Chapter 3: Search techniques (6 hrs)

- Remaining portion of class notes

Compiled by Ankit Bhattarai
Cosmos College of Management &
Technology, AI (2021 Edited)

Searching: Steps

1. Check whether the current state is the goal state or not
2. Expand the current state to generate the new sets of states
3. Choose one of the new states generated for search which entire depend on the selected search strategy
4. Repeat the above steps until the goal state is reached or there are no more states to be expanded

Searching: Criteria to Measure Performance

- **Completeness**: Ability to find the solution if the solution exists
- **Optimality**: Ability to find out the highest quality solution among the several solutions
  - Should maintain the information about the number of steps or the path cost from the current state to the goal state
- **Time Complexity**: Time taken to find out the solution
- **Space Complexity**: Amount of Memory required to perform the searching

Searching: Evolution Function

- A number to indicate how far we are from the goal
- Every move should reduce this number or if not never increase
- When this number becomes zero, the problem is solved (there may be some exceptions)

Outline

- Searching
- Uninformed Search Techniques
  - **Breadth First Search**
  - Uniform Cost Search
  - **Depth First Search**
  - Depth Limited Search
  - **Iterative Deepening Depth First Search**
  - Bidirectional Search
  - Search Strategy Comparison
- **Informed Search Techniques**
  - **Generate & test algorithm**
  - **Best First Searching**
    - **Greedy Search**
    - **A* Search**
  - **Iterative Improvement Algorithm**
    - **Hill Climbing Search**
    - **Simulated Annealing Search**
    - **Local Beam Search**
    - **Genetic Algorithm**
  - **Adversarial Search Techniques**
    - **Mini-max Procedure**
    - **Alpha Beta Procedure**

Remaining portion of Uninformed Search

Iterative deepening DFS:

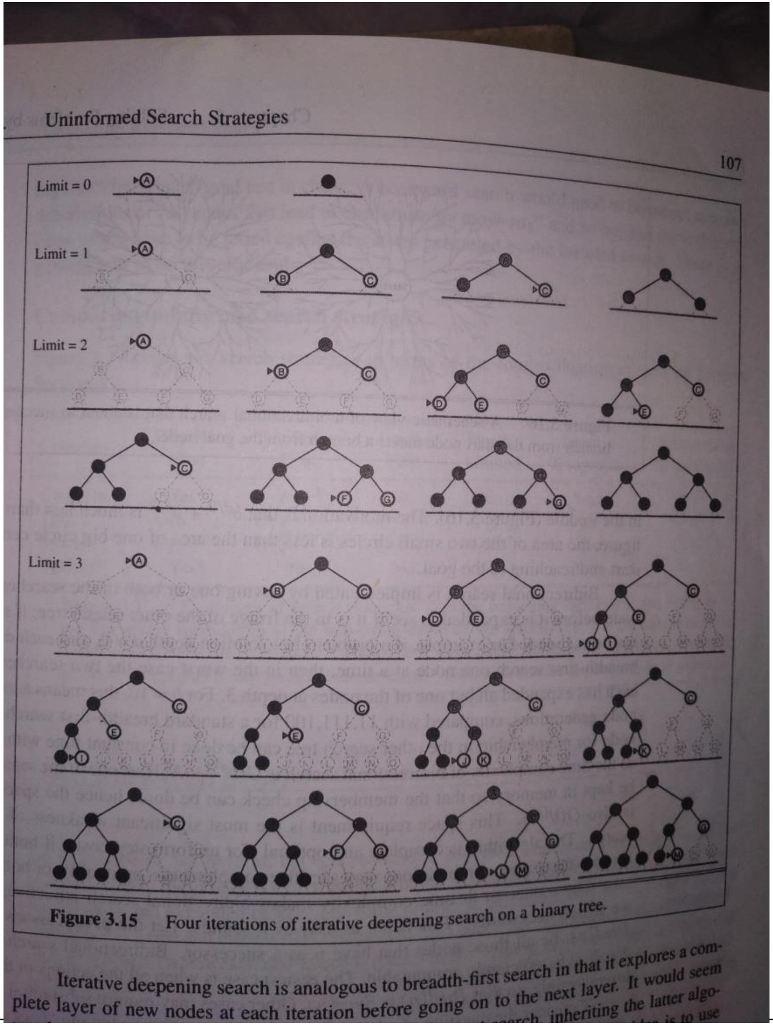
- Explanation in copy.
- Book page 107 fig 3.15
- *Book name:*

  Artificial Intelligence

  (A Modern Approach)

  Second Edition

  - *Stuart Russell*
  - *Peter Norvig*



Uninformed Search Strategies

107

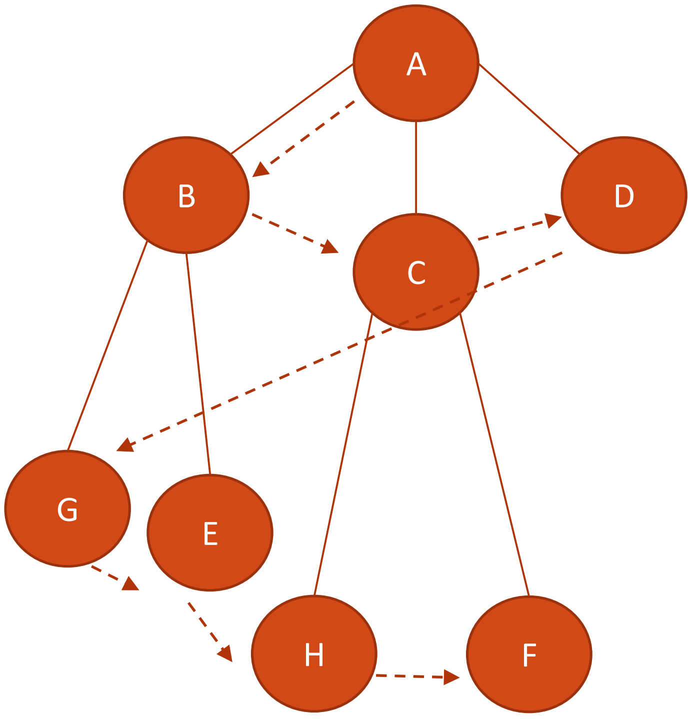
Limit = 0
A
Limit = 1
A
B
C
Limit = 2
A
B
C
D
E
F
G
Limit = 3
A
B
C
D
E
F
G
H
I
J
K
L
M
N
O

Figure 3.15 Four iterations of iterative deepening search on a binary tree.

Iterative deepening search is analogous to breadth-first search in that it explores a complete layer of new nodes at each iteration before going on to the next layer. It would seem

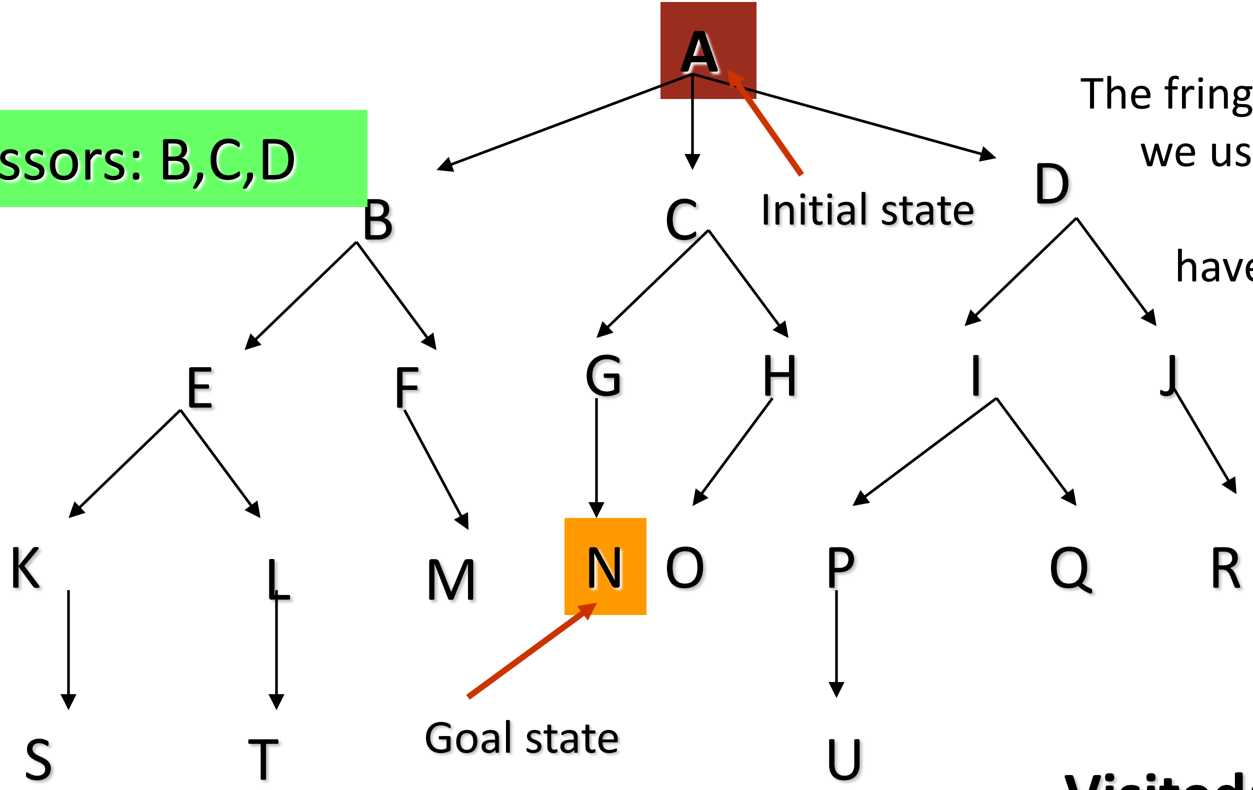
Breadth First Search

- Root node is expanded first
- Then all the successors of the root node are expanded
- Then their successors are expanded and so on.
- Nodes, which are visited first will be expanded first (FIFO)
- All the nodes of depth ‘d’ are expanded before expanding any node of depth ‘d+1’



Breadth-first Search

Successors: B,C,D

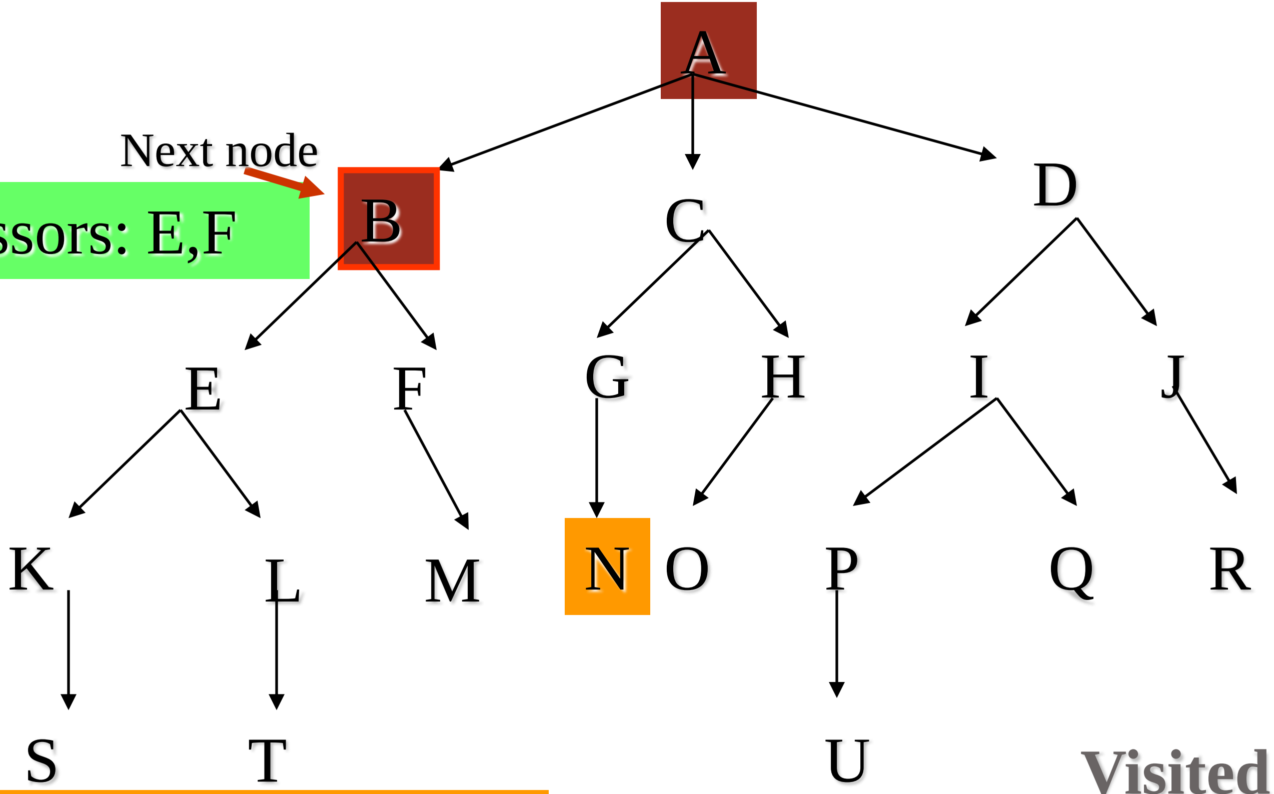


The fringe is the data structure we use to store all of the nodes that have been generated

Fringe: A (FIFO)

Visited:

Breadth-first Search



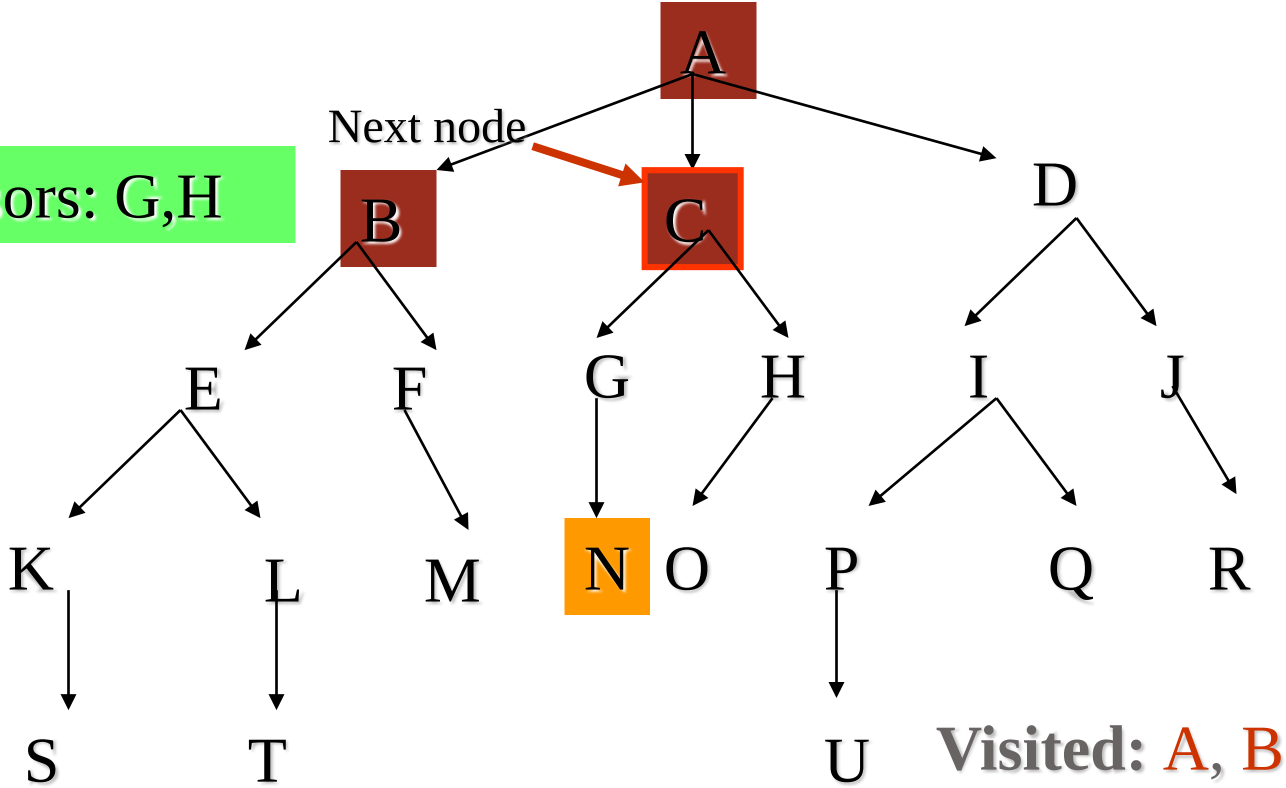
Next node

Successors: E,F

Visited: A

Fringe: B,C,D (FIFO)

Breadth-first Search

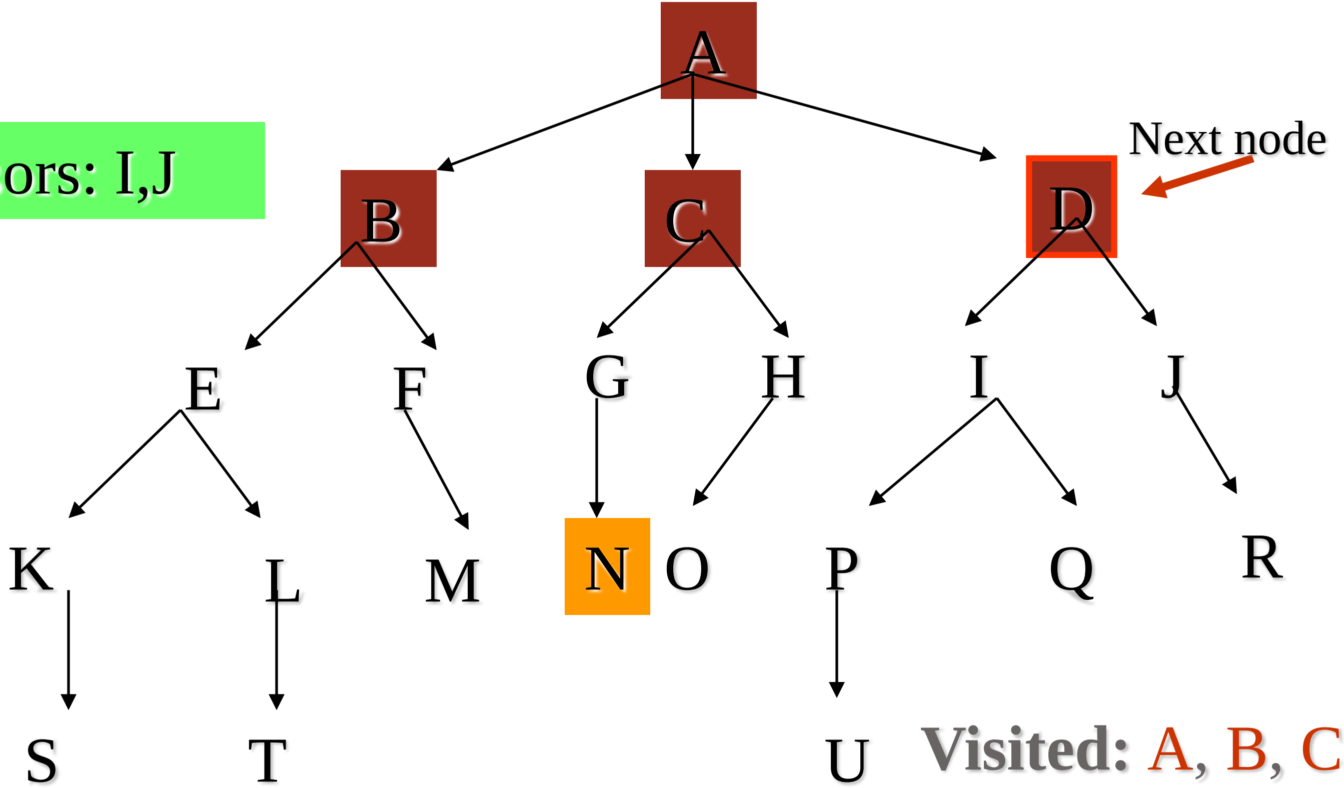


Successors: G,H

Visited: A, B

Fringe: C,D,E,F (FIFO)

