Unit-3 : Problem Solving by Searching

-Ankit Pangeni

DATE

Searching:

→ Searching is a commonly used method in AI for solving problems. Problem solving in AI is a systematic search through a range of possible action in order to reach some predefined goal or solution.

→ The solutions or 'goal states' could sometimes be an object, a goal, a sub goal or a path to the searched item.

Problem solving agent:-

→ It decides what to do by finding sequence of actions that leads to the goal. This is a kind of goal based agent which always tries to achieve goal.

→ In many situations, "to solve a problem" can be described as to change the current situation, step by step, from an initial state to a final state.

→ If each state is represented by a node, & each possible change is represented by a link, then a "problem" can be represented as a graph (the "state space"), with a "solution" corresponding to a path from the initial state to a final state.

* Four general steps in problem solving

PAGE

DATE

1. Goal formulation : A goal is a state that the agent is trying to reach. This step formulates the goal based on current situation & the agent's performance.

2. Problem formulations : This is the process of deciding what actions & states to consider, given goal.

3. Search method : It determine the possible sequence of actions that lead to the states of known values and then choosing the best sequence.

4. Execute : Once the solution is found, the actions it recommends can be carried out. This is called the execution phase.

Problem formulation

Before finding the algorithm for evaluating the problem and searching for the solution, we first need to define and formulate the problem. A problem can be defined formally by 4 components.

1. Initial state : It is the state from which our agents start solving the problem.
2. State description : It is the description of the possible actions available to the agent. It is common to describe it by means of a successor function. The initial state & the successor function together defined is called state space which is the set of all possible states reachable from the initial state.

DATE

3. Goal Test: We should be able to decide whether the current state is a goal state.

4. Path cost: A function that assigns a numeric value to each path, each step we take in solving the problem should be somehow weighted.

→ A solution to a problem is the path from initial state to a goal state & the quality is measured by the path cost, and the optimal solution has the lowest path cost among all possible solutions.

Ex:

7	2	4
5		6
8	3	1

Start State

	1	2
3	4	5
6	7	8

8-puzzle.

* Initial state:- Our board can be in any state resulting from making it on any configuration

* State description:
  - Successor function generates legal states resulting from applying the three actions { move blank UP, Down, Left, or Right }
  - State description specifies the location of each of the eight titles & the bank.
* Goal test: Checks whether the state matches the goal configured in the goal state shown in the picture
* Path cost: Each step costs 1, so the path cost is the sum of steps in the path.

CLASSMATE PAGE

DATE

State Space Representation

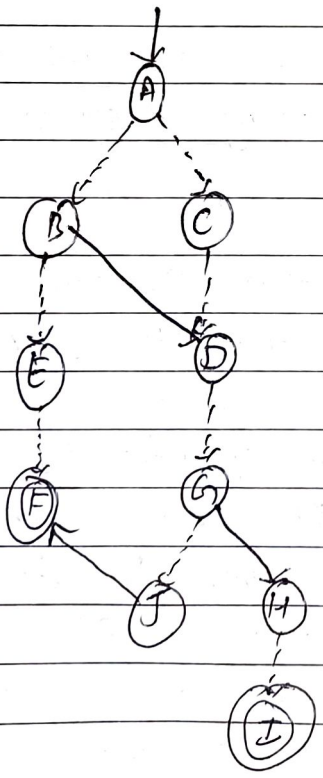
→ State space is the set of all states reachable from the initial state. This is defined implicitly by initial state and successor function (Actions) together.

→ The state space is commonly represented as a directed graph in which each node is a state and each arc represents the application of an operator transforming a state to a successor state.

→ A solution is a path from the initial state to a goal state

→ State space searching assumes that
- the agent has perfect knowledge of state space & can observe what state it is in
- the agent has a set of actions that have known deterministic effects
- the agent can recognize the goal state.

