Problem Solving Methods

Unit 2

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.
- The 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 start over.
- This over all process is described in the following four steps:
  - Goal Formulation
  - Problem Formulation
  - Search
  - Execute

Contd..

- **Goal Formulation**
  - Intelligent agent maximize their performance measure by adapting a goal and aim at satisfying it.
  - Goal help organize behavior by limiting the objectives that the agent is trying to achieve and hence the actions it needs to consider.
  - Goal are the set of world states in which the goal is satisfied.
  - Therefore, during goal formulation step, specify what are the successful world states.

Contd..

- **Problem Formulation:**
  - Problem formulation is the process of deciding what actions and states to consider, given a goal.
  - Therefore, the agent's task is to find out how to act, now and in the future, so that it reaches a goal state.
  - Before it can do this, it needs to decide what shorts of ations and states it should consider.

Contd..

- **Search a solution**
  - The process of looking for a sequence of actions that reaches the goal is called a searching.
  - Search algorithm takes a problem as a input and return a solution in the form of an sequence.

Contd..

- **Execution**
  - Once a solution is found, the actions it recommends can be carried out this is called the execution phase.
  - Once a solution has been executed, the agent will formulate a new goal.

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 form 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 it 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 whther 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 form the initial state by any sequence of actions is called state space.
- The state space from 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 simpel 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.
  - **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..

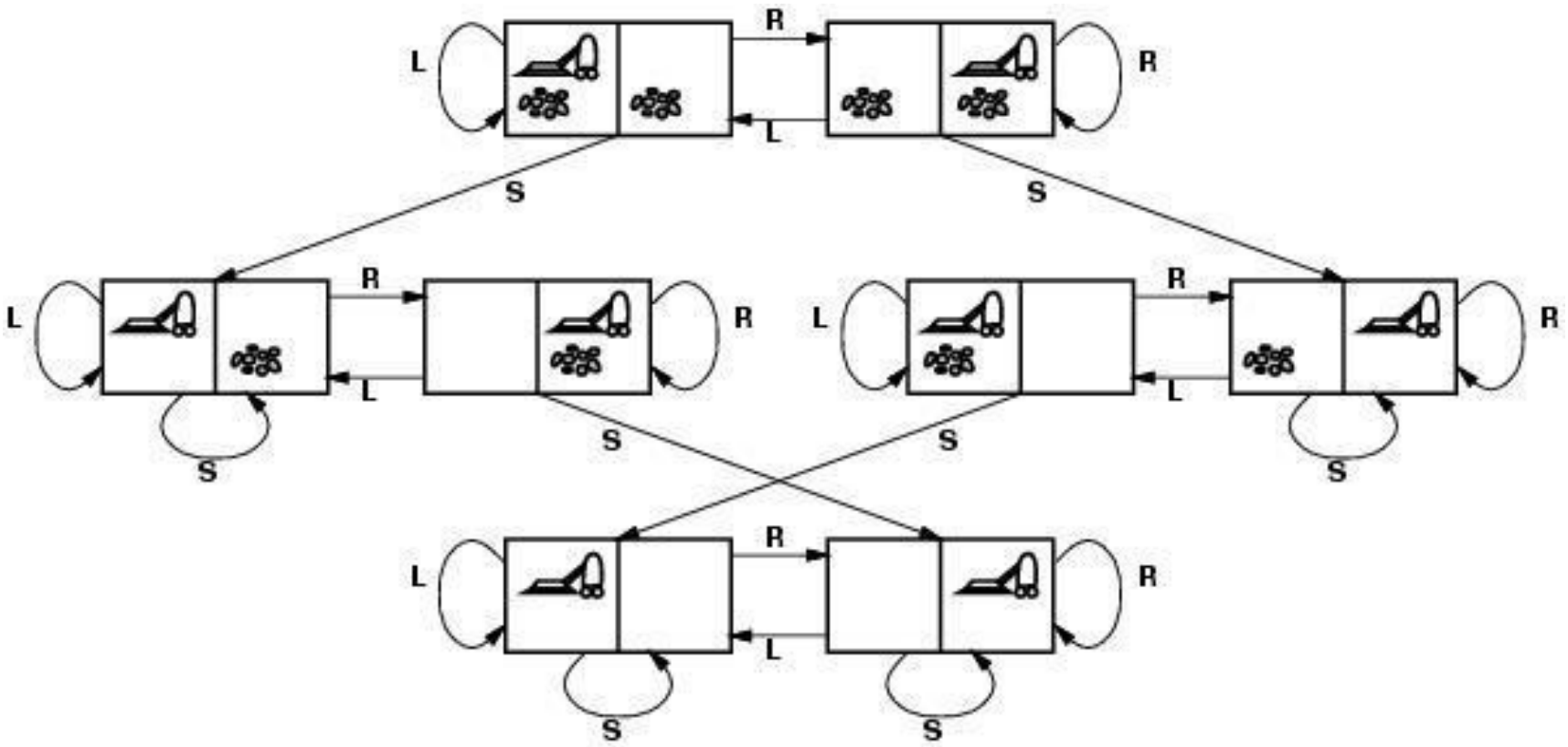


Fig: State space representation for the vacuum world
Here, links denote actions: L=left, R=Right, S=Suck

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 corresponding 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:
    1. Check whether the current state is the goal state or not?
    2. Expand the current state to generate the new set of states.
    3. Choose one of. the new states generated for seach depending upon search strategy.
    4. 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 task underlying AI can be pharsed in term 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 leads to their solutions
  - For the production systems, need to search for a sequence of rule applications that leads to the required fact or action.
  - For neural network system, 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 algorithm can be evaluated in four ways

1. **Completeness:** An algorithm is said to be complete if it definitely finds solution to the problem, if exist.
2. **Time Complexity:** How long does it take to find a solution? Usually measured in terms of the number of nodes expanded during the search.
3. **Space Complexity:** how much space is used by the algorithm? Usually measure in term of the maximum number of nodes in memory at a time.
4. **Optimality:** If a solution is found is it guranteed 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:
  A. Uninformed(or blind) search methods.
  B. Heuristically informed search methods.
- **Uniformed (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.

  Types:

  Breadth First search (BFS)
    - Variation: Uniform cost search

  Depth First search (DFS)
    - Variations: Depth limit search, Iterative deepending DFS, Bidirectional search

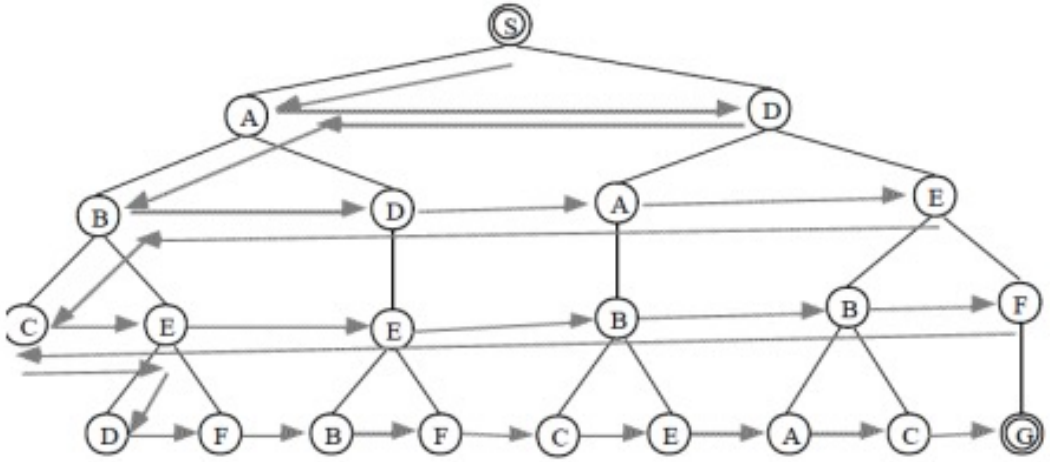
Contd..

- **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 case of the heuristically informed search methods, one uses domain-dependent information in order to search the space more efficiently.
  - Example
    - Greedy best-first search
    - A* search

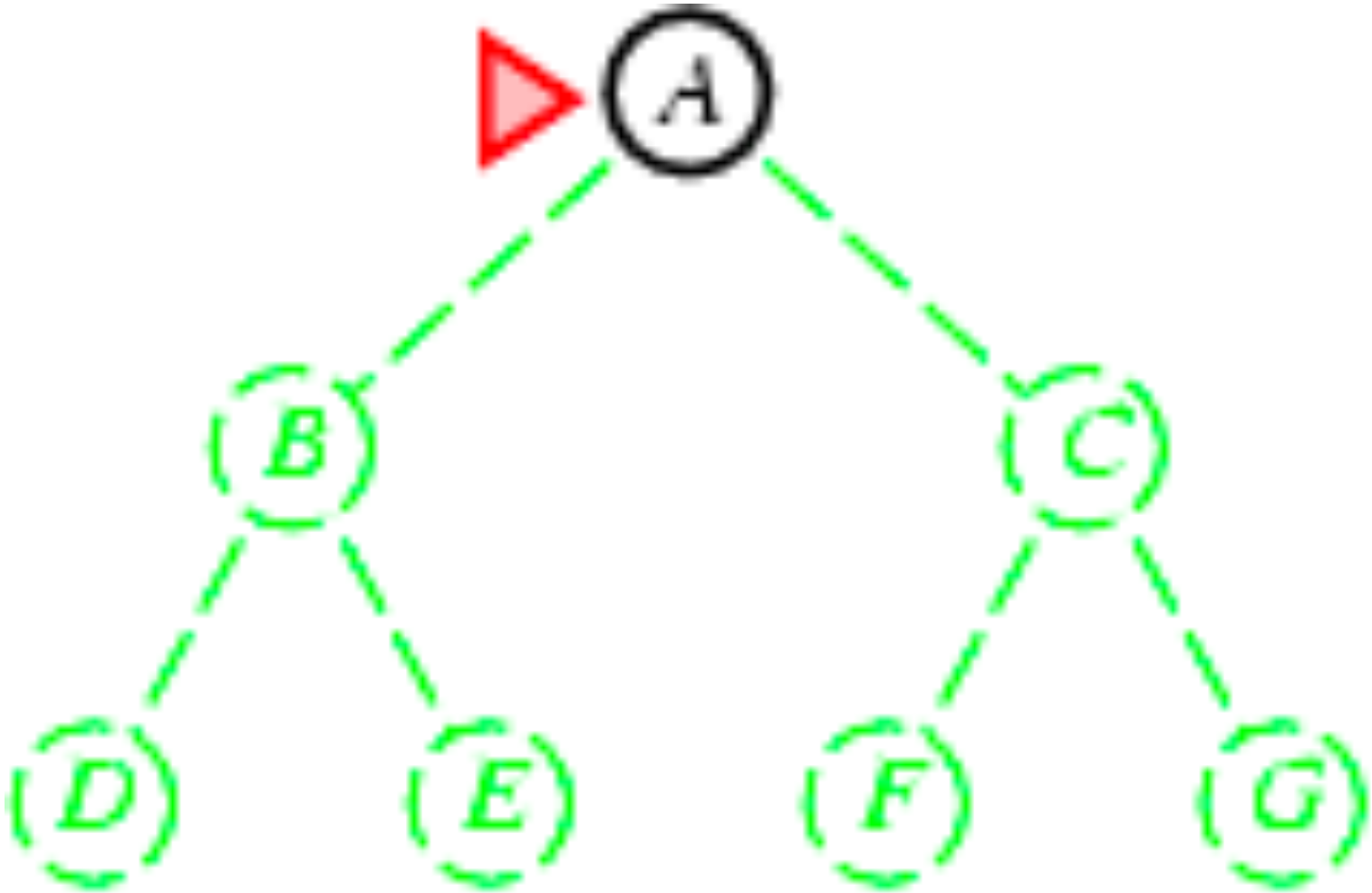
Breadth First Search

- ✓ 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 level in the search tree before any nodes at the next level are expanded until the goal reached.

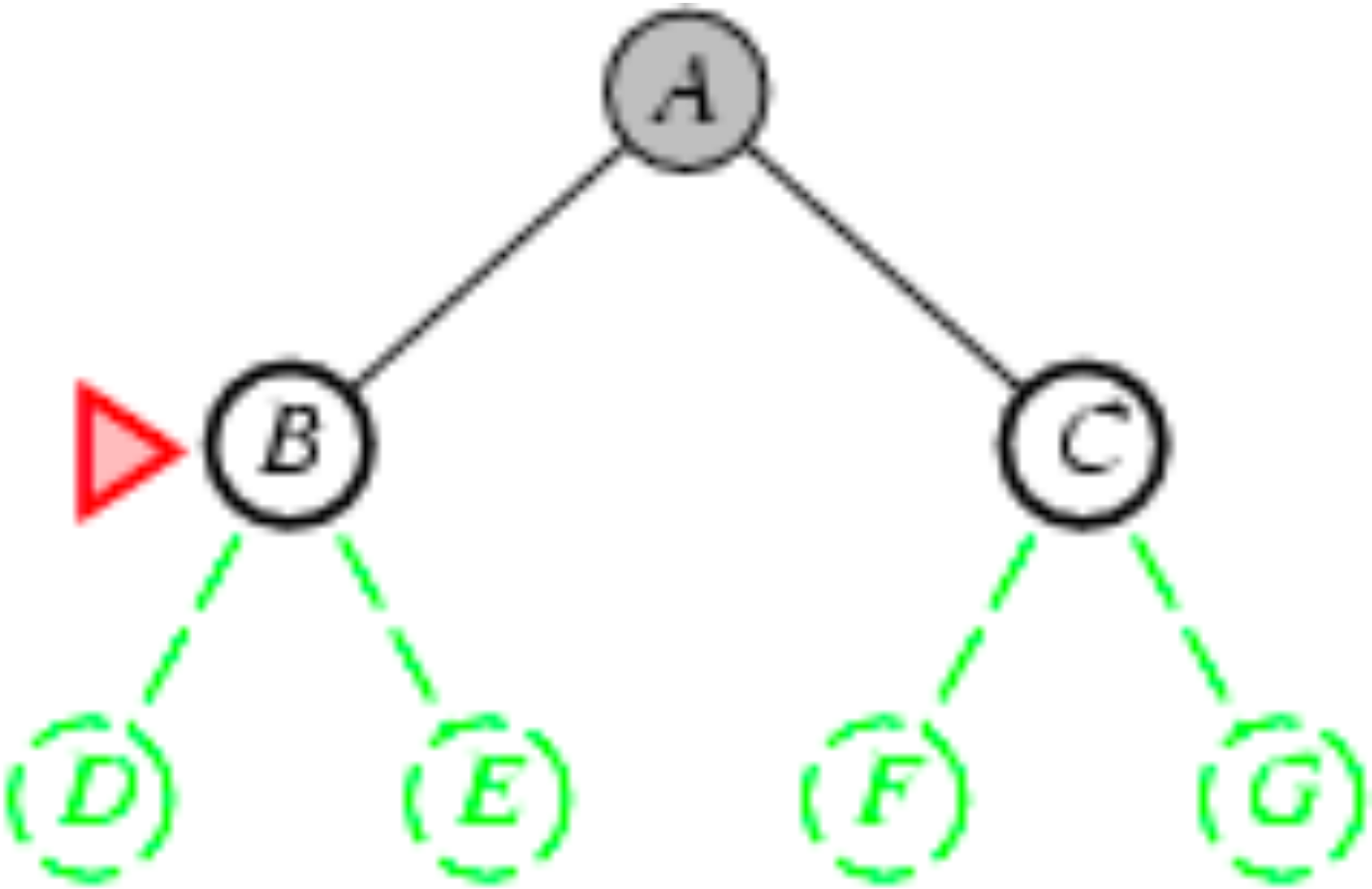
i.e., Expand shallowest unexpanded node. (Queue is used)



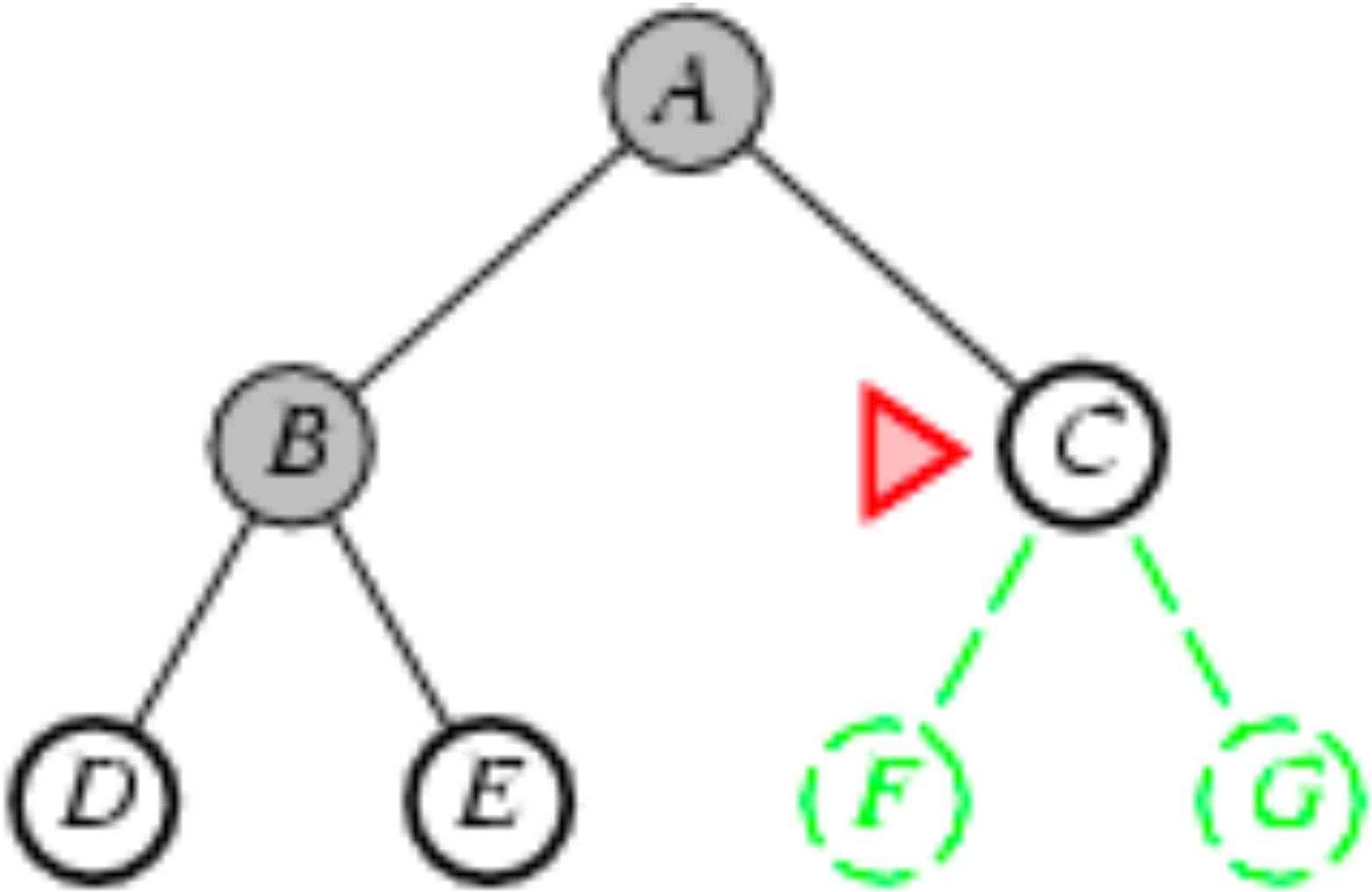
Contd..



Contd..



Contd..



Contd..

Completeness: Does it always find a solution if one exists?

YES

If shallowest goal node is at some finite depth d and If b is finite

Time complexity:

Assume a state space where every state has b successors, root has b successors, each node at the next level has again b successors (total b^2), …

Assume solution is at depth d

Worst case; expand all except the last node at depth d

Total no. of nodes generated: $b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots b^d + (b^{d+1} - b) = \mathbf{O(b^{d+1})}$

Space complexity: Each node that is generated must remain in memory

Total no. of nodes in memory: $1 + b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots b^d + (b^{d+1} - b) = \mathbf{O(b^{d+1})}$

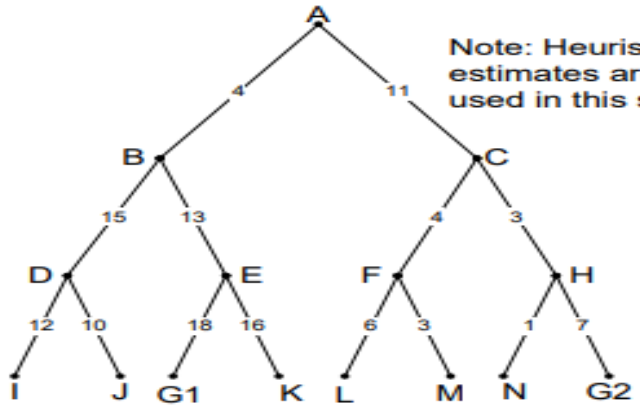
Optimal (i.e., admissible): if all paths have the same cost.

Otherwise, not optimal but finds solution with shortest path length (shallowest solution).

If each path does not have same path cost shallowest solution may not be optimal

Uniform Cost Search

- The search begin at root node. The search continue 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 uninform cost search keeps going, choosing a node (with less total cost from the root node to the 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.

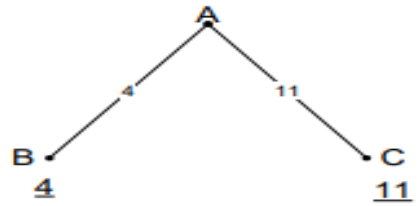


Note: Heurisitic estimates are not used in this search!

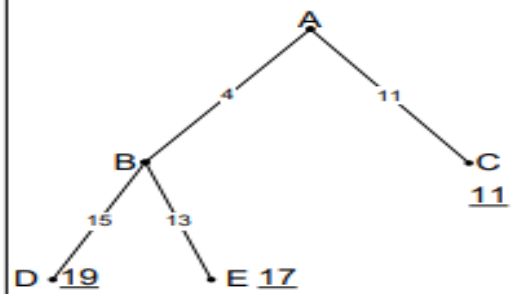
- Start with root node A.

A
0

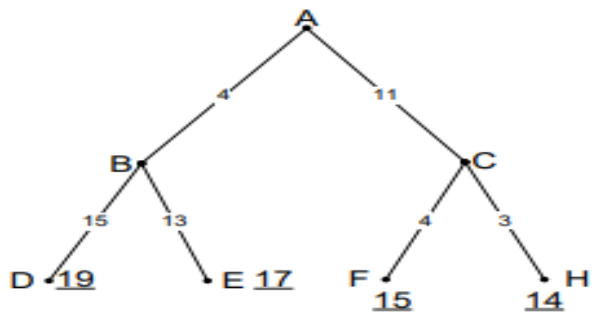
Paths from root are generated.



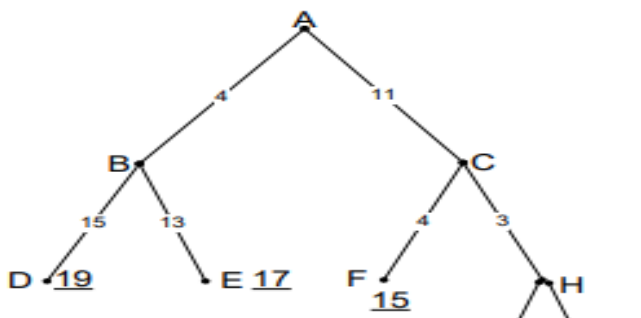
Since B has the least cost, we expand it.



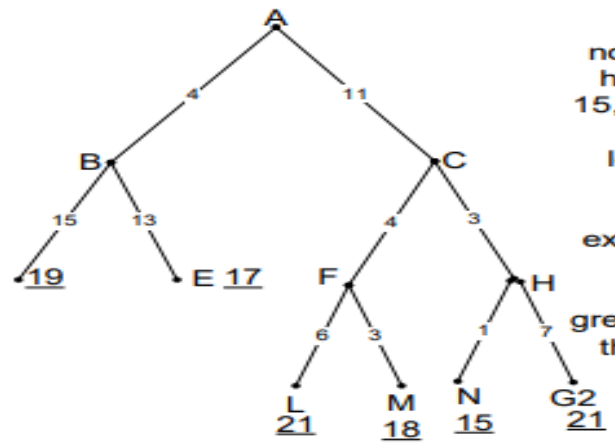
Of our 3 choices, C has the least cost so we'll expand it.



Node H has the least cost thus far, so we expand it.



We have a goal, G2 but need to expand other branches to see if there is another goal with less distance.

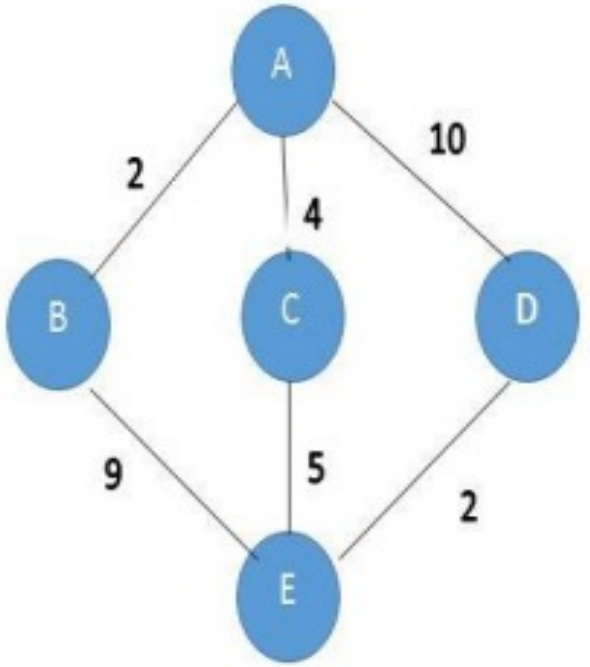


Note: Both nodes F and N have a cost of 15, we chose to expand the leftmost node first. We continue expanding until all remaining paths are greater than 21, the cost of G2

Uniform Cost Search

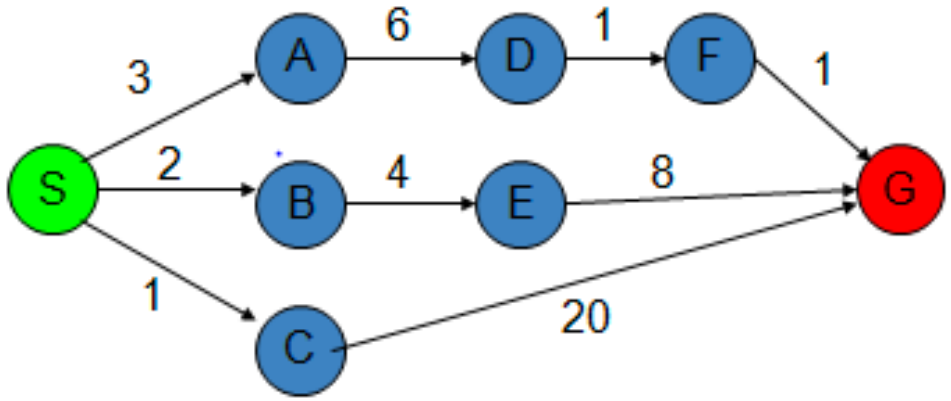
Find path from A to E

- ✓ 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



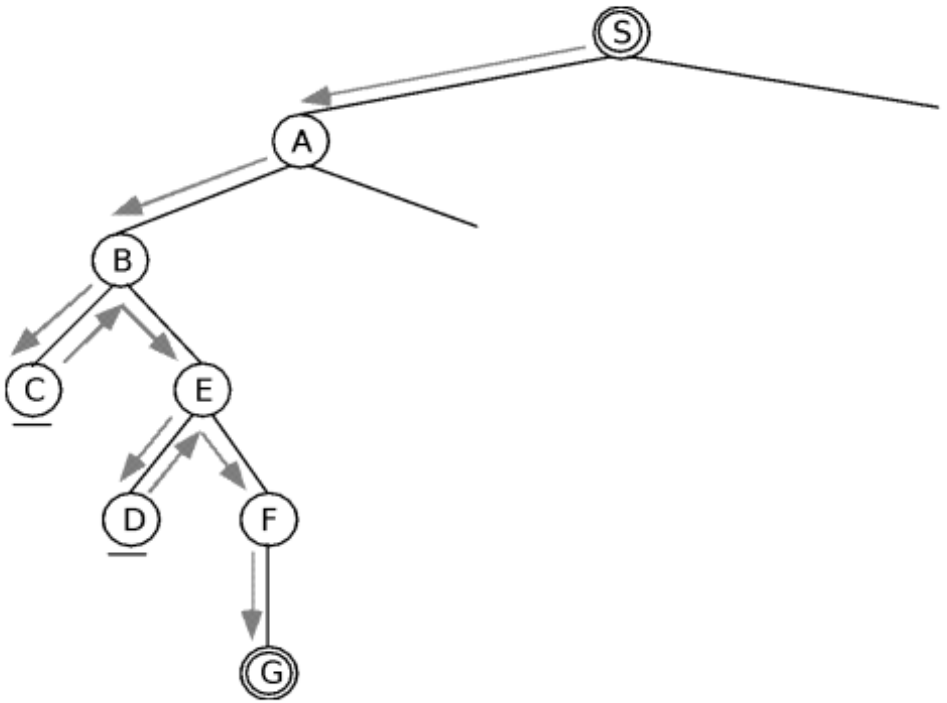
Class Work

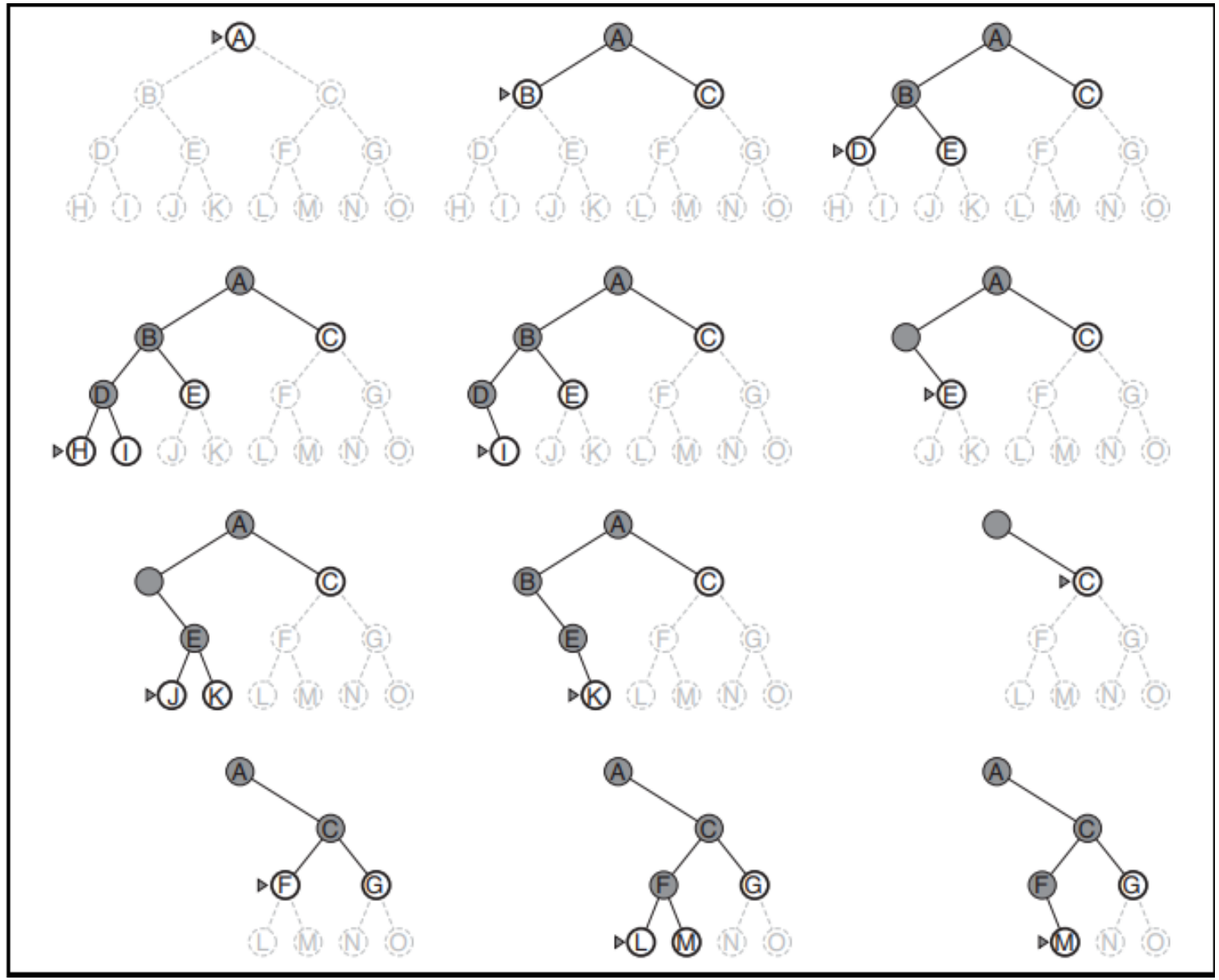
- The graph below shows the step-costs for different paths going from the start (S) to the goal (G).
- Use uniform cost search to find the optimal path to the goal.



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.).




A
B C
D E F G
H I J K L M N O
A
B C
D E F G
H I J K L M N O
A
B C
D E F G
H I J K L M N O
A
B C
D E F G
H I J K L M N O
A
B C
D E F G
I J K L M N O
A
C
E F G
J K L M N O
A
C
E F G
J K L M N O
A
B C
E F G
K L M N O
C
F G
L M N O
A
C
F G
L M N O
A
C
F G
L M N O
A
C
F G
M N O

DFS Evaluation

Completeness

- Does it always find a solution if one exists?
- **NO,** If search space is infinite and search space contains loops then DFS may not find solution.

