Problem Solving Using Search

Background

Q: How do Human handles any problem?

A: Using Brain/Mind

Q: How does Human Mind solve the problem?

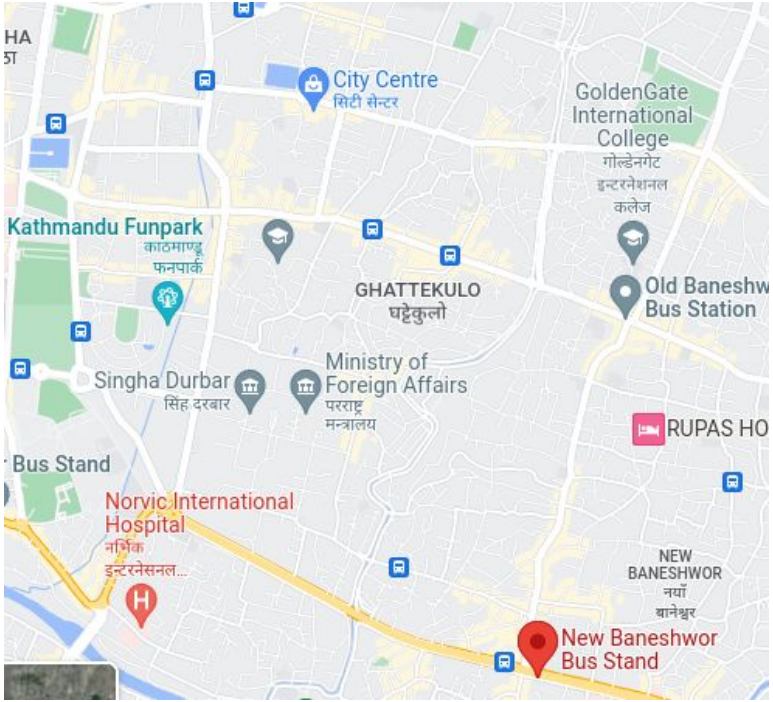
A: Human mind uses search to solve problems, (might not be always successful.)

Q: What is Human Mind?

A: It’s an agent.

Example:

Problem: How to reach city center from new baneshwor?



Why this chapter??

- To Understand and Learn:
  - How we can make an AI / Agent to solve the problem.
  - Problem solving using State Space Search
  - State Space and way to formulate a well defined problem
  - Strategies of search in state space/ ways to find solution in the state space.
  - Methods to compare the strategies
- Solving a problem means finding a sequence of actions that will eventually lead to desired goal.
- Finding a sequence of action when correct action to take is not obvious is called plan.

Problem Solving - Definition

- Problem Solving is a process of generating solutions from observed data.
- Finding a sequence of actions that form a path to a goal state.
- Problem is characterized by set of states and set of operations and a set of goals.
- The method of solving problem through AI involves the process of:
  - Defining the search space,
  - Deciding start and goal states
  - Finding the path from start state to goal state through search space.

Problem Solving

- Search space or problem space is an abstract space which has all valid states that can be generated by the application of any combination of operators on states.
- Problem space can have one or more solutions
- Solution is a combination of operators and states that achieve the goals
- Search refers a search of solution in problem space.

Problem Solving

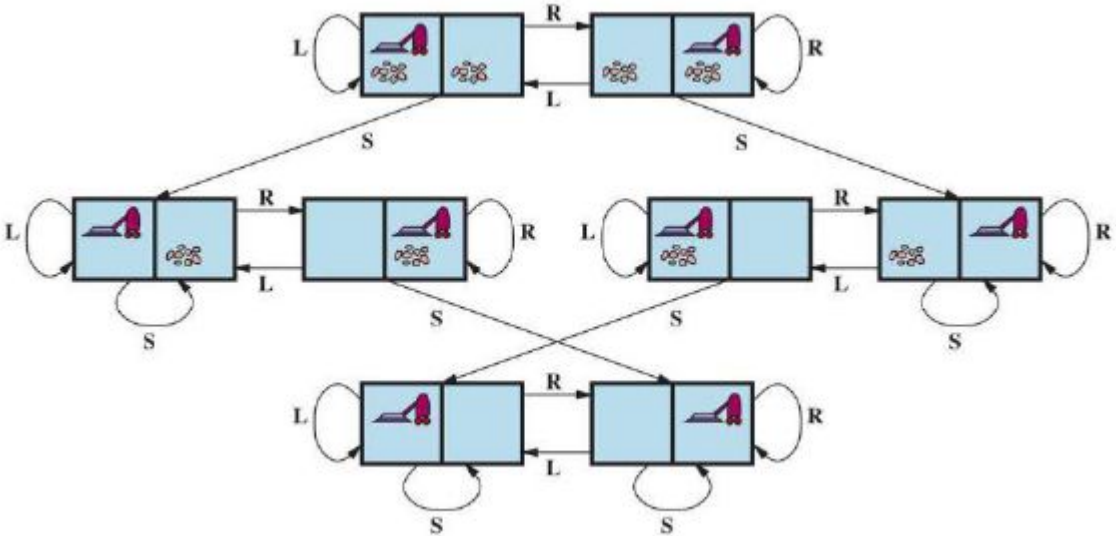
Here we assume that the agents always have access to the information of the world, such as a map. Thus, the four method/phases to solve the problem by an agent or AIs:

- Goal Formulation: Decide the goal
- Problem Formulation:
  - Define a problem
  - Define state and actions allowed to reach a goal
  - This is abstract modeling of the problem
- Search:
  - Sequence of action agents take to reach the goal
  - Such a sequence is the solution.
  - Agent might have to simulated multiple sequences
- Execution:
  - Agent can now execute the solution.

State Space

- Definition:
  - State space is defined as a set of all possible states for a given problem
- Problem modeling technique.
- formalizes a problem in terms of initial state, goal state and actions

Problem Formulation: Using Cleaning Robot



Formulation

- States: All possible state
- Initial State: Any state can be designed as an initial state.
- Actions: {moveLeft(), moveRight(), doClean()}
- Transition Model:
  - doClean() removes any dirt from the cell
  - moveLeft() moves toward the left, left cell exist
  - moveRight() moves toward the right, right cell exist
- Goal States:
  - The states in which every cell is clean
- Action Cost: Each action costs 1.