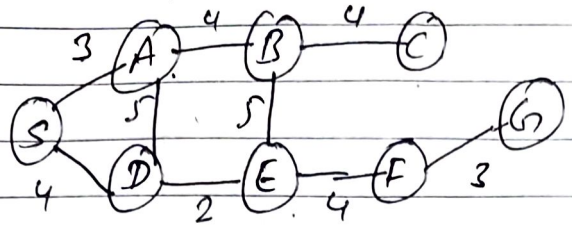
Ex:



States: $S = \{A, B, C, D, E, F, G, H, I, J\}$
Initial: $S = A$
Goal state: $G = \{F, I\}$

DATE

Example of problem formulation: Route finding problem



Problem formulation may be as follows:

- States: A, B, C, D, S, E, F, G
- Initial state: S
- Successor function (operators): Move from one location to another.
- Goal test: G
- Path cost: Total distance, money, time, (Here, path cost is in km. A to B is 4km)

Well defined Problem

→ In problem solving, any problem in which the initial state / starting position, the allowable operations, goal state, and a unique solution can be shown to exist is called well defined problem.

→ Ex: Tower of Hanoi, waterjar problem etc.

→ If any problem lacks on or more of these specified properties, then it is an ill-defined problem, and most problems that are encountered in everyday life fall into this category.

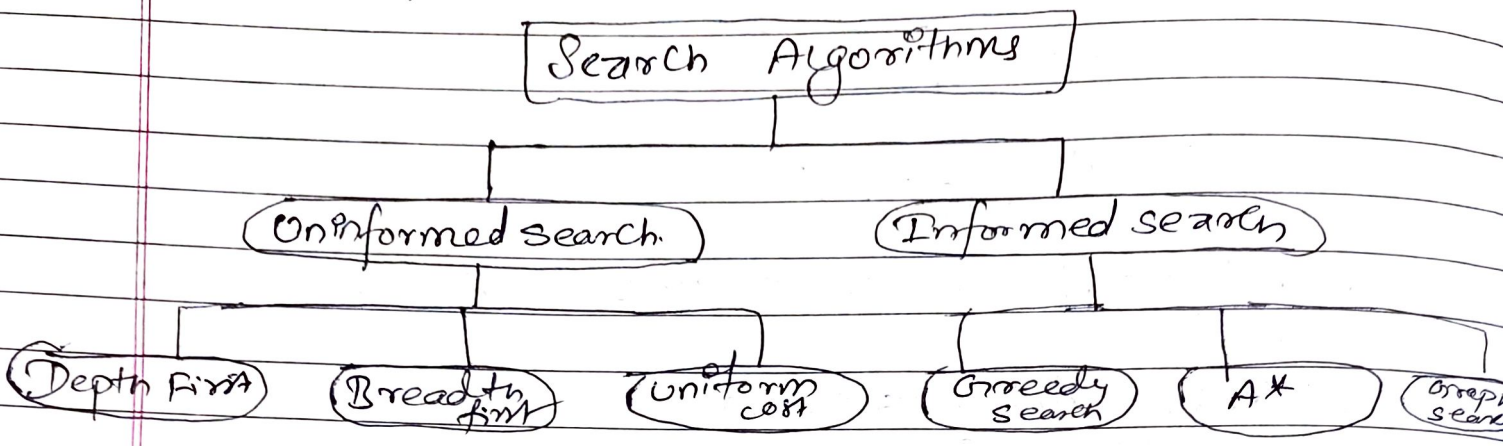
→ A well defined problem can be described by:

- Initial state: ---
- State description: ---
- Goal test: ---
- Path cost: ---
- State space: ...
- operator or successor function: for any state (x) returns S(x), the set of states reachable from x.

classmate PAGE

DATE

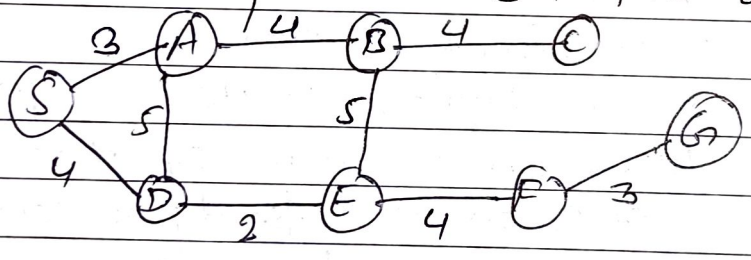
Types of search Algorithms



Solving Problems by Searching.