Time complexity

- Let, m is the maximum depth of the search tree. In the worst case Solution may exist at depth m, root has b successors, each node at the next level has again b successors (total b^2), …
- Worst case; expand all except the last node at depth m
- Total no. of nodes generated: $b + b^2 + b^3 + \ldots\ldots\ldots\ldots\ldots\ldots b^m = \mathbf{O(b^m)}$

Space Complexity:

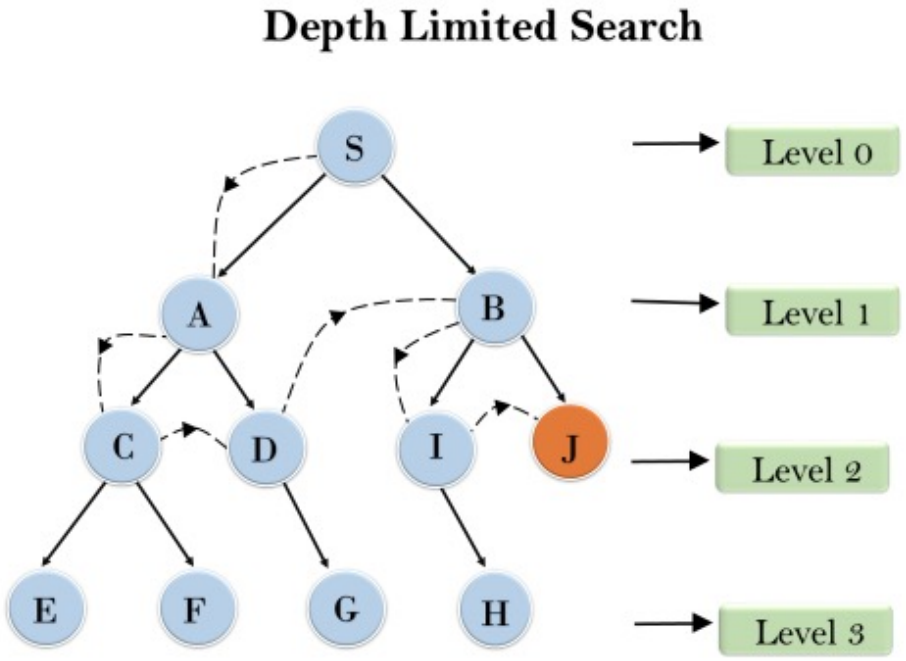
- It needs to store only a single path from the root node to a leaf node, along with remaining unexpanded sibling nodes for each node on the path.
- Total no. of nodes in memory: 1+ b + b + b + …….. b m times = **O(bm)**

Optimal (i.e admissible)

- DFS expand deepest node first, if expands entire let sub-tree even if right sub-tree contains goal nodes at levels 2 or 3.
- Thus we can say DFS may not always give optimal solution.

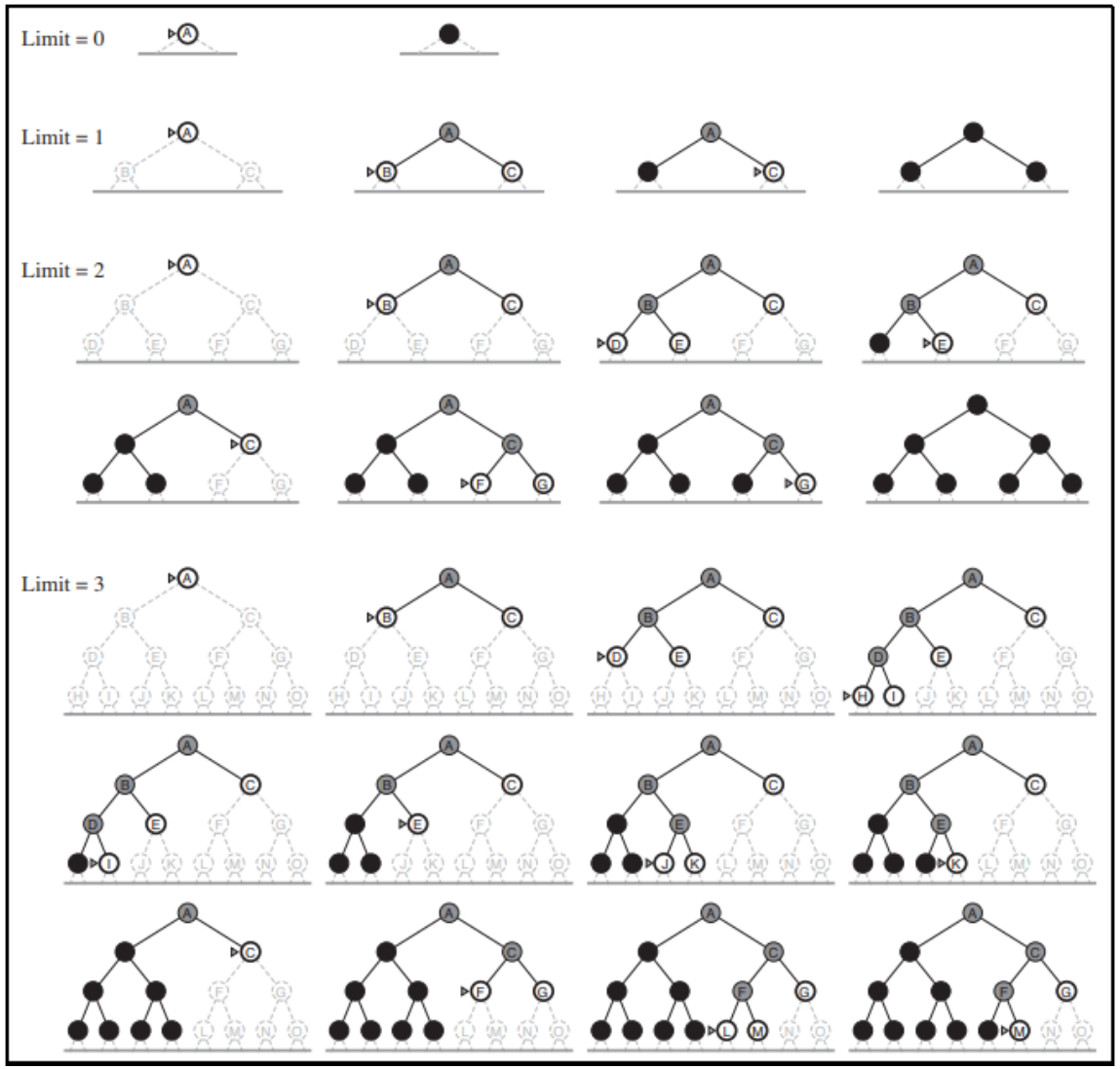
Depth Limited Search

- ✓ Variations of DFS
- ✓ It works exactly like depth-first search, but avoids its drawbacks regarding completeness by imposing a maximum limit on the depth of the search.
- ✓ It will not follow infinitely deep paths or get stuck in cycles.
- ✓ **This** will find a solution if it is within the depth limit, which guarantees at least completeness on all graphs.
- ✓ It solves the infinite-path problem of DFS. 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.
- ✓ **Completeness:** If l < d then incompleteness results.
- ✓ **Optimal:** If l > d then not optimal.
- ✓ **Time complexity:** O(b^l)
- ✓ **Space complexity:** O (bl)



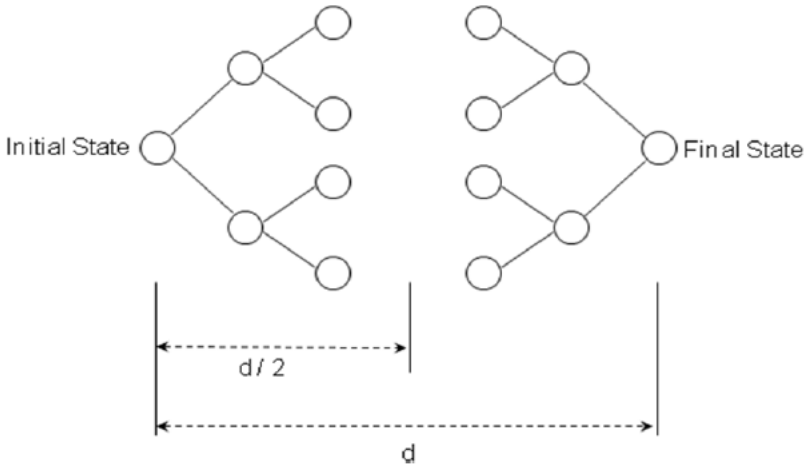
- ✓ In this strategy, depth-limited search is run repeatedly, increasing the depth limit with each iteration until it reaches d, the depth of the shallowest goal state.
- ✓ It does this by gradually increasing the limit—first 0, then 1, then 2, and so on—until a goal is found
- ✓ On each iteration, IDDFS visits the nodes in the search tree in the same order as depth first search, but the cumulative order in which nodes are first visited, is effectively breadth-first.
- ✓ IDDFS combines depth-first search's space-efficiency and breadth-first search's completeness (when the branching factor is finite). It is optimal when the path cost is a non-decreasing function of the depth of the node.
- ✓ Iterative deepening is depth-first search to a fixed depth in the tree being searched. If no solution is found up to this depth then the depth to be searched is increased and the whole `bounded' depth-first search begun again.

Iterative Deepening Depth-First Search



Bidirectional Search

- The idea behind bidirectional search is to run two simultaneous searches—one forward from the initial state and the other backward from the goal—hoping that the two searches meet in the middle.
- It then, expands nodes from the start and goal state simultaneously.
- It checks at each stage if the nodes of one have been generated by the other, i.e, they meet in the middle.



- **Completeness:** yes
- **Optimality:** yes (If done with correct strategy- e.g. breadth first)
- **Time complexity:** $O(b^{d/2})$
- **Space complexity:** $O(b^{d/2})$

Contd

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

Heuristic Search

- Heuristic search uses domain-dependent(heuristic) information beyond the definition of the problem itself in order to search the space mode efficiently.
- Ways of using heuristic information:
  - Decide 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 successor 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..

- Infromed search define 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

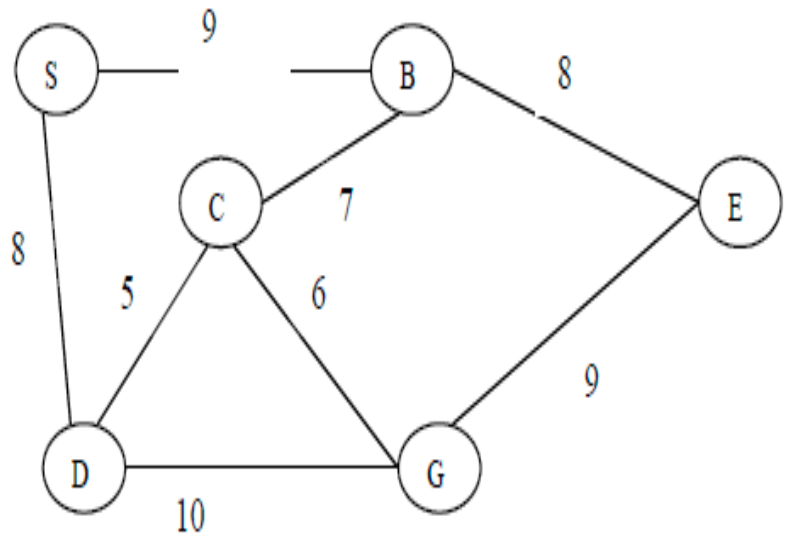
Contd..
Greedy Best First Search

- 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: g(root)=0 and h(goal)=0

Contd..

Example to illustrate greedy best-first search:

- For example consider the following graph.



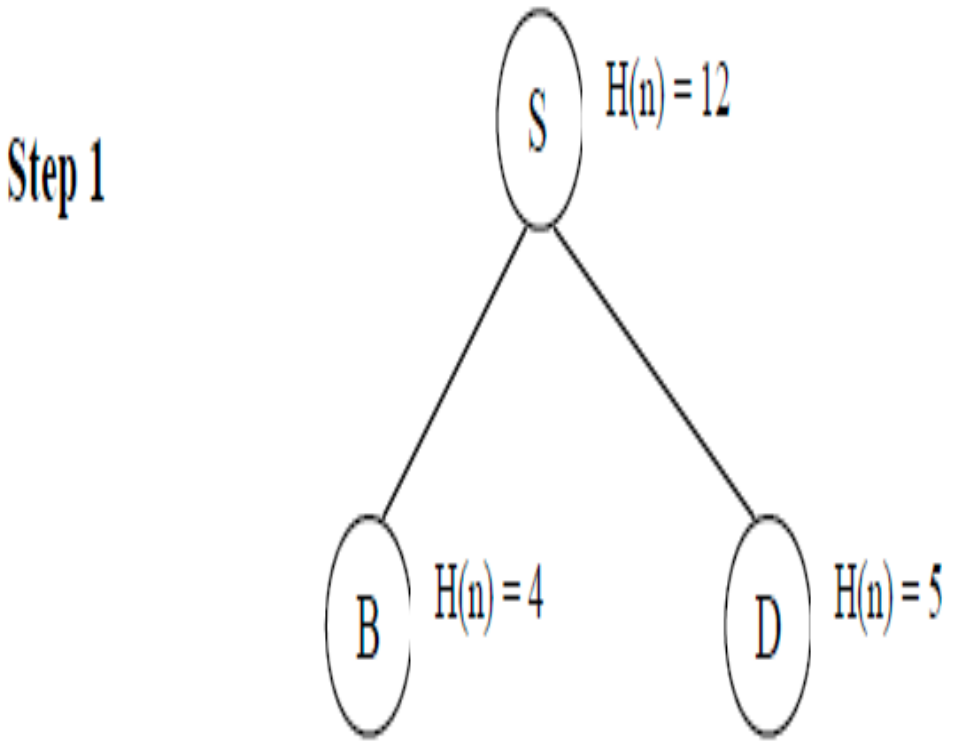
- Straight line distance to node G(goal node) from other nodes is given below:

S→ G = 12
B→ G = 4
E→ G = 7
D→ G = 5
C→ G = 3

- Let H(n)=straight line distance
- Now greedy search operation is done as below:

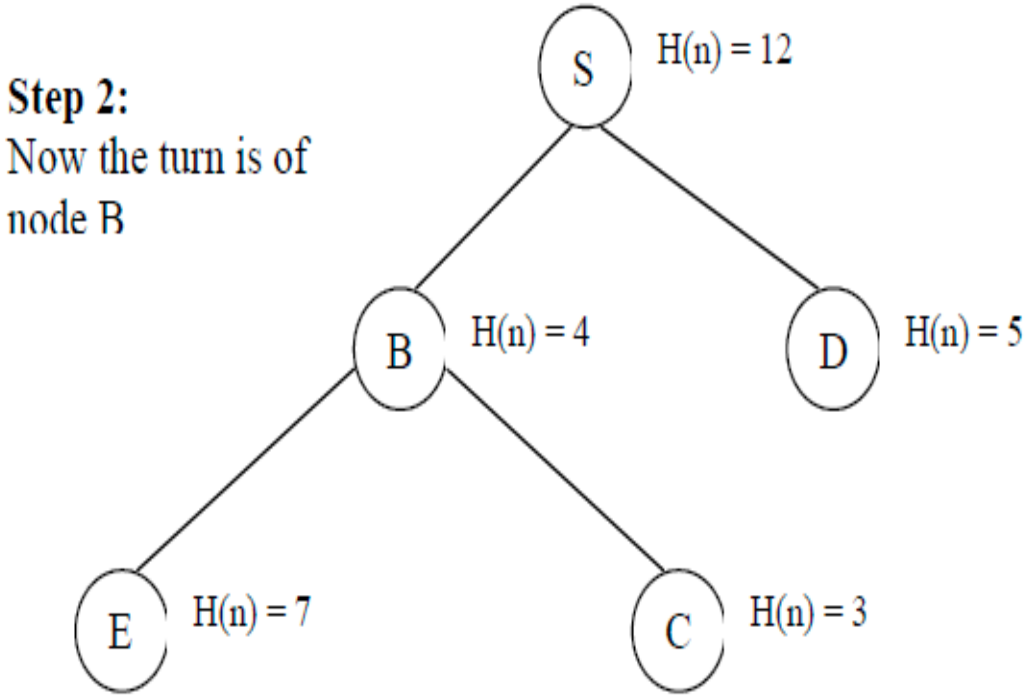
Contd..

- Start node “S” the start state
- Childern of S={B(4), D(5)}
- Therefore, best =B



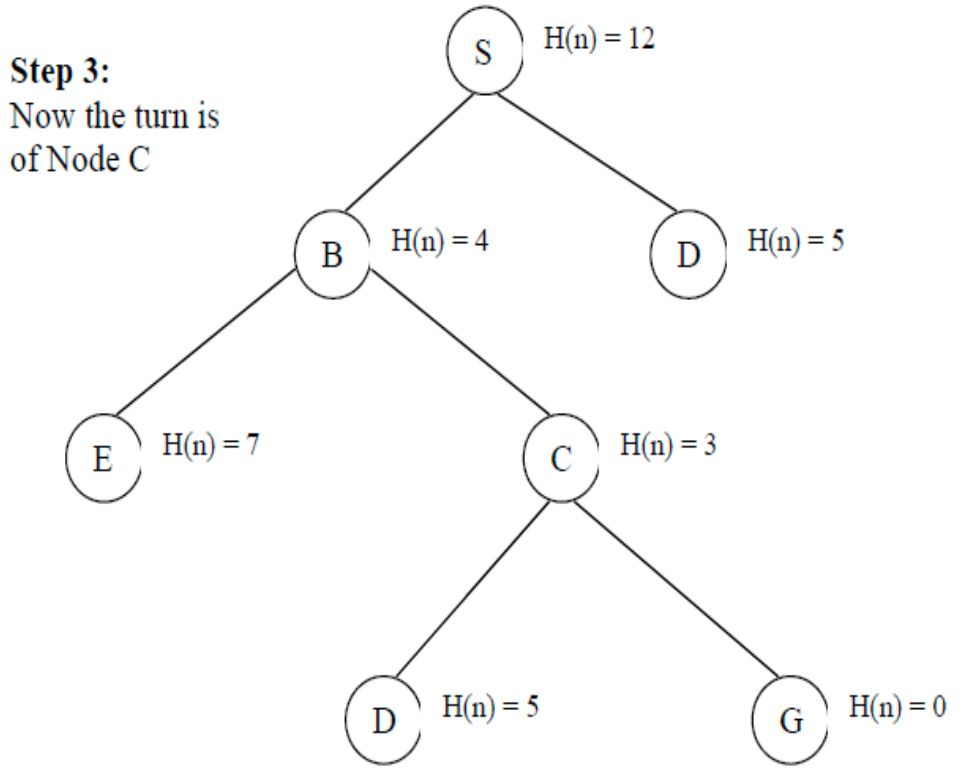
Contd..

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



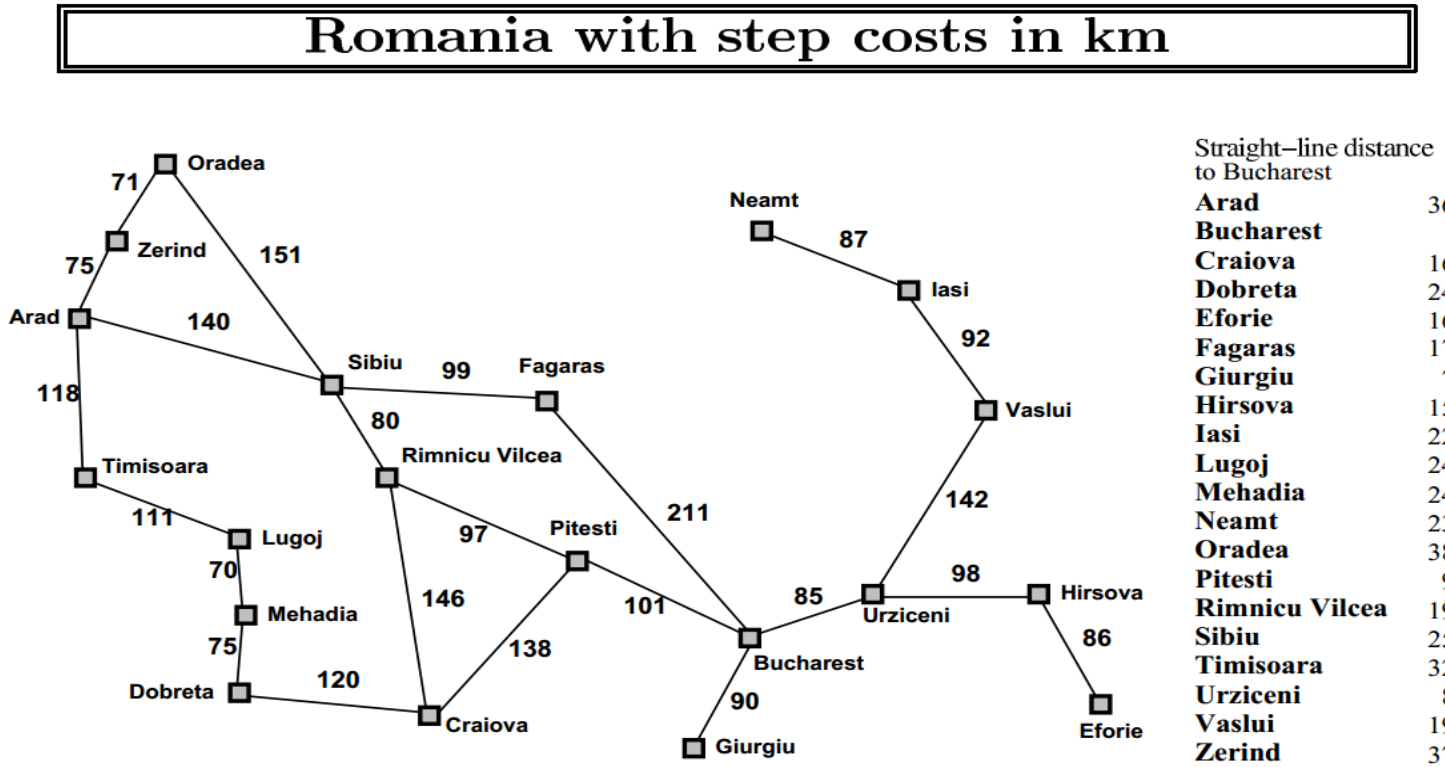
Contd..

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



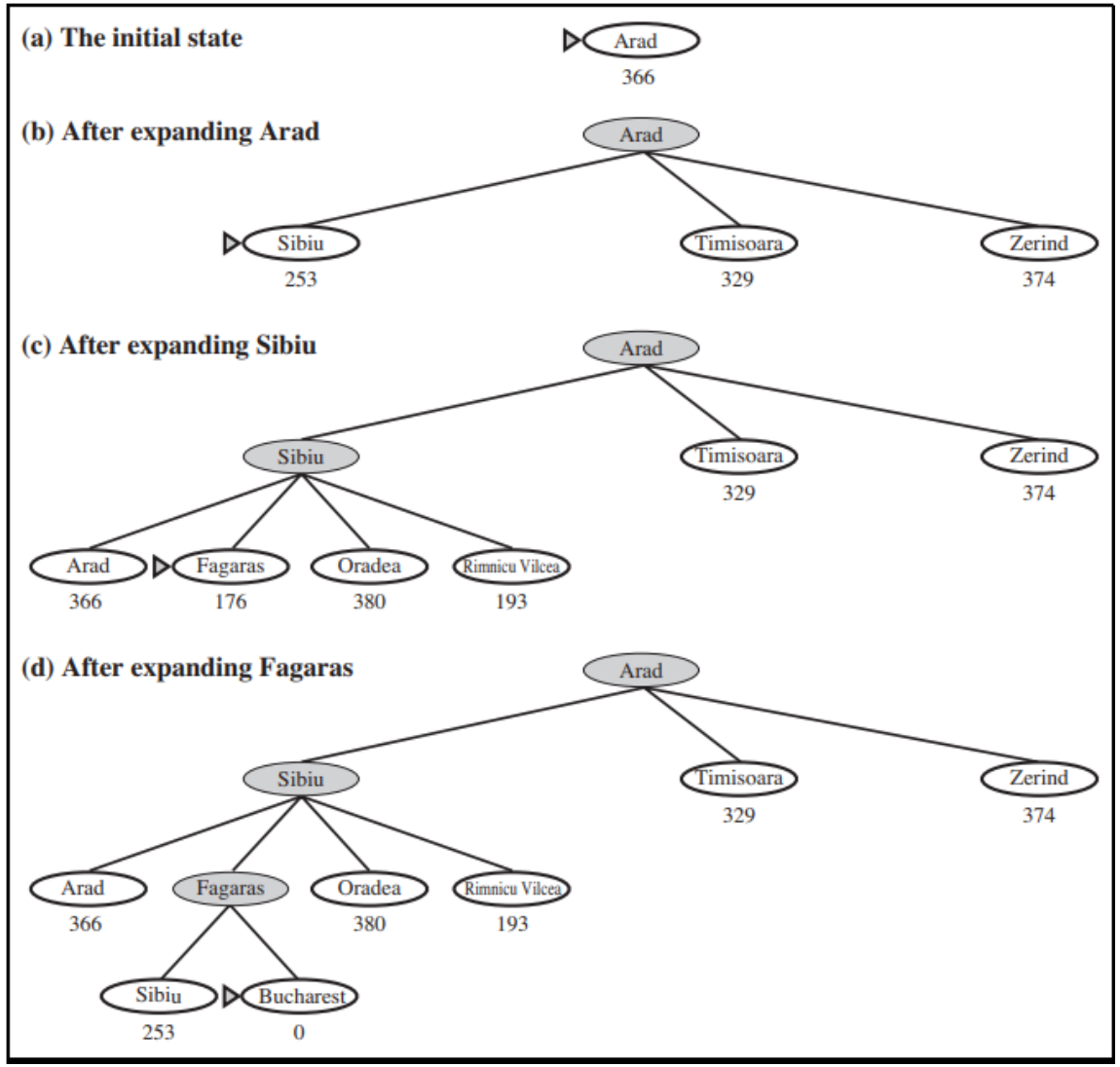
Contd..

- Example: Given following graph of cities, starting at Arad city, problem is to reach to the Bucharest



Straight–line distance to Bucharest	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Contd..


(a) The initial state
Arad
366
(b) After expanding Arad
Arad
Sibiu
253
Timisoara
329
Zerind
374
(c) After expanding Sibiu
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
176
Oradea
380
Rimnicu Vilcea
193
(d) After expanding Fagaras
Arad
Sibiu
Timisoara
329
Zerind
374
Arad
366
Fagaras
Oradea
380
Rimnicu Vilcea
193
Sibiu
253
Bucharest
0

Contd..

A* Search

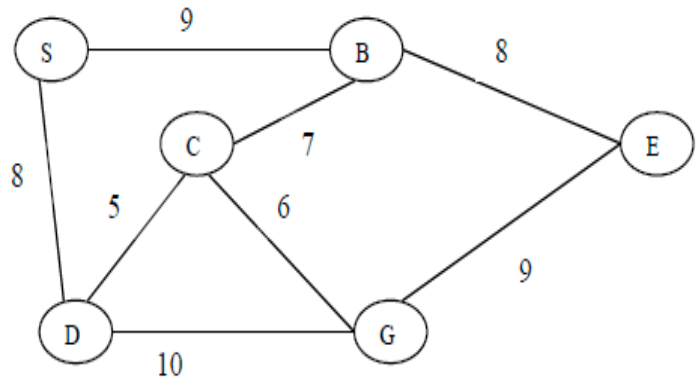
- A* is a best first, informed search algorithm. The search begins at root node.
- The search continues by visiting the next node which has the least evaluation.
- It evaluats nodes by using the following evaluation function.
- Node are visited in the following manner until a goal is reached.

- 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.

Contd..

Example to illustrate A* search:

- For example consider the following graph



- Straight line distance to node G(goal node) from other nodes is given below:

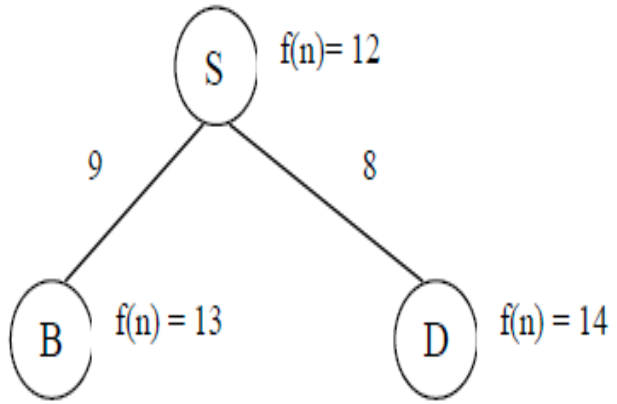
$S \rightarrow G = 12$
$B \rightarrow G = 4$
$E \rightarrow G = 7$
$D \rightarrow G = 6$
$C \rightarrow G = 3$

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

Contd..

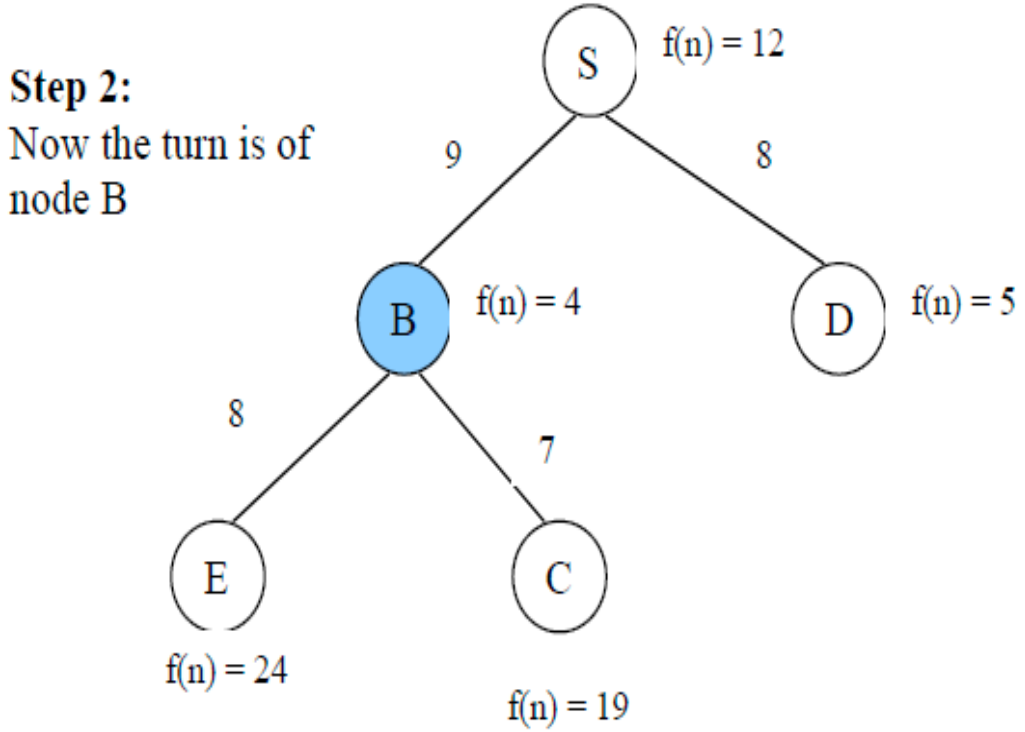
- A* Algorithm start s, the start state.
- Childern of S={B,D}
- Now evaluate function for each child of S is
  - F(B)=g(B)+h(B)=9+4=13
  - F(D)=g(D)+h(D)=8+6=14
- Here, candidate node for expansion are {B,D} among these candidate nodes the node B is least evaluated, so it is selected for expansion.

Step 1



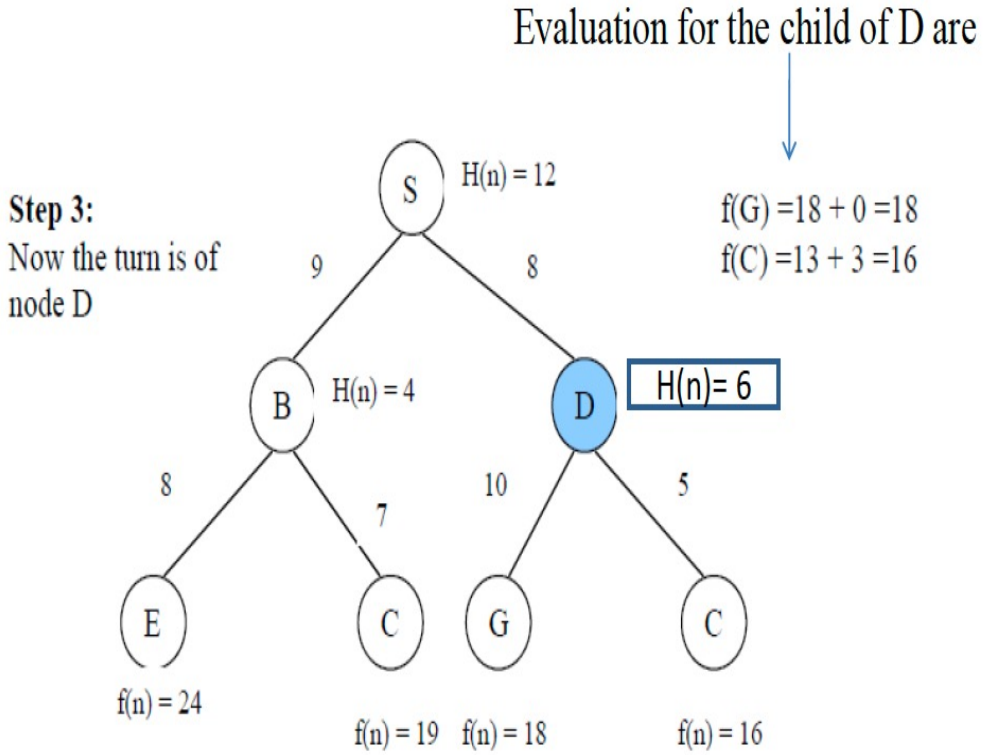
Contd..

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



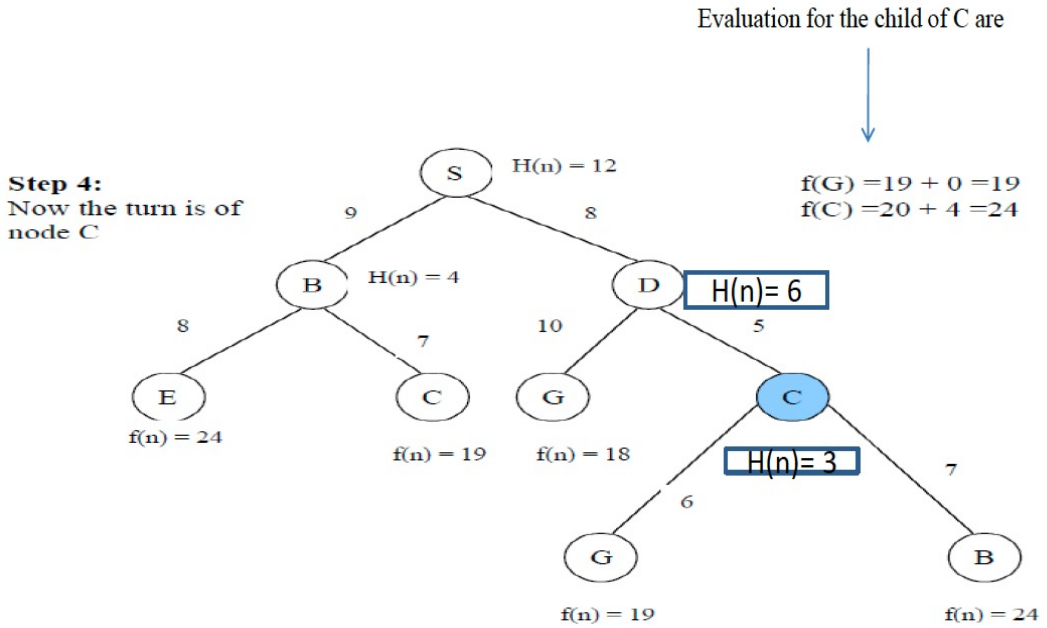
Contd..

