Contd…

- **Unit 3: Informed and Uninformed Search LH 8**
  - 3.1 Why search in AI?
  - 3.2 Blind search (Un-informed search)
    - 3.2.1 Breadth first search (BFS)
      - **Variations: Uniform cost search**
    - 3.2.2 Depth first search (DFS)
      - **Variations: Depth limited search, Iterative deepening DFS**
  - 3.3 Heuristic search (Informed search)
    - 3.3.1 Hill climbing
      - The Foothills Problem
      - The Plateau Problem
      - The Ridge Problem
    - 3.3.2 Greedy (Best-first) search
    - 3.3.3 A* algorithm search

Contd…

- **Unit 3: Informed and Uninformed Search contd…**
  - 3.4 General Problem Solving (GPS): Problem solving agents
    - 3.4.1 Constraint satisfaction problem
      - Constraint Satisfaction Search
      - AND/OR trees
      - The bidirectional search
      - Crypto-arithmatics
  - 3.5 Game playing and AI
    - 3.2.1 Game Trees and Mini-max Evaluation
    - 3.2.2 Heuristic Evaluation
    - 3.2.3 Min-max algorithm (search)
    - 3.2.4 Min-max with alpha-beta
    - 3.2.5 Games of chance
    - 3.2.6 Game theory

Four General steps in problem solving

- Problem solving is a systematic search through a range of possible actions in order to reach some predefined goal or solution.
- For problem solving a kind of goal based agent called problem solving agents are used.
- This agent first formulates a goal and a problem, searches for a sequence of actions that would solve the problem, and then executes the actions one at a time. When this is complete, it formulates another goal and starts over.
- This over all process is described in the following four steps:
  - Goal formulation
  - Problem formulation
  - Search
  - Execute

Problem Formulation

- A problem can be defined formally by five components:
  - Initial state
  - Actions
  - Transition model
  - Goal Test
  - Path Cost

Contd…

- **Initial state:** The state from which agent start.
- **Actions:** A description of the possible actions available to the agent. During problem formulation we should specify the all possible actions available for each state "s".
- **Transition model:** A description of what each action does is called the transition model. For formulating transition model in problem formulation we take state "s" and action "a" for that state and then specify the resulting state "s'"
- **Goal Test:** Determine whether the given state is goal state or not.
- **Path Cost:** Sum of cost of each path from initial state to the given state

One way to formally define a problem: State space Representation

- The set of all states reachable from the initial state by any sequence of actions is called state space.
- The state space forms a directed graph in which nodes are states and the links between nodes are actions.
- A State space representation allows for the formal definition of a problem which makes the movement from initial state to goal state quite easy.
- **Disadvantage:** it is not possible to visualize all states for a given problem. Also, the resources of the computer system are limited to handle huge state space representation.

Contd…

- State Space representation of Vacuum World Problem: Vacuum world can be formulated as a problem as follows:
  - **States:** The state is determined by both the agent location and the dirt locations. The agent is in one of two locations, each of which might or might not contain dirt.
  - **Initial state:** Any state can be designated as the initial state.
  - **Actions:** In this simple environment, each state has just three actions: Left, Right, and Suck. Larger environment may might also include Up and Down.
  - **Transition Model:** The actions have their expected effects , except that moving Left in leftmost square, moving Right in the rightmost square, and sucking in a clean square having no effect. The complete state space is shown in figure below.
  - **Goal Test:** This checks whether all the squares are clean.
  - **Path cost:** Each step costs 1, so the path cost is the number of steps in the path.

Contd…

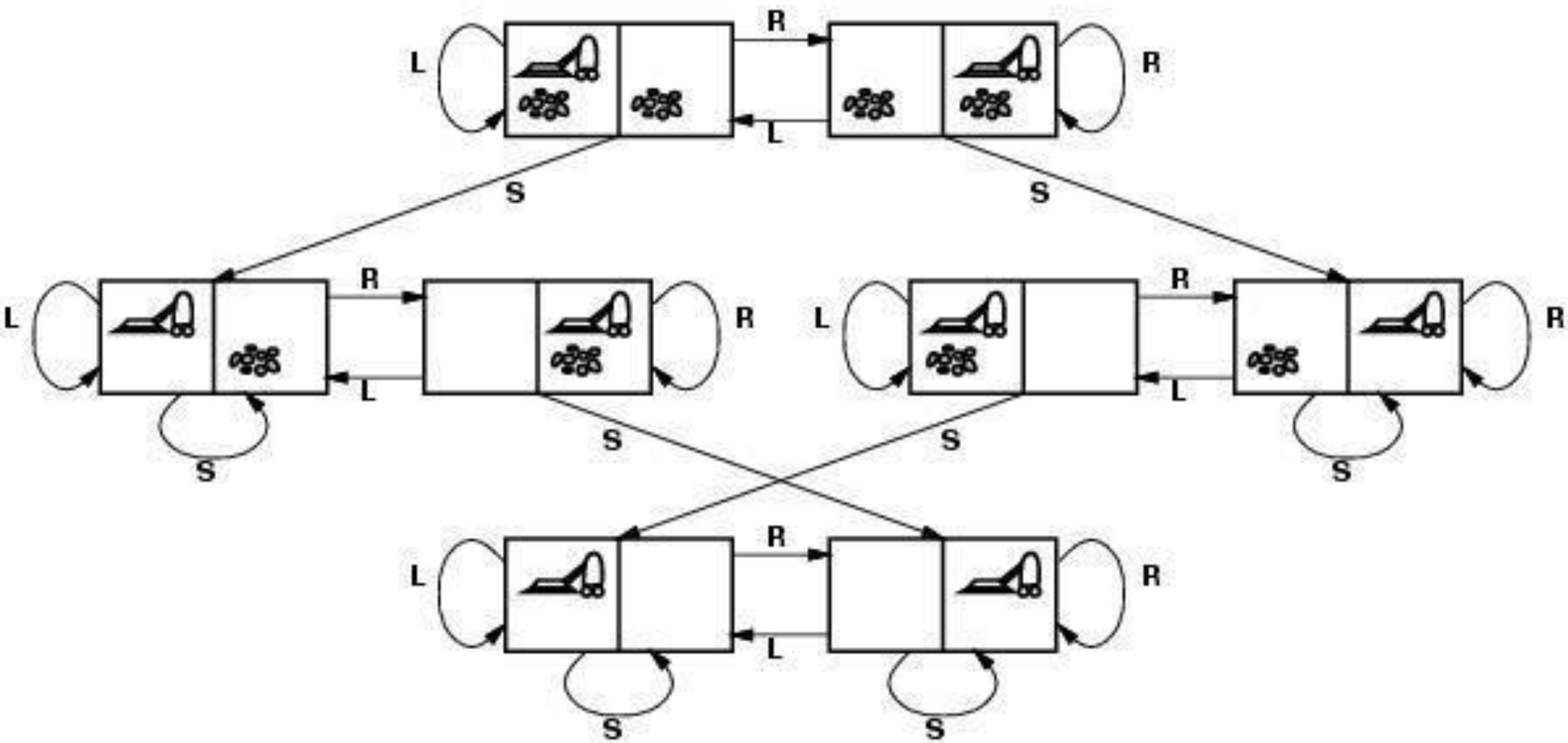


Fig: State space representation for the vacuum world

Here, links denote actions: L: left, R= Right, S= Suck

Searching For Solution

- Having formulated some problems, we now need to solve them.
- To solve a problem we should perform a systematic search through a range of possible actions in order to reach some predefined goal or solution.
- A solution is an action sequence, so search algorithms works by considering various possible action sequences.
- The possible action sequences starting at the initial state form a search tree with the initial state at the root; the branches are actions and the nodes correspond to states in the state space of the problem.

Contd…

- General search:
  - The searching process starts from the initial state (root node)and proceeds by performing the following steps:
    - Check whether the current state is the goal state or not?
    - Expand the current state to generate the new sets of states.
    - Choose one of the new states generated for search depending upon search strategy.
    - Repeat step 1 to 3 until the goal state is reached or there are no more state to be expanded.

Contd…

- The Importance of Search in AI:
  - Many of the tasks underlying AI can be phrased in terms of a search for the solution to the problem at hand.
  - Many goal based agents are essentially problem solving agents which must decide what to do by searching for a sequence of actions that lead to their solutions.
  - For the production systems, need to search for a sequence of rule applications that lead to the required fact or action.
  - For neural network systems, need to search for the set of connection weights that will result in the required input to output mapping.

Contd…

- Measuring problem Solving Performance:
  - The performance of the search algorithms can be evaluated in four ways:
  - **Completeness:** An algorithm is said to be complete if it definitely finds solution to the problem, if exist.
  - **Time complexity:** How long does it take to find a solution? Usually measured in terms of the number of nodes expanded during the search.
  - **Space Complexity:** How much space is used by the algorithm? Usually measured in terms of the maximum number of nodes in memory at a time
  - **Optimality/Admissibility:** If a solution is found, is it guaranteed to be an optimal one? For example, is it the one with minimum cost?

Classes of Search methods

- There are two broad classes of search methods:
- **Uninformed (or blind) search methods:**
  - Strategies have no additional information about states beyond that provided in the problem definition. All they can do is generate successors and distinguish a goal state from a non-goal state.
  - All search strategies are distinguished by the order in which nodes are expanded .
- **Heuristically informed search methods.**
  - Strategies that know whether one non-goal state is “more promising” than another are called informed or heuristic search strategies.
  - I.e., In the case of the heuristically informed search methods, one uses domain-dependent (heuristic) information in order to search the space more efficiently.

Uninformed (Or Blind) Search Methods:

- Blind search do not have additional information about state beyond the problem definition to search a solution in the search space.
- It proceeds systematically by exploring nodes either randomly or in some predetermined order.
- Based on the order in which nodes are expanded, it is of the following types:
  - Breadth First search (BFS)
    - Variation: Uniform cost search
  - Depth First search (DFS)
    - Variations: Depth limit search, Iterative deepening DFS.
  - Bidirectional search

Breadth First Search

- Breadth First search is a simple strategy in which the root node is expanded first, then all the successors of the root node are expanded next, then their successors, and so on.

- In general, All nodes are expended at a given depth in the search tree before any nodes at the next level are expanded until the goal reached.
  - I.e., Expand shallowest unexpended node.
- The search tree generated by the BFS is shown in figure below:

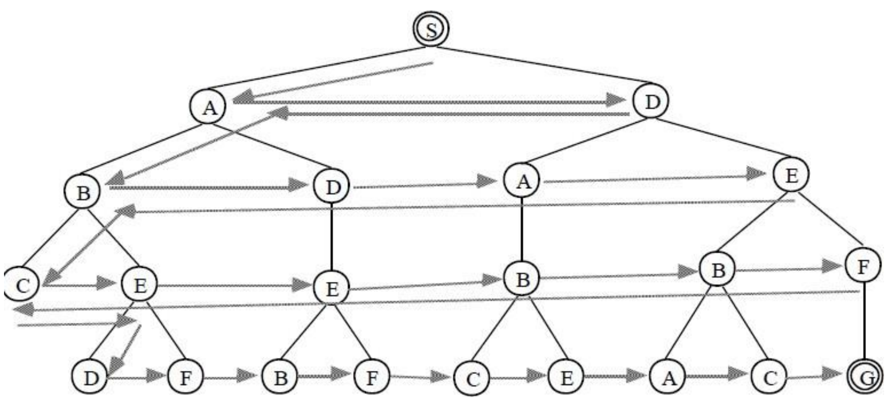
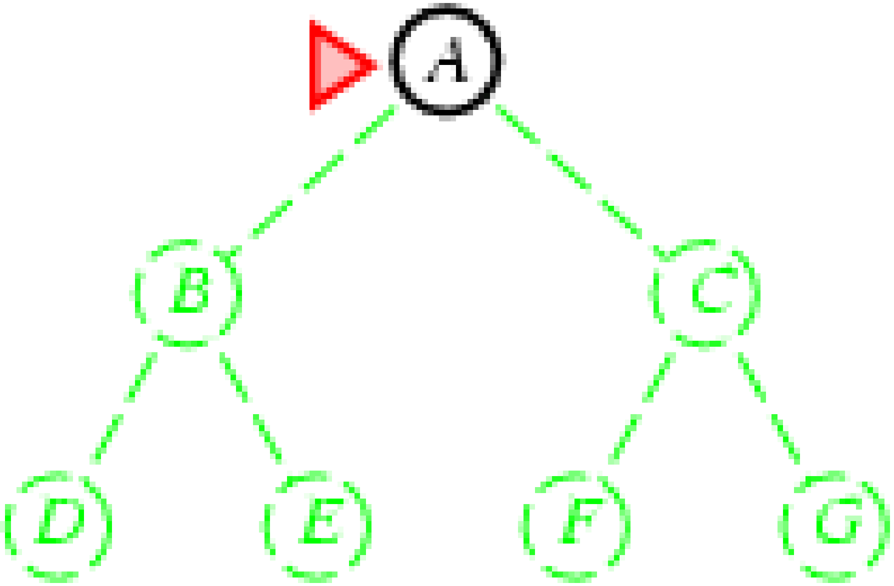


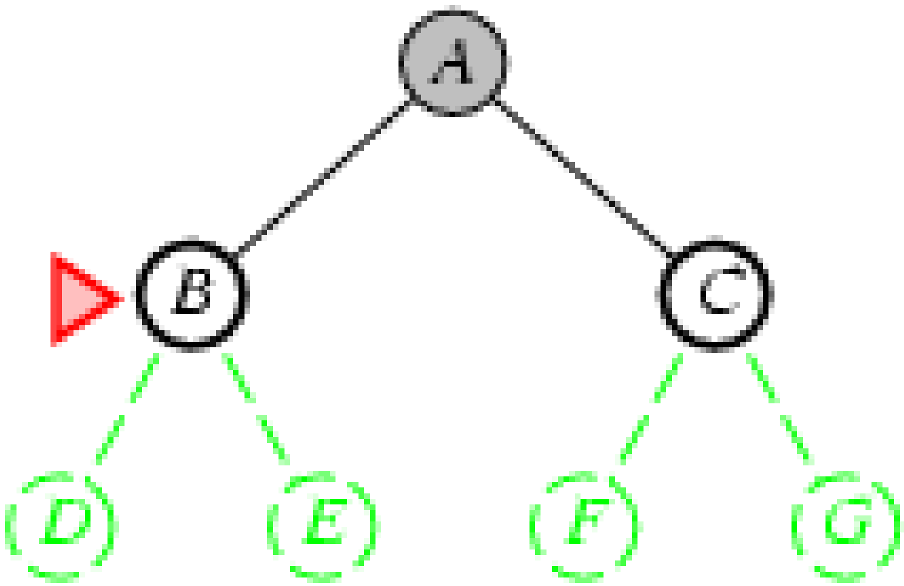
Fig: search Tree For BFS

Note: We are using the convention that the alternatives are tried in the left to right order.

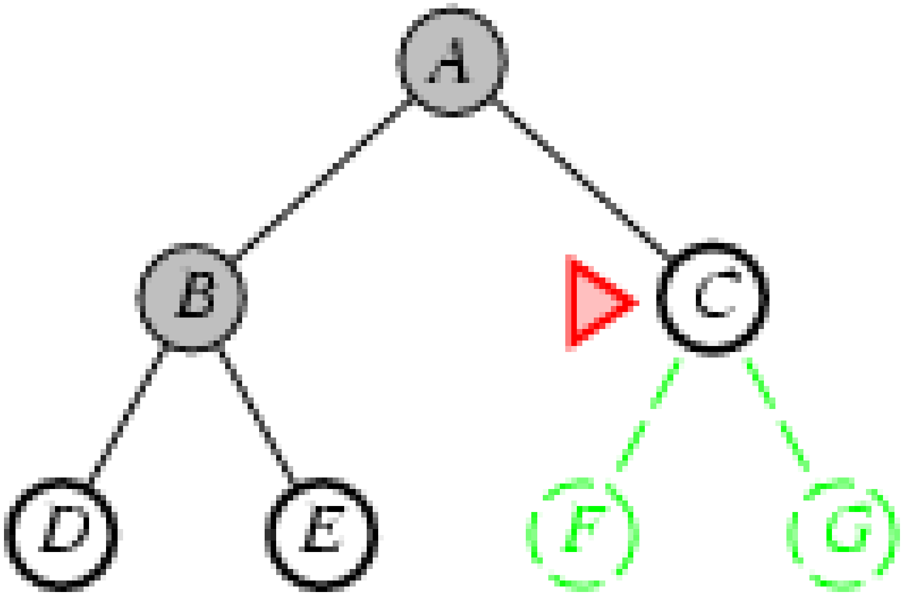
Breadth-first search



Breadth-first search



Breadth-first search



BFS Evaluation

- **Completeness**: Yes, if shallowest goal node is at some finite depth m and if b is finite.
- **Time Complexity:**
    - Let m is the maximum depth of the search tree. At worst the solution may lie at same depth m.
    - Similarly root has b successors, each node at the next level has b successors then total b^2
    - Worst case: expand all nodes except the last node at depth m.
    - Total nodes generated = $b + b^2 + b^3 + \ldots\ldots + b^m = O(b^m)$
- **Space Complexity:**
    - Each node generated must remain in memory hence,
    - Total nodes in memory = $b + b^2 + b^3 + \ldots\ldots + b^m = O(b^m)$
- Optimal: Since BFS expands shallowest node, the search may incur some extra intermediate nodes. Hence DFS may not be optimal

Uniform Cost Search

- The search begins at root node. The search continues by visiting the next node which has the least total cost from the root node. Nodes are visited in this manner until a goal is reached.

- Now goal node has been generated, but uniform cost search keeps going, choosing a node (with less total cost from the root node to that node than the previously obtained goal path cost) for expansion and adding a second path.

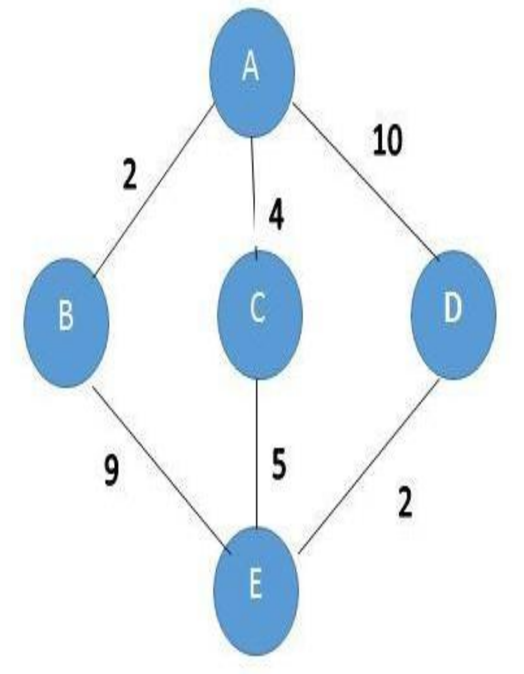
- Now the algorithm checks to see if this new path is better than the old one; if it is so the old one is discarded and new one is selected for expansion and the solution is returned.

Uniform cost search example1 (Find path from A to E)

- Expand A to B,C,D
- The path to B is the cheapest one with path cost 2.
- Expand B to E
  - Total path cost = 2+9 =11
- This might not be the optimal solution since the path

  AC as path cost 4 (less than 11)
- Expand C to E
  - Total path cost = 4+5 =9
- Path cost from A to D is 10 (greater than path cost, 9)

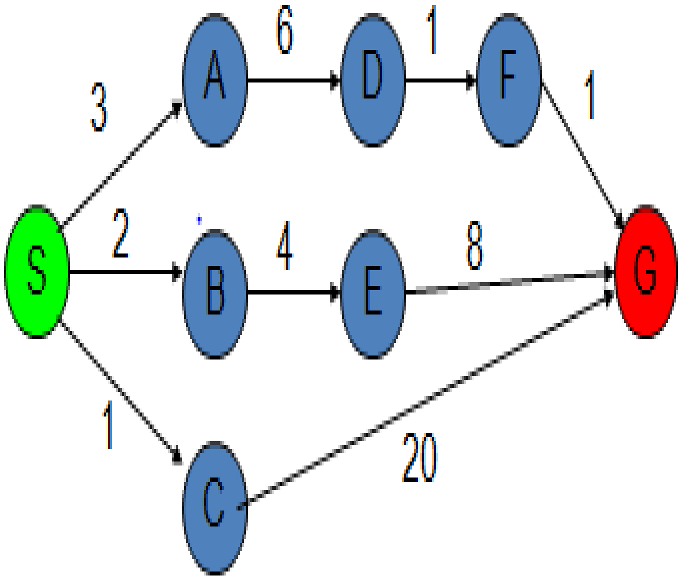
  Hence optimal path is ACE



Home work: Uniform cost search

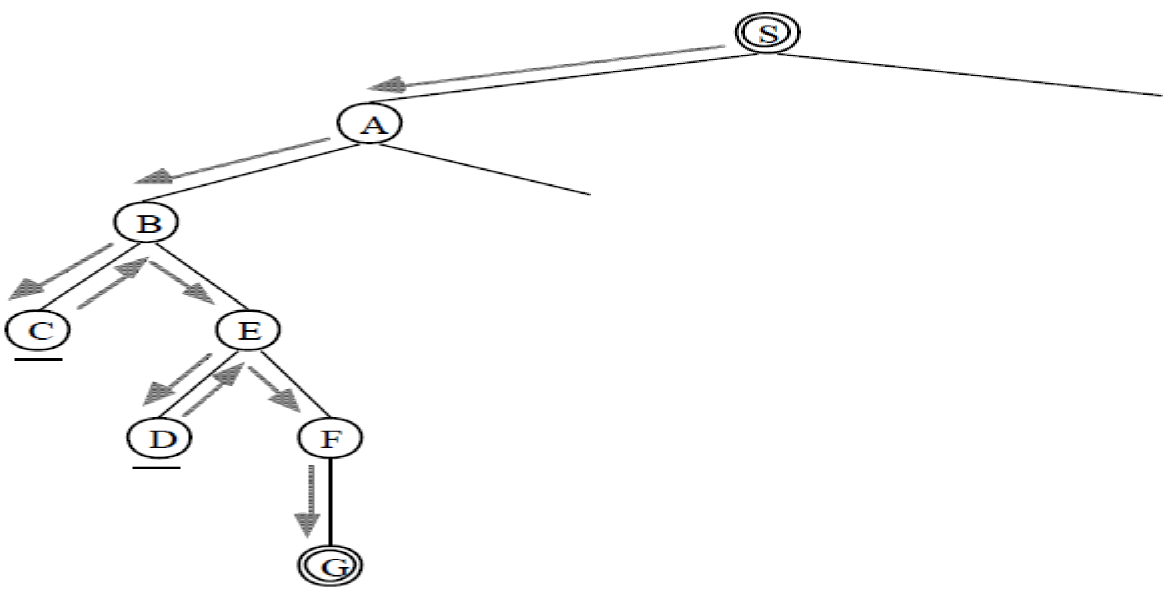
- The graph below shows the step-costs for different paths going from the start (S) to the goal (G).

- Use uniform cost search to find the optimal path to the goal.



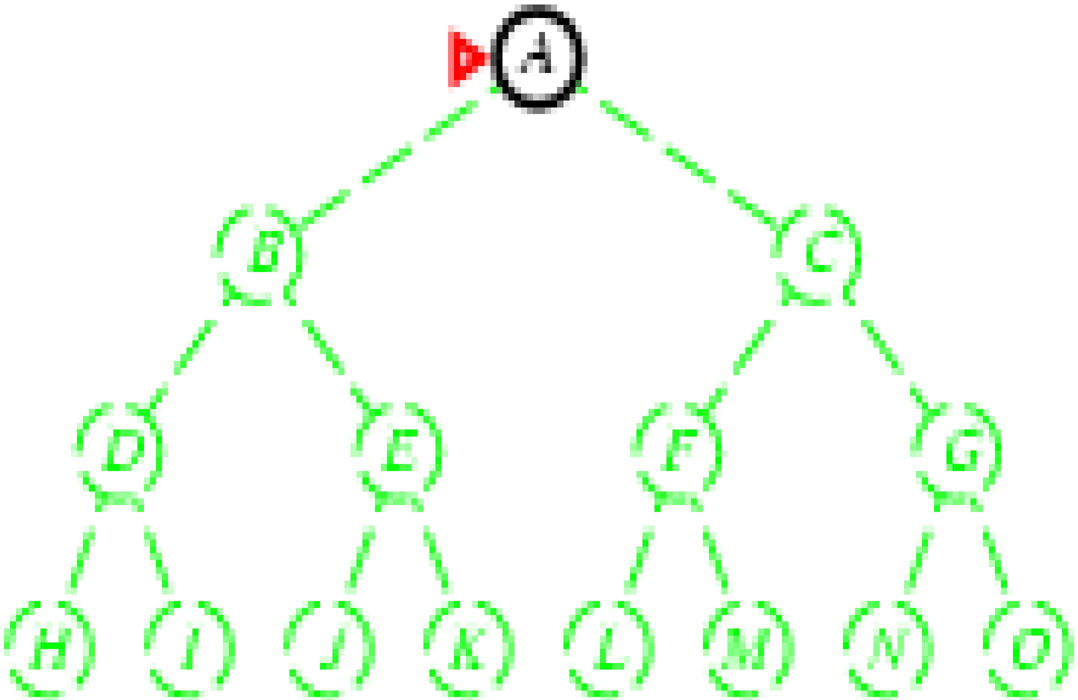
Depth First Search

- DFS also begins by expanding the initial node.
- Looks for the goal node among all the children of the current node before using the sibling of this node
- i.e. expand deepest unexpanded node(expand most recently generated deepest node first.).
- The search tree generated by the DFS is shown in figure below:



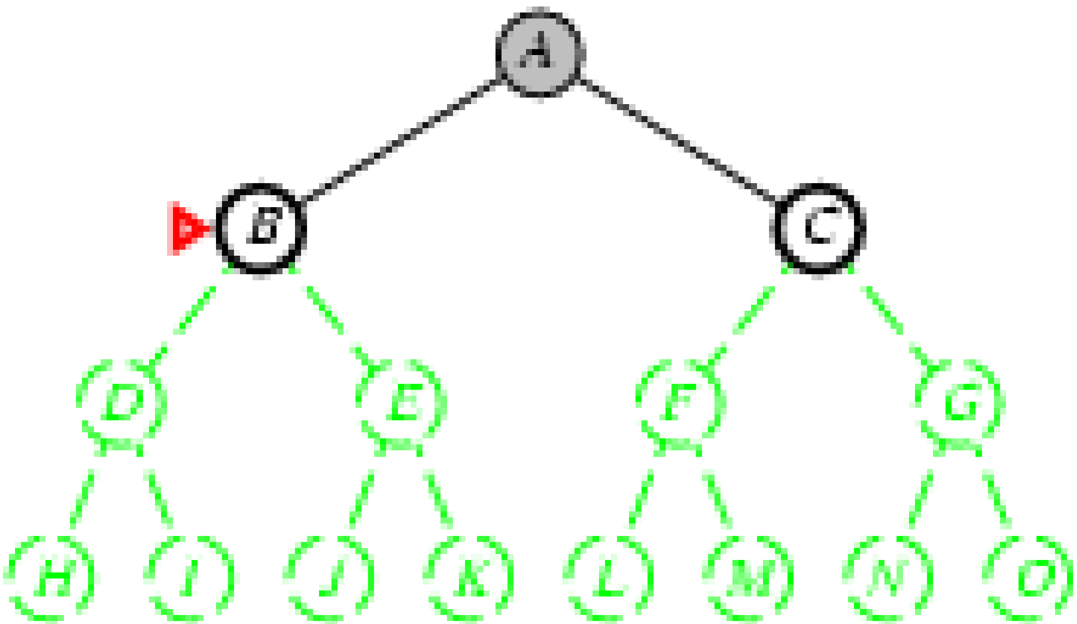
Depth-first search example

- Expand deepest unexpanded node
- Here initial state is A and goal state is M



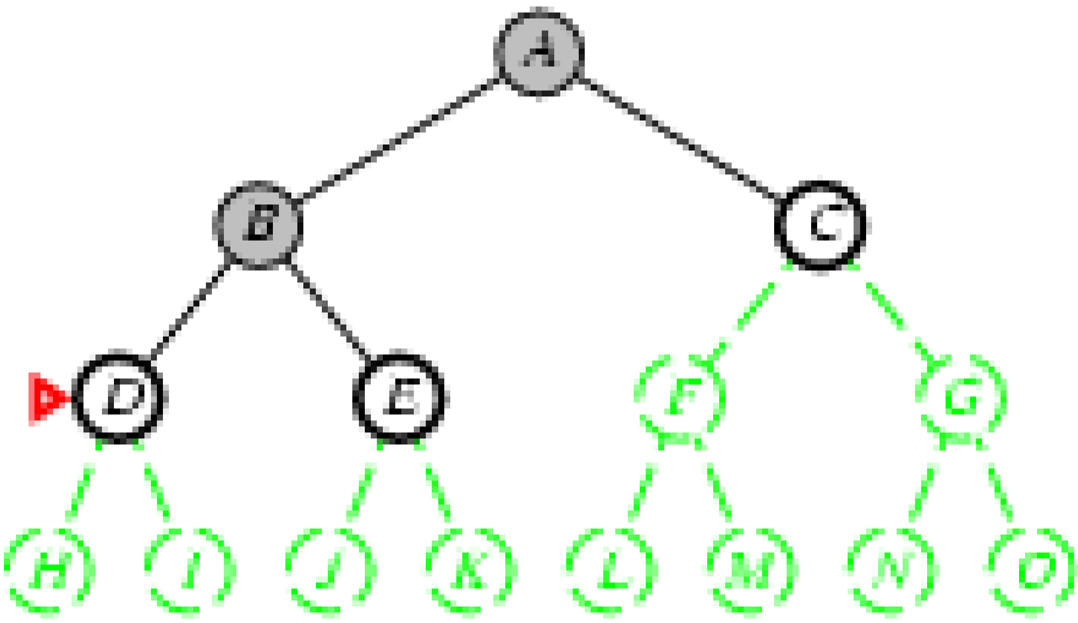
Depth-first search example

- Expand deepest unexpanded node



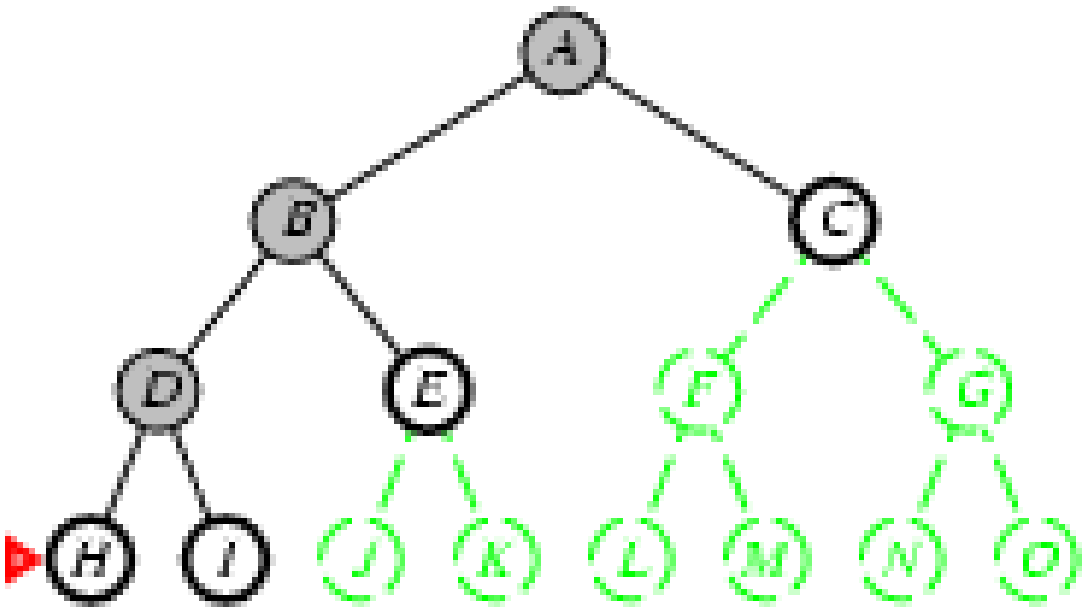
Depth-first search example

- Expand deepest unexpanded node



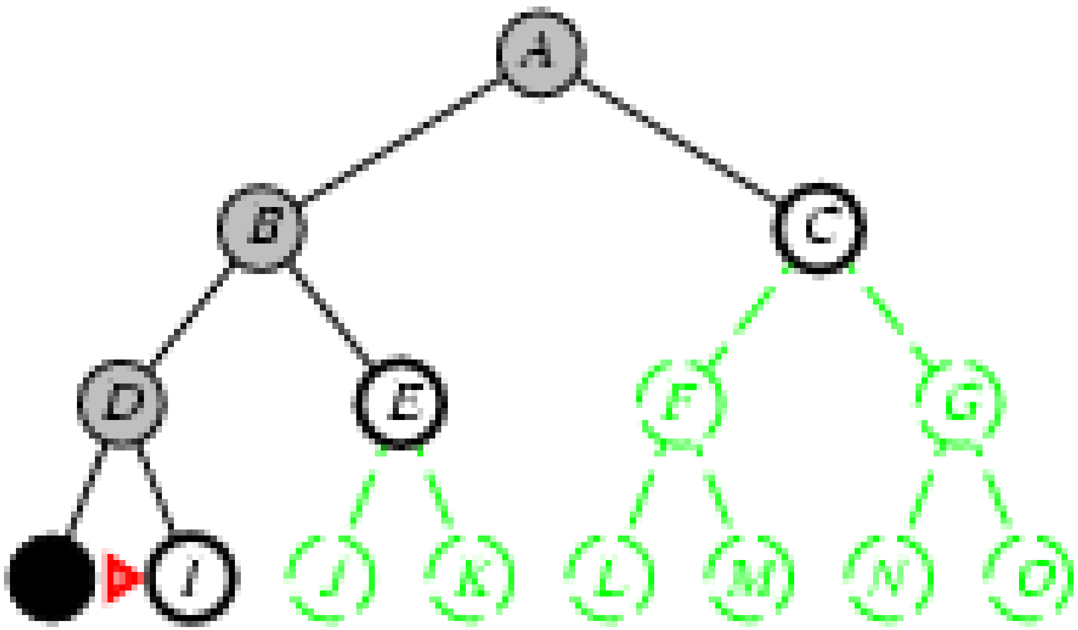
Depth-first search example

- Expand deepest unexpanded node



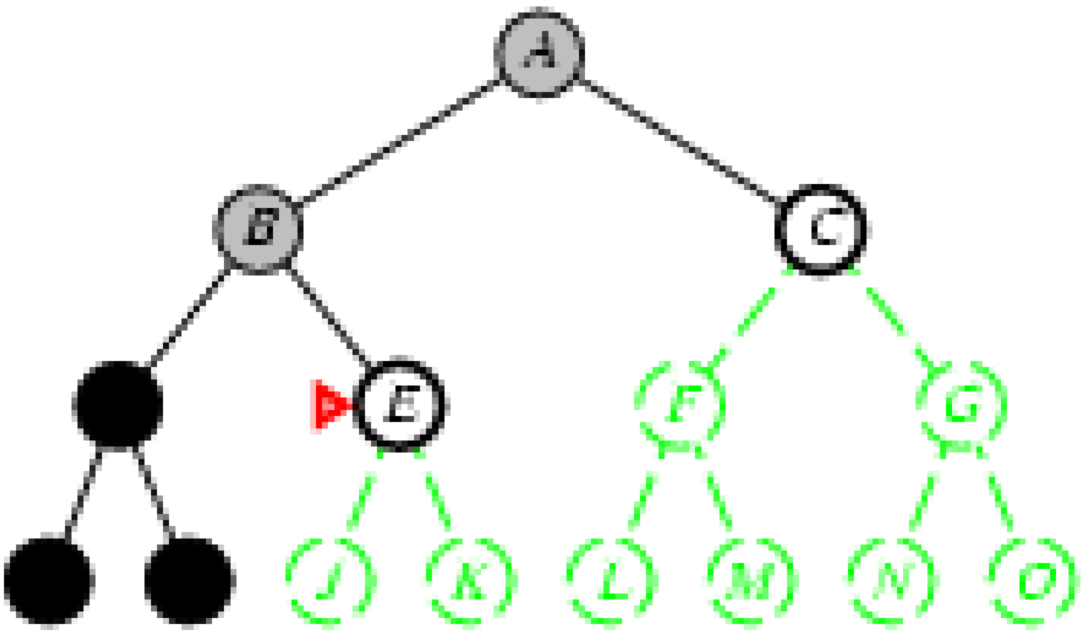
Depth-first search example

- Expand deepest unexpanded node



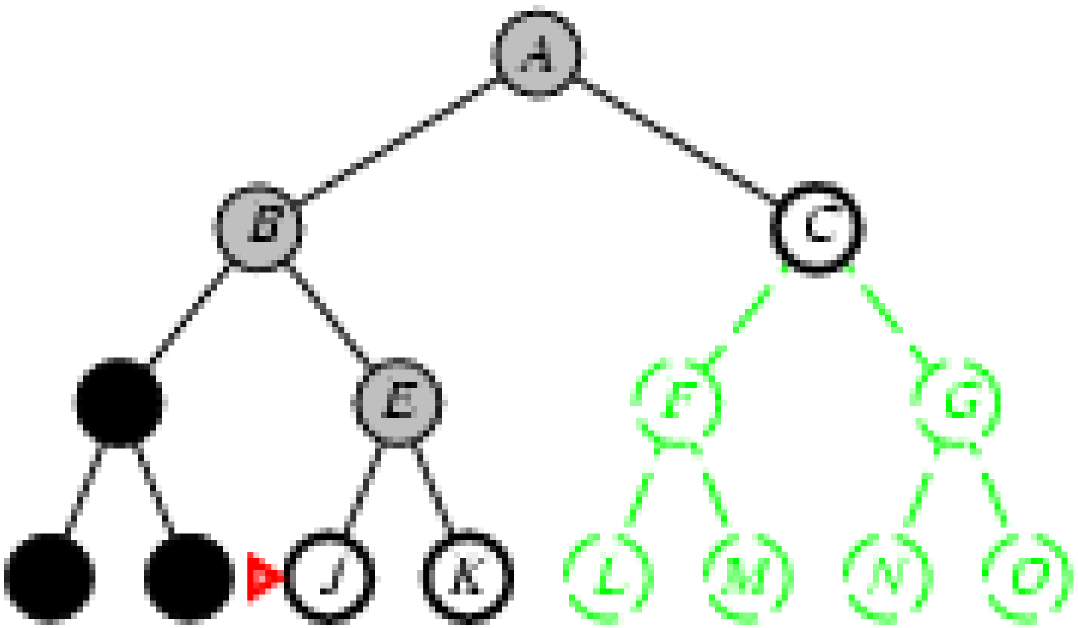
Depth-first search example

- Expand deepest unexpanded node



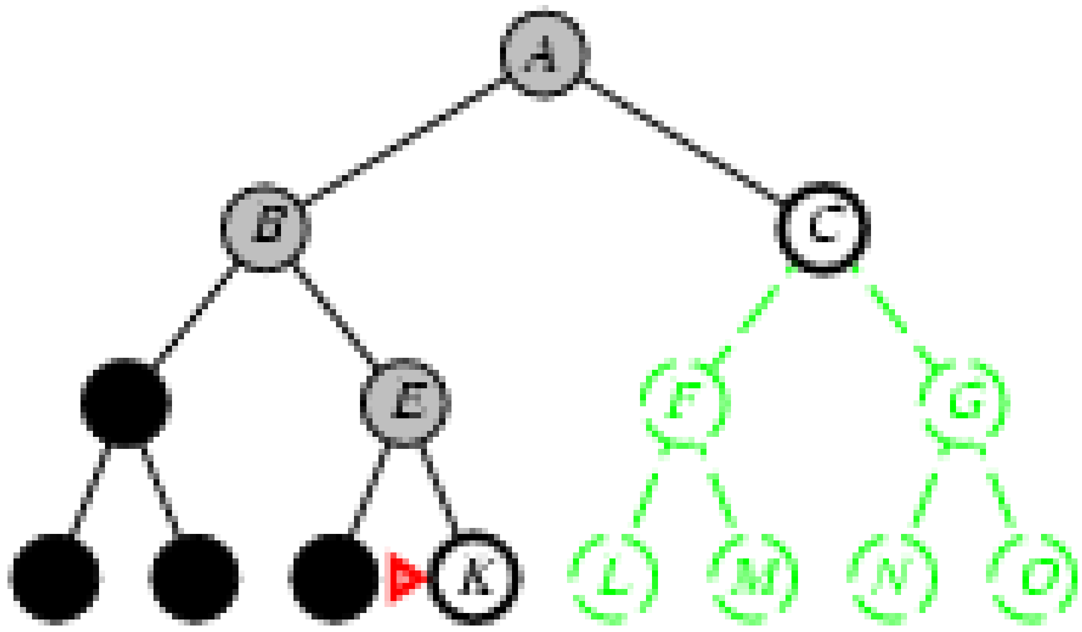
Depth-first search example

- Expand deepest unexpanded node



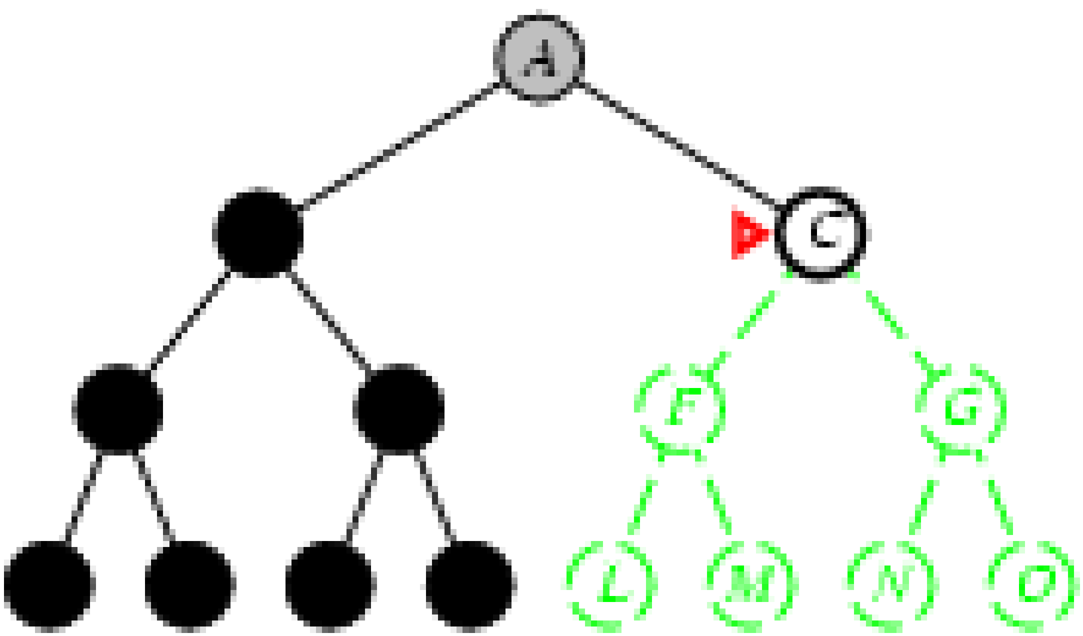
Depth-first search example

- Expand deepest unexpanded node



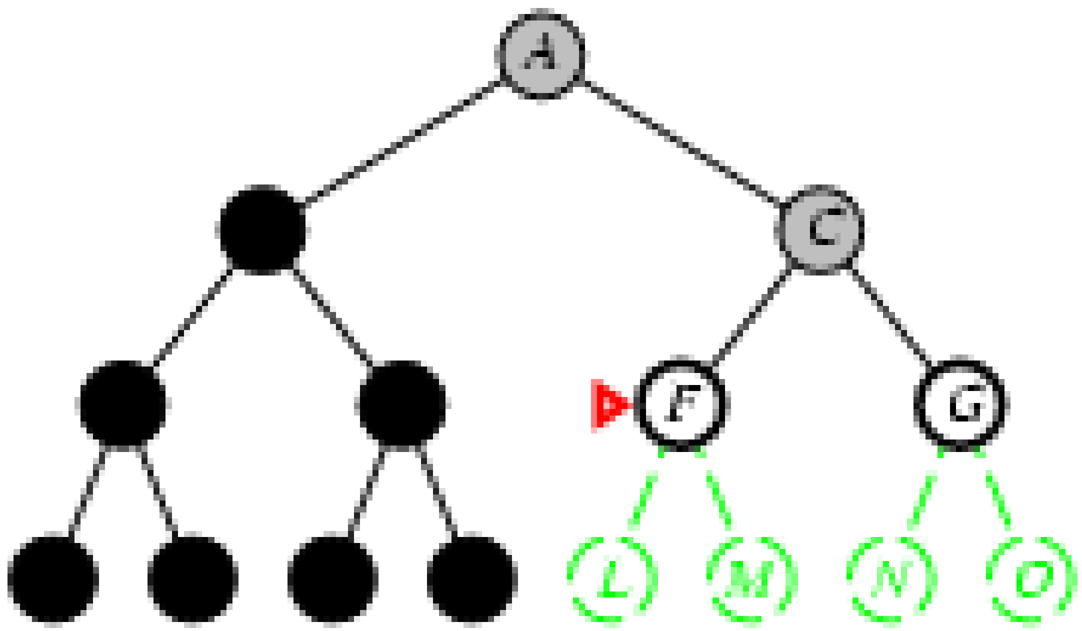
Depth-first search example

- Expand deepest unexpanded node



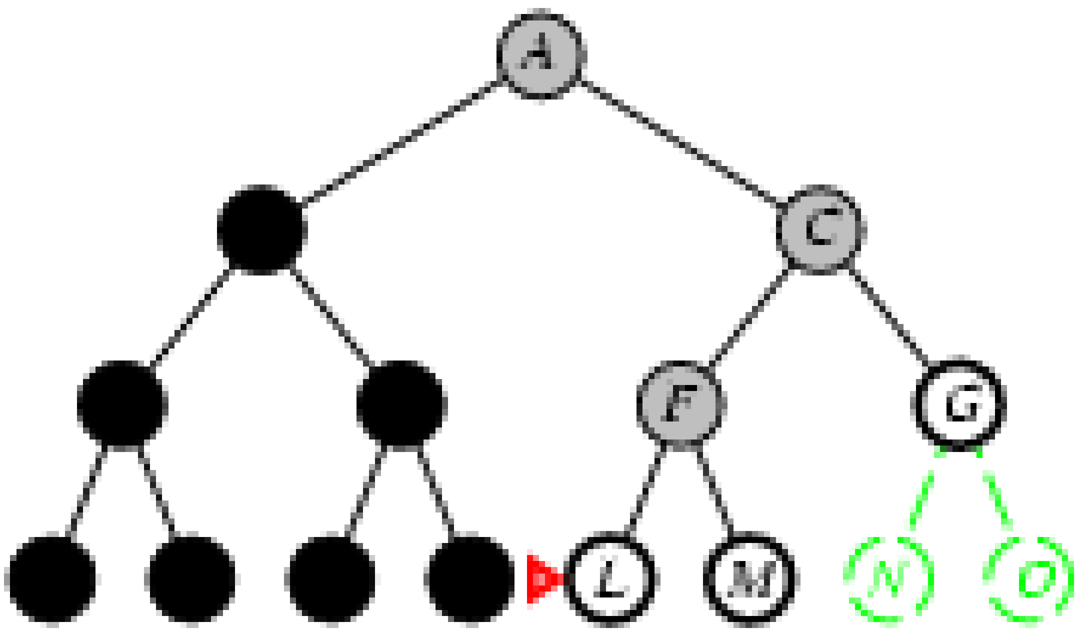
Depth-first search example

- Expand deepest unexpanded node



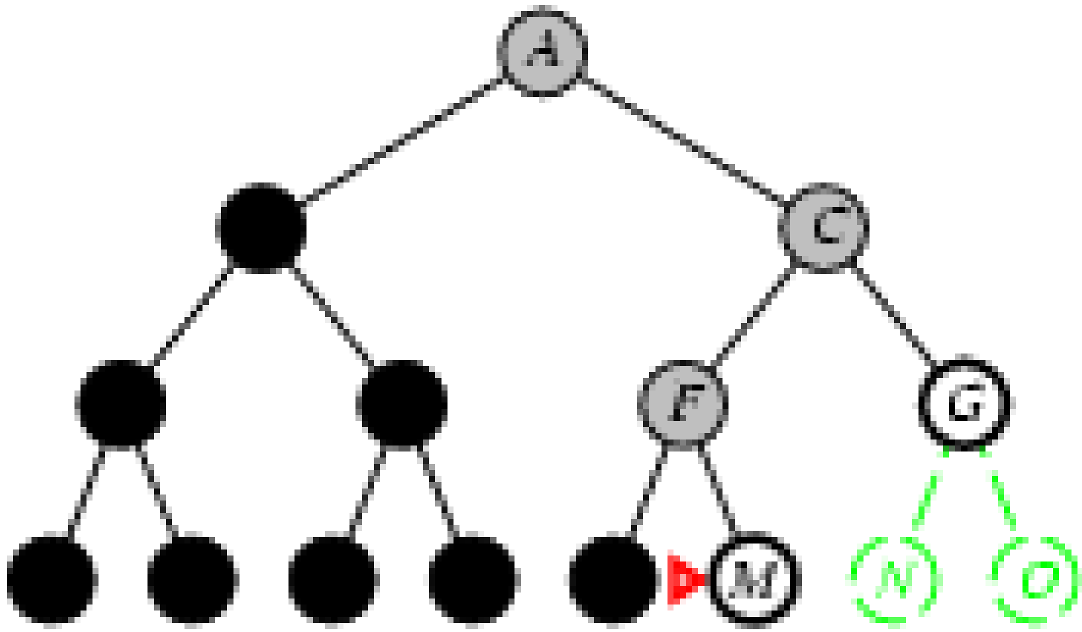
Depth-first search example

- Expand deepest unexpanded node



Depth-first search example

- Expand deepest unexpanded node



- But this type of search can go on and on, deeper and deeper into tree search space and thus, we can get lost. This is referred to as blind alley.

DFS Evaluation

- **Completeness**: Yes, if search space is finite and search space does not loops.
- **Time Complexity:**
  - Let m is the maximum depth of the search tree. At worst the solution may lie at same depth m.
  - Similarly root has b successors, each node at the next level has b successors then total b^2
  - Worst case: expand all nodes except the last node at depth m.
  - Total nodes generated = $b + b^2 + b^3 + \ldots\ldots + b^m = O(b^m)$
- **Space Complexity:**
  - It only stores a simgle path from root node to the leaf node along with remaining unexpanded sibling nodes.
  - Total nodes in memory = $(1 + b + b + \ldots\ldots + b)\ m = O\ (bm)$
- Optimal: Since DFS expands deepest node, if goal is at right sub tree, left subtree might be expanded first. Hence DFS may not be optimal

Depth-limited search(Same as DFS if L= ∞**)**

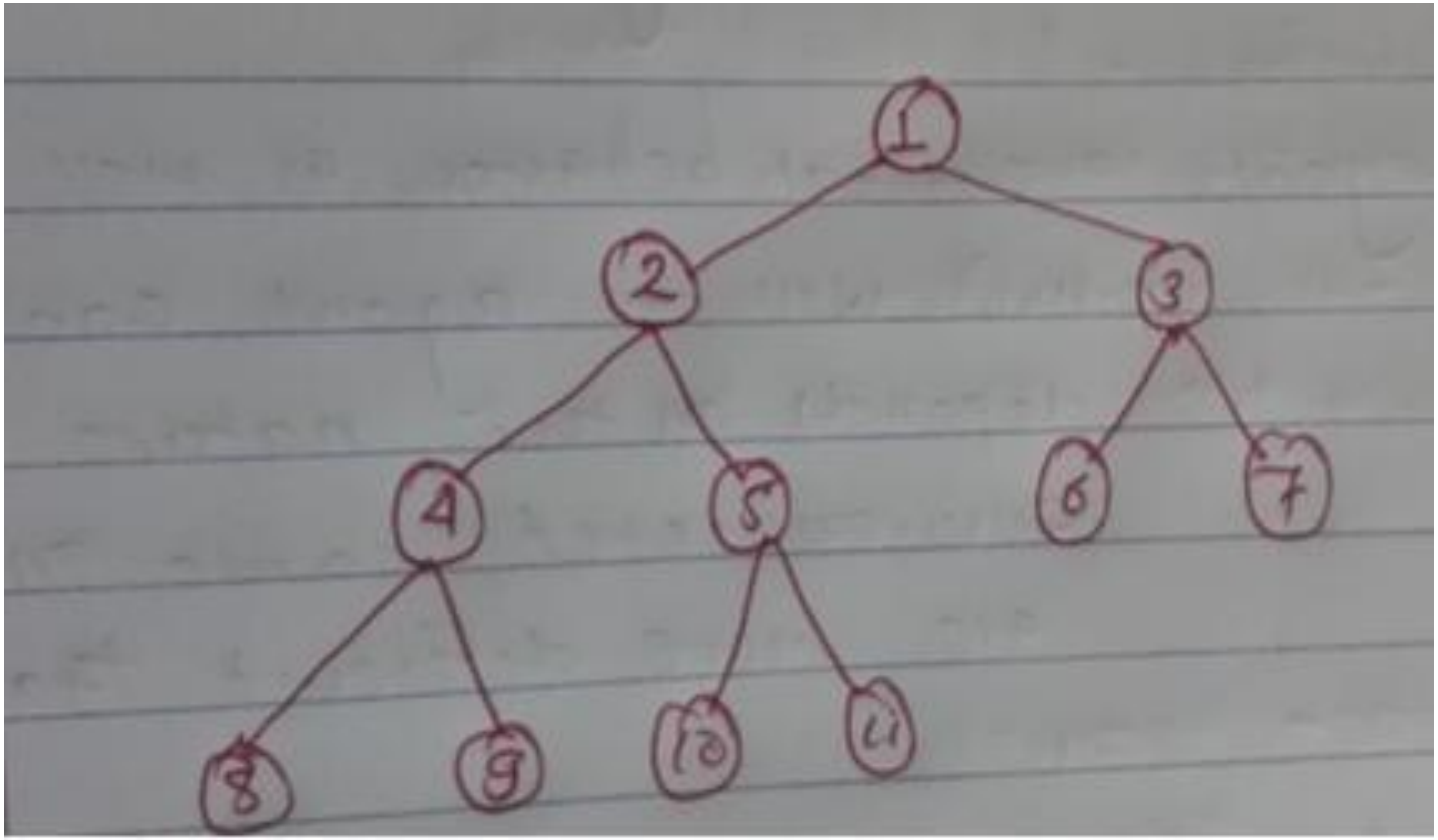
- Depth limit search is depth-first search with depth limit L.
  - i.e., nodes at depth L are treated as they have no successors.

- The depth limit solves the infinite path problem of DFS by placing limit on the depth.

- Yet it introduces another source of problem if we are unable to find good guess of L. Let d is the depth of shallowest solution.
  - If L < d then incompleteness results because no solution within the depth limit.
  - If L > d then not optimal.

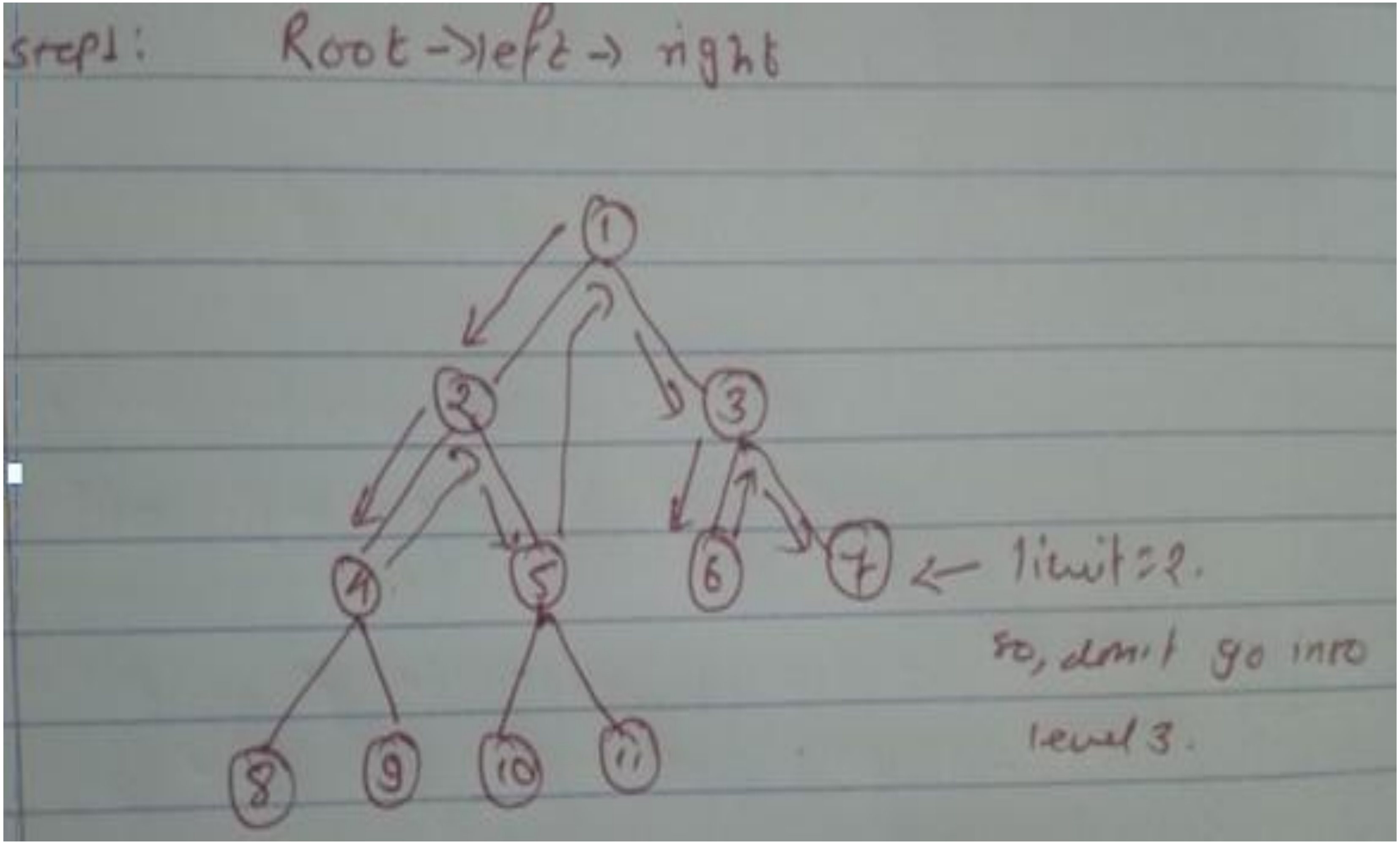
Depth-limited search Example 1

- Depth limited search= DFS+ limit for the depth
  - Let goal node= 11 and limit= 2
  - Trace the path to the goal node using Depth limited search .



Depth-limited search Example 1

- Depth limited search= DFS+ limit for the depth



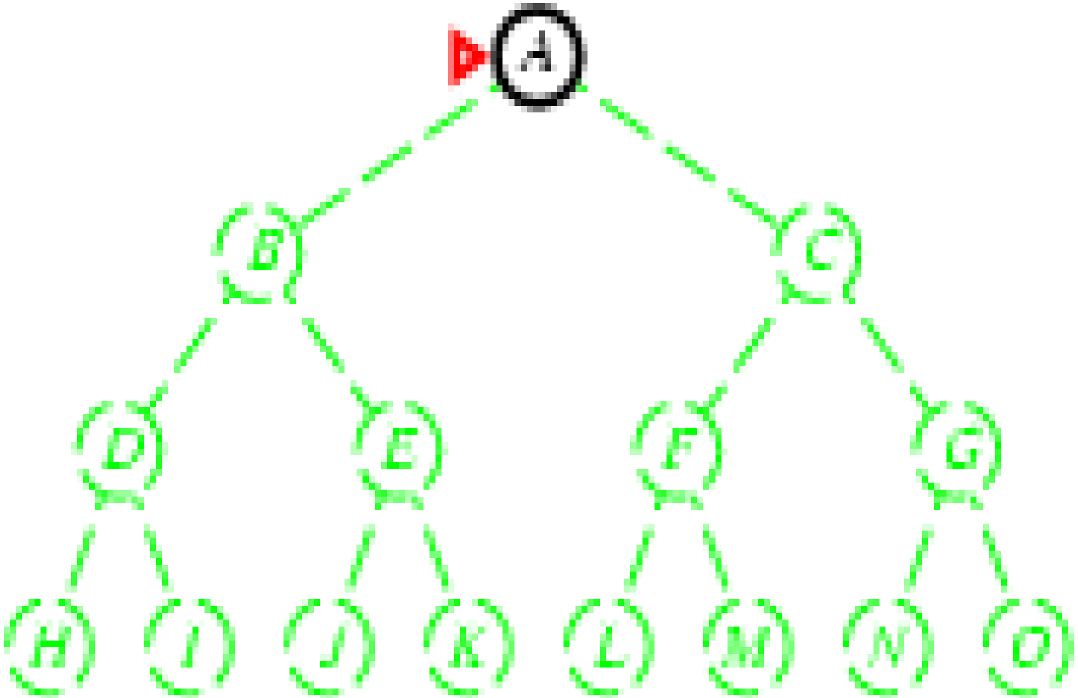
- No path is found from root node to goal node because L< d.

Iterative deepening search

- It is a general strategy to find best depth limit L.
- It begin by performing DFS to a depth of zero, then depth of one, depth of two , and so on until a solution is found or some maximum depth is reached.
- It is similar to BFS in that it explores a complete layer of new nodes at each iteration before going to next layer.
- It is similar to DFS for a single iteration.
- It is preferred when there is a large search space and the depth of a solution is not known.
- But it performs the wasted computation before reaching the goal depth.

Iterative Deepening search example

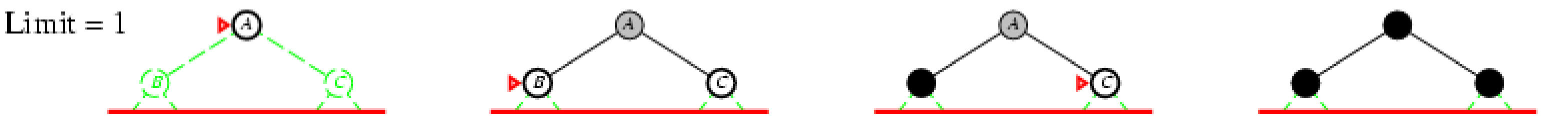
- Here initial state is A and goal state is M.



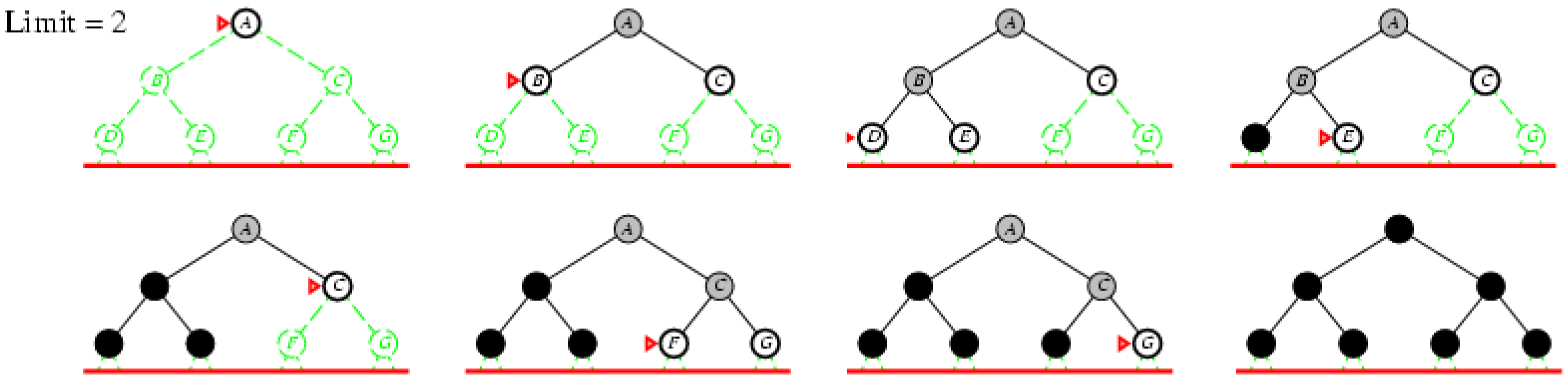
- Let we don‟t know the depth of M then the Iterative deepening search proceeds as follows:

Iterative deepening search l =0

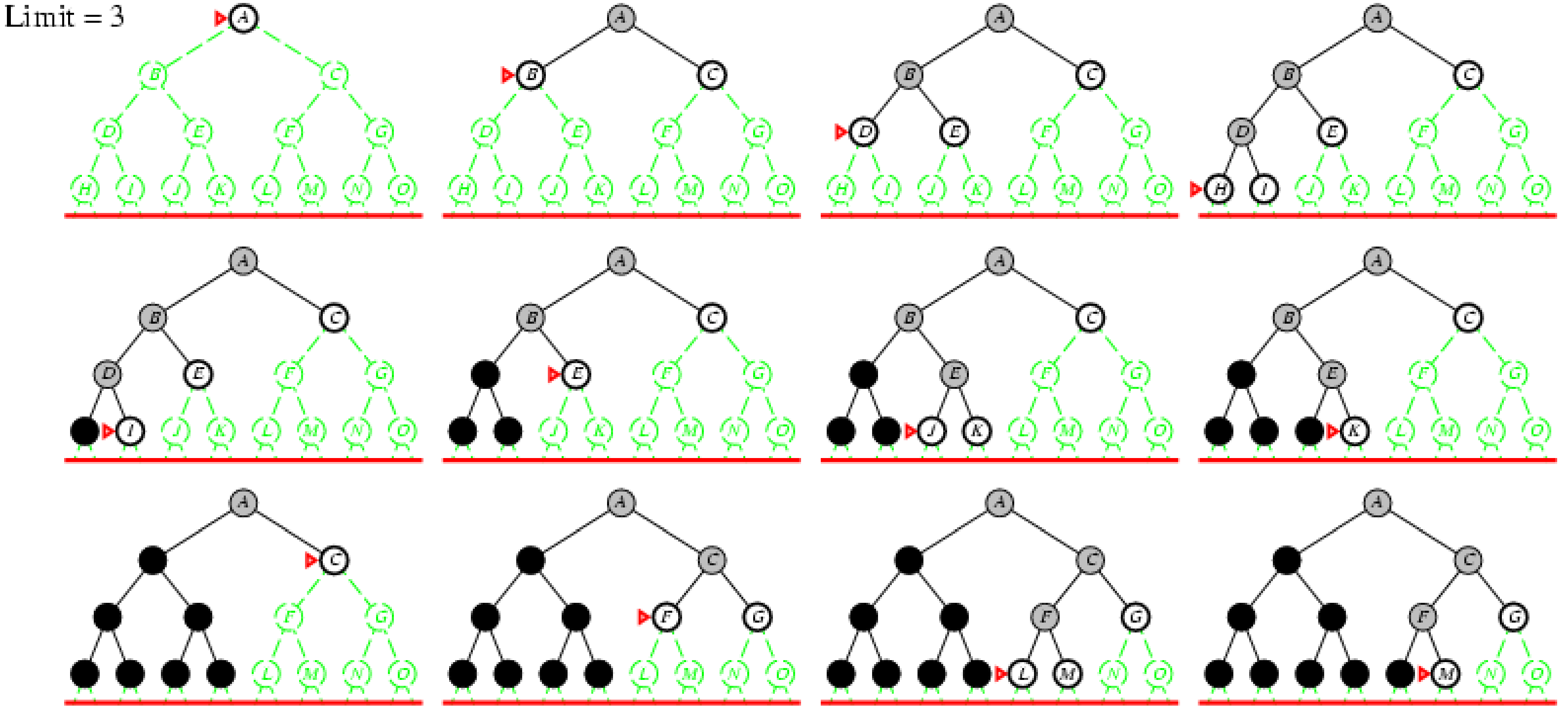
Iterative deepening search l =1



Iterative deepening search *l* =2



Iterative deepening search l =3

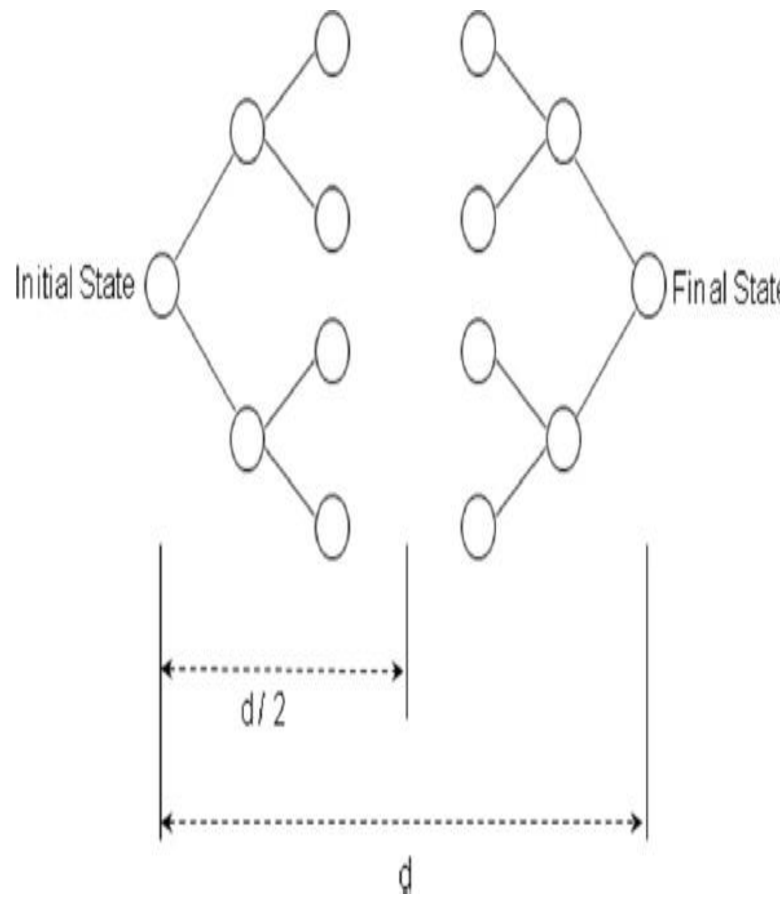


Bidirectional search

- This search is used when a problem has a single goal state that is given explicitly and all the node generation operators have inverses.
- So it is used to find shortest path from an initial node to goal node instead of goal itself along with path.
- It works by searching forward from the initial node and backward from the goal node simultaneously, by hoping that two searches meet in the middle.
- Check at each stage if the nodes of one have been generated by the other,. i.e., they meet in the middle.
- If so, the path concatenation is the solution.

Bidirectional search contd..

- **Advantages:**
  - Only slight modification of DFS and BFS can be done to perform this search.
  - Theoretically effective than unidirectional search.
- **Disadvantage:**
  - Problem if there are many goal states.
  - Practically inefficient due to additional overhead to perform intersection operation at each point of search



Drawbacks of uninformed search :

- Criterion to choose next node to expand depends only on a global criterion: level.
- Does not exploit the structure of the problem.

Heuristic Search:

- Heuristic Search Uses domain-dependent (heuristic) information beyond the definition of the problem itself in order to search the space more efficiently.
- Ways of using heuristic information:
  - Deciding which node to expand next, instead of doing the expansion in a strictly breadth-first or depth-first order;
  - In the course of expanding a node, deciding which successor or successors to generate, instead of blindly generating all possible successors at one time;
  - Deciding that certain nodes should be discarded, or pruned, from the search space.

Contd…

- Informed Search defines a heuristic function, h(n), that estimates the "goodness" of a node n.

- The heuristic function is an estimate, based on domain-specific information that is computable from the current state description, of how close we are to a goal.

- Specifically, h(n) = estimated cost (or distance) of minimal cost path from state "n" to a goal state.