Let's take an example of a map of a place. The nodes represent cities, and the links represent direct road connection between cities. The number associated with the link represents the length of the corresponding road.

The search problem is to find a path from a city S to G.



This problem will be used to illustrate some search methods. Some real world applications are:

- VLSI layout
- Web search
- Path Planning
- Robot Navigation.

classmate

PAGE

DATE

Search strategies

Searching is a process to find the solution for a given set of problems. There are two classes of search methods.

1. Uninformed/blind/bruteforce search methods
2. Heuristically informed search methods

* Uninformed search method

→ In uninformed search method, the order in which the paths are considered is arbitary
→ There is no domain specific information to judge where the solution is likely to lie.
→ Examples:- Depth first search, Iterative deepening search, Breadth first search, Uniform cost search and Bidirectional search.

* Informed search method

→ In case of heuristically informed search method, one uses domain-dependant (heuristic) information to judge where the solution is likely to lie.
→ It is more efficient method of searching.
→ Example: Greedy Best first search, A* search, Hill climbing search, stimulated Annealing search, etc.

DATE

* Difference between uninformed & informed search.

Informed search	Uninformed search.
i) It uses domain knowledge for the searching process	i) It doesn't use any knowledge for searching process
ii) It finds solution more quickly	ii) It finds solution slow as compared to informed search.
iii) It may or may not be complete.	iii) It is always complete.
iv) Cost is low	iv) Cost is high.
v) It consumes less time.	v) It consumes moderate time.
vi) It provides the direction regarding the solution.	vi) No suggestions is given regarding the solution in it.
vii) It is less lengthy while implementation.	vii) It is more lengthy while implementation
viii) Ex: Greedy search, A* search, Graph search	viii) Ex: Depth first search, Breadth first search

classmate

PAGE

DATE

Performance evaluation of search techniques

There are four ways to evaluate the performance of search techniques:

- **Time Complexity:** How long (worst or average case) does it take to find a solution? It is usually measured in terms of the number of nodes expanded. Ex: $O(b^d)$. (represented in terms of big Oh.)
- **Space Complexity:** How much space is used by the algorithm? It is usually measured in terms of the maximum number of nodes in the memory at a time. Ex: $O(b^d)$ or $O(b^m)$
  where, b = maximum branching factors. (ie. max no. of successor of any node)
  d = maximum depth of any path
  m = depth of the least-cost solution
- **Completeness:** An algorithm is said to be complete if it definitely finds solution to the problem if exists.
- **Optimality/Admissibility:** If a solution is found, is it guaranteed to be an optimal one? Is it the path with the minimum cost than all other possible paths of the solution?

DATE

Uninformed Search.

* Breadth first Search
* Depth first Search
* Depth limited Search
* Iterative Deepening search
* Uniform cost Search
* Bidirectional search.

1) Breadth first Search

→ It is an algorithm for traversing or searching tree or graph data structures

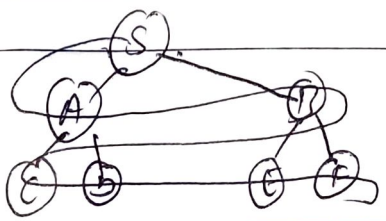
→ It starts from the root node or sometimes referred to as a 'search key' and explores all the neighbor nodes at the present depth prior to moving onto the nodes at the next depth level.

→ It performs searching level wise. It means all the nodes at a given level are explored before any nodes at the next level are expanded. until the goal reached. It uses FIFO Queue data structure.

→ As the name itself suggests, you are required to traverse the graph breadth wise as follows:

- 1st move horizontally & visit all nodes of the current layer.
- Move to the next layer.