Breadth-first Search



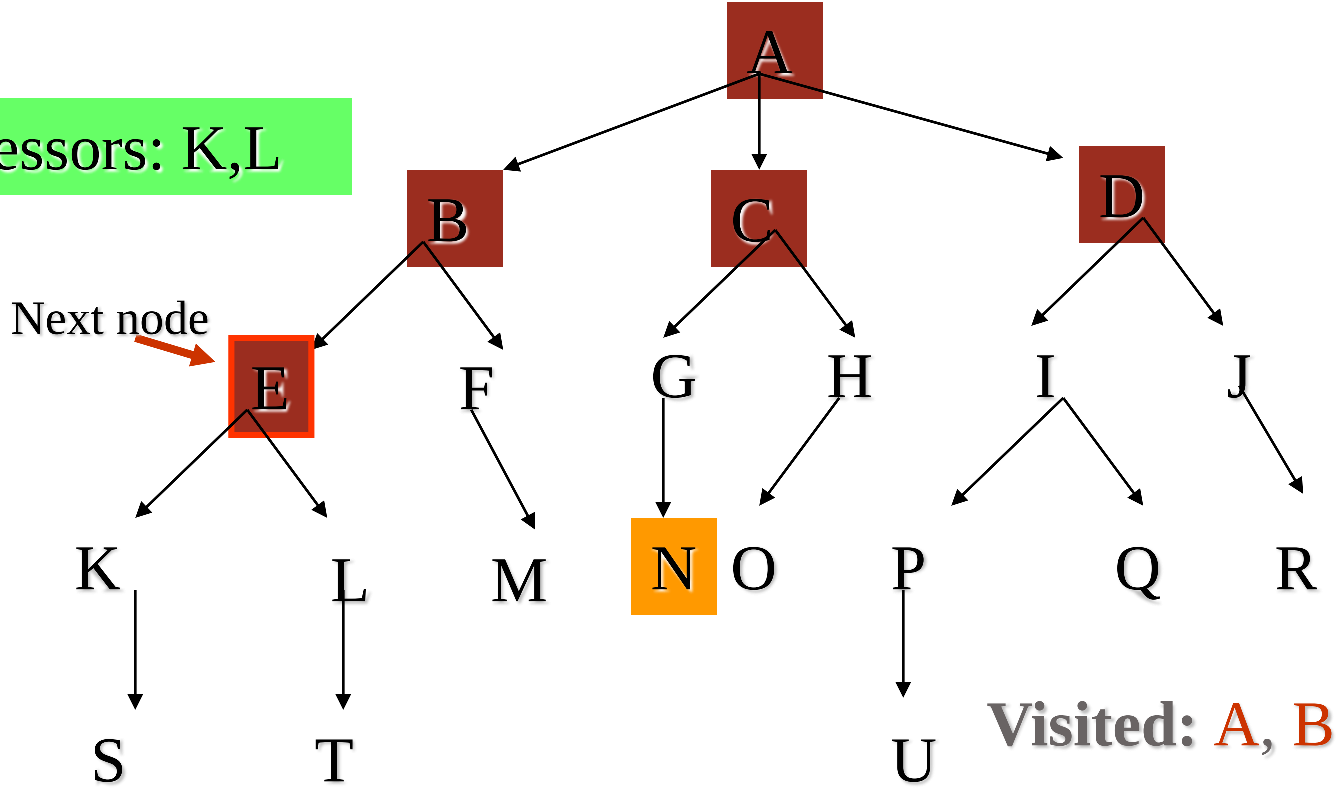
Successors: I,J

Visited: A, B, C

Fringe: D,E,F,G,H (FIFO)

Breadth-first Search

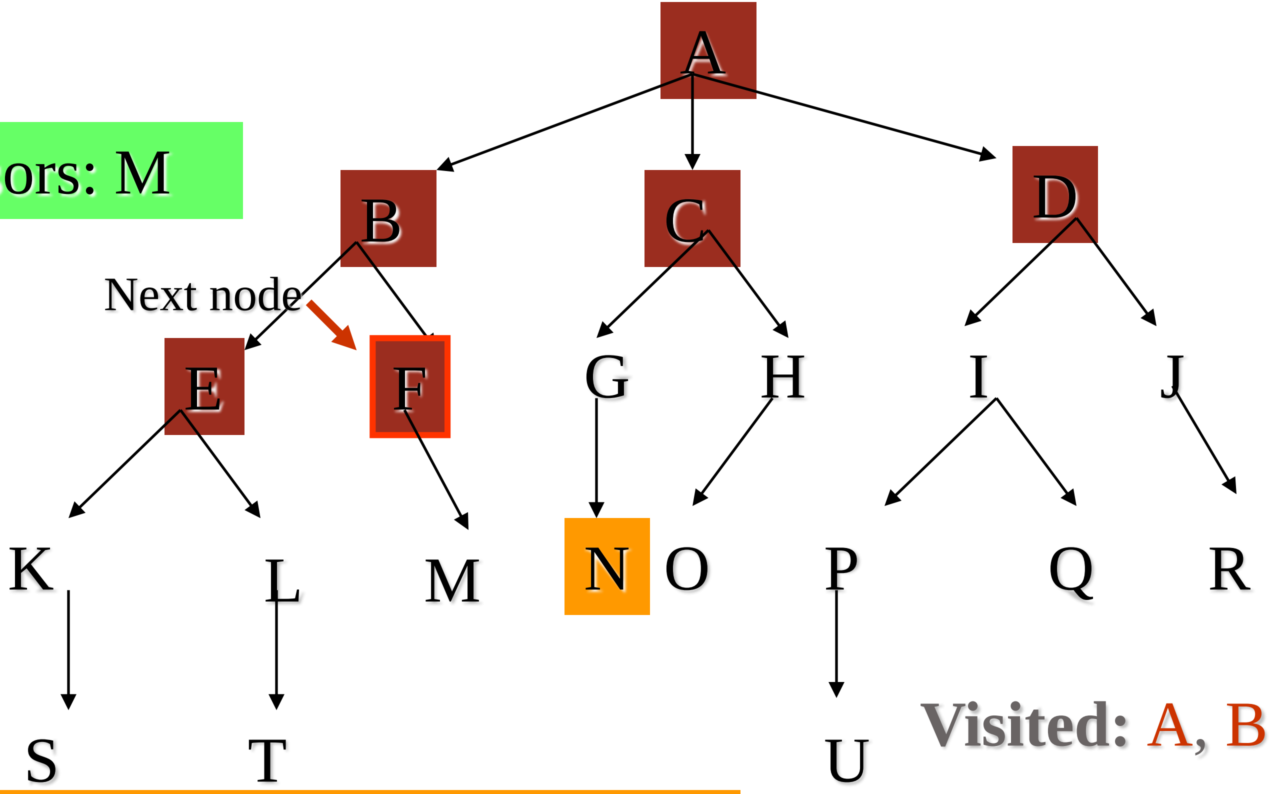
Successors: K,L



Visited: A, B, C, D

Fringe: E,F,G,H,I,J (FIFO)

Breadth-first Search



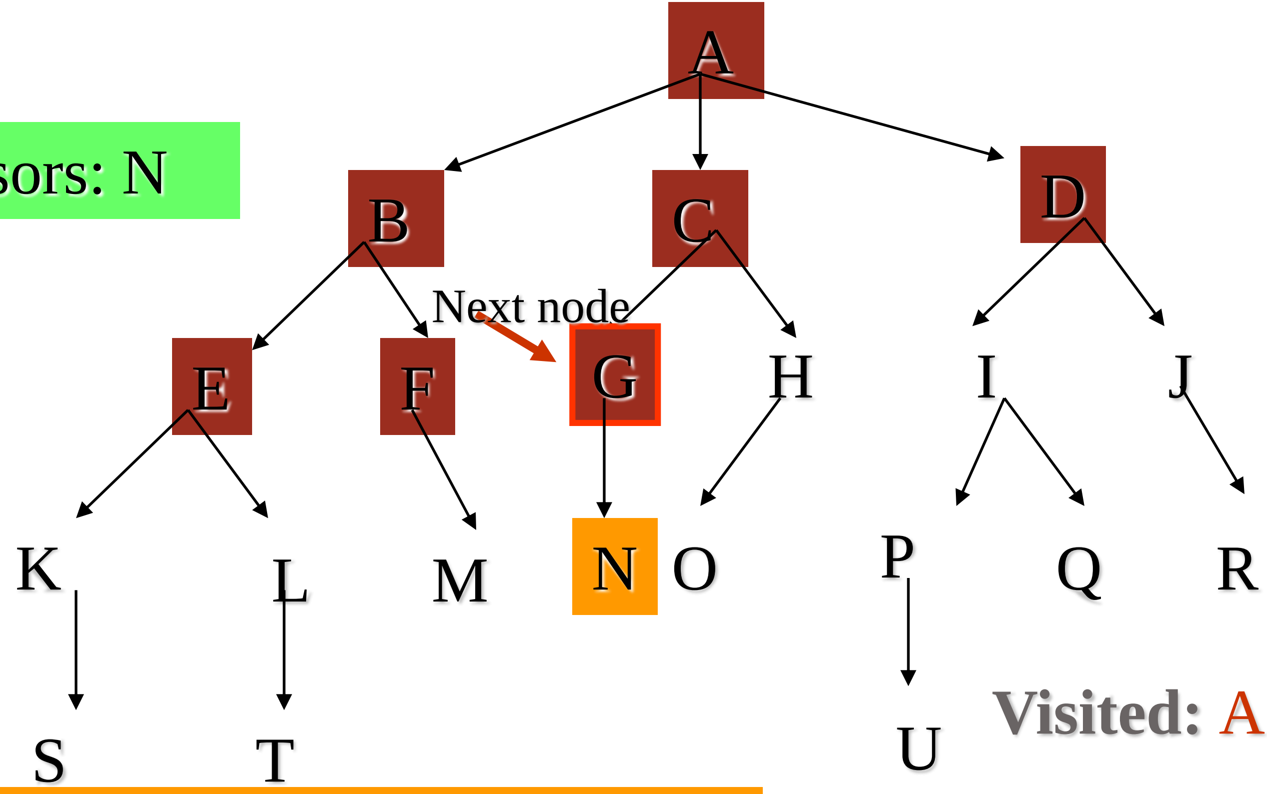
Successors: M

Visited: A, B, C, D, E

Fringe: F,G,H,I,J,K,L (FIFO)

Breadth-first Search

Successors: N

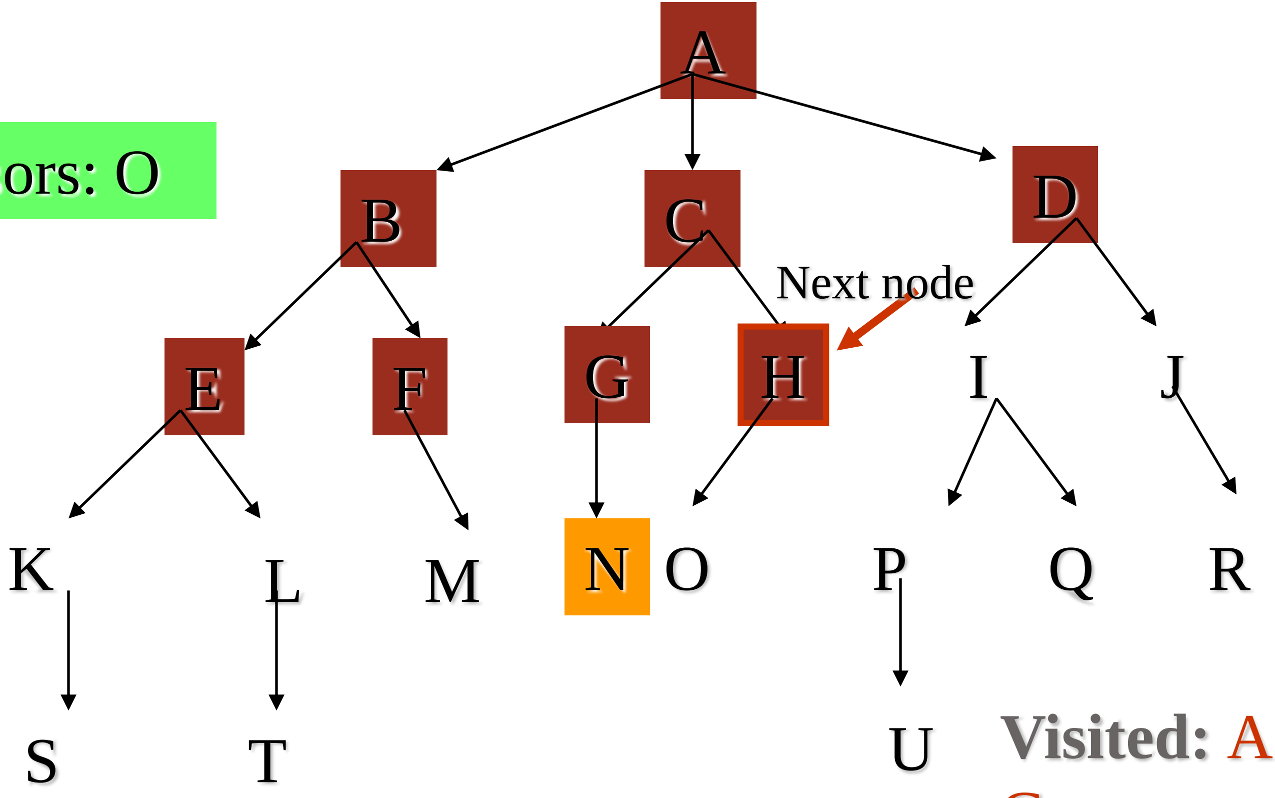


Visited: A, B, C, D, E, F

Fringe: G,H,I,J,K,L,M (FIFO)

Breadth-first Search

Successors: O

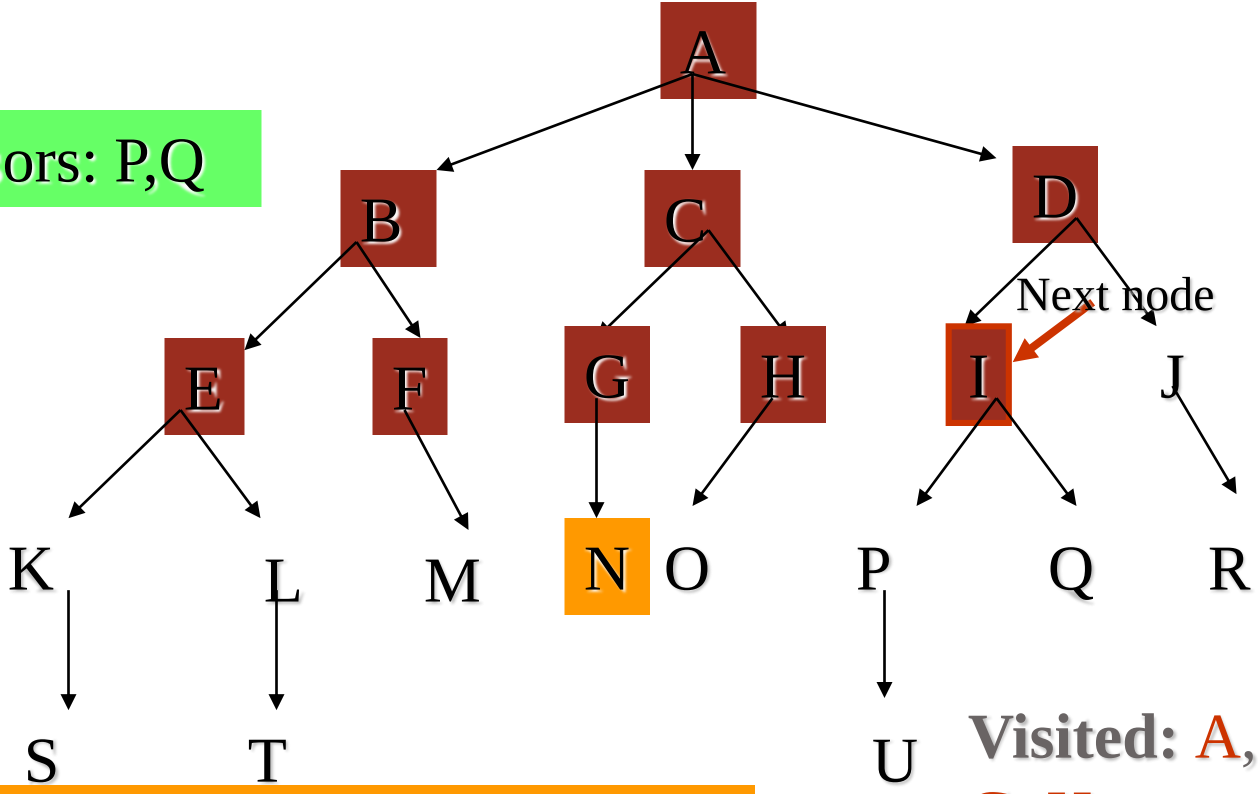


Visited: A, B, C, D, E, F, G

Fringe: H,I,J,K,L,M,N (FIFO)