- Child of D={G,C}
- 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.

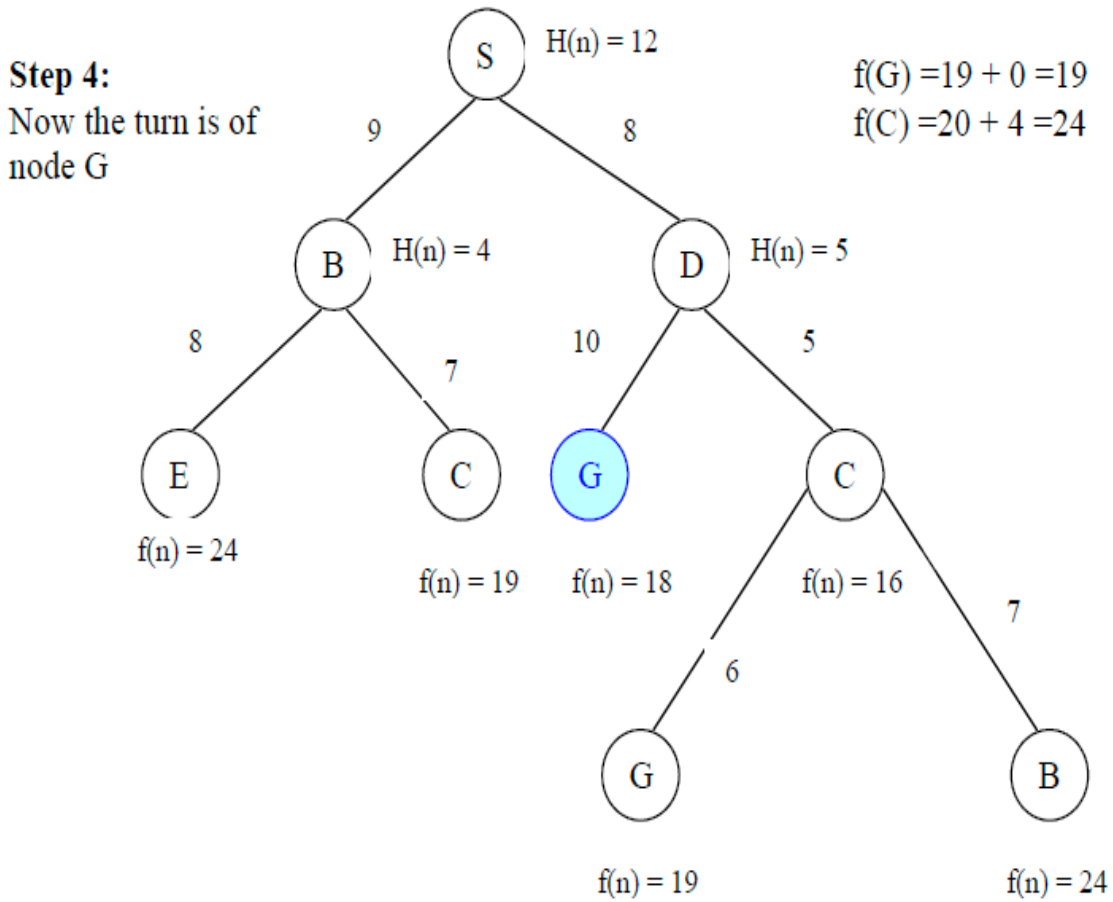


Contd..

- Child of C={G,B}
- Here the candidate for expansion are {E,C,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
S
H(n) = 12
9
8
f(G) =19 + 0 =19
f(C) =20 + 4 =24
B
H(n) = 4
D
H(n) = 5
8
7
10
5
E
C
G
C
f(n) = 24
f(n) = 19
f(n) = 18
f(n) = 16
7
6
G
B
f(n) = 19
f(n) = 24

Contd..
B. Local Search Algorithms:

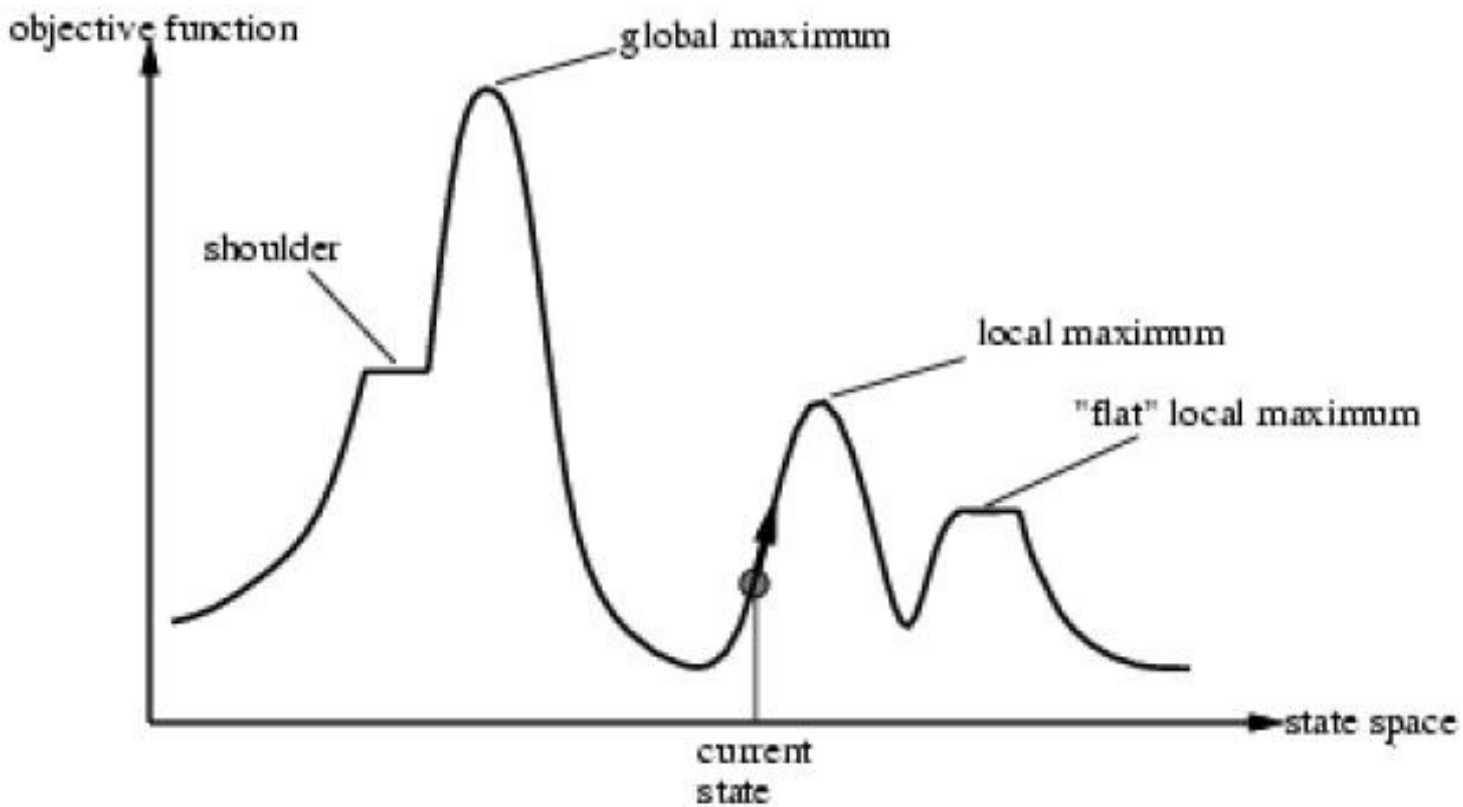
- Local search algorithm operate using a single current node that move only to neighbors of this node rather than systematically exploring paths from an initial state E.g, Hill Climbing.
- These algorithms are suitable for problems in which all that matters is the solution state, not the path cost to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

Contd..

Hill Climbing Search

- Hill climbining can be used to solve problem 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 the point the current solution is close to optimal, but it is not guaranteed that hill climbining will ever come close to the optimal solution.
- Note: The algorithm does not maintained 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

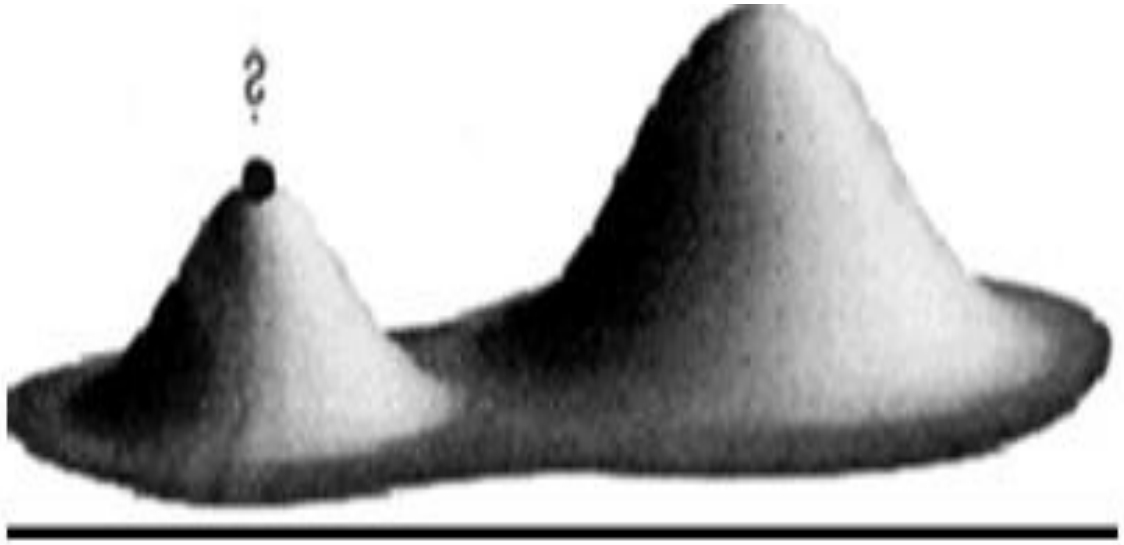
Contd..

- 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 set which is better than all of its neighbors but is not better than some other states which are farthe 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..

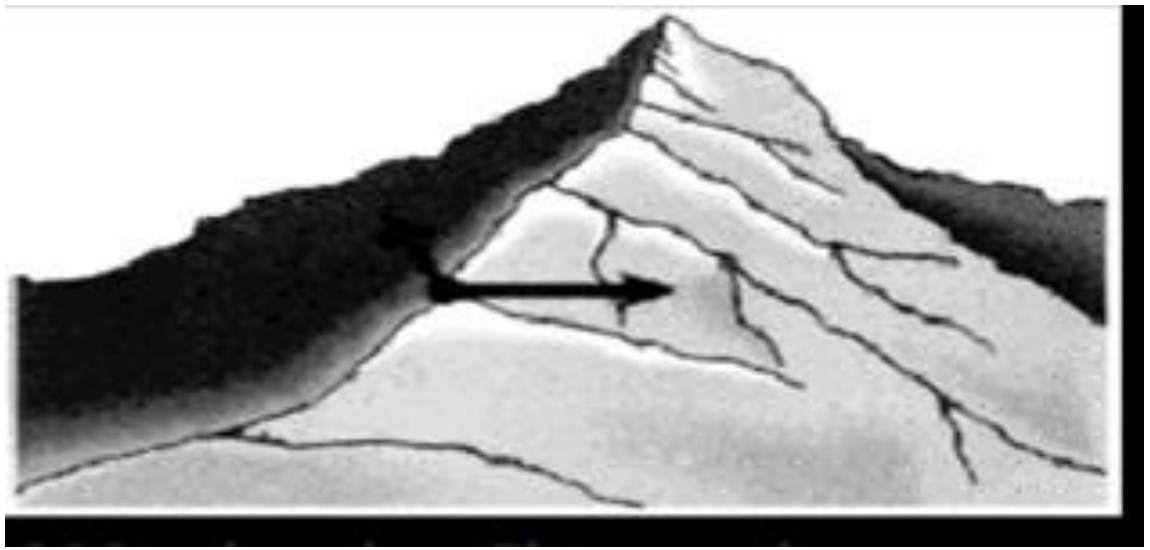
The plateau Problem:

- 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(oscilates from side to side)
- Such a problem is called ridge problem.
- **Soultion:** Apply two or more rules such as bi-direction search before doing the test.



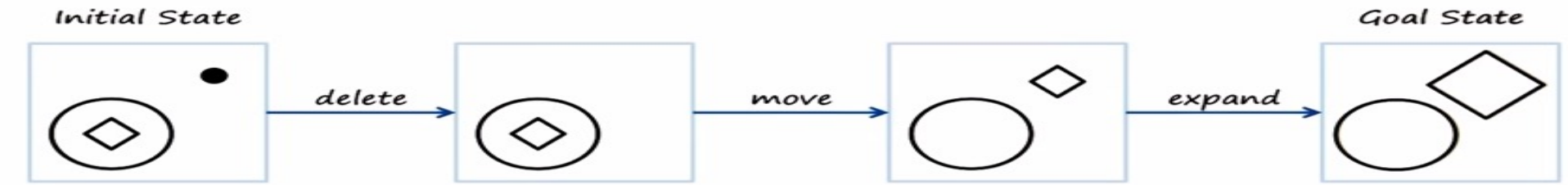
Mean-End Anlaysis

- Mean end analysis is a problem solving technique. Allows us to solve the major parts of a problem first and then go back and solve the smaller problem that arise while assembling the final solution.
- **Technique:**
  - The Mean End analysis process first detects the difference between current state and the goal state.
  - Once such a difference is isolated, an operator that can reduce the difference has to be found.
  - But sometimes it is not possible to apply this operator to the current state.
  - So, we try to get a sub-problem out of it and try to apply our oprator to this new state.
  - If this also does not produce the desired goal state then we try to get second sub-problem and apply this operator again. This process may be continued.

Contd..

- Problem

- Solution



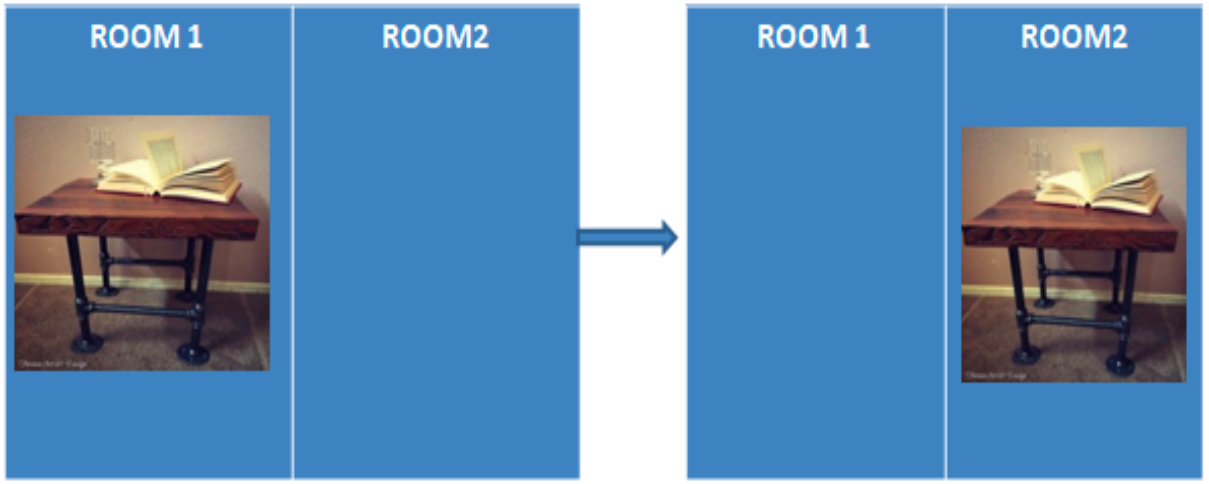
Contd..

- Why it is called mean end analysis?
  - Given a description of the desired state of the world(the end)
  - It works by selecting and using operators that will achieve it (the means)
  - Hence the name, mean end analysis (MEA)

Contd..

Mean End Analysis(Household Robot):

- Consider a simple household robot domain. Problem Given to the robot is: Nove's desk with two things on it from one room to another. The objects on top must also moved.



Initial state

Goal state

Contd..

- The available oprators are shown below with their pre-consitions and results.

Operator	Preconditions	Results
PUSH(Obj, Loc)	At(robot, obj)^ Large(obj)^ Clear(obj)^ armempty	At(obj, loc)^ At(robot, loc)
CARRY(Obj, loc)	At(robot, obj)^ Small(obj)	At(obj, loc)^ At(robot, loc)
WALK(loc)	None	At(robot, loc)
PICKUP(Obj)	At(robot, obj)	Holding(obj)
PUTDOWN(obj)	Holding(obj)	¬holding(obj)
PLACE(Obj1, obj2)	At(robot, obj2)^ Holding(obj1)	On(obj1, obj2)

- The table below shows when each of the operators is appropriate:

	Push	Carry	Walk	Pickup	Putdown	Place
Move object	*	*				
Move robot			*			
Clear object				*		
Get object on object						*
Get arm empty					*	*
Be holding object				*		

Contd..

- The Robot solve this problem as follows:
  - The main difference between the initial state and the goal state whould be the location of the desk.
  - To reduce this difference either PUSH or CARRY could be chosen.
  - If CARRY is chosen first, it conditions must be met. This results in two more difference that must be reduced
    - ❖The location of the robot and the size of the desk.
    - ❖The location of the robot is handled by applying WALK, but there are no operators that can change the size of an object, so this path leads to a dead-end.

Contd..

- Following the order branch , we attempt to apply PUSH. Figure below shows the robot's progress at this point.

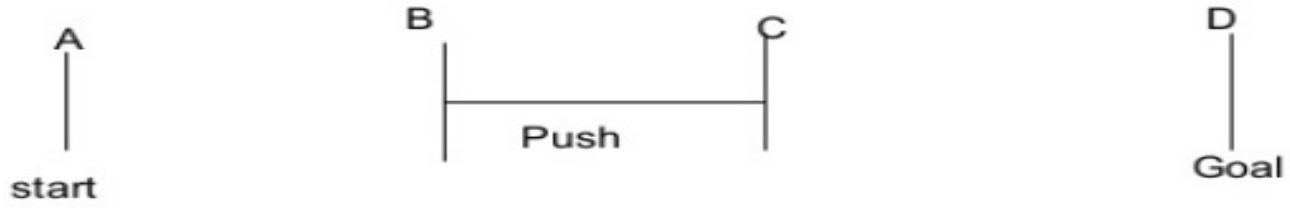


Fig: The progress of mean end analysis method

- Now the difference between A and B and between C and D must be reduced.

Contd..

- PUSH has preconsitions:
  - The robot must be at the desk and
  - The desk must be clear
- The robot can be brought to the correct location by using WALK.
- And the surface of the desk can be cleared by two uses of the PICKUP.
- But after one PICKUP, an attempt to do the second result in another difference (the arm must be empty).
- PUTDOWN can be used to reduce the difference.

Contd..

- Once PUSH is perform, the problem state is closed to the goal state, but not quite. The object must be placed back on the desk. PLACE will put them there. The progress of the robot at this point is as shown in the figure below:

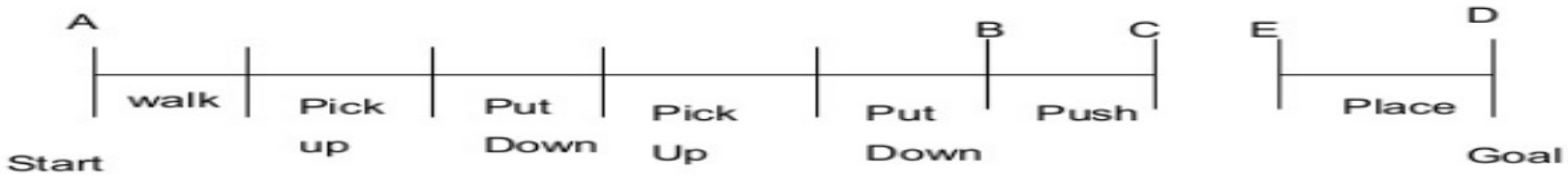


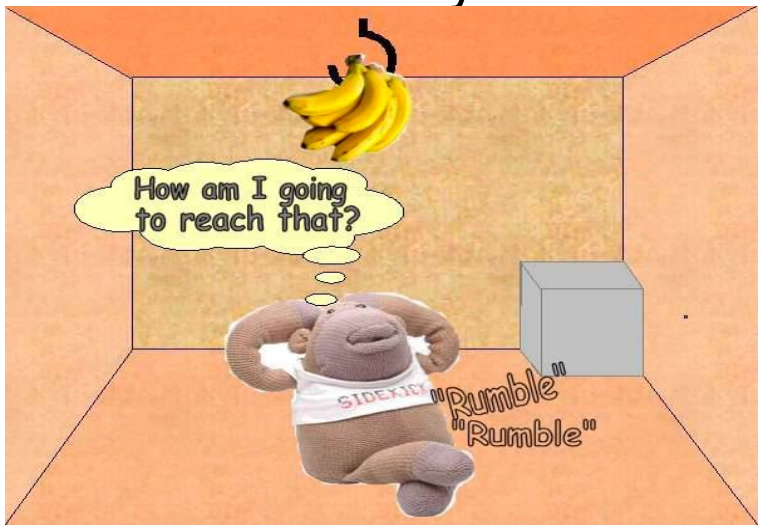
Fig: more progress on the mean end analysis method

- The final difference between C and E can be reduced by using WALK to get the robot back to the object followed by PICKUP and CARRY

Contd..

Mean-End Analysis(Monkey Banana Problem)

- The monkey is in a closed room in which there is a small Box.
- There is a bunch of bananas hanging from the ceiling but the monkey cannot reach them.
- Assuming that the monkey can move the Box and if the monkey stands on the Box the monkey can reach the bananas.
- Establish a method to instruct the monkey on how to capture the bananas

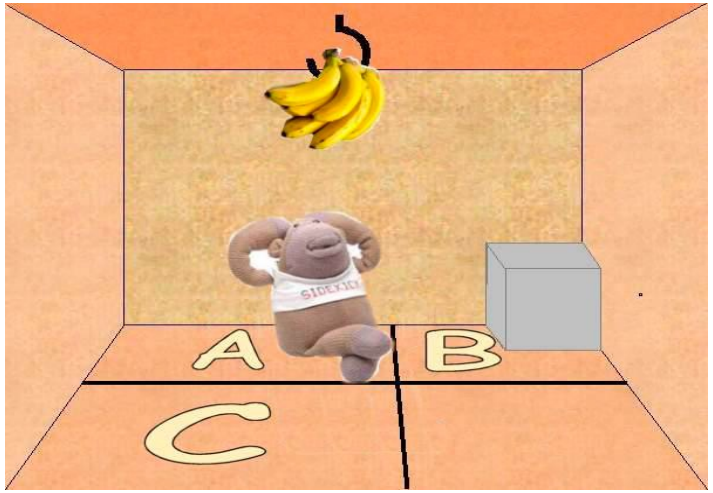


Contd..

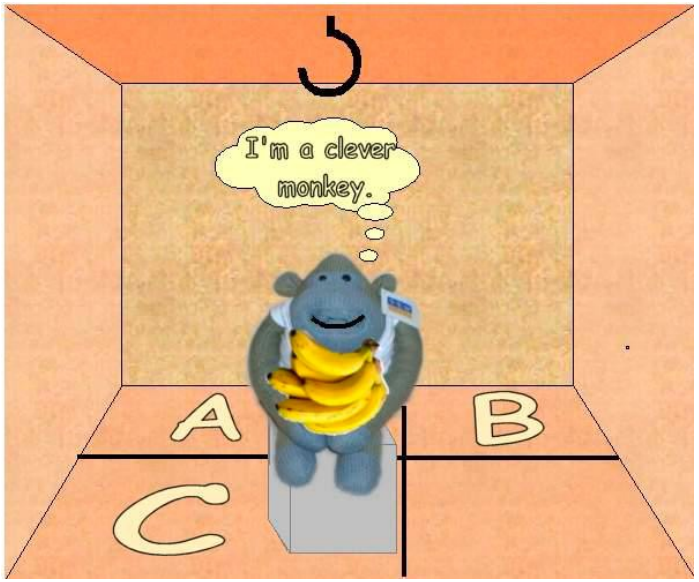
- Representing the world:
- In the monkey Banana Problem we have:
  - Object: A monkey, a Box, The Bananas, and a floor.
  - Locations: we'll call them a,b and c.
  - Relations of objects to locations. For example
    - The monkey is at location a
    - The monkey is on the floor
    - The bananas are hanging
    - The box is in the same locations as the bananas
- Formally representing the relations of object to location as
- At(monkey,a)
- On (monkey, floor)
- Status (bananas, hanging)
- At (box, X0, at (bananas, X)

Contd..

- Initial State:
  - At(Monkey,a)
  - On(monkey, floor)
  - At(box,b)
  - On(box,floor)
  - At(bananas,c)
  - Status(bananas, hanging)



- Goal State:
- On(monkey, box)
- On(box, floor)
- At(monkey,c)
- At(box,c)
- At(bananas,c)
- Status(bananas, grabbed)



Contd..

- All Operators:

Operator	Preconditions	Results
go(X,Y)	at(monkey,X)	at(monkey, Y)
	on(monkey, floor)	
push(B,X,Y)	at(monkey,X)	at(monkey, Y)
	at(B,X)	at(B,Y)
	on(monkey, floor)	
	on(B,floor)	
climb_on(B)	at(monkey,X)	on(monkey, B)
	at(B,X)	
	on(monkey,floor)	
	on(B,floor)	
grab(B)	on(monkey,box)	status(B, grabbed)
	at(box,X)	
	at(B,X)	
	status(B, hanging)	

Contd..

- Instructions to the monkey to grab the banana
  - First of all monkey should move from location a to location b
    - Goal(a,b)
  - Then push the box from location b to location a
    - Push(B,b,a)
  - Push the box again from location a to c
    - Push(B,a,c)
  - Monkey should climb on the Box
    - Climb_on(B)
  - Then Grab the banana
    - Grab(B)

A constraint satisfaction problem

- ✓ A constraint satisfaction problem consists of three components, **X, D, and C**:
  - ✓ X is a set of variables, $\{X_1,...,X_n\}$.
  - ✓ D is a set of domains, $\{D_1,...,D_n\}$, one for each variable.
  - ✓ C is a set of constraints that specify allowable combinations of values. i.e. relations, that are assumed to hold between the values of the variables.
- ✓ Each domain Di consists of a set of allowable values, $\{v_1,...,v_k\}$ for variable X_i.

Contd..

- ✓ To solve a CSP, we need to define a state space and the notion of a solution.
- ✓ Each state in a CSP is defined by an assignment of values to some or all of the variables, $\{X_i = v_i, X_j = v_j, ...\}$.

Consistent

- ✓ An assignment that **does not violate any constraints** is called a consistent or legal assignment.

Complete

- ✓ A complete assignment **is one in which every variable is assigned.**

- ✓ *A solution to a CSP is a consistent, complete 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: Di = { red, green, blue}

Constraints: adjacent regions must have different colors

- e.g., WA ≠ NT

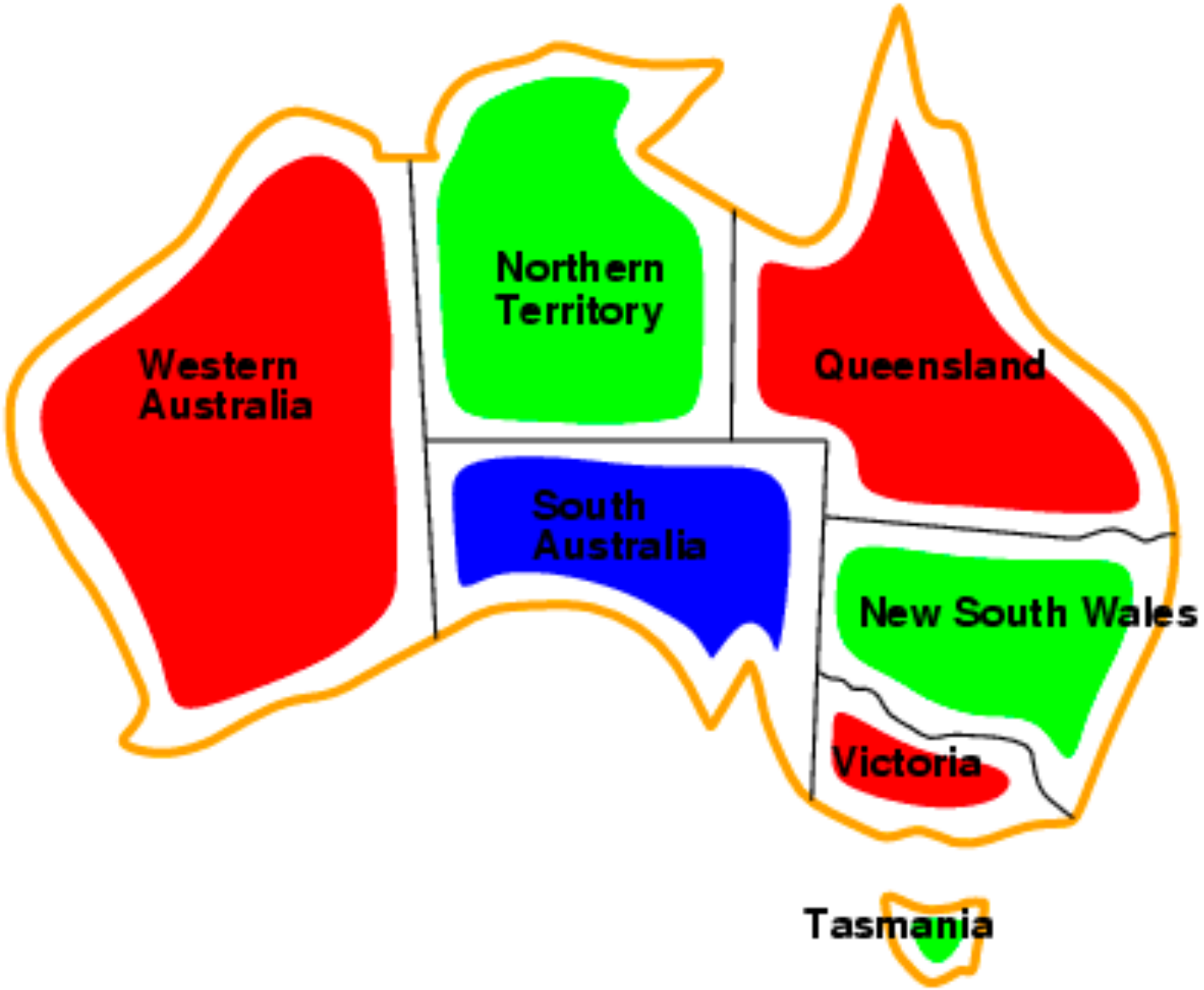
Example: Map Coloring

- **Variables:** WA, NT, Q, NSW, V, SA, T
- **Domains:** {red, green, blue}
- **Constraints:** adjacent regions must have different colors
  e.g., WA ≠ NT, or (WA, NT) in {(red, green), (red, blue), (green, red), (green, blue), (blue, red), (blue, green)}

Contd..

- Solution 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 expredded as a standard search problem, as follows.
  - **Initial State:** The empty assignmanet{}, in which all variables are unassigned.
  - **Successor Function:** Assign value to unassigned variable provided that there is not conflict.
  - **Goal Test:** The current assinment is complete and consistent.
  - **Path Cost:** A constant cost for every step.

Contd..

- **Constraint Satisfication 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 deleted, then backtrack return failure, causing the previous call to try to another values.

Contd..

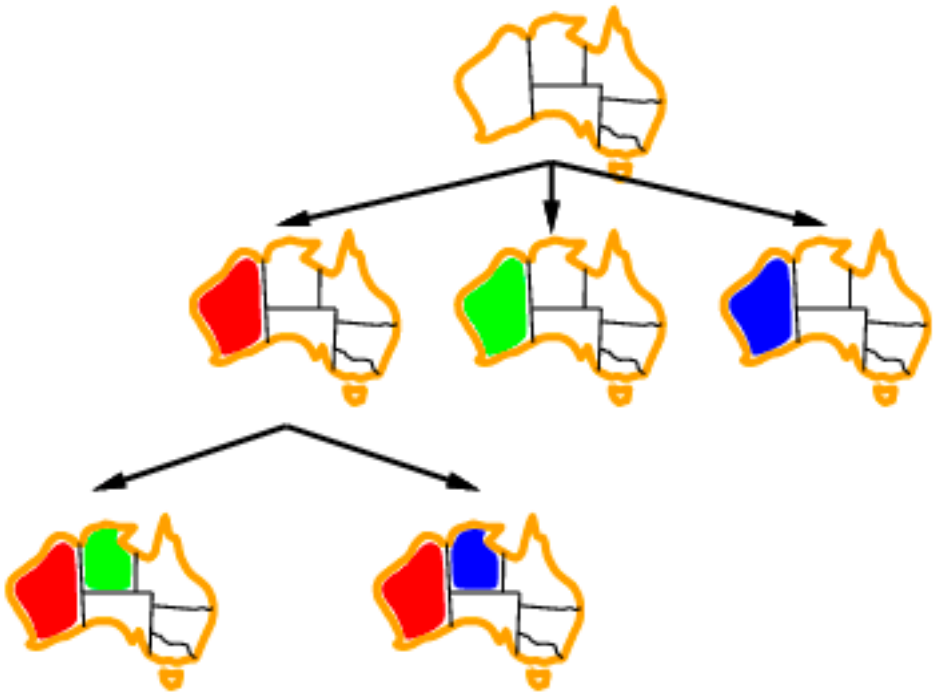
Backtracking Example

Contd..