Note: Don't generate as child node if the node is already parent to avoid more loop.



classmate

PAGE

DATE

Performance measures:

* Complexity of Time (Time complexity)

Its time complexity is $O(b^d)$

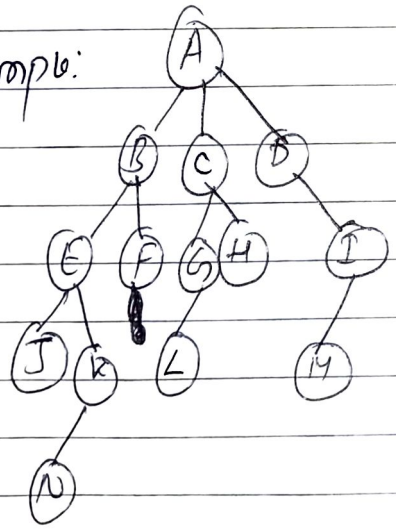
where, d = maximum depth of the path
b = maximum branching factors

* Space complexity:
Each node that is generated must remain in the memory. So, its space complexity is also $O(b^d)$

* Completeness
It always finds the solution if exists. If the shallowest goal node is at some finite depth d and if b is finite, it is always complete.

* Optimality:
The solution given by BFS is definitely optimal if all paths have the same cost. Otherwise, not optimal but it finds solution with shortest path length.

Example:



Here, path is (ie. traversing order)
A, B, C, D, E, F, G, H, I, J, K

Let's take a queue
Start node A, search node K

A [A]
B [B C D]
C [C D E F]
D [D E F G H]
E [E F G H I]
F [F G H I J K]
G [G H I J K]
H [H I J K L]
I [I J K L]
J [J K L M]
K [K L M]

PAGE

DATE

Two lessons:

- Memory requirements are a bigger problem than its execution time.
- Exponential complexity can't be computed in uninformed search.

Applications:

- Social networking websites use BFS
- In GPS navigation systems
- Broadcasting in network
- Crawler in Search Engines.

DATE

2) Depth first search.

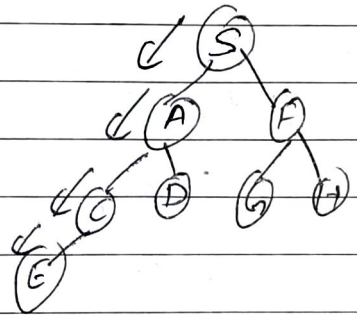
→ It is another algorithm for traversing or searching tree or graph data structures.

→ It starts at the root node (selecting some arbitrary node as the root node in case of graph) and explores as far as possible along each branch before backtracking.

→ The main strategy of DFS is to explore deeper into the graph whenever possible.

→ When all edges have been explored, the search backtracks until it reaches an unexplored node. This process continues until all nodes/vertices that are reachable from start node is discovered.

→ It uses LIFO queue (stack) data structure.



Performance Measures

* Time complexity:

If m = maximum depth of search tree — in worst case. solution may exist at depth m

b = branching factors. (ie. root has b successors, each node at next level has again b successors (b^2)

Then,

Total nodes = $b + b^2 + b^3 + \dots b^m = O(b^m)$

∴ Time complexity = $O(b^m)$

DATE

* 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:

$1 + b + b + b + \dots - b$ m times : $O(bm)$

* Completeness.

It doesn't find a solution always. If search space is infinite & search space contains loops then DFS may not find solution. So, it is not complete.

* Optimality.

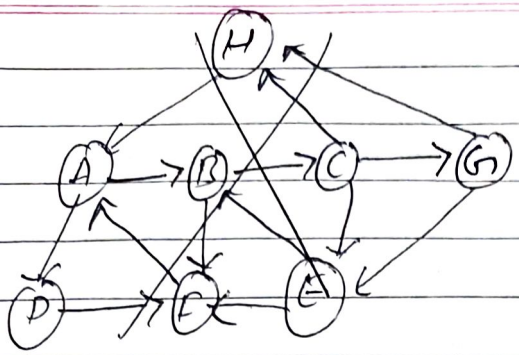
It expands deepest node first, suppose, our goal node is at right sub-tree at levels 2 or 3. but if it expands entire left subtree first, then we can say it can't give shortest path. Thus, it may not always give optimal solution. (Note: we are talking about worst case)