Breadth-first Search

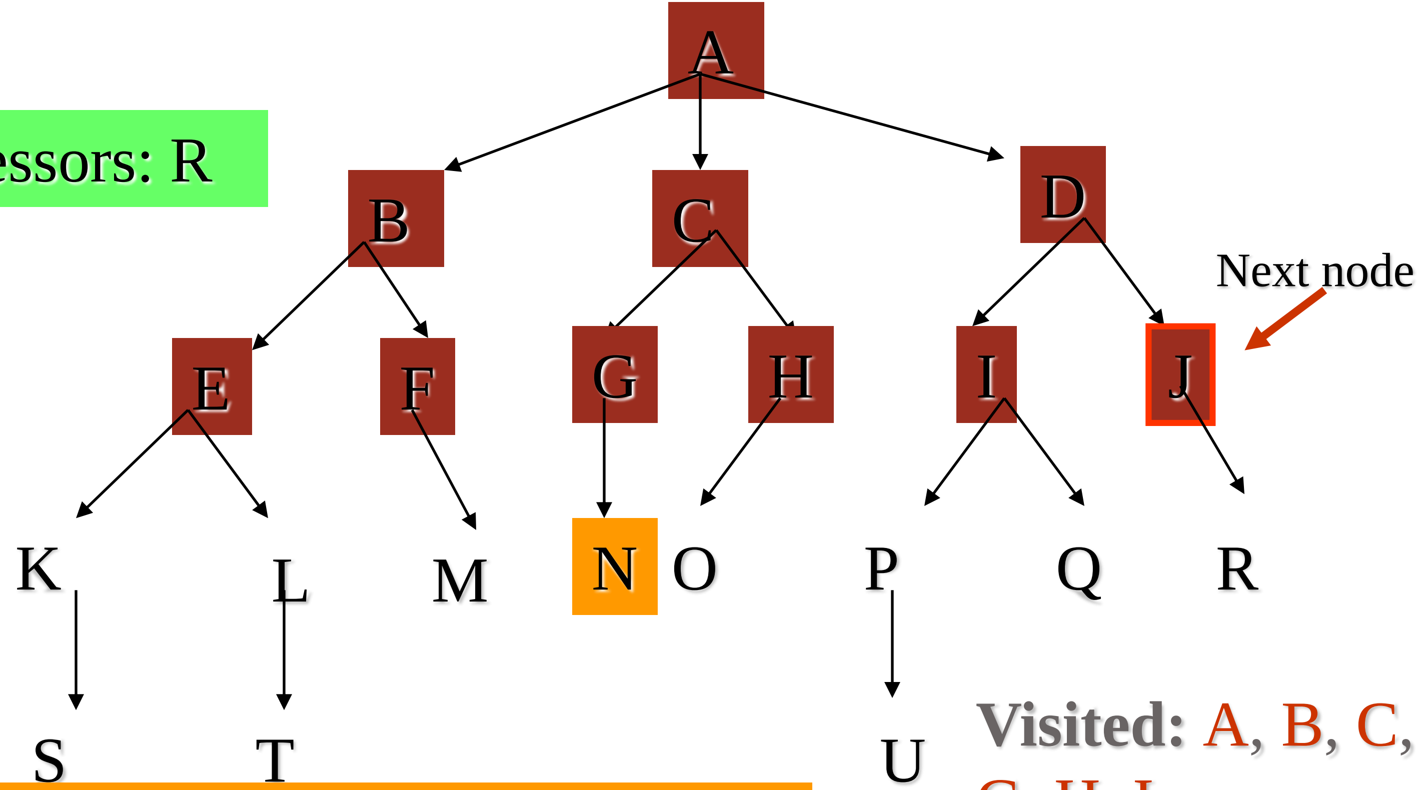
Successors: P,Q



Visited: A, B, C, D, E, F, G, H

Fringe: I,J,K,L,M,N,O (FIFO)

Breadth-first Search



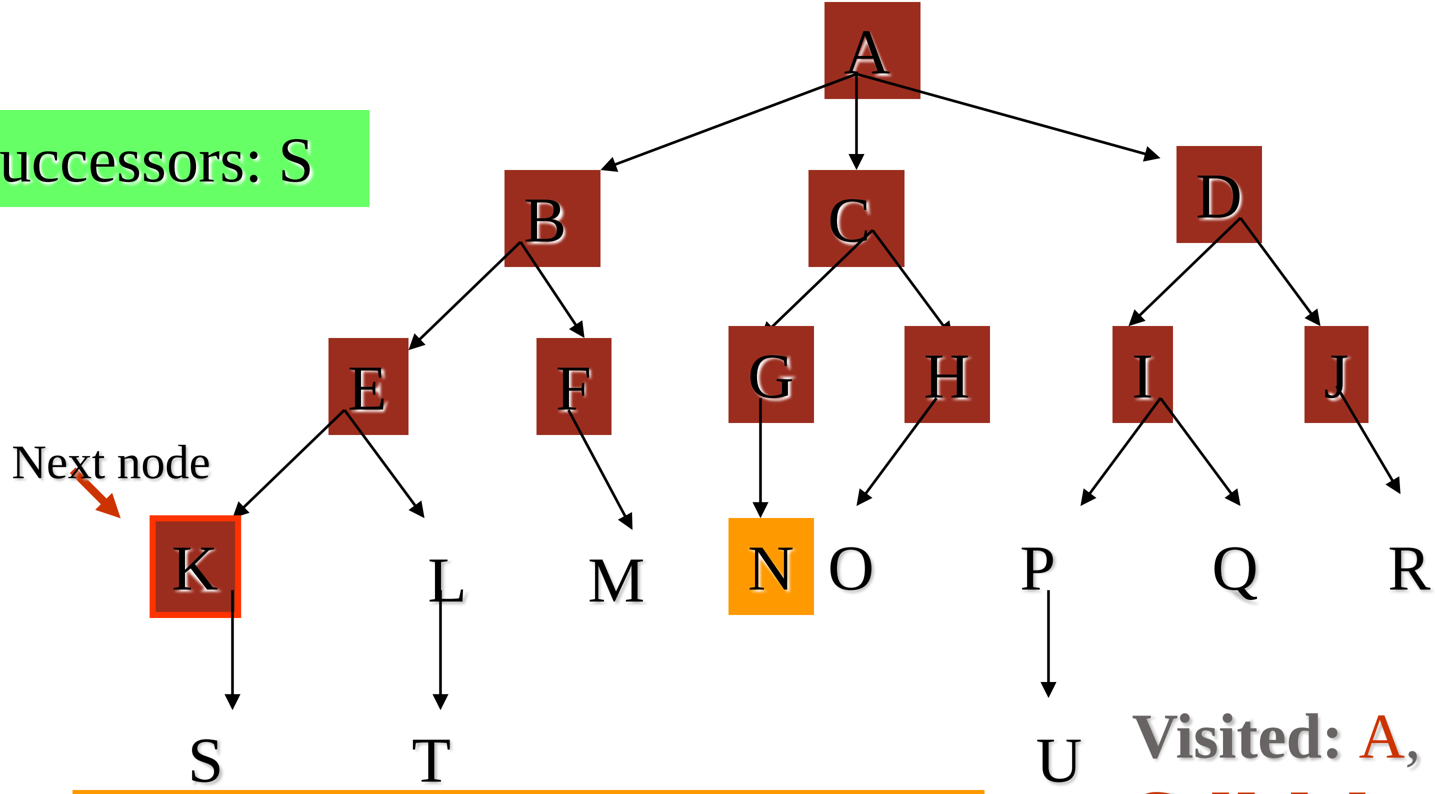
Successors: R

Visited: A, B, C, D, E, F, G, H, I

Fringe: J,K,L,M,N,O,P,Q (FIFO)

Breadth-first Search

Successors: S

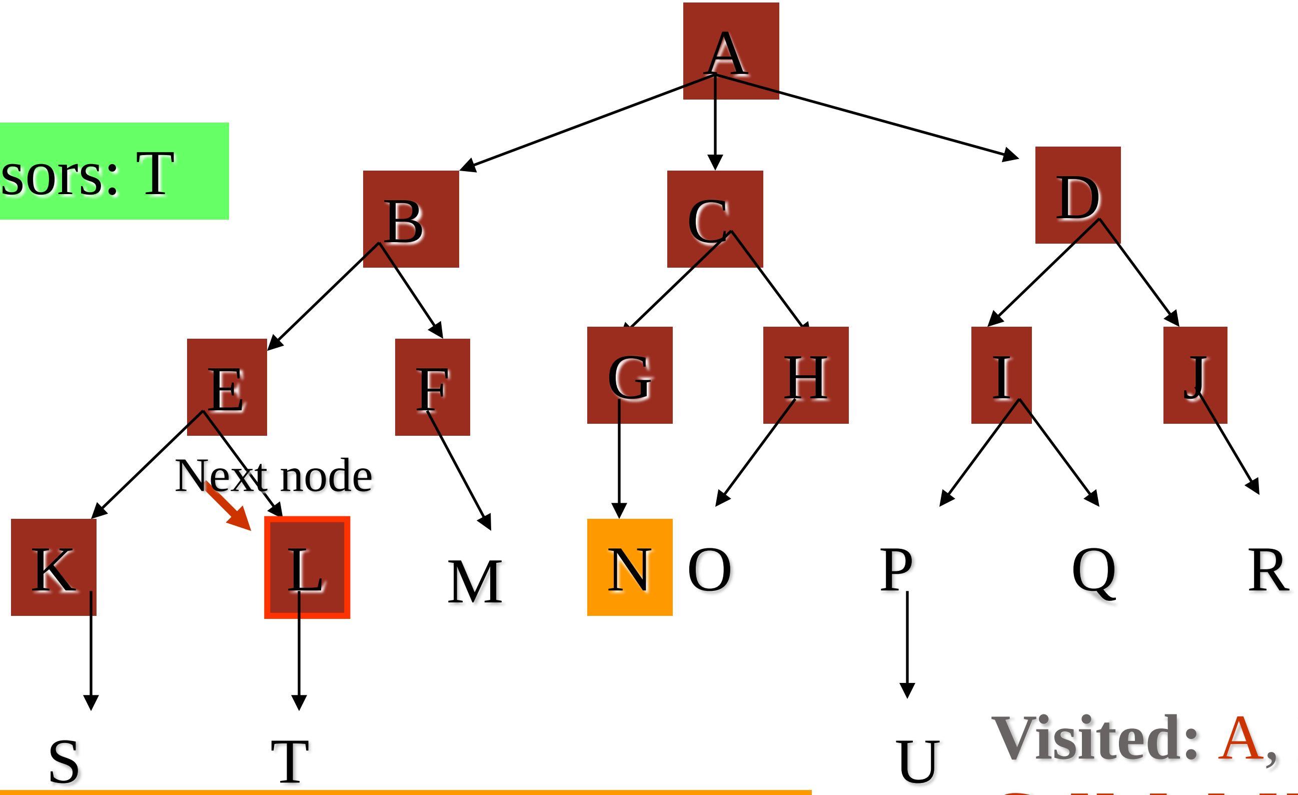


Visited: A, B, C, D, E, F, G, H, I, J

Fringe: K,L,M,N,O,P,Q,R (FIFO)

Breadth-first Search

Successors: T

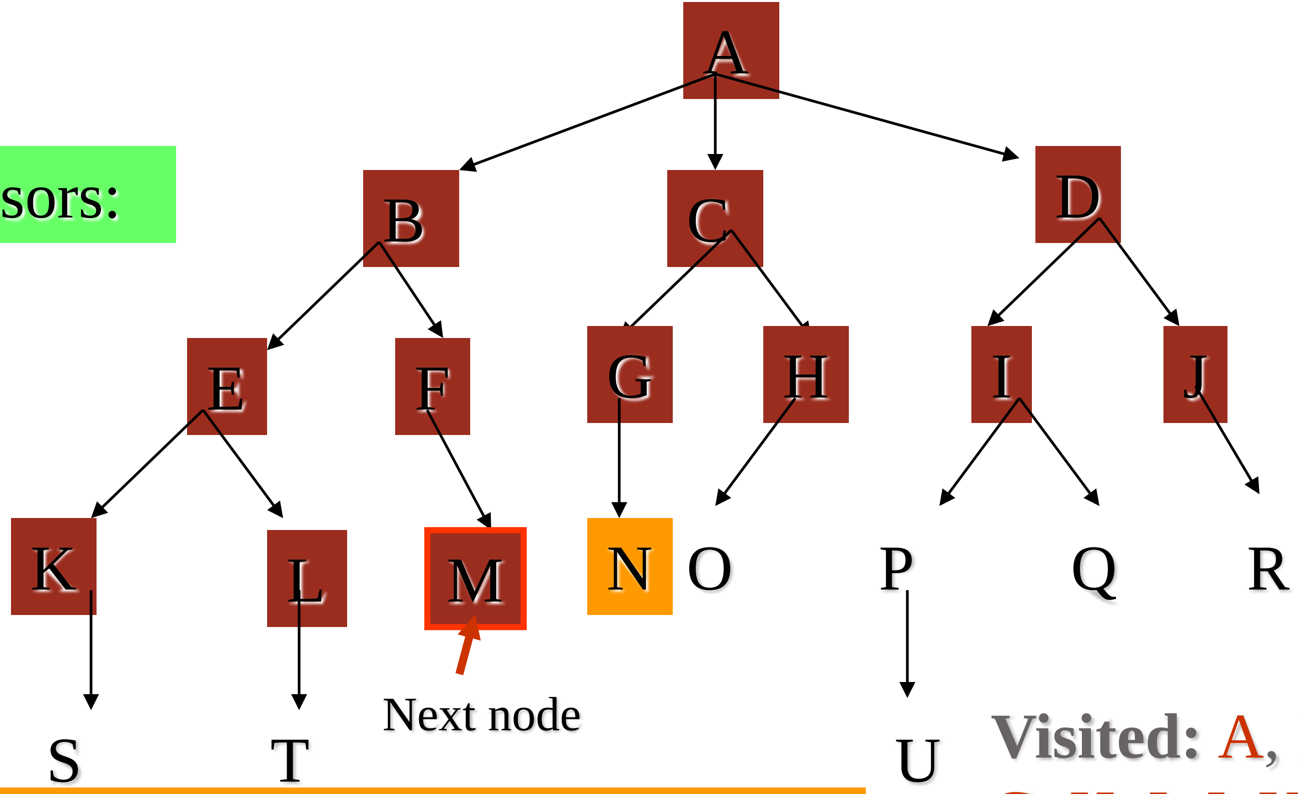


Visited: A, B, C, D, E, F, G, H, I, J, K

Fringe: L,M,N,O,P,Q,R,S (FIFO)

Breadth-first Search

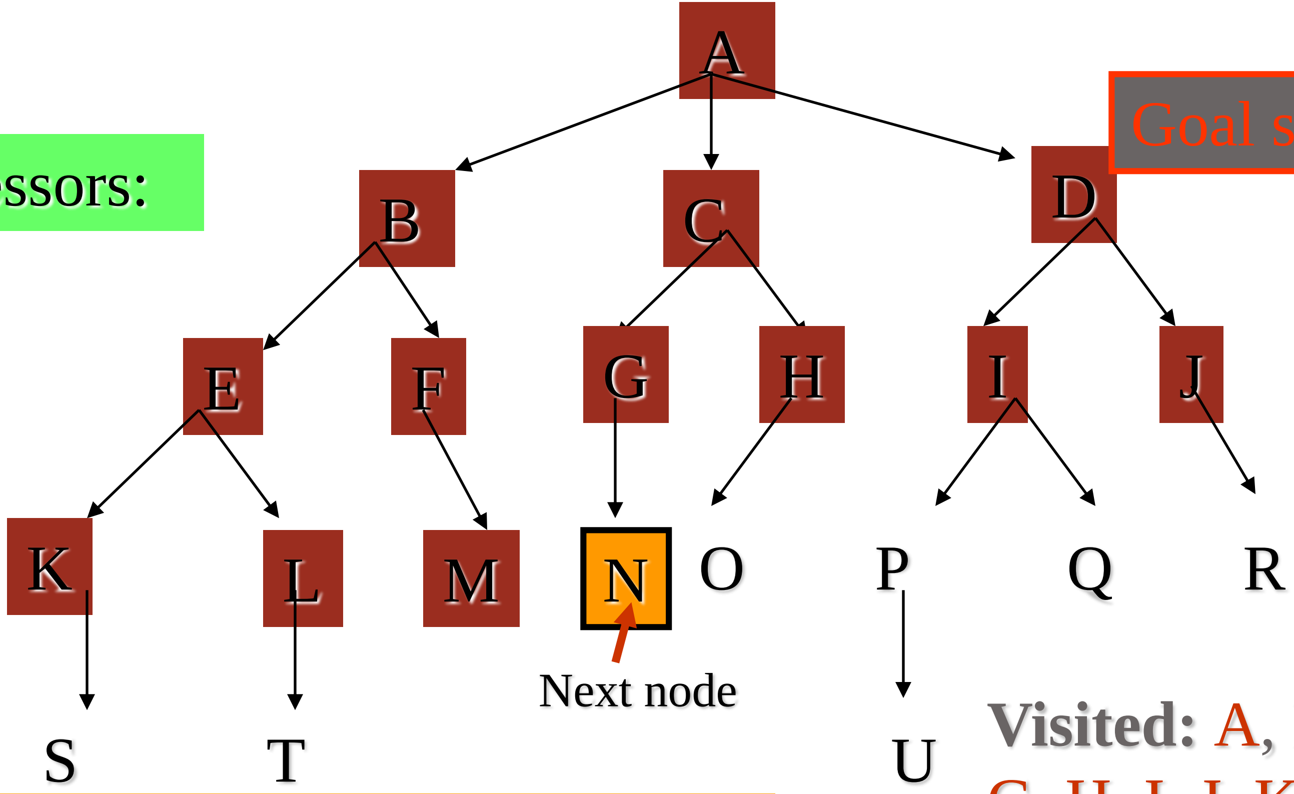
Successors:



Fringe: M,N,O,P,Q,R,S,T (FIFO)

Visited: A, B, C, D, E, F, G, H, I, J, K, L

Breadth-first Search



Goal state achieved

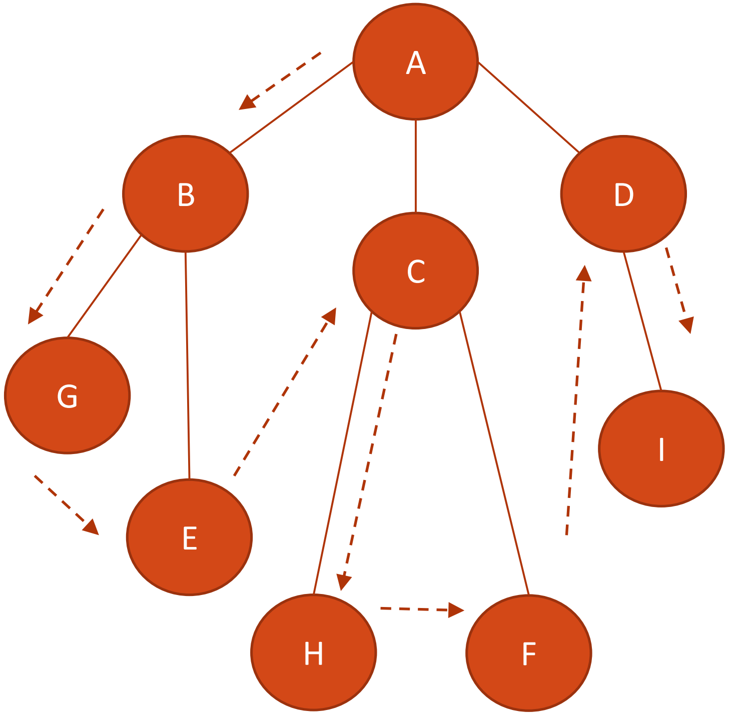
Successors:

Visited: A, B, C, D, E, F, G, H, I, J, K, L, M

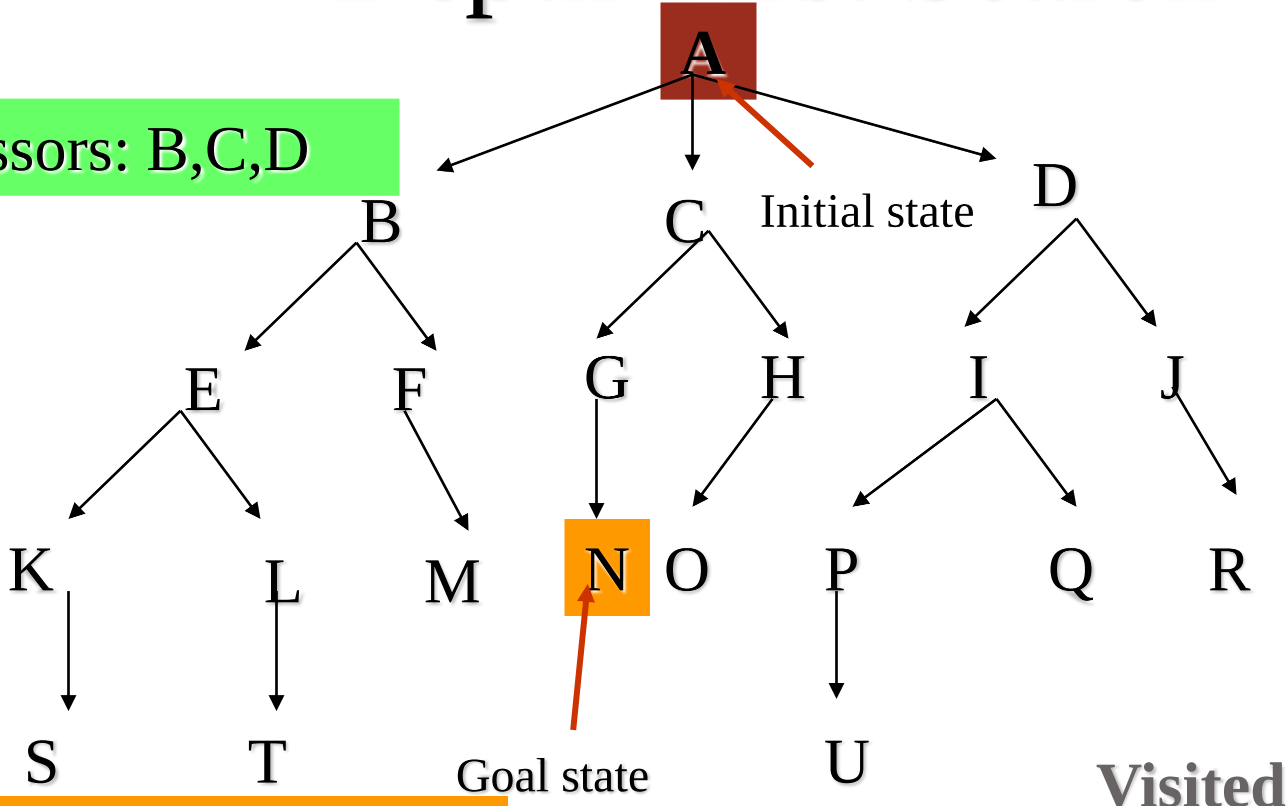
Fringe: N,O,P,Q,R,S,T (FIFO)

Depth First Search

- Expands the nodes at the deepest level of the search tree (LIFO)
- When a dead end is reached, the search backup to the next node that still has unexplored successors



Depth-First Search

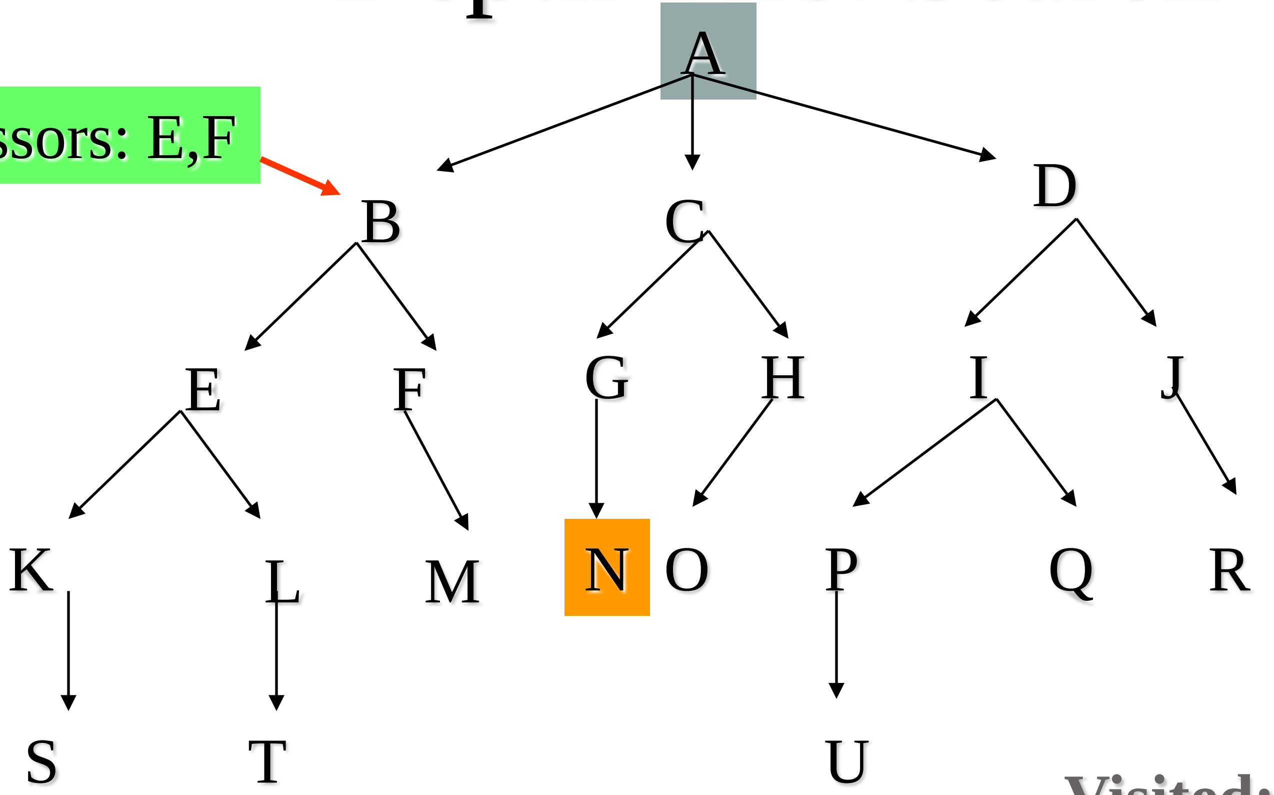


Successors: B,C,D

Fringe: A (LIFO)

Visited:

Depth-First Search

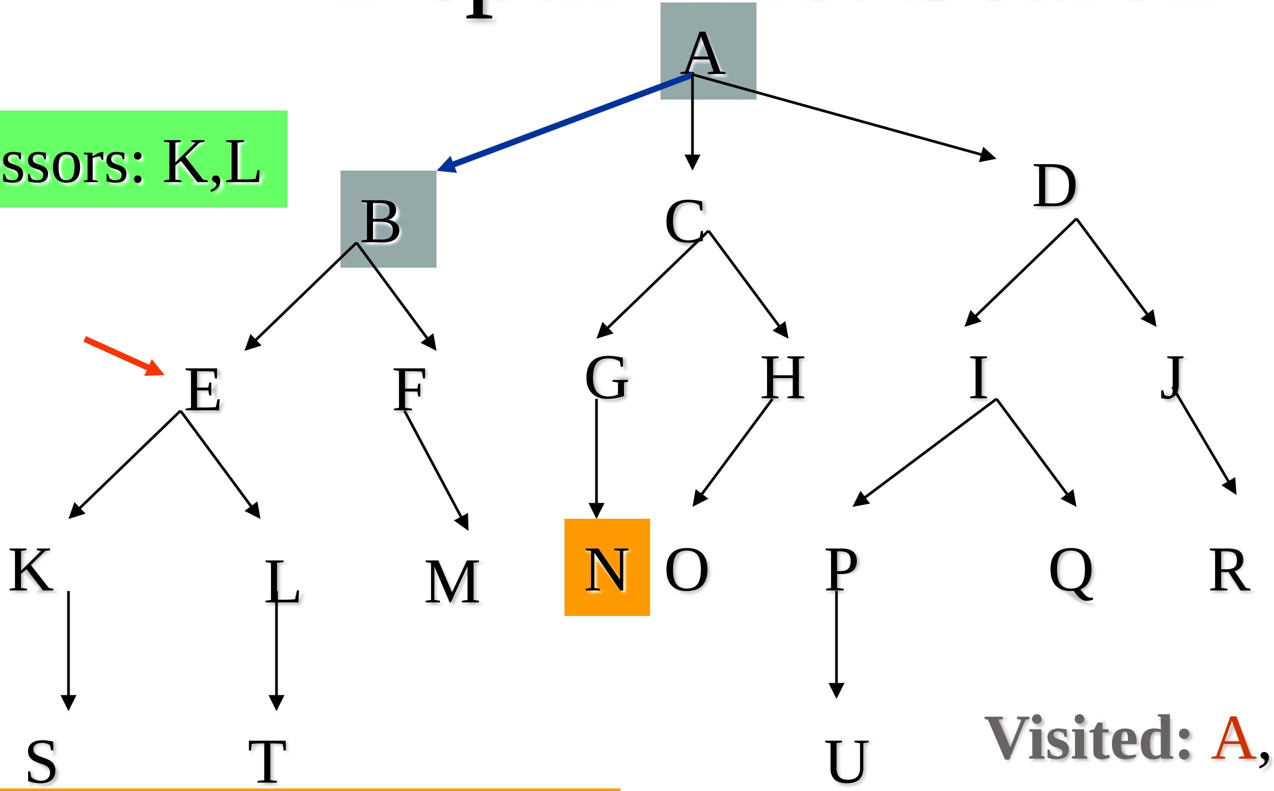


Successors: E,F

Visited: A

Fringe: B,C,D (LIFO)

Depth-First Search

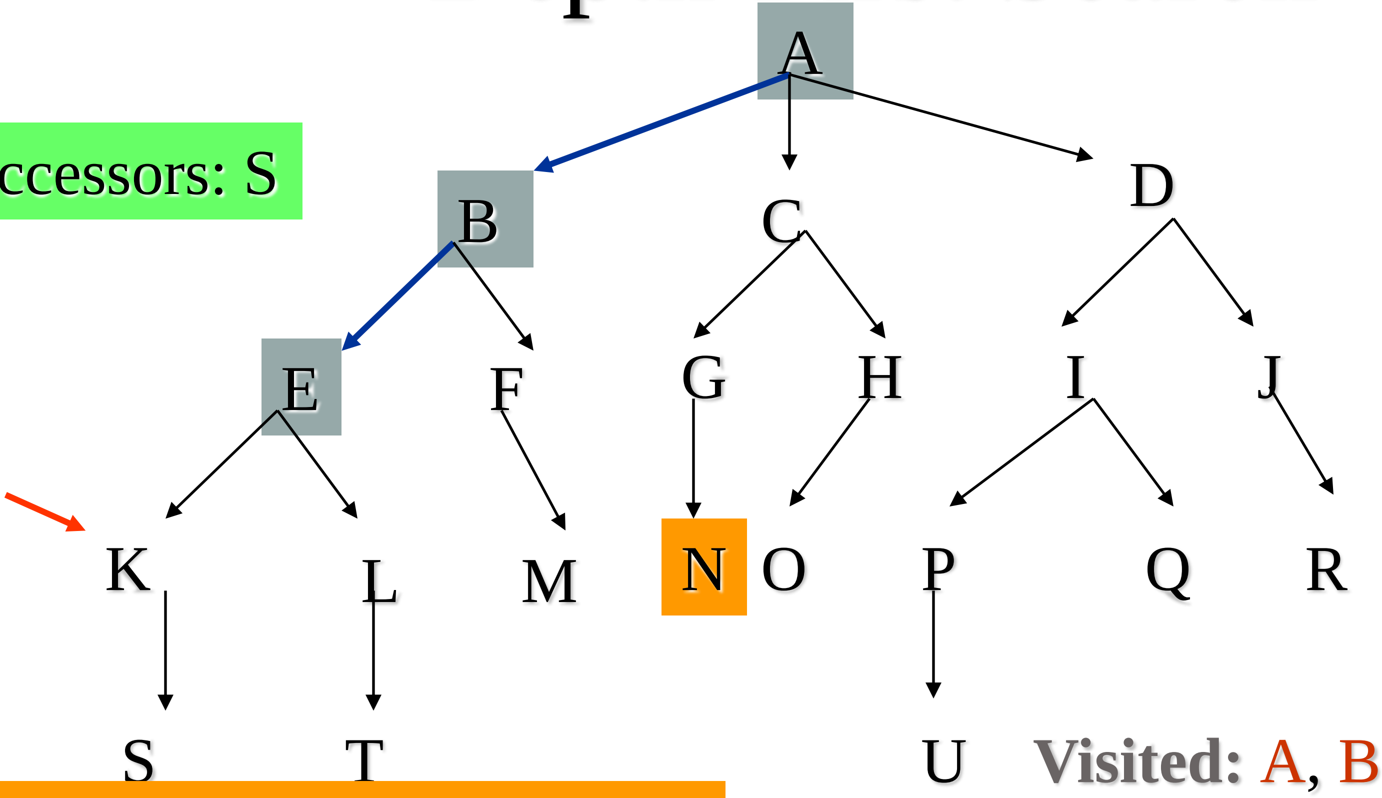


Successors: K,L

Visited: A, B

Fringe: E,F,C,D (LIFO)

Depth-First Search

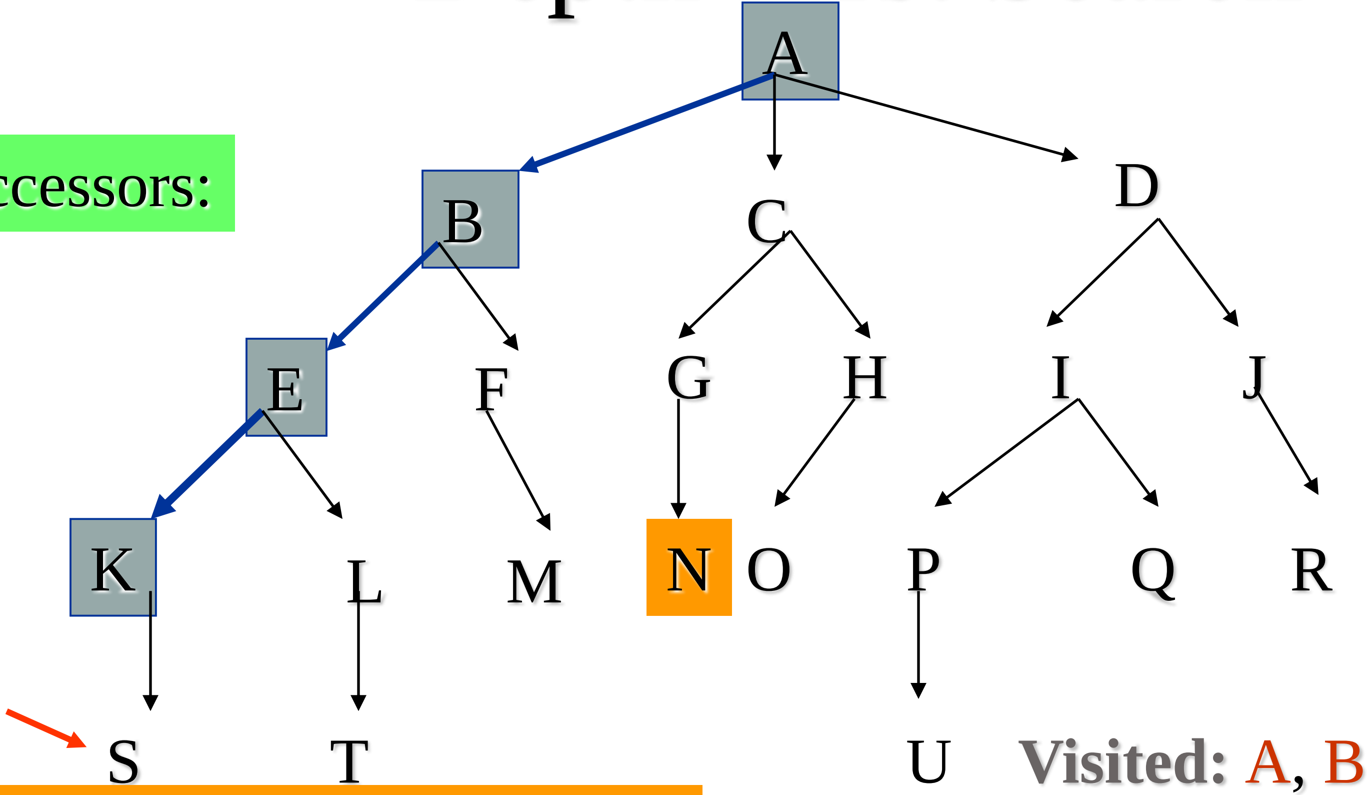


Successors: S

Visited: A, B, E

Fringe: K,L,F,C,D (LIFO)

Depth-First Search

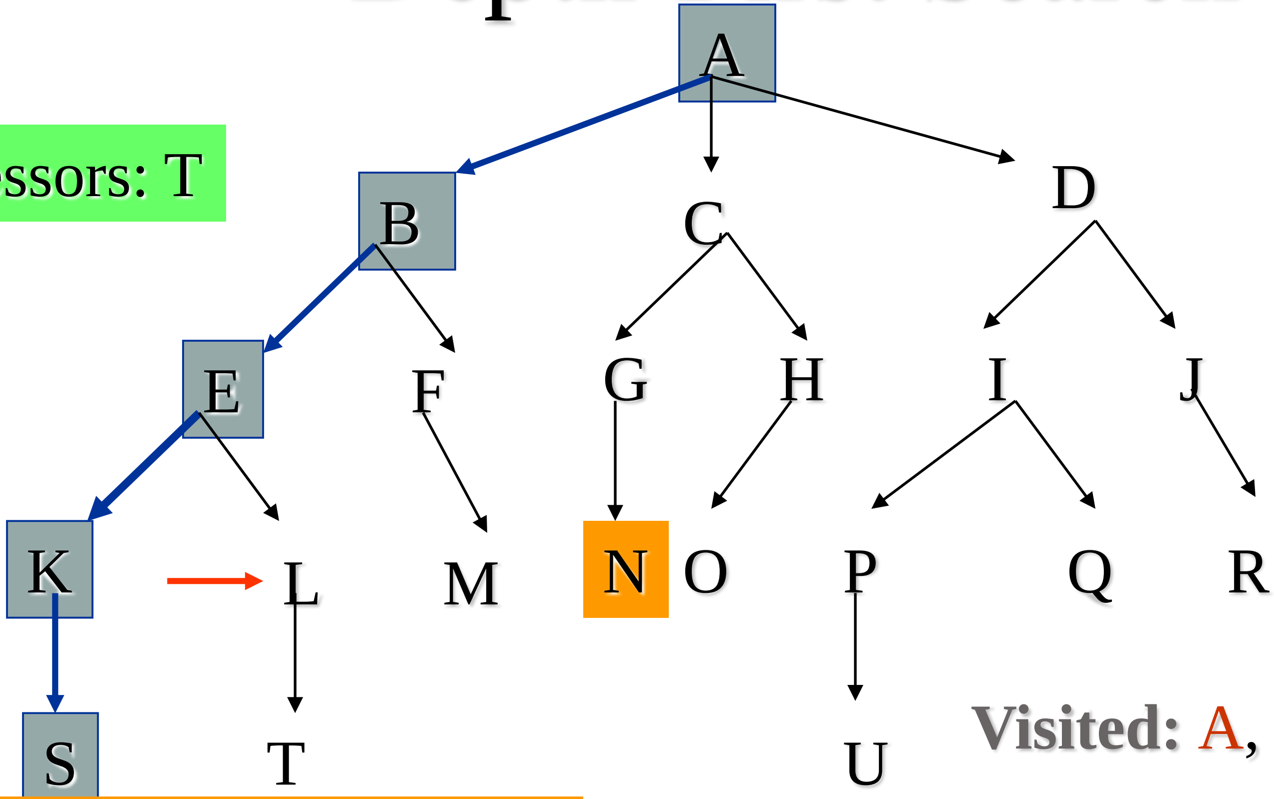


Successors:

Visited: A, B, E, K

Fringe: S,L,F,C,D (LIFO)

Depth-First Search



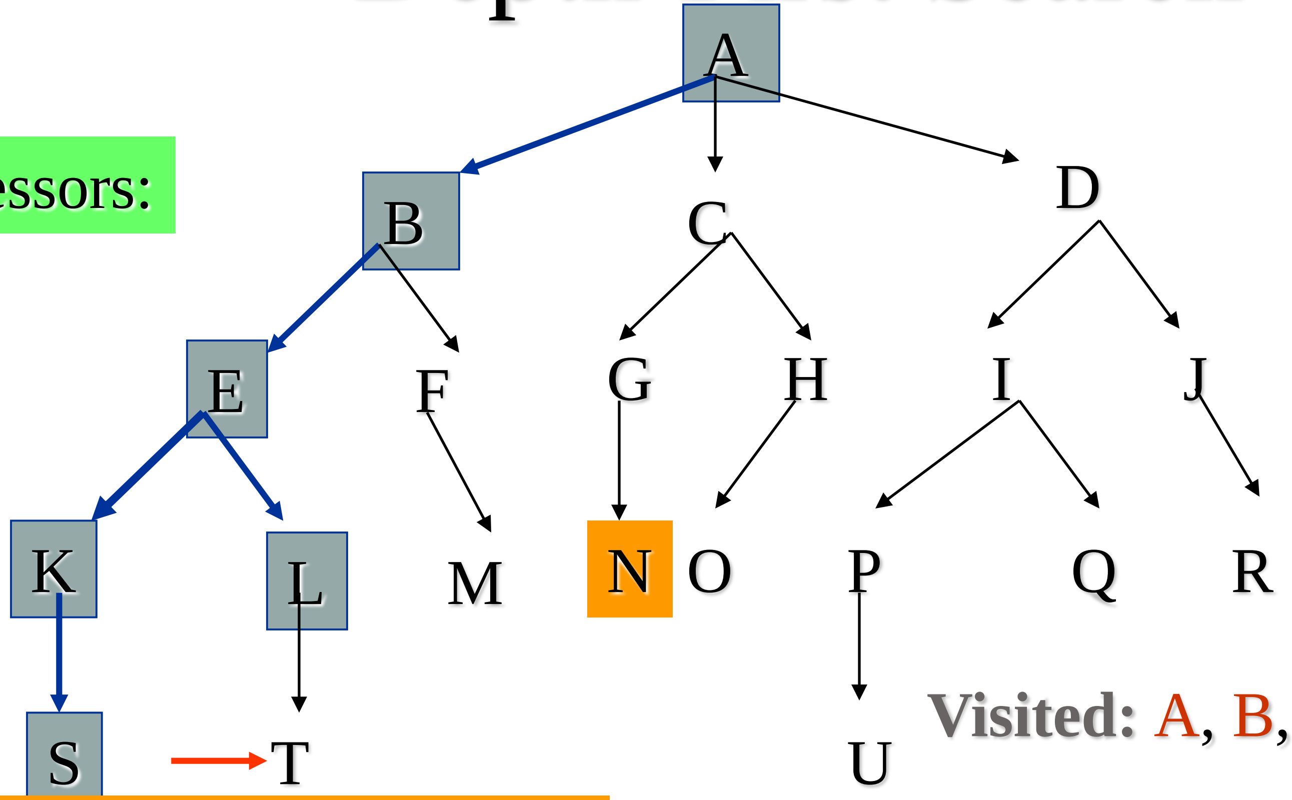
Successors: T

Visited: A, B, E, K, S

Backtracking

Fringe: L,F,C,D (LIFO)

Depth-First Search

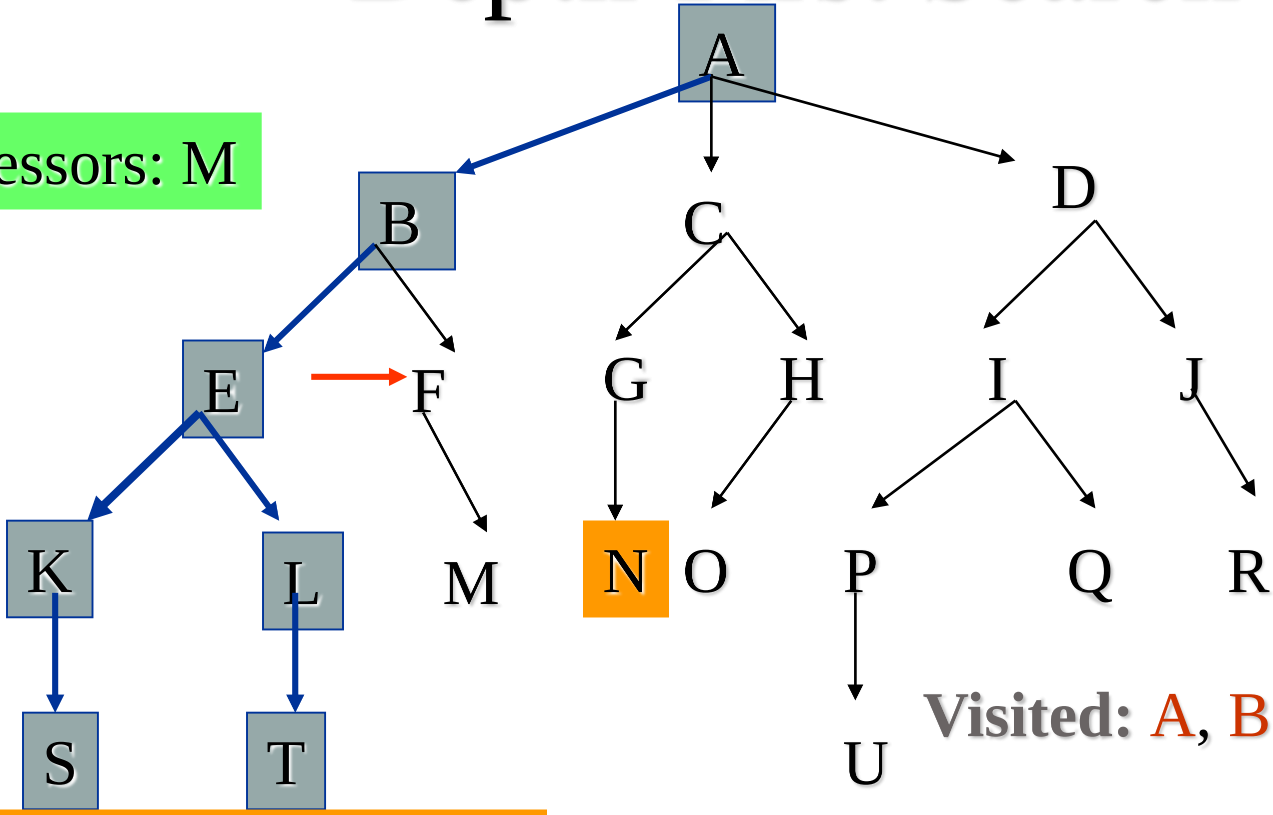


Successors:

Visited: A, B, E, K, S, L

Fringe: T,F,C,D (LIFO)

Depth-First Search



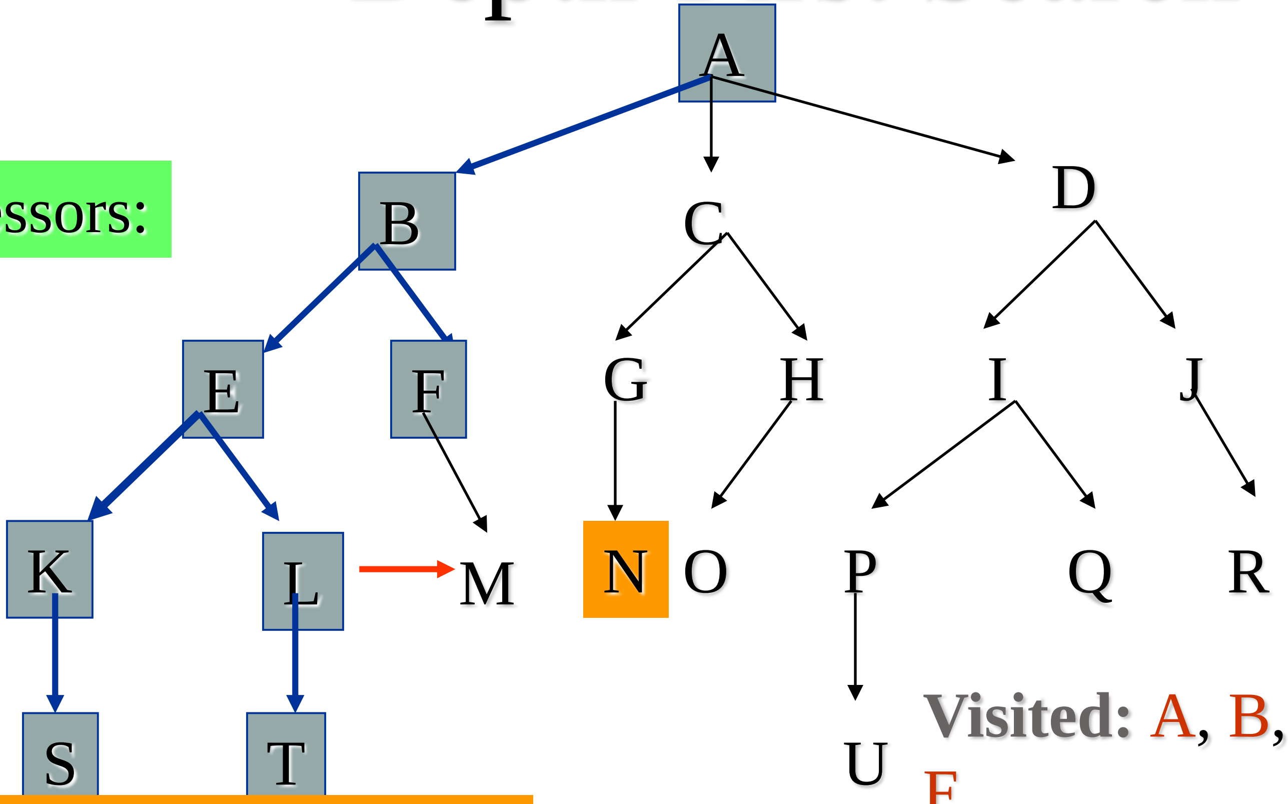
Successors: M

Visited: A, B, E, K, S, L, T

Backtracking

Fringe: F,C,D (LIFO)

Depth-First Search



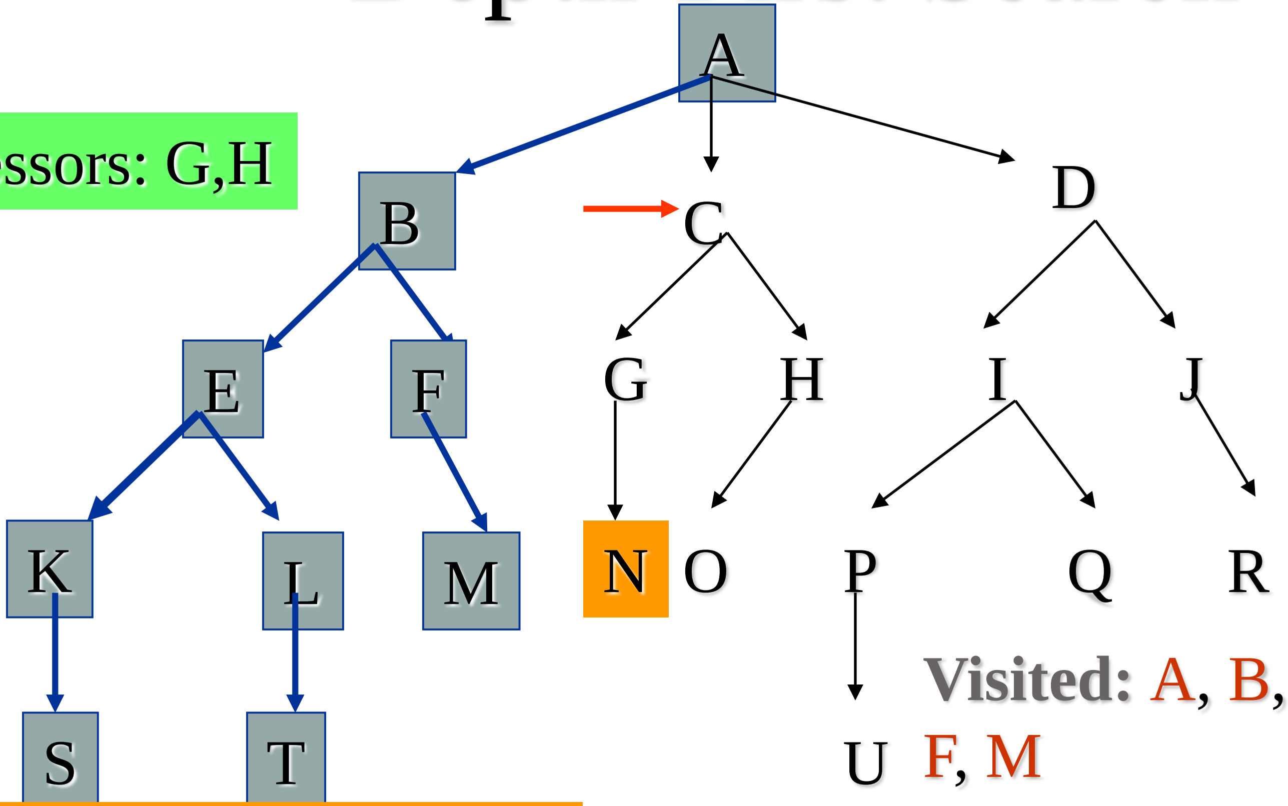
Successors:

Visited: A, B, E, K, S, L, T, F

Fringe: M,C,D (LIFO)

Depth-First Search

Successors: G,H

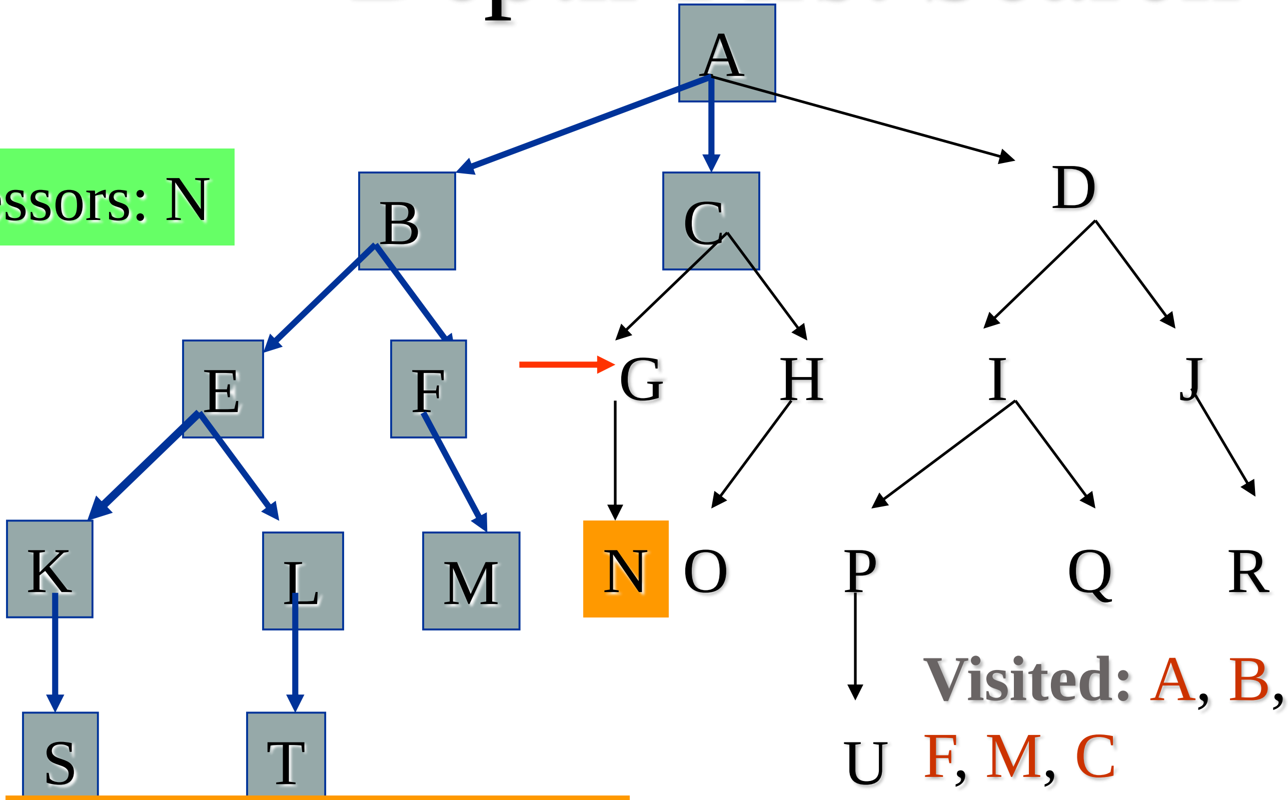


Visited: A, B, E, K, S, L, T, F, M

Fringe: C,D (LIFO)