Example: 8 tile problem

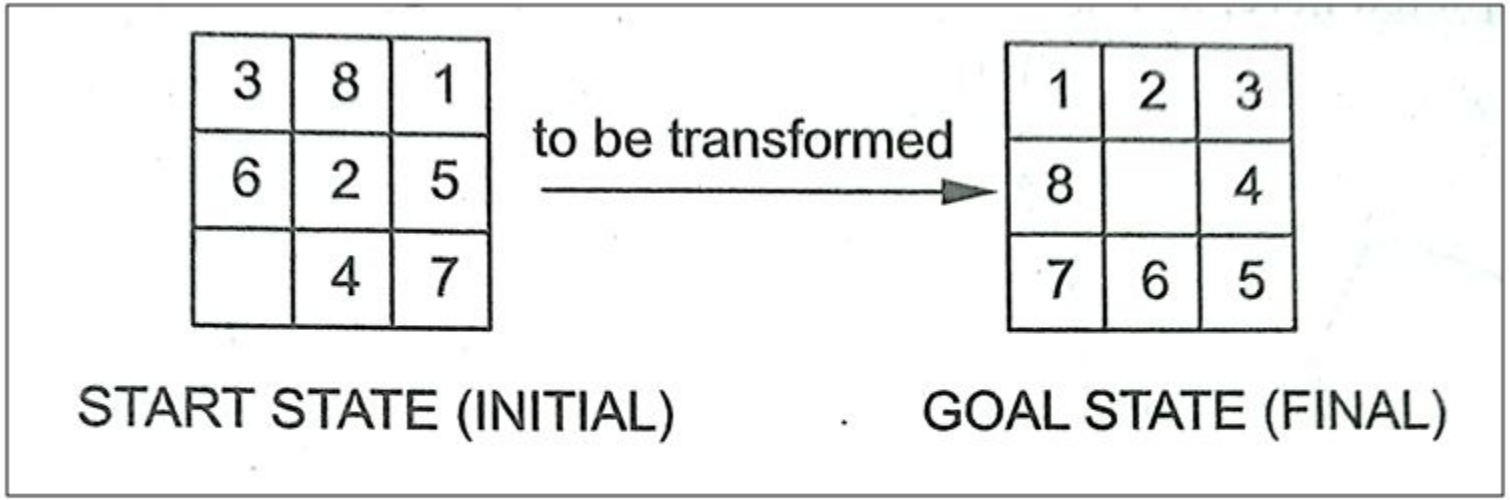


Fig: State Space Search

Problem Solving By Searing

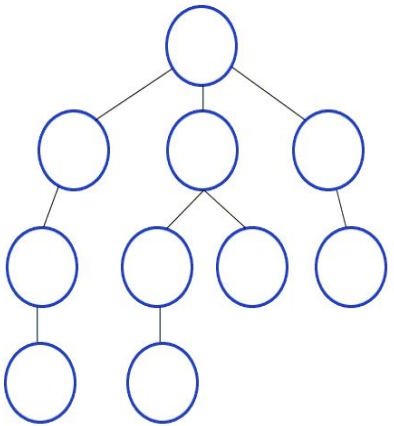
- Since we can formulate a problem and represent it using state space, which in turn is graph, we can use graph search algorithms
- There are two strategies to solve:
  - Uninformed Search
  - Informed Search

Performance Evaluation of Search Strategies

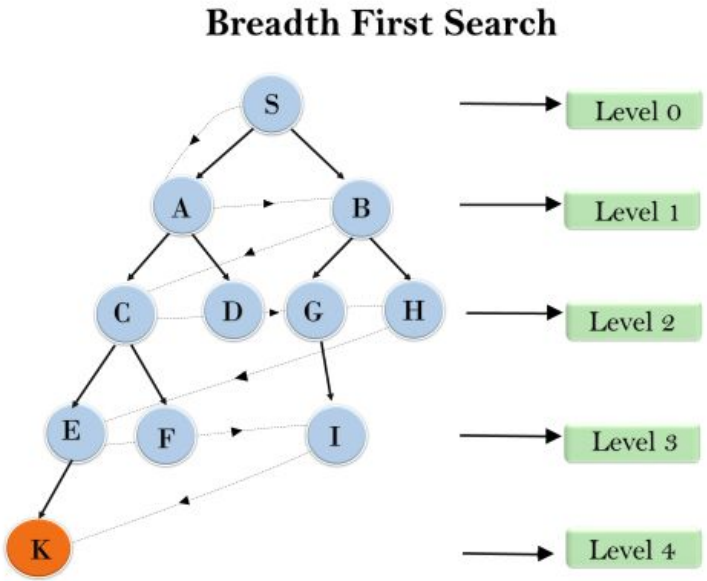
- It is criteria used to choose search algorithms
- Evaluating the algorithms performance considering four factors:
  - **COMPLETENESS**
    - Is the algorithm guaranteed to find a solution when there is one, and to correctly report failure when there is not?
  - **COST OPTIMALITY**
    - Does the algorithm find a solution to the problem with the lowest path cost of all solutions?
  - **TIME COMPLEXITY**
    - How long does it take to find the solution? This can be measured in seconds or more abstractly by the number of states and actions considered?
  - **SPACE COMPLEXITY**
    - How much memory is needed to perform the search?

Breadth First Search (BFS)

- Searches breadthwise in tree or graph, so it is called BFS
- Starts searching from root node of the tree and expands all the successor node at the current level before moving
- to next nodes
- It is implemented using queue data structure.



Diagram


Breadth First Search
S
A
B
C
D
G
H
E
F
I
K
Level 0
Level 1
Level 2
Level 3
Level 4

Breadth First Search Continued …

Advantages

- BFS provides a solution if any solution exists
- If there are more than one solutions for a given problem, then BFS will provide the minimal solution which requires the least number of steps

Disadvantages

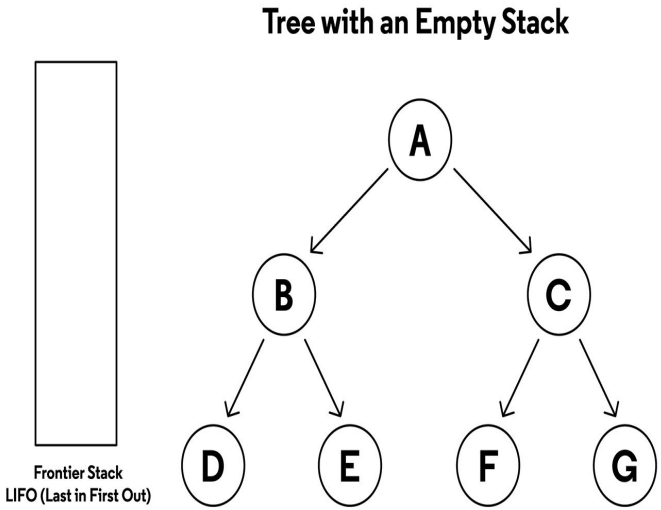
- Consumes a lot of memory since each level of tree must be saved into the memory to expand the next level
- BFS needs a lots of time if the solution is far away from the root node.

Properties

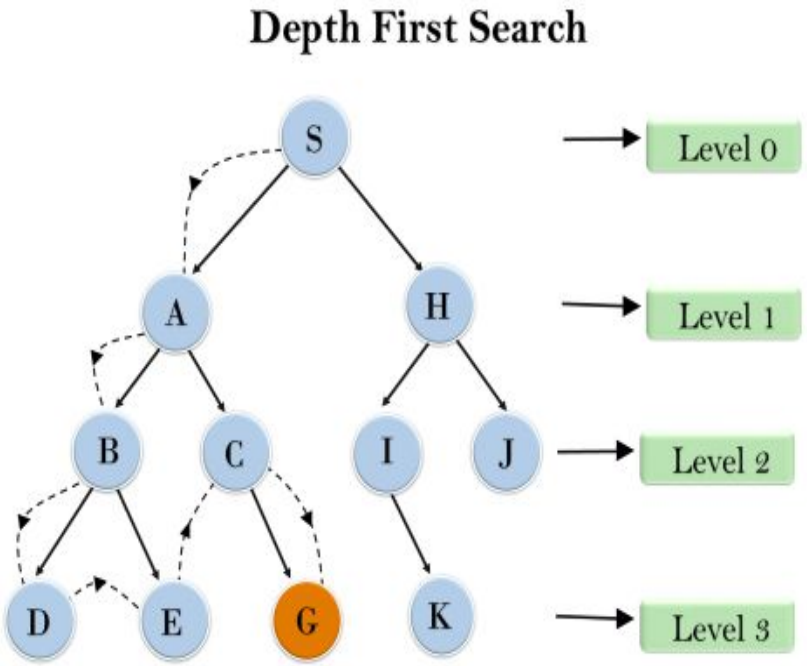
- **Time Complexity**: Number of nodes traversed in BFS until the shallowest node. d = depth of the shallowest solution and b is a node at every state. $T(b) = 1 + b^2 + b^3 + ... + b^d = O(b^d)$
- **Space Complexity**: $O(b^d)$
- **Completeness:** If the shallowest goal node is at some finite depth then BFS will find a solution.
- **Optimality:** If the past cost is non-decreasing function of the depth of the node.