Backtracking Example

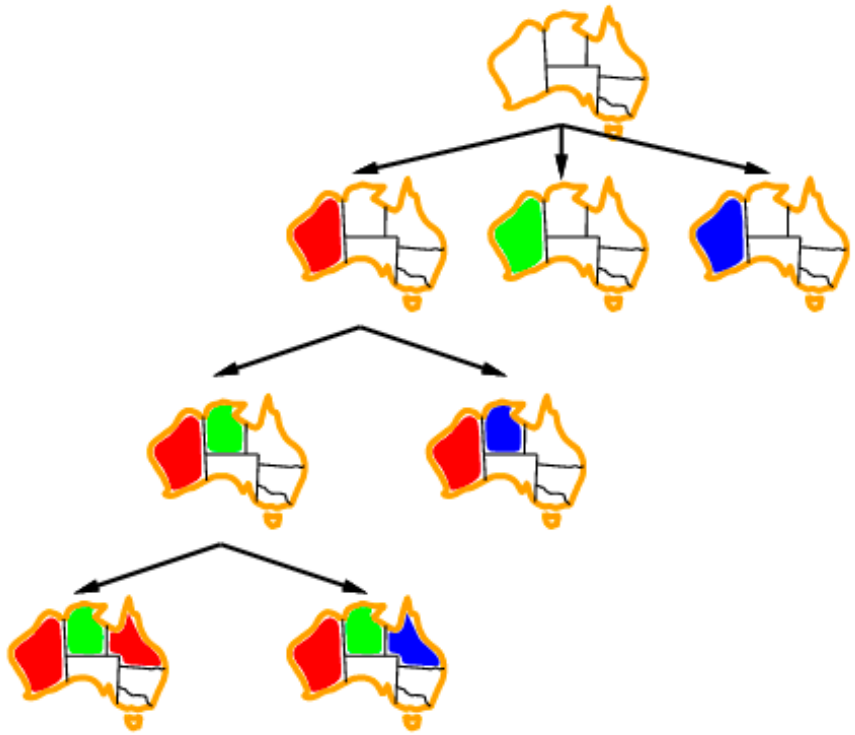
Contd..

Backtracking Example



Contd..

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 problem in AI can be considered as problems of constraint satisfication, 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 zeros are allowed.

Contd..

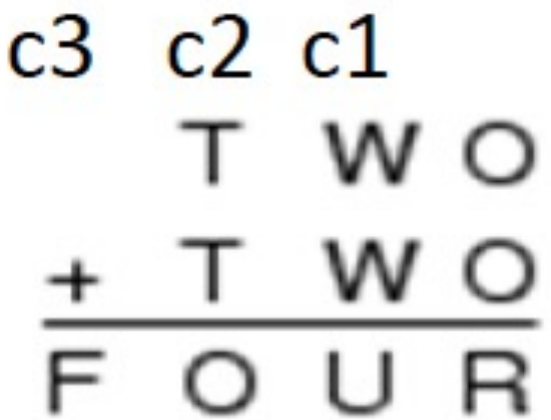
- **Example1:** Solve the following puzzle by assigning numeral (0-9) in such a way that each letter is assigned unique digit which satisfy the following addition.

$$\begin{array}{r} \text{T W O} \\ +\ \text{T W O} \\ \hline \text{F O U R} \end{array}$$

Conts..

- Solution: Here,
  - Variables:{F,T,U,W,R,O,c1,c2,c3}
  - Domain: {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.

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



Conta...

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

```
  928
+ 928
------
 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	9235
+FOUR	+ 9235
EIGHT	18470

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.