Contd…

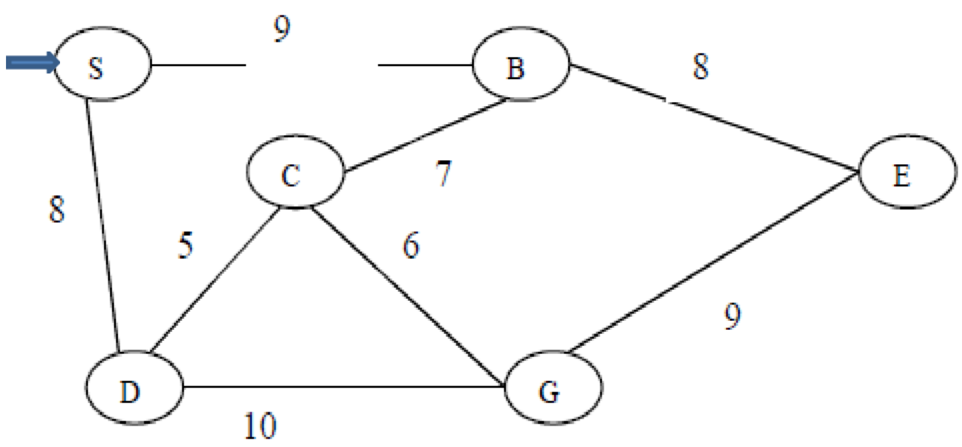
- A) Best-First Search:
  - **Best first search** uses an evaluation function f(n) that gives an indication of which node to expand next for each node.
  - A key component of f(n) is a heuristic function, h(n),which is a additional knowledge of the problem.
  - Based on the evaluation function best first search can be categorized into the following categories:
    - Greedy best-first search
    - A*search

Contd…

- Greedy Best First Search :
  - Greedy best first search expands the node that seems to be closest to the goal node.
  - Evaluation function based on Heuristic function is used to estimate which node is closest to the goal node.
  - Therefore, Evaluation function f(n) = heuristic function h(n)= estimated cost of the path from node n to the goal node.
  - E.g., $h_{SLD}(n)$ = *straight-line distance from n to goal*
  - Note: h(goal) = 0

Contd…

- Example1 To illustrate Greedy Best- First Search:
  - For example consider the following graph



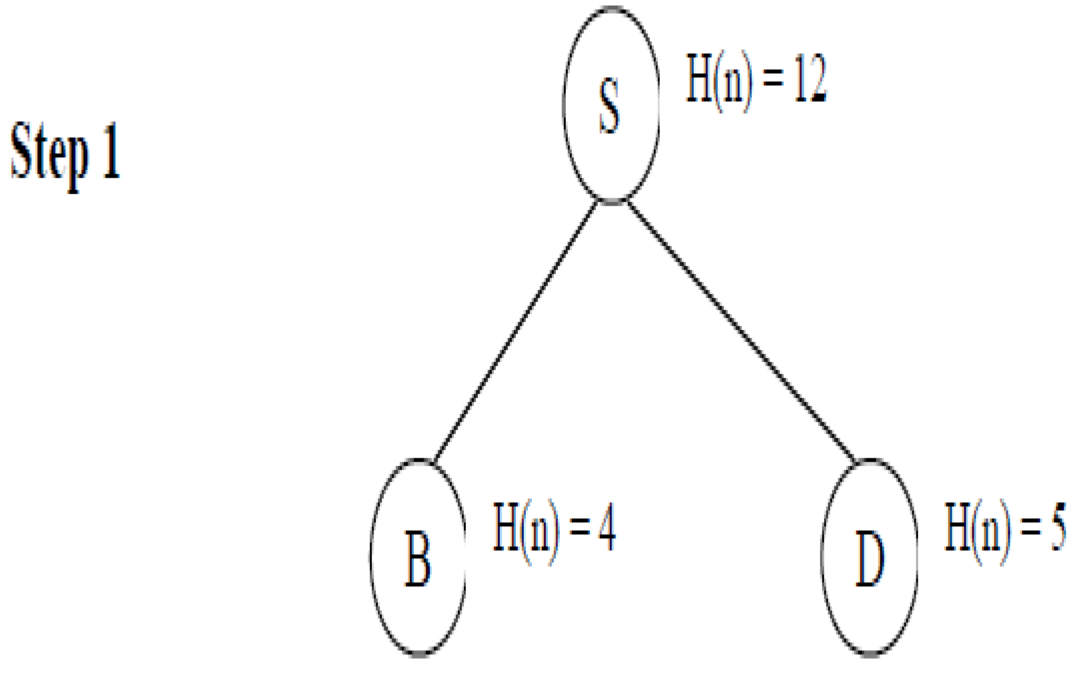
  - Straight Line distances to node G (goal node) from other nodes is given below:

$S \rightarrow G = 12$
$B \rightarrow G = 4$
$E \rightarrow G = 7$
$D \rightarrow G = 5$
$C \rightarrow G = 3$

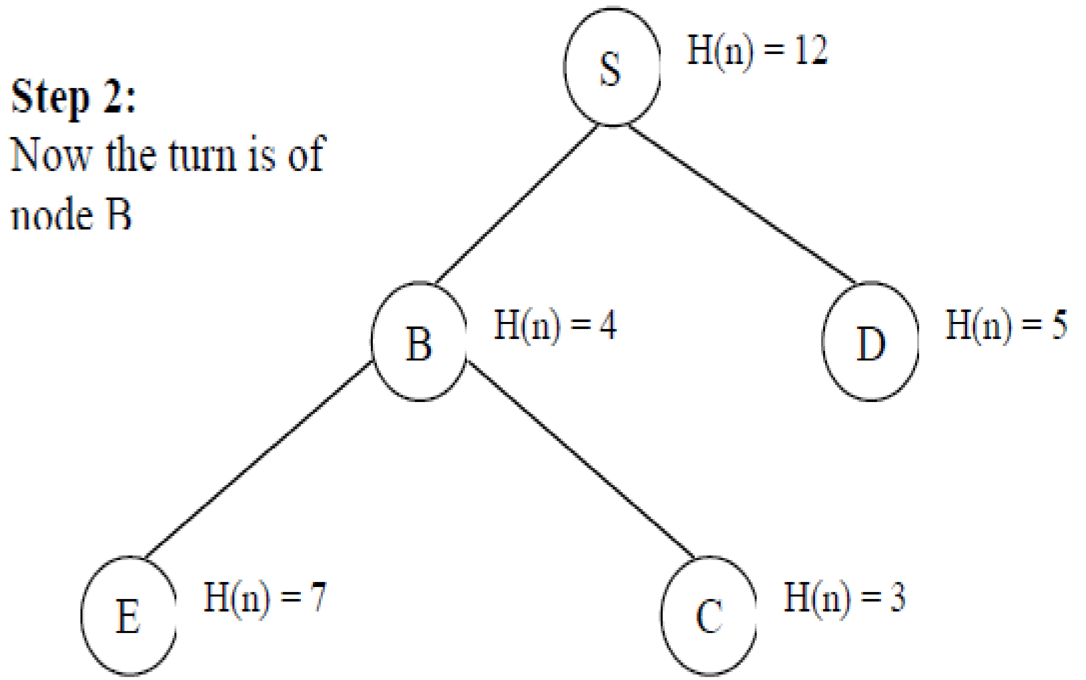
  - Let H(n)= Straight Line distance
  - Now Greedy Search operation is done as below:

Contd…

- Start at node "s", the start state
- Children of s= {B(4), D(5)}
- Therefore, best = B

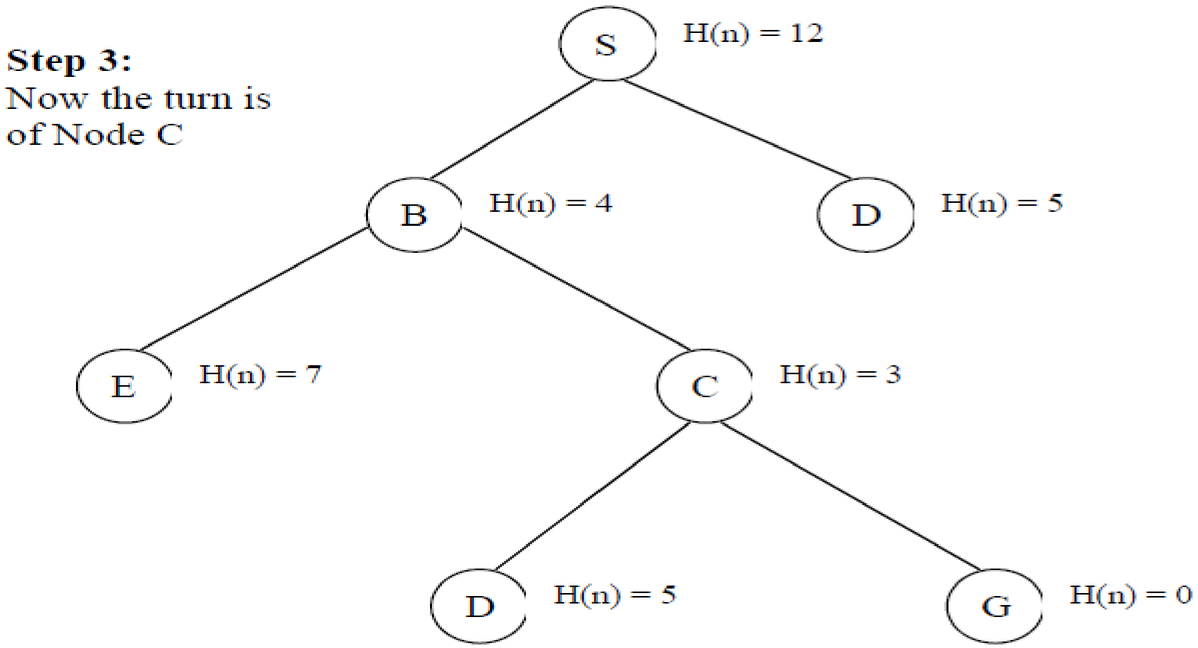


Contd…



- Children of B= {E(7), C(3)}
- Considered= {D(5), E(7), C(3)}
- Therefore, Best= C

Contd…



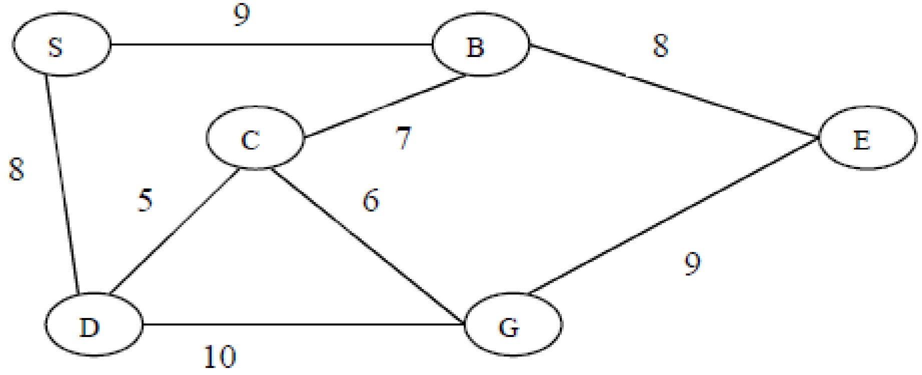
- Children of C= {D(5), G(0)}
- Considered= {D(5), E(7), G(0)}
- Therefore, Best= G, is the goal node.

Contd…

- A * Search :
  - A* is a best first , informed search algorithm. The search begins at root node. The search continues by visiting the next node which has the least evaluation.
  - It evaluates nodes by using the following evaluation function
    - f(n) = h(n) + g(n) = estimated cost of the cheapest solution through n.
    - Where, g(n): the actual shortest distance traveled from initial node to current node , It helps to avoid expanding paths that are already expensive
    - h(n): the estimated (or "heuristic") distance from current node to goal, it estimate which node is closest to the goal node.
  - Nodes are visited in this manner until a goal is reached.

Contd…

- Example to illustrate A* Search:
  - For example consider the following graph



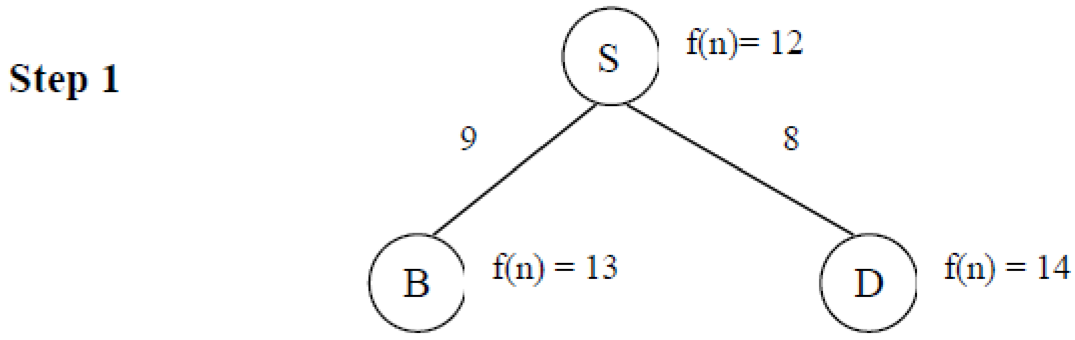
  - Straight Line distances to node G (goal node) from other nodes is given below:

$S \rightarrow G = 12$
$B \rightarrow G = 4$
$E \rightarrow G = 7$
$D \rightarrow G = 6$
$C \rightarrow G = 3$

  - Labels in the graph shows actual distance.
  - Let H(n)= Straight Line distance
  - Now A* Search operation is done as below

Contd…

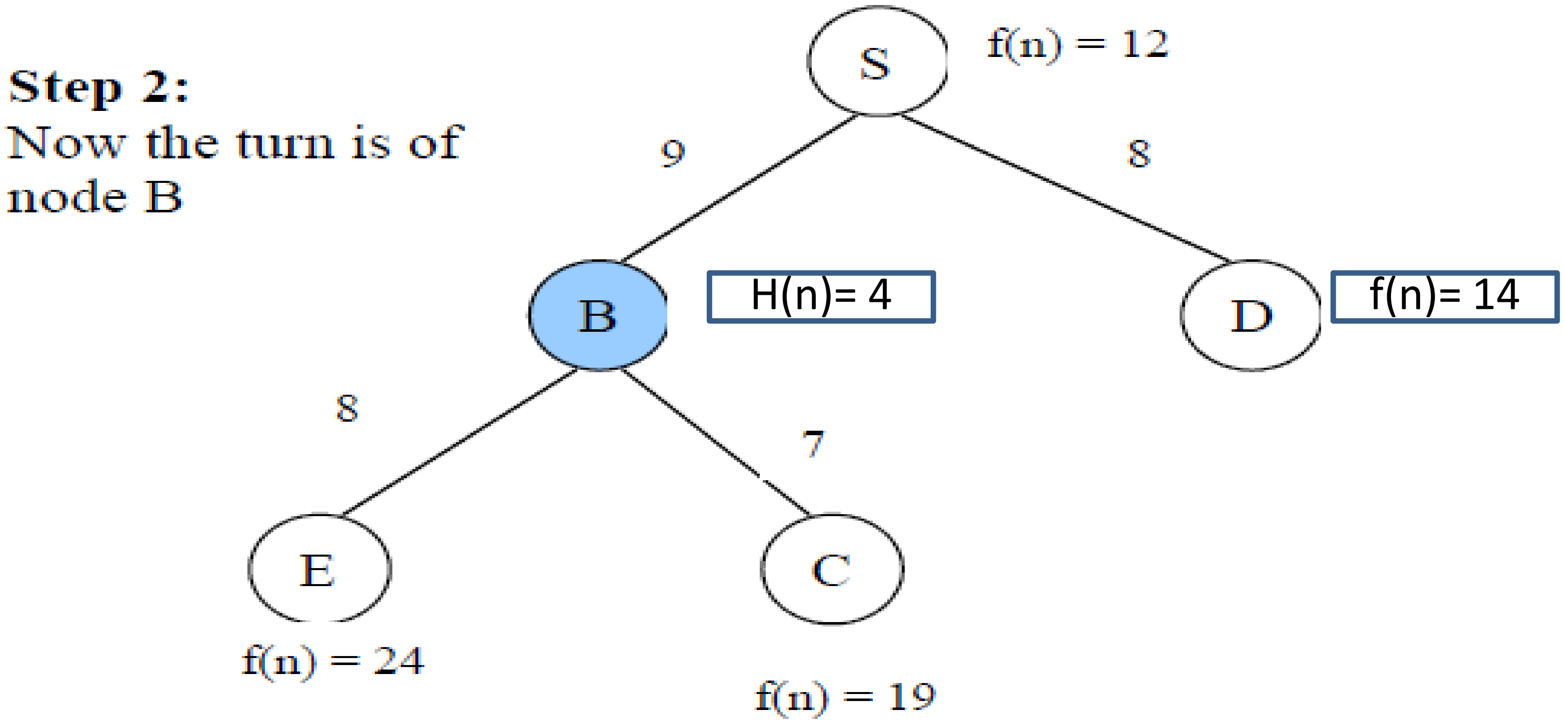
- A* algorithm starts at s, the start state.
- Children of S={B, D}



- Now, evaluation function for each child of S is:
  - f(B) =g(B) + h(B) =9+4=13
  - f(D)=g(D)+h(D)= 8+6=14
- Here, candidate node for expansion are{B, D}, among these candidate nodes the node B is least evaluated, so it is selected for expansion.

Contd…

- Child of B ={E, C}

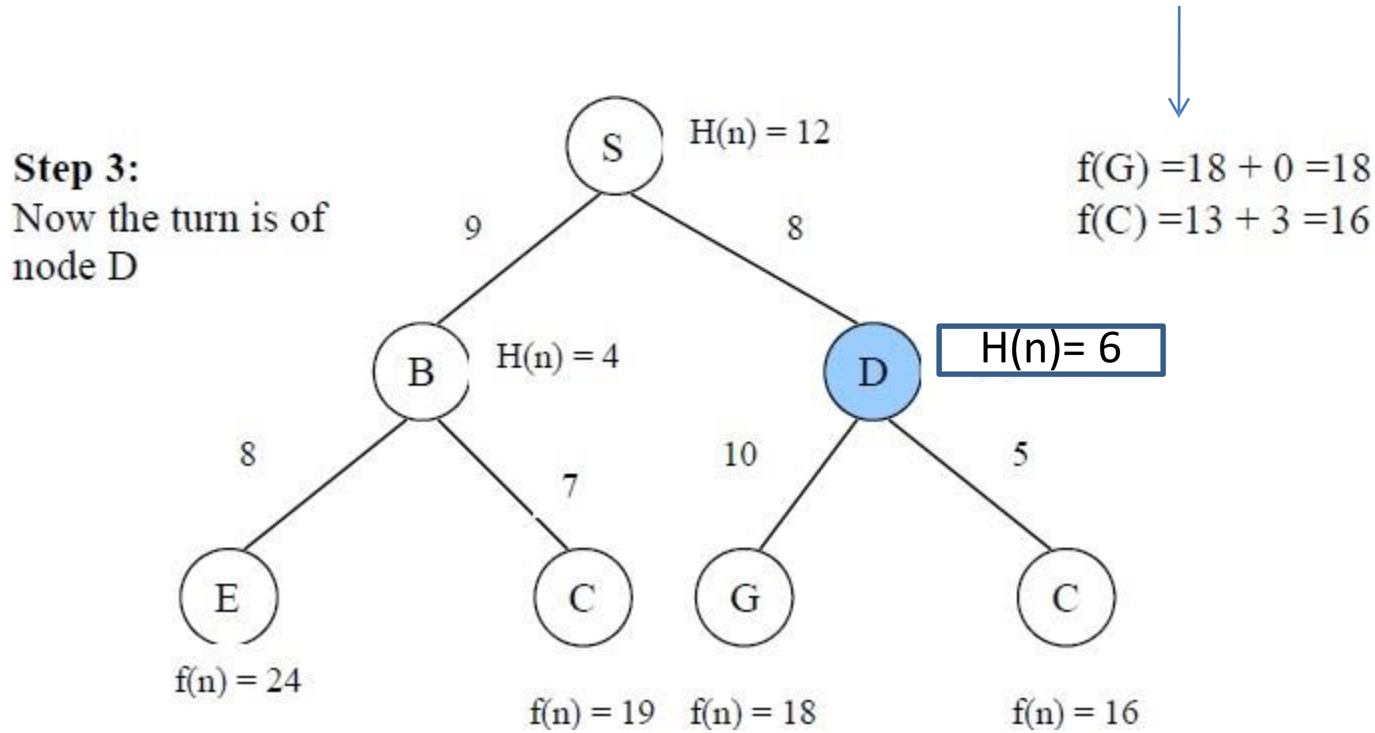


- Now, evaluation for each child of B are
  - F(E)= (9+8)+7=24
  - F(c)=(9+7)+3=19
- Here, candidate nodes for expansion are{E,C, and D}
- Among these candidate the node D is least evaluated, so it is selected for expansion

Contd….

- Child of D={G, C}

Evaluation for the child of D are

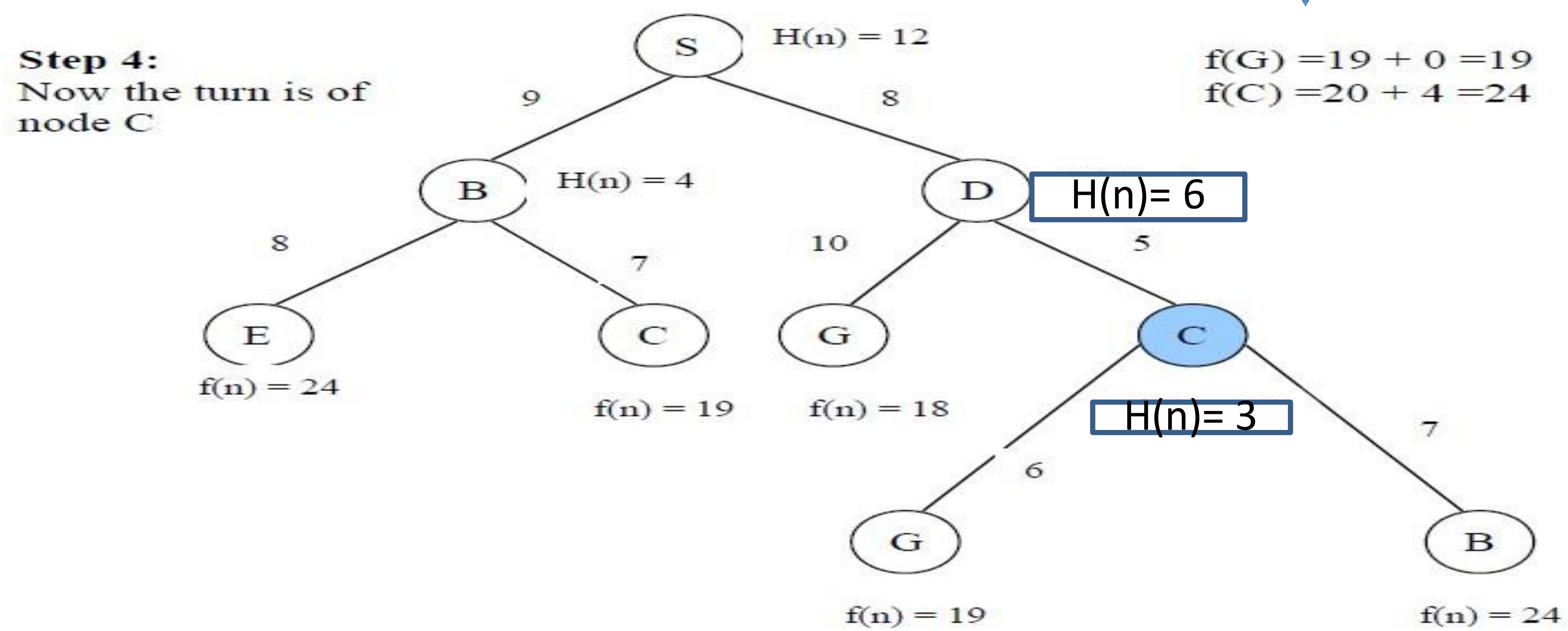


- Here, the candidate nodes for expansion are{ E, C,G and C}.
- Among these candidate the node C child of D is least evaluated, so It is selected for expansion.

Contd…

- Child of C= {G,B}

Evaluation for the child of C are

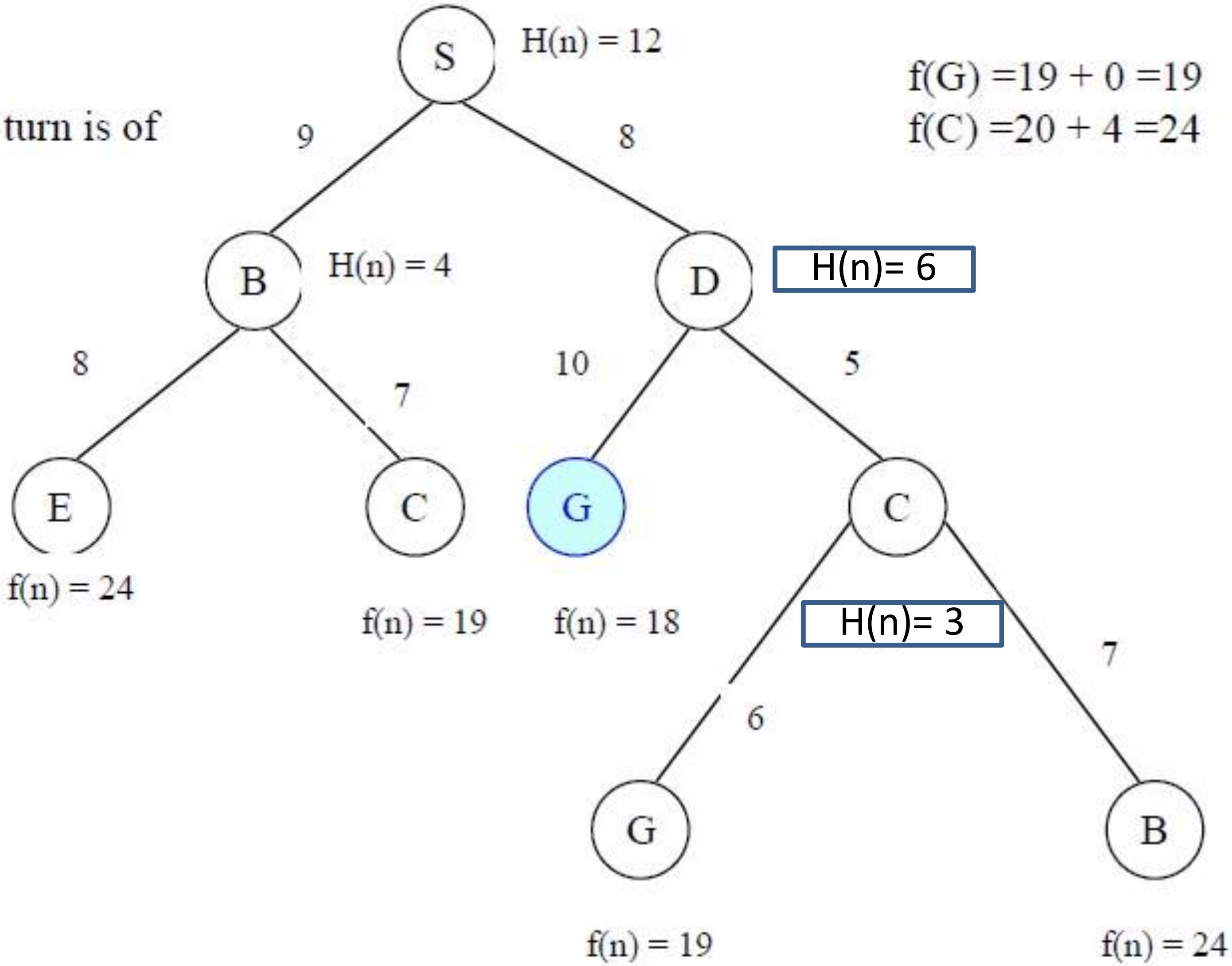


- Here the candidate for expansion are{E,C,G,G, and B}.
- Among these candidate the node G, which is child of D is least evaluated, so it is selected for expansion, but it is the goal node, hence we are done

Contd…

Step 4:
Now the turn is of node G

$f(G) = 19 + 0 = 19$
$f(C) = 20 + 4 = 24$



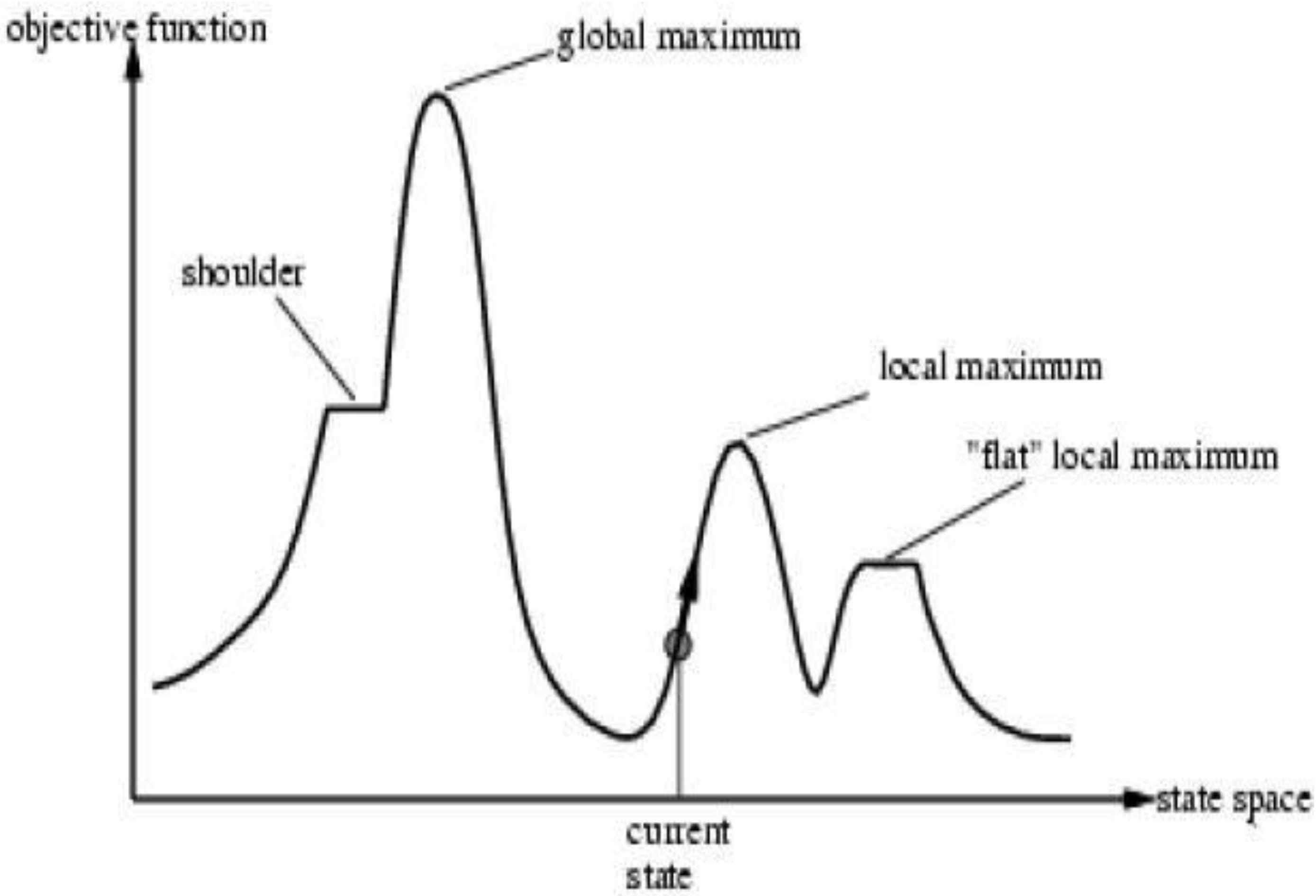
Contd…

- B) Local search algorithms:
  - Local search algorithms operate using a single current node and move only to neighbors of this node rather than systematically exploring paths from an initial state. E.g. Hill climbing.
  - These algorithms are suitable for problems in which all that matter is the solution state, not the path cost to reach it. Typically, the paths followed by the search are not retained.
  - Although local search algorithms are not systematic, they have two key advantages:
    - They use very little memory.
    - They can often find reasonable solutions in large or infinite state spaces for which systematic algorithms are unsuitable.

Contd…

- Hill Climbing Search:
  - Hill climbing can be used to solve problems that have many solutions, some of which are better than others.
  - It starts with a random (potentially poor) solution, and iteratively makes small changes to the solution, each time improving it a little. When the algorithm cannot see any improvement anymore, it terminates.
  - Ideally, at that point the current solution is close to optimal, but it is not guaranteed that hill climbing will ever come close to the optimal solution.
  - Note: The algorithm does not maintain a search tree and does not look ahead beyond the immediate neighbors of the current state.

Contd…

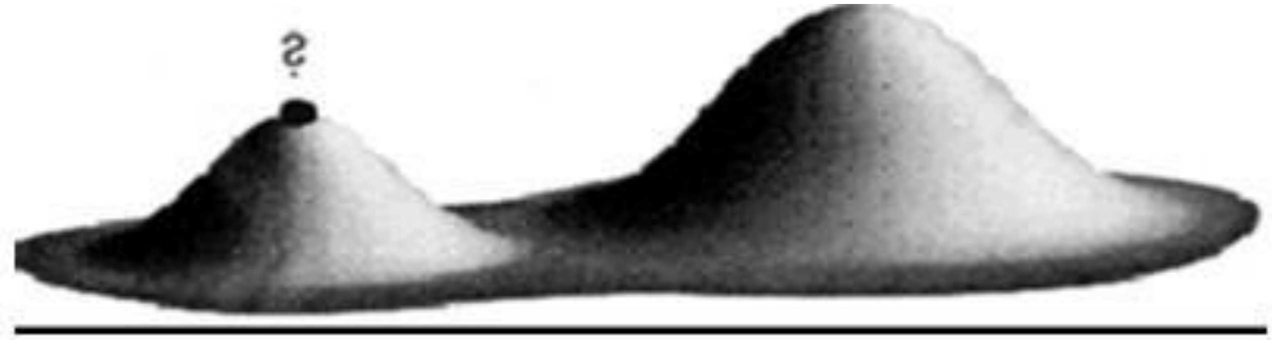

objective function
global maximum
shoulder
local maximum
"flat" local maximum
state space
current
state

Contd…

- Hill climbing suffers from the following problems:
  - The Foot- hills problem(local maximum)
  - The Plateau problem
  - The Ridge problem

Contd…

- Foot- Hill problem:
  - Local maximum is a state which is better than all of its neighbors but is not better than some other states which are farther away.
  - At local maxima, all moves appear to make the things worse.
  - This problem is called the foot hill problem.
  - **Solution:** Backtrack to some earlier node and try going to different direction.

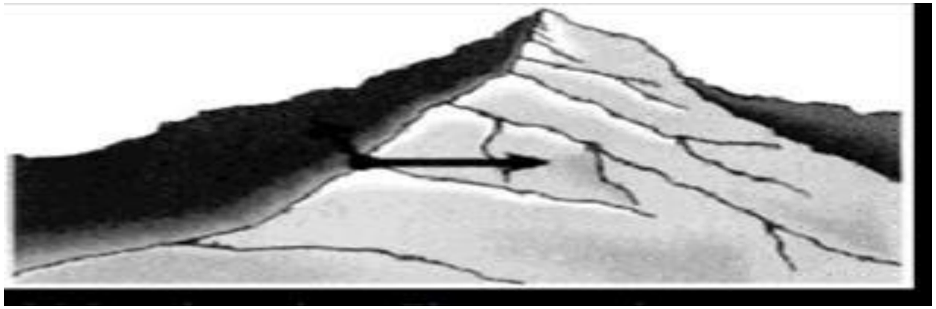


Contd…

- The Plateau problem:
  - Plateau is a flat area of the search space in which a whole set of neighboring states have the same value.
  - On plateau, it is not possible to determine the best direction in which to move by making local comparison.
  - Such a problem is called plateau problem.
  - **Solution:** Make a big jump in some direction to try to get a new section of the search space .

Contd…

- The Ridge problem:
  - Ridge is an area of the search space which is higher than the surrounding areas and that itself has a slope.
  - Due to the steep slopes the search direction is not towards the top but towards the side(oscillates from side to side).
  - Such a problem is called Ridge problem.
- **Solution**: Apply two or more rules such as bi-direction search before doing the test.



Constraint satisfaction problems

- A CSP consists of three components: X, D and C
  - X is a Finite set of variables $\{X_1, X_2, \ldots, X_n\}$
  - D is a set of Nonempty domain of possible values for each variable $\{D_1, D_2, \ldots D_n\}$
  - C is a Finite set of constraints $\{C_1, C_2, \ldots, C_m\}$
  - Each constraint C_i limits the values that variables can take, e.g., $X_1 \neq X_2$

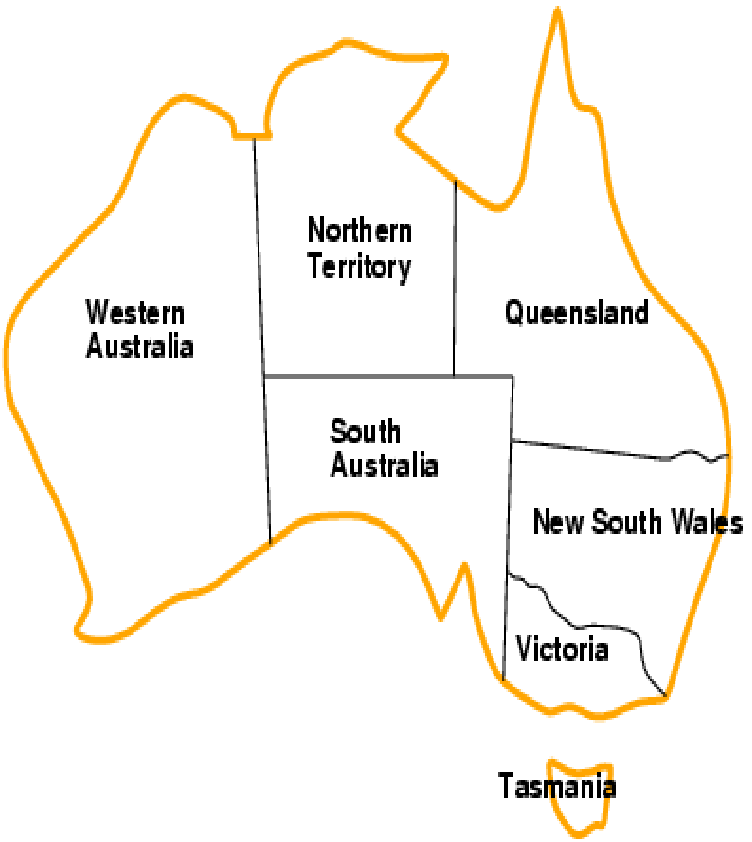
Contd…

- In CSP:
  - State: A state is defined as an assignment of values to some or all variables.
  - Consistent assignment: consistent assignment is an assignment that does not violate the constraints.
  - An assignment is complete when every variable is assigned a value.
  - A solution to a CSP is a complete assignment that satisfies all constraints i.e., complete and consistent assignment.

Contd…

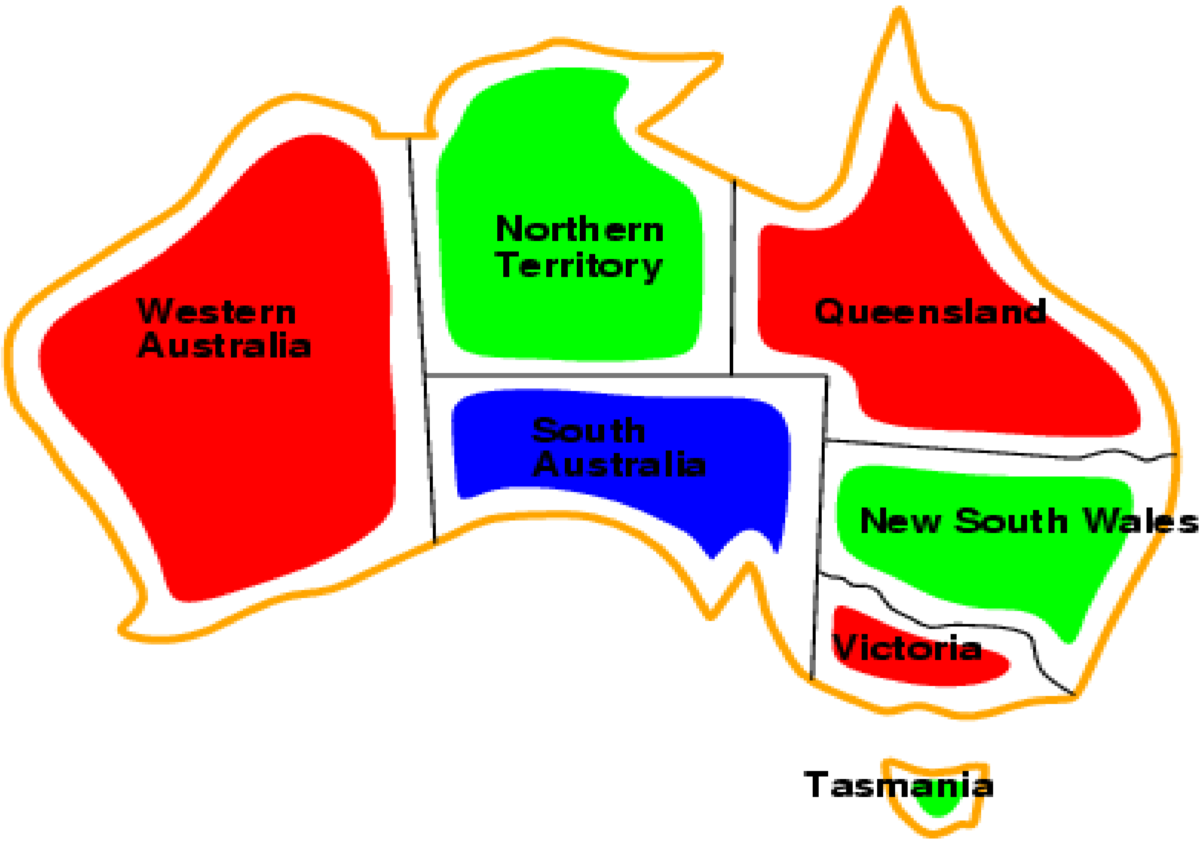
- **Example (Map-Coloring problem):** Given, a map of Australia showing each of states and territories. The task is color each region either red, green, or blue in such a way that no neighboring regions have the same color.

- This problem can be formulated as CSP as follows:
  - Variables: *WA, NT, Q, NSW, V, SA, T*
  - Domains: D_i = { red, green, blue}
  - Constraints: adjacent regions must have different colors
    - e.g., $WA \neq NT$



Contd…

- Solutions are complete and consistent assignments,
  - e.g., WA = red, NT = green, Q = red, NSW = green, V = red, SA = blue, T = green



Contd…

- CSP as a standard search problem:
  - A CSP can easily expressed as a standard search problem, as follows:
    - Initial State: the empty assignment {}, in which all variables are unassigned..
    - Successor function: Assign value to unassigned variable provided that there is not conflict.
    - Goal test: the current assignment is complete and consistent.
    - Path cost: a constant cost for every step.

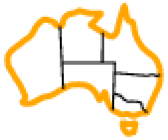
Contd…

- Constraint satisfaction search:
  - For searching a solution of CSPs following algorithms can be used:
    - Backtracking search: Works on partial assignments.
    - Local search for CSPs: works on complete assignment.