Depth First Search

- DFS is recursive a recursive algo for tree traversing
- It starts from the root node and follows each path to its greatest depth node before moving to the next path.
- DFS uses stack for its implementation.



Diagram


Depth First Search
S
A
H
B
C
I
J
D
E
G
K
Level 0
Level 1
Level 2
Level 3

Depth First Search Continued..

Advantages:

- Requires very less memory as it only store a stack of nodes on the path from root node to the current node.
- Takes less time to reach to the goal node than BFS if it traverses in the right path

Disadvantages

- Possibility that many states keep reoccurring and there is no guarantee of finding the solution.
- DFS goes deep down searching and sometime it may go to the infinite loop

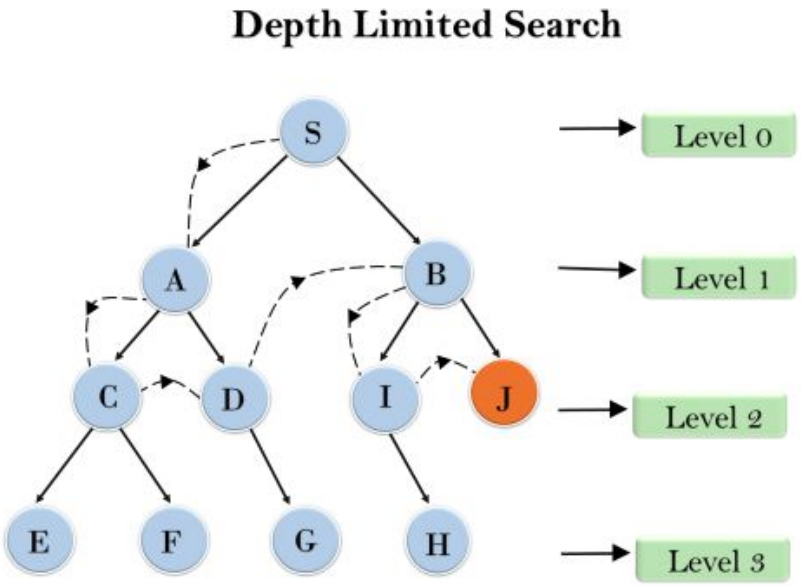
Properties

- **Time Complexity:** node traversed by the algorithm Given as **T(n) = 1+n^2+n^3+...+n^m = O(n^m)** where **m** is maximum depth much larger than shallowest solution depth(**d**)
- **Space complexity:** stores only single path from root node, hence equivalent to the size of list which **O(bm)**
- **Optimal:** it is non optimal as it may generate large number of steps or high cost to reach to goal state.
- **Completeness:** Complete with in finite state space

Depth Limited Search

- To overcome the problem of infinite depth in DFS, it can be limited to predetermined depth.
- Node at the depth limit will treat as it has no successor nodes further.
- It can be terminated with two conditions of failure:
  - Standard value Failure: problem does not have solution
  - Cutoff failure value: no solution for the problem within a given depth limit.

Diagram


Depth Limited Search
S
A
B
C
D
I
J
E
F
G
H
Level 0
Level 1
Level 2
Level 3

Depth Limit Search Continued ..

Advantages

- Depth Limit Search is Memory efficient

Disadvantages

- Depth limited search also has disadvantage of incompleteness
- It may not be optimal if the problem has more than one solution

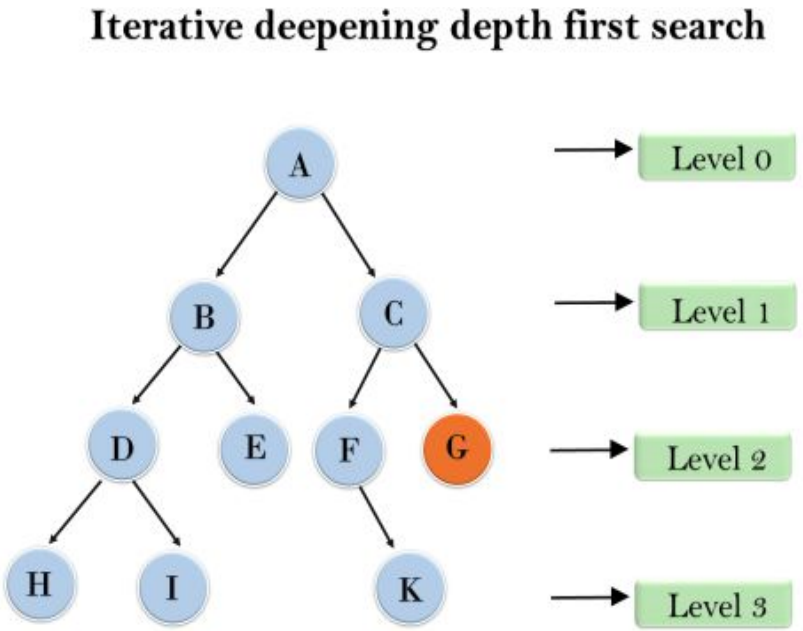
Properties

- **Time Complexity: O(b^l)**
- **Space Complexity: O(bl)**
- **Optimal:** special case of DFS, not optimal even if **l > d**
- **Completeness:** complete if solution exist is the depth limit.

Iterative Deepening Search

- Combination of DFS and BFS. FInds out the best depth limit and does search by gradually increasing the limit until a goal is found.
- Performs depth first search up to certain depth limit and keeps increasing depth limit after each iteration until the goal is reached.
- Combines the benefit of Breadth-first search fast search and depth-first search memory efficiency.
- Useful when space is large and depth of goal node is unknown.

Diagram



1'st Iteration-----> A

2'nd Iteration----> A, B, C

3'rd Iteration------>A, B, D, E, C, F, G

4'th Iteration------>A, B, D, H, I, E, C, F, K, G

In the fourth iteration, the algorithm will find the goal node.

Iterative Deepening Search Continued..

