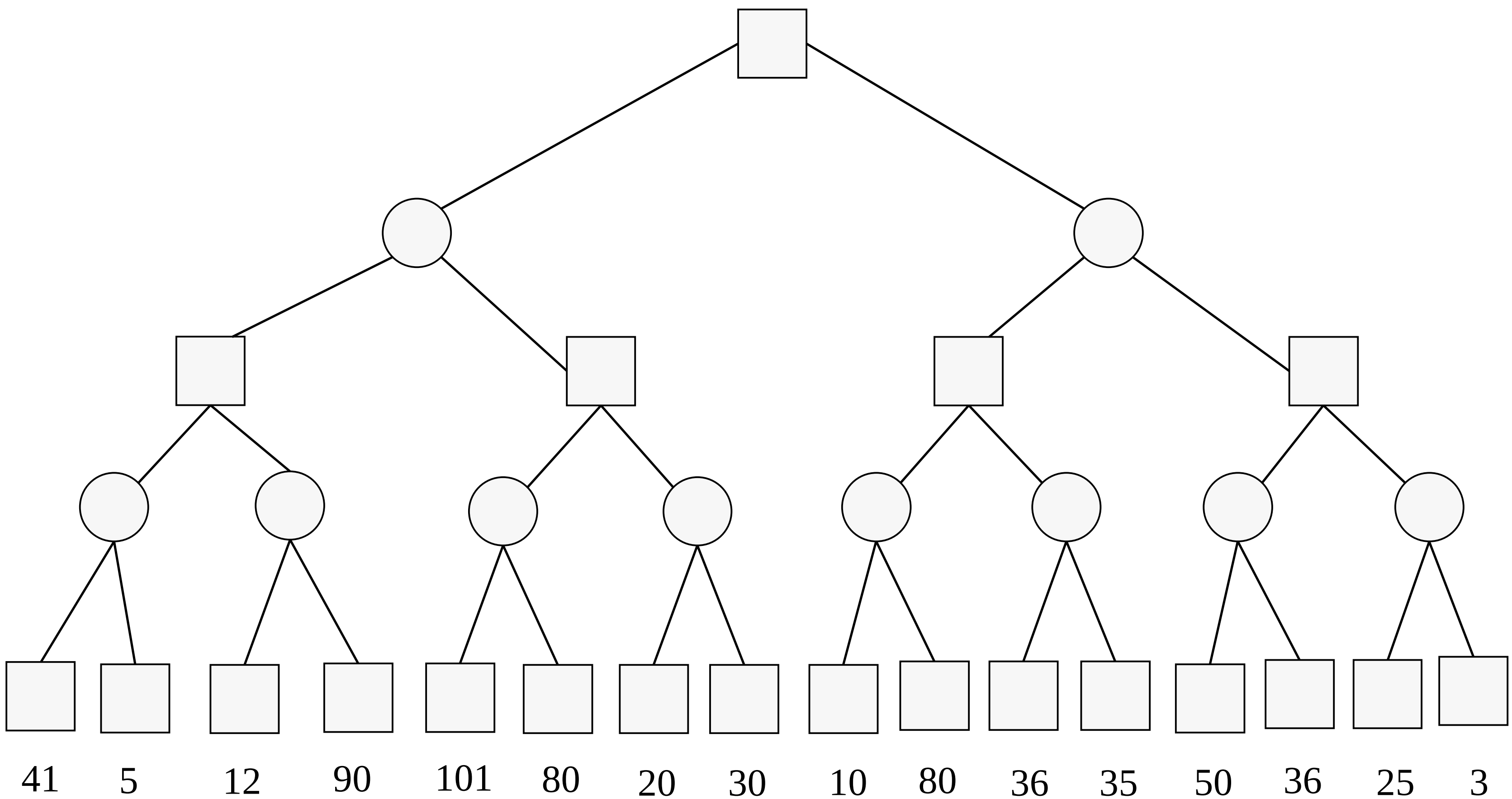


Illustration