Backtracking

Depth-First Search



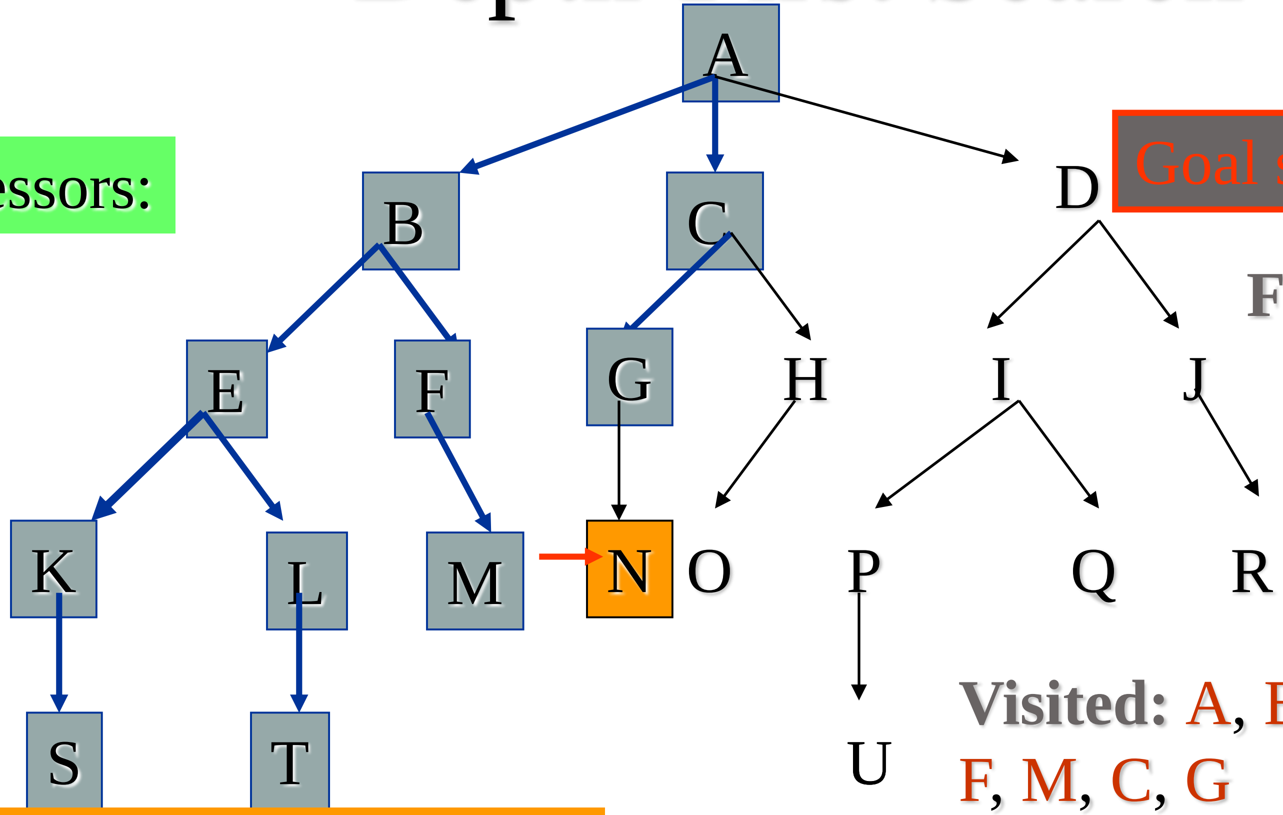
Successors: N

Visited: A, B, E, K, S, L, T, F, M, C

Fringe: G,H,D (LIFO)

Depth-First Search

Successors:



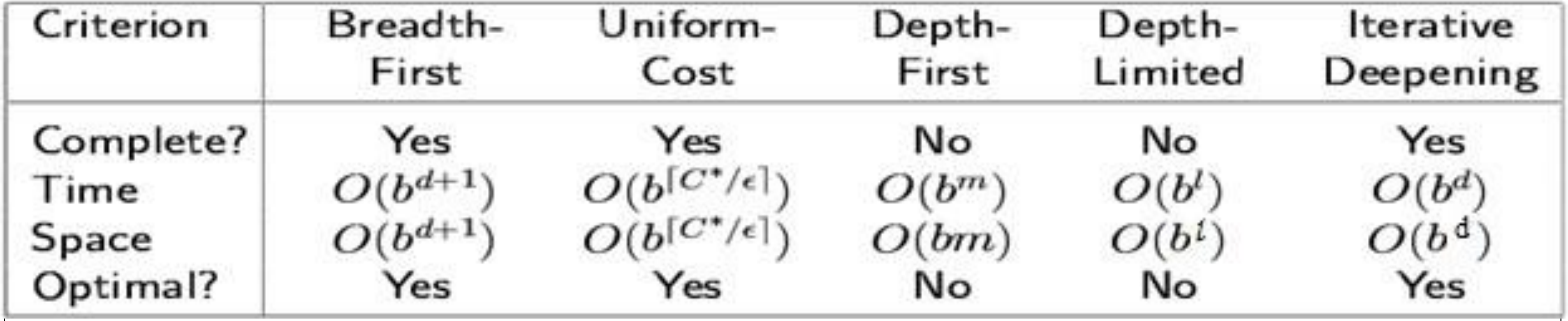
Goal state achieved

Finished search

Visited: A, B, E, K, S, L, T, F, M, C, G

Fringe: N,H,D (LIFO)

Search Comparison:



Criterion	Breadth-First	Uniform-Cost	Depth-First	Depth-Limited	Iterative Deepening
Complete?	Yes	Yes	No	No	Yes
Time	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(b^m)$	$O(b^l)$	$O(b^d)$
Space	$O(b^{d+1})$	$O(b^{\lceil C^*/\epsilon \rceil})$	$O(bm)$	$O(b^l)$	$O(b^d)$
Optimal?	Yes	Yes	No	No	Yes

Page no: 109 Fig. 3.17

Outline

- Searching
- Uninformed Search Techniques
  - Breadth First Search
  - Uniform Cost Search
  - Depth First Search
  - Backtracking Search
  - Depth Limited Search
  - Iterative Deepening Depth First Search
  - Bidirectional Search
  - Search Strategy Comparison
- **Informed Search Techniques**
  - Best First Searching
    - Greedy Search
    - A* Search
  - Local Search Algorithm & Optimization
  - Iterative Improvement Algorithm
    - Hill Climbing Search
    - Simulated Annealing Search
    - Local Beam Search
    - Genetic Algorithm
  - Adversarial Search Techniques
    - Mini-max Procedure
    - Alpha Beta Procedure

Informed Search

- Strategy of problem solving where problem specific knowledge is known along with problem definition
- These search find solutions more efficiently by the use of heuristics
- Heuristic is a search technique that improves the efficiency of the search process
- By eliminating the unpromising states and their descendants from consideration, heuristic algorithms can find acceptable solutions

Informed Search

- Heuristics are fallible i.e. they are likely to make mistakes as well
- It is the approach following an informed guess of next step to be taken
- It is often based on experience or intuition
- Heuristic have limited information and hence can lead to suboptimal solution or even fail to find any solution at all

Outline

- Searching
- Uninformed Search Techniques
  - Breadth First Search
  - Uniform Cost Search
  - Depth First Search
  - Backtracking Search
  - Depth Limited Search
  - Iterative Deepening Depth First Search
  - Bidirectional Search
  - Search Strategy Comparison
- **Informed Search Techniques**
  - **Best First Searching**
    - **Greedy Search**
    - **A* Search**
  - Local Search Algorithm & Optimization
  - Iterative Improvement Algorithm
    - Hill Climbing Search
    - Simulated Annealing Search
    - Local Beam Search
    - Genetic Algorithm
  - Adversarial Search Techniques
    - Mini-max Procedure
    - Alpha Beta Procedure

Best First Search

- Best-first search is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule.
- A node is selected for expansion based on evaluation function f(n).
- A node with lowest evaluation function is expanded first.
- The measure i.e. evaluation function must incorporate some estimate of the cost of the path from a state to the closest goal state
- The algorithm may have different evaluation function, one of such important function is the heuristic function h(n)
  where, h(n) is the estimated cost of the cheapest path from node n to the goal

Best First Search: Types

- Greedy Best First Search
- A* Search

Greedy Best First Search

- The node whose state is judged to be the closest to the goal state is expanded first
- At each step it tries to be as close to the goal as it can
- It evaluates the nodes by using heuristic function
  hence, f(n)=h(n)
  where, h(n)=0, for the goal node
- This search resembles depth first search in the way that it prefers to follow a single path all the way to the goal or if not found till the dead end and returns back up

Greedy Best First Search

- Completeness
    - Can start down an infinite path and never return to any possibilities
    - Not complete
- Optimality
    - Looks for immediate best choice and doesn't make careful analysis of long term options
    - May give longer solution even if shorter solution exists
    - Not optimal

- Space Complexity
    - $O(b^m)$ where, m is the maximum depth of search space, since all nodes have to be kept in memory
- Time Complexity
    - $O(b^m)$

A* Search

- Evaluates node by combining g(n), the cost to reach the node and h(n) the cost to get from node to goal
  f(n)=g(n)+h(n)
  where f(n) is the estimated cost of the cheapest solution through node n
- To find the cheapest solution, the first try node is the mode with lowest g(n)+h(n)

A* Search

- Admissible Heuristic: h(n) is admissible if it never overestimates cost to reach the solution
  example: h_{SLD} (straight line distance) as g(n) is exact, so f(n) is never overestimated

A* Search

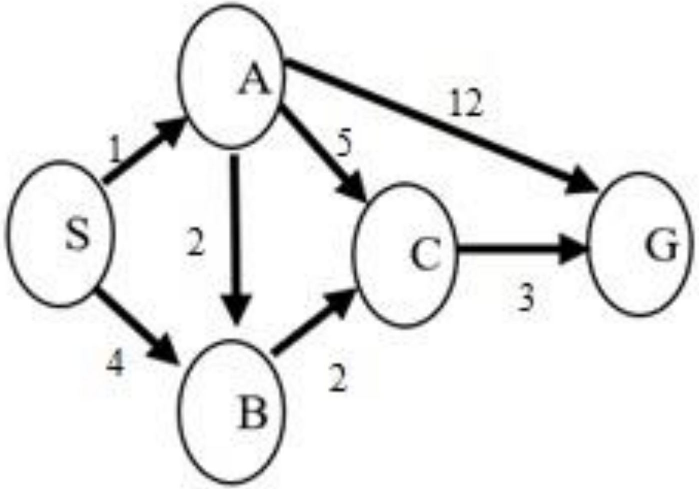
- When h(n) = actual cost to goal.
  - Only nodes in the correct path are expanded.
  -Optimal solution is found.
- When h(n) < actual cost to goal
  -Additional nodes are expanded
  -Optimal solution is found
- When h(n) > actual cost to goal
  -Optimal solution can be overlooked

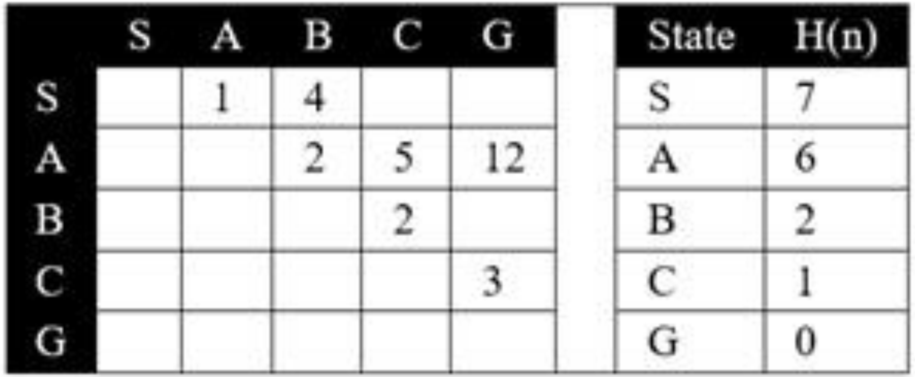
A* Search

- Optimality
  - Optimal if h(n) is admissible
- Completeness
  - Complete if h(n) is admissible
- Space Complexity
  - O(b^d) if h(n) is admissible
- Time Complexity
  - O(b^d) if h(n) is admissible

A* Search

- Example…





	S	A	B	C	G
S		1	4		
A			2	5	12
B				2	
C					3
G					

State	H(n)
S	7
A	6
B	2
C	1
G	0

H(n) is the number paths to reach goal from that state

Outline

- Searching
- Uninformed Search Techniques
    - Breadth First Search
    - Uniform Cost Search
    - Depth First Search
    - Backtracking Search
    - Depth Limited Search
    - Iterative Deepening Depth First Search
    - Bidirectional Search
    - Search Strategy Comparison
- Informed Search Techniques
    - Best First Searching
        - Greedy Search
        - A* Search
    - **Local Search Algorithm & Optimization**
    - Iterative Improvement Algorithm
        - Hill Climbing Search
        - Simulated Annealing Search
        - Local Beam Search
        - Genetic Algorithm
    - Adversarial Search Techniques
        - Mini-max Procedure
        - Alpha Beta Procedure

Local Search Algorithm and Optimization

- Optimization
  - Aim to find the best state according to an objective function

- Local Search Algorithm
  - It operates using a single current state rather than multiple path and generally move only to neighbour of that state

Outline

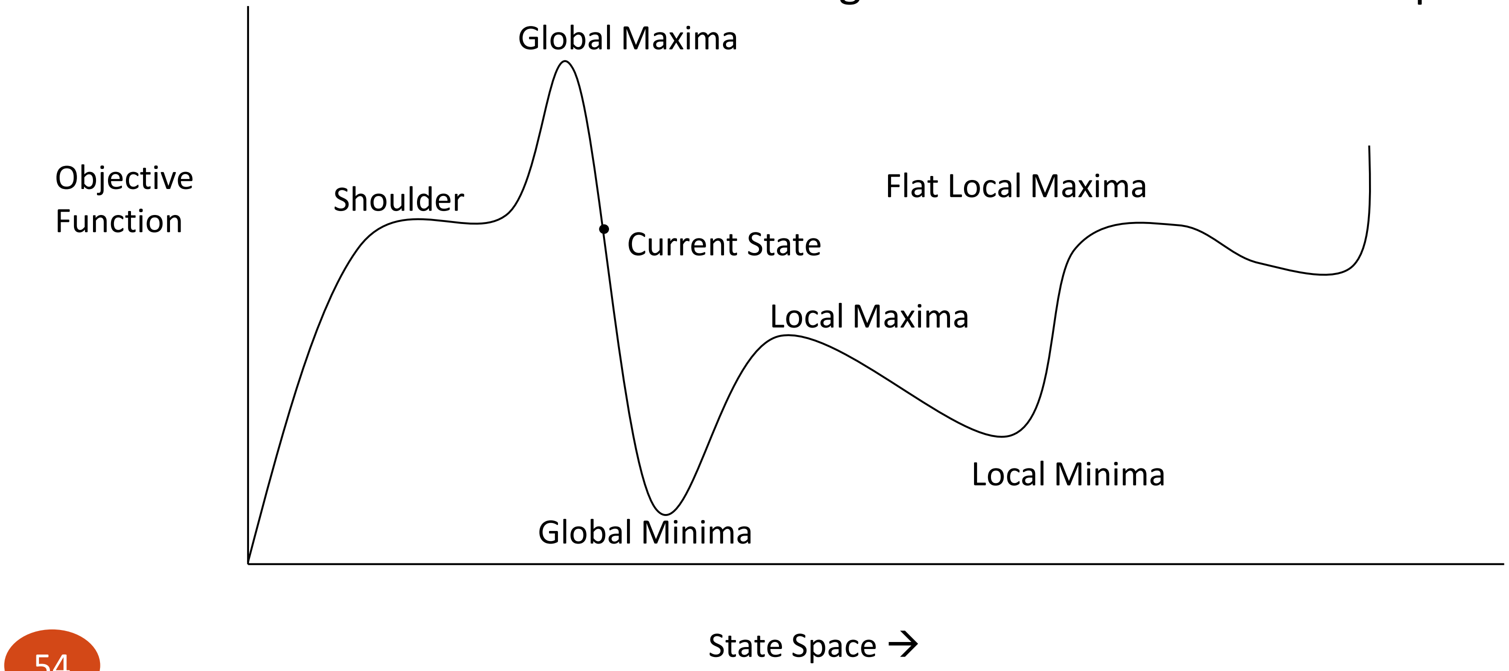
- Searching
- Uninformed Search Techniques
  - Breadth First Search
  - Uniform Cost Search
  - Depth First Search
  - Backtracking Search
  - Depth Limited Search
  - Iterative Deepening Depth First Search
  - Bidirectional Search
  - Search Strategy Comparison
- Informed Search Techniques
  - Best First Searching
    - Greedy Search
    - A* Search
  - Local Search Algorithm & Optimization
  - **Iterative Improvement Algorithm**
    - **Hill Climbing Search**
    - **Simulated Annealing Search**
    - **Local Beam Search**
    - **Genetic Algorithm**
  - Adversarial Search Techniques
    - Mini-max Procedure
    - Alpha Beta Procedure

Iterative Improvement Algorithm

- Consider that all the states are laid down on the surface of a landscape
- The height of a point on a landscape corresponds to process to move around the landscape trying to find the highest peaks, which are the optimal solutions
- Algorithm is suitable for problems where the path of the goal is irrelevant and only final configuration matters

Iterative Improvement Algorithm

Fig.: A One Dimensional State Space Landscape



Iterative Improvement Algorithm: Types

- Hill Climbing Search
- Simulated Annealing Search
- Local Beam Search
- Genetic Algorithm

Hill Climbing Search

- Hill climbing is an optimization technique for solving computationally hard problems.
- Used in problems with “the property that the state description itself contains all the information”
- The algorithm is memory efficient since it does not maintain a search tree
- Hill climbing attempts to iteratively improve the current state by means of an evaluation function
- Searching for a goal state = Climbing to the top of a hill

Hill Climbing Search

- Moves continuously in the direction of increasing value (uphill)
- Doesn’t maintain a search tree so the current node data structure needs only record the state and its objective function value
- Hill climbing doesn’t look ahead beyond the immediate neighbours of the current state
- Also called greedy local search sometimes because it grabs a good neighbour state without thinking about where to go next

Hill Climbing Search

- it often makes very rapid progress towards the solution because it is usually quite easy to improve a bad state
- One move is selected and all other nodes are rejected and are never considered
- Halts if there is no successor

Hill Climbing Search: Problems

- Local Maxima
  - Peak that is higher than each of its neighbouring states but lower than the global maxima
  - Hill climbing halts whenever a local maxima is reached
- Plateau
  - An area of the state space landscape where the evaluation function is flat
  - Can be flat local maxima where no uphill exists or shoulder from which it is possible to progress
  - A hill climbing search might be unable to find its way off the plateau
- Ridges
  - A special kind of local maxima which is the result of a sequence of local maxima that is very difficult for greedy algorithms to navigate
  - It is an area of search space that is higher than the surrounding areas and that itself is at a slope

Simple Hill Climbing

Algorithm

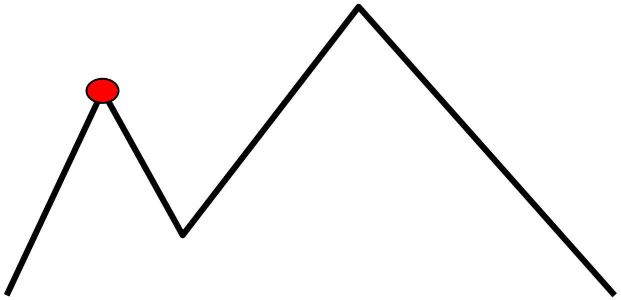
1. determine successors of current state
2. choose successor of maximum goodness
3. if goodness of best successor is less than current state's goodness, stop
4. otherwise make best successor the current state and go to step 1

Hill Climbing (Gradient Search)

- Considers all the moves from the current state.
- Selects the best one as the next state.