```
    S  E  N  D
+   M  O  R  E
-------------
M   O  N  E  Y
-------------
```

- **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	←	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

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.

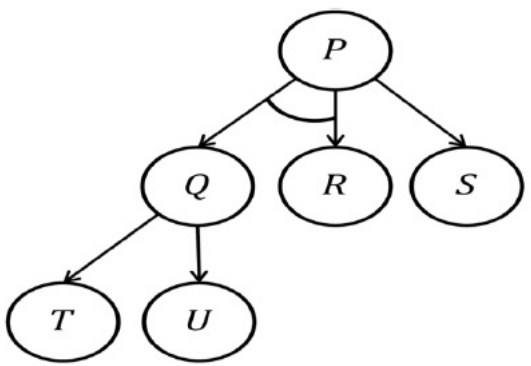
- Hence we get the final solution as given below and on backtracking, we may find more solutions.

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

Problem Reduction: AND/ OR Tree

- The basic intuition behind the method of problem reduction is:
  - Reduce a hard problem to a number of simple problems and, when each of the simple problems is solved, then the hard problem has been solved.
- To represent problem reduction we can use an AND-OR tree. An AND–OR tree is a graphical representation of the reduction of problems to conjunctions and disjunctions of sub-problems.

- Example:

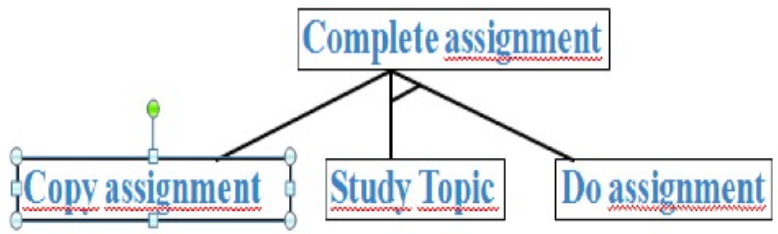


represents the search space for solving the problem P, using the problem-reduction methods:

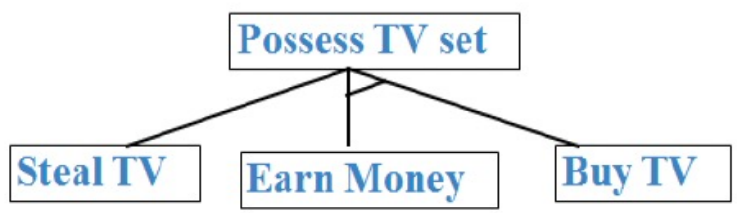
P if Q and R
P if S
Q if T
Q if U

Contd…

- Example1



- Example2:



Contd….

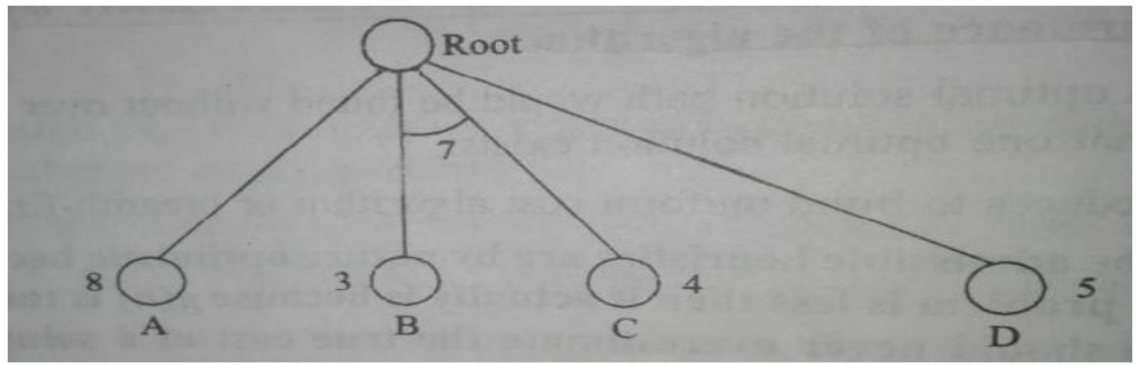
- Searching AND/OR Tree:
  - To find a solution in AND–OR tree, an algorithm similar to Best first search algorithm is used but with the ability to handle AND arc appropriately.
  - This algorithm evaluates the nodes based on the following evaluation function.
    - If the node is OR node then cost function f(n)= h(n)
    - If the node is AND node then cost function is the sum of costs in AND node.

      – $f(n) = f(n_1) + f(n_2) + \ldots + f(n_k) = h(n_1) + h(n_2) + \ldots + h(n_k)$

      – Where, $n_1, n_2, \ldots N_k$ are AND nodes.

Contd…

- Here, the number show the value of the heuristic function at that node.
- In the figure the minimal is B which is the value of 3. But B forms a part of the AND tree so we need to consider the other branch also of this AND tree i.e., C which has a weight of 4. So, our estimate now is (3+4) = 7.



- Now this estimate is more costlier than that of branch D i.e., 5. So we explore node D instead of B as it has the lowest value.
- This process continues until either a solution is found or all paths led to dead ends, indicating that there is no solution.

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…

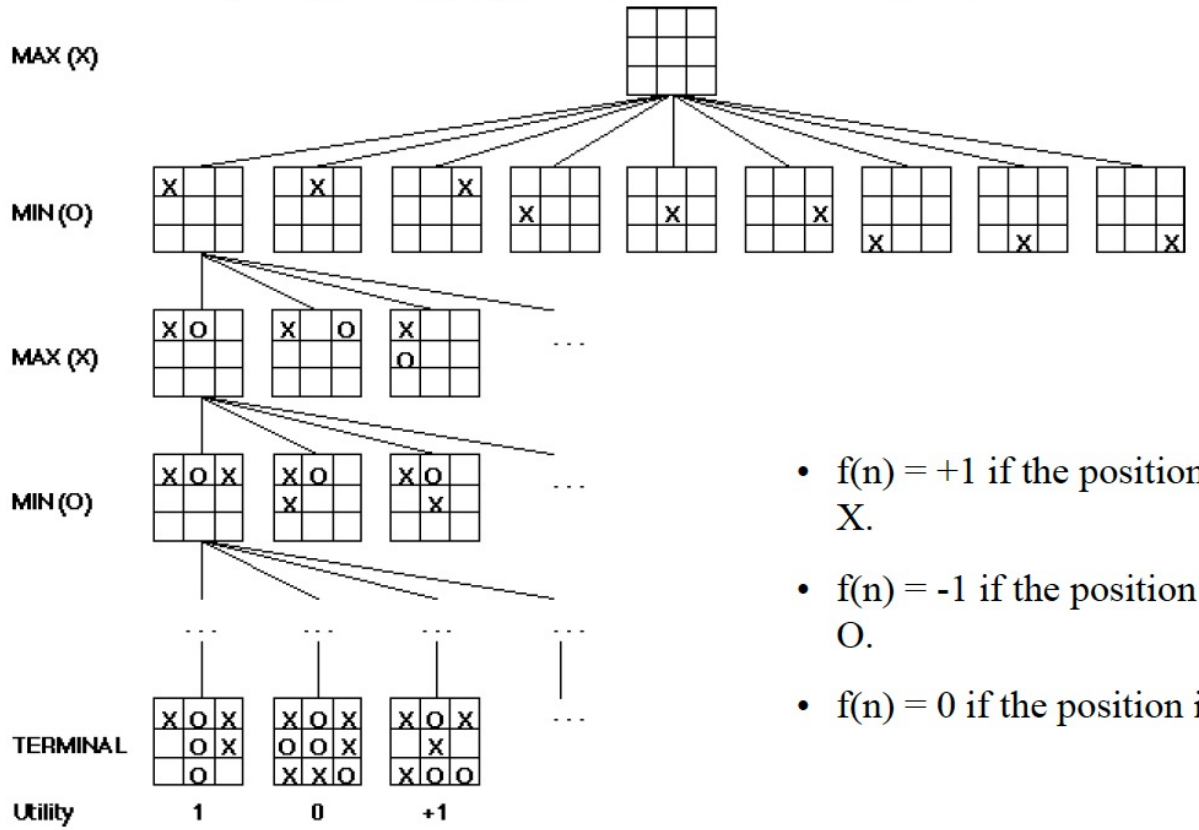
- Evaluation Function:
  - An evaluation function is used to evaluate the "goodness" of a game position.
  - i.e., estimate of the expected utility of the game position
  - The performance of a game playing program depends strongly on the quality of its evaluation function.
  - An inaccurate evaluation 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 err 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…

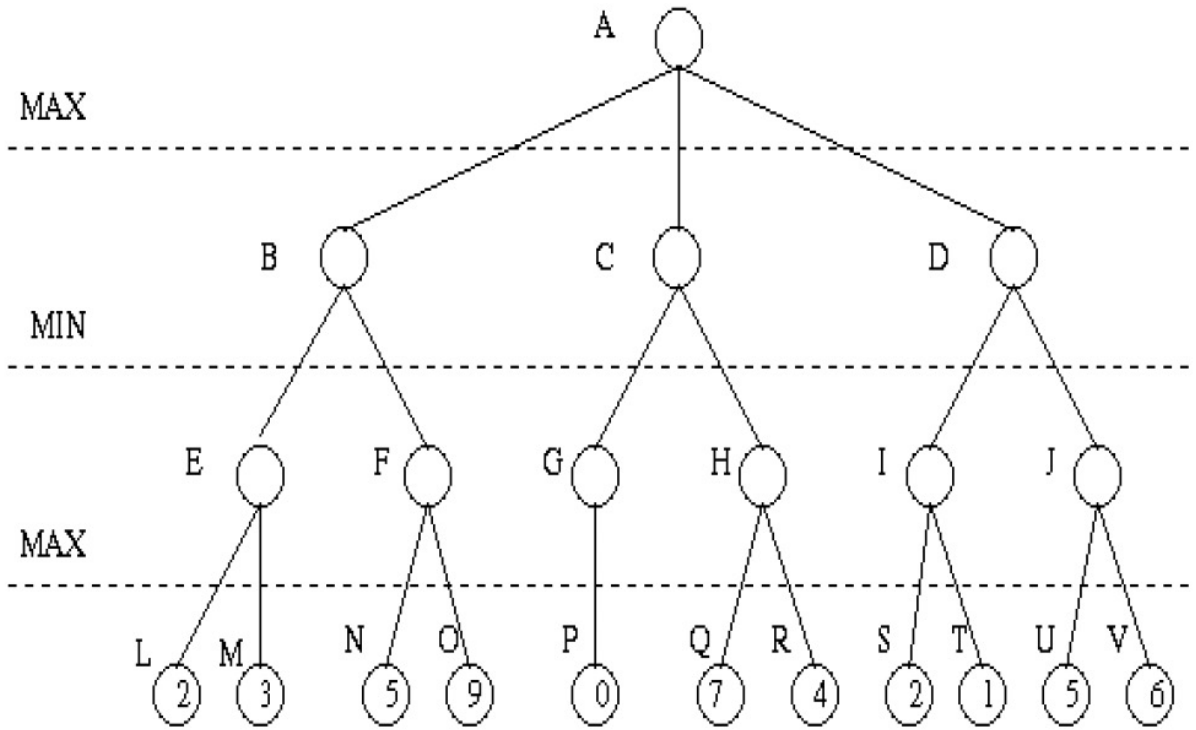
- There are two players denoted by X and O. They are alternatively writing their letter in one of the 9 cells of a 3 by 3 board. The winner is the one who succeeds in writing three letters in line.