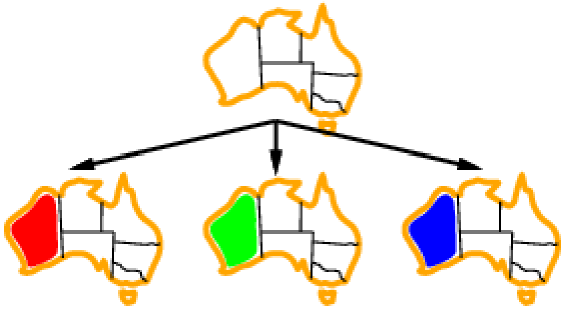
Contd…

- Backtracking search:
  - The term backtracking search is used for a depth first search that chooses values for one variable at a time and backtracks when a variable has no legal values left to assign.
  - The algorithm repeatedly choose an unassigned variable, and then tries all values in the domain of that variable in turn, trying to find a solution.
  - If an inconsistency is detected, then backtrack returns failure, causing the previous call to try to another values.

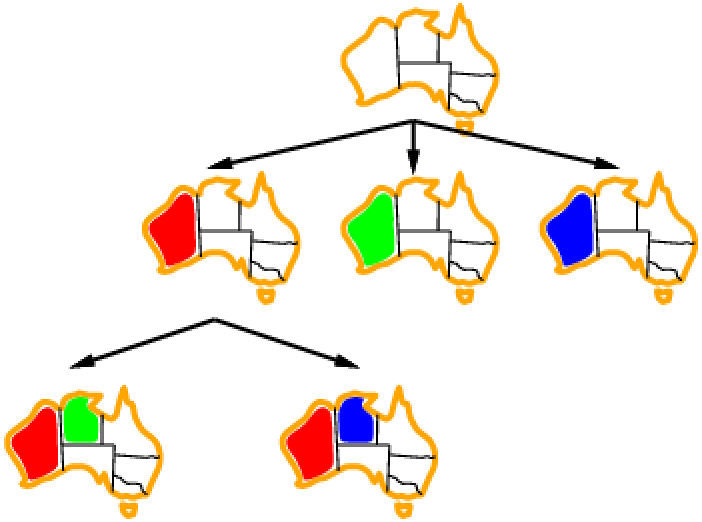
Backtracking example



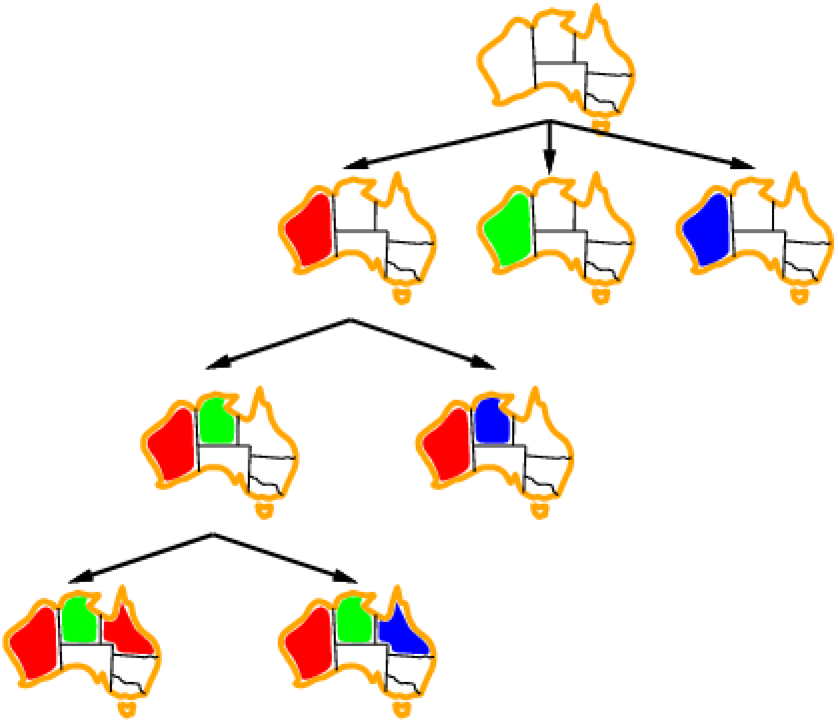
Backtracking example



Backtracking example



Backtracking example



Contd…

- Local search for CSPs:
  - They use a complete state formulation:
    - The initial state assigns a value to every variable, and the search changes the value of one variable at a time because the initial guess violates several constraints.
  - E.g.:
    - First randomly select any conflicted variable.
    - Then choose the value of that conflicted variable that violates the fewest constraints.

Crypt-arithmetic Problem

- Many problems in AI can be considered as problems of constraint satisfaction, in which the goal state satisfies a given set of constraint.
- Example of such a problem is Crypt-Arithmetic problem (a mathematical puzzle), in which the goal state(solution) satisfies the following constraints:
  - Values are to be assigned to letters from 0 to 9 only.
  - No two letters should have the same value.
  - If the same letter occurs more than once, it must be assigned the same digit each time.
  - The sum of the digits must be arithmetically correct with the added restriction that no leading zeroes are allowed.

Contd…

- **Example1**: Solve the following puzzle by assigning numeral (0-9) in such a way that each letter is assigned unique digit which satisfy the following addition.

$$
\begin{array}{cccc}
 & T & W & O \\
+ & T & W & O \\
\hline
F & O & U & R
\end{array}
$$

Contd…

- Solution: Here,
  - Variables: $\{F, T, U, W, R, O, c_1, c_2, c_3\}$
  - Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
  - Constraints: *Alldiff (F,T,U,W,R,O)*
- where c1, c2, and c3 are auxiliary variables representing the digit (0 or 1) carried over into the next column.

  - $O + O = R \longrightarrow c_1$
  - $c_1 + W + W = U \longrightarrow c_2$
  - $c_2 + T + T = O \longrightarrow c_3$
  - $c_3 = F,\ T \neq 0,\ F \neq 0$

$$
\begin{array}{cccc}
c3 & c2 & c1 & \\
 & T & W & O \\
+ & T & W & O \\
\hline
F & O & U & R
\end{array}
$$

Contd…

- Here we are adding two three letters words but getting a four letters word. This indicates that F= c3=1
- Now, c2+T+T= O+ 10……. Because c3=1
- C2 can be 0 or 1 . Let c2=0 then T should be > 5 i.e T can be {6, 7,8,9}
- Let T= 9 then C2+T+T= O+10 0+9+9=O+10 from this O= 8
- Now O+O=R+10 8+8= R+10 From this R=6
- Now, c1+W+W=U here c1=1 and U and W can be {2,3,4,5,7}
- But , c2= 0 so let W= 2 then 1+2+2=U i.e., U=5
- Now replacing each letter in the puzzle by its corresponding digit and testing their arithmetic correctness:

```
  T W O         928
+ T W O       + 928
-------       -----
F O U R        1856
```

- This assignment satisfies all the constraint so this is the final solution

Contd…

- **Example2**: Solve the following puzzle by assigning numeral (0-9) in such a way that each letter is assigned unique digit which satisfy the following addition.

```
 FOUR
+FOUR
-----
EIGHT
```

- Solution: Here,
  - Variables: $\{F,O,U,R,E,I,G,H,T, c_1, c_2, C_3, c_4\}$
  - Domains: $\{0,1,2,3,4,5,6,7,8,9\}$
  - Constraints: *Alldiff (F,O,U,R,E,I,G,H,T)*
- where c1, c2, and c3 are auxiliary variables representing the digit (0 or 1) carried over into the next column.

```
c4 c3 c2 c1
  FOUR
+ FOUR
------
 EIGHT
```

Contd…

- Here we are adding two three letters words but getting a four letters word. This indicates that E= c4=1 ….. Because E is left most letter so it should not be 0.
- Now c3+F+F= I+10 , here c3 can be 0 or 1 and F should be greater than 5. i.e {6,7,8,9}
- Let c3= 0 and F= 9 then 0+9+9=I+10 from this I=8
- Now, c2+O+O=G…… since c3=0
- C2 can be 0 or 1 and O can be {2,3,4}. Let c2= 0 and O =2 Then G= 4.
- R+R= T here R can be {3,5,6,7}
- If we let R= 3 this leads to dead end so let R=5 then T=0 and c1=1
- C1+U+U=H here c1= 1 and c2=0 and U can be{3}
- Form this U=3 then H=7

FOUR	9 2 3 5
+FOUR	+ 9 2 3 5
EIGHT	1 8 4 7 0

Contd…

Problem: Crypt-Arithmetic puzzle : Solve the following puzzle by assigning numeral (0-9) in such a way that each letter is assigned unique digit which satisfy the following addition.

$$\begin{array}{cccccc} & & S & E & N & D \\ + & & M & O & R & E \\ \hline & M & O & N & E & Y \\ \hline \end{array}$$

- **Initial problem state:**
  - S = ? M= ? C1 = ?
  - E = ? O = ? C2 = ?
  - N = ? R = ? C3 = ?
  - D = ? E = ? C4 = ?
- **Goal states:** A goal state is a problem state in which all letters have been assigned a digit in such a way that all constraints are satisfied

Contd…

Carries :

$$C_4 = ?\ ; C_3 = ?\ ; C_2 = ?\ ; C_1 = ?$$

C_4	C_3	C_2	C_1	$\longleftarrow$	Carry
	S	E	N	D	
+	M	O	R	E	
M	O	N	E	Y	

Constraint equations:

$$Y = D + E \longrightarrow C_1$$
$$E = N + R + C_1 \longrightarrow C_2$$
$$N = E + O + C_2 \longrightarrow C_3$$
$$O = S + M + C_3 \longrightarrow C_4$$
$$M = C_4$$

Contd…

- We can easily see that M has to be non zero digit, so the value of C_4 is 1.

1. $M = C_4 \Rightarrow \mathbf{M = 1}$
2. $O = S + M + C_3 \longrightarrow C_4$

   For $C_4 = 1$, $S + M + C_3 > 9 \Rightarrow S + 1 + C_3 > 9 \Rightarrow S + C_3 > 8$. If $C_3 = 0$, then $S = 9$ else if $C_3 = 1$, then $S = 8$ or 9.

- We see that for $S = 9$
  - $C_3 = 0$ or 1
  - It can be easily seen that $C_3 = 1$ is not possible as $O = S + M + C_3 \Rightarrow O = 11 \Rightarrow$ O has to be assigned digit 1 but 1 is already assigned to M, so not possible.
  - Therefore, for $C_3 = 0$, $O = 10$ and thus O is assigned 0 (zero).

Therefore, O = 0 $\boxed{\mathbf{M = 1, O = 0}}$

Contd…

3. $N = E + O + C_2 \longrightarrow C_3 = 0$

- Since $O = 0$, $N = E + C_2$. Since $N \neq E$, therefore, $C_2 = 1$.

Hence $N = E + 1$

- Now E can take value from 2 to 8 {0,1,9 already assigned so far }
  - If $E = 2$, then $N = 3$.
    - Since $C_2 = 1$, from $E = N + R + C_1$, we get $12 = N + R + C_1$
    - If $C_1 = 0$ then $R = 9$, which is not possible as we are on the path with $S = 9$
    - If $C_1 = 1$ then $R = 8$, then
      - From $Y = D + E$, we get $10 + Y = D + 2$.
      - For no value of D, we can get Y.
  - Try similarly for $E = 3, 4$. We fail in each case.

Contd…

- If $E = 5$, then $N = 6$
    - Since $C_2 = 1$, from $E = N + R + C_1$, we get $15 = N + R + C_1$,
    - If $C_1 = 0$ then $R = 9$, which is not possible as we are on the path with $S = 9$.
    - If $C_1 = 1$ then $R = 8$, then
        - ❑ From $Y = D + E$, we get $10 + Y = D + 5$ i.e., $5 + Y = D$.
        - ❑ If $Y = 2$ then $D = 7$. These values are possible.

- Hence we get the final solution as given below and on backtracking, we may find more solutions.

S = 9 ; E = 5 ; N = 6 ; D = 7 ; M = 1 ; O = 0 ; R = 8 ;Y = 2

Home Work

- Solve the following puzzles by assigning numeral (0-9) in such a way that each letter is assigned unique digit which satisfy the following addition.

1.
```
   TOM
+  NAG
  GOAT
```

2.
```
   YOUR
  + YOU
  HEART
```

3.
```
   CROSS
+  ROADS
  DANGER
```

4.
```
   BASE
+  BALL
  GAMES
```

Adversarial Search(Game Playing)

- Competitive environments in which the agents goals are in conflict, give rise to adversarial search, often known as games.

- In AI, games means deterministic, fully observable environments in which there are two agents whose actions must alternate and in which utility values at the end of the game are always equal and opposite.

  – E.g., **If first player wins, the other player necessarily loses**

- This Opposition between the agent's utility functions make the situation adversarial.

Contd…

- Games as Adversarial Search:
  - A Game can be formally defined as a kind of search problem with the following elements:
    - States: board configurations.
    - Initial state: the board position and which player will move.
    - Successor function: returns list of (move, state) pairs, each indicating a legal move and the resulting state.
    - Terminal test: determines when the game is over.
    - Utility function: gives a numeric value in terminal states
      - (e.g., -1, 0, +1 for loss, tie, win)

Contd…

- Game Trees: Problem spaces for typical games represented as trees, in which:
  - Root node: Represents the state before any moves have been made.
  - Nodes: Represents possible states of the games. Each level of the tree has nodes that are all MAX or all MIN; nodes at level i are of the opposite kind from those at level i+1. and
  - Arcs: Represents the possible legal moves for a player. Moves are represented on alternate levels of the game tree so that all edges leading from root node to the first level represent moves for the first(MAX) player and edges from the first level to second represents moves for the second(MIN) player and so on.
  - Terminal nodes represent end-game configurations.

Contd…

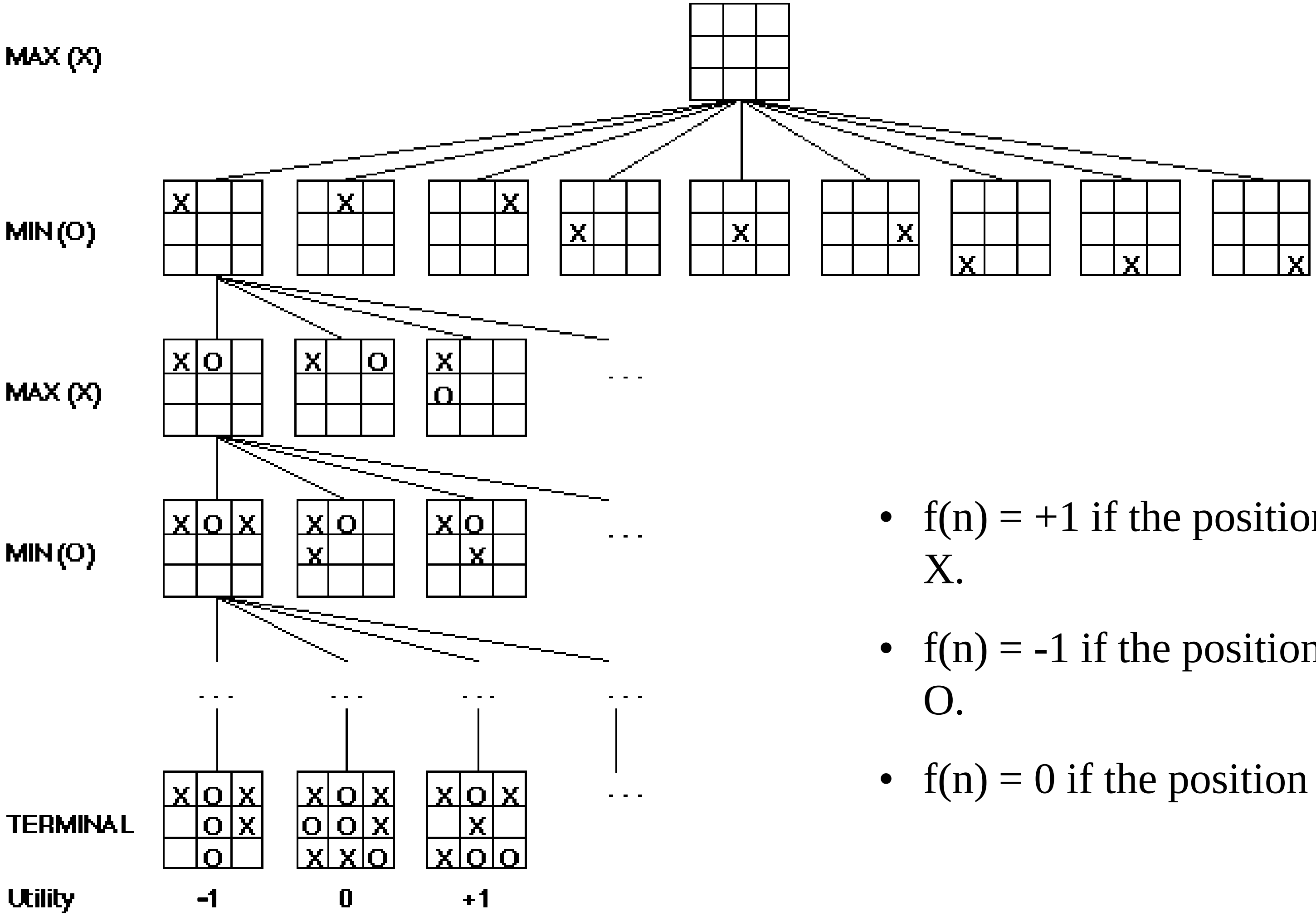
- Evaluation Function:
  - An evaluation function is used to evaluate the "goodness" of a game position.
  - i.e., estimate of the expected utility of the game position
  - The performance of a game playing program depends strongly on the quality of its evaluation function.
  - An inaccurate evaluation function will guide an agent toward positions that turn out to be lost.

Contd…

- A good evaluation function should:
  - Order the terminal states in the same way as the true utility function:
    - i.e., States that are wins must evaluate better than draws, which in turn must be better than losses. Otherwise, an agent using the evaluation function might make a mistake even if it can see ahead all the way to the end of the game.
  - For non-terminal states, the evaluation function should be strongly correlated with the actual chances of winning.
  - The computation must not take too long i.e., evaluate faster.

Contd…

- An example (partial) game tree for Tic-Tac-Toe:



- f(n) = +1 if the position is a win for X.
- f(n) = -1 if the position is a win for O.
- f(n) = 0 if the position is a draw.

Contd…

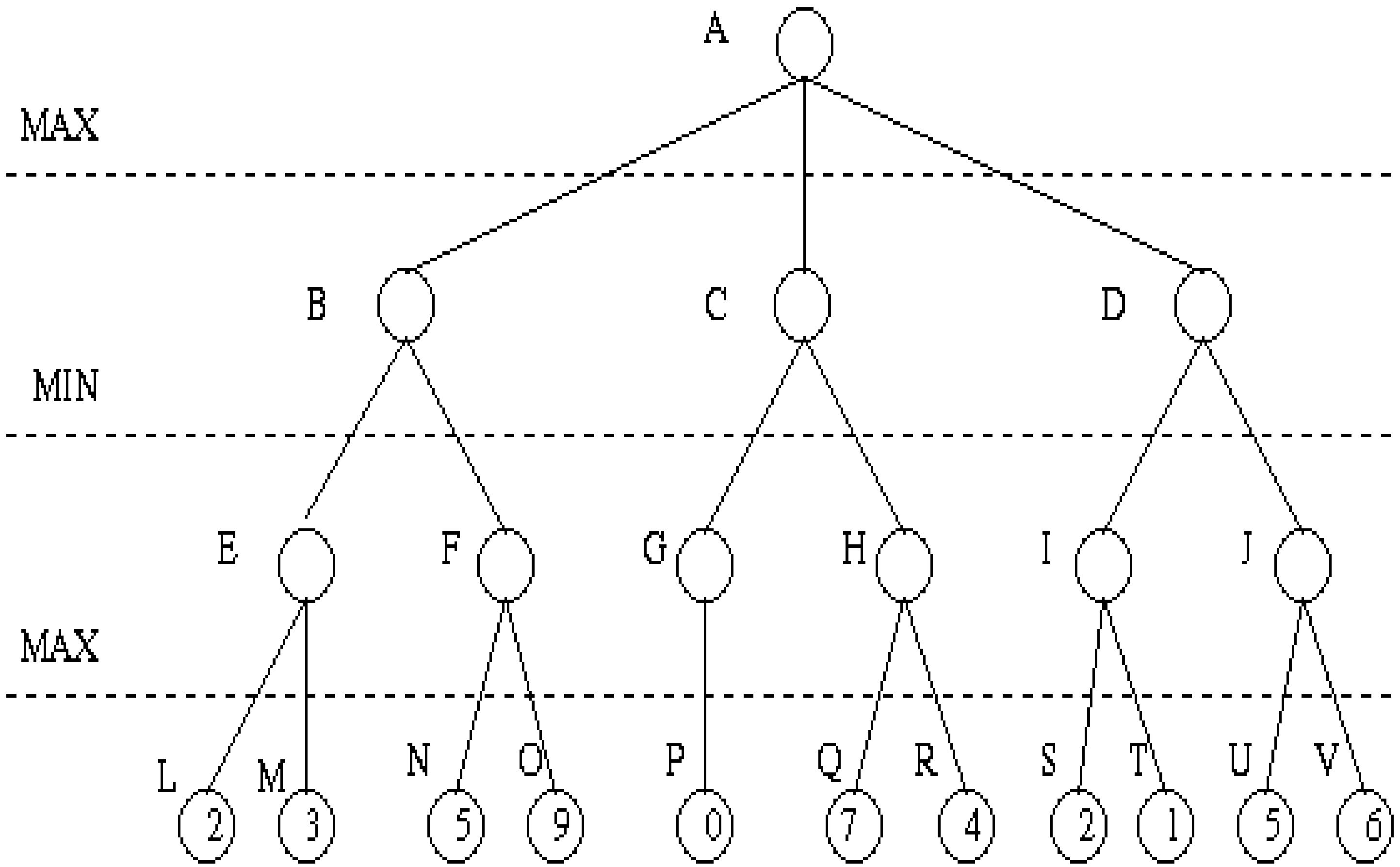
- There are two players denoted by X and O. They are alternatively writing their letter in one of the 9 cells of a 3 by 3 board. The winner is the one who succeeds in writing three letters in line.
- The game begins with an empty board. It ends in a win for one player and a loss for the other, or possibly in a draw.
- A complete tree is a representation of all the possible plays of the game. The root node is the initial state, in which it is the first player's turn to move (the player X).
- The successors of the initial state are the states the player can reach in one move, their successors are the states resulting from the other player's possible replies, and so on.
- Terminal states are those representing a win for X, loss for X, or a draw.
- Each path from the root node to a terminal node gives a different complete play of the game. Figure given above shows the search space of Tic-Tac-Toe.

MIN-MAX search algorithm

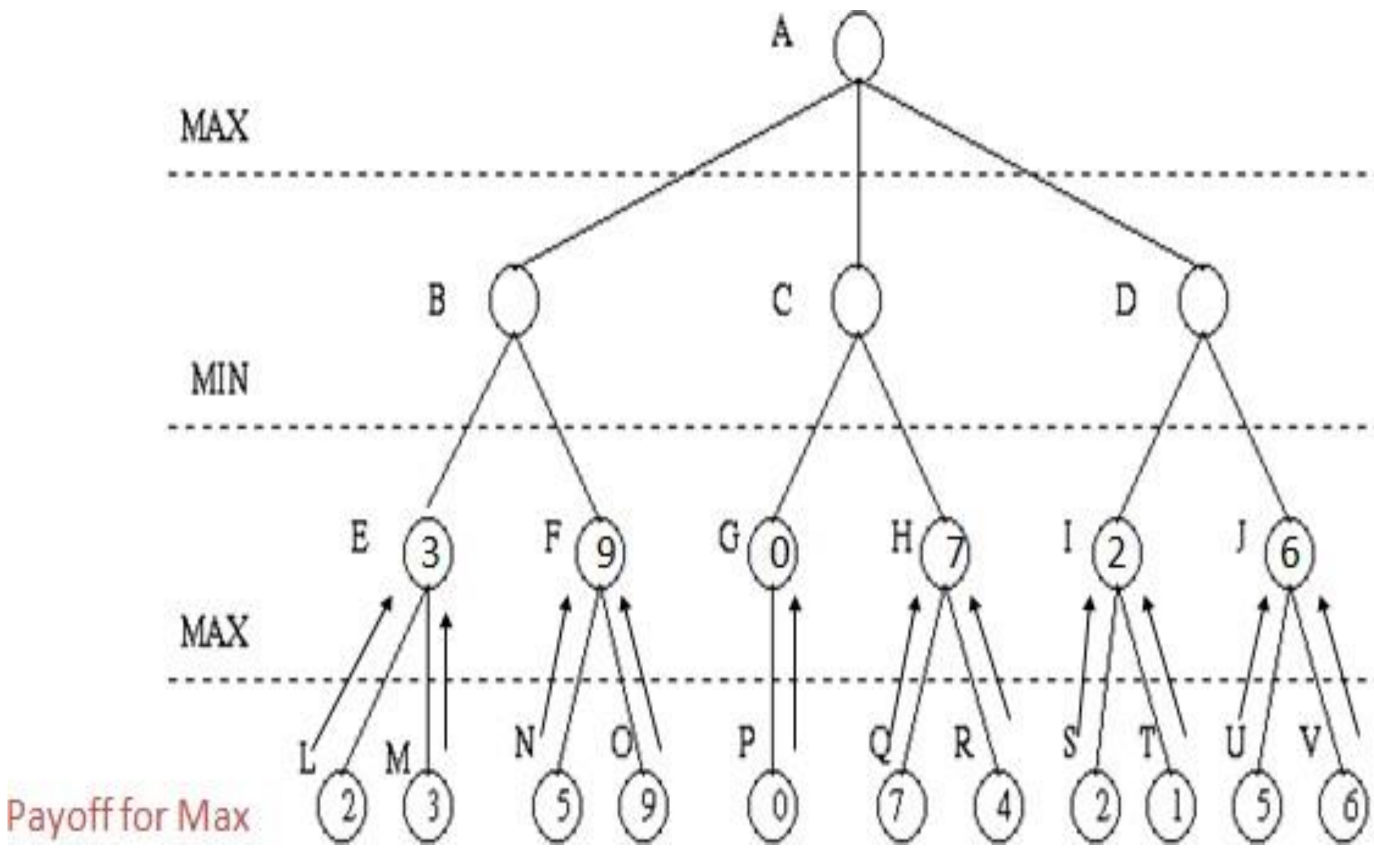
- Mini-max search algorithm is a game search algorithm with the application of DFS procedure.
- It assumes:
  - Both the player play optimally from there to the end of the game.
  - A suitable value of static evaluation(utility) is available on the leaf nodes.
- Given the value of the terminal nodes, the value of each node (MAX and MIN) is determined by (back up from) the values of its children until a value is computed for the root node.
  - For a MAX node, the backed up value is the **maximum** of the values associated with its children
  - For a MIN node, the backed up value is the **minimum** of the values associated with its children

Contd…

- Mini-max search Example:

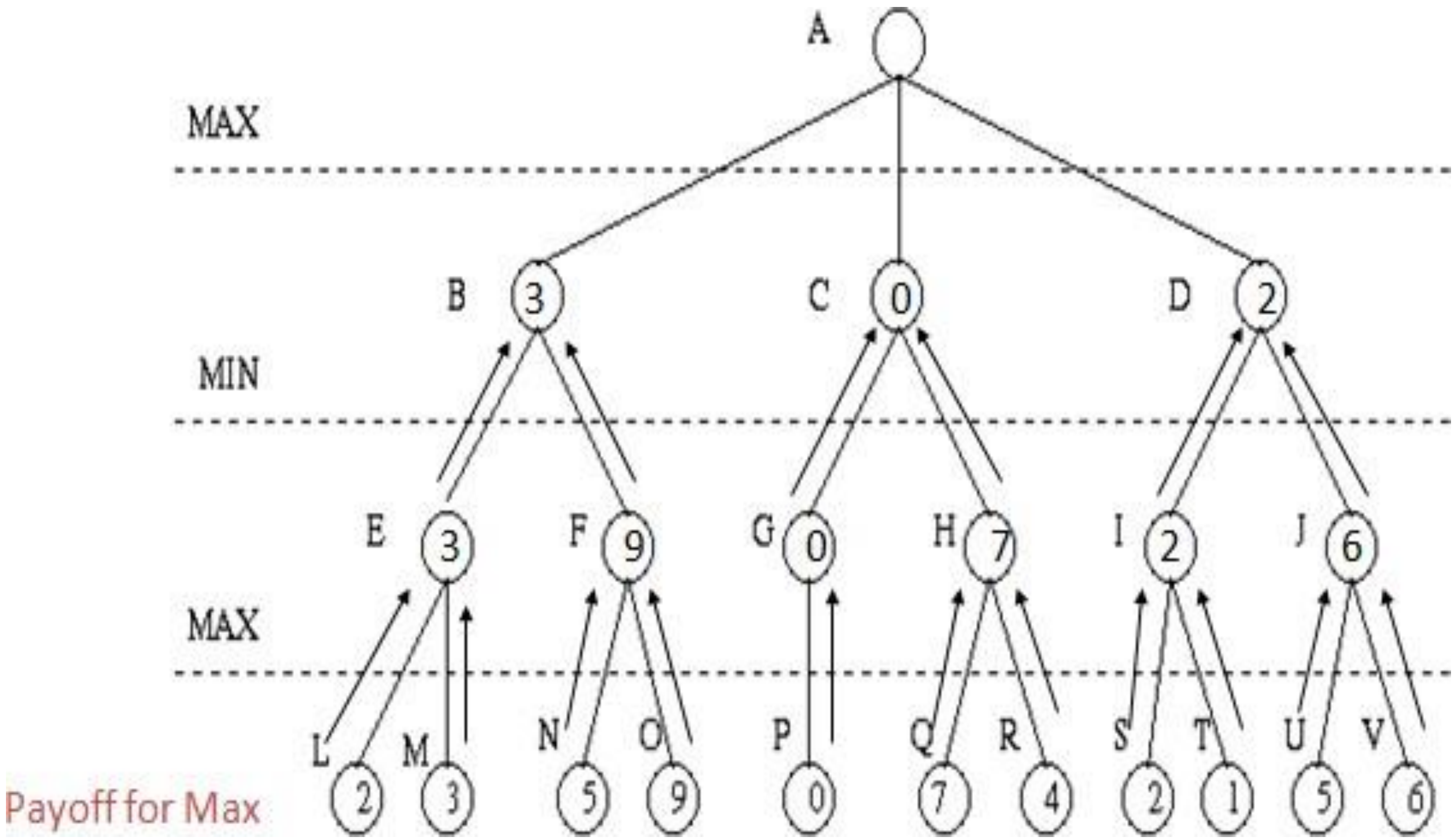


Contd…

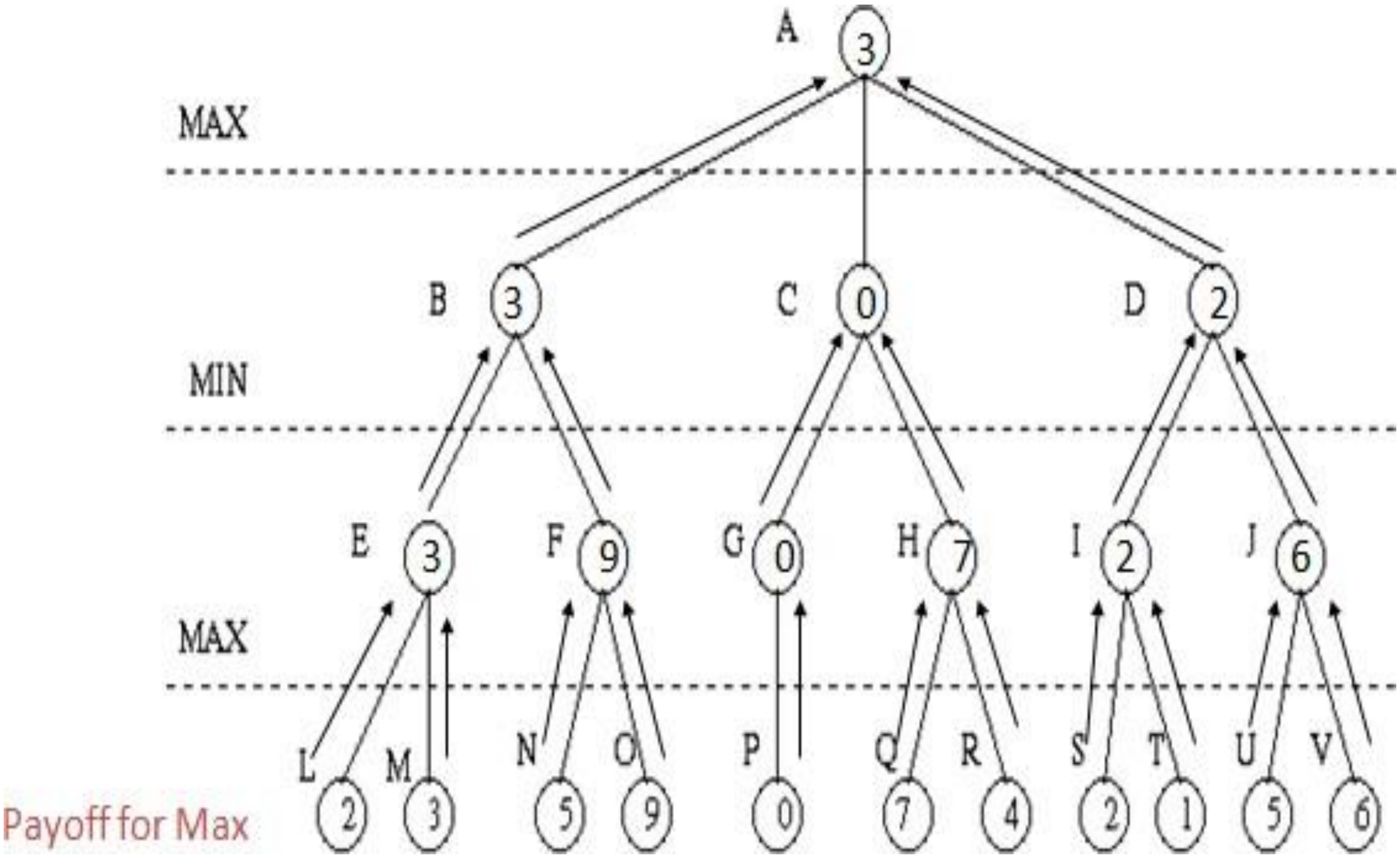


A
MAX
B
C
D
MIN
E
3
F
9
G
0
H
7
I
2
J
6
MAX
L
M
N
O
P
Q
R
S
T
U
V
Payoff for Max
2
3
5
9
0
7
4
2
1
5
6

Contd…


A
MAX
B
3
C
0
D
2
MIN
E
3
F
9
G
0
H
7
I
2
J
6
MAX
L
M
N
O
P
Q
R
S
T
U
V
Payoff for Max
2
3
5
9
0
7
4
2
1
5
6

Contd…



A
3
MAX
B
3
C
0
D
2
MIN
E
3
F
9
G
0
H
7
I
2
J
6
MAX
L
M
N
O
P
Q
R
S
T
U
V
Payoff for Max
2
3
5
9
0
7
4
2
1
5
6

Contd…

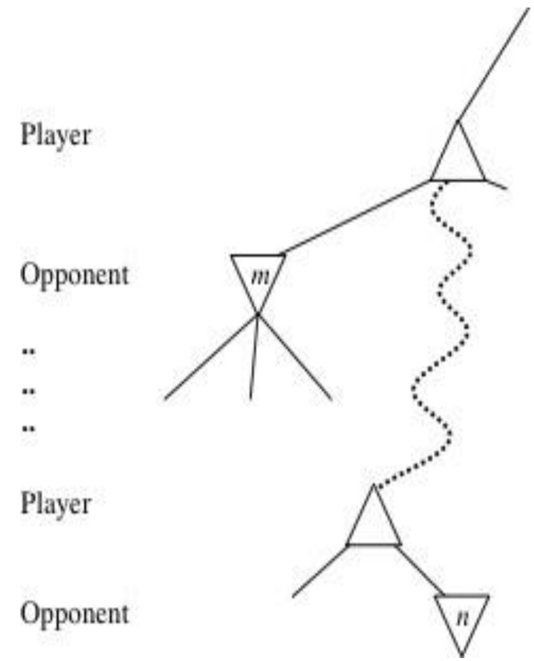
- Limitations of Mini-max search:
  - Mini-max search traverse the entire search tree but it is not always feasible to traverse entire tree.
  - Time limitations

Alpha-beta pruning

- We can improve on the performance of the mini-max algorithm through alpha-beta pruning.

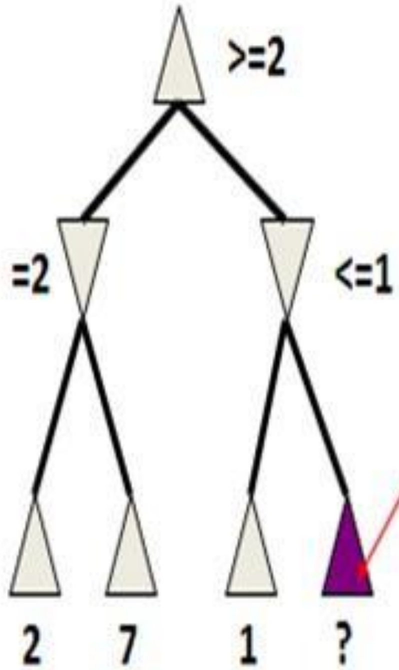
- **Basic idea:** If a move is determined worse than another move already examined, then there is no need for further examination of the node.

- **For Example:** Consider node *n* in the tree.
- If player has a better choice at:
  - Parent node of n
  - Or any choice point further up
- Then *n* is never reached in play.
- So, When that much is known about *n*, it can be pruned.