Advantages:

- Combines benefits of BFS and DFS search algorithm in terms of fast search and memory efficiency

Disadvantages:

- Repeats all the work of the previous phase

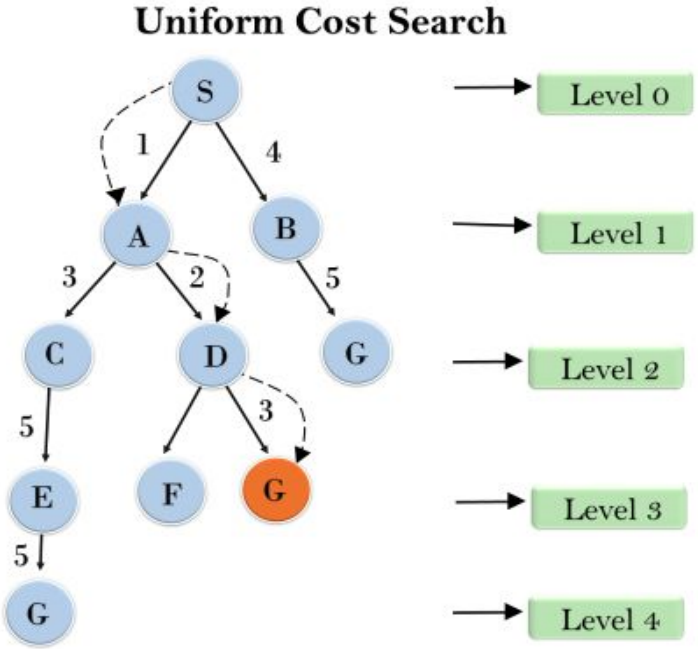
Properties

- **Time Complexity: b** is the branching factor and depth is **d** then the time complexity is **O(b^d)**
- **Space Complexity: O(bd)**
- **Optimal:** optimal if the path cost is non-decreasing function of the depth of the node.
- **Completeness:** complete if the branching factor is finite

Uniform Cost Search

- Search algorithm used for traversing a weighted tree or graph
- Goal: to find path which has lowest cumulative cost to reach the goal
- Expands node according to their path cost from the root node.
- Priority queue is used to implement UCS
- It gives maximum priority to lowest cumulative cost.
- SImilar to BFS if path cost of all edge is same.

Diagram


Uniform Cost Search
S
A
B
C
D
G
E
F
G
G
1
4
3
2
5
5
3
5
Level 0
Level 1
Level 2
Level 3
Level 4

Uniform Cost Search continued…

Advantages:

- Optimal because at every state path with the least cost is chosen

Disadvantages:

- Does not cares about the number of steps involved in searching and only concerned about path cost. Due to which this algorithm may be stuck in an infinite loop.

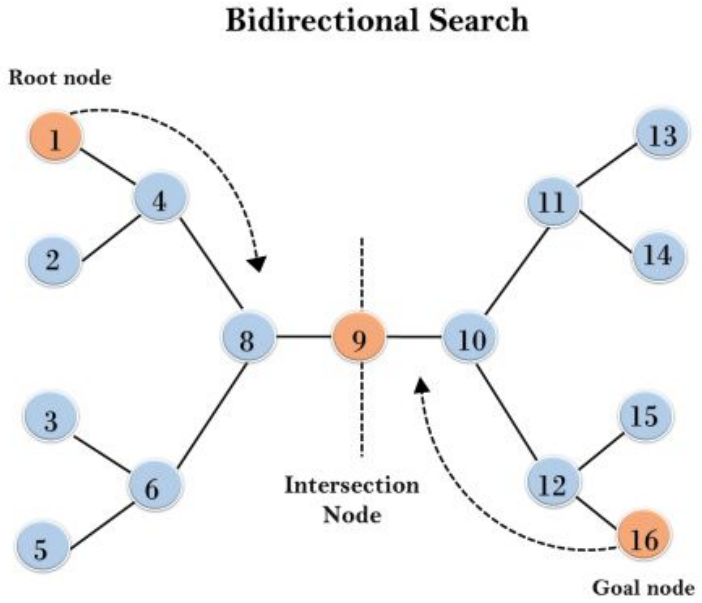
Properties

- **Time Complexity**: Let **C*** be the cost of the optimal solution, and **ε** be the each step to get closer to the goal node. Then the number of steps is **C*/ε+1**. Here +1 is taken as we start from state **0** and end to **C*/ε**. Hence **O(b^(1+[C*/ε]))**
- **Space complexity:** The same logic is for space complexity. So the worst case space complexity is **O(b^(1+[C*/ε]))**
- **Optimal:** optimal as it selects the path with lowest path cost
- **Completeness:** it is complete, if there is solution, it will find it.

Bidirectional Search

- Runs two simultaneous searches, one from initial state called as forward search and other from the goal state called as backward search.
- Replaces one simple search graph with two small subgraph.
- The search stops when these two graphs intersect each other.
- It can use search techniques such as BFS, DFS, DLS etc.

Diagram


Bidirectional Search
Root node
1
4
2
8
9
10
13
11
14
3
6
5
Intersection
Node
15
12
16
Goal node

Bidirectional Search Continued..

Advantages:

- It is fast
- Requires less memory

Disadvantages:

- DIfficult to implement
- Should know the goal state in advance

Properties

- **TIme Complexity:** bidirectional using BFS is **O(b^d)**
- **Space Complexity: O(b^d)**
- **Optimal:** It is optimal
- **Completeness:** complete if we use BFS in both searches

Informed/Heuristic Search

- Search which incorporates the use of domain-specific knowledge in the process of choosing which node to visit next in the search process.
- Why??
  - Because it helps to reduce the size of the search space using some evaluation function called heuristics
  - An evaluation function is applied to each node to assess how promising it is in leading to the goal.

Admissible Heuristic

- Admissible Heuristic
  - A heuristic function h(n) is said to be admissible on a graph with goals iff h(n) <= h*(n) for every node in G.
  - It never overestimated the cost to reach the goal state.
  - It is by nature optimistic because they think the cost of solving the problem is less than it actually is.
  - Examples:
    - Route Finding Problem: Straight Line distance
    - 8-puzzle problem : Number of misplaced tiles

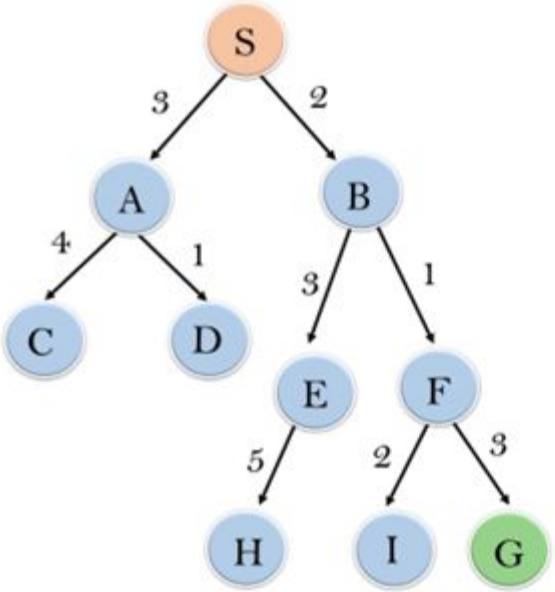
Informed Search Strategy

- Greedy Best First Search
- A*
- Hill Climbing Search
- Simulated Annealing Search

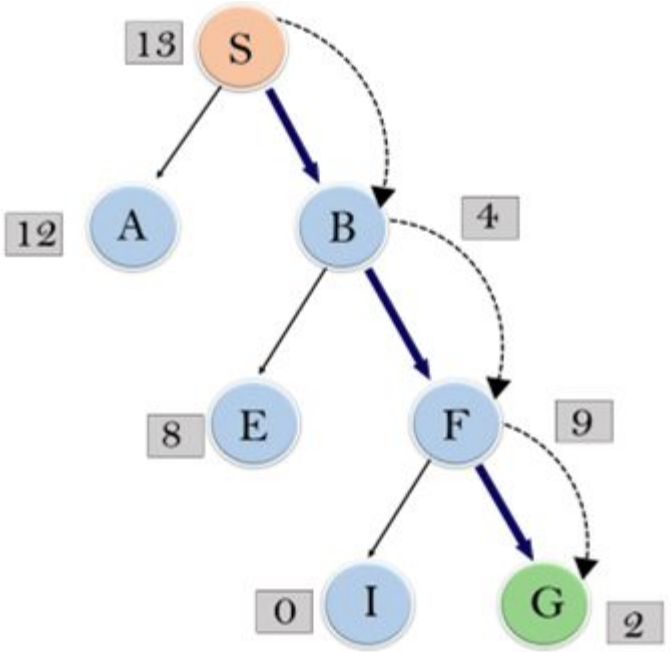
Greedy Best First Search

- Greedy best first search always selects the path which appears best at that moment.
- It is combination of DFS and BFS search allows us to take advantage of both algo.
- It uses heuristic function h(n) ; estimated cost from node n to goal
- Here we expand the node which is closed to the goal node and the closest cost is estimated using heuristic function.
- Implemented using priority queue

Diagram: Greedy Best First Search



node	H (n)
A	12
B	4
C	7
D	3
E	8
F	2
H	4
I	9
S	13
G	0



Greedy Best First Search Continued..