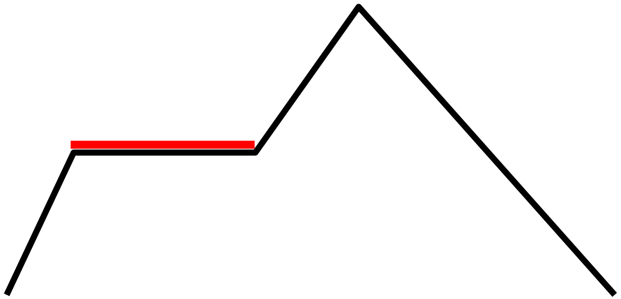
Hill Climbing: Disadvantages

Local maximum: A state that is better than all of its neighbours, but not better than some other states far away.



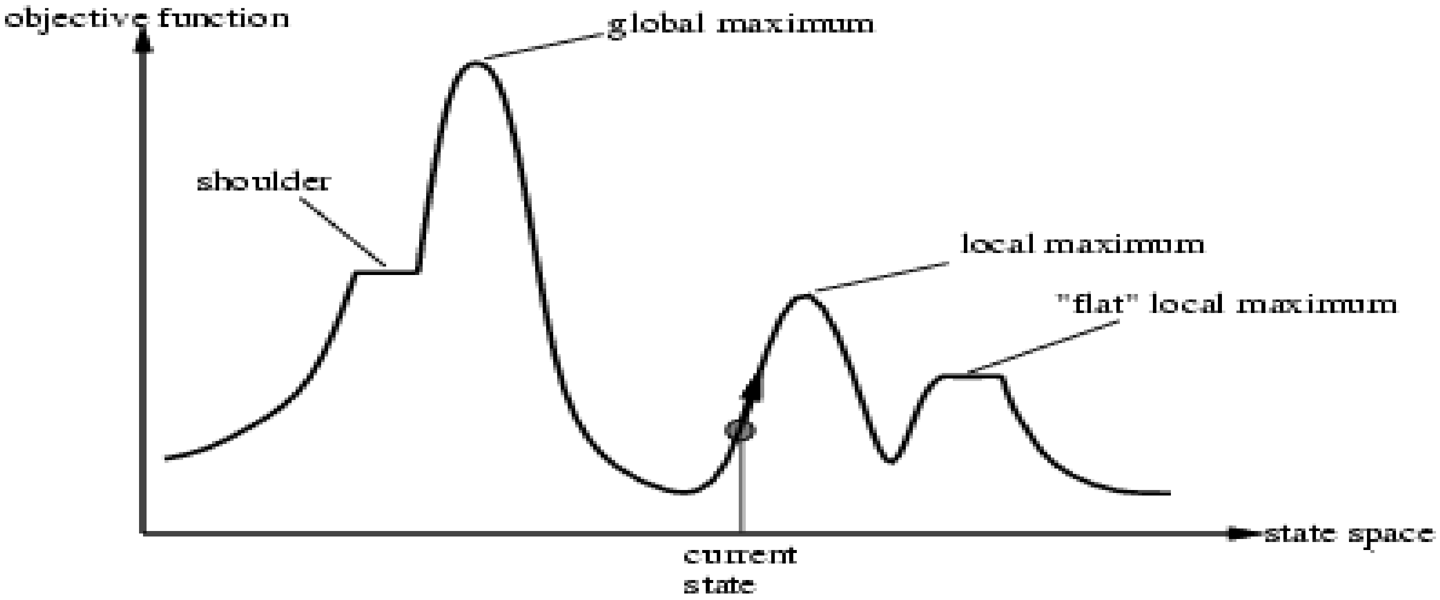
Hill Climbing: Disadvantages

Plateau: A flat area of the search space in which all neighbouring states have the same value.



Hill-climbing search

- Problem: depending on initial state, can get stuck in local maxima



Hill Climbing: Conclusion

- Can be very inefficient in a large, rough problem space.
- Global heuristic - computational complexity.
- Often useful when combined with other methods, getting it started right in the right general neighbourhood.

Hill Climbing Search: Solution to the Problems

- Local Maxima
  - Backtrack to some earlier node and try going to different direction
- Plateau
  - Make a big jump in some direction to try to get a new section of the search space
  - If rule apply single small steps, apply them several times in the same direction
- Ridges
  - Apply two or more rules such as bidirectional search before doing the test
  - Moving in several directions at once

Simulated Annealing Search

- Rather than starting from a different initial state all over again, when a current state shows no progress in the technique
- This search takes some downhill steps so that it can escape that particular local maxima and continue with other peaks in the state space
- A random pick is made for the move
  - If it improves the situation, it is accepted straight away
  - If it worsen the situation, it is accepted with some probability less than 1 which decreases exponentially with the badness of the move i.e. for bad moves the probability is low and for comparatively less bad one, it’s higher

Simulated Annealing Search

- The degree of badness of the move is determined by the amount ΔE, by which the evaluation is worsened
- The probability also depends on the value of a objective function parameter ‘T’
- For low value of T, probability is high and vice versa
- Hence, bad moves are more likely to be allowed at the beginning only
- This method is more common in VLSI layout problem solving, factory scheduling and travelling salesman problems

Local Beam Search

- A path based algorithm
- Keeps track of k-states rather than just one
- Begins with k randomly generated states, at each step, all the successors of all k states are generated
- If any one is the goal, the algorithm halts
- Can quickly become concentrated in a small region of the state space

Genetic Algorithm

- They are adaptive search heuristic algorithm that belongs to a larger part of evolutionary algorithm.
- They are based on the ideas of natural selection and genetic.
- They are commonly used to generate high quality solutions for optimization problem and search problem.

Genetic Algorithm

- A successor state is generated by combining two parent states
- Start with *k* randomly generated states (population)
- A state is represented as a string over a finite alphabet (often a string of 0s and 1s)
- Evaluation function (fitness function). Higher values for better states.
- Produce the next generation of states by selection, crossover, and mutation

Genetic Algorithm

Step 1: Randomly initialize population ‘p’.

Step 2: Determine fitness of population.

Step 3: Until convergence repeat

a. Select parent from population.

b. Cross over and generate new population.

c. Perform mutation on new population.

d. Calculate fitness for new population.

Applications of Genetic Algorithm

1. GA concepts can be applied to the engineering problem such as optimization of gas pipeline systems.
2. GAs are also used to train neural networks, particularly recurrent neural networks.
3. GAs are used for various digital image processing (DIP) tasks as well like dense pixel matching.

Adversarial Search Techniques

- Often known as Games or Game Playing
- Used in competitive multi-agent environments
- Based on game theory
- Deterministic and fully observable environments in which there are two agents whose actions must alternate and in which the utility values at the end of the game are always equal and opposite
- This creates adversarial situation

Optimal Decision in Adversarial Search

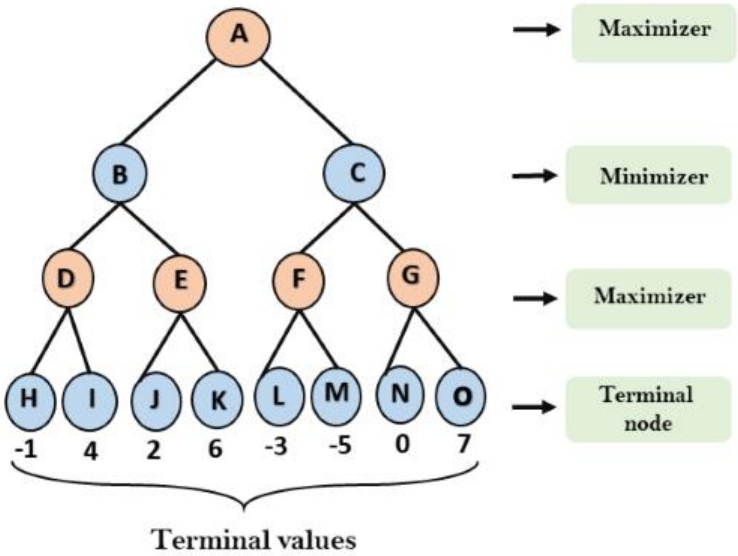
- A game can be defined as a kind of search problem with the following components:
  - Initial State identifying the initial position in the game and identification of the first player
  - Successor Function returning a list of (move, state) pairs
  - Terminal Test which determine that the game is over
  - Utility function which gives a numeric value for the terminal states.

Minimax Algorithm

- Max is considered as the first player in the game and Min as the second player
- This algorithm computes the minimax decision from the current state
- It uses a recursive computation of minimax values of each successor state directly implementing some defined function
- The recursion proceeds from the initial node to all the leaf nodes
- Then the minimax values are backed up through the tree as the recursion unwinds
- It performs the depth first exploration of a game tree in a complete way

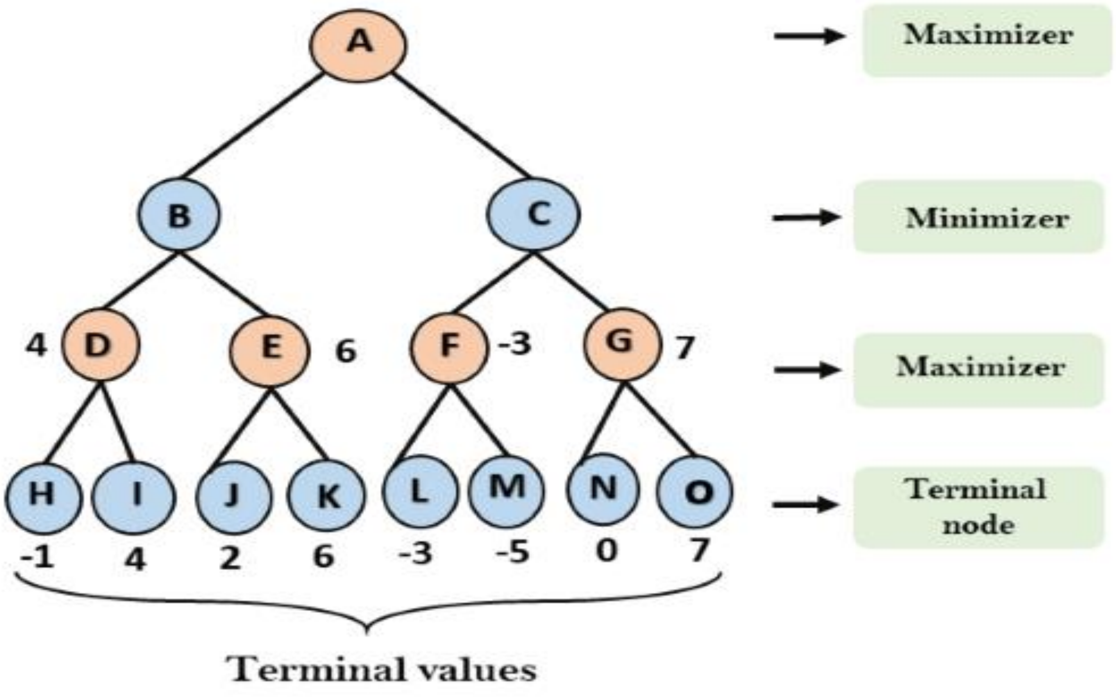
Minimax Algorithm: Working

Step-1: In the first step, the algorithm generates the entire game-tree and apply the utility function to get the utility values for the terminal states. In the below tree diagram, let's take A is the initial state of the tree. Suppose maximizer takes first turn which has worst-case initial value =- infinity, and minimizer will take next turn which has worst-case initial value = +infinity.



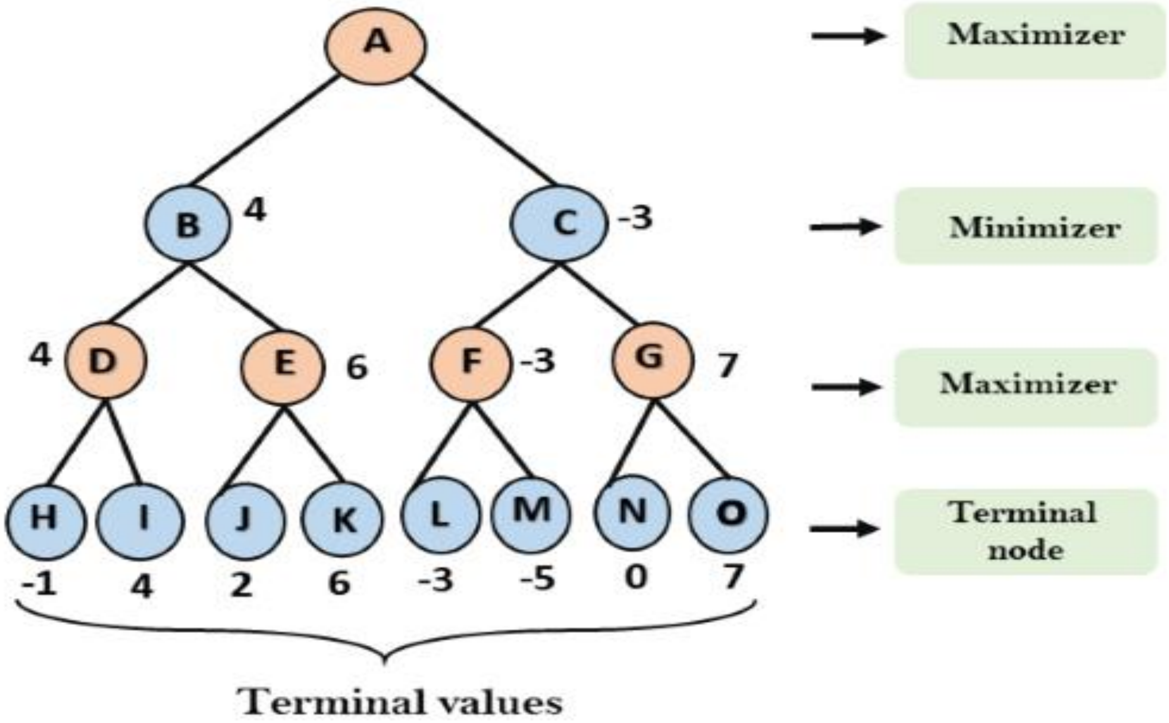
Step 2: Now, first we find the utilities value for the Maximizer, its initial value is -∞, so we will compare each value in terminal state with initial value of Maximizer and determines the higher nodes values. It will find the maximum among the all.

- For node D max(-1, -∞) => max(-1,4)= 4
- For Node E max(2, -∞) => max(2, 6)= 6
- For Node F max(-3, -∞) => max(-3,-5) = -3
- For node G max(0, -∞) = max(0, 7) = 7



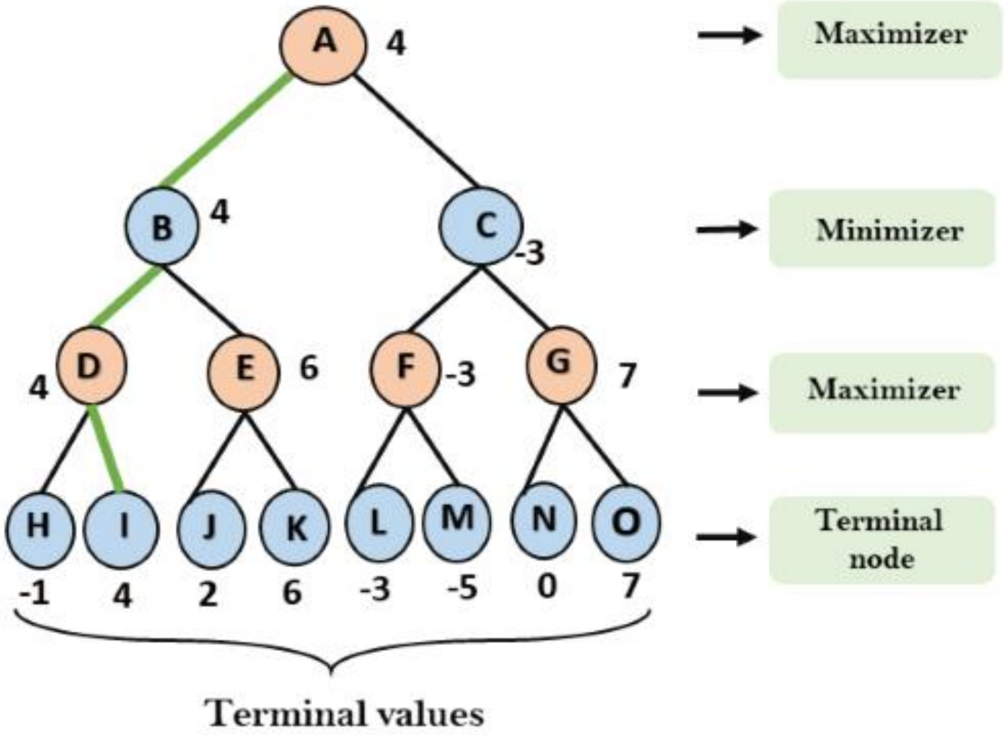
Step 3: In the next step, it's a turn for minimizer, so it will compare all nodes value with +∞, and will find the 3rd layer node values.

- For node B= min(4,6) = 4
- For node C= min (-3, 7) = -3



Step 4: Now it's a turn for Maximizer, and it will again choose the maximum of all nodes value and find the maximum value for the root node. In this game tree, there are only 4 layers, hence we reach immediately to the root node, but in real games, there will be more than 4 layers.