Contd…

- Example:



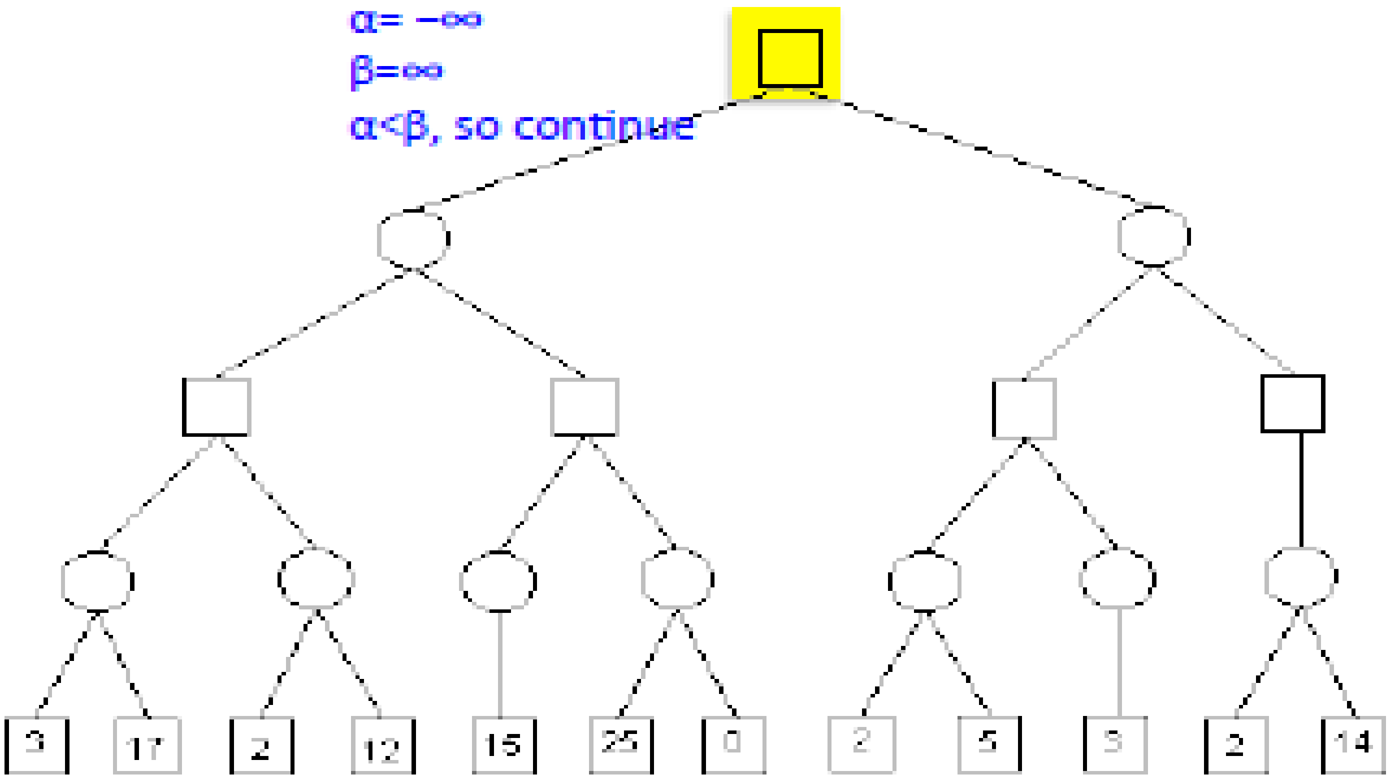
- We don't need to compute the value at this node.
- No matter what it is it can't effect the value of the root node.

Contd…

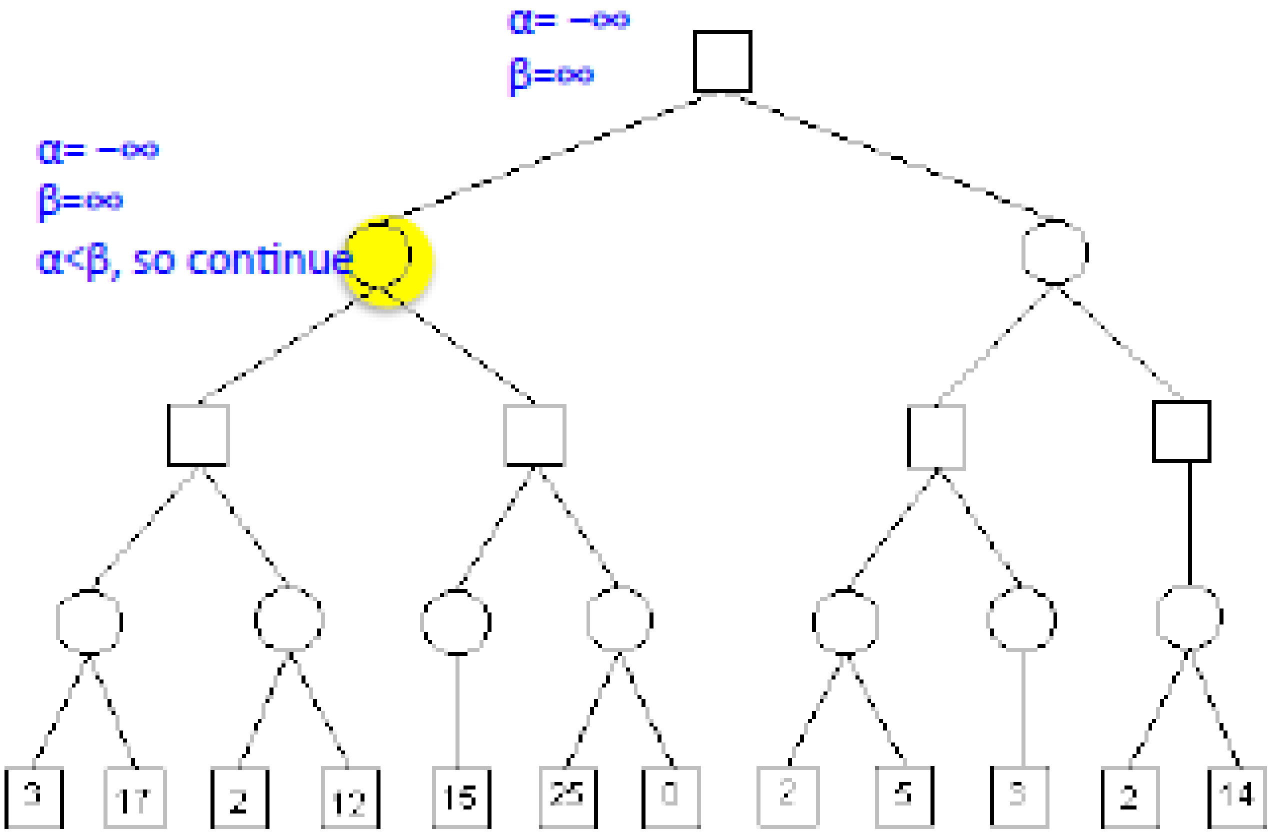
- Alpha-Beta pruning procedure:
  - Traverse the search tree in depth-first order. During traversing Alpha and Beta values inherited from the parent to child never from the children. Initially alpha =-infinity and always try to increase, and beta=+infinity and always try to decrease. Alpha value updates only at max node and beta value update only at min node.
  - Max player :
    - Val > Alpha?(val is Value back up form the children of max player)
      - Update Alpha
    - Is Alpha >= Beta?
      - Prune (called alpha cutoff)
    - Return Alpha
  - MIN player:
    - Val< Beta? (val is Value back up form the children of min player)
      - Update Beta
    - Is Alpha>= Beta?
      - Prune (called beta cutoff)
    - Return Beta

Contd…

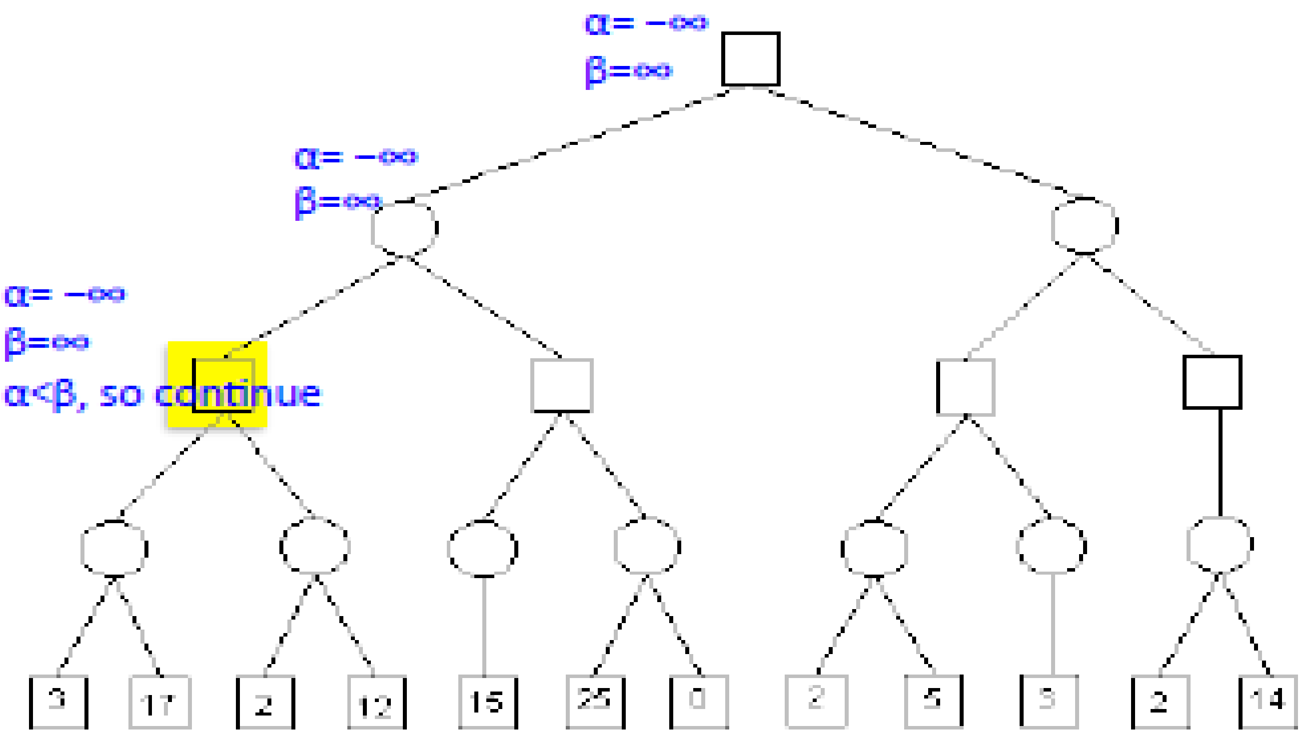
- Alpha-Beta pruning example:



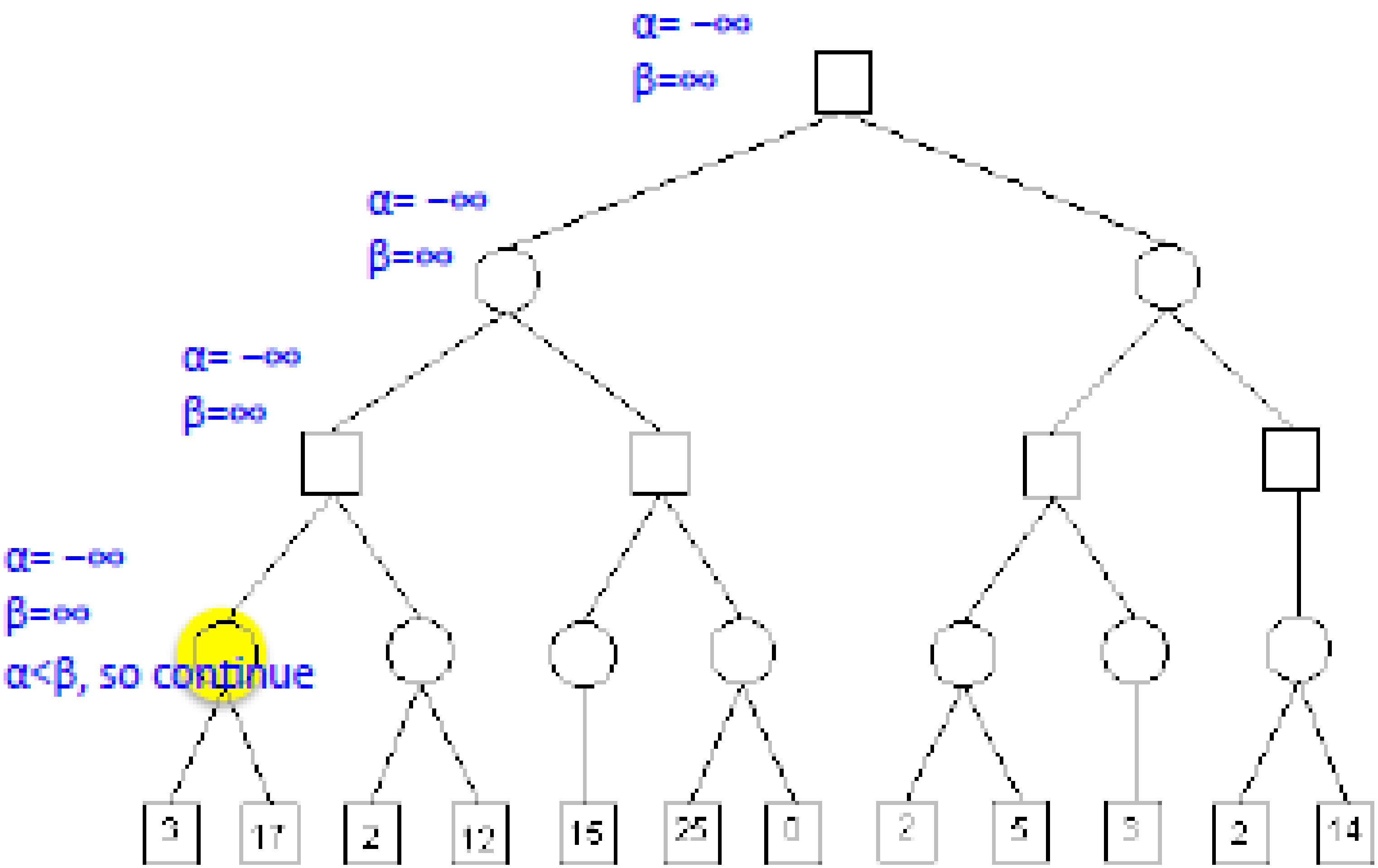
Contd…



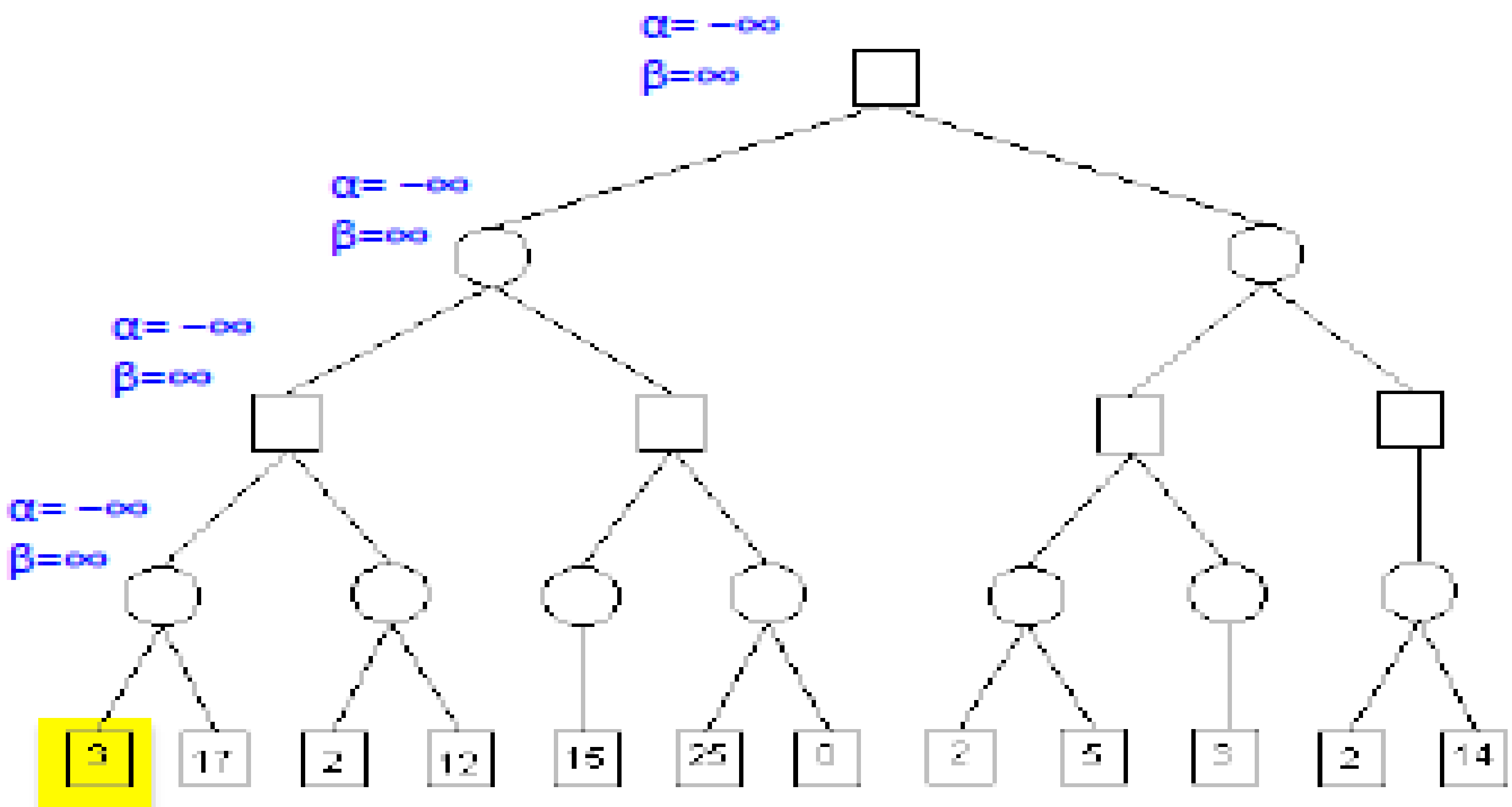
Contd…



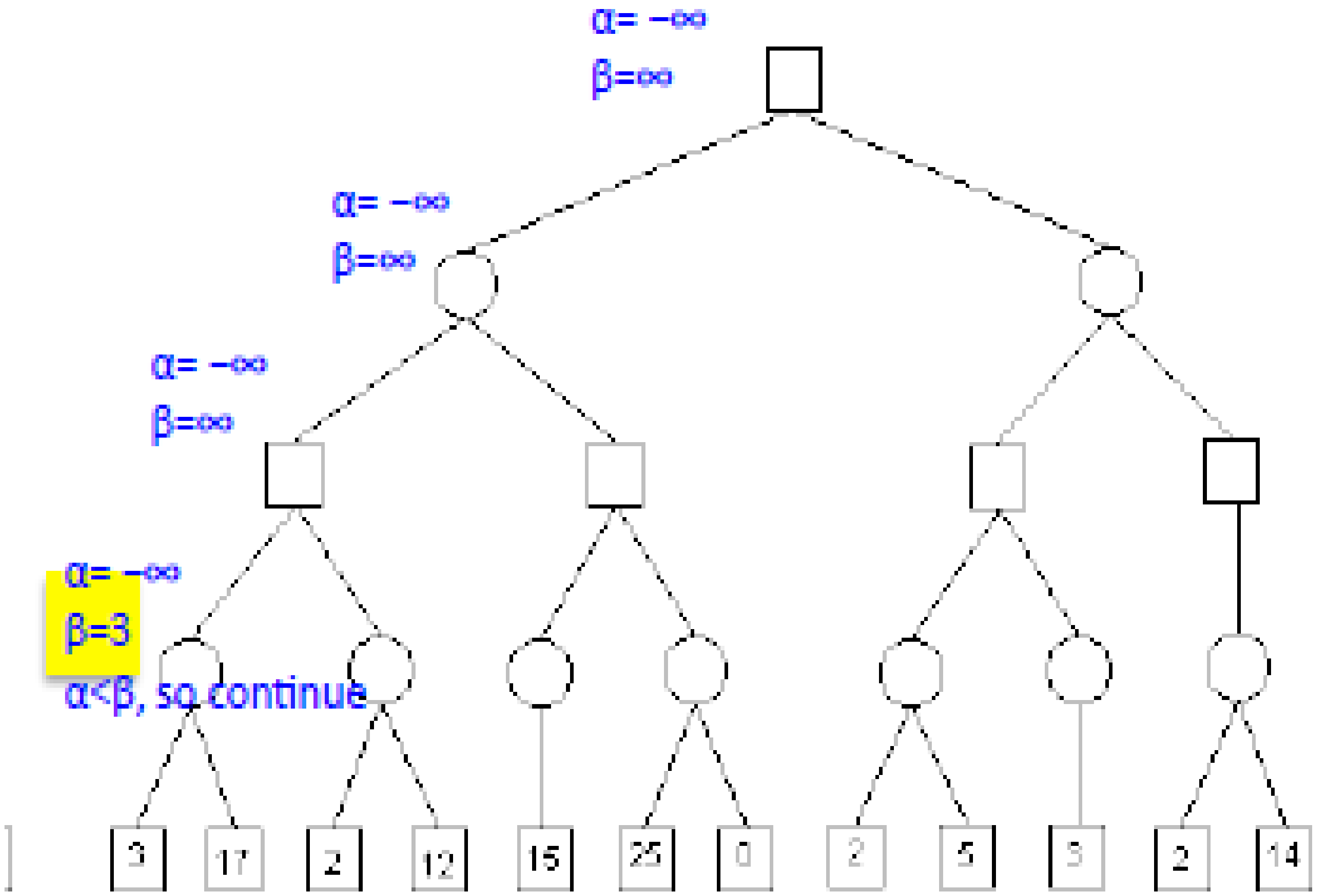
Contd…



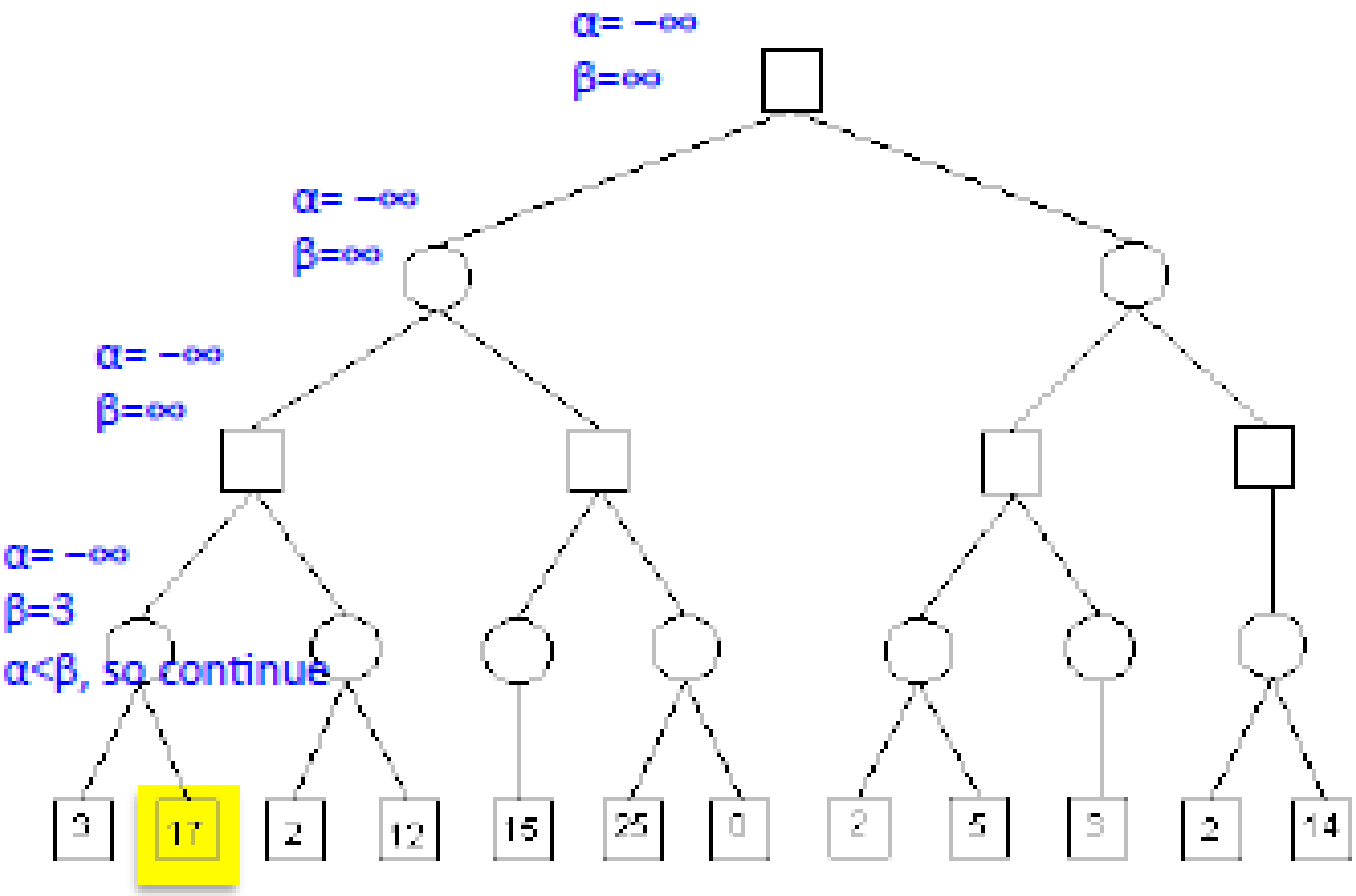
Contd…



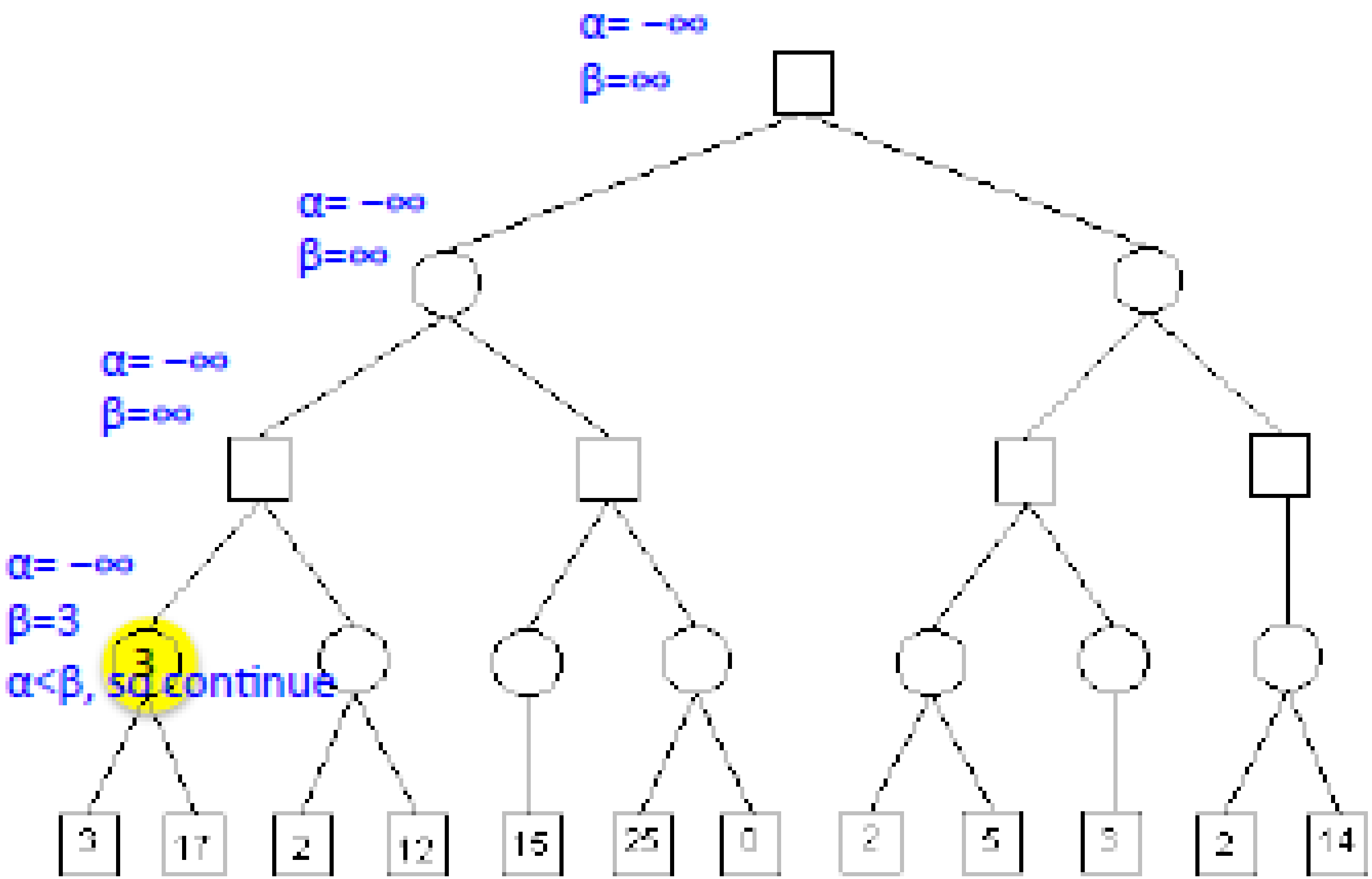
Contd…



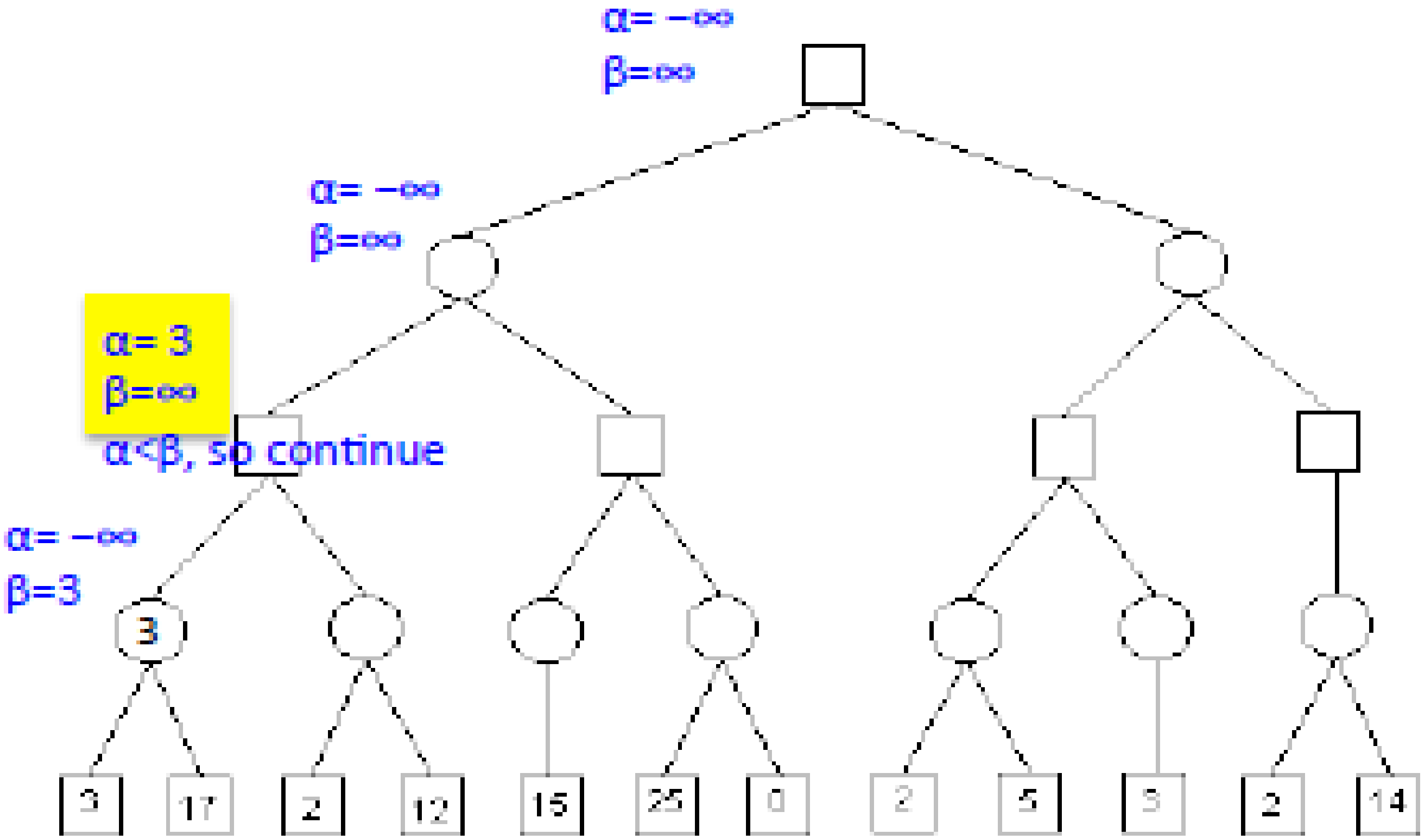
Contd…



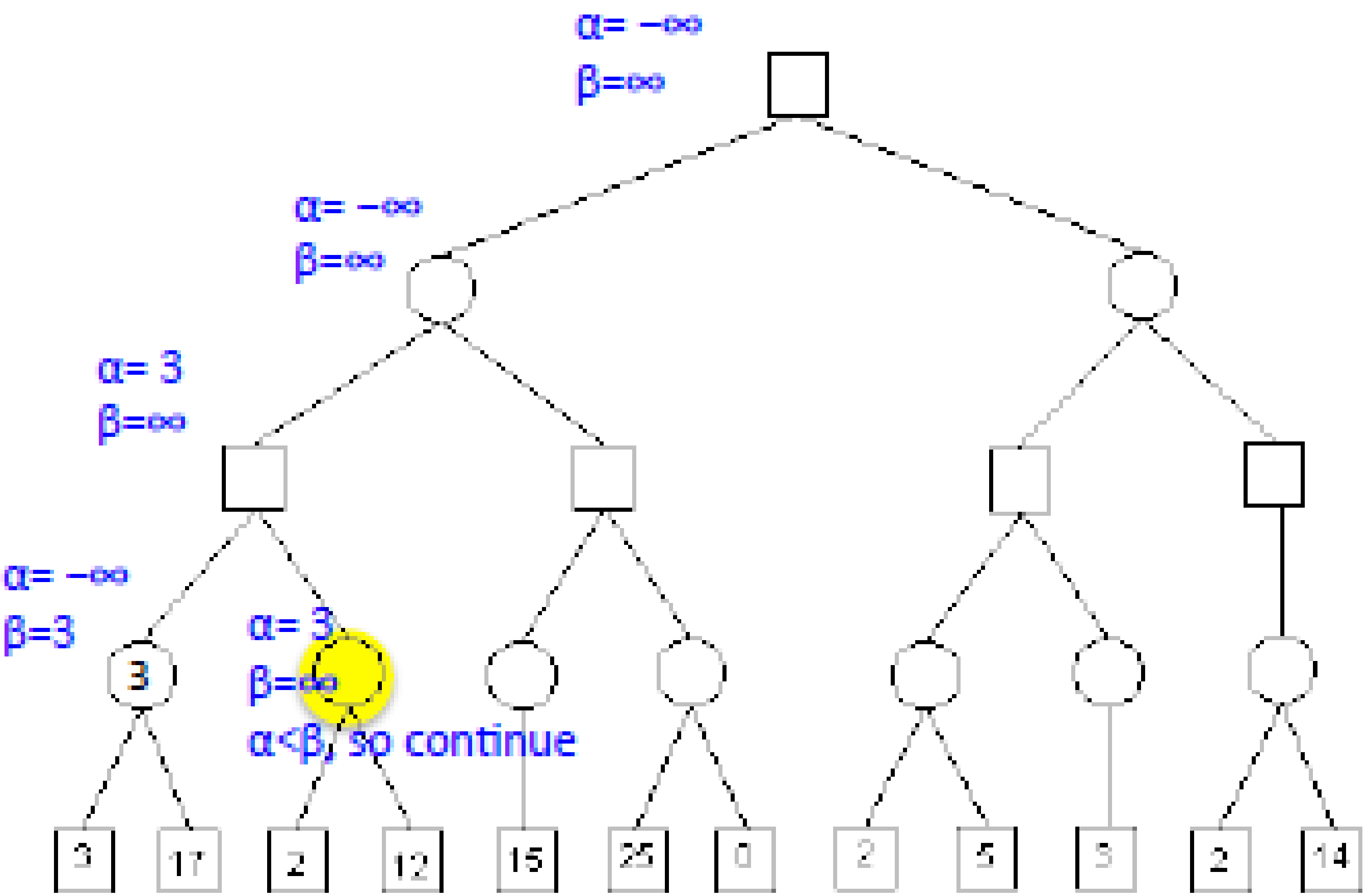
Contd…



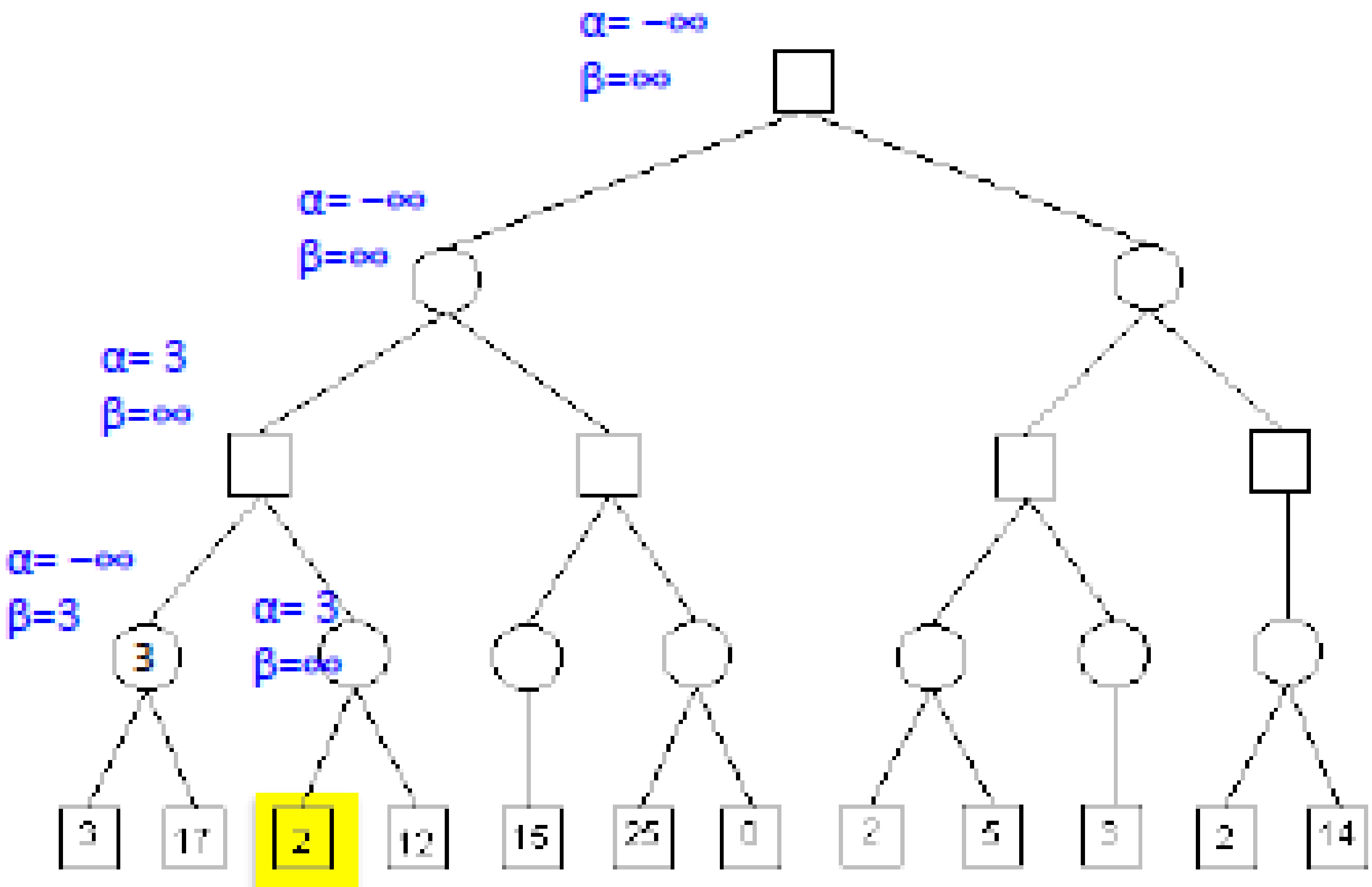
Contd…



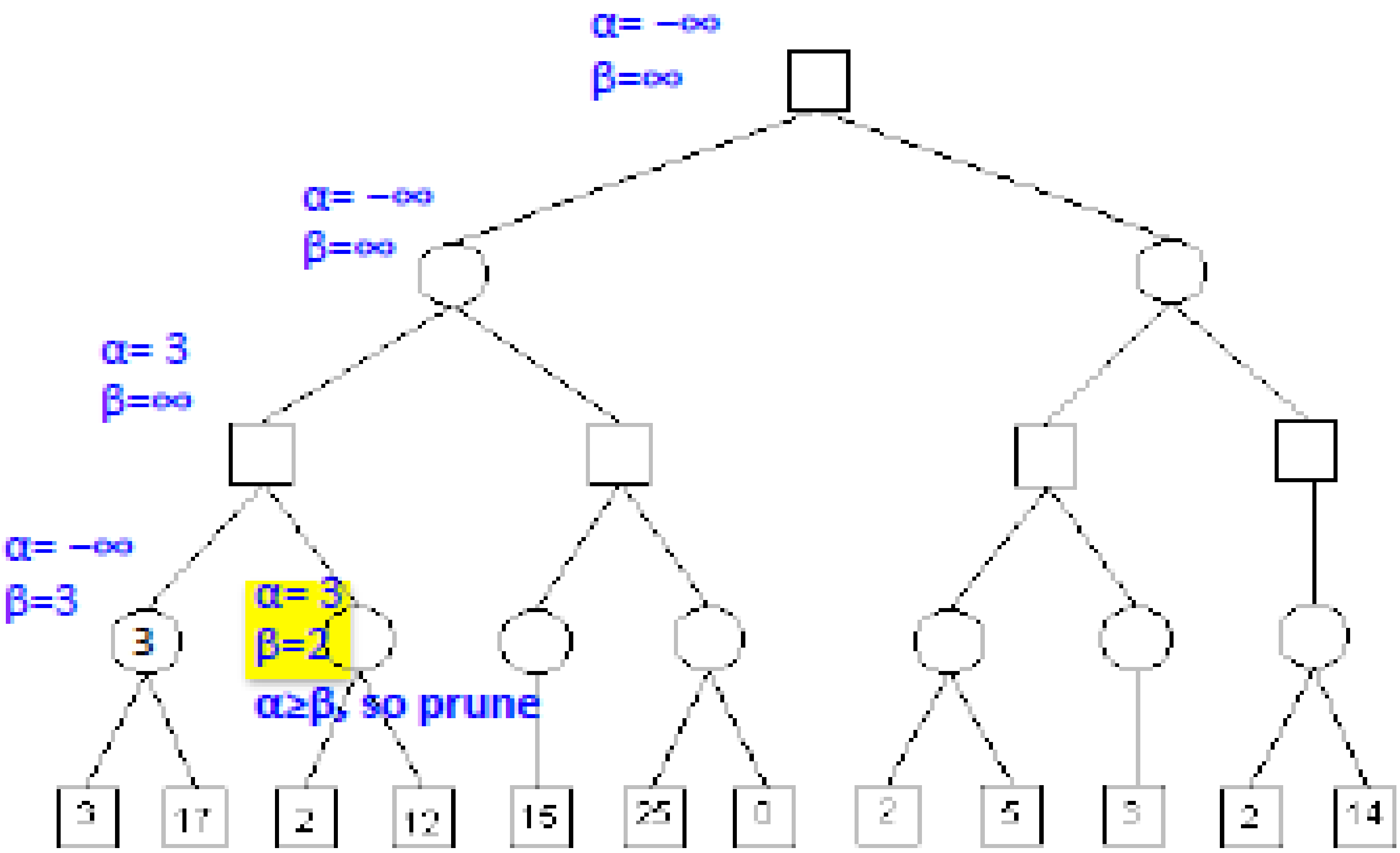
Contd…



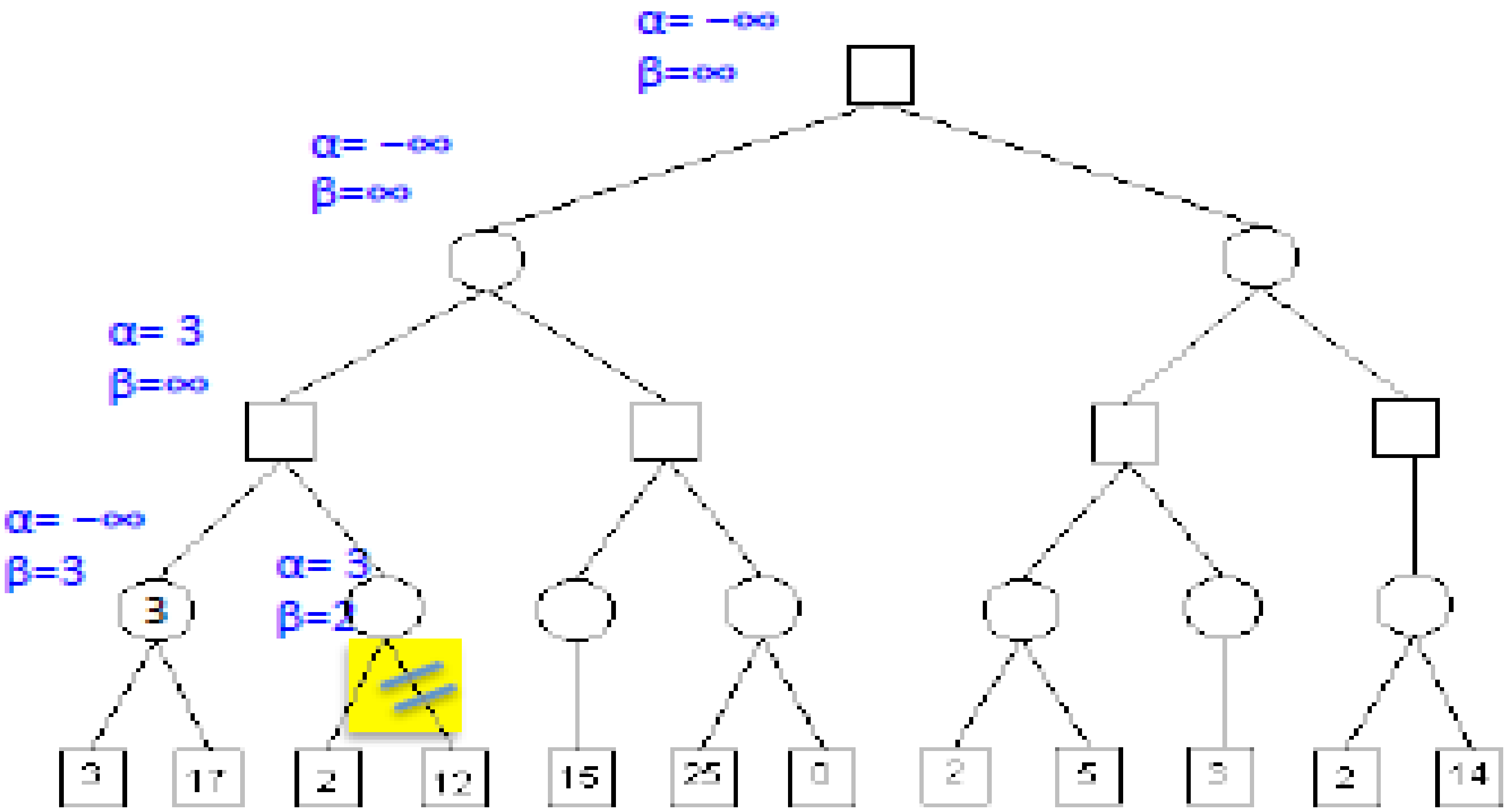
Contd…



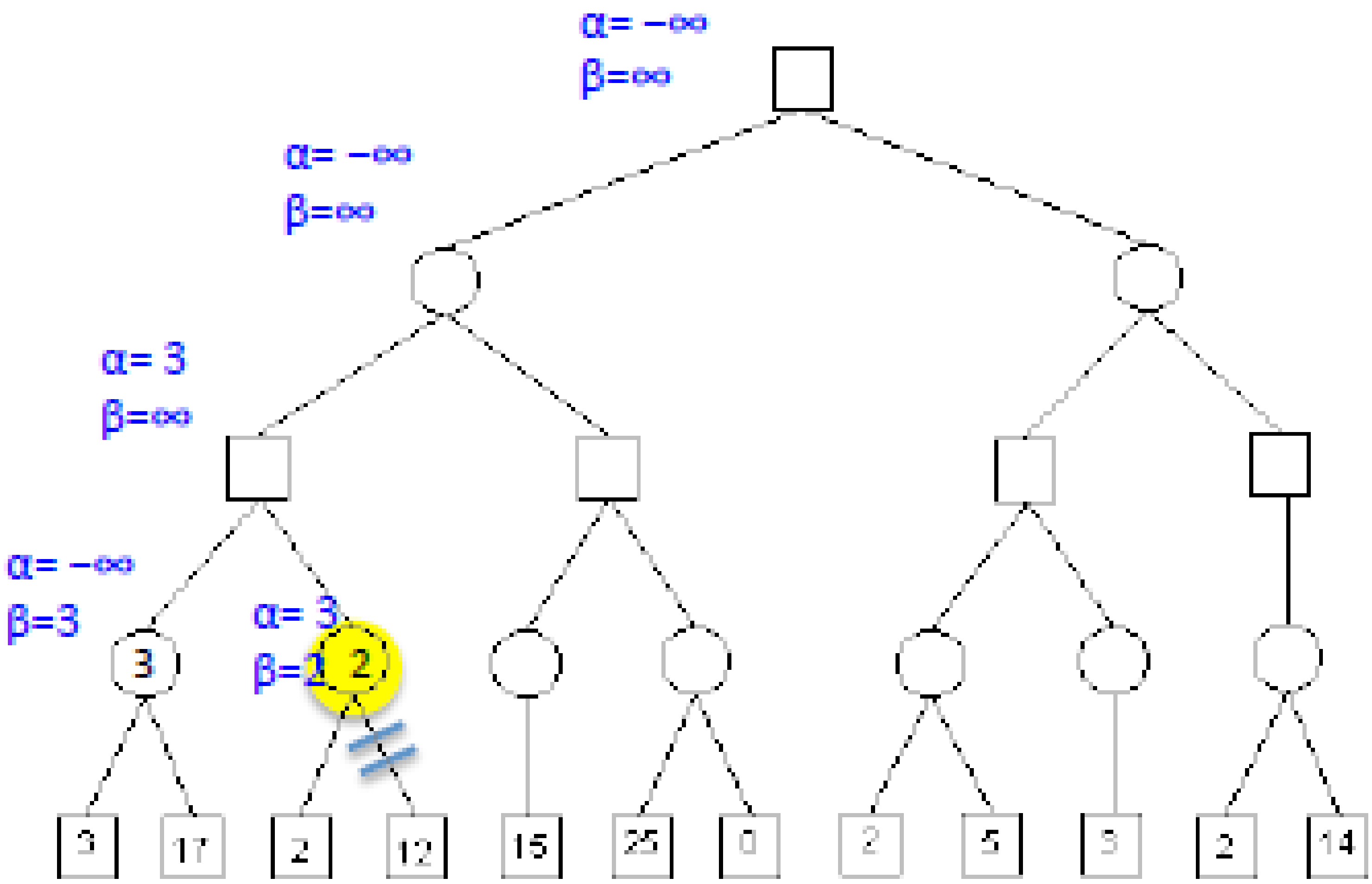
Contd…



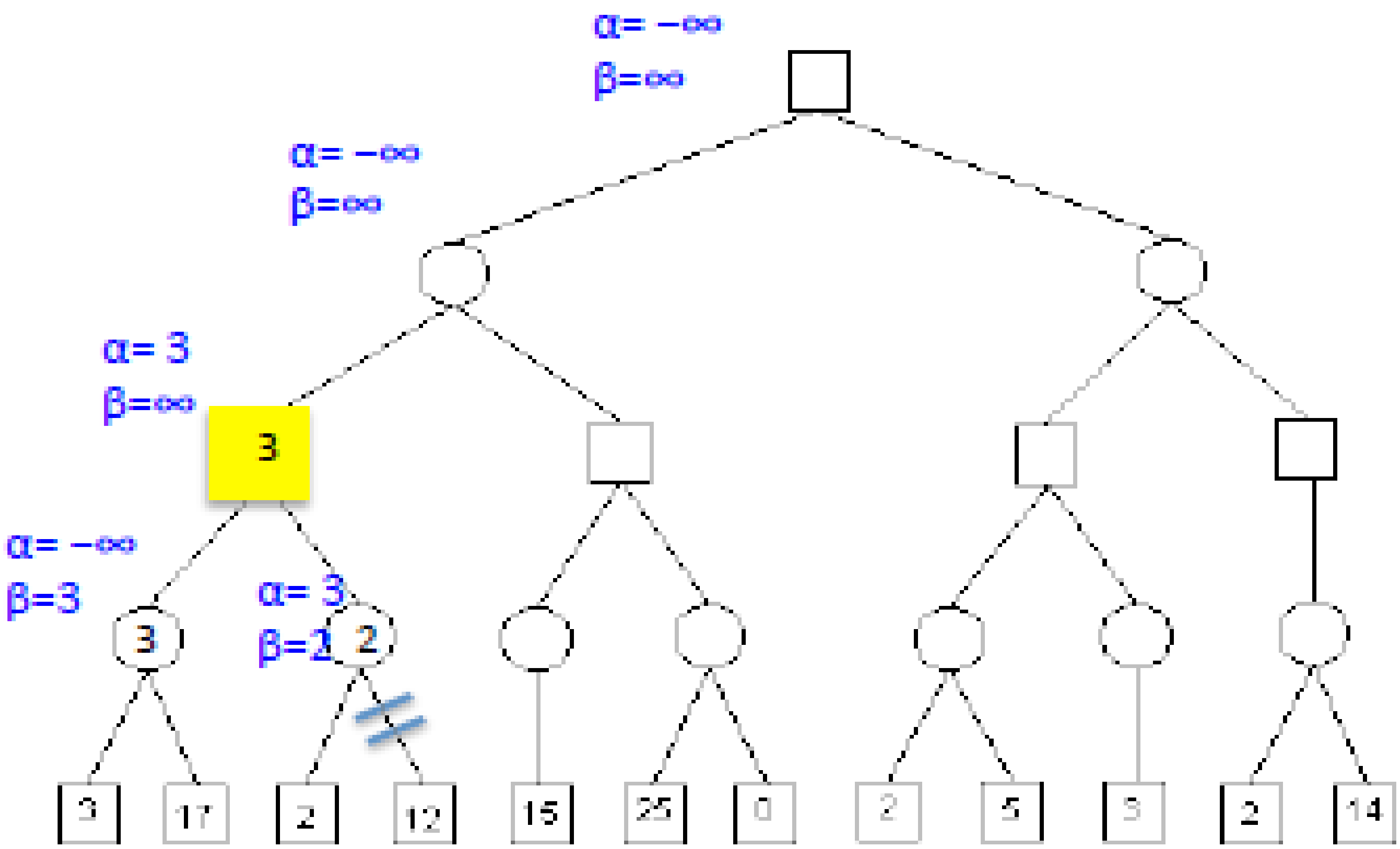
Contd…



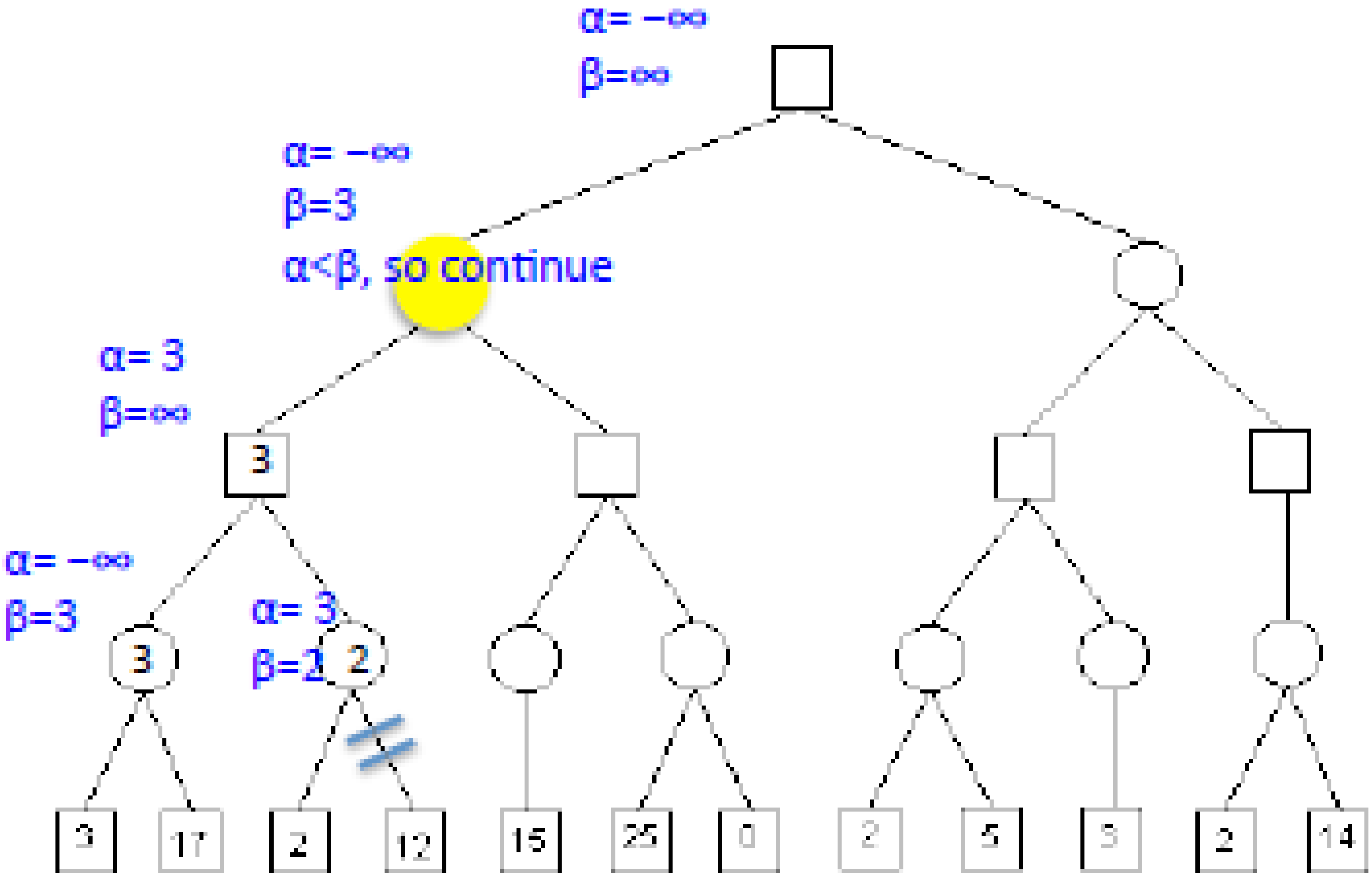
Contd…



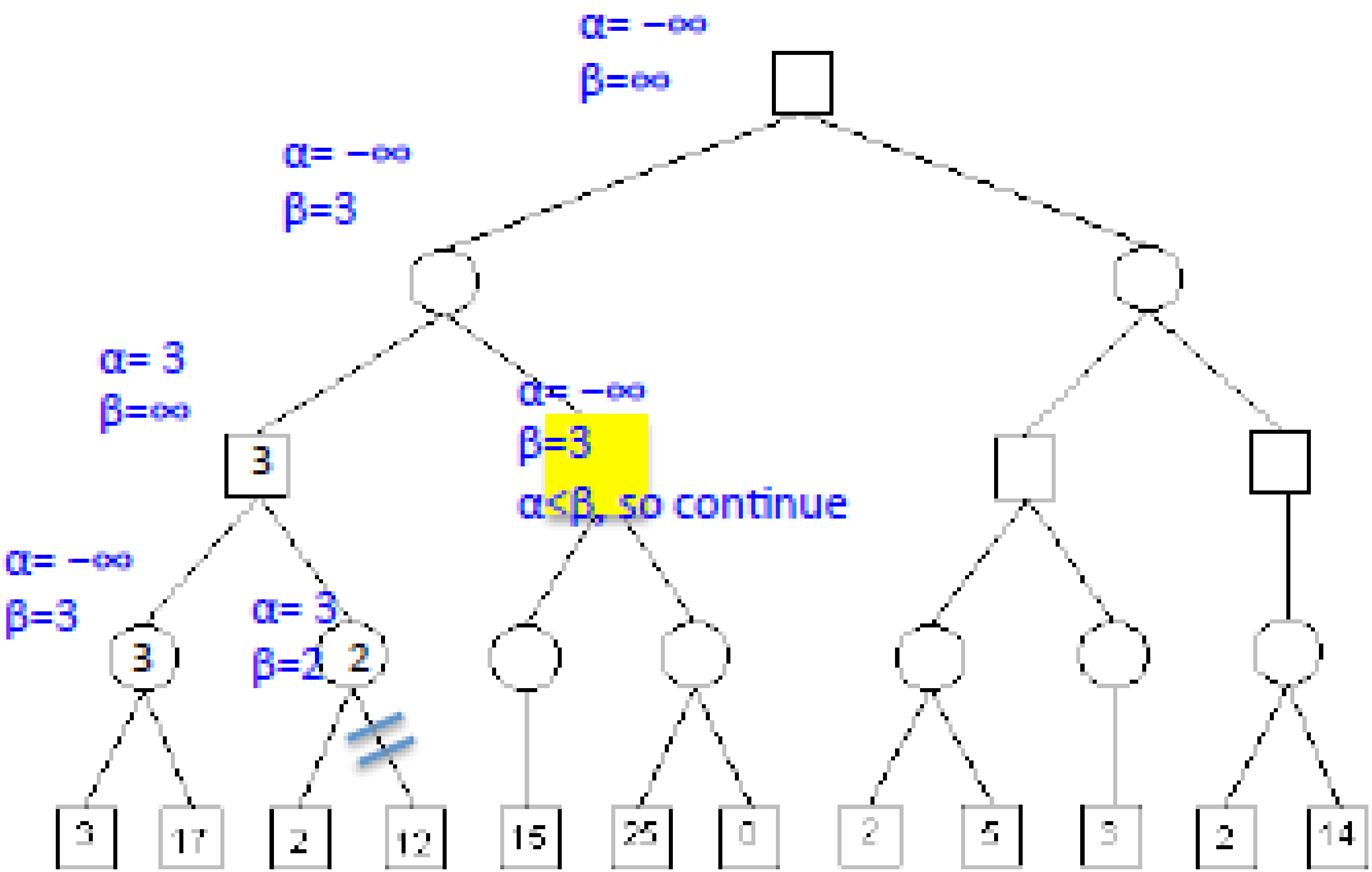
Contd…



Contd…

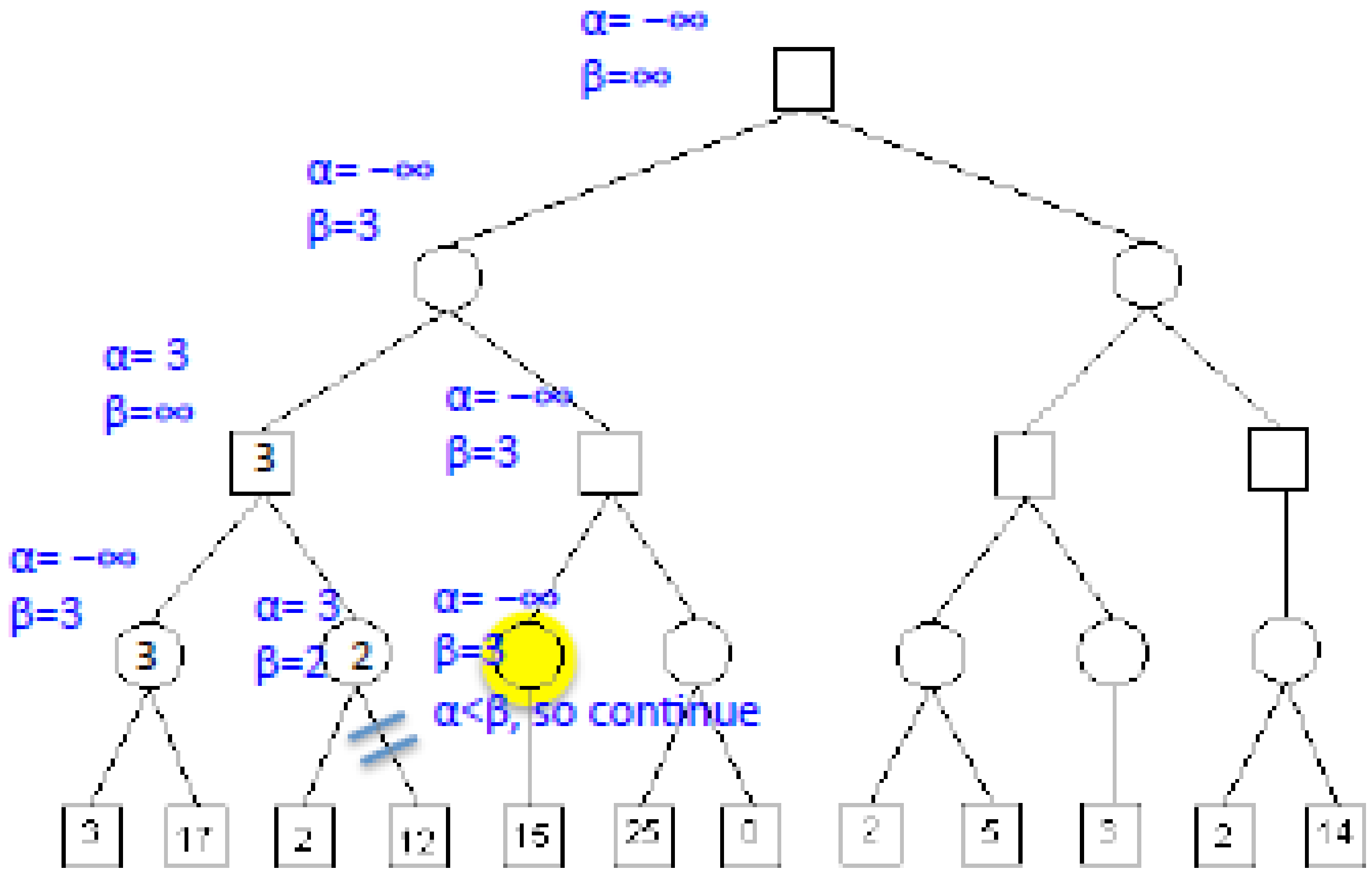


Contd…



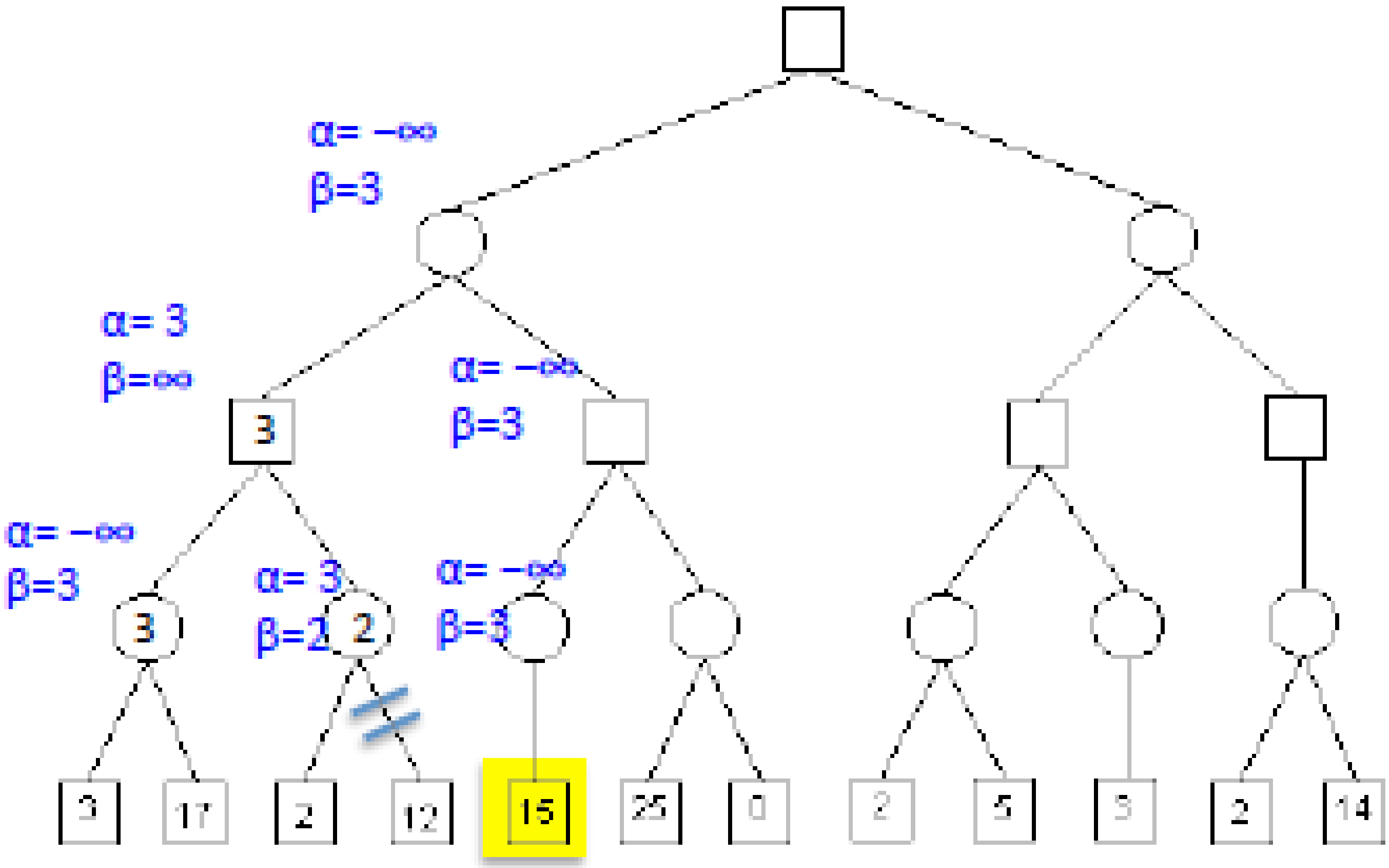
α= −∞
β=∞
α= −∞
β=3
α= 3
β=∞
3
α= −∞
β=3
α<β, so continue
α= −∞
β=3
3
α= 3
β=2
2
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

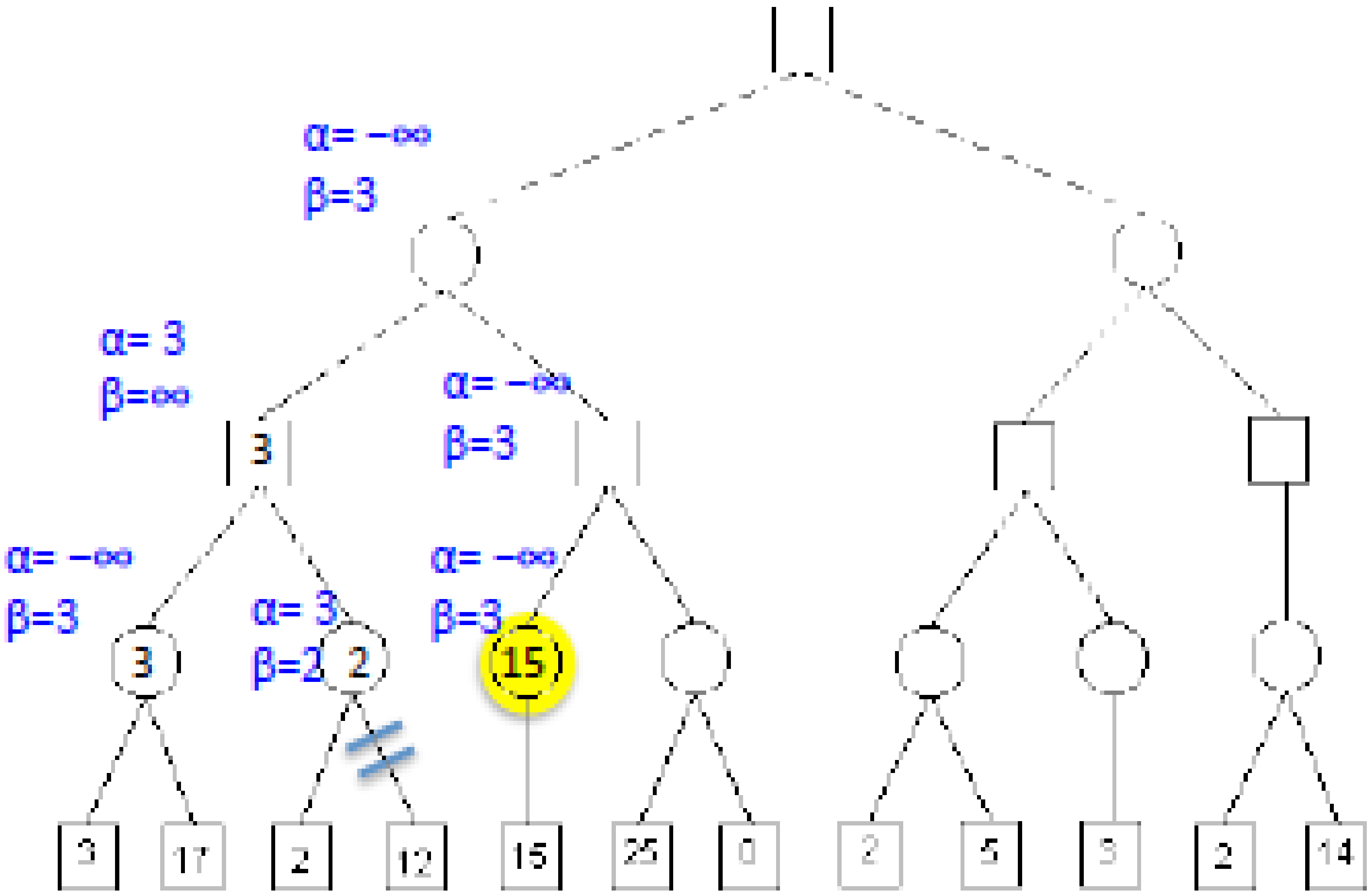


α= −∞
β=∞
α= −∞
β=3
α= 3
β=∞
α= −∞
β=3
3
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
α<β, so continue
3
17
2
12
15
25
0
2
5
3
2
14