- The game begins with an empty board. It ends in a win for one player and a loss for the other, or possibly 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.

Mini-max search algorithm

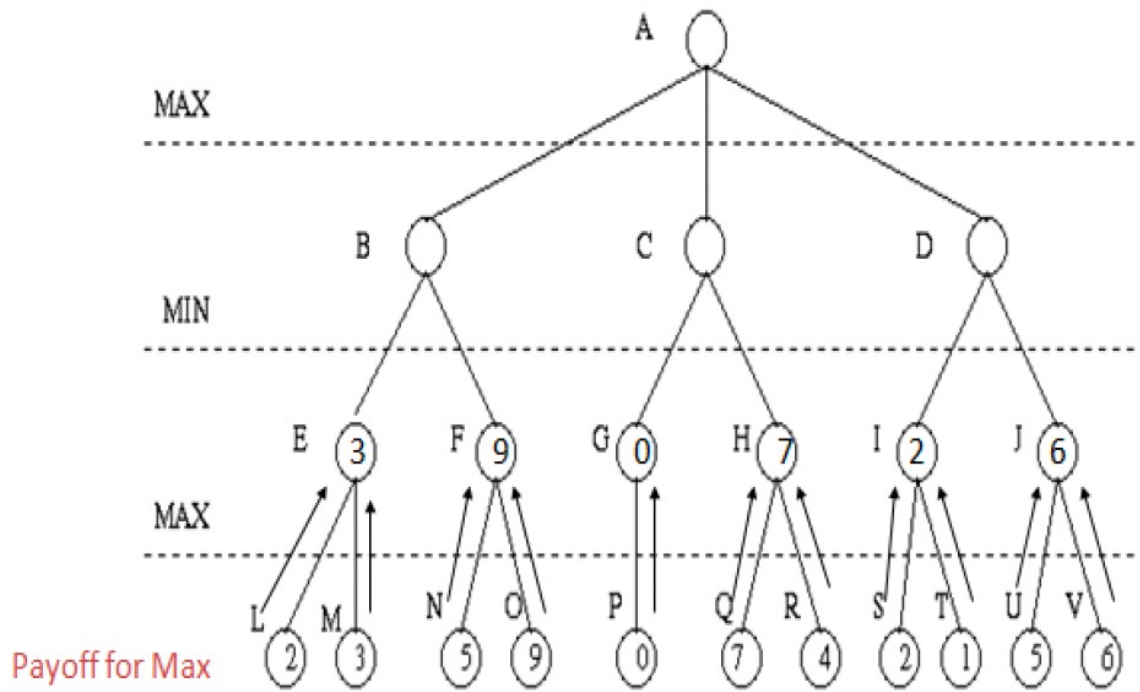
- Mini-max search algorithm is a game search algorithm with the application of DFS procedure.
- It assumes:
  - Both the player play optimally from there to the end of the game.
  - A suitable value of static evaluation(utility) is available on the leaf nodes.
- Given the value of the terminal nodes, 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…

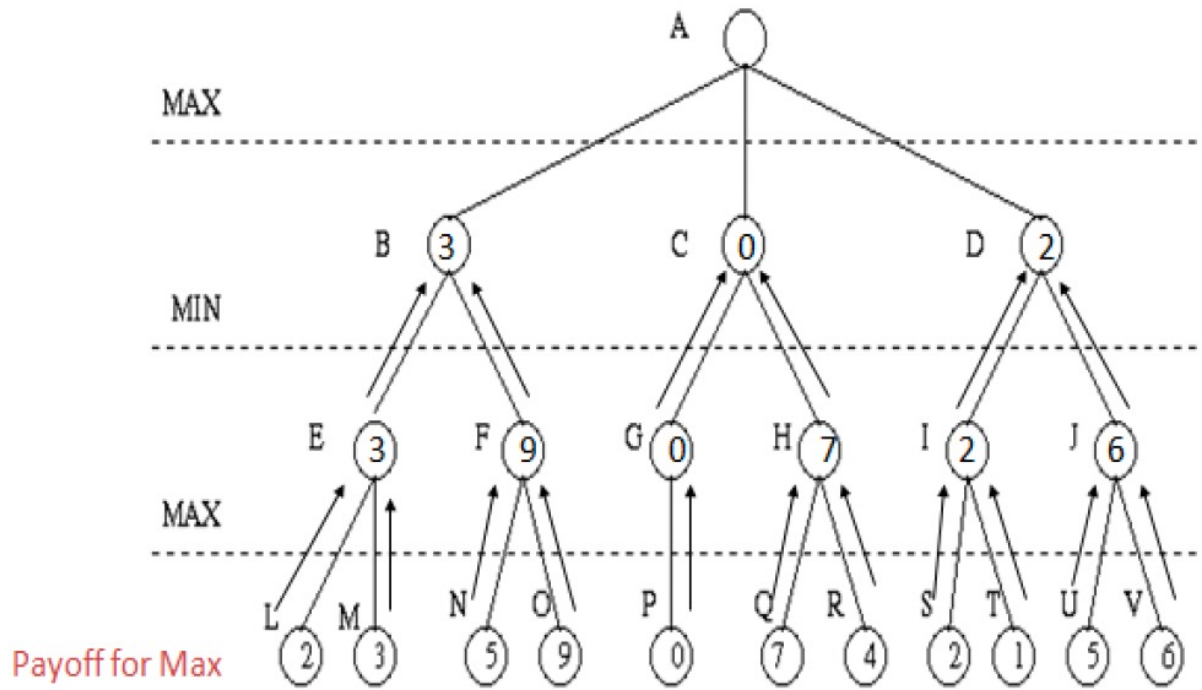
- Mini-max search Example:



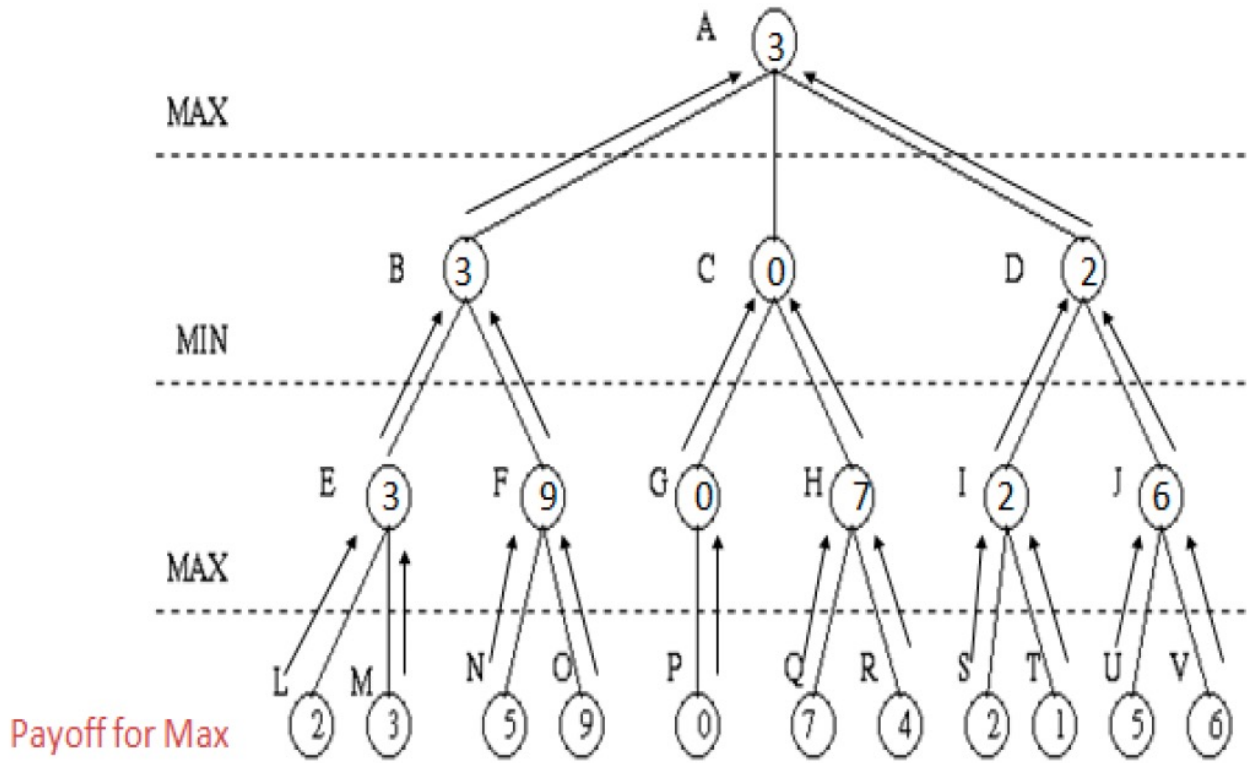
Contd…


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
2
3
5
9
0
7
4
2
1
5
6
Payoff for Max

Contd…


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

Contd…

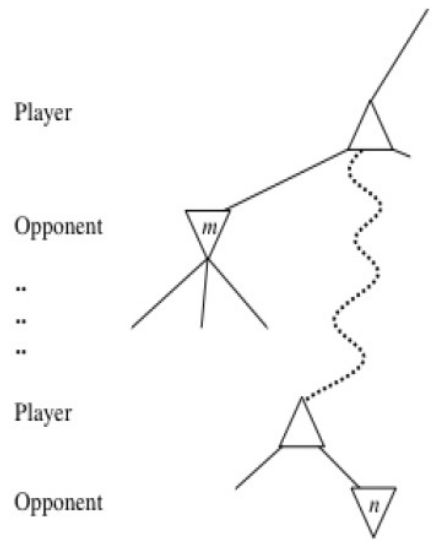

A
3
MAX
B
3
C
0
D
2
MIN
E
3
F
9
G
0
H
7
I
2
J
6
MAX
L
M
N
O
P
Q
R
S
T
U
V
2
3
5
9
0
7
4
2
1
5
6
Payoff for Max

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

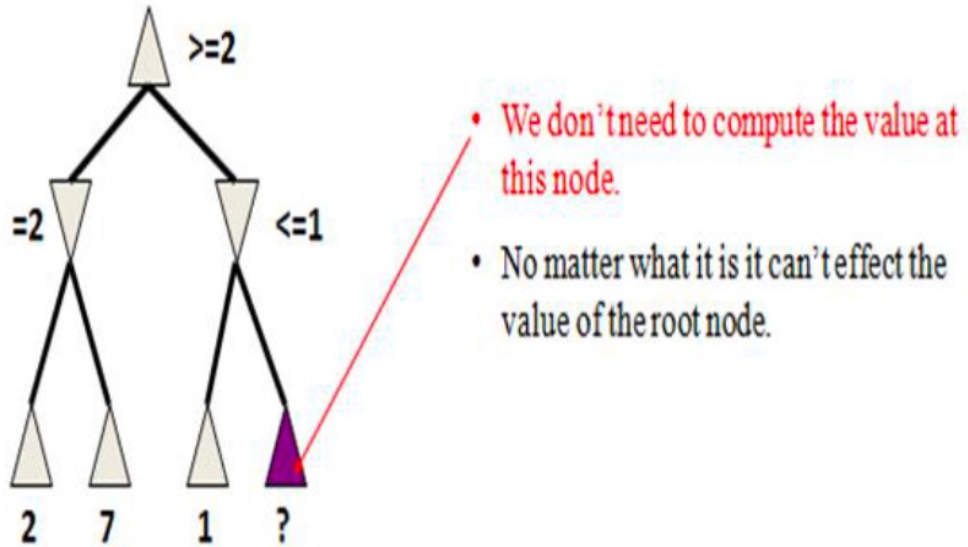
Alpha-beta pruning

- 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:

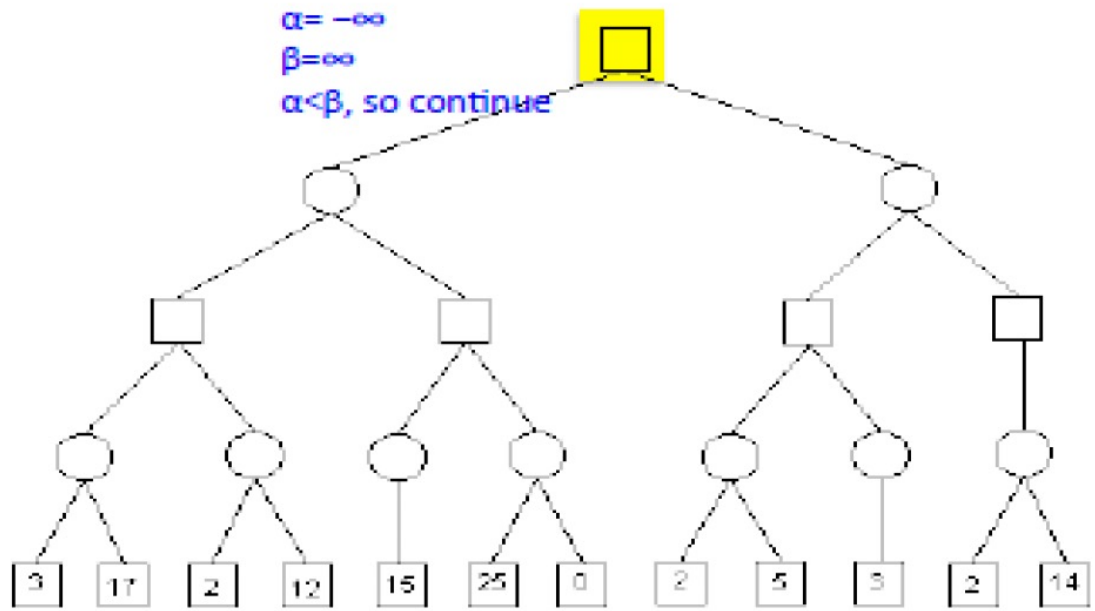


Contd…

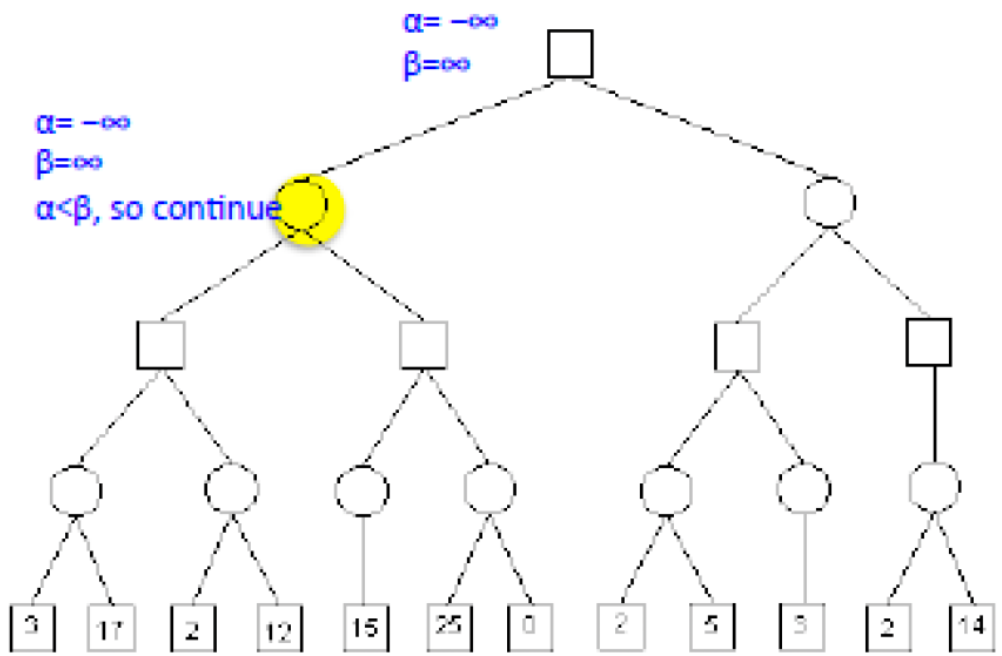
- Alpha-Beta pruning procedure:
  - Traverse the search tree in depth-first order. During traversing Alpha and Beta values inherited from the parent to child never from the children. Initially alpha =-infinity and always try to increase, and beta=+infinity and always try to decrease. Alpha value updates 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…

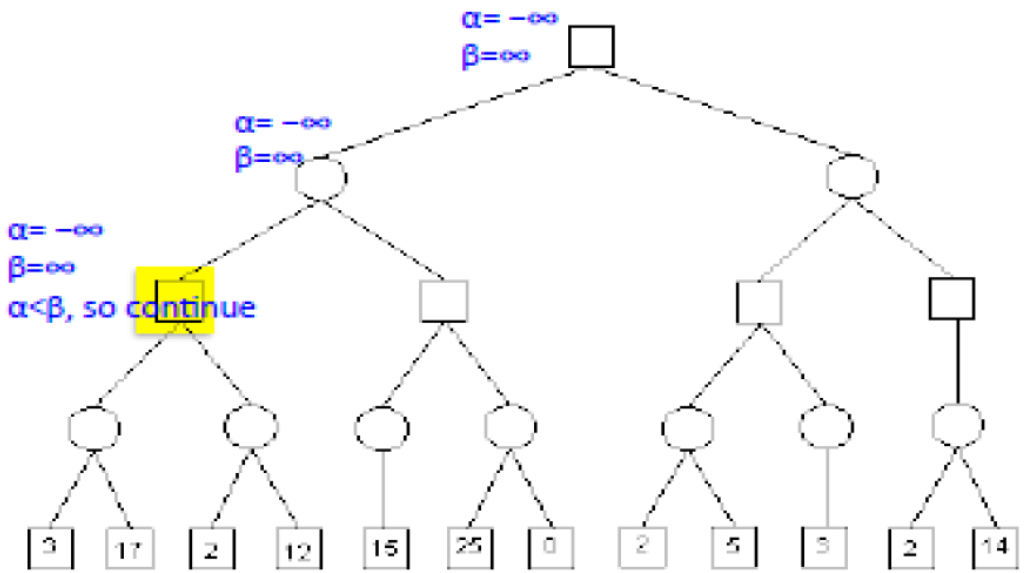
- Alpha-Beta pruning example:



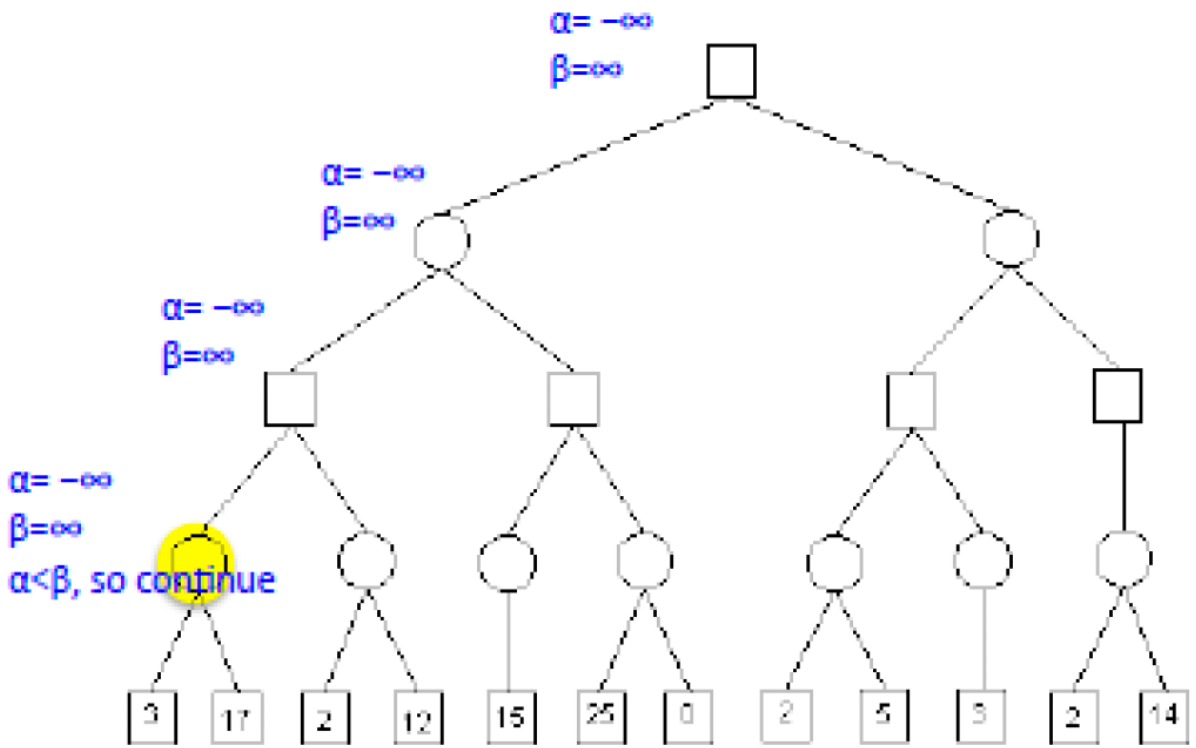
Contd…



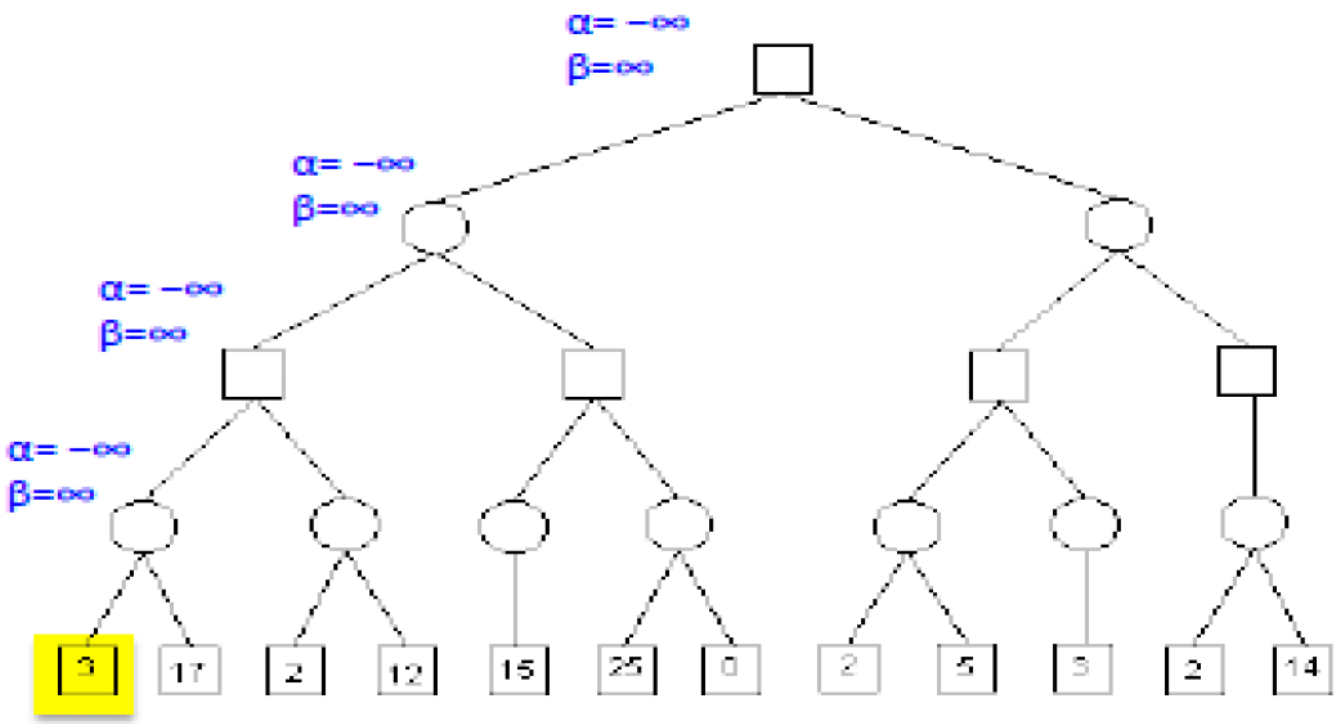
Contd…



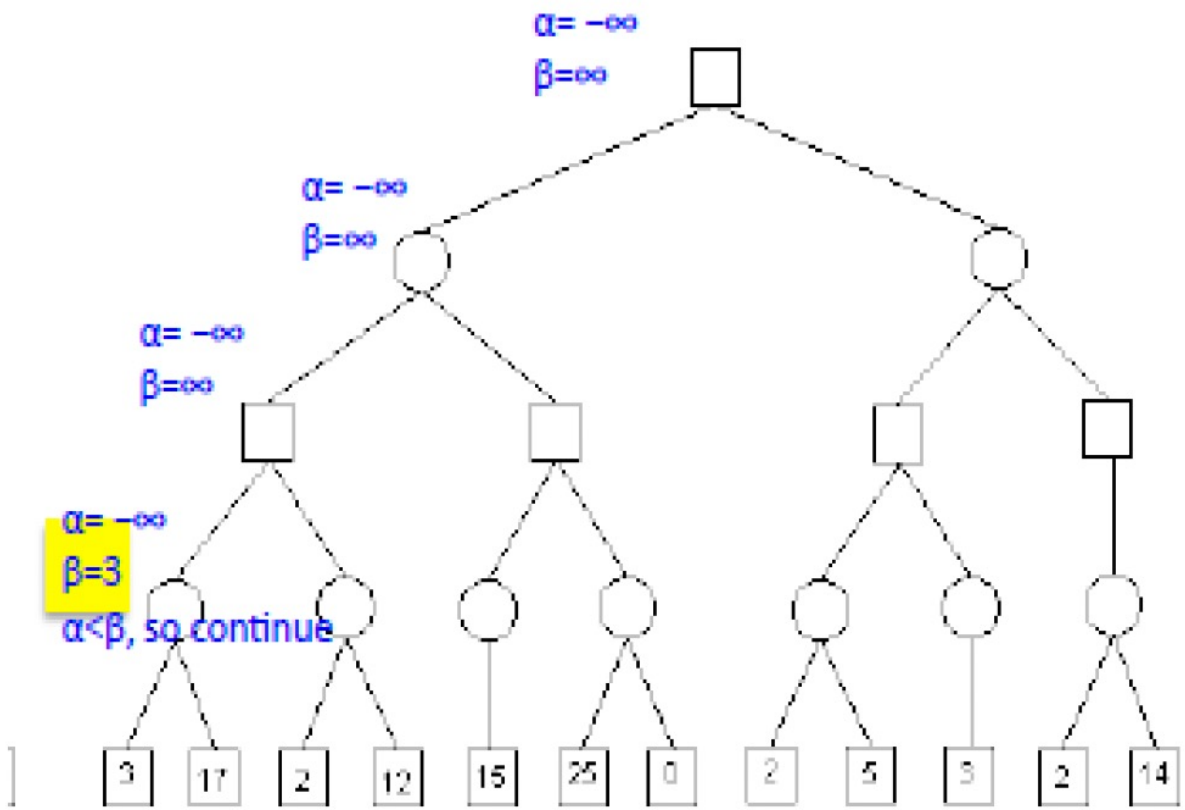
Contd…



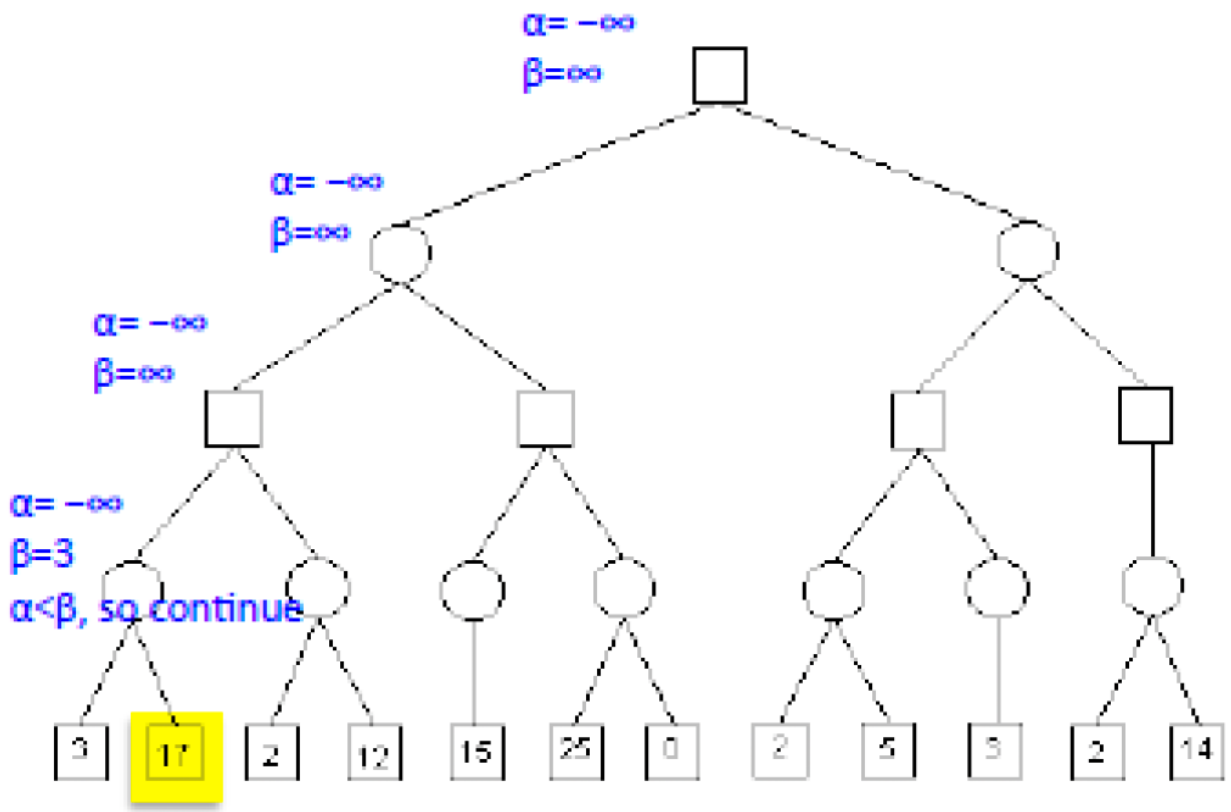
Contd…



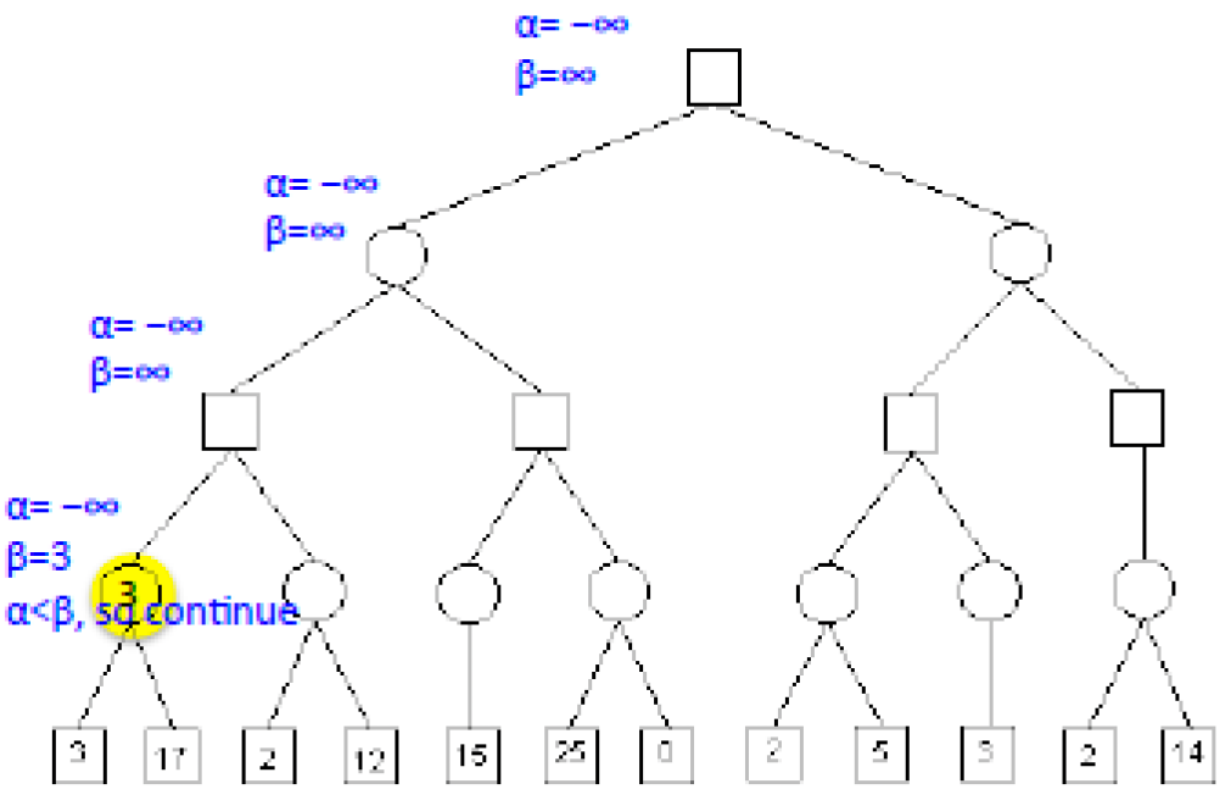
Contd…



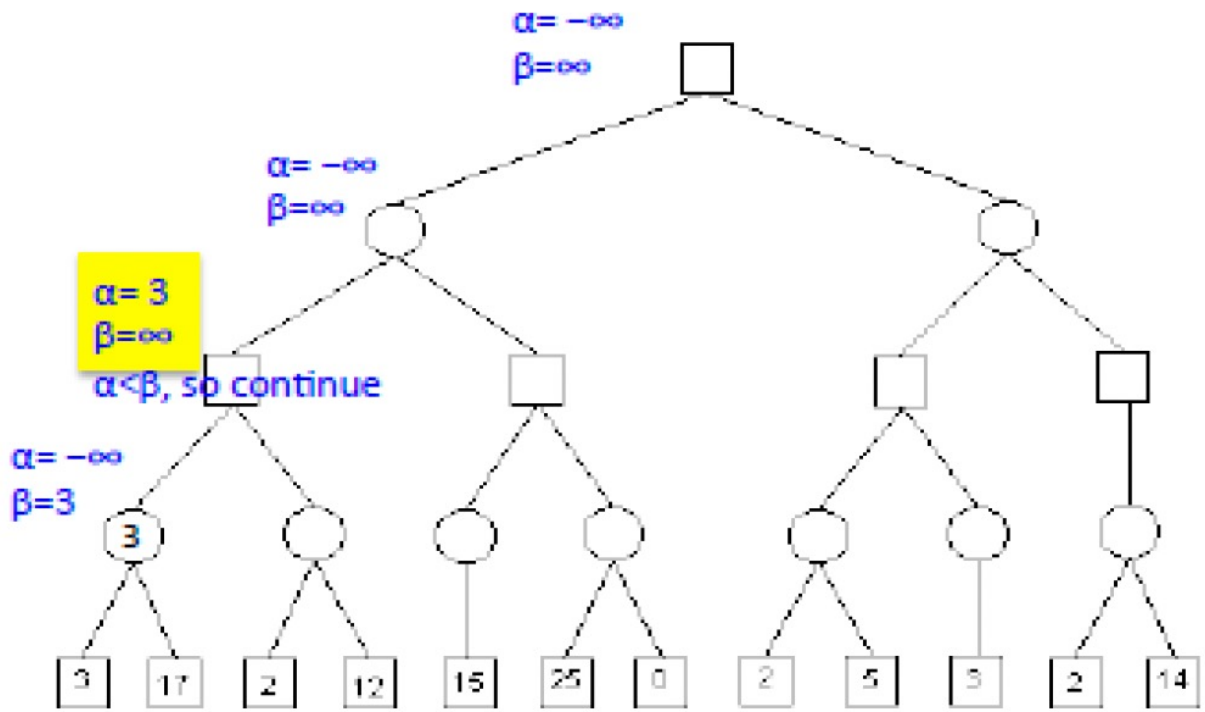
Contd…



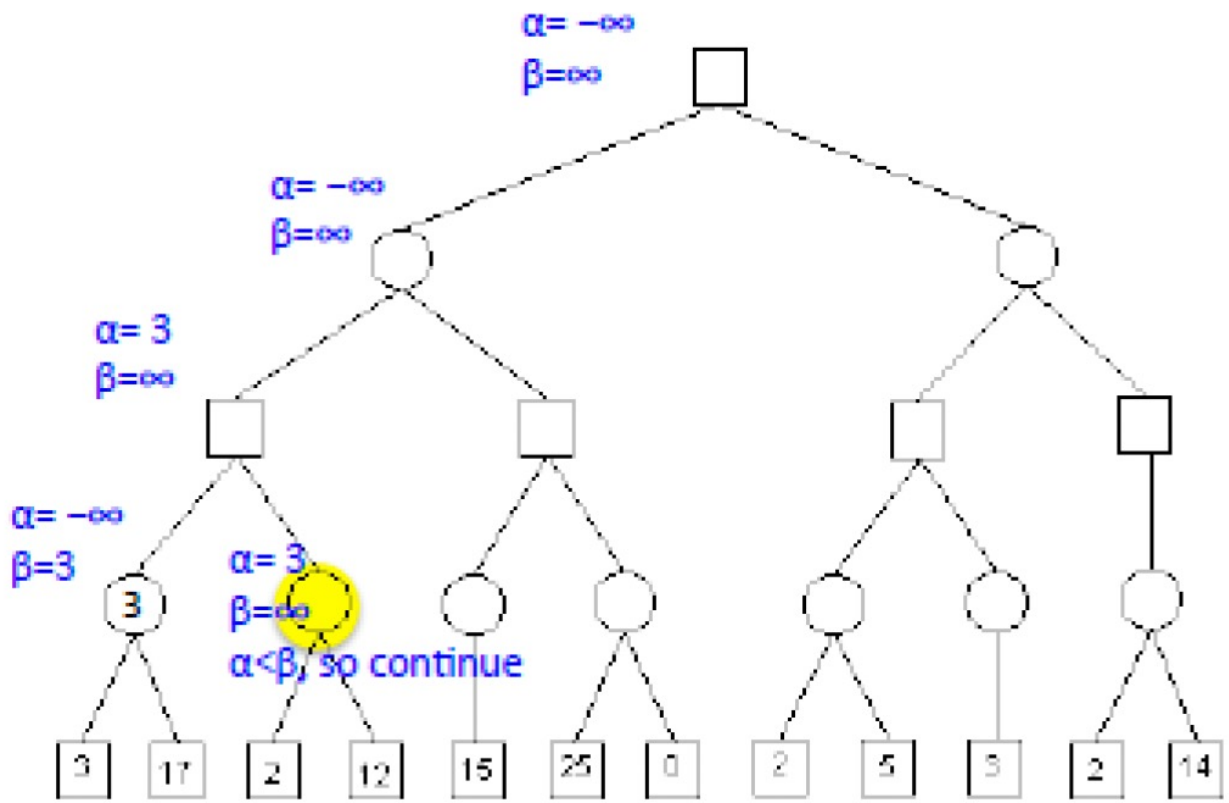
Contd…



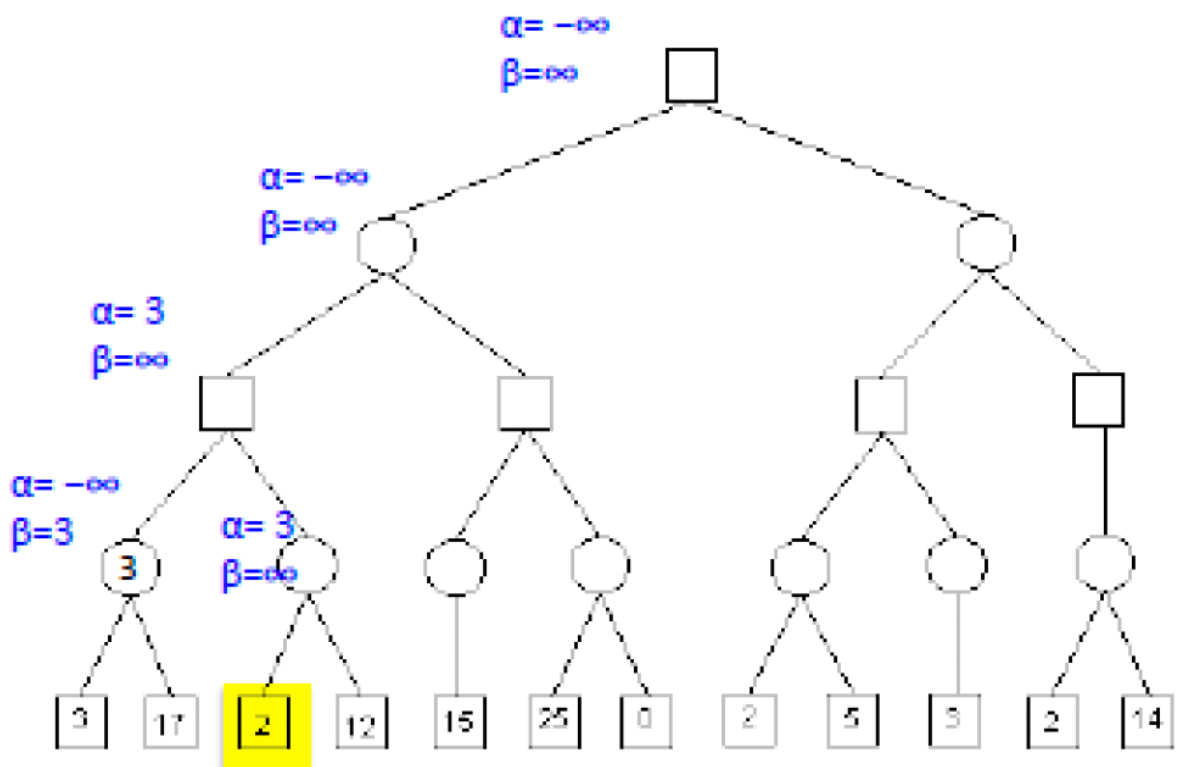
Contd…



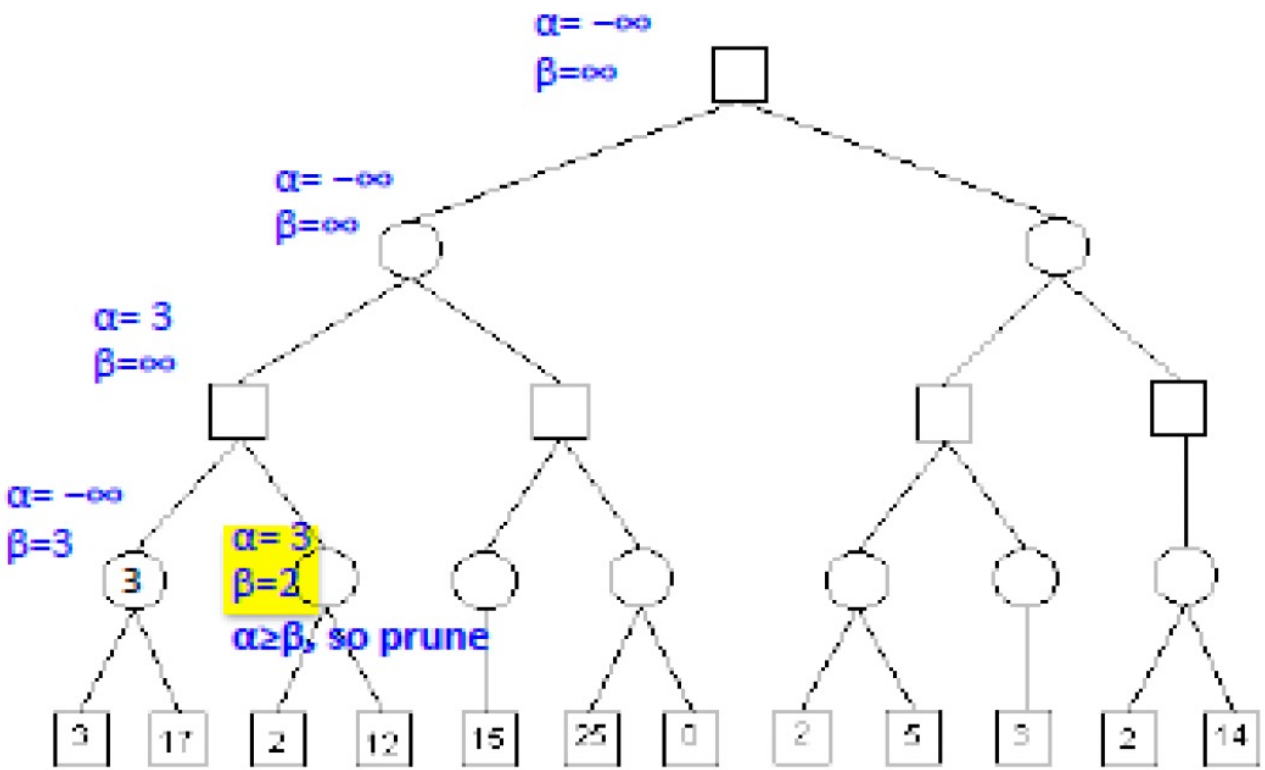
Contd…



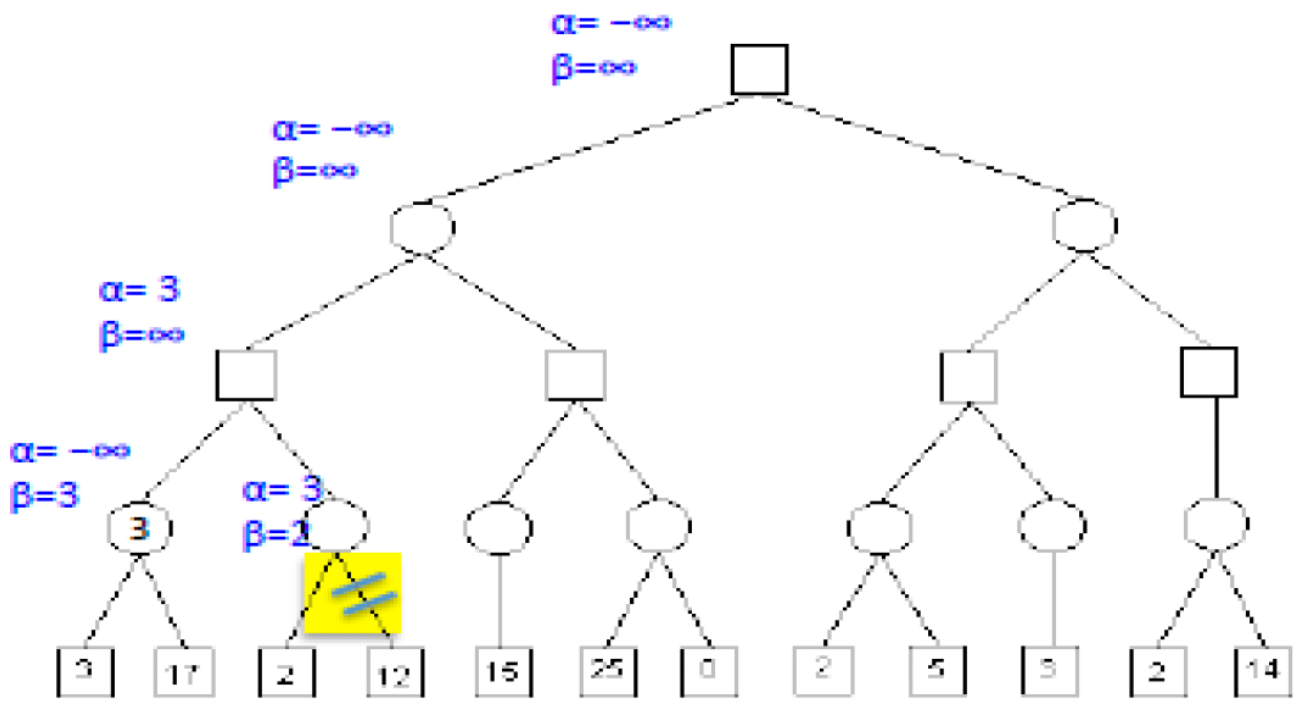
Contd…



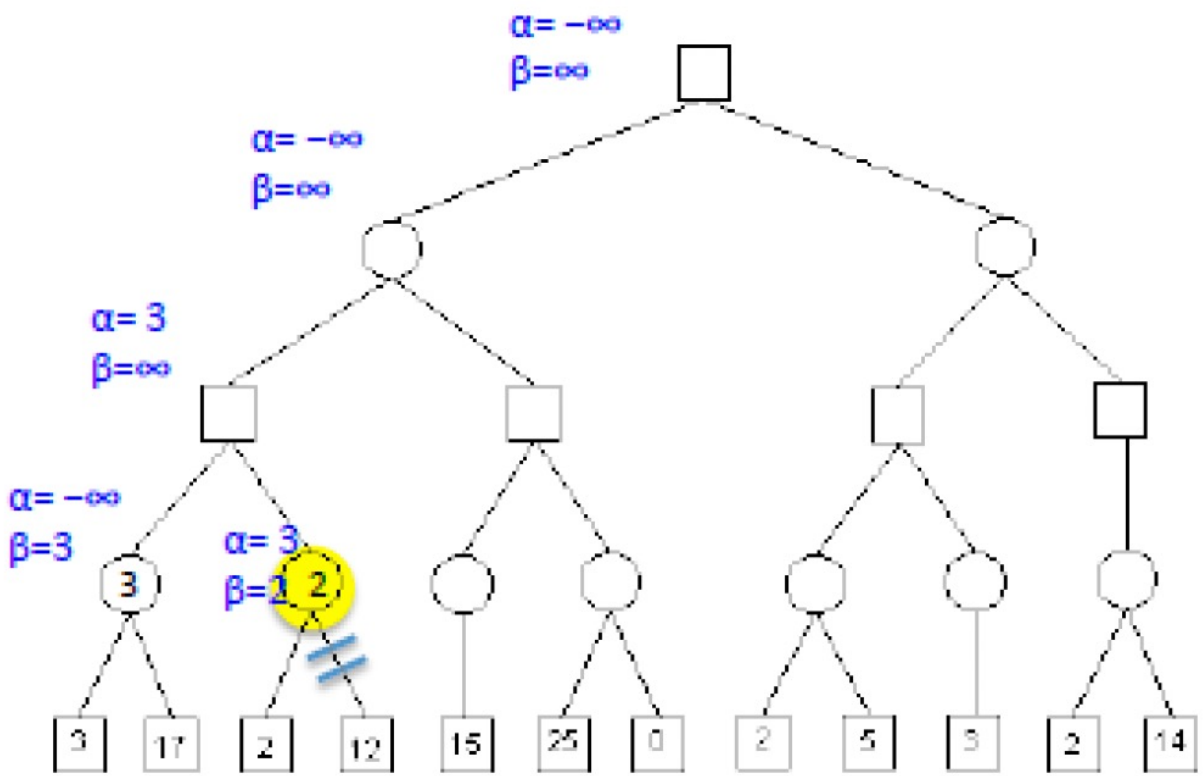
Contd…



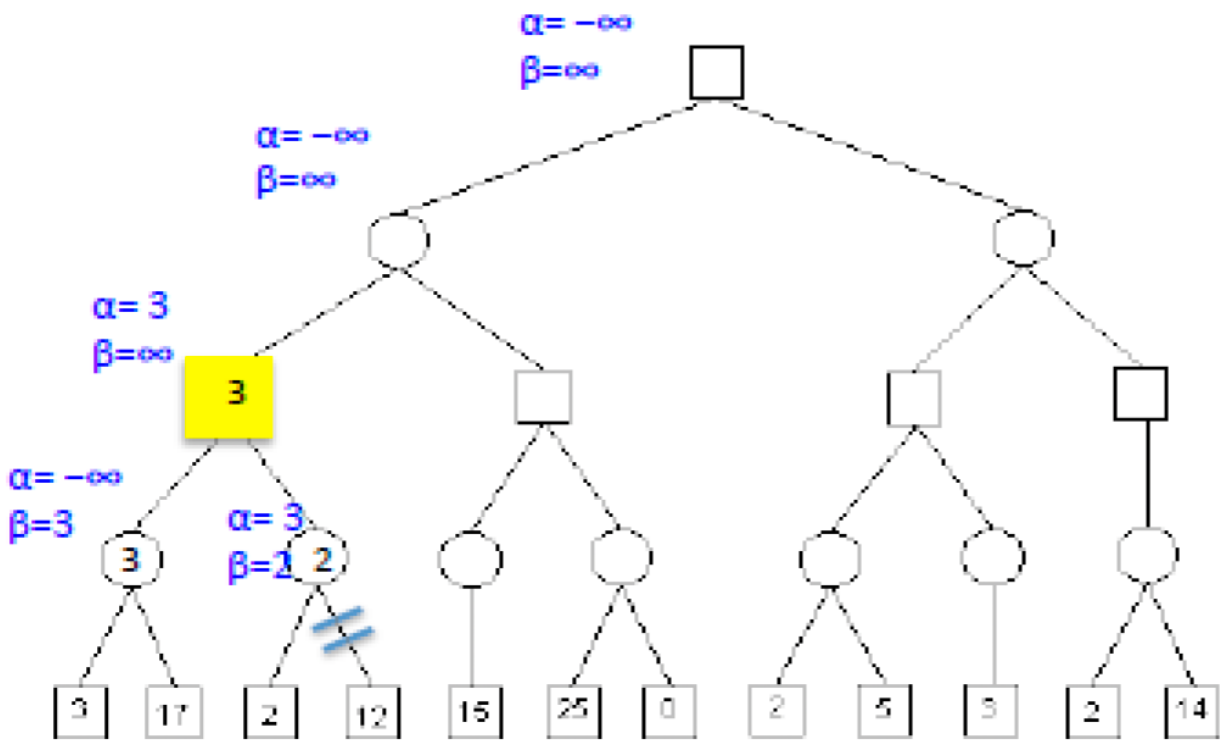
Contd…

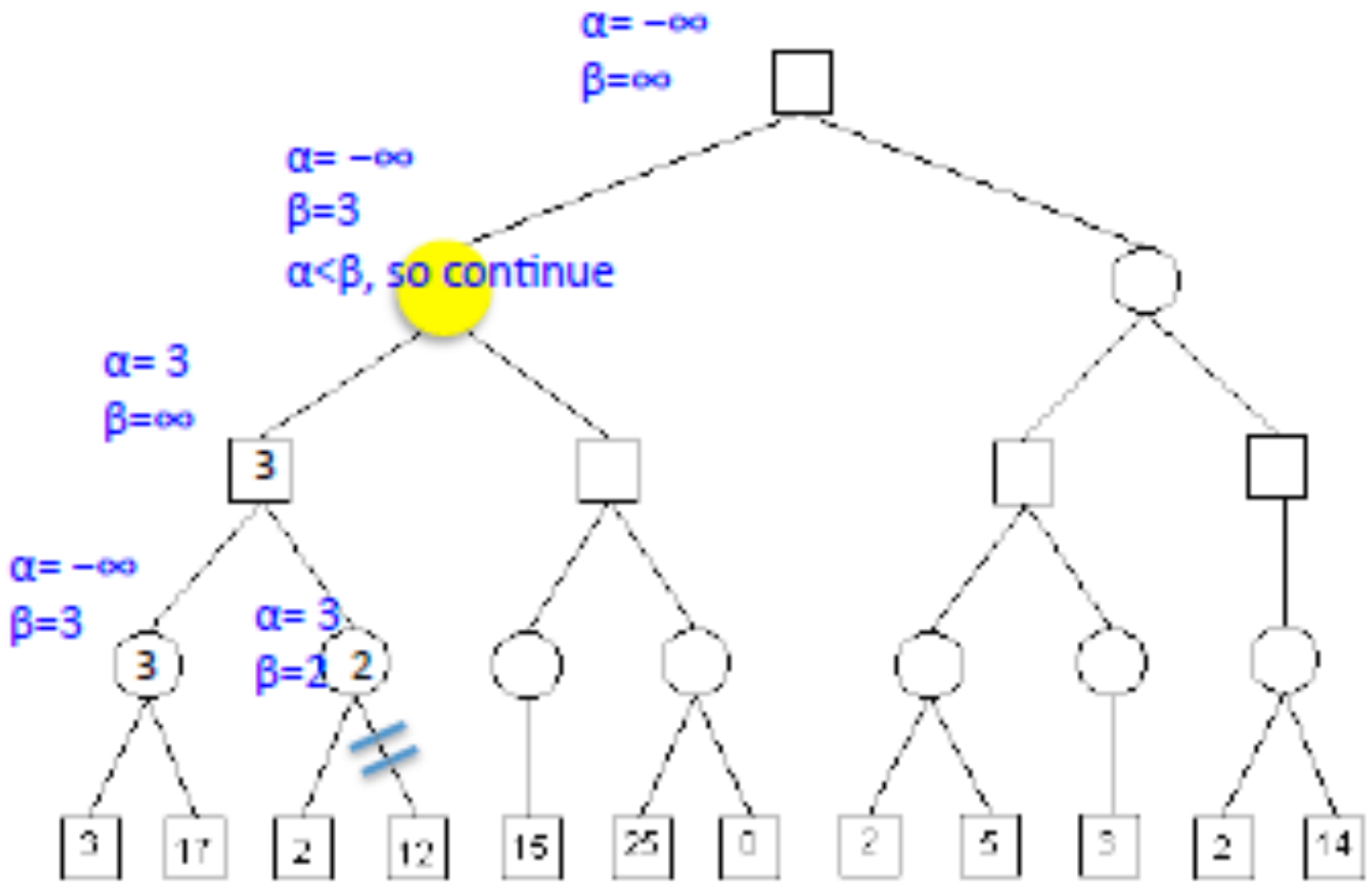


Contd…

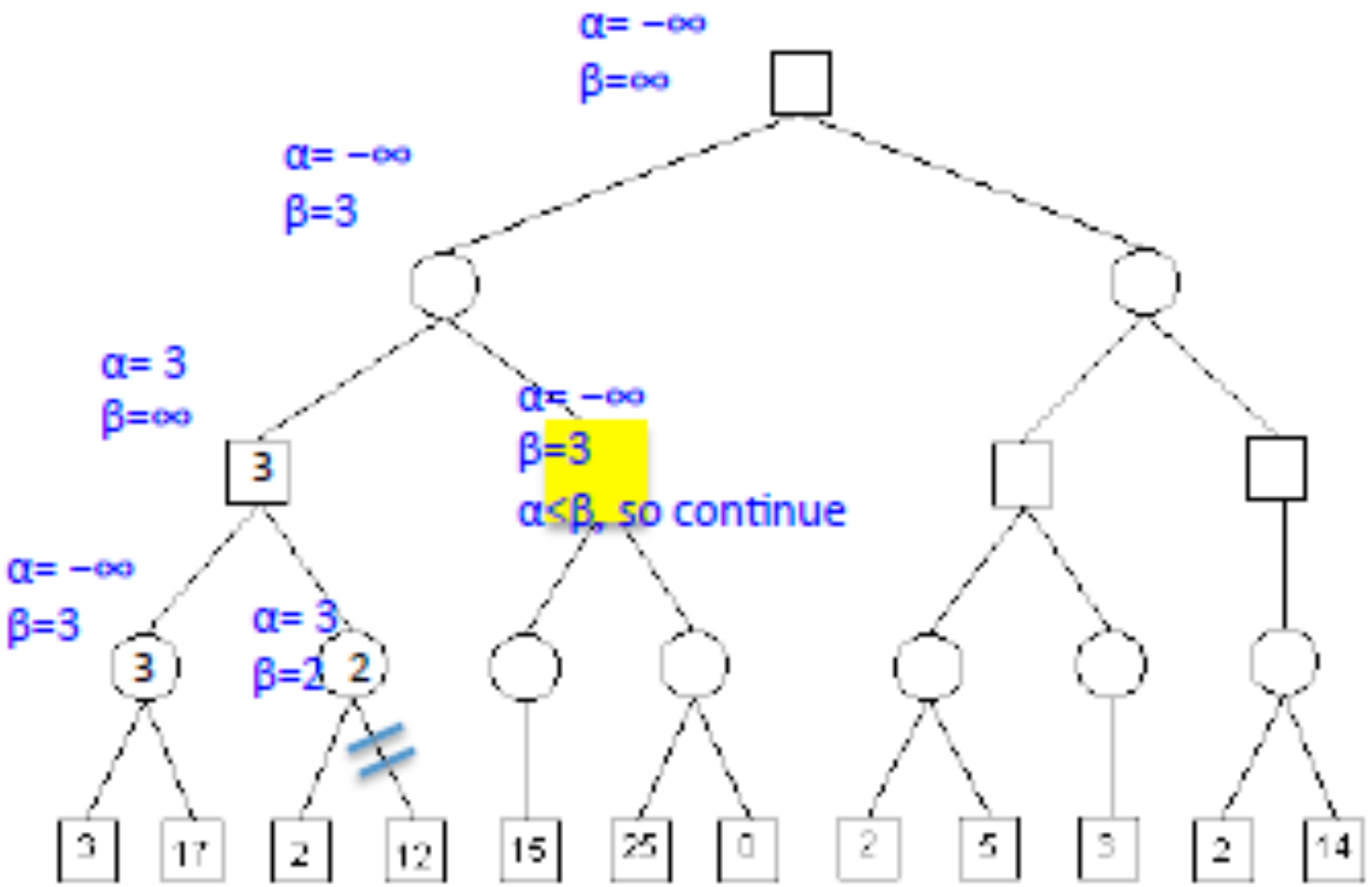


Contd…

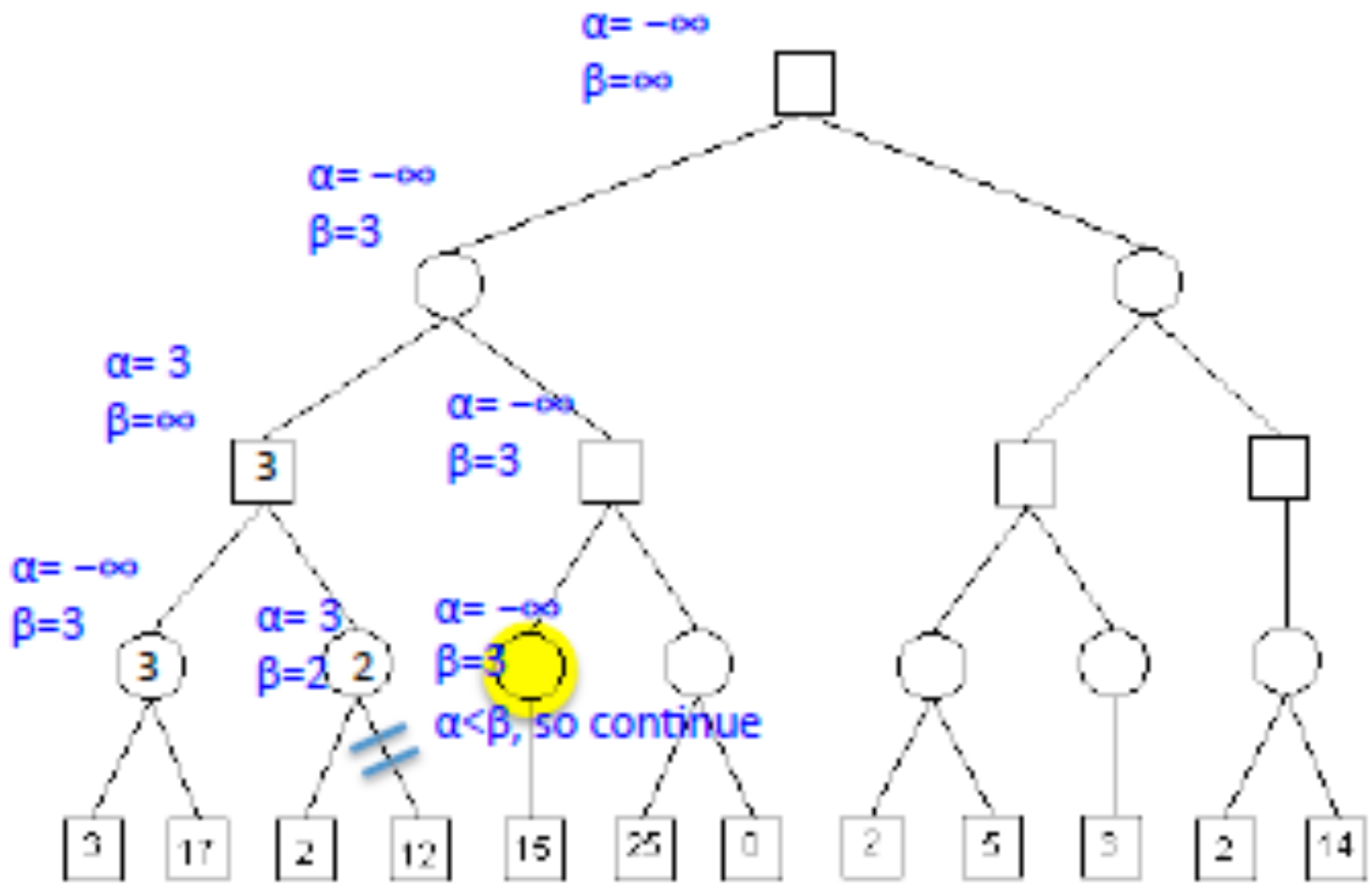


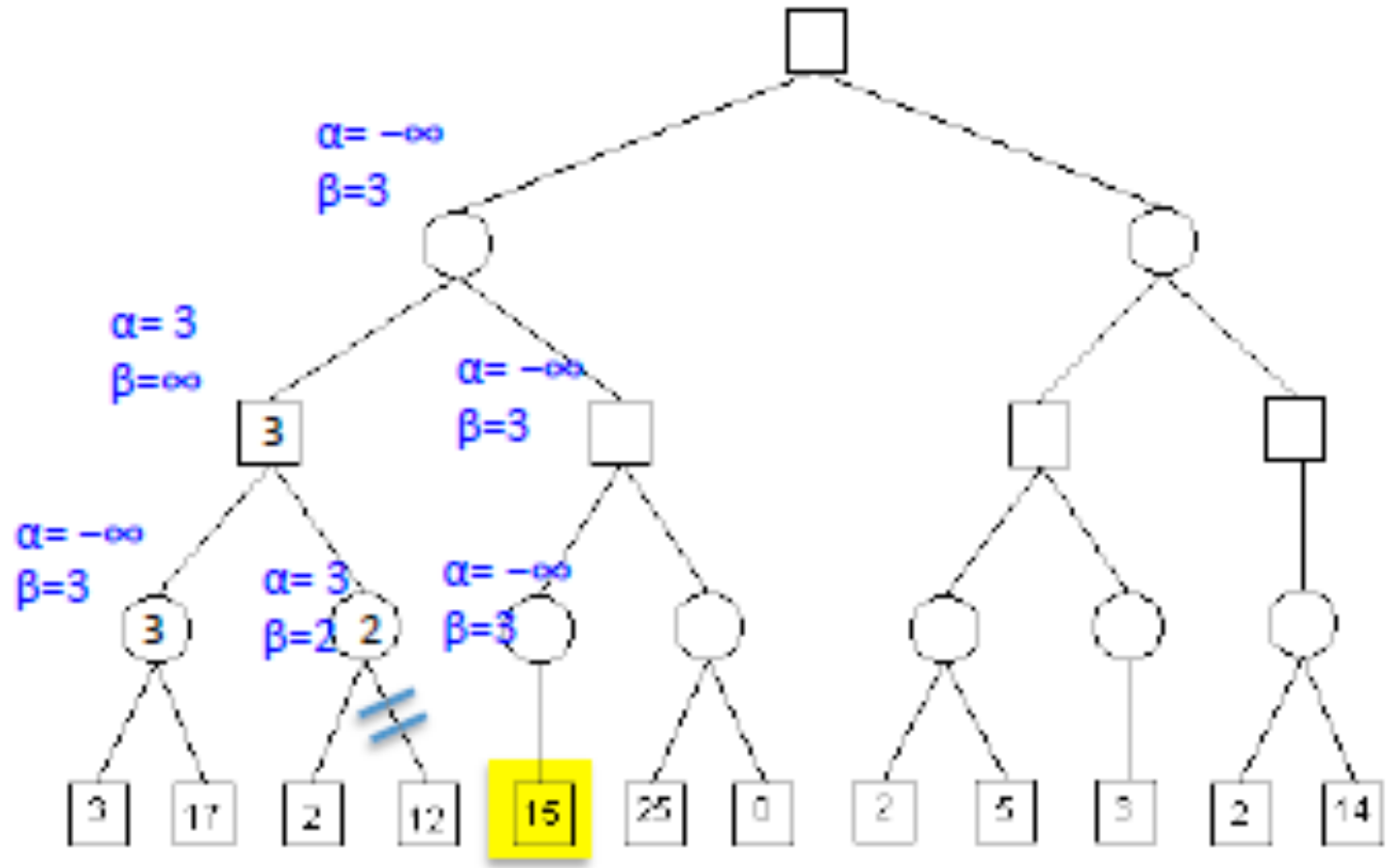


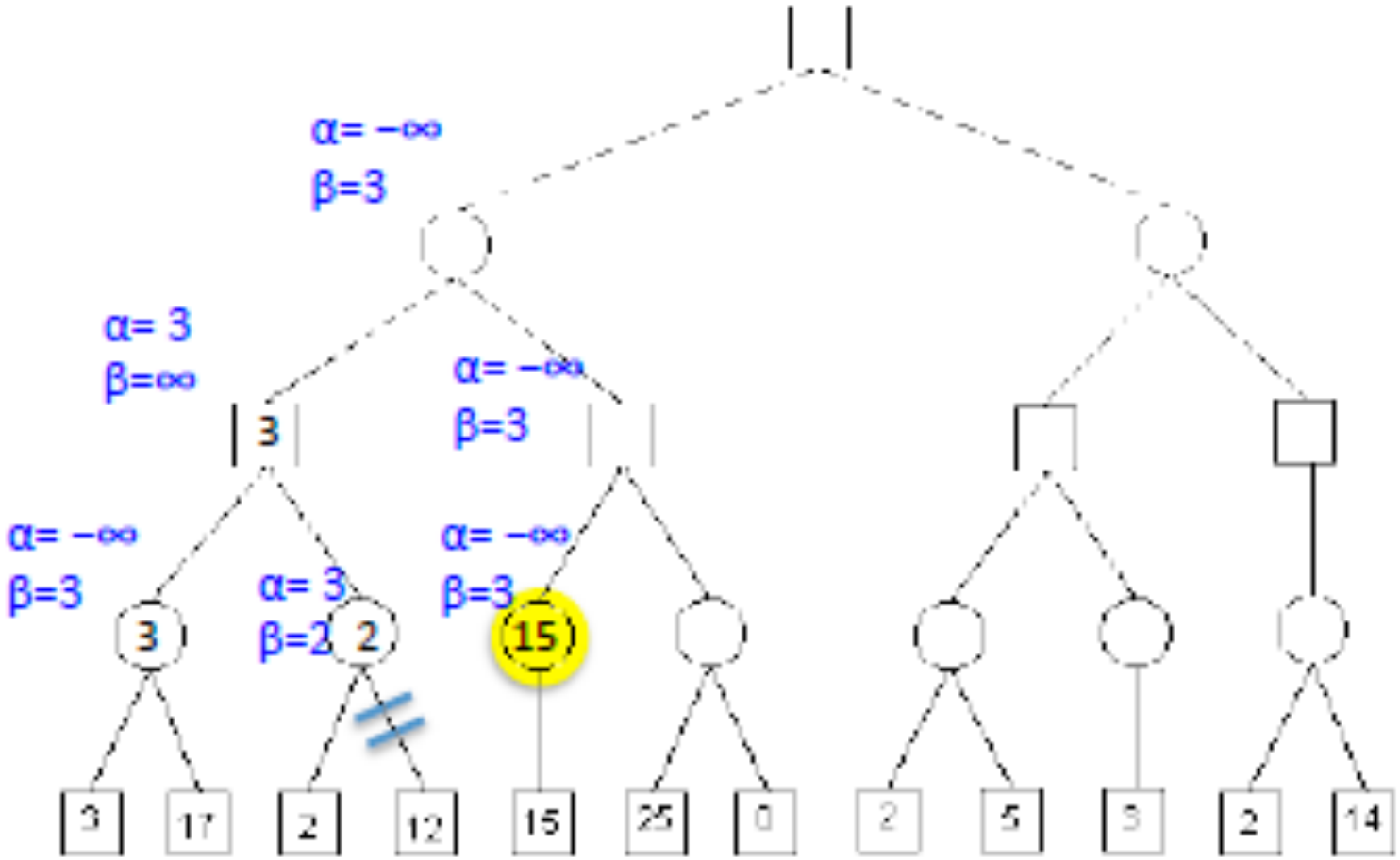
Contd..

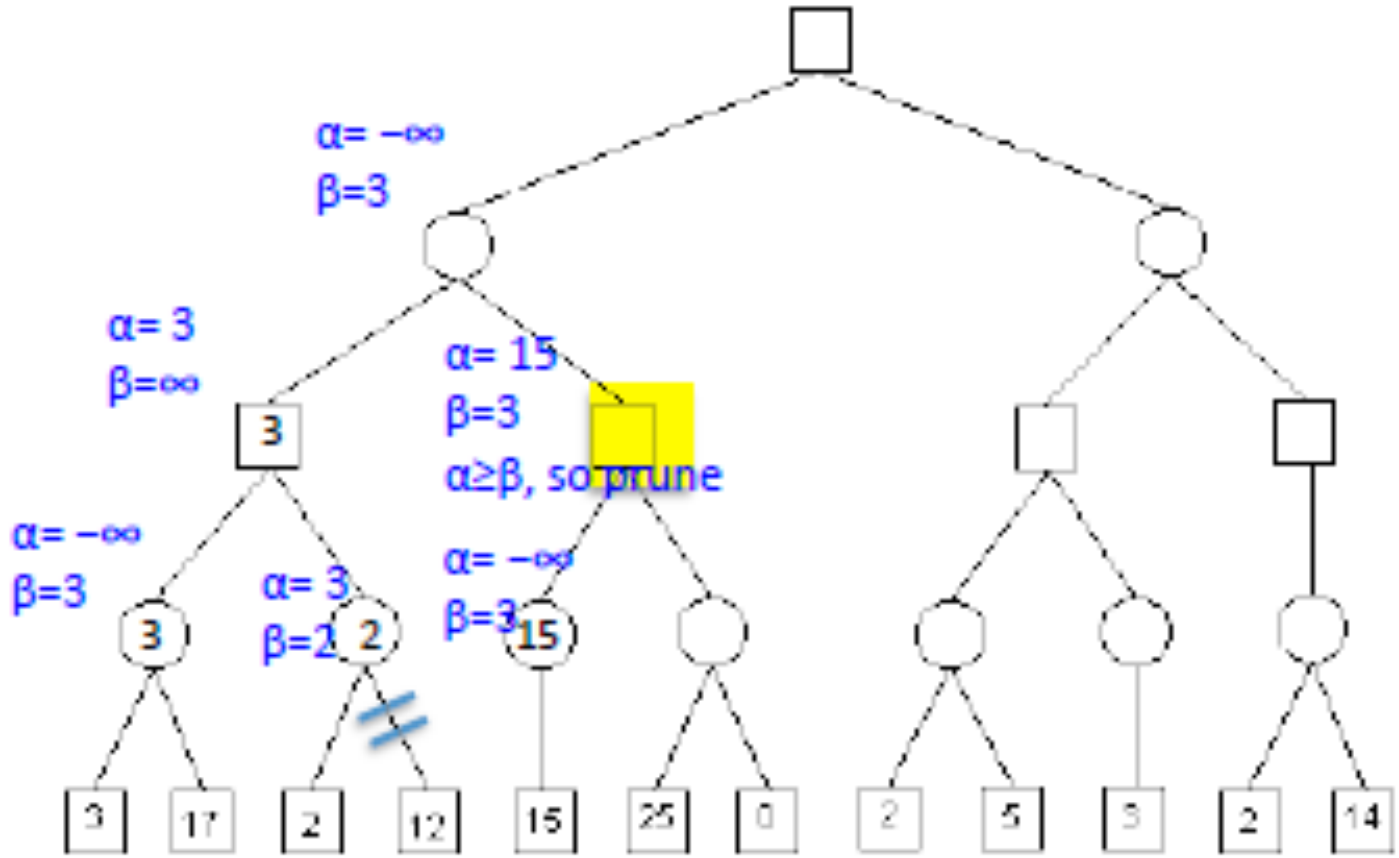

α= −∞
β=∞
α= −∞
β=3
α= 3
β=∞
3
α= −∞
β=3
α<β, so continue
α= −∞
β=3
3
α= 3
β=2
2
3
17
2
12
15
25
0
2
5
3
2
14

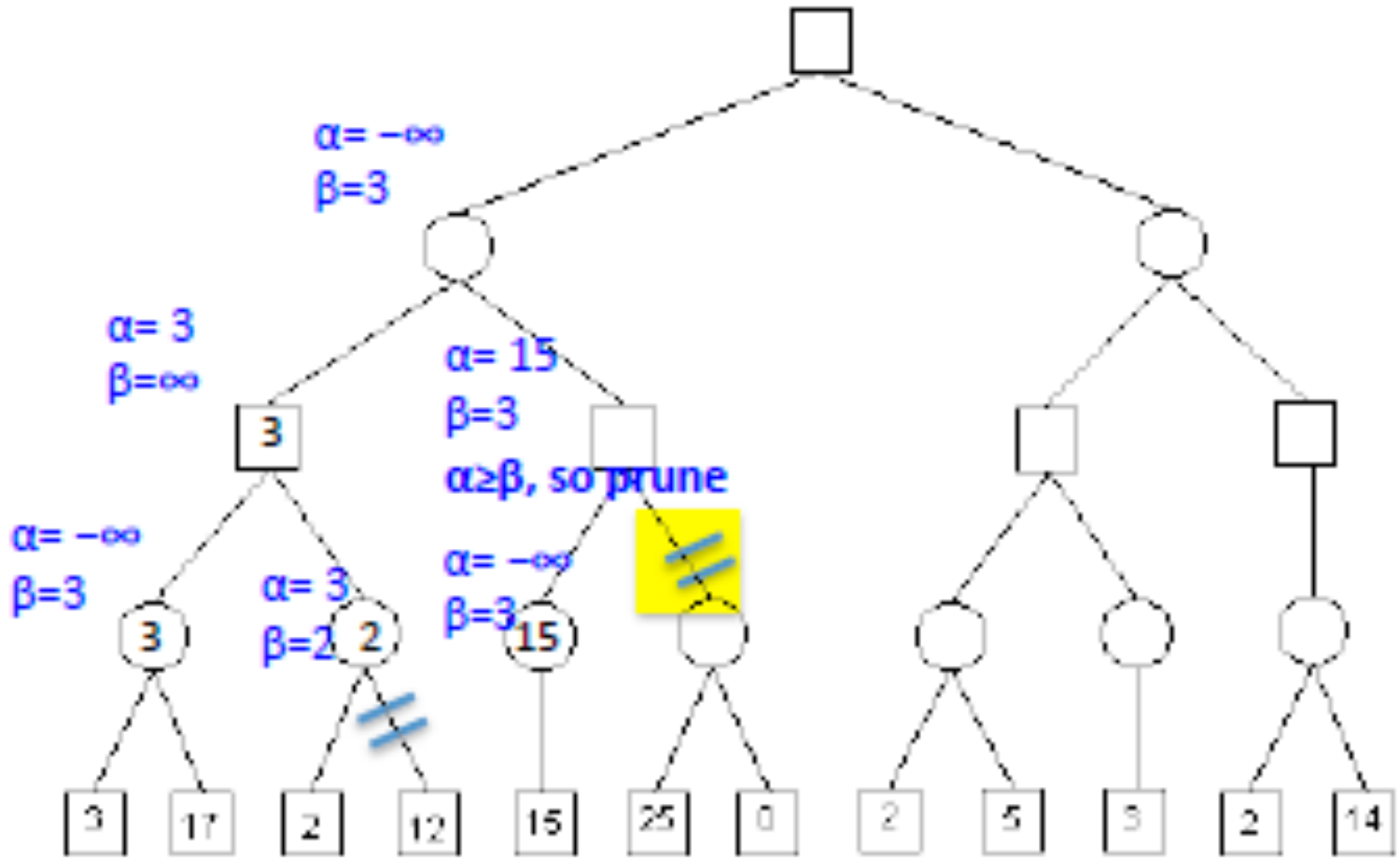
Contd..

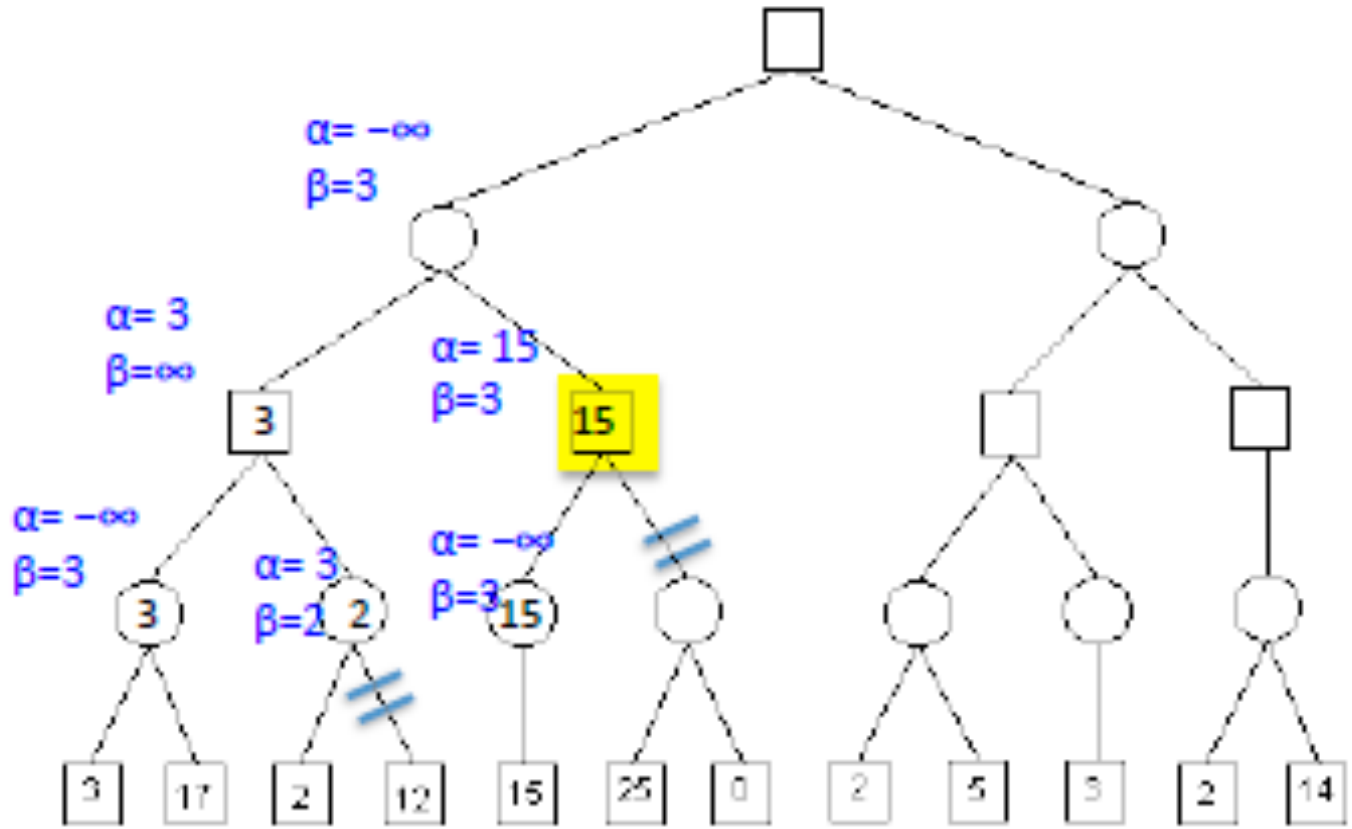

α= −∞
β=∞
α= −∞
β=3
α= 3
β=∞
α= −∞
β=3
3
α= −∞
β=3
α= 3
β=2
3
2
α= −∞
β=3