of Alpha-Beta Search

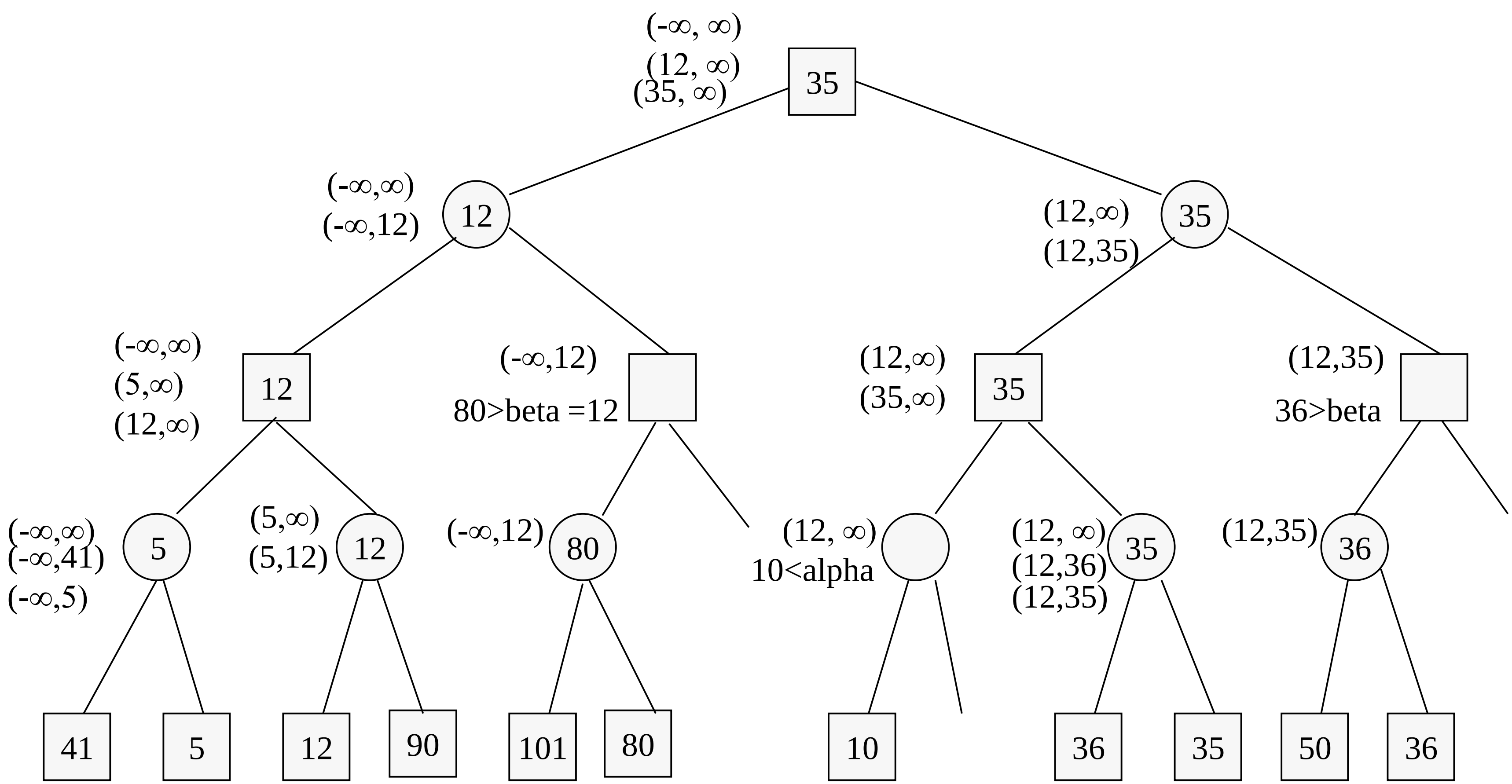