Advantages:

- It can switch between BFS and DFS by gaining the advantages of both the algorithm
- More efficient than BFS and DFS algorithm

Disadvantages:

- Can behave as an unguided depth first search in worst case
- Can get stuck in a loop as DFS
- Not so optimal

Properties

- **TIme Complexity: O(b^m)**
- **Space Complexity: O(b^m)**
- **Optimal:** Not optimal
- **Completeness:** It is also incomplete, even if the given state space is finite

A*

- Originally called as “Algorithm A”
- * is used to denote optimality, and Algorithm A is optimal given a heuristic it became, as it known now as, A*
- A* is specific case of best first search, where the heuristic evaluation function is defined as f(n) = g(n) + h(n)
    - g(n) : sum of actual cost incurred while travelling from root node to node n
    - h(n) : an estimate of cost from node n to goal node
    - f(n) : estimated total cost od path through node to goal

A* Continued…

Advantages:

- A* search algorithm is best algorithm than other search algo
- A* is complete and optimal
- Can solve complex problems

Disadvantages:

- Does not always produce the shortest path as it mostly based on heuristics and approximations
- A* search has complexity issues
- Main drawback is memory requirement as it keeps all generated node in the memory

Optimality

A* is cost optimal depending on the properties of the heuristic. A key property is admissibility: an admissible heuristic is one that never overestimates the cost to reach a goal. With an admissible heuristic, A* is cost optimal.

Proof(Using Contradiction)

Suppose the optimal path has cost C*, but the algorithm returns a path with cost C>C*. Then there must be some node n which is on the optimal path and unexpanded because if all the nodes on the optimal path had been expanded then it would have returned that optimal solution. So, using the notation,

Proof continues..

g*(n) = the cost of the optimal path from the start to n

h*(n) = the cost optimal path from n to the nearest goal

We have,

f(n) > C* (otherwise n would have been expanded)

f(n) = g(n) + h(n) (by definition)

f(n) = g*(n) + h(n) (because n is on an optimal path)

f(n) <= g*(n) + h*(n) (because of admissibility, h(n)<=h*(n))

f(n) <= C* (by definition, C* = g*(n) + h*(n))

Proof continues…

The first and last line form a contradiction, so the supposition that algo returns the optimal path must be wrong. And Hence A* returns only cost optimal paths. QED

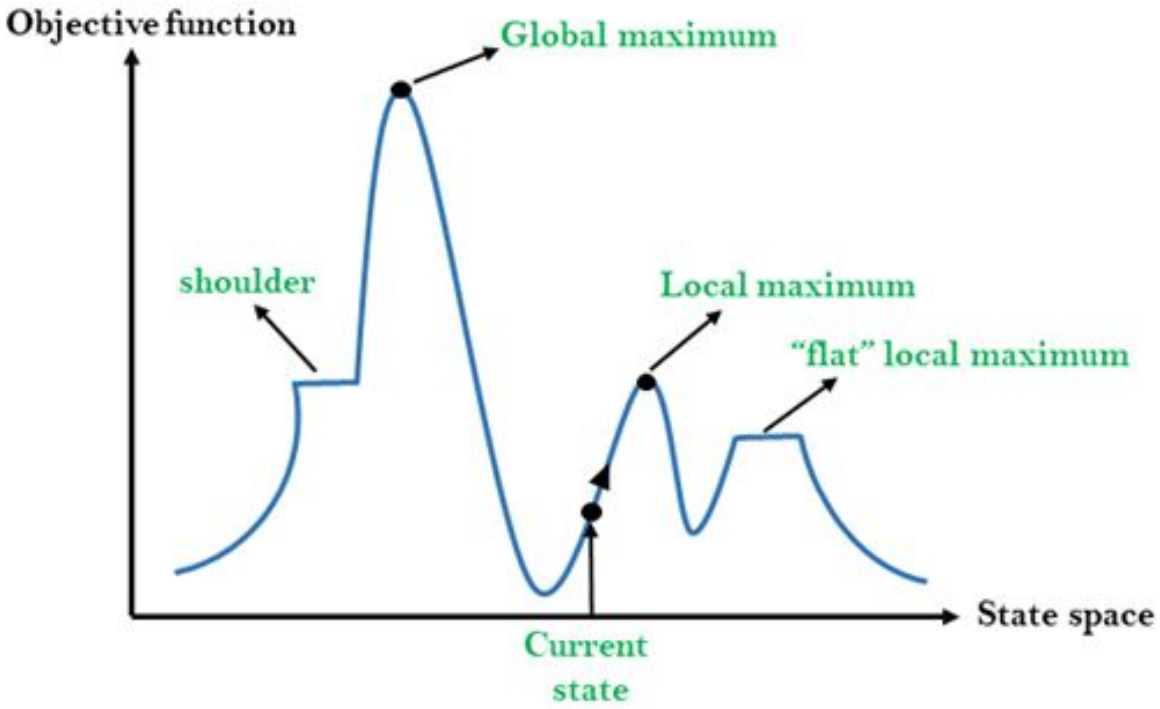
Consistency

- Slightly stronger property
- h(n) is consistent if, for every node n and every successor n’ of n generated by an action a, we have:
  - h(n) <= c(n,a,n’)+h(n’) (triangle inequality property)
  - Where c(n,a,n’) cost of the action from n to n’
- Every consistent heuristic is admissible (but not vice versa), so with consistent heuristic A* is cost optimal.

Hill Climbing

- Start from base of Hill and walk upward till the top
- Alternatively, Starting with initial state and keep improving the solution until its optimal
- Discards all the states which do not look promising or seem unlikely to lead us to the goal state
- To take decision to discard the states it uses heuristics which indicates how close the current state is to the goal state

State Space Diagram of Hill climbing



State Space Diagram Continued…

The state-space landscape is a graphical representation of the hill-climbing algorithm which is showing a graph between various states of algorithm and Objective function/Cost.