α<β, so continue
3
17
2
12
15
25
0
2
5
3
2
14

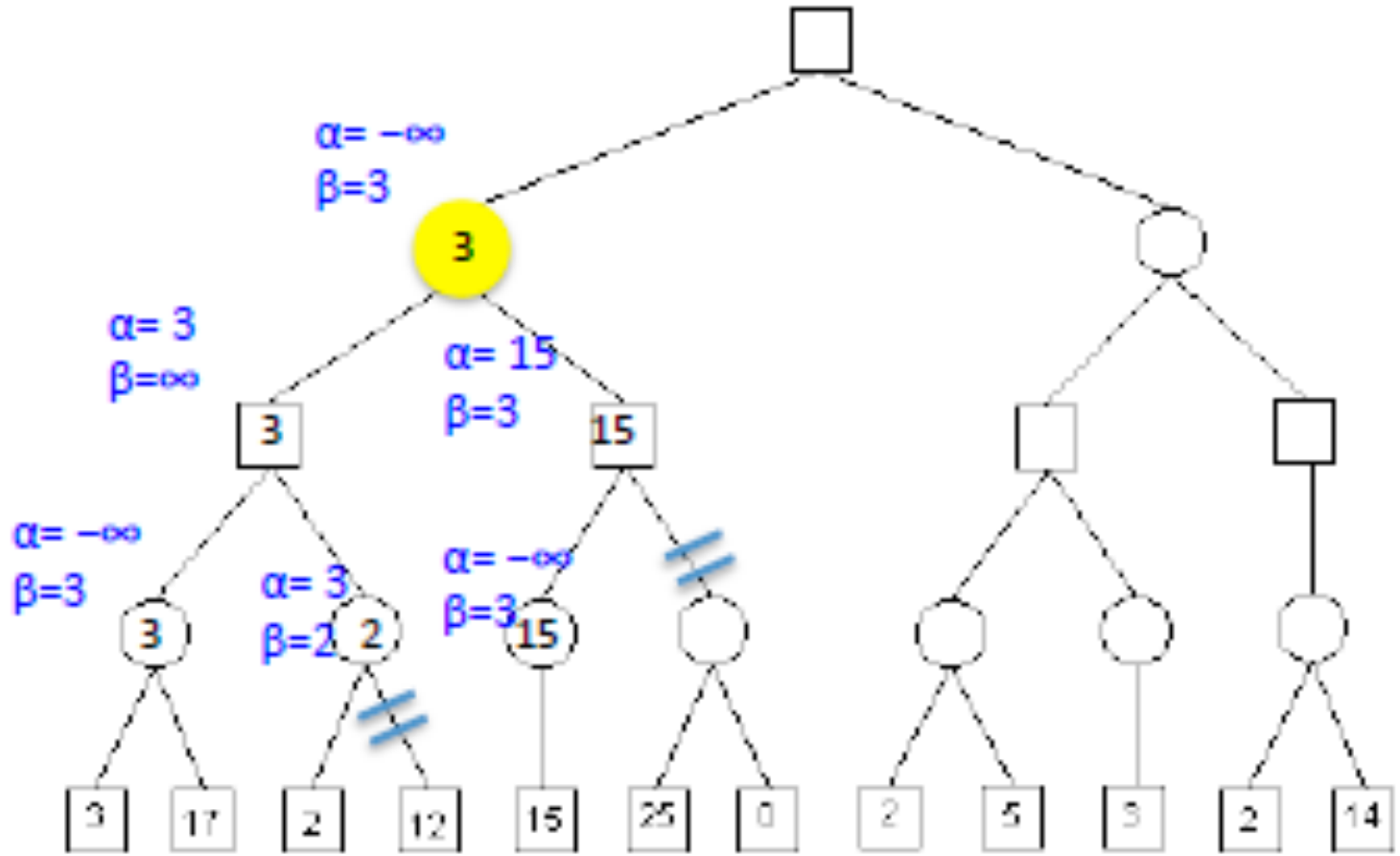

α= −∞
β=3
α= 3
β=∞
α= −∞
β=3
3
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
3
17
2
12
15
25
0
2
5
9
2
14

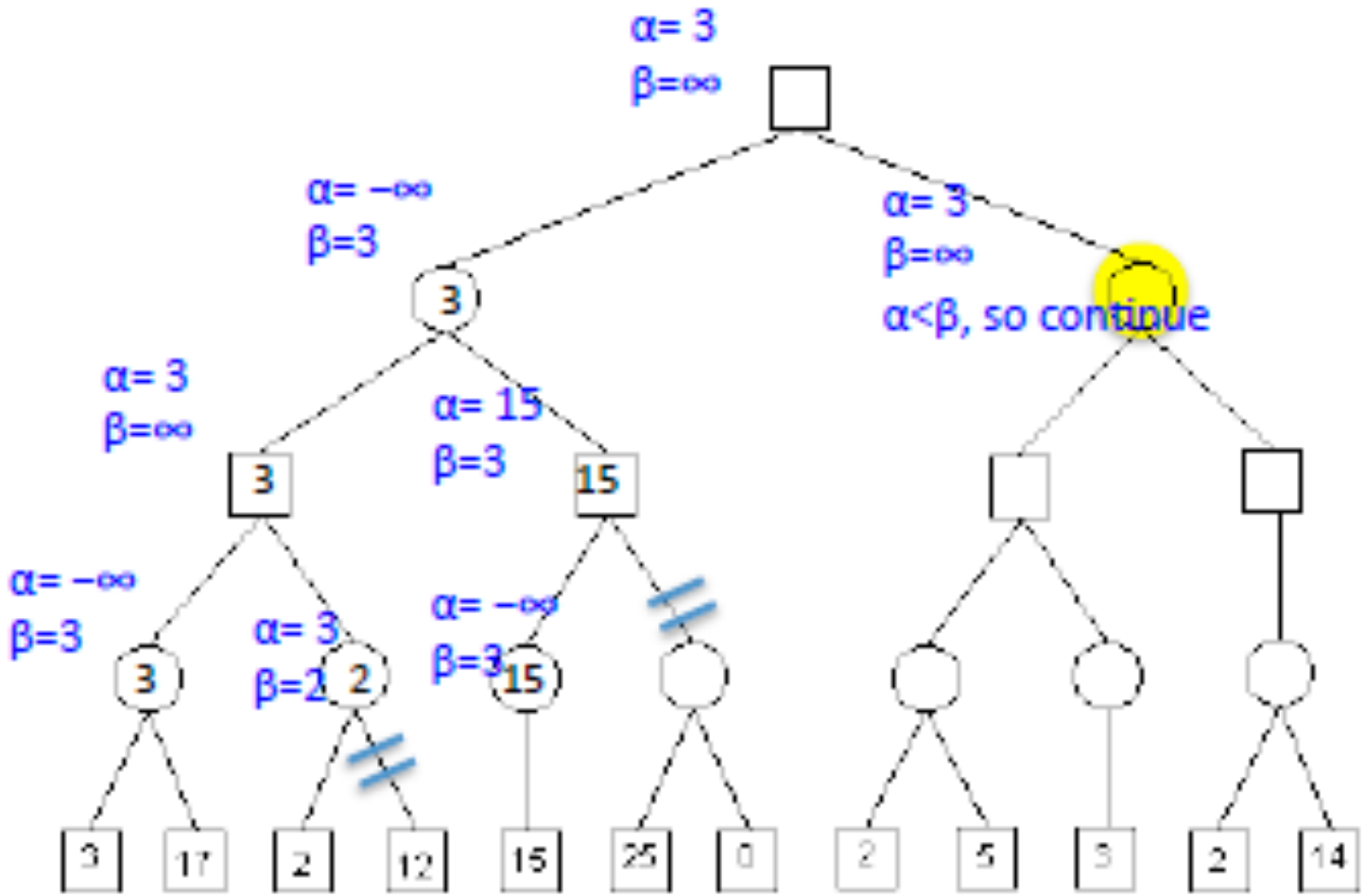

α= −∞
β=3
α= 3
β=∞
3
α= −∞
β=3
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
3
17
2
12
15
25
0
2
5
3
2
14

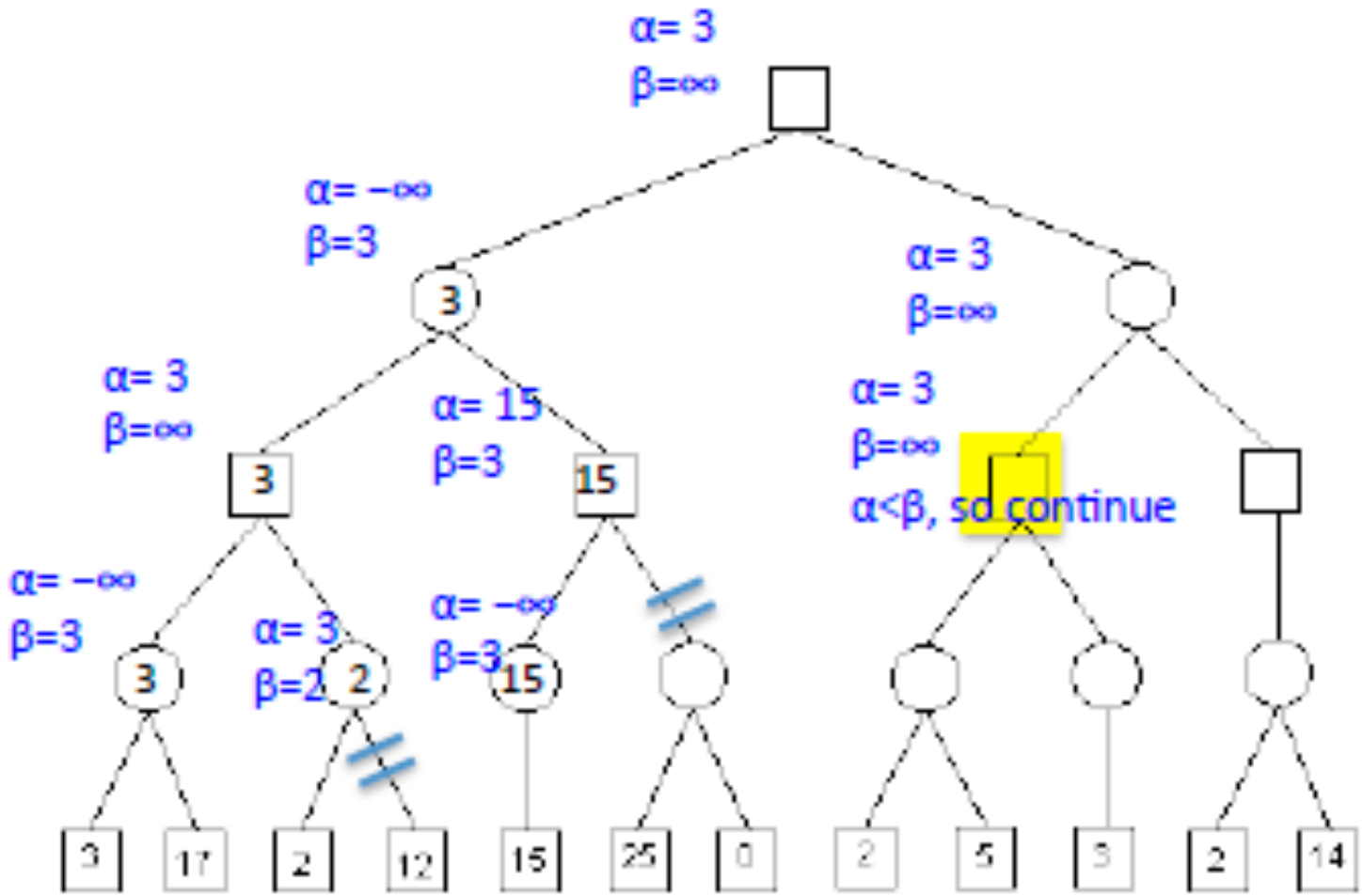

α= −∞
β=3
α= 3
β=∞
α= 15
β=3
α≥β, so prune
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
15
3
17
2
12
15
25
0
2
5
3
2
14

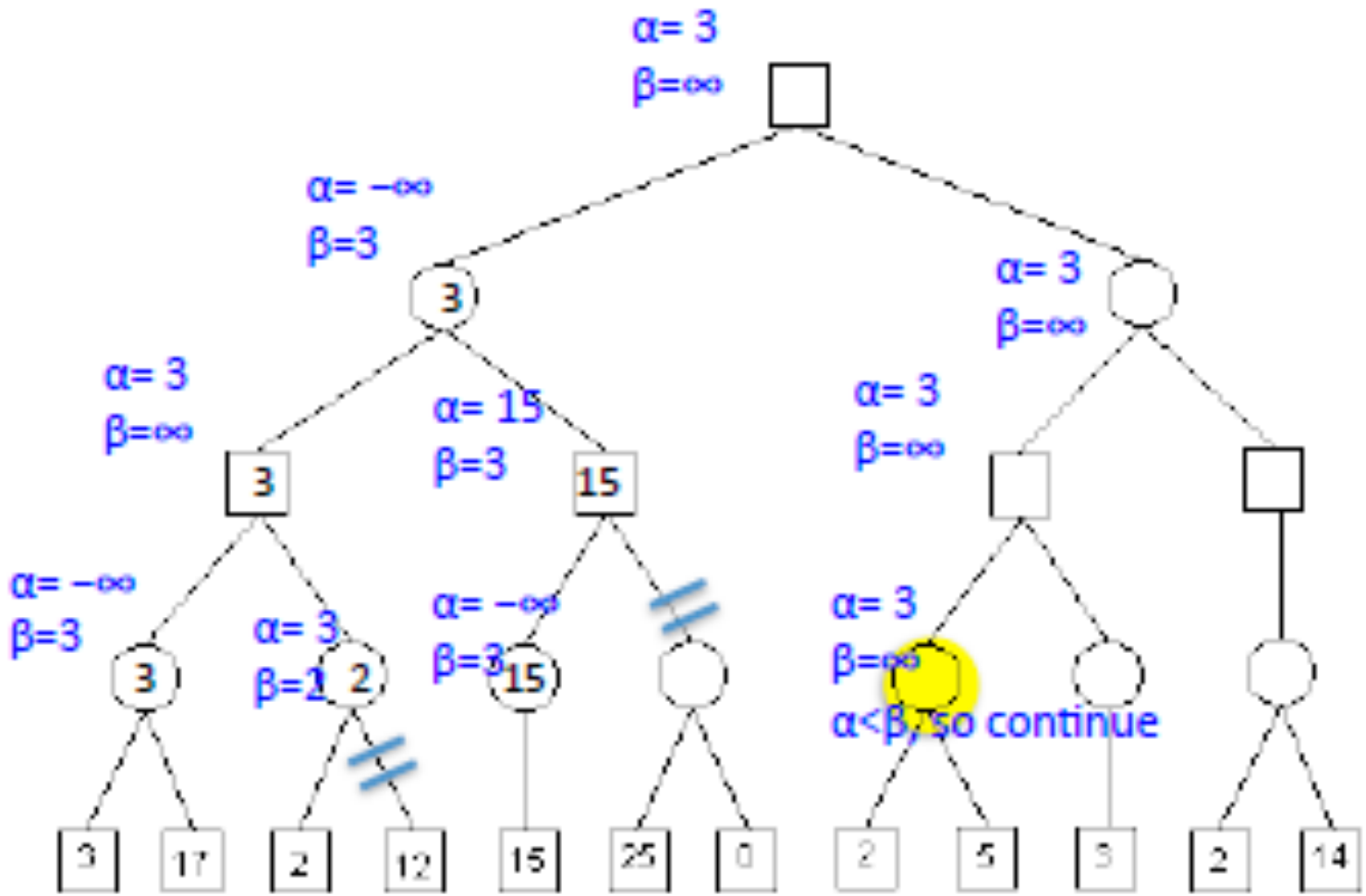

α= −∞
β=3
α= 3
β=∞
α= 15
β=3
α≥β, so prune
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
15
3
17
2
12
15
25
0
2
5
3
2
14

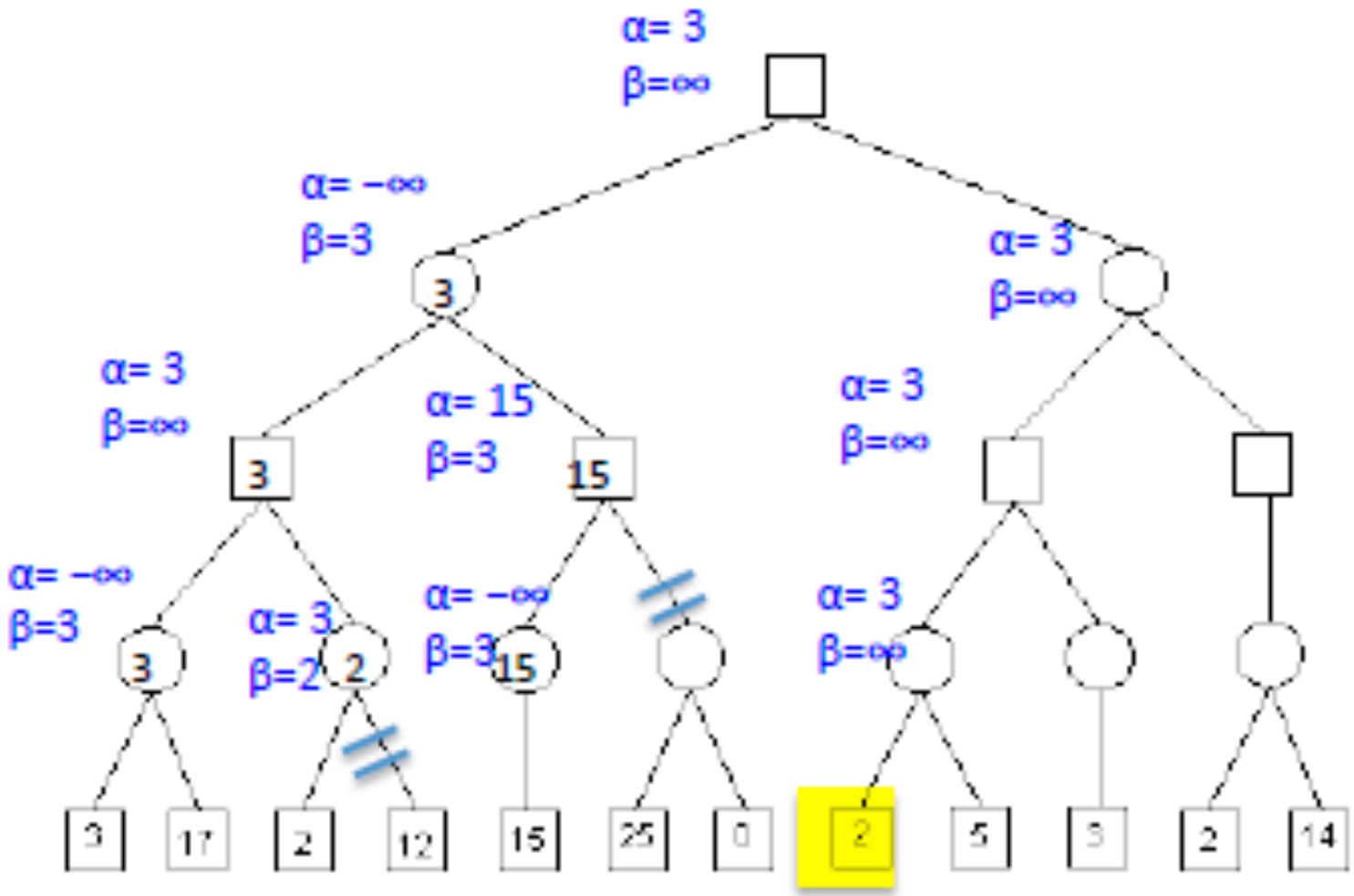

α= −∞
β=3
α= 3
β=∞
α= 15
β=3
3
15
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
15
3
17
2
12
15
25
0
2
5
8
2
14

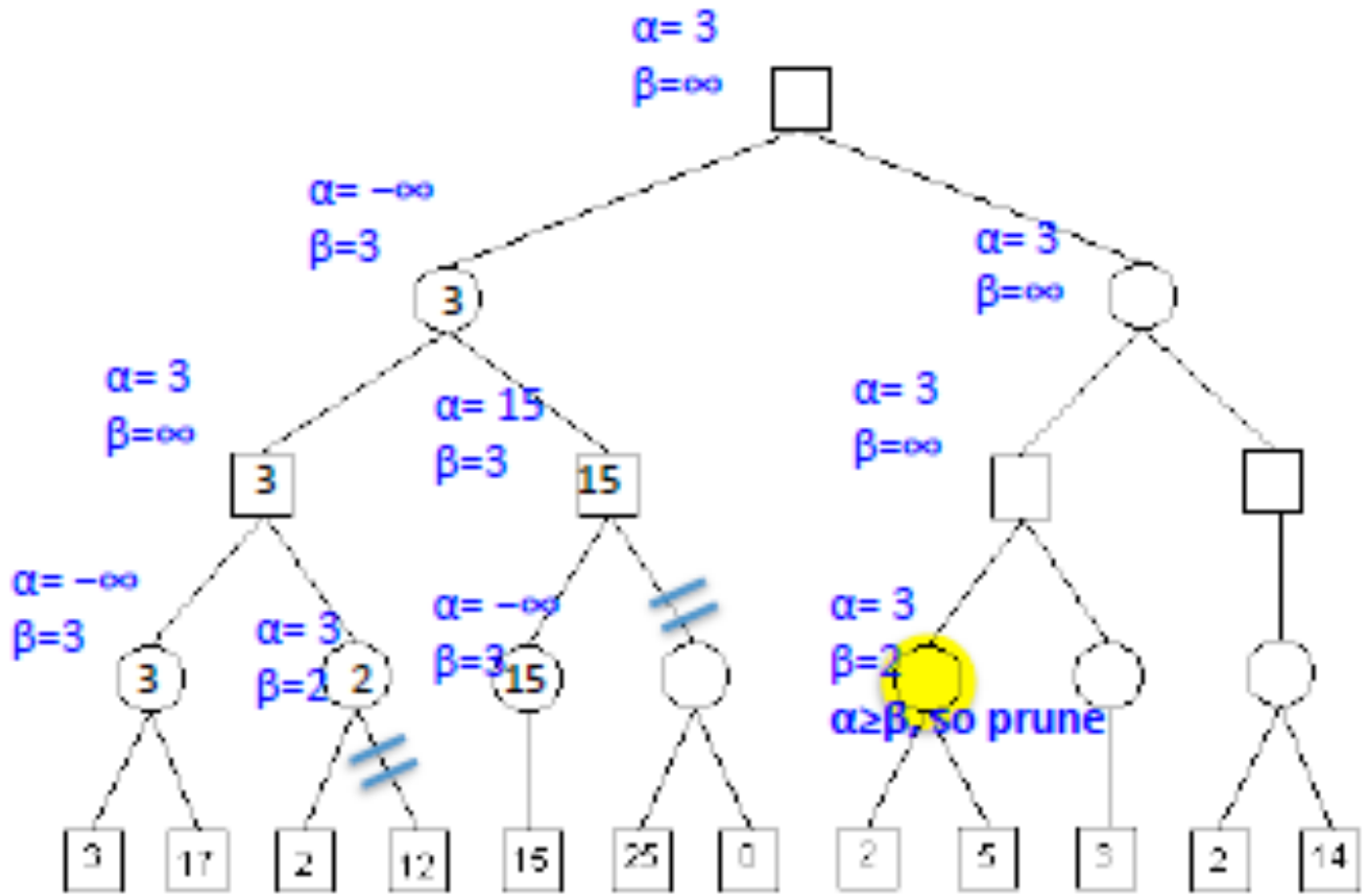

α= −∞
β=3
3
α= 3
β=∞
3
α= 15
β=3
15
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
3
17
2
12
15
25
0
2
5
3
2
14

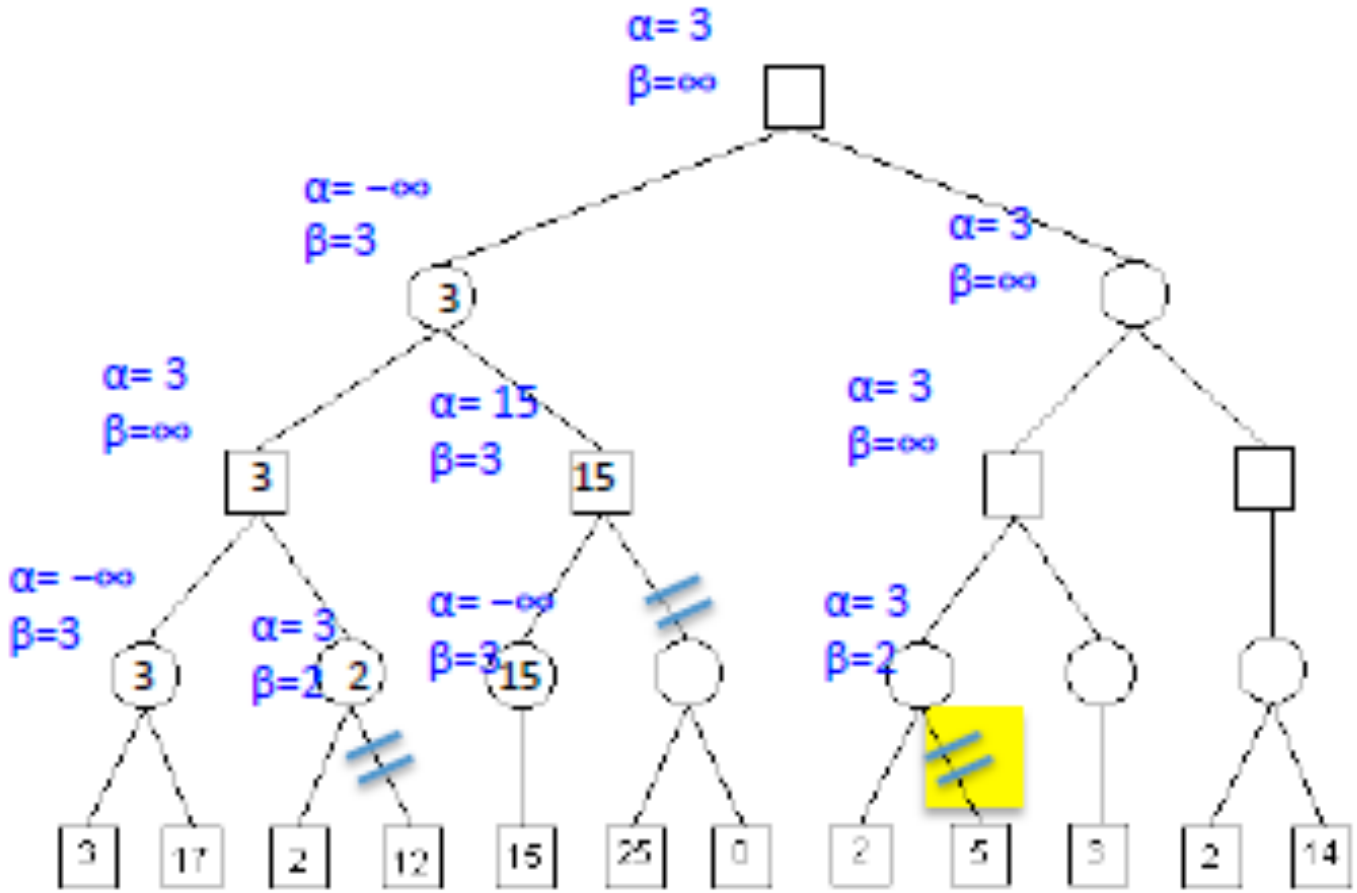

α= 3
β=∞
α= −∞
β=3
α= 3
β=∞
α<β, so continue
3
α= 3
β=∞
α= 15
β=3
3
15
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
15
3
17
2
12
15
25
0
2
5
3
2
14

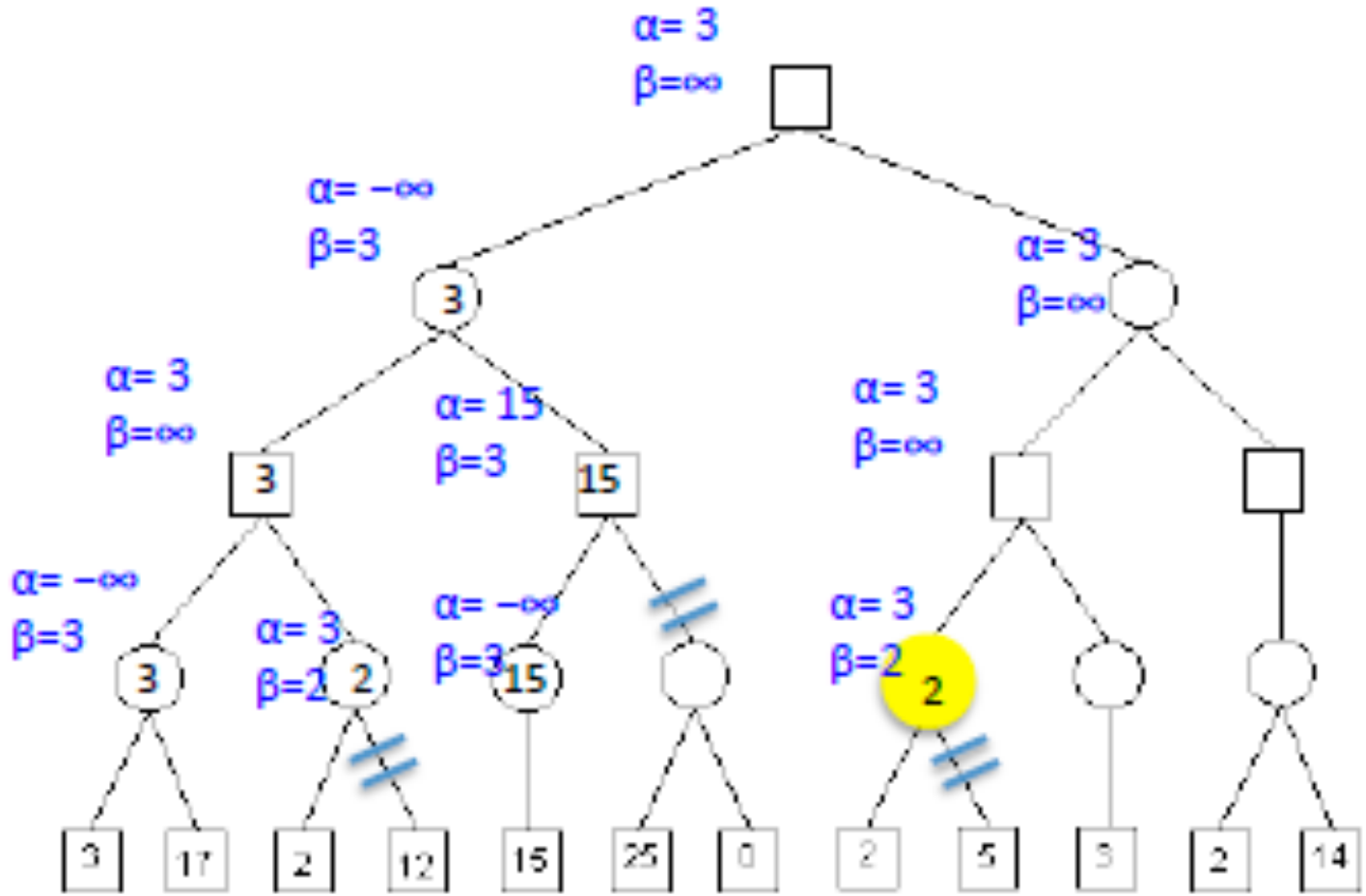

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
α<β, so continue
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
3
2
15
3
17
2
12
15
25
0
2
5
3
2
14

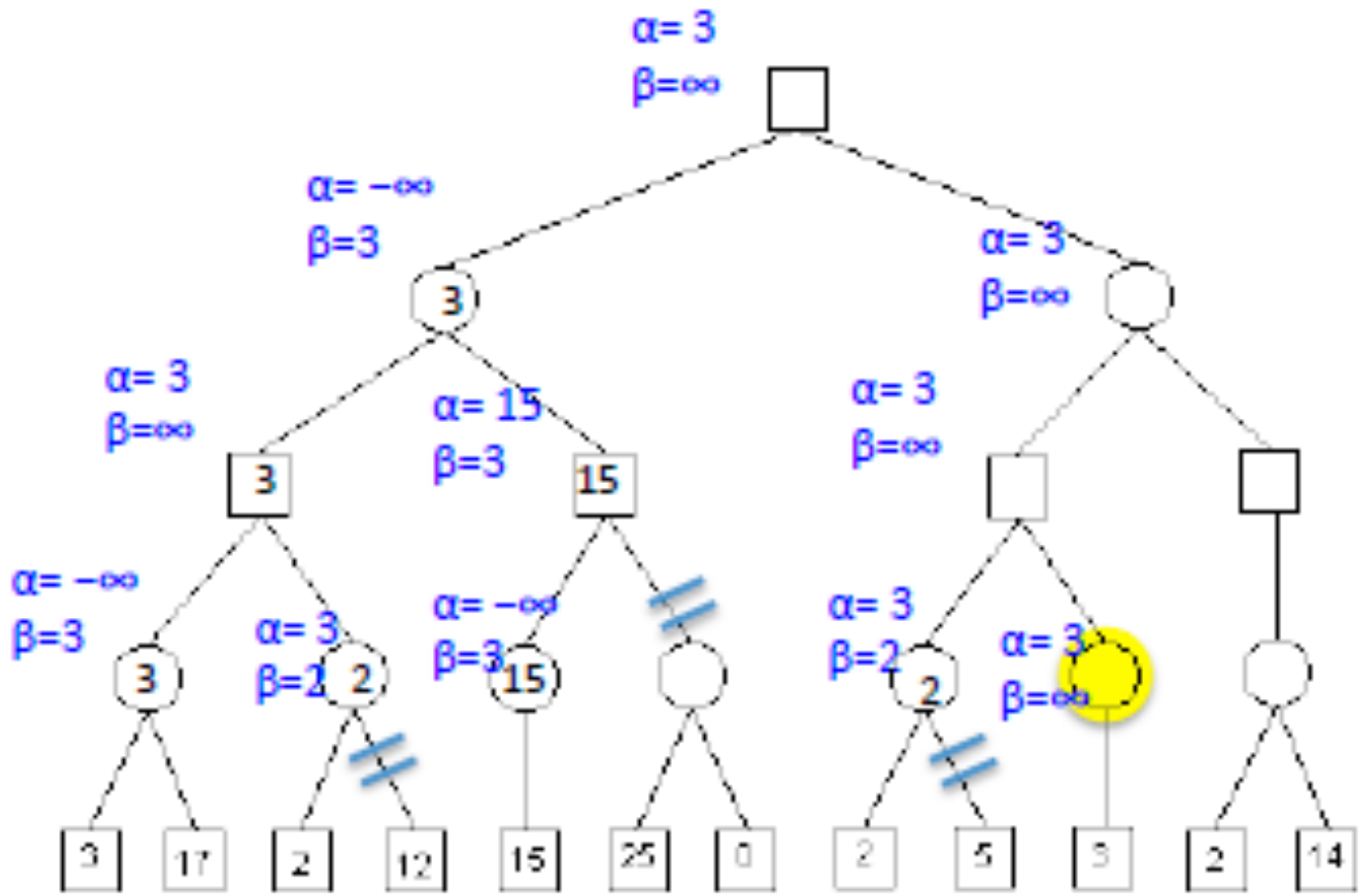

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
α= 15
β=3
3
15
α= 3
β=∞
α= −∞
β=3
α= 3
β=1
α= −∞
β=3
3
2
15
α= 3
β=∞
α<β, so continue
3
17
2
12
15
25
0
2
5
3
2
14

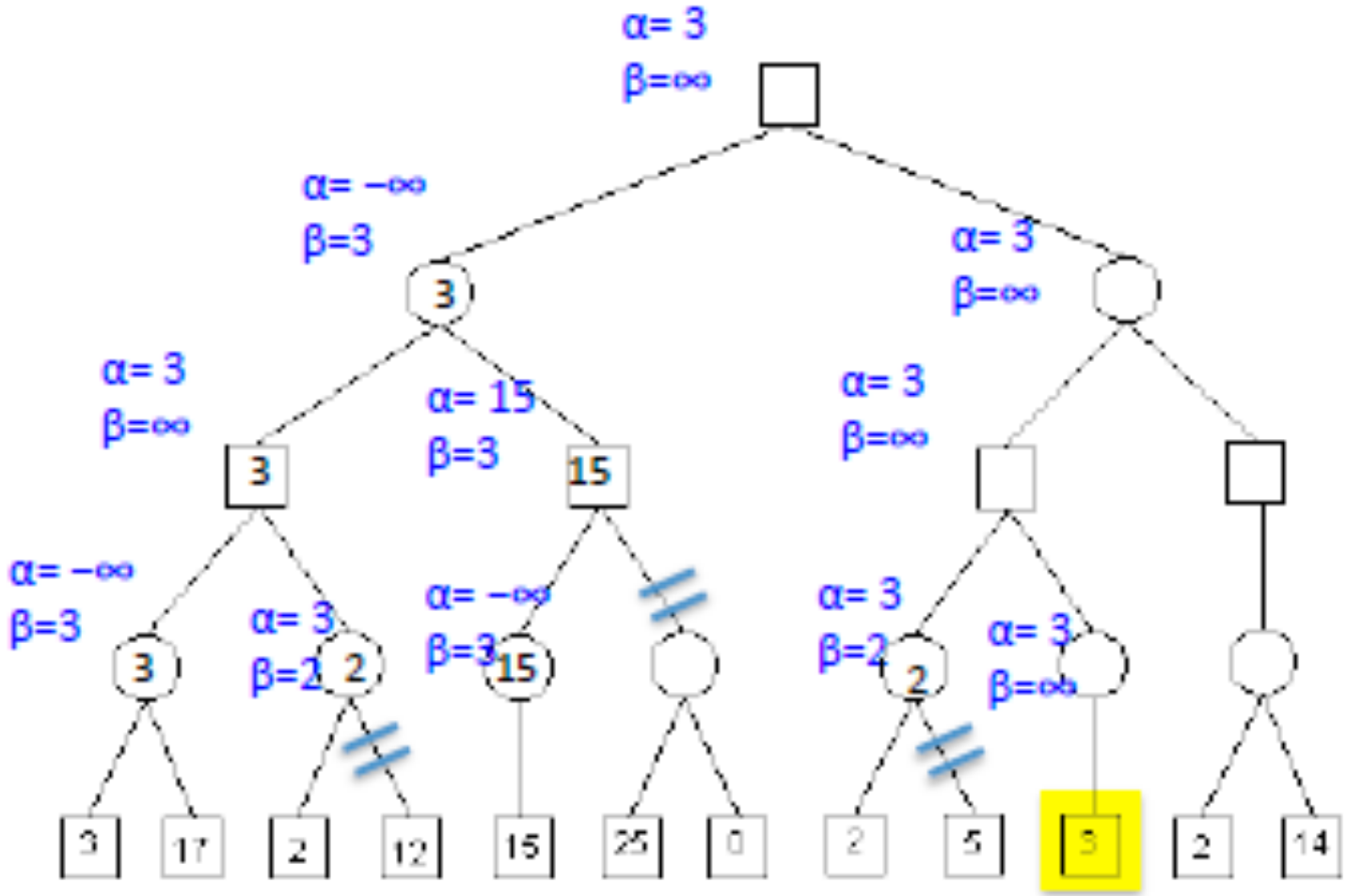

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
α= 3
β=∞
α= 15
β=3
3
15
α= −∞
β=3
α= 3
β=2
α= −∞
β=3
α= 3
β=∞
3
2
15
3
17
2
12
15
25
0
2
5
3
2
14

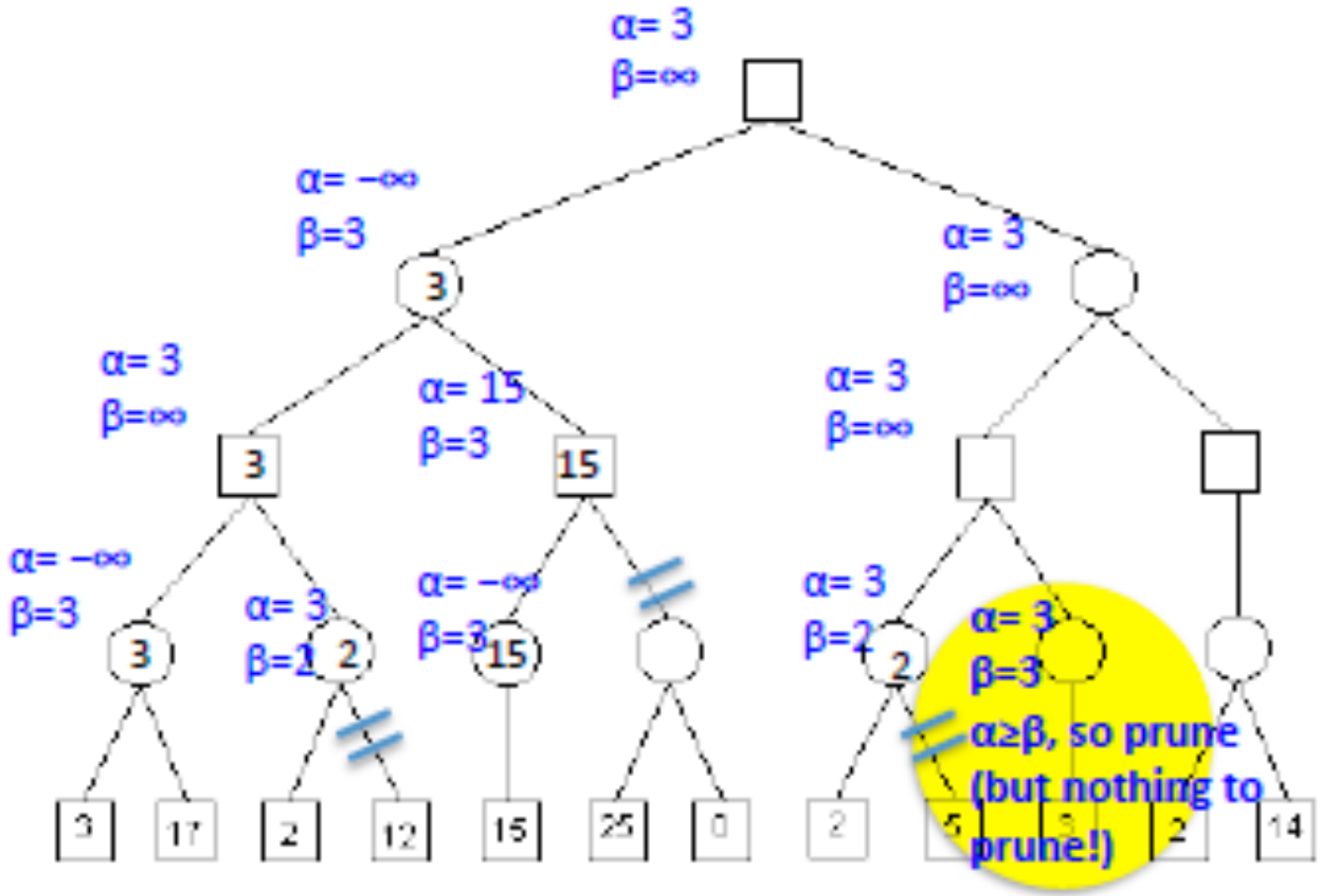

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=2
α≥β, so prune
3
17
2
12
15
25
0
2
5
3
2
14

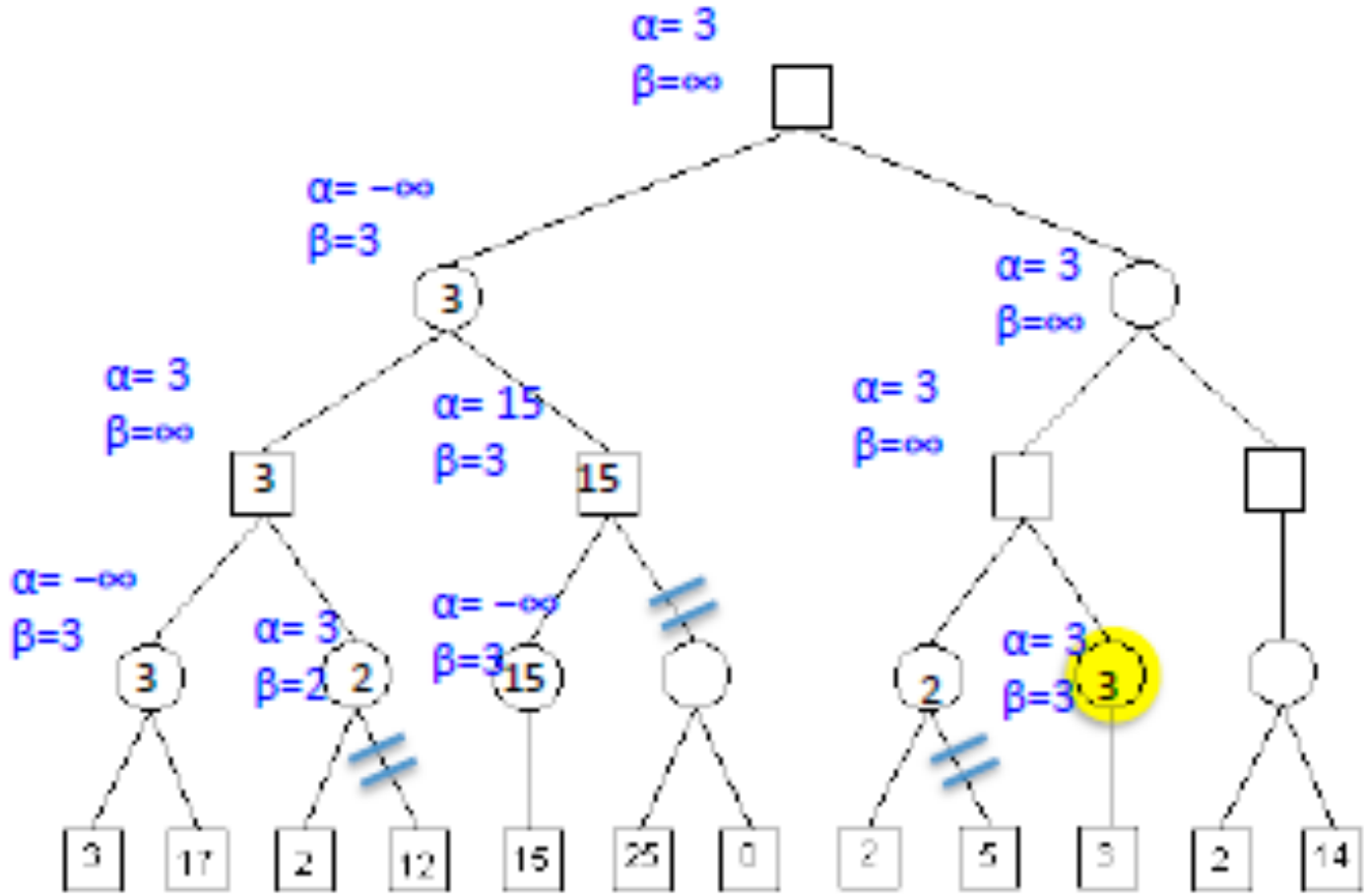

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=1
2
α= −∞
β=3
15
α= 3
β=2
3
17
2
12
15
25
0
2
5
3
2
14

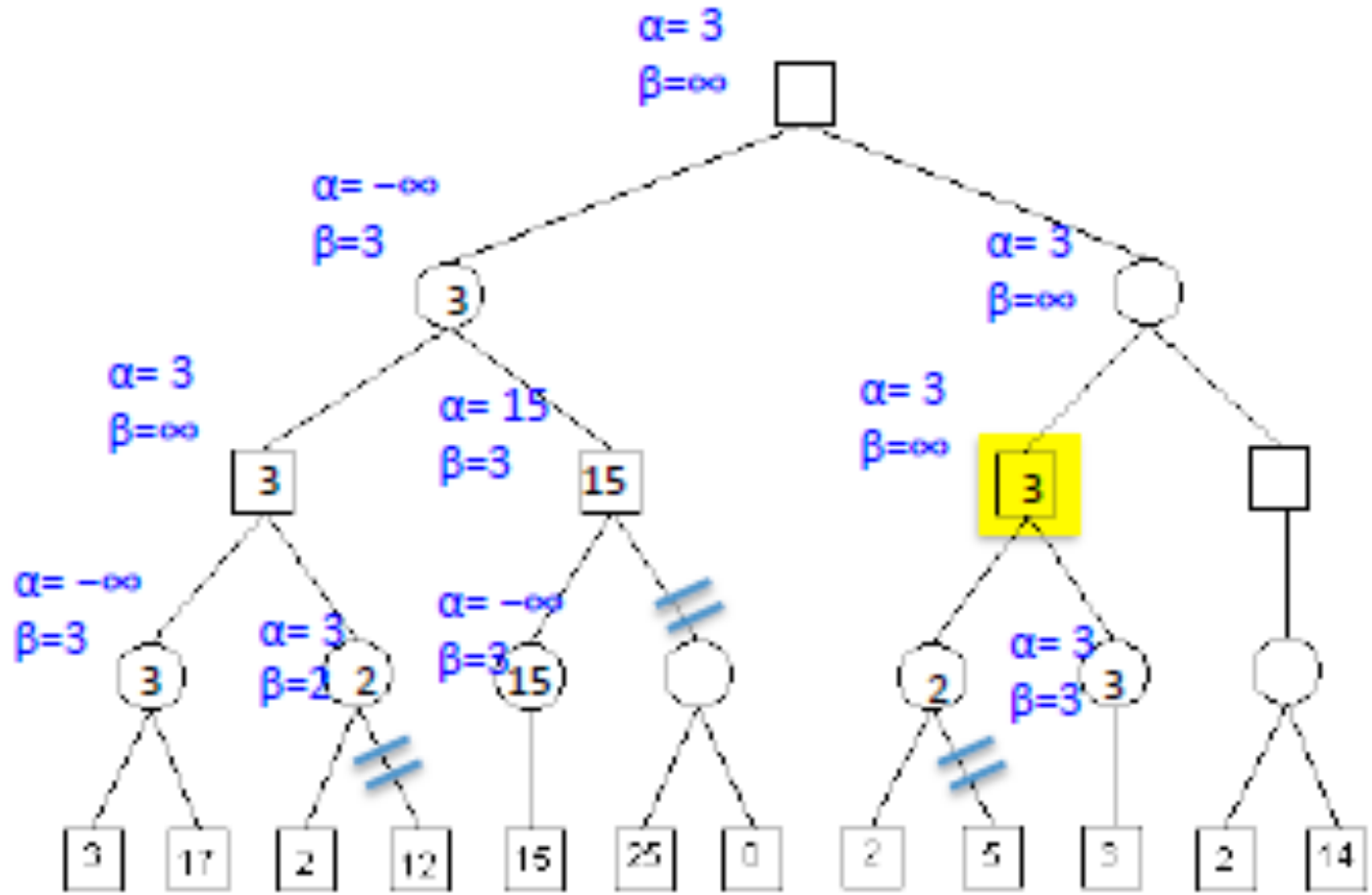

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=2
2
3
17
2
12
15
25
0
2
5
3
2
14

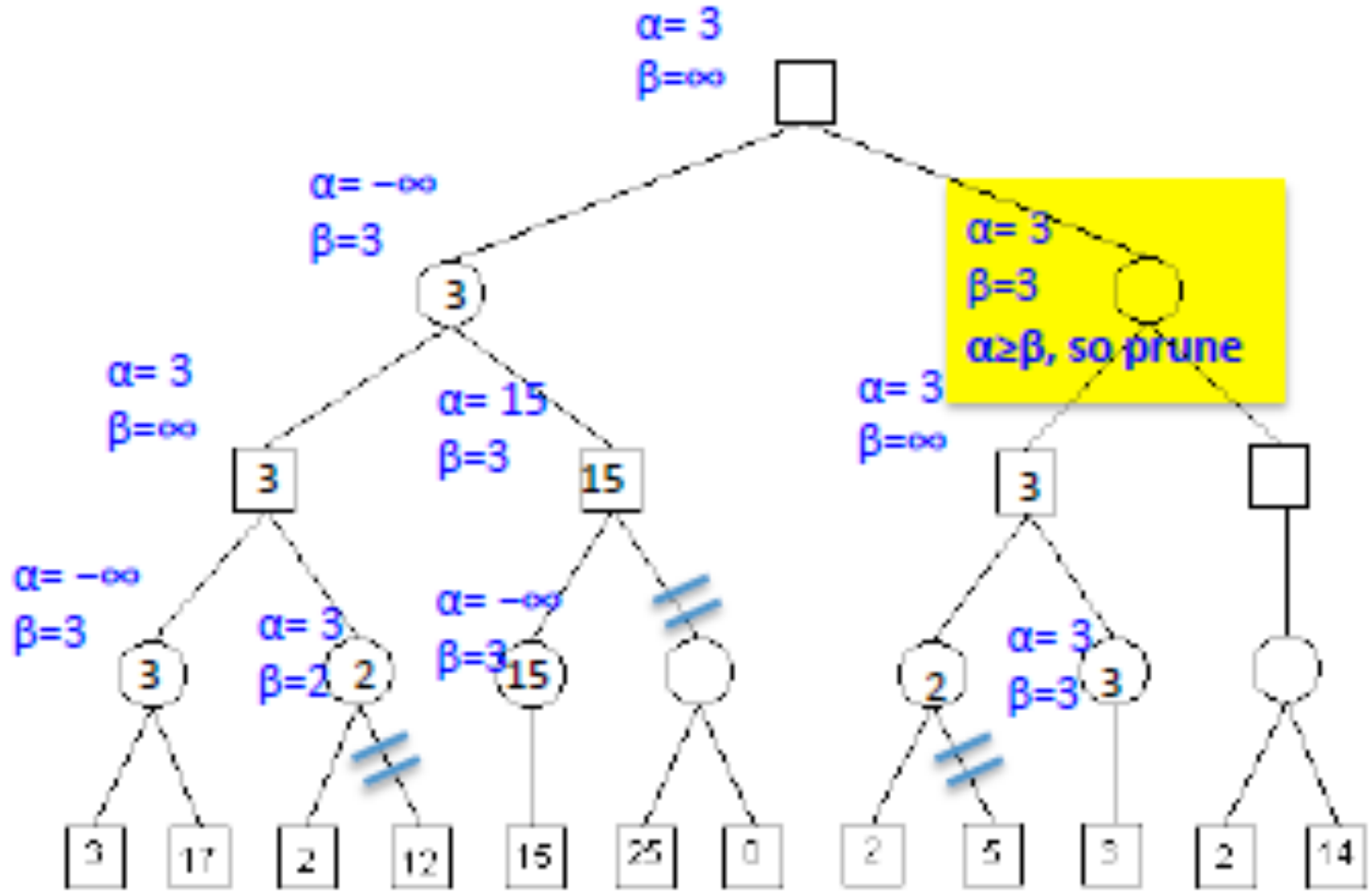

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=1
2
α= −∞
β=3
15
α= 3
β=2
2
α= 3
β=∞
3
17
2
12
15
25
0
2
5
3
2
14

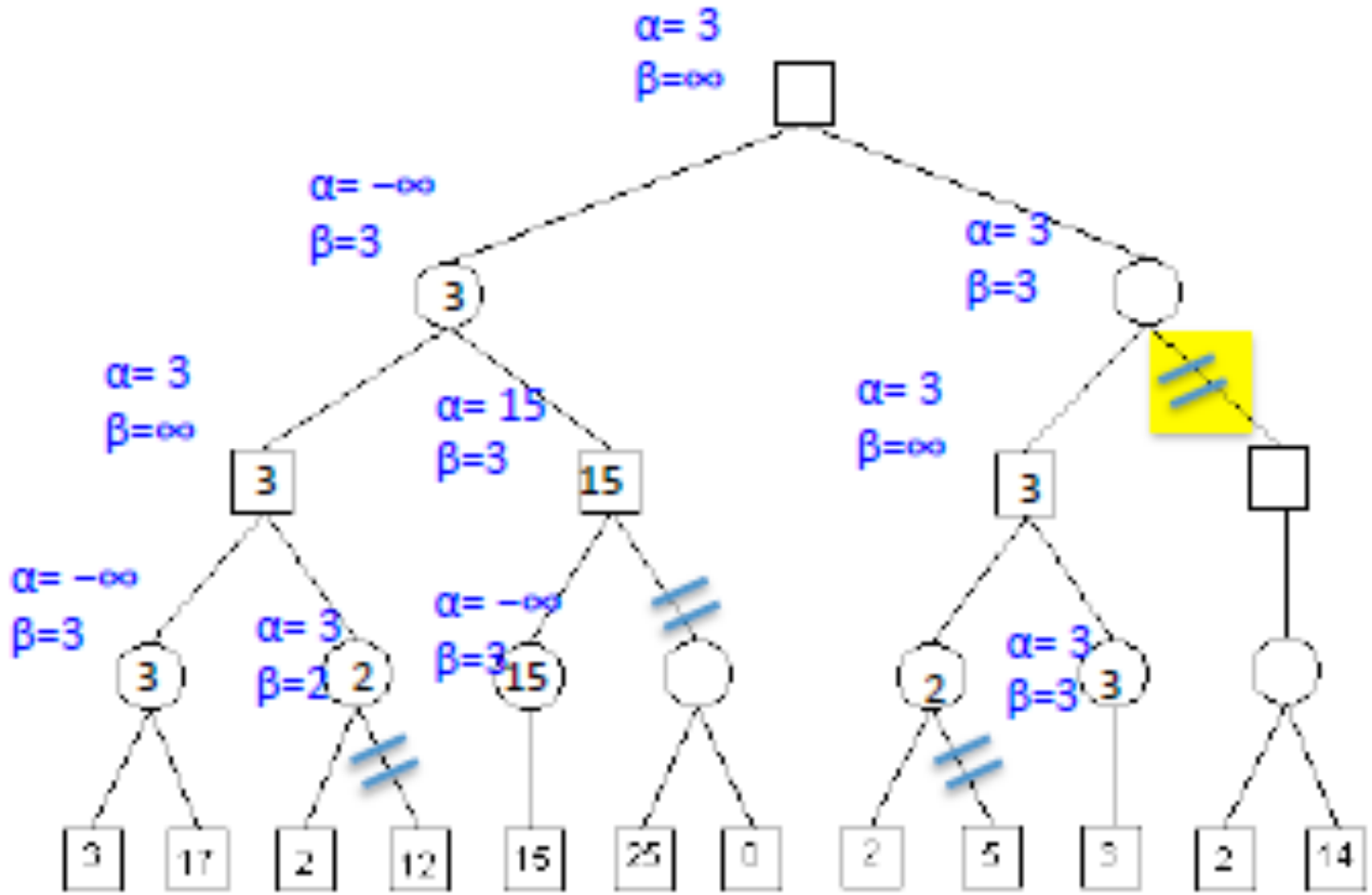

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=2
2
α= 3
β=∞
3
17
2
12
15
25
0
2
5
5
2
14

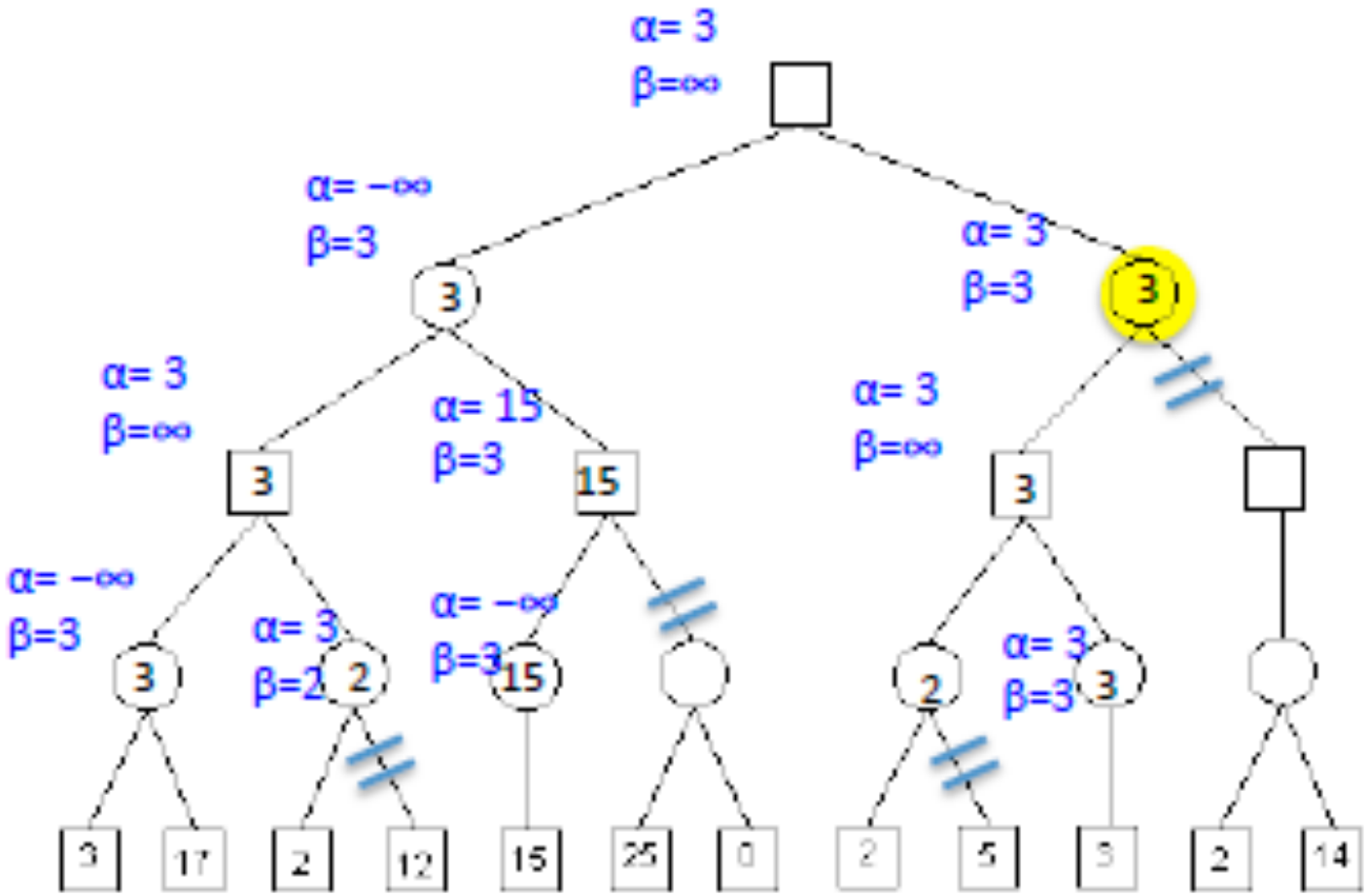

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=1
2
α= −∞
β=3
15
α= 3
β=2
2
α= 3
β=3
α≥β, so prune
(but nothing to
prune!)
3
17
2
12
15
25
0
2
5
3
2
14

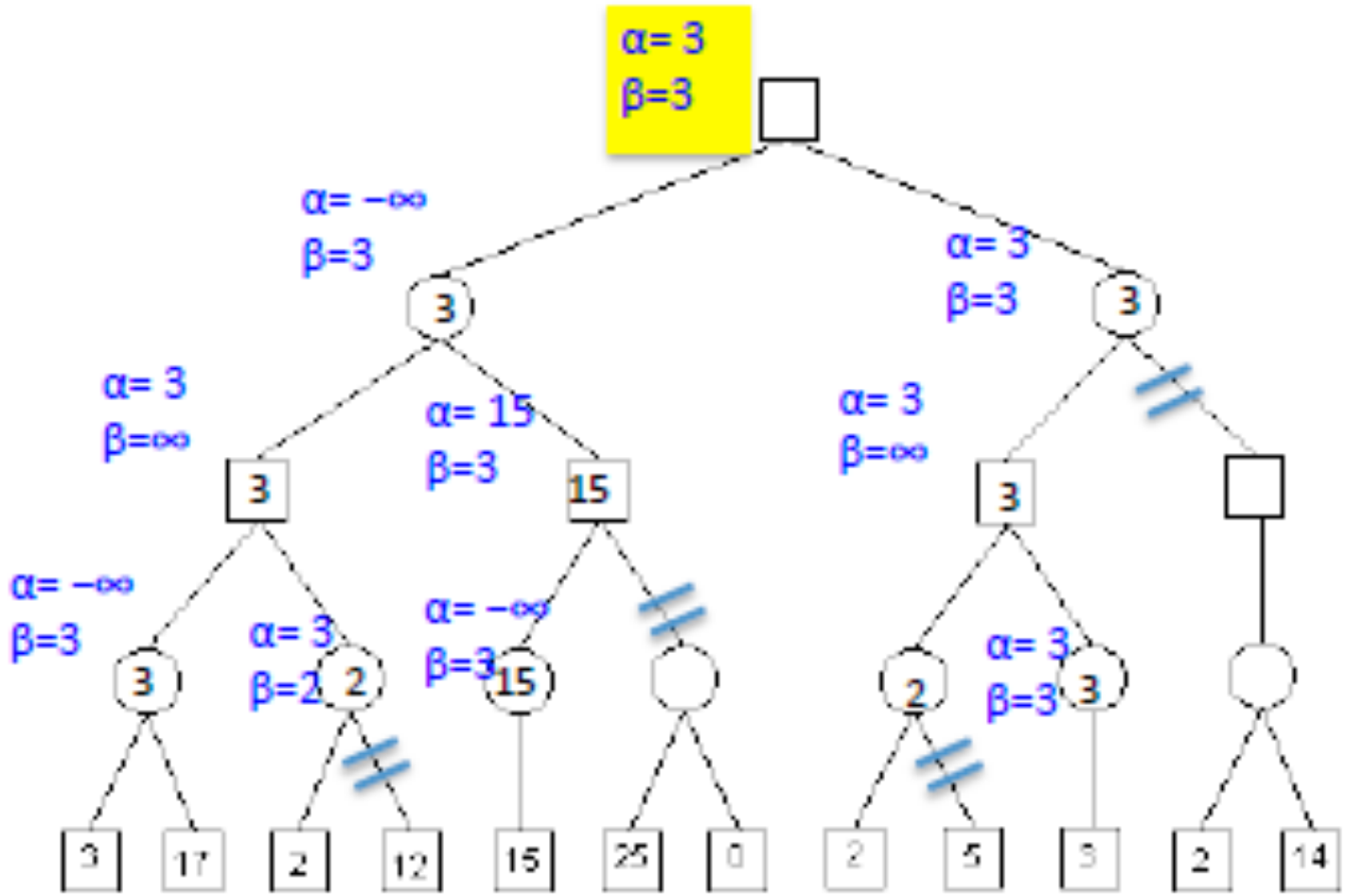

α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
α= −∞
β=3
3
α= 3
β=2
2
α= −∞
β=3
15
α= 3
β=3
2
3
3
17
2
12
15
25
0
2
5
3
2
14


α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 3