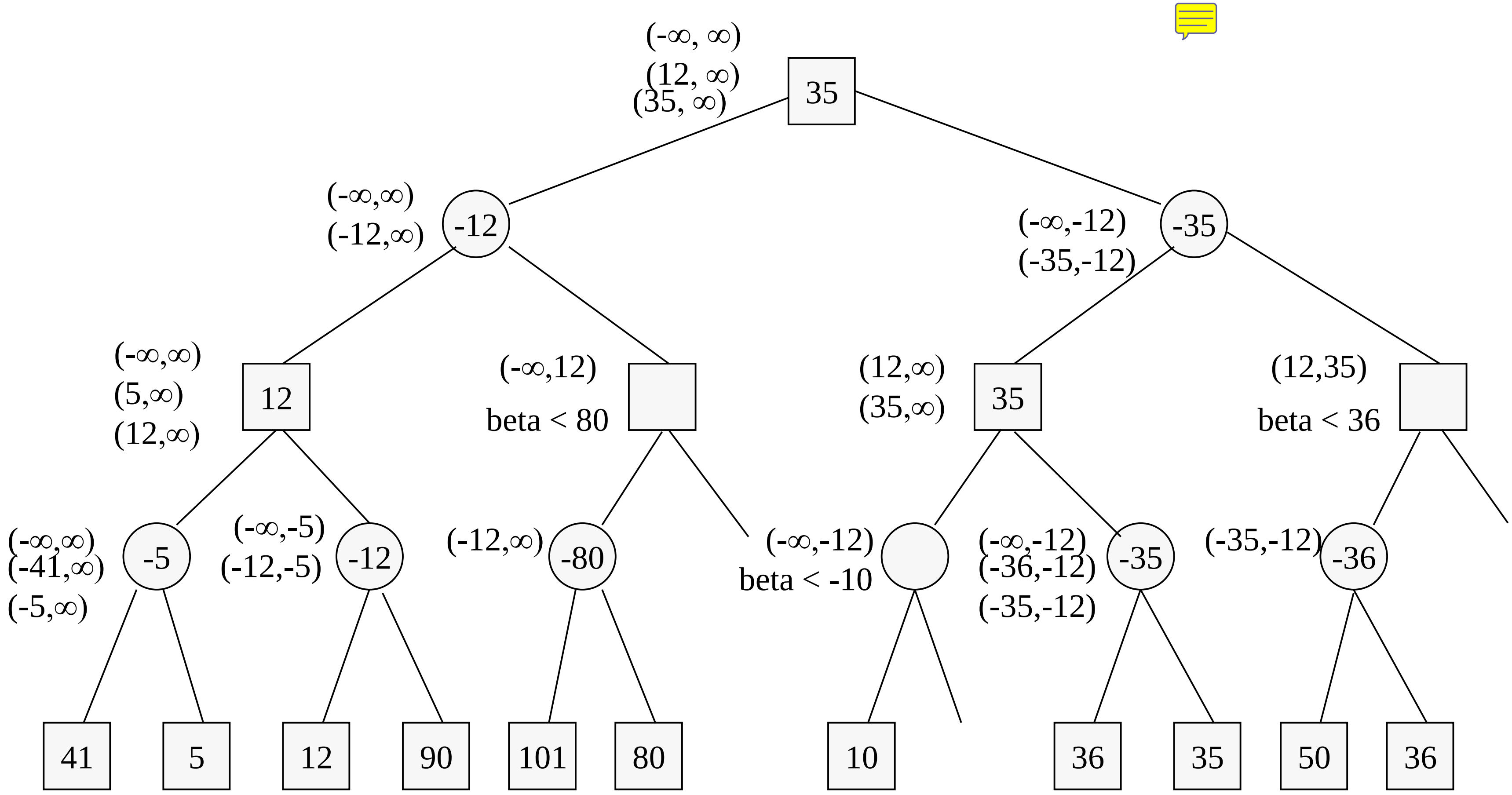