- For node A max(4, -3)= 4



Alpha Beta Pruning

- Minimax algorithm has to examine exponentially increasing number of moves
- As the exponential rise can’t be avoided Pruning cut it into halves
- By not considering a large part of the tree number of states to be calculated is cut down
- When applied to a standard minimax tree, alpha beta pruning returns the same move as minimax would, but prunes away the branches which couldn’t possibly influence the final decision
- Alpha beta pruning could be applied to the trees of any depth

Alpha Beta Pruning: Example

The two-parameter can be defined as:

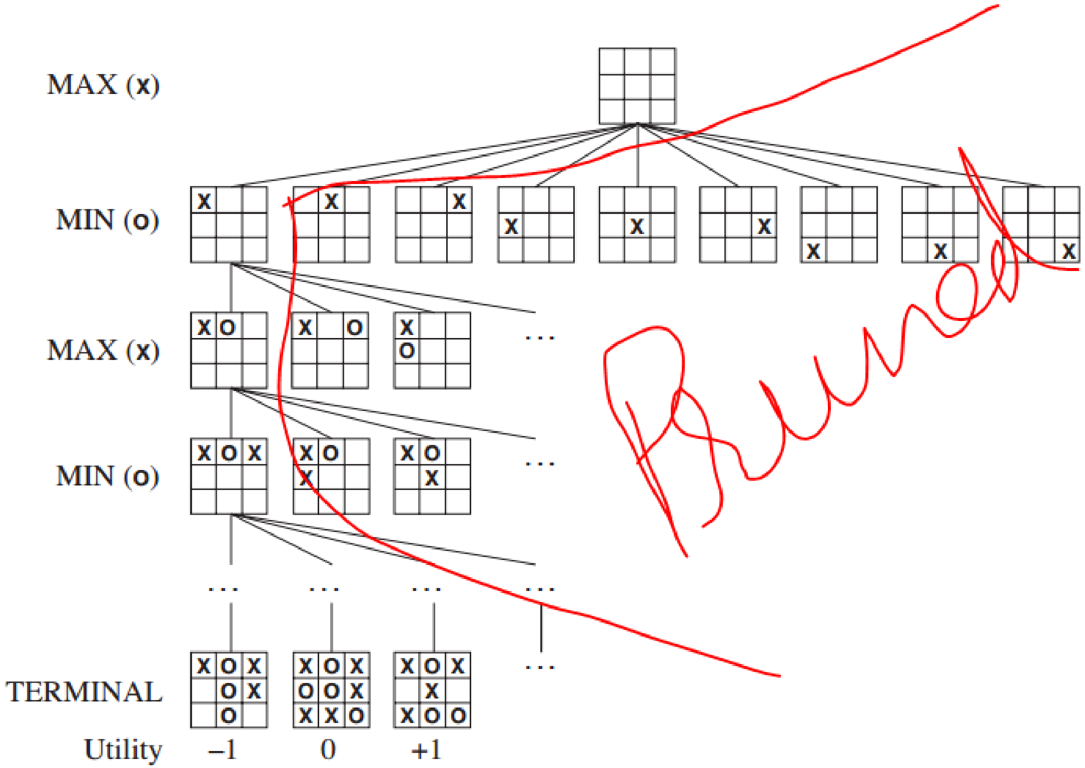
1. **Alpha:** The best (highest-value) choice we have found so far at any point along the path of Maximizer. The initial value of alpha is -∞.

2. **Beta:** The best (lowest-value) choice we have found so far at any point along the path of Minimizer. The initial value of beta is +∞.

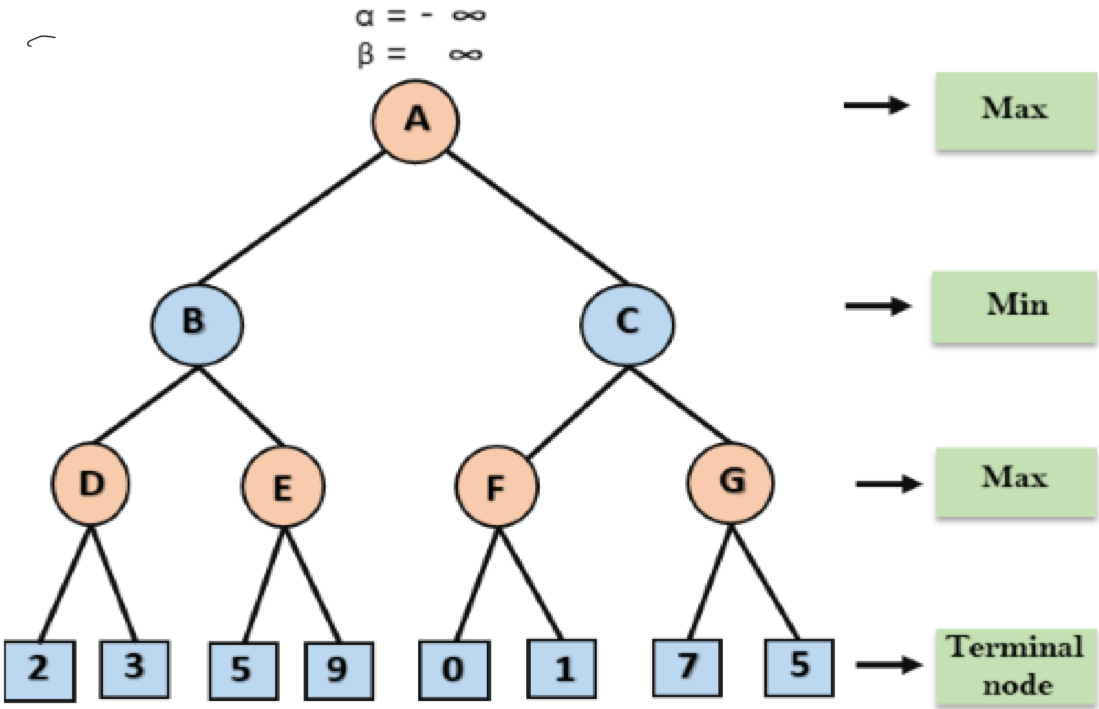
For pruning,

$$\boldsymbol{\alpha >= \beta}$$

Alpha Beta Pruning

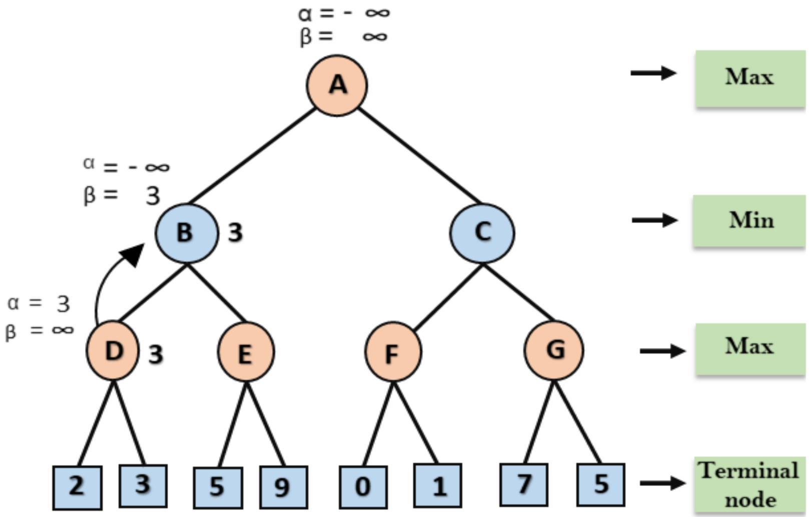


➢**Step 1:** At the first step the, Max player will start first move from node A where α= -∞ and β= +∞, these value of alpha and beta passed down to node B where again α= -∞ and β= +∞, and Node B passes the same value to its child D.



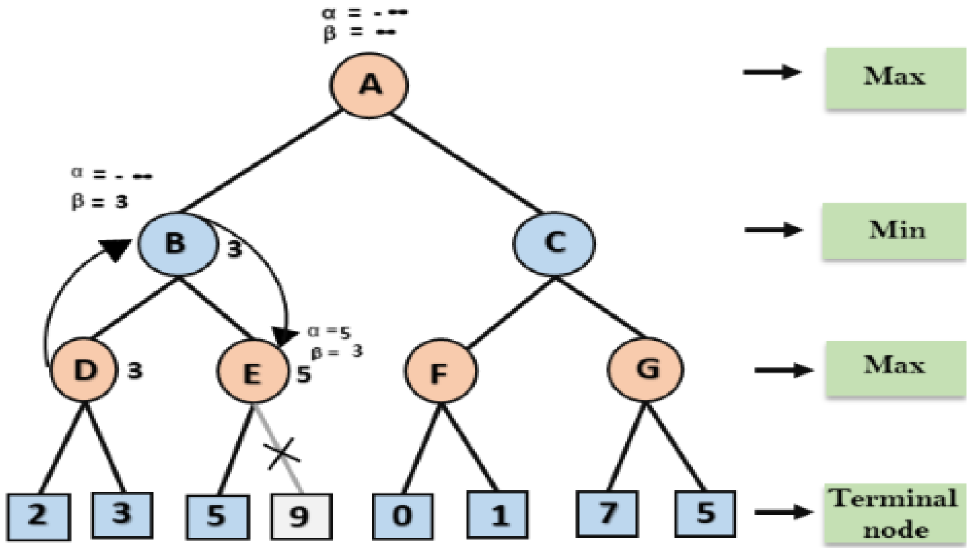
Step 2: At Node D, the value of α will be calculated as its turn for Max. The value of α is compared with firstly 2 and then 3, and the max (2, 3) = 3 will be the value of α at node D and node value will also 3.

Step 3: Now algorithm backtrack to node B, where the value of β will change as this is a turn of Min, Now β= +∞, will compare with the available subsequent nodes value, i.e. min (∞, 3) = 3, hence at node B now α= -∞, and β= 3.



In the next step, algorithm traverse the next successor of Node B which is node E, and the values of α= -∞, and β= 3 will also be passed

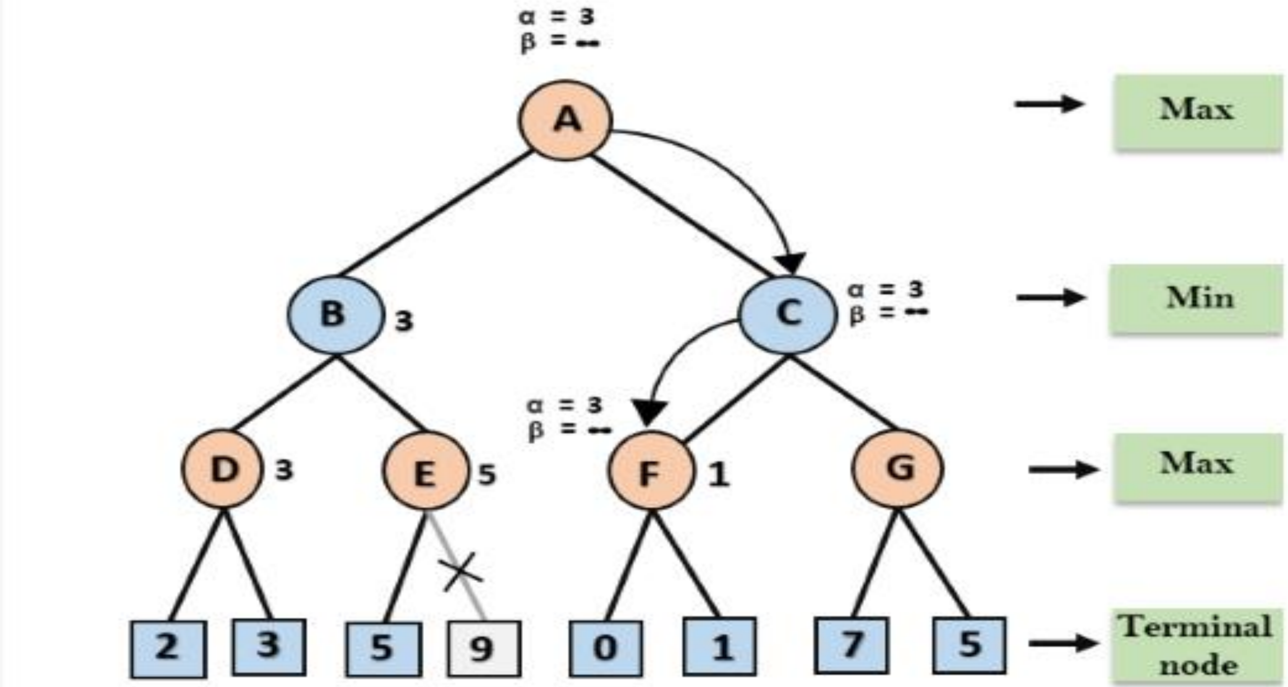
Step 4: At node E, Max will take its turn, and the value of alpha will change. The current value of alpha will be compared with 5, so max (-∞, 5) = 5, hence at node E α= 5 and β= 3, where α>=β, so the right successor of E will be pruned, and algorithm will not traverse it, and the value at node E will be 5.



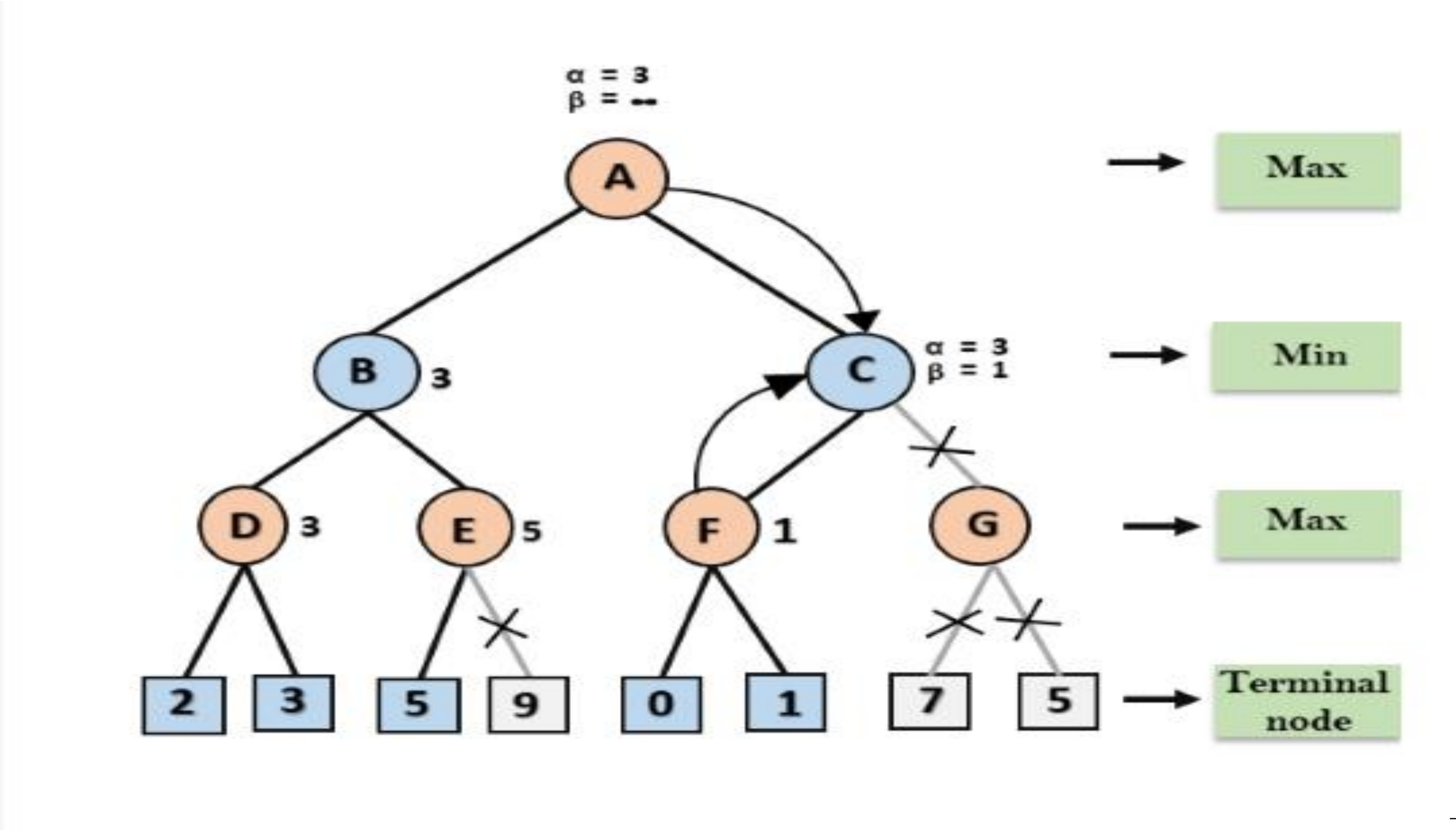
Step 5: At next step, algorithm again backtrack the tree, from node B to node A. At node A, the value of alpha will be changed the maximum available value is 3 as max (-∞, 3)= 3, and β= +∞, these two values now passes to right successor of A which is Node C.

At node C, α=3 and β= +∞, and the same values will be passed on to node F.

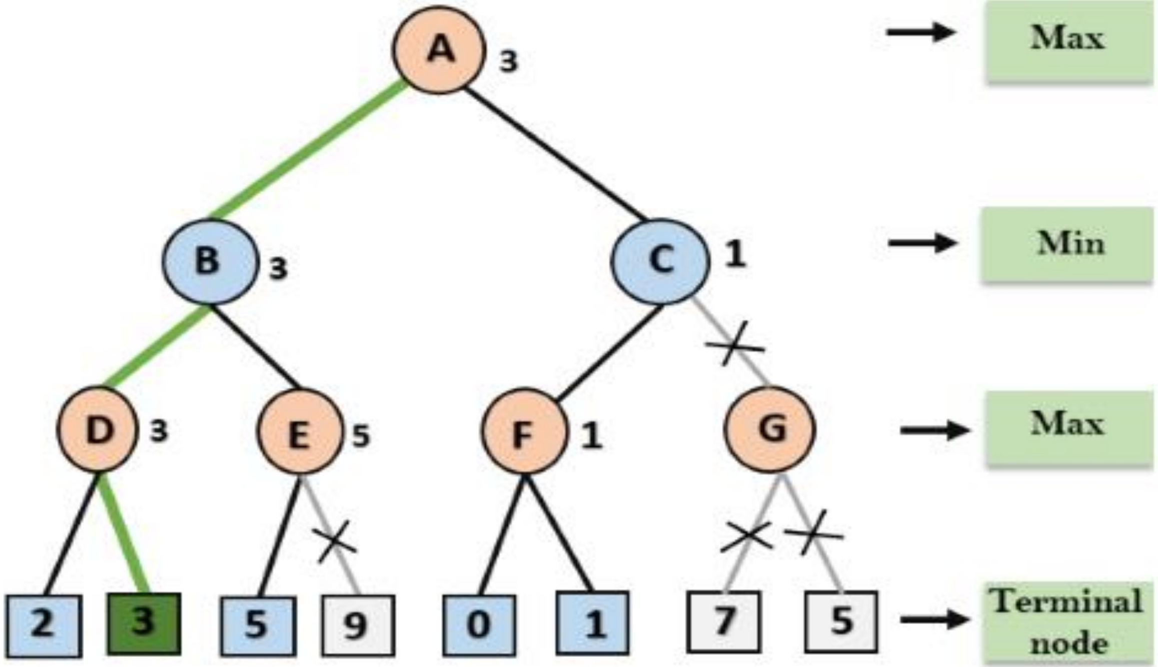
Step 6: At node F, again the value of α will be compared with left child which is 0, and max(3,0)= 3, and then compared with right child which is 1, and max(3,1)= 3 still α remains 3, but the node value of F will become 1.



Step 7: Node F returns the node value 1 to node C, at C α= 3 and β= +∞, here the value of beta will be changed, it will compare with 1 so min (∞, 1) = 1. Now at C, α=3 and β= 1, and again it satisfies the condition α>=β, so the next child of C which is G will be pruned, and the algorithm will not compute the entire sub-tree G.



Step 8: C now returns the value of 1 to A here the best value for A is max (3, 1) = 3. Following is the final game tree which is the showing the nodes which are computed and nodes which has never computed. Hence the optimal value for the maximizer is 3 for this example.



References

- Russell, S. and Norvig, P., 2011, Artificial Intelligence: A Modern Approach, Pearson, India.
- Rich, E. and Knight, K., 2004, Artificial Intelligence, Tata McGraw hill, India.

- ***Note: For more details, read Book provided by the college***