β=∞
3
α= 15
β=3
15
α= 3
β=∞
3
α= −∞
β=3
3
α= 3
β=1
2
α= −∞
β=3
15
2
α= 3
β=3
3
3
17
2
12
15
25
0
2
5
3
2
14


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


α= 3
β=∞
α= −∞
β=3
3
α= 3
β=∞
α= 15
β=3
15
α= 3
β=∞
3
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


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


α= 3
β=3
α= −∞
β=3
α= 3
β=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

Games of Chance

- In the game with **uncertainty** Players include a random element(roll dice, flip a coin, etc.) to determine what moves to make
  - i.e., Dice are rolled at the beginning of a player's turn to determine the legal moves. Such games are called game of chance.
- Chance games are good for exploring decision making in adversarial problems involving skill and luck.

Contd…

- Game Trees with Chance Nodes:
  - The game tree in the chance game include chance nodes in addition to MAX and MIN nodes. **Chance nodes** (shown as circles) represent the dice rolls.
  - The branches leading from each chance node denote the possible dice rolls; each branch is labeled with the roll and its probability.

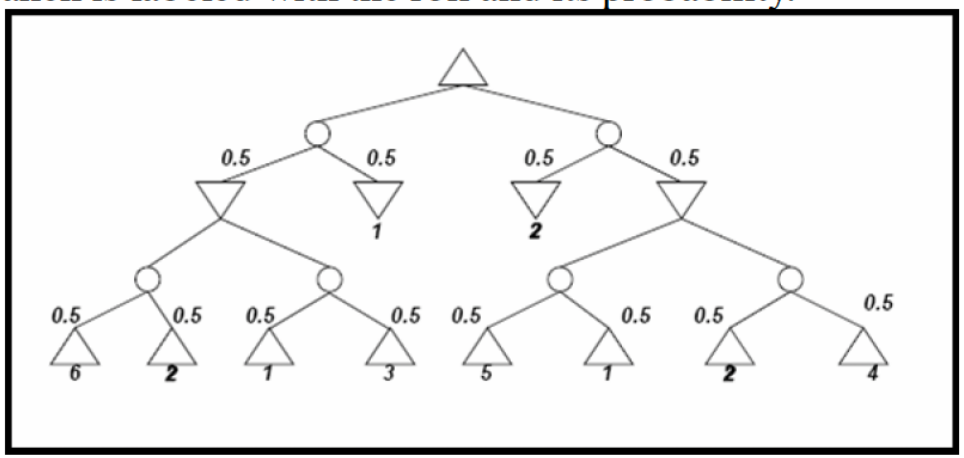
Max

Chance

min

Chance

Terminal node

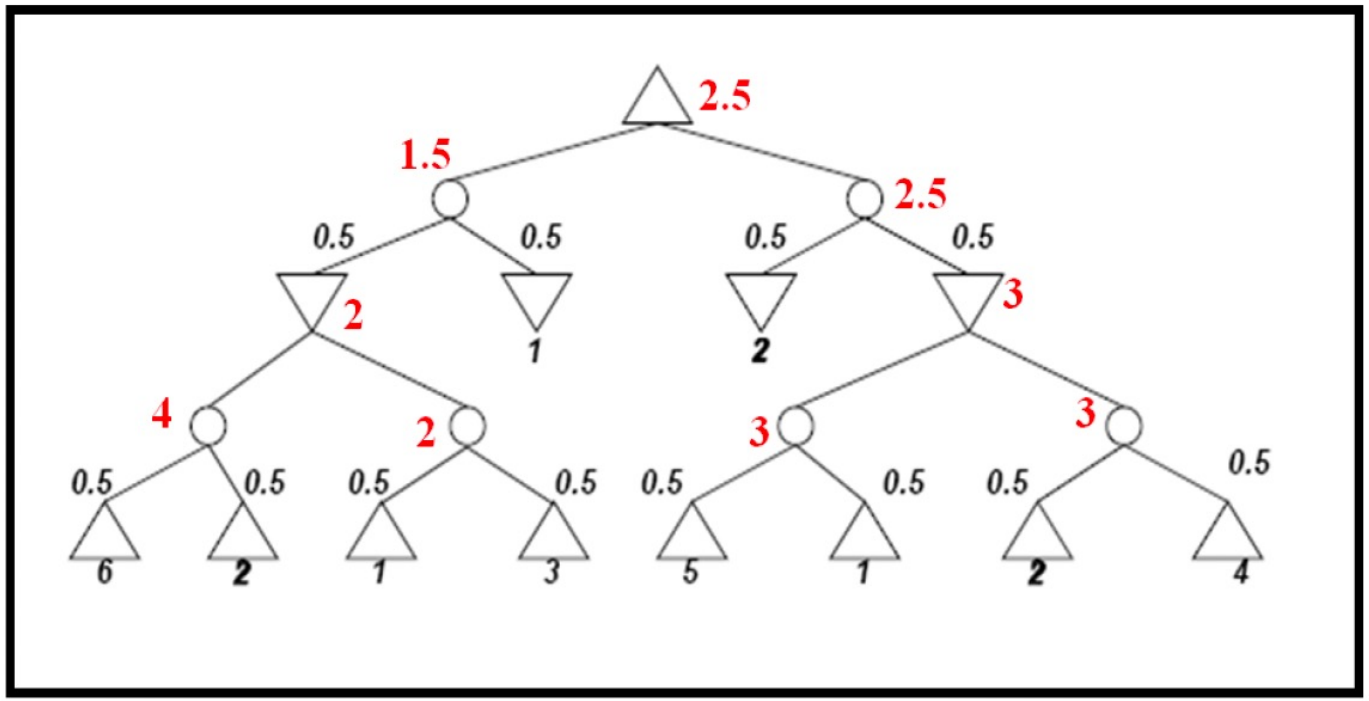


Contd…

- Algorithm for chance Games:
- Generalization of minimax for games with chance nodes. Also known as Expectiminimax give perfect play
  - If the state is terminal node then Return the utility value.
  - if state is a MAX node then return highest Expectiminimax – Value of Successors
  - if state is a MIN node then return lowest Expectiminimax – Value of Successors
  - if state is a CHANCE node then
    - For chance nodes 'C' over a max node, compute: epectimax(C) = $Sum_i(P(d_i) * maxvalue(i))$
    - For chance nodes 'C' over a min node compute: epectimin(C) = $Sum_i(P(d_i) * minvalue(i))$

Contd…

- Example:



What is Game Theory?

- Game theory is a study of how to mathematically determine the best strategy for given conditions in order to optimize the outcome
- Finding acceptable, if not optimal, strategies in conflict situations.
- Abstraction of real complex situation
- Game theory is highly mathematical
- Game theory assumes all human interactions can be understood and navigated by presumptions.

Contd…

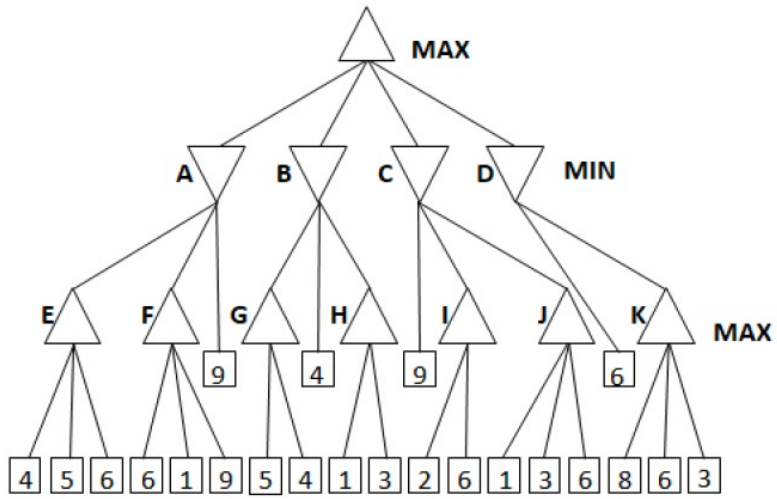
- Why is game theory important?
  - All intelligent beings make decisions all the time.
  - AI needs to perform these tasks as a result.
  - Helps us to analyze situations more rationally and formulate an acceptable alternative with respect to circumstance.
  - Useful in modeling strategic decision-making
    - Games against opponents
    - Games against nature
  - Provides structured insight into the value of information

Assignment-3

- 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
  - 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?

- 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
```

Thank you!!!