Illustration of Alpha-Beta Search



Alpha-beta Pruning Algorithm: Mini-Max

- Algorithm *F2'*(position *p*, integer *alpha*, integer *beta*)

  determine the successor positions $p_1, \ldots, p_d$
  if $d = 0$, then return $f(p)$ else begin
      $m := alpha$
      for $i := 1$ to d do
          $t := G2'(p_i, m, beta)$
          if $t > m$ then $m := t$
          if $m \geqq$ beta then return(m) //cutoff
  end;
  return m

- Algorithm *G2'*(position *p*, integer *alpha*, integer *beta*)

  determine the successor positions $p_1, \ldots, p_d$
  if $d = 0$, then return $g(p)$ else begin
      $m := beta$
      for $i := 1$ to d do
          $t := F2'(p_i, alpha, m)$
          if $t < m$ then $m := t$
          if $m \leqq$ alpha then return(m) //cutoff
  end; return m

Alpha-beta pruning algorithm: Nega-max

- Algorithm *F2*(position *p*, integer *alpha*, integer *beta*)

  determine the successor positions $p_1, \ldots, p_d$

  if $d = 0$, then return $f(p)$ else

  begin

  $m := alpha$

  for $i := 1$ to d do

  t $:= -F2(p_i, -beta, -m)$

  if $t > m$ then $m := t$

  if $m \geqq beta$ then return(m)