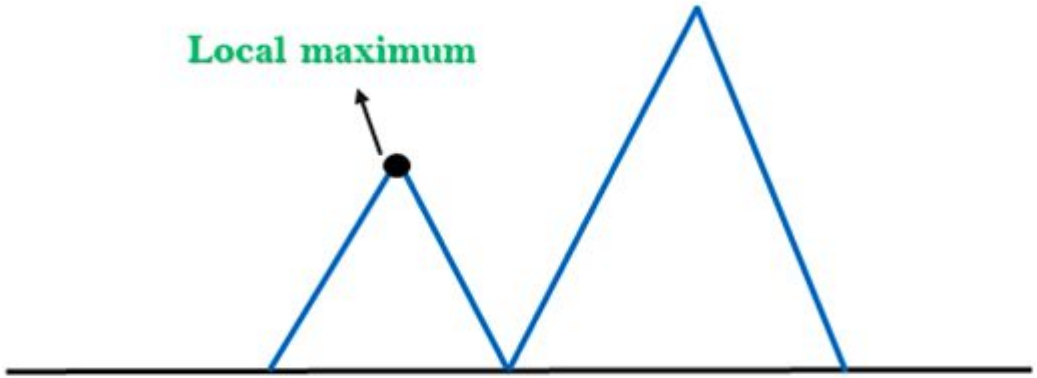
On Y-axis we have taken the function which can be an objective function or cost function, and state-space on the x-axis. If the function on Y-axis is cost then, the goal of search is to find the global minimum and local minimum. If the function of Y-axis is Objective function, then the goal of the search is to find the global maximum and local maximum.

Different regions in the state space landscape

- **Local Maximum:** state which is better than its neighbor states, but there is also another state which is higher than it.
- **Global Maximum:** best possible state of state space,highest value of the objective function
- **Current state:** state in the diagram where the agent is currently present
- **Flat local maximum:**

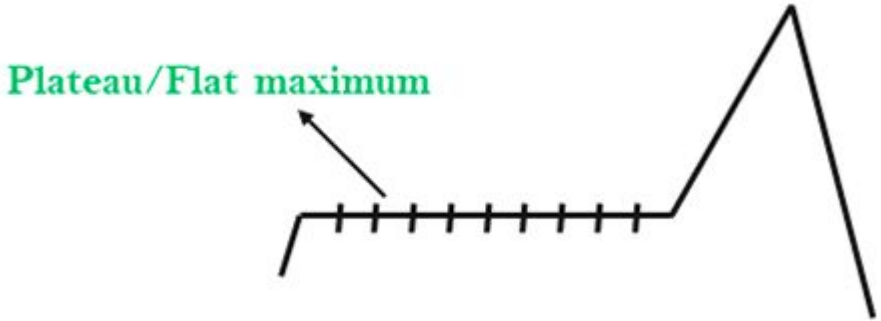
Problems in Hill Climbing Algorithm

- **Problem(i)**
  - **Local Maximum**: It is just peak state which is better than neighboring states but not the best state
- **Solution**: Backtracking technique can be a solution of the local maximum in state space landscape. Create a list of promising path so that algorithm can backtrack the search space and explore other paths as well.



Problems in Hill Climbing Algorithm

- **Problem(ii)**
  - **Plateau**: Flat area where all the neighboring states contains the same value, because of this algo does not find any best direction to move. It might be lost in the plateau area.
- **Solution**: The solution for the plateau is to take big steps or very little steps while searching, to solve the problem. Randomly select a state which is far away from the current state so it is possible that the algorithm could find non-plateau region.



Problems in Hill Climbing Algorithm

- **Problem(iii)**
  - **Ridges**:A ridge is a special form of the local maximum. It has an area which is higher than its surrounding areas, but itself has a slope, and cannot be reached in a single move.
- **Solution**: with the use of bidirectional search, or by moving in different directions, we can improve this problem.

Simulated Annealing

- Similar to Hill Climbing Algorithm
- Picks random move instead of picking the best move.
- If the move leads to the improvement of the current situation ,
  - It is accepted as a step towards the solution state,
- Else it accepts the move having a probability less than 1
- Yields both efficiency and completeness.

Game Playing

- One of the most interesting areas of AI
- Game Playing is often called as Adversarial Search
- It is a search problem with following components:
  - Initial state of the game
  - Operators defining the legal moves/valid actions
  - Successor function/ Transition model
  - Terminal and Goal Test
  - Path Cost
- Good Domain to explore machine learning, because;
  - Provides structure task making easier to measure success or failure
  - Does not require a large amount of knowledge. Solvable by straightforward search from starting to final state.

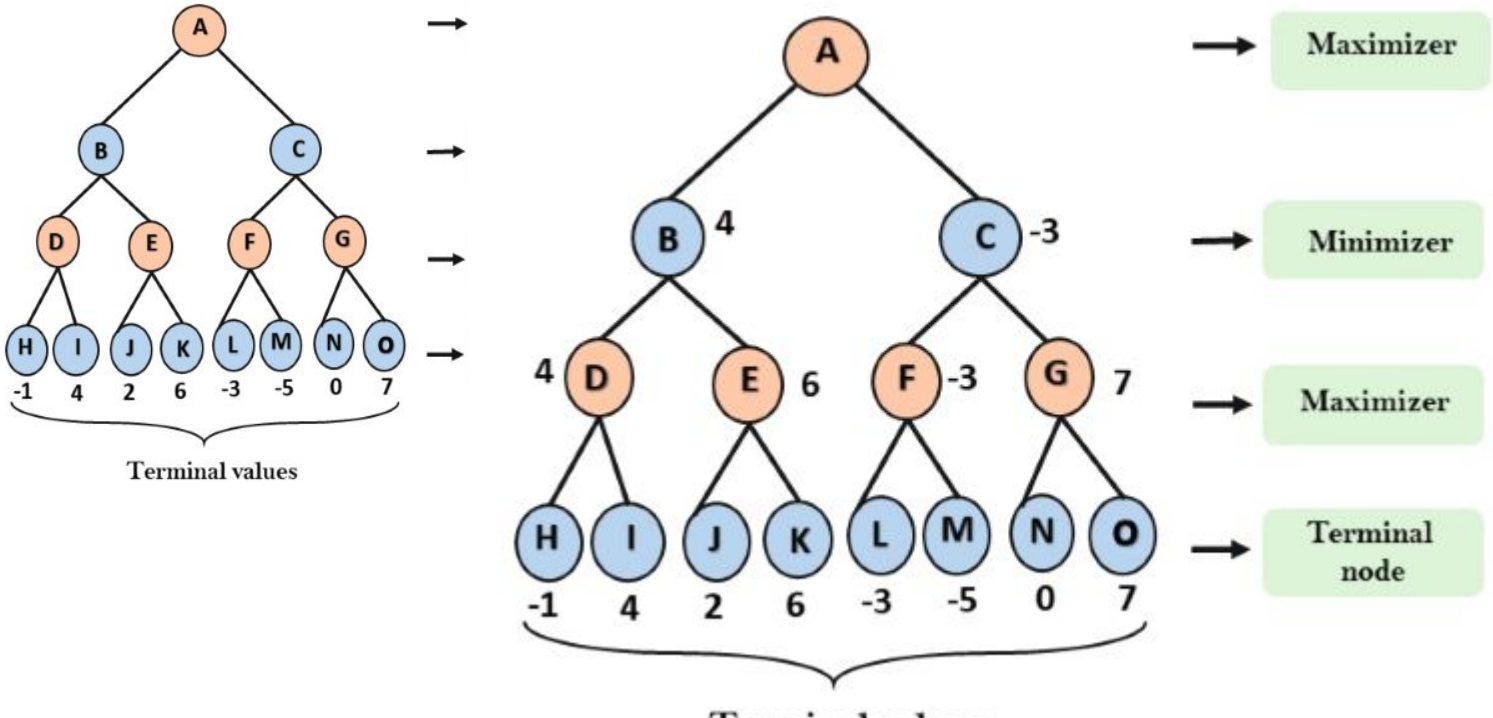
MiniMax

- Recursive/ Backtracking algorithm which is used on decision-making, game theory and AI
- Used to find the optimal move for a player, assuming that the opponent is also playing optimally.
- Eg: Two player onlines games: Tic Tac Toe, Checkers, Chess etc