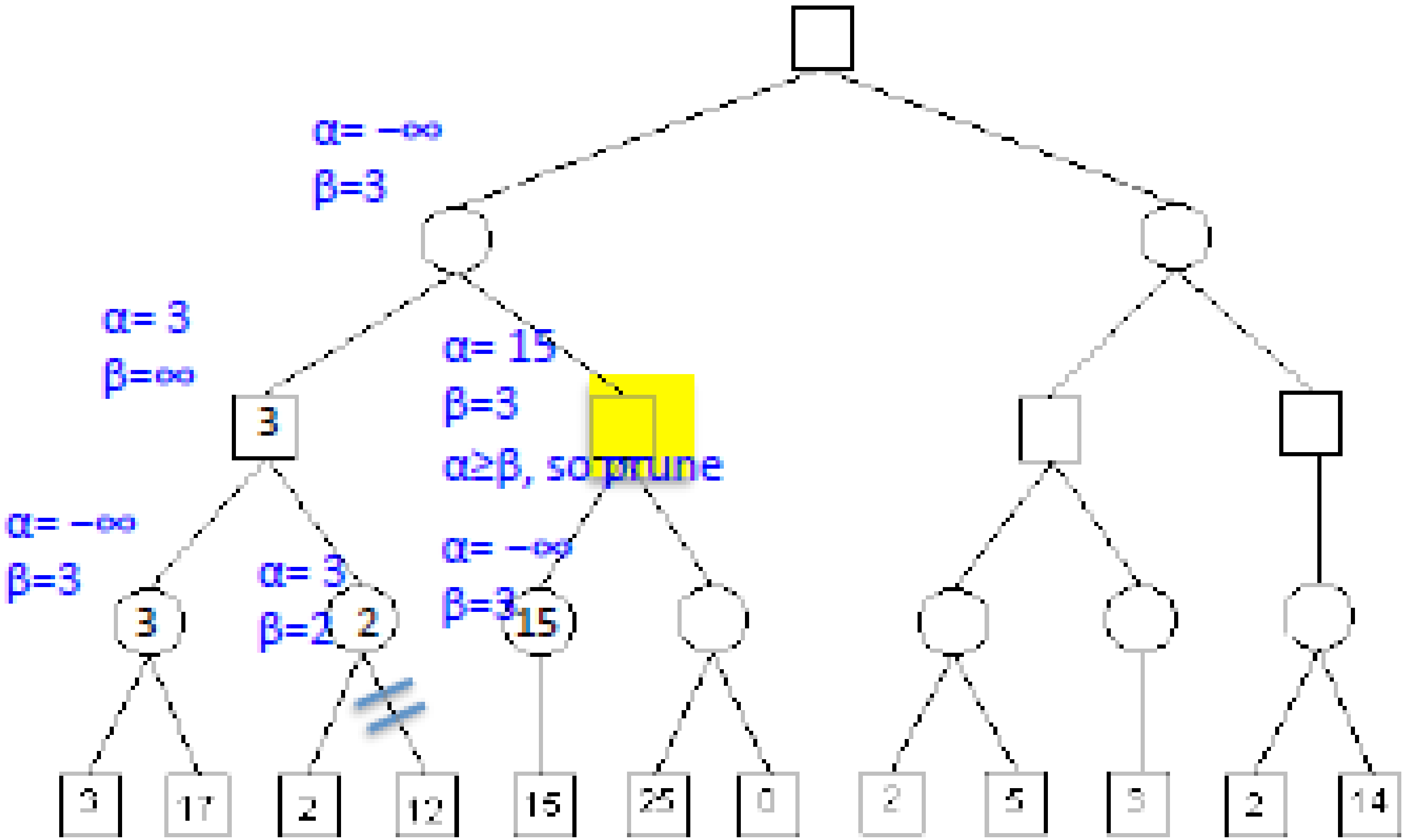
Contd…



Contd…

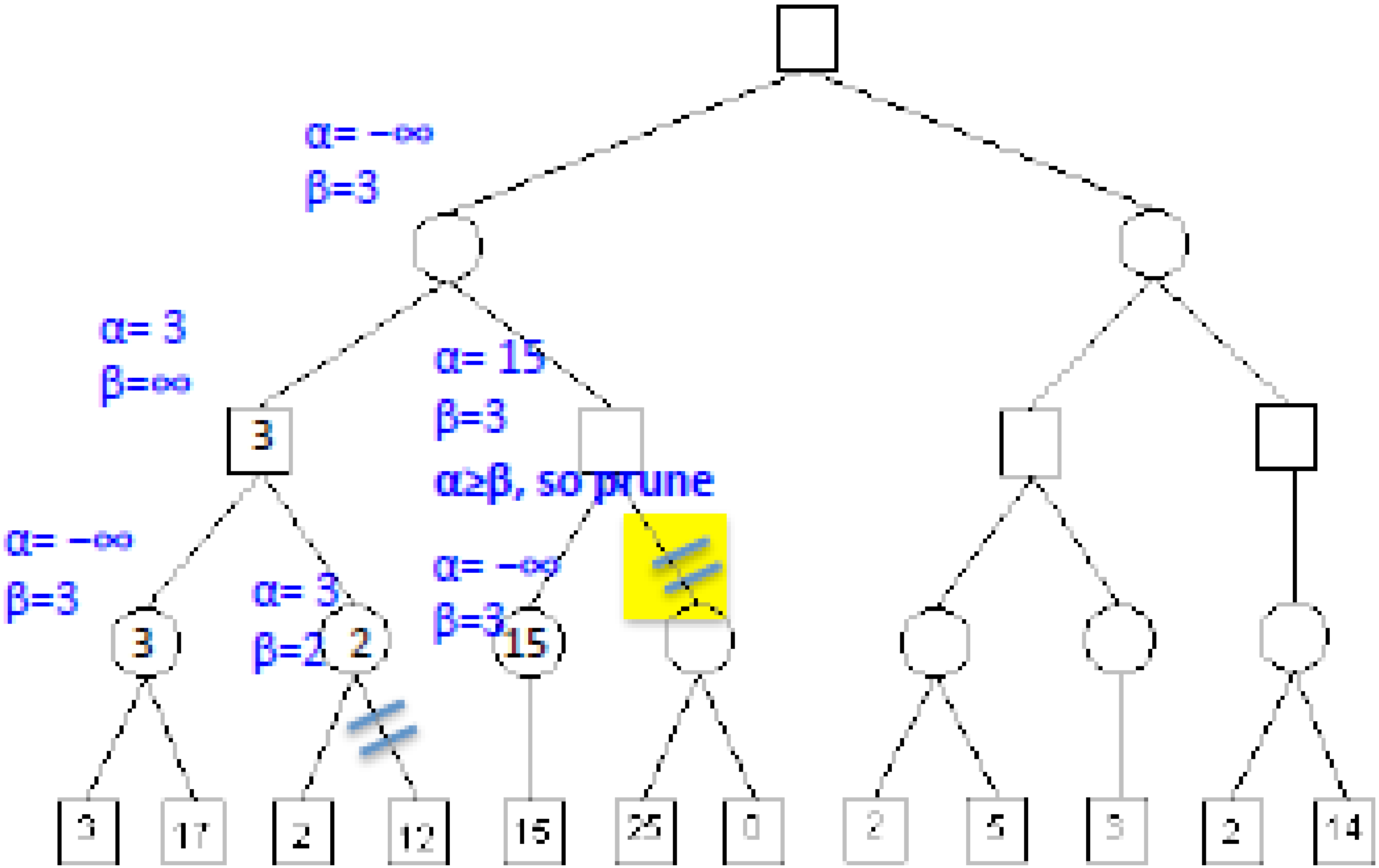


Contd…



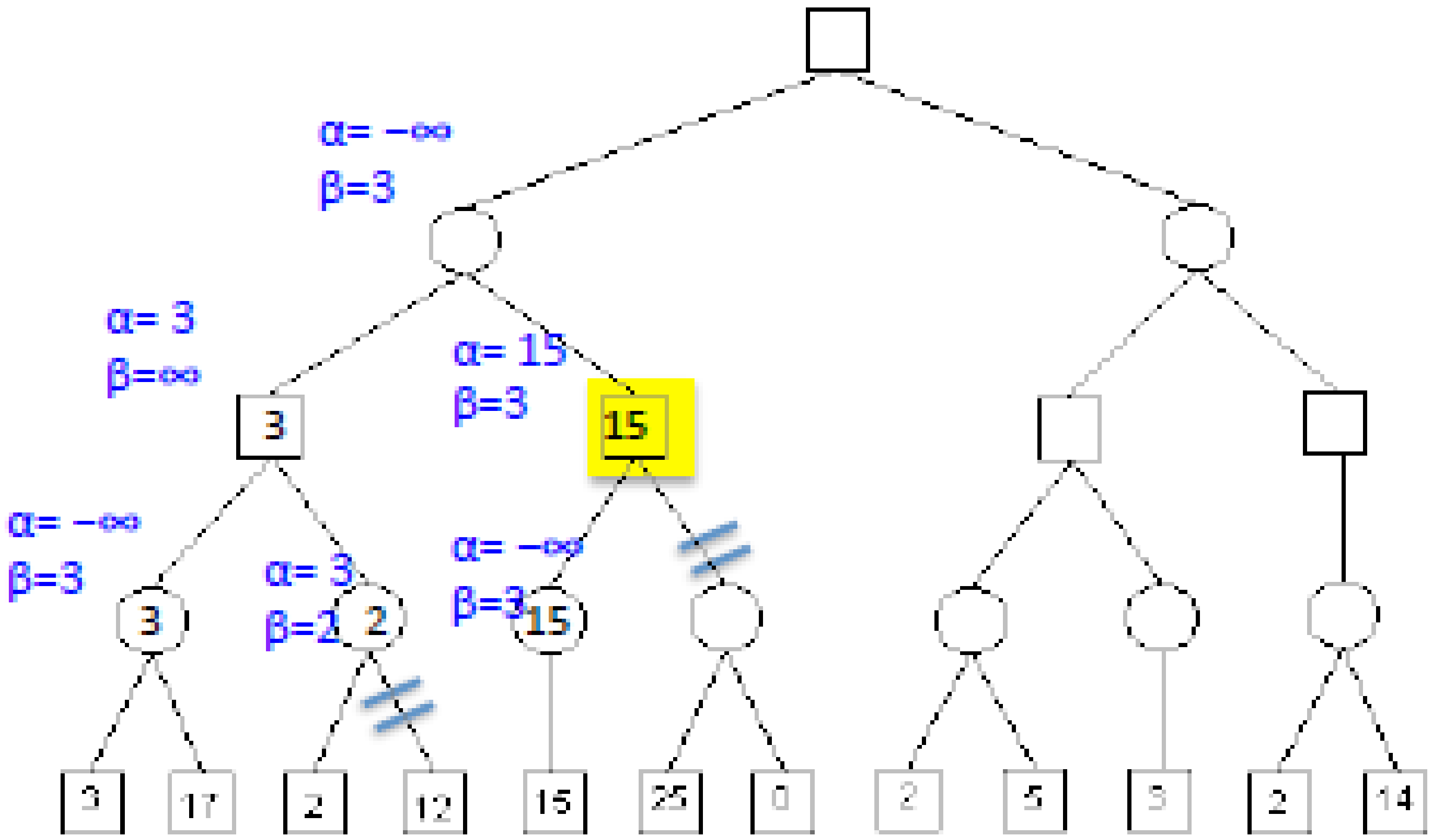
α= −∞
β=3
α= 3
β=∞
3
α= 15
β=3
α≥β, so prune
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

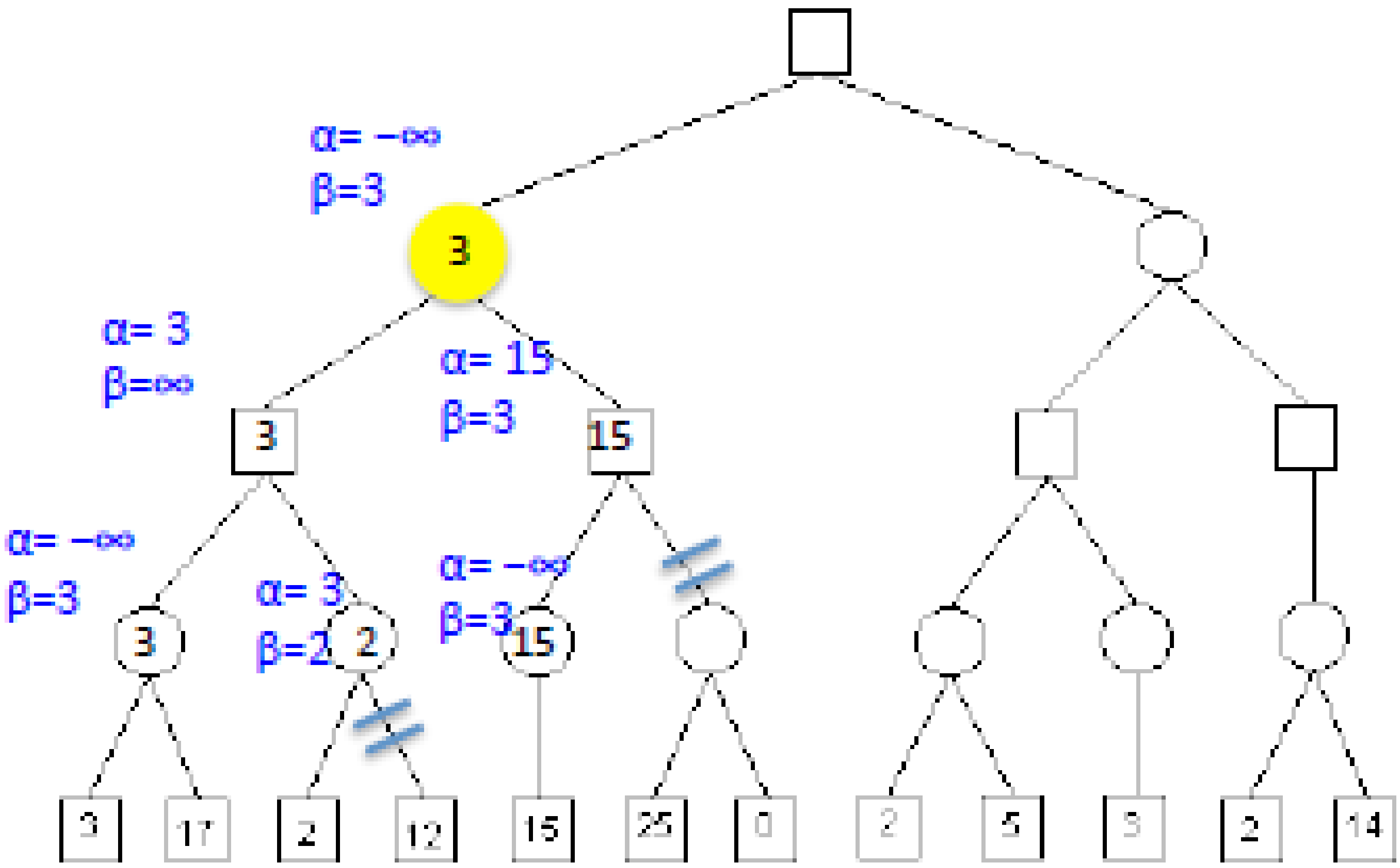


α= −∞
β=3
α= 3
β=∞
3
α= 15
β=3
α≥β, so prune
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
3
17
2
12
15
25
0
2
5
3
2
14

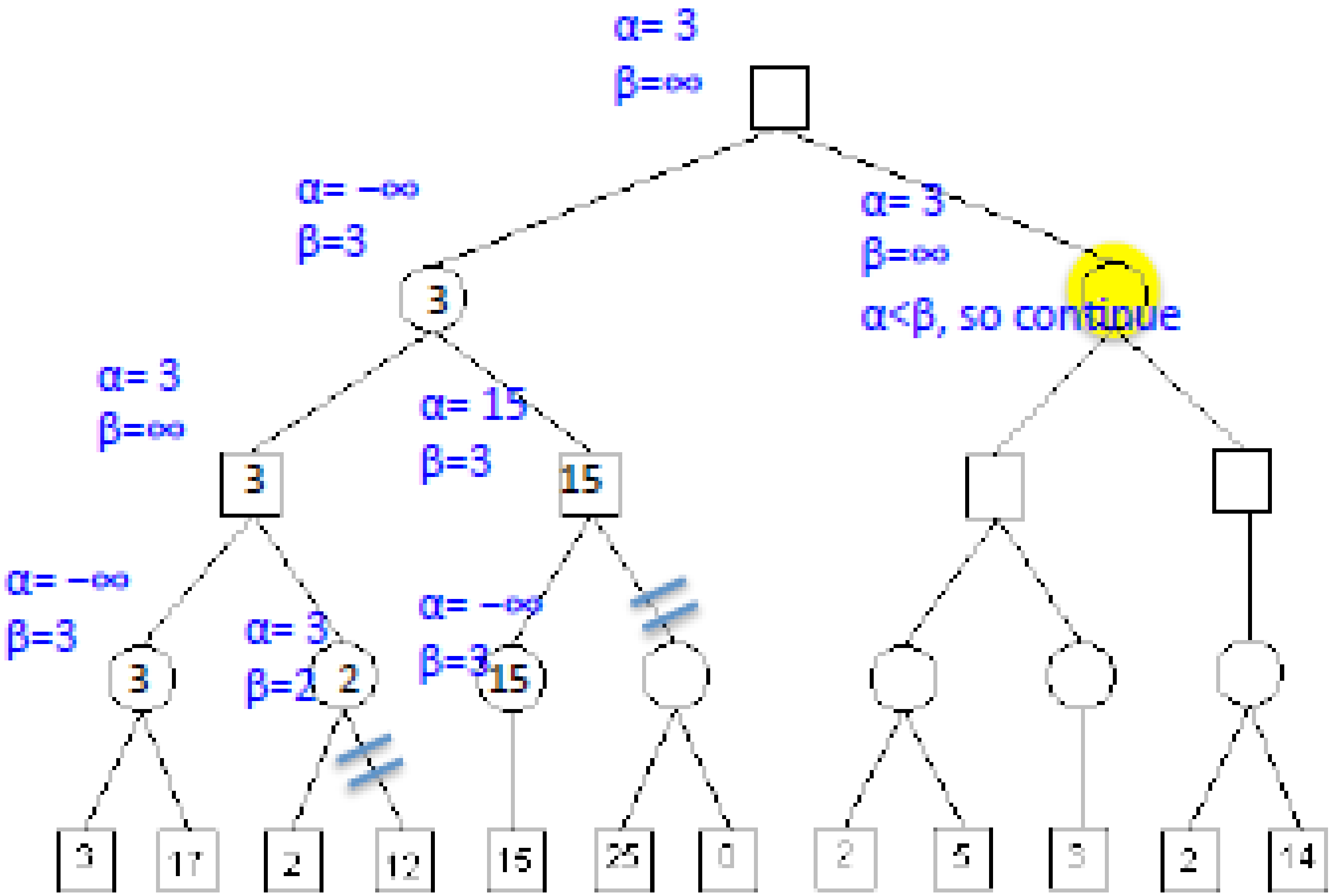
Contd…



Contd…

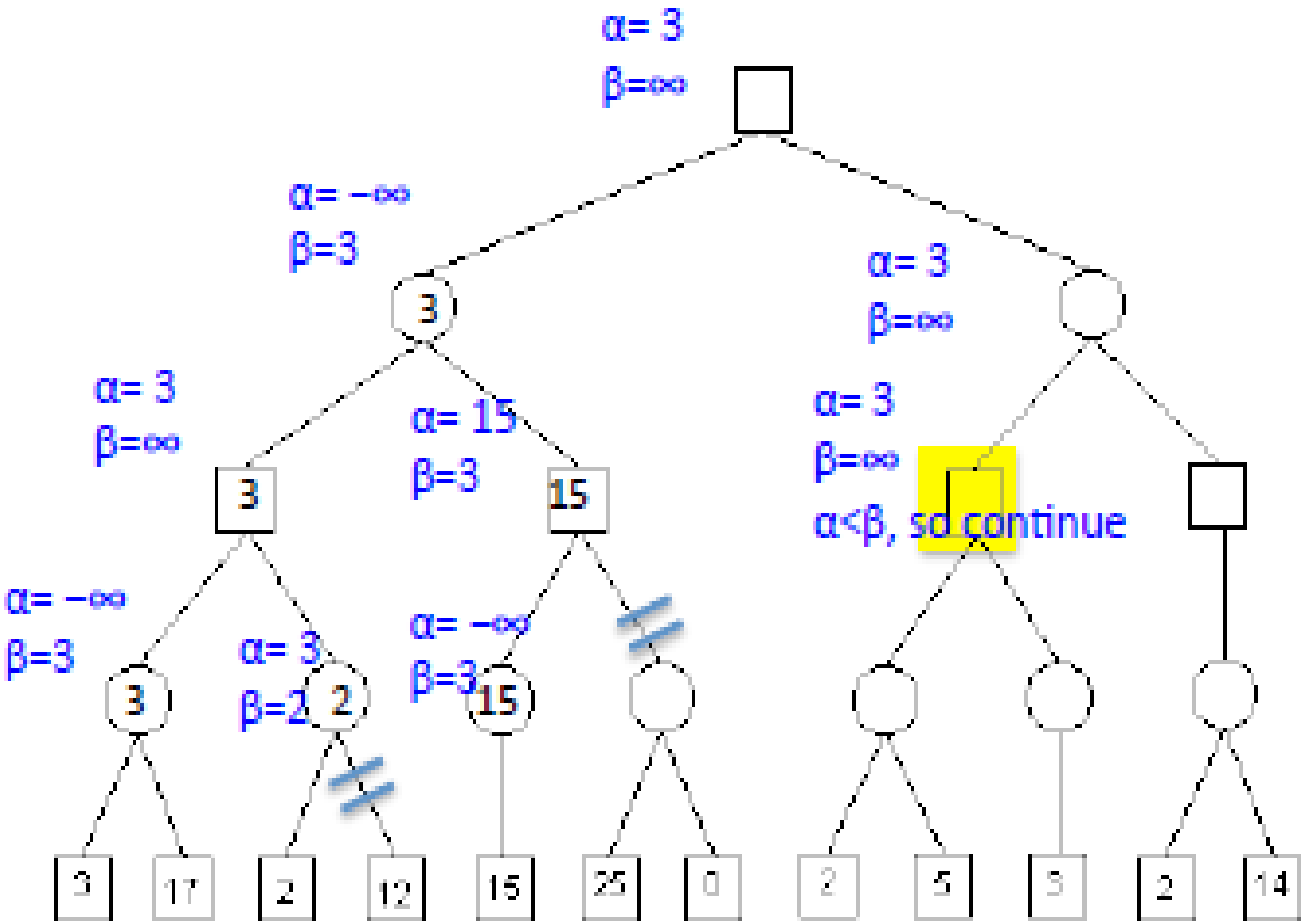


Contd…



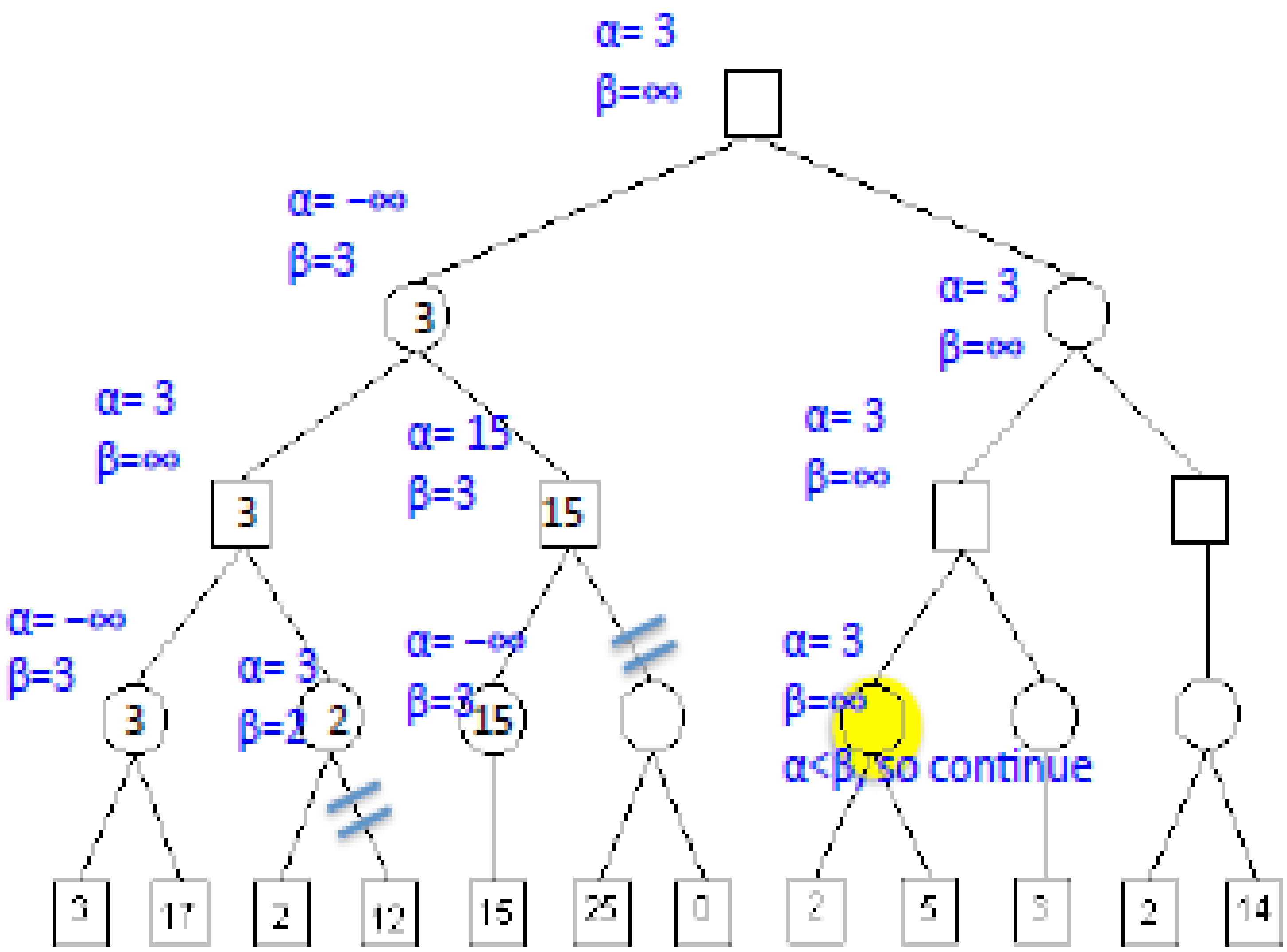
α= 3
β=∞
α= −∞
β=3
α= 3
β=∞
α<β, so continue
3
α= 3
β=∞
α= 15
β=3
3
15
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
15
3
17
2
12
15
25
0
2
5
3
2
14

Contd…



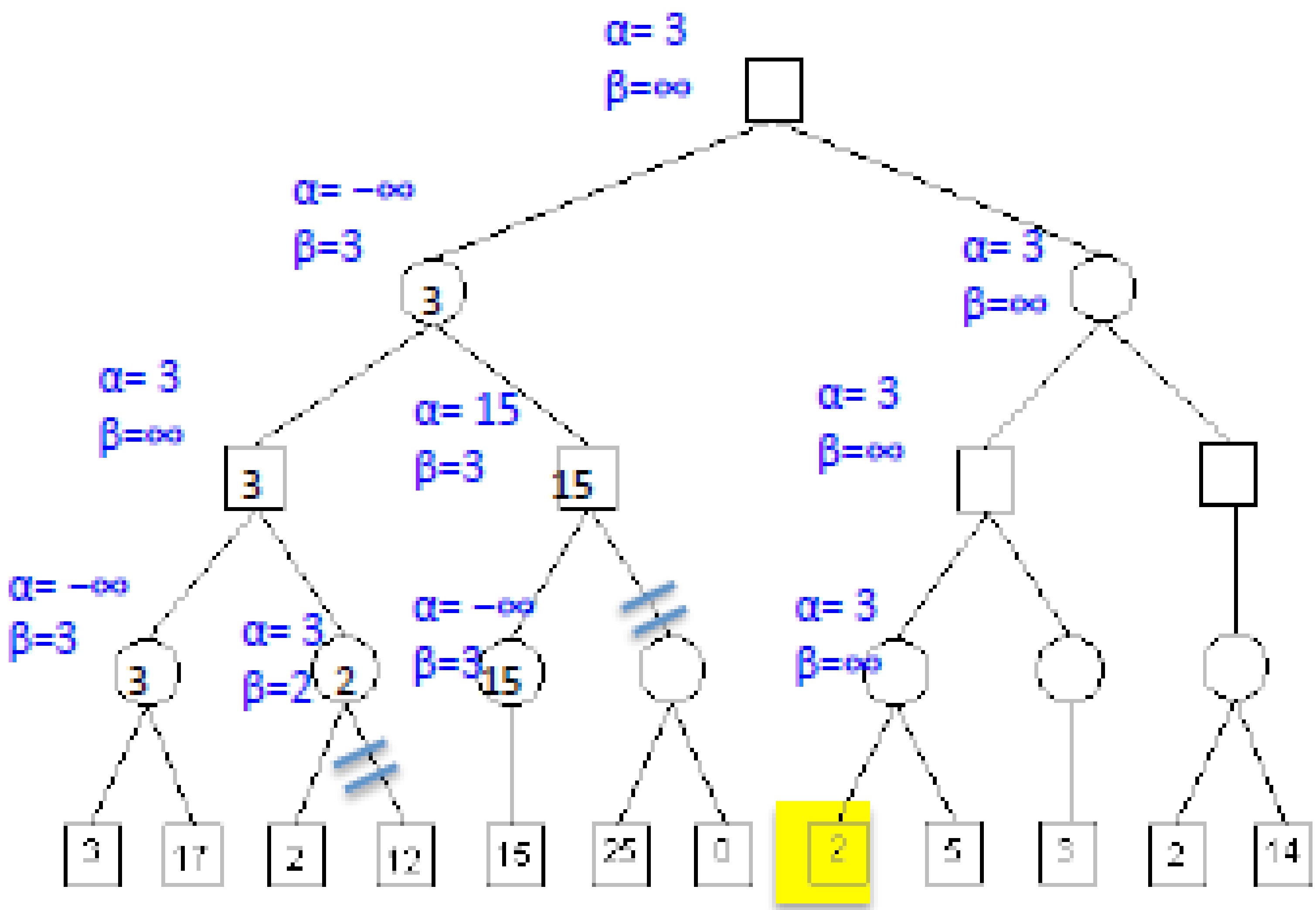
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α<β, so continue
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
3
17
2
12
15
25
0
2
5
3
2
14

Contd…



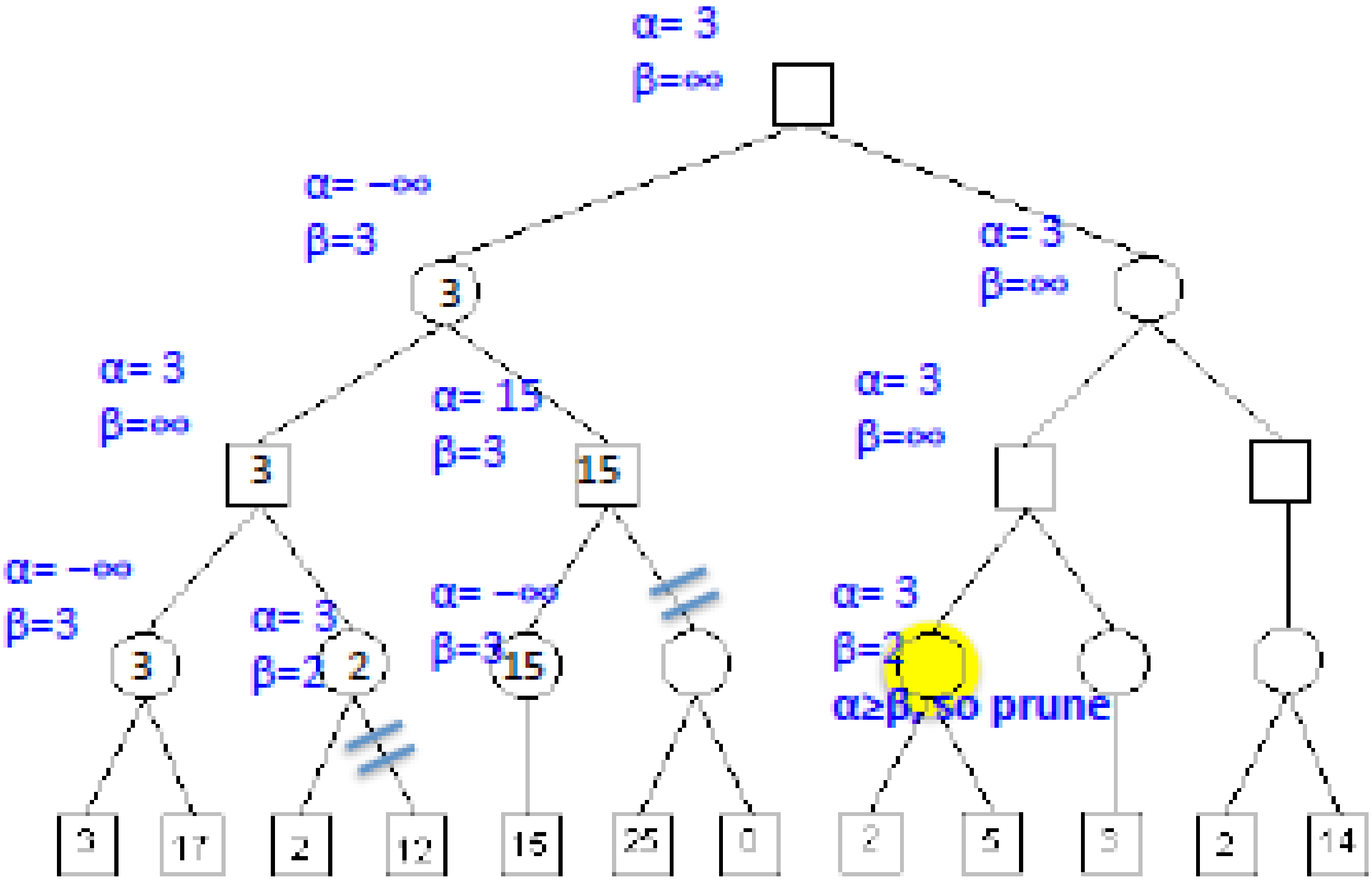
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=∞
α<β, so continue
3
17
2
12
15
25
0
2
5
3
2
14

Contd…



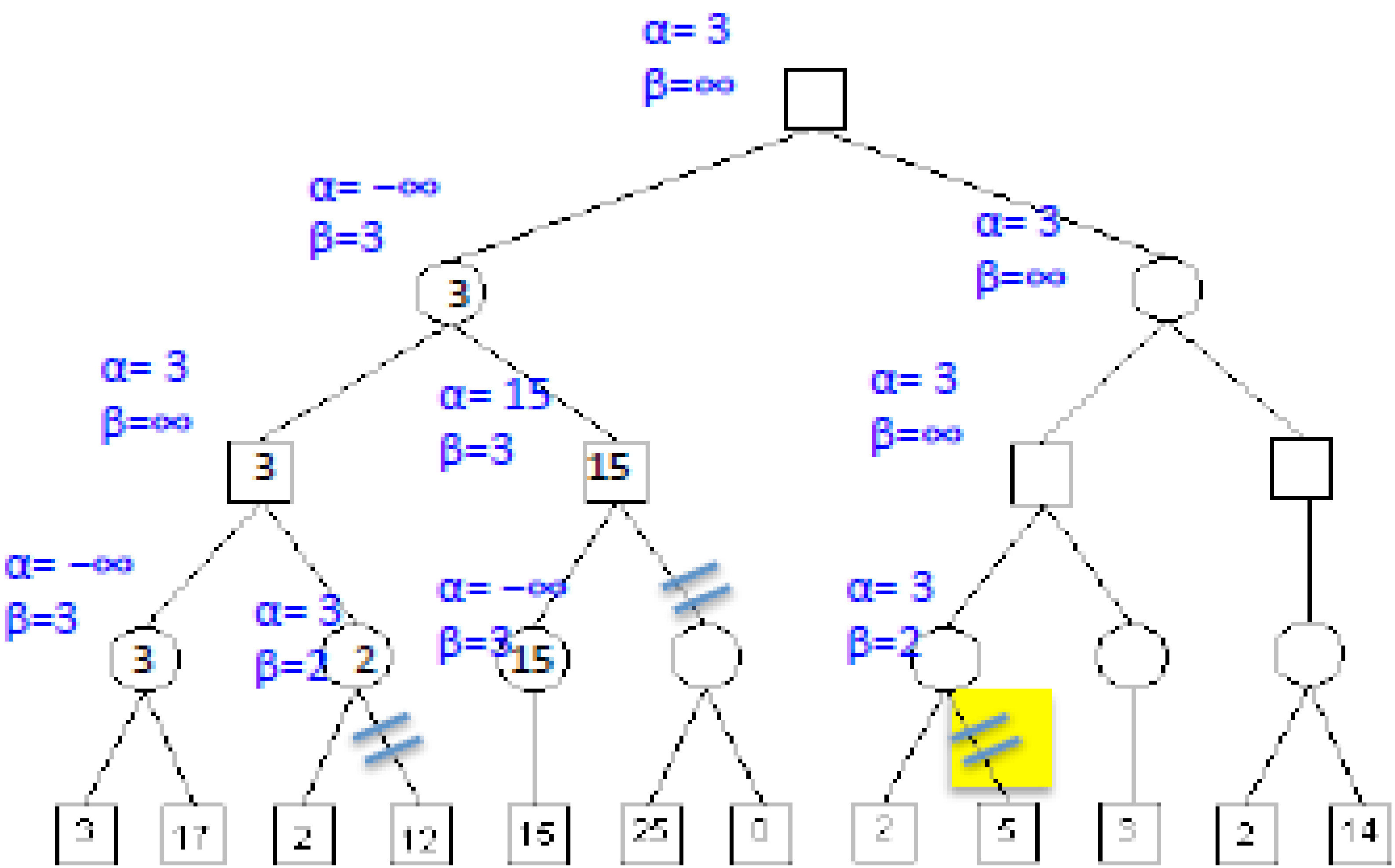
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=∞
3
17
2
12
15
25
0
2
5
3
2
14

Contd…



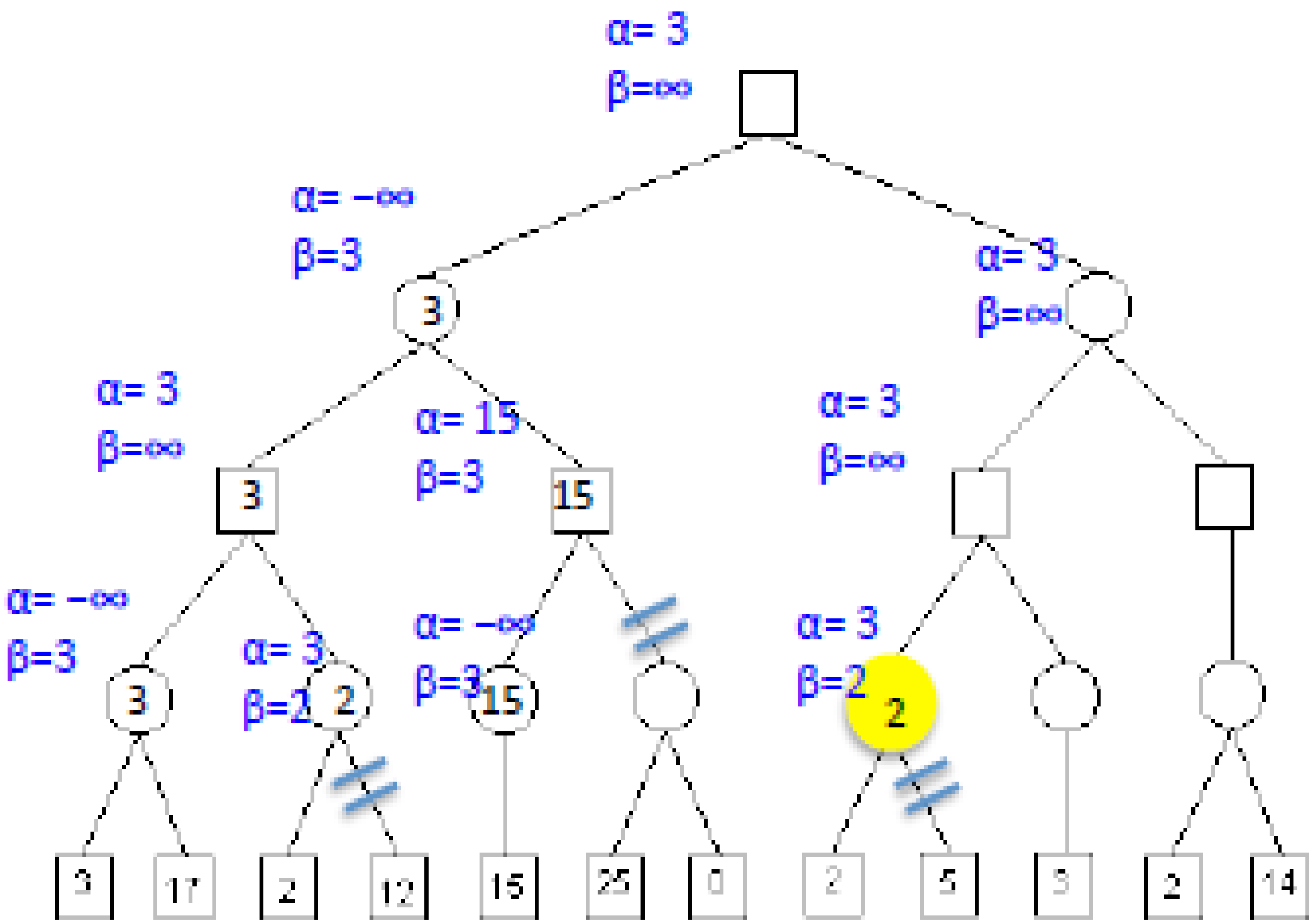
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=2
α≥β, so prune
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