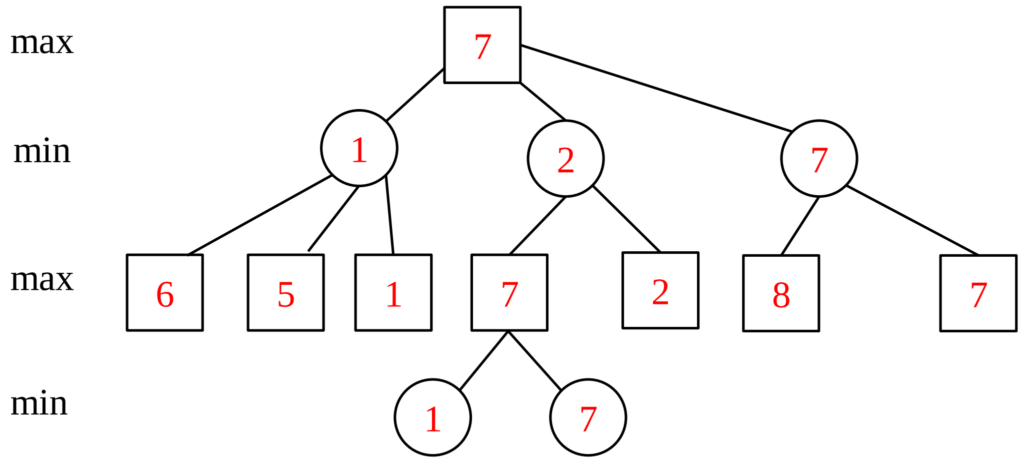
  end

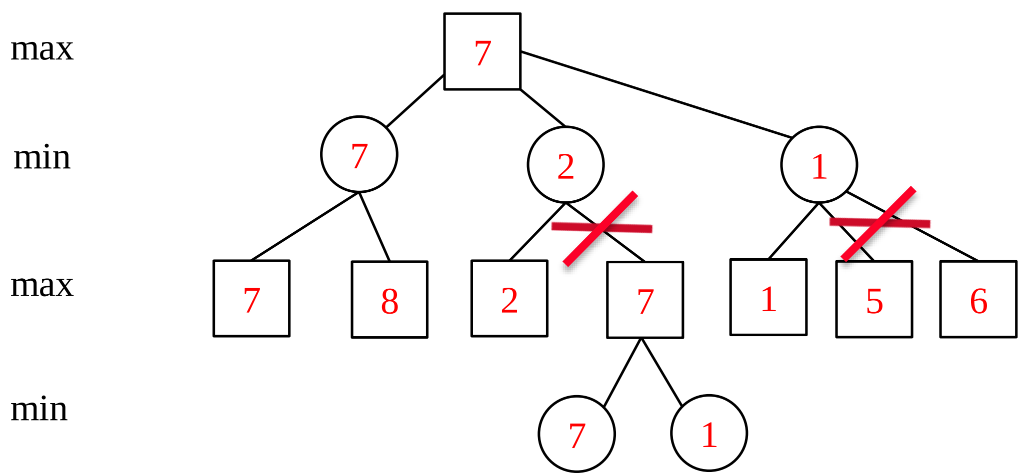
  return m

Properties and Comments

- Properties:
  - *alpha* < *beta*
  - *F2(p, alpha, beta)* ≦ *alpha*, if *F(p)* ≦ *alpha*
  - *F2(p, alpha, beta)* = *F(p)*, if *alpha* < *F(p)* < *beta*
  - *F2(p, alpha, beta)* ≧ *beta*, if *F(p)* ≧ *beta*
  - $F2(p, -\infty, +\infty) = F(p)$
- Comments:
  - *F2(p, alpha, beta)*: find the best possible value according to a mini-max formula for the position p with the constraints that
    - If *F(p)* is less than the lower bound *alpha*, then *F2(p, alpha, beta)* returns a terminal positions value that is less than *alpha*.
    - If *F(p)* is more than the upper bound *beta*, then *F2(p, alpha, beta)* returns a terminal positions value that is more than *beta*.
  - The meanings of *alpha* and *beta* during searching:
    - For a max node: the current best value is at least *alpha*.
    - For a min node: the current best value is at most *beta*.

Examples with Different Ordering





Analysis of a Possible Best Case

- Q: In the best case, what branches are cut?
- Definitions:
  - A path in a search tree is a sequence of numbers indicating the branches selected in each level using the Dewey decimal system.
  - A position is denoted as a path $a_1.a_2.\cdots.a_l$ from the root.
  - A position $a_1.a_2.\cdots.a_l$ is critical if
    - $a_i = 1$ for all even values of i or
    - $a_i = 1$ for all odd values of i
    - Examples: 2.1.4.1.2, 1.3.1.5.1.2, 1.1.1.2.1.1.1.3 and 1.1 are critical
    - Examples: 1.2.1.1.2 is not critical
  - A perfect-ordering tree:

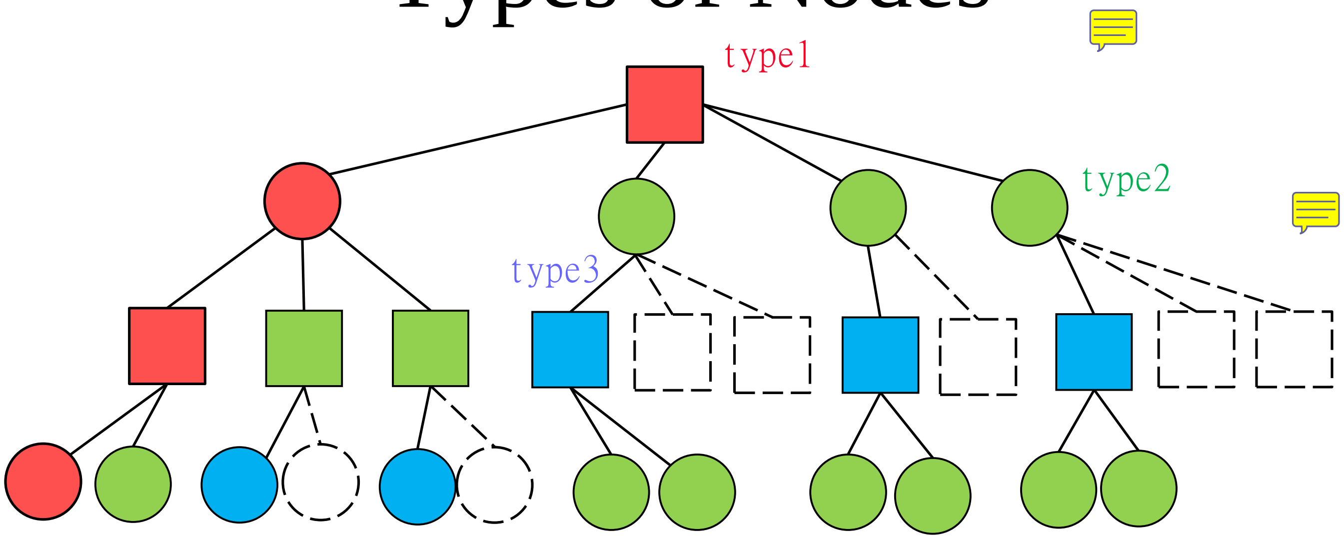
$$F(a_1.\cdots.a_\ell) = \begin{cases} f(a_1.\cdots.a_\ell) & \textbf{if } a_1.\cdots.a_\ell \textbf{ is a terminal} \\ -F(a_1.\cdots.a_\ell.1) & \textbf{otherwise} \end{cases}$$