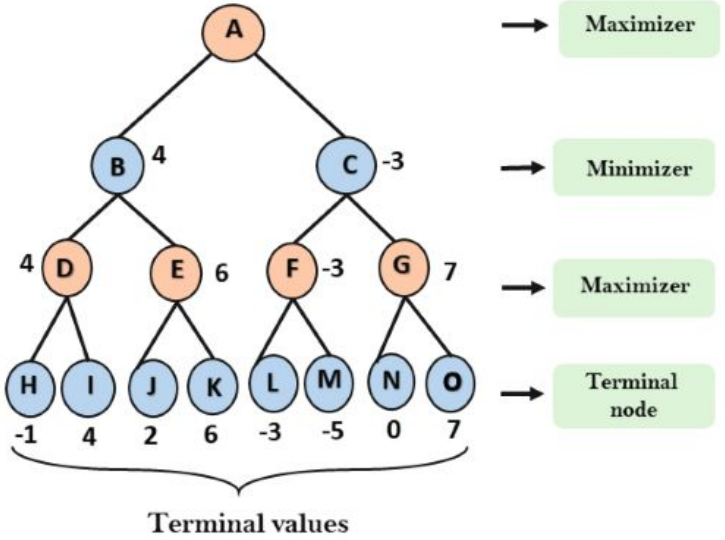
Working

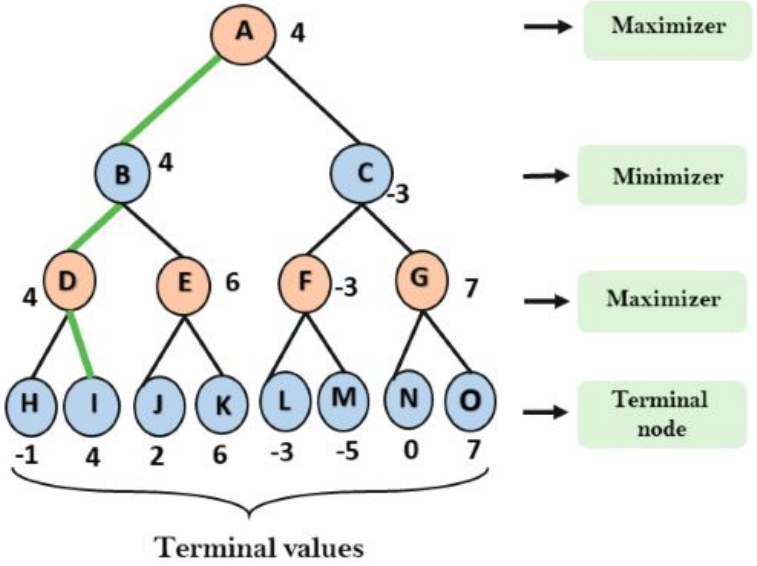
- Create an entire game tree
- Evaluate the scores for the leaf nodes based on the evaluation function.
- Backtrack from the leaf to the root nodes:
  - For maximizer, choose the node with maximum score
  - For minimizer, choose the node with the minimum score
- At the root node, choose the node with maximum value and select the respective move.

Diagram


A
B
C
D
E
F
G
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
4
-3
4
6
-3
7
Maximizer
Minimizer
Maximizer
Terminal node

Diagram


A
B
4
C
-3
4
D
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node


A
4
B
4
C
-3
4
D
E
6
F
-3
G
7
H
I
J
K
L
M
N
O
-1
4
2
6
-3
-5
0
7
Terminal values
Maximizer
Minimizer
Maximizer
Terminal node

Properties

- **Time Complexity:** As it performs DFS for the game tree, so the time complexity is **O(b^m)**
- **Space Complexity: O(bm)**
- **Optimal:** optimal if both opponents are playing optimally
- **Completeness:** Definitely find the solution in the finite search tree.

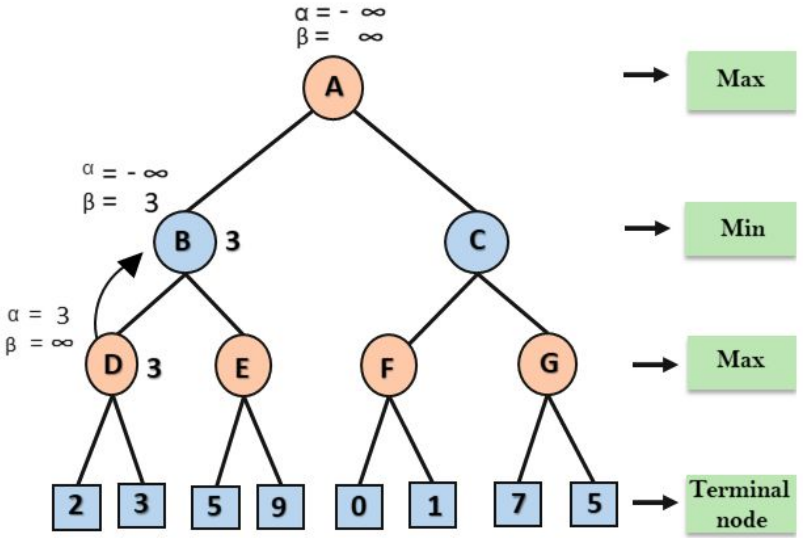
Limitation of MiniMax Algorithm

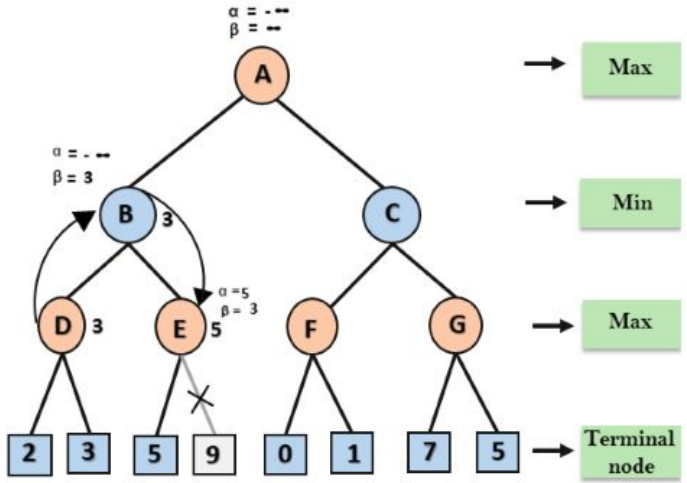
- Slow for complex games: which has huge branching factor and the players has a lots of choices to decide.