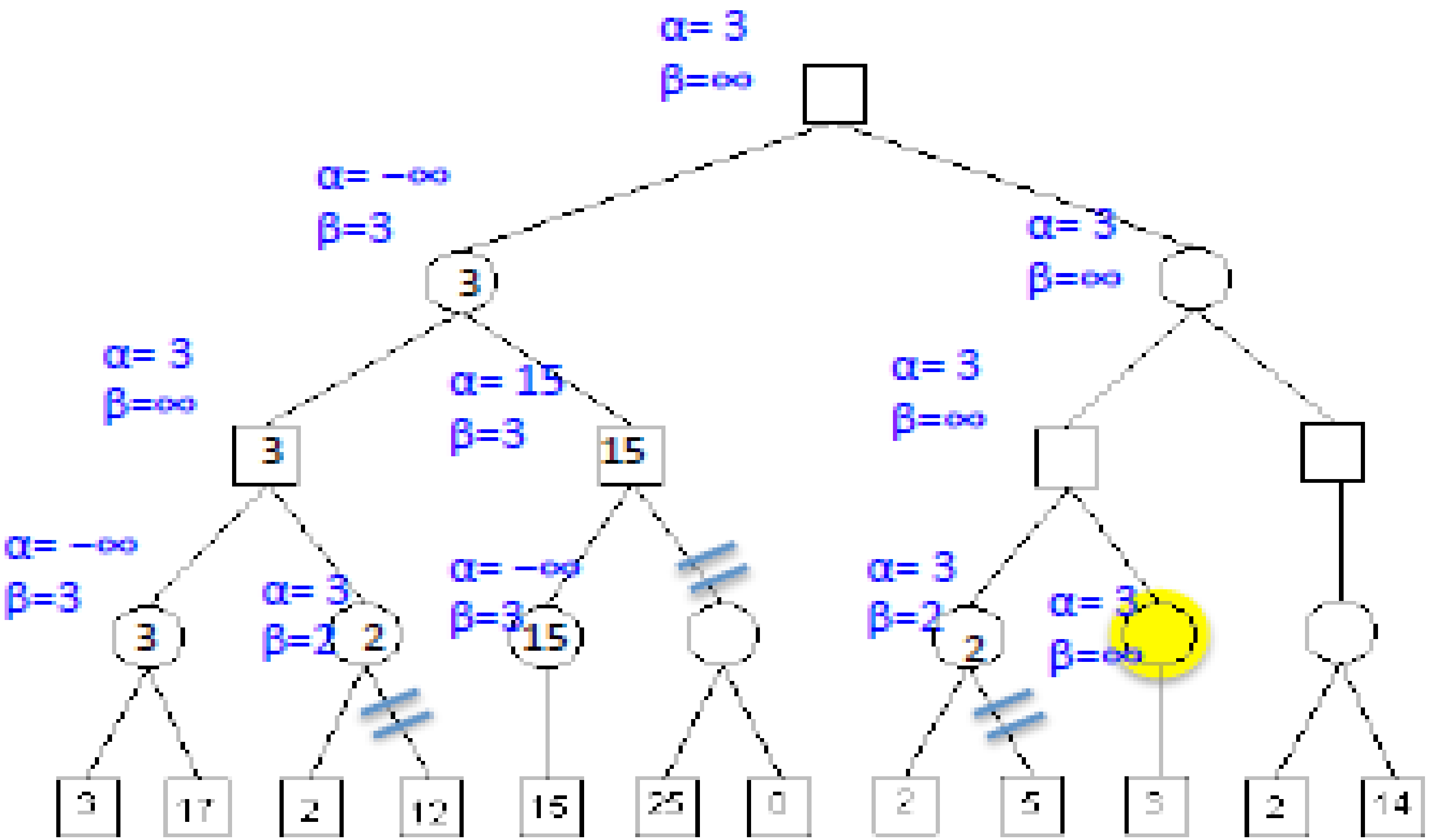


α= 3
β=∞
α= −∞
β=3
α= 3
β=∞
3
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
α= 3
β=2
3
2
15
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

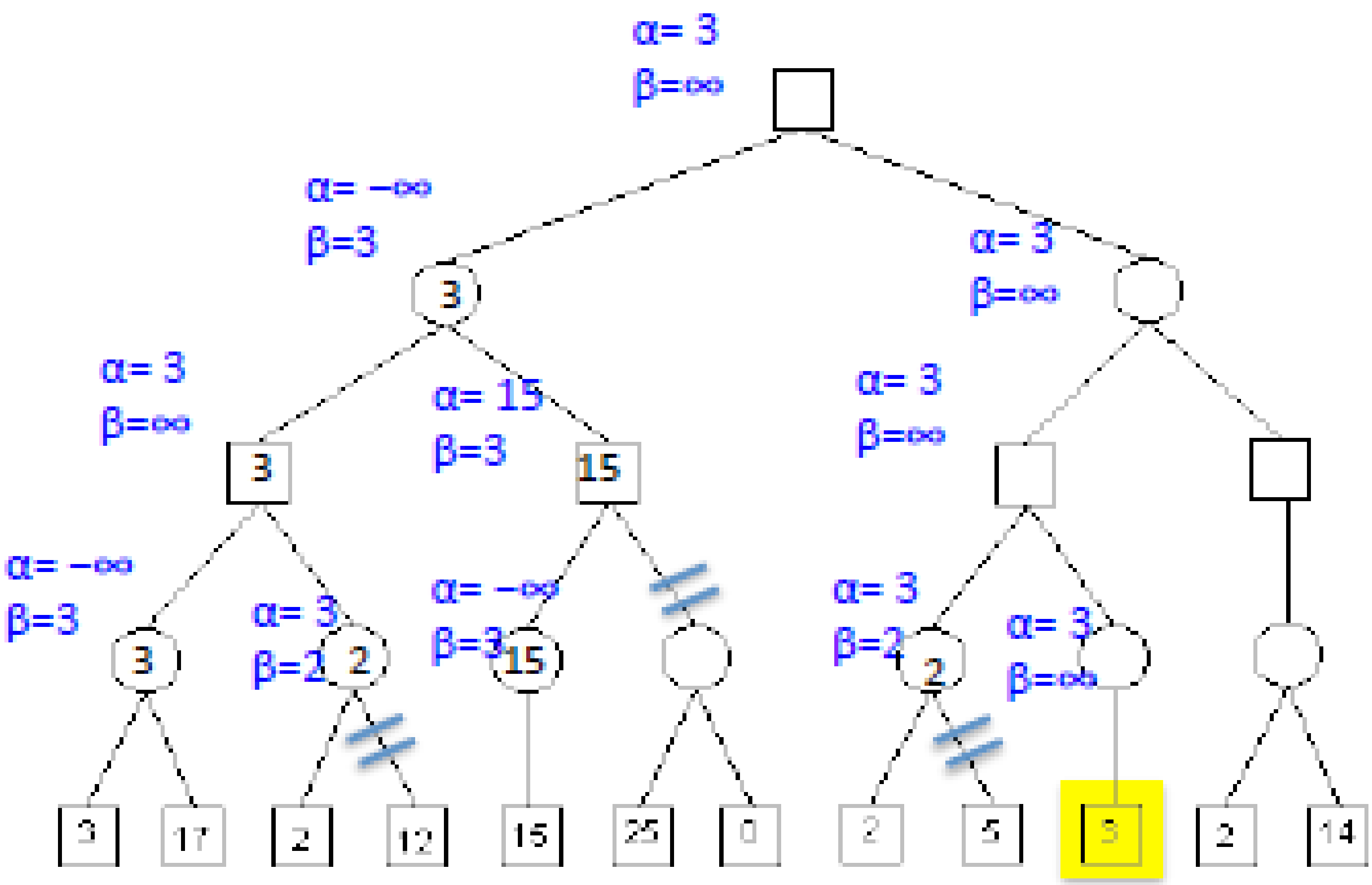

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=2
2
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

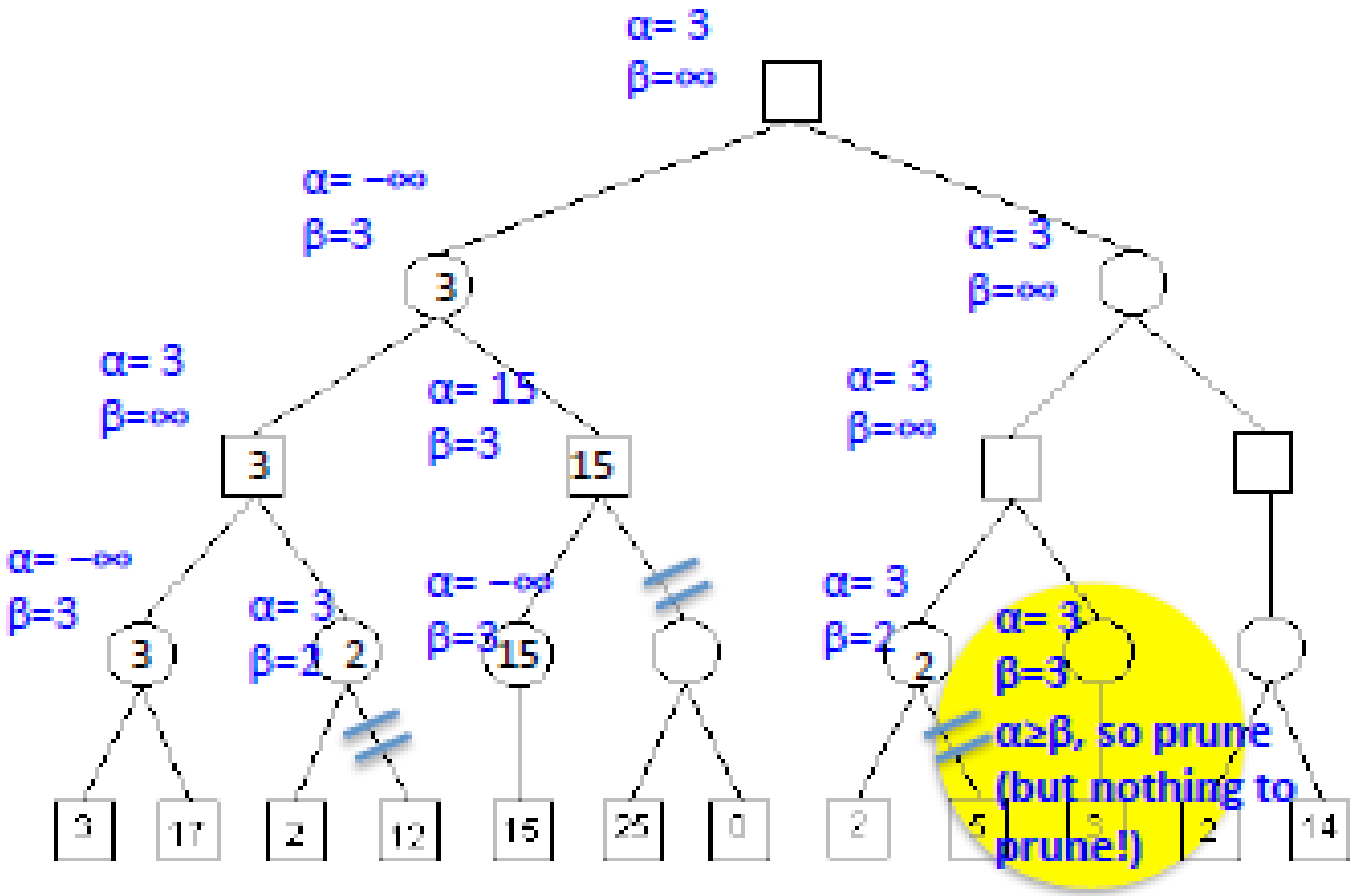


α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=2
2
α= 3
β=∞
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

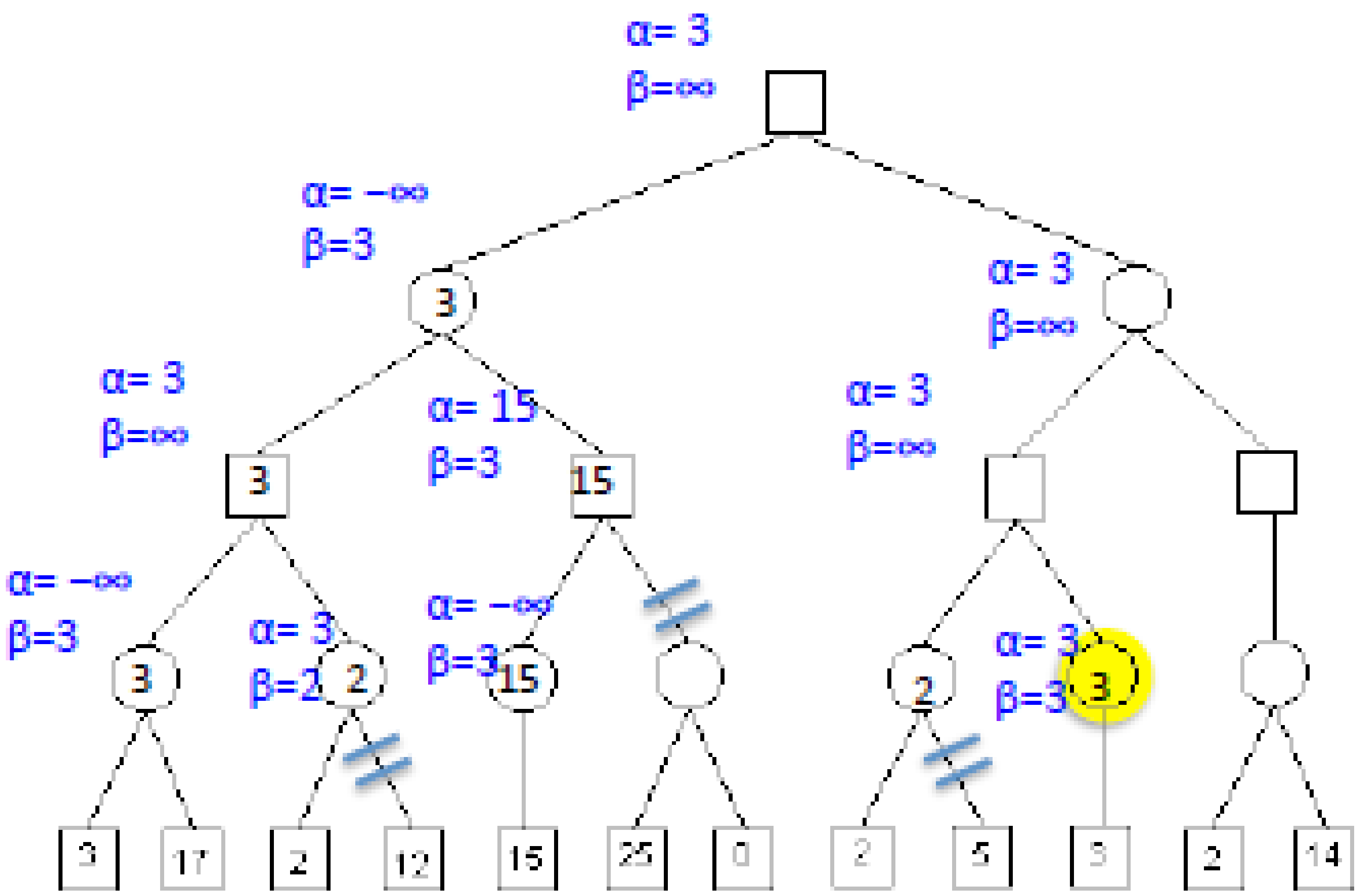

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
α= 3
β=2
α= 3
β=∞
3
2
15
2
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

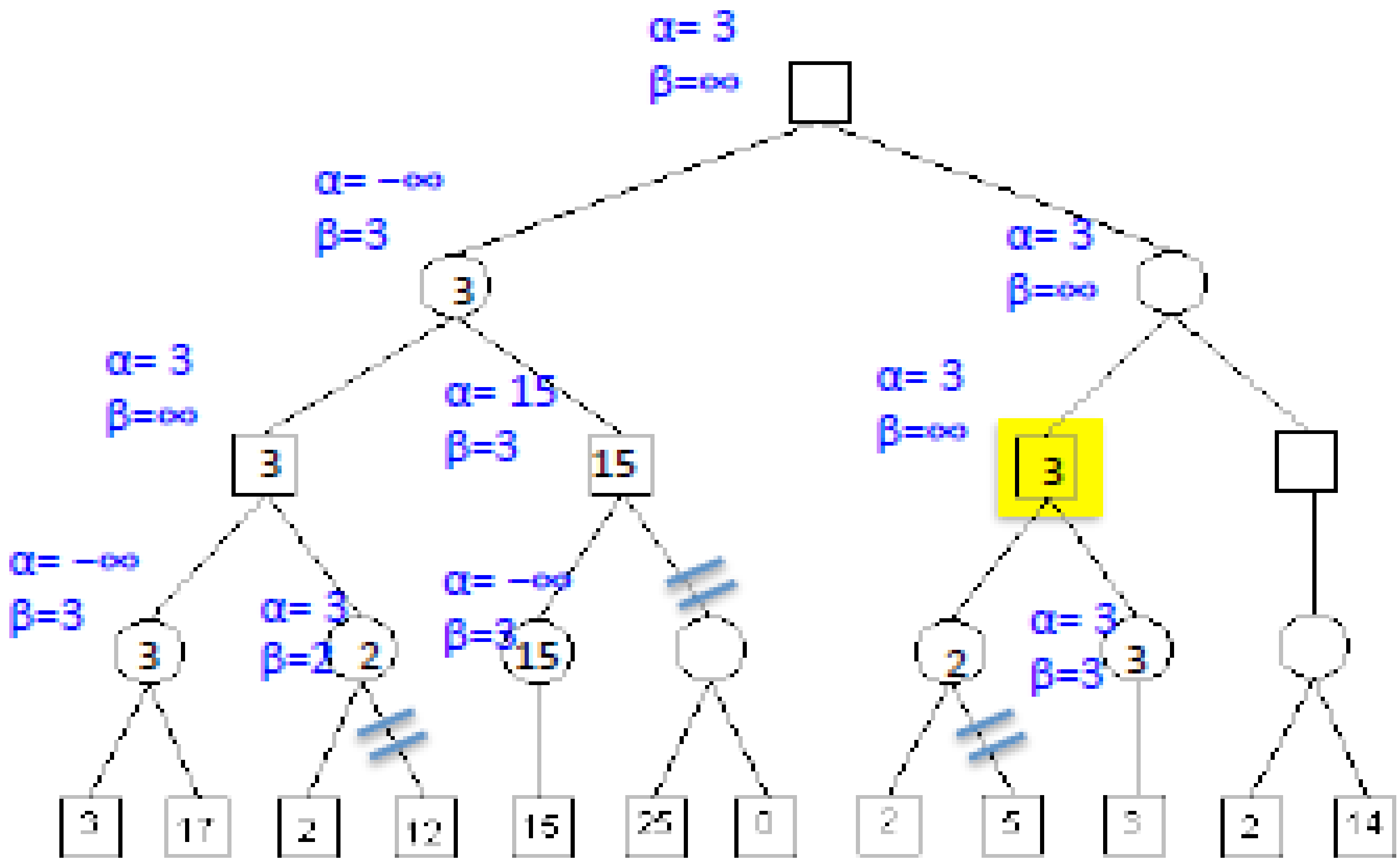


α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=2
2
α= 3
β=3
α≥β, so prune
(but nothing to
prune!)
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

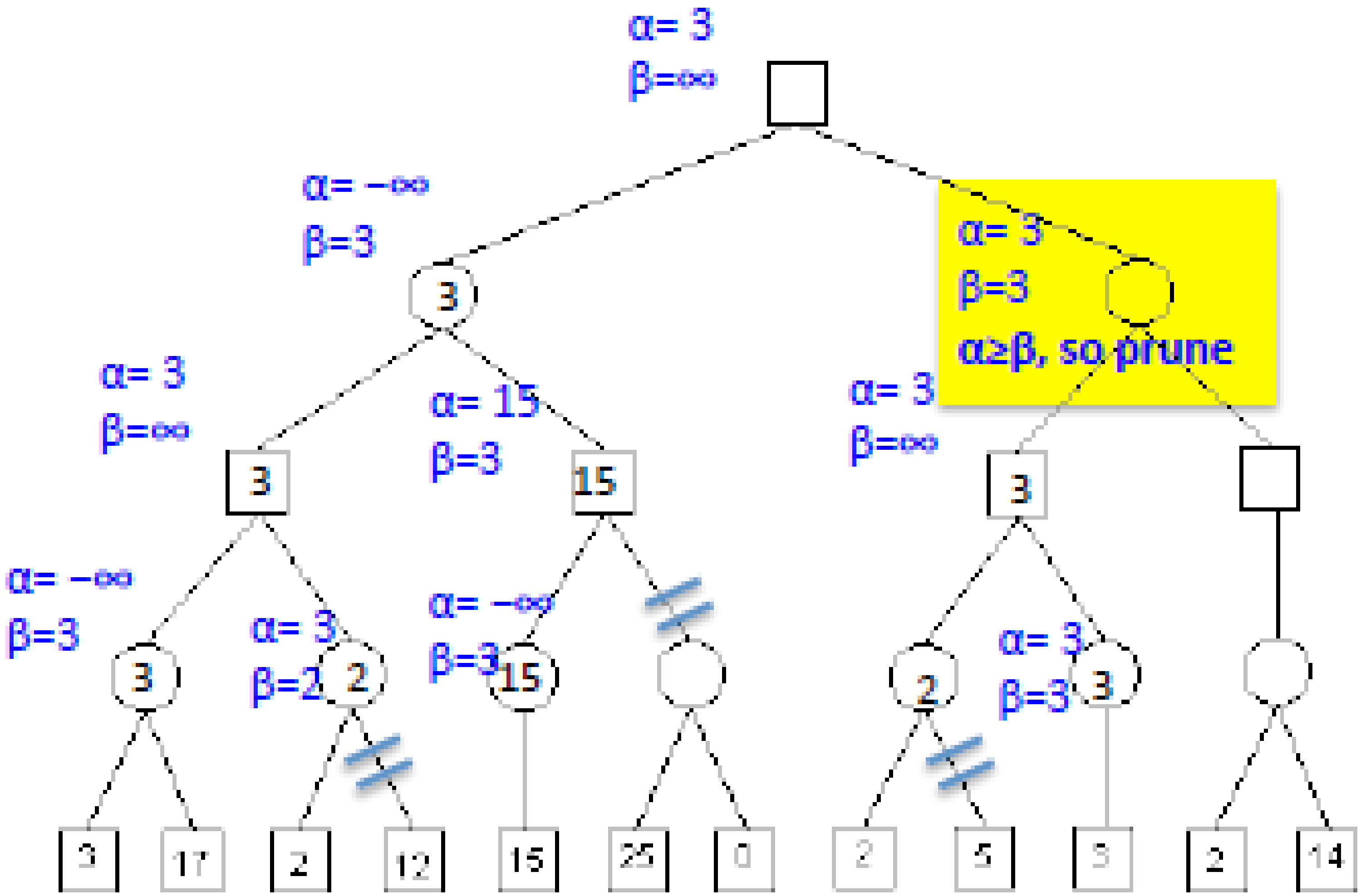

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
2
α= 3
β=3
3
3
17
2
12
15
25
0
2
5
3
2
14

Contd…



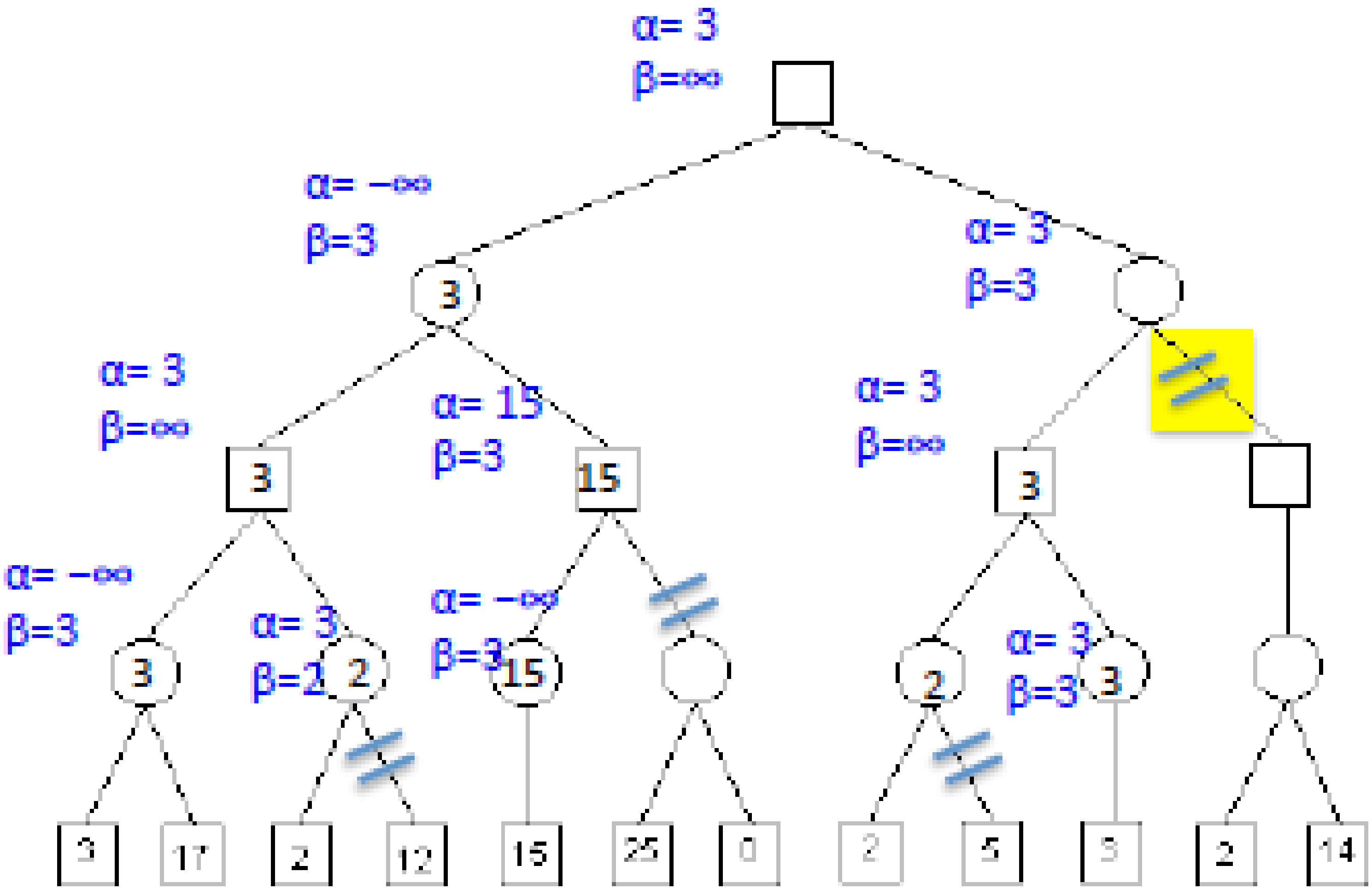
α= 3
β=∞
α= −∞
β=3
α= 3
β=∞
3
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
3
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
α= 3
β=3
3
2
15
2
3
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

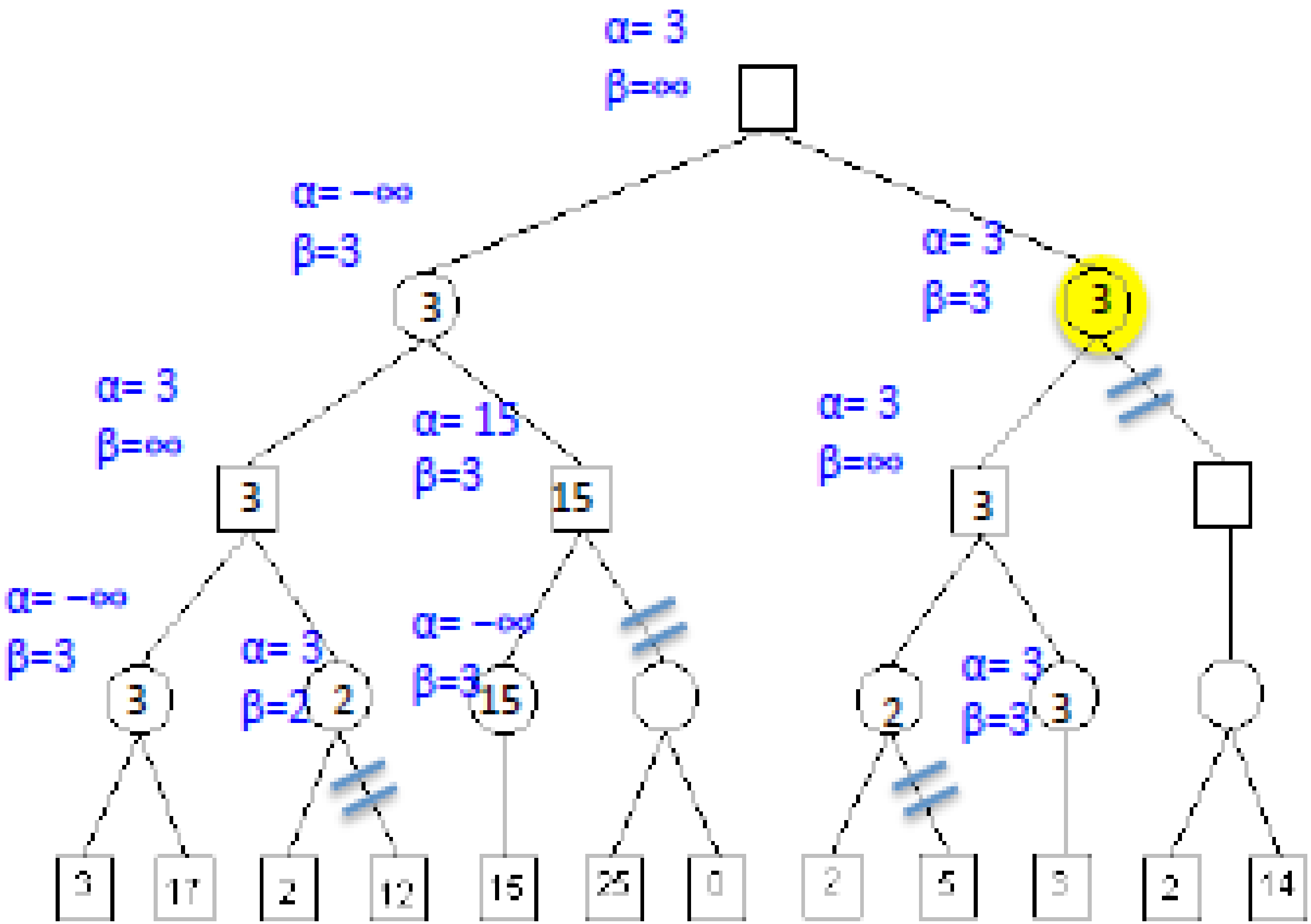


α= 3
β=∞
α= −∞
β=3
3
α= 3
β=3
α≥β, so prune
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
3
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
15
2
α= 3
β=3
3
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

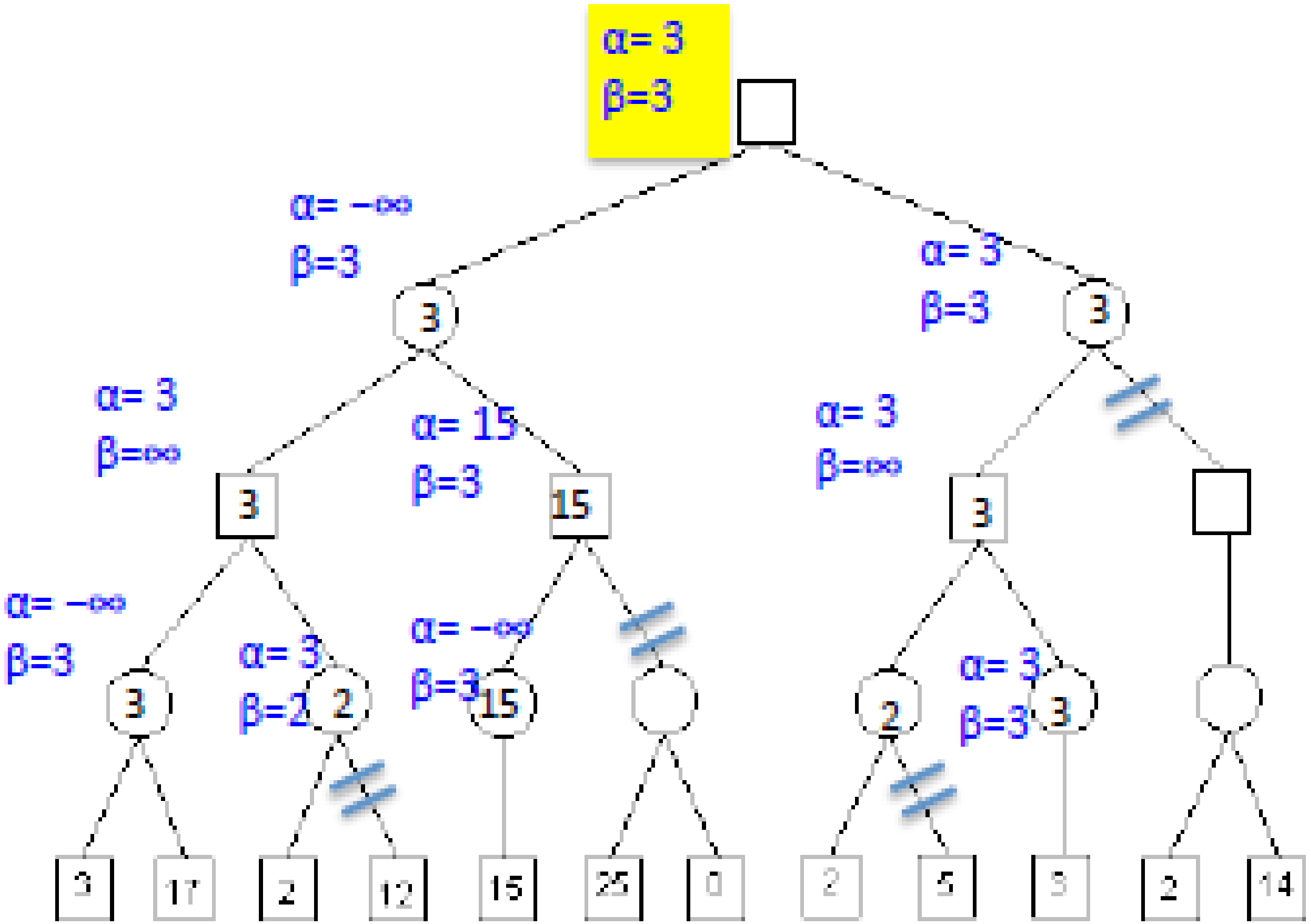

α= 3
β=∞
α= −∞
β=3
α= 3
β=3
α= 3
β=∞
α= 15
β=3
α= 3
β=∞
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
α= 3
β=3
3
3
15
3
3
2
15
2
3
3
17
2
12
15
25
0
2
5
3
2
14

Contd…



α= 3
β=∞
α= −∞
β=3
3
α= 3
β=3
3
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
3
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
15
2
α= 3
β=3
3
3
17
2
12
15
25
0
2
5
3
2
14

Contd…

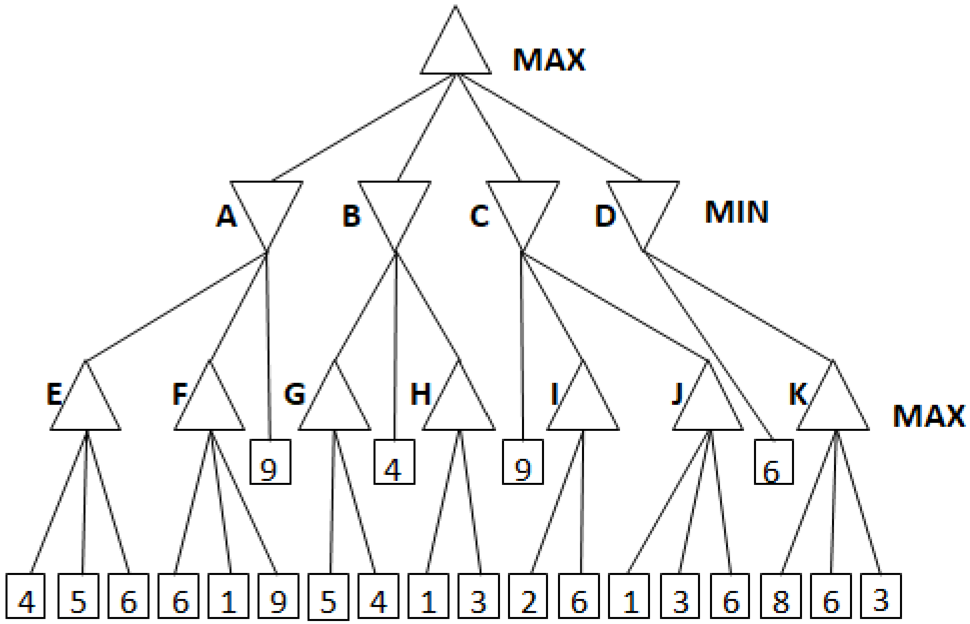

α= 3
β=3
α= −∞
β=3
α= 3
β=3
3
3
α= 3
β=∞
α= 15
β=3
α= 3
β=∞
3
15
3
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
α= 3
β=3
3
2
15
2
3
3
17
2
12
15
25
0
2
5
3
2
14

Homework

- Explain the differences and similarities between depth-first search and breadth-first search. Give examples of the kinds of problems where each would be appropriate.
- Explain what is meant by the following terms in relation to search methods:
  - Complexity, completeness and Optimality
- Provide a definition of the word “heuristic.” In what ways can heuristics be useful in search? Name three ways in which you use heuristics in your everyday life.
- Explain the components of the path evaluation function f(node) used by A*. Do you think it is the best evaluation function that could be used? To what kinds of problems might it be best suited? And to what kinds of problems would it be worst suited?

Contd…

- What is alpha-beta pruning procedure? Solve the following problem by using this procedure



Contd…

- What are the different steps involved in simple problem solving?
- What is the main difference between Uninformed Search and Informed Search strategies?
- What are the advantages and disadvantages of bidirectional search strategy?
- What are the advantages of local search?