    - The first successor of every non-terminal position gives the best possible value.

Critical Positions

- The minimum positions needed to derive the value.
- Theorem 1: *F2* examines precisely the critical positions of a perfect-ordering tree.

Types of Nodes



- Classification of critical positions $a_1.a_2.\cdots.a_l$:
  - type 1: all the a_i are 1;
  - type 2: if a_j is its first entry such that $a_j > 1$, for all odd k, $a_{j+k} = 1$;
    - All the children of type 1 nodes, except for the first.
    - All the children of type 3 nodes.
  - type 3: the first child of type 2 nodes.

Proof Sketch for Theorem 1

- Properties (invariants)
  - A type 1 position p is examined by calling F2(p, $-\infty$, ∞)
    - p's first successor p_1 is type 1 ➔ $F(p_1) = -F(p_1) \neq \pm\infty$
    - p's other successors $p_2, \ldots, p_d$ are of type 2
      ➔ p_i, $i > 1$, are examined by calling $F2(p_2, -\infty, F(p_2))$
  - A type 2 position p is examined by calling $F2(p, -\infty, beta)$ where $-\infty < beta \leqq F(p)$
    - p's first successor p_1 is type 3 ➔ $F(p) = -F(p_1)$
    - p's other successors $p_2, \ldots, p_d$ are not examined
  - A type 3 position p is examined by calling $F2(p, alpha, \infty)$ where $\infty > alpha \geqq F(p)$
    - p's successors $p_1, \ldots, p_d$ are of type 2
    - they are examined by calling $F2(p_i, -\infty, -alpha)$
- Using an induction argument to prove all and also only critical positions are examined.

Analysis: Best Case

- Corollary 1: Assume each position has exactly d successors
  - The alpha-beta procedure examines exactly
    $$d^{\lceil l/2 \rceil} + d^{\lfloor l/2 \rfloor} - 1$$
    positions on level l.
- Proof:
  - There are $d^{\lceil l/2 \rceil}$ sequences of the form $a_1.a_2.\cdots.a_l$ with $1 \leqq a_i \leqq d$ for all i such that $a_i = 1$ for all odd values of i.
  - There are $d^{\lfloor l/2 \rfloor}$ sequences of the form $a_1.a_2.\cdots.a_l$ with $1 \leqq a_i \leqq d$ for all i such that $a_i = 1$ for all even values of i.
  - We substrate 1 for the sequence $1.1.\cdots.1.1$ which are counted twice.

Analysis: Average Case

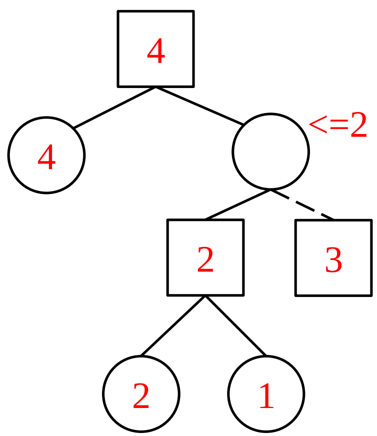
- Assumptions: Let a random game tree be generated in such a way that
  - each position on level j has probability q_j of being nonterminal
  - has an average of d_j successors
- Properties of the above random game tree
  - Expected number of positions on level l is $d_0.d_1. \cdot \cdot \cdot .d_{l-1}$.
  - Expected number of positions on level l examined by an alpha-beta procedure assumed the random game tree is perfectly ordered is