Alpha Beta Pruning

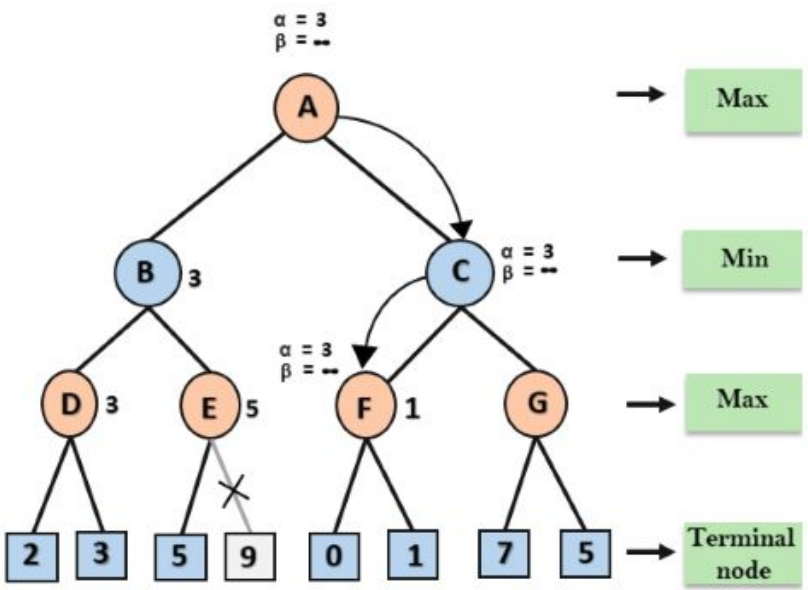
- Modified minimax algorithm
- As we have seen in the minimax search algorithm that the number of game states it has to examine are exponential in depth of the tree. Since we cannot eliminate the exponent, but we can cut it to half. Hence there is a technique by which without checking each node of the game tree we can compute the correct minimax decision and this technique is called pruning .
- This involves two parameter alpha and beta for future expansion
- It can be applied at any depth of the tree.
- Two parameter:
    - Alpha: Best(highest value) choice we found so far at any point along the path of maximizer. Initially, -∞
    - Beta: Best(lowest-value) choice we found so fat at the path of minimizer. The initial value of beta is +∞

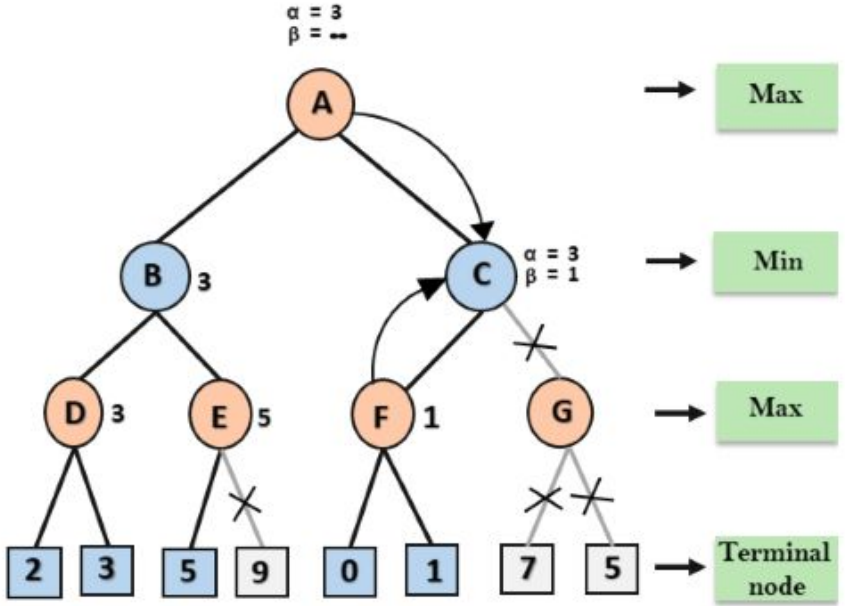
Diagram


α = - ∞
β = ∞
A
Max
α = - ∞
β = 3
B
3
C
Min
α = 3
β = ∞
D
3
E
F
G
Max
2
3
5
9
0
1
7
5
Terminal node

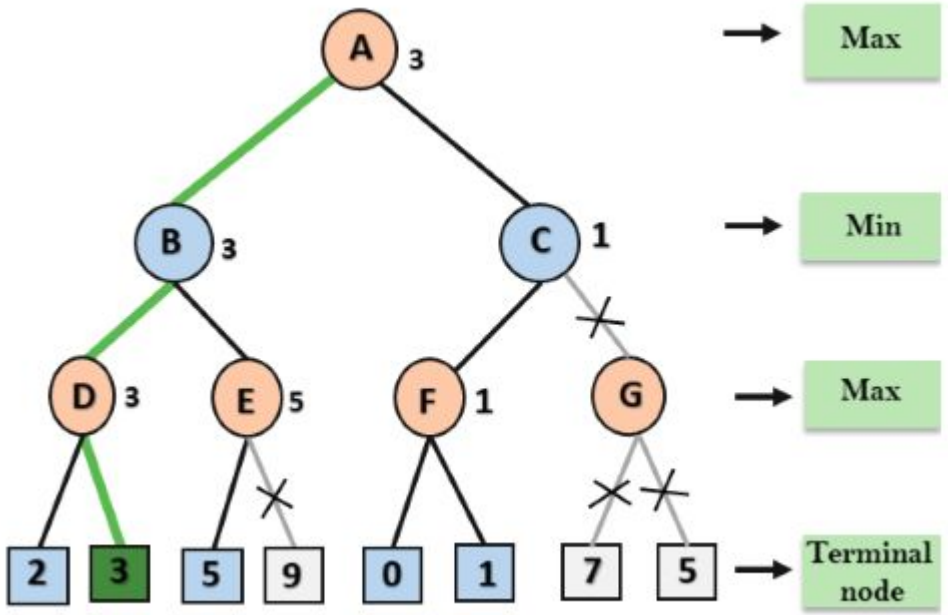

α = - ∞
β = ∞
A
Max
α = - ∞
β = 3
B
3
C
Min
α =5
β = 3
D
3
E
5
F
G
Max
2
3
5
9
0
1
7
5
Terminal node

Diagram


α = 3
β = ∞
A
Max
B
3
C
α = 3
β = ∞
Min
α = 3
β = ∞
D
3
E
5
F
1
G
Max
2
3
5
9
0
1
7
5
Terminal node


α = 3
β = ∞
A
Max
B
3
C
α = 3
β = 1
Min
D
3
E
5
F
1
G
Max
2
3
5
9
0
1
7
5
Terminal node

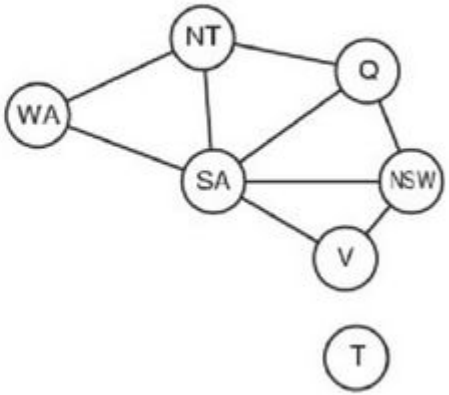
Diagram



Constraint Satisfaction Problem

- A constraint satisfaction problem consists of three components X, D and C:
  - X is a set of Variables {X1,.....,Xn}
  - D is a set of Domains, {D1,....,Dn}, one for each variable.
  - C is a set of constraints that specify allowable combinations of values.
- Example Problem: Map Coloring
  - Problem: Coloring a Map of Australia each of the states with red, green or blue in such a way that no two neighboring regions have the same color.
  - Formulation:
    - To formulate this as a CSP, we define the variables to be the regions:
      - X = {WA, NT, Q, NSW, V, SA, T}

Constraint Satisfaction Problem



Reference:

- For Detail Study of Uninformed algorithm and how it works looks at this **link**
- Book , Artificial-Intelligence-A-Modern-Approach-4th-Edition