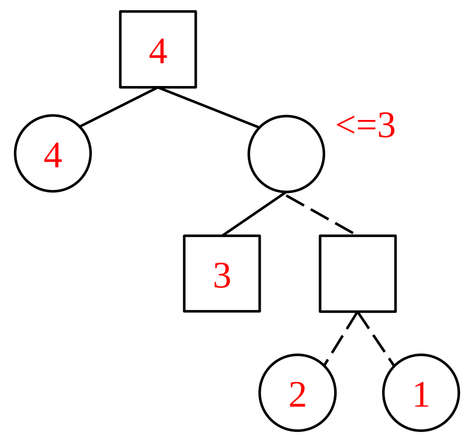
    $d_0 q_1 d_2 q_3 \cdot \cdot \cdot d_{l-2} q_{l-1} + q_0 d_1 q_2 d_3 \cdot \cdot \cdot q_{l-2} d_{l-1} + q_0 q_1 \ldots q_{l-1}$ if l is even;

    (skipped for odd)
- Proof sketch:
  - If x is the expected number of positions of a certain type on level j, then xd_j is the expected number of successors of these positions, and xq_j is the expected number of "number 1" successors.
  - The above numbers equal to those of Corollary 1 when $q_j = 1$ and $d_j = d$ for $0 \leqq j < l$.

Perfect Ordering is Not Always Best

- Intuitively, we may "think" alpha-beta pruning would be most effective when a game tree is perfectly ordered.
  - That is, when the first successor of every position is the best possible move.
  - This is not always the case!





- Truly optimum order of game trees traversal is not obvious.

Theorem 2

- Theorem 2: Alpha-beta pruning is optimum in the following sense:
  - Given any game tree and any algorithm which computes the value of the root position, there is a way to permute the tree
    - by reordering successor positions if necessary;
  - so that every terminal position examined by the alpha-beta method under this permutation is examined by the given algorithm.
  - Furthermore if the value of the root is not 1 or −1, the alpha-beta procedure examines precisely the positions which are critical under this permutation.

Questions

- What is a good move ordering?
  - It may not be good to search the best possible move first.
  - It is best to cut off a branch with more nodes.
- How about the case when the tree is not uniform?
- What is the effect of using iterative-deepening alpha-beta cut off?
- How about the case for searching a game graph instead of a game tree?
  - Can some nodes be visited more than once?

History

- McCarthy thought of the method during the Dartmouth Summer Research Conference on Artificial Intelligence in 1956.
  - No formal specification of the algorithm was given at that time.
  - McCarthy's remarks at that conference led to the use of alpha-beta pruning in game-playing programs of the late 1950s.
  - [15] McCarthy, J. Personal communication, December 1, 1973.
- Samuel has stated that the idea was present in his checker-playing programs.
  - but he did not allude to it in his classic article [21] because he felt that the other aspects of his program were more significant.
  - [21] Samuel, A. L. Some studies in machine learning using the game of checkers. IBMJ. Res. and Develop. 3 (1959).
- The first published discussion of a method for game tree pruning appeared in Newell, Shaw and Simon's description [16] of their early chess program.
  - However, they illustrate only the "one-sided" technique used in procedure F1 above, so it is not clear whether they made use of "deep cutoffs".
- McCarthy coined the name "alpha-beta" when he first wrote a LISp program embodying the technique.
  - Hart and Edwards, who wrote a memorandum [10] on the subject in 1961.
- The first published account of alpha-beta pruning actually appeared in Russia, quite independently of the American work, in 1963.
  - [4] Brudno, A, L. Bounds and valuations for shortening the scanning of variations. Problemy Kibernet. 10 (1963), 141-150 (in Russian).
- Excellent presentations of the method appear in the textbooks by Nilsson [18, Section 4] and Slagle [23, pp. 16-24],
  - [18] Nilsson, N, J. Problem-Solving Methods in Artificial Intelligence. McGraw-Hill, New York. 1971.
- This paper is still valuable even when
  - alpha-beta pruning has been in use for more than 15 years