* Applications

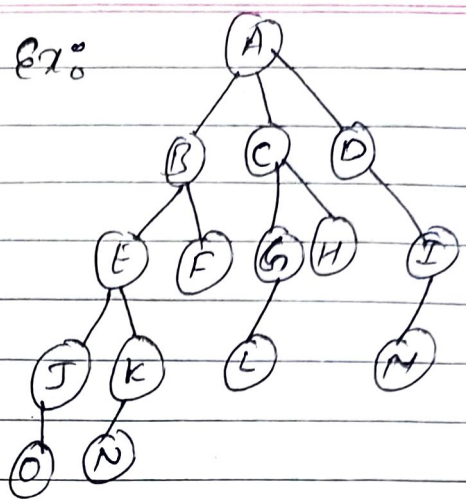
- Detecting cycle in a graph
- Path finding
- Topological Sorting
- Solving puzzles with only one solution, such as mazes.

DATE

Ex:

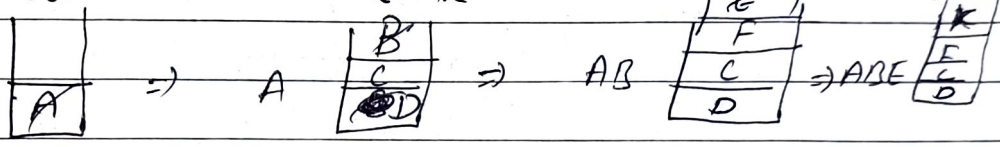


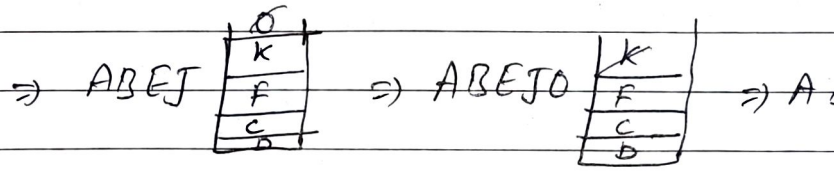
Ex:



Let's take a stack..

Start Node A, search node K.





So, paths A, B, E, K (Note here we write only its corresponding parent nodes:
K's parent is E & E's is B & B's is A)

DATE

3) Depth limited search

→ It is a modification of depth-first search
→ Working is similar to DFS but with a predetermined limit of the depth.
→ The main problem of DFS, which is the infinite path, is solved by Depth limited search
→ Here, like the normal depth first search, it is an uninformed search
→ Even the search could expand a vertex beyond that predetermined depth limit, it will not do so and thereby it will not follow infinitely depth paths and get stuck in cycles.
→ Therefore, depth-limited search will find a solution if it is within the depth limit, which guarantees at least completeness on all graphs.
→ Terminates when: a) failure value: no soln & b) cutoff value: no soln within depth limit

Advantages
i) It is memory efficient and space efficient than DFS
ii) Assures that it will find the solution if it is inside the depth limit.

Disadvantages:
i) Can be terminated without finding solution, if solution is outside the depth limit.
ii) It is not optimal

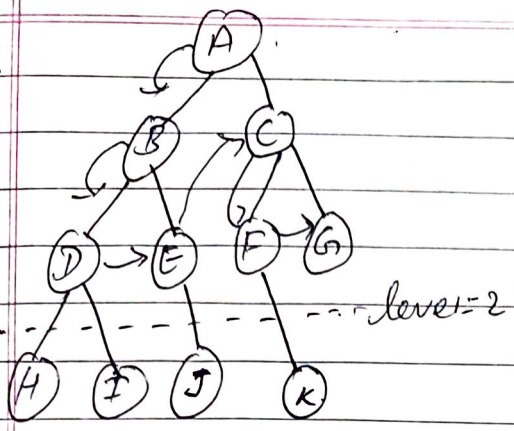
Applications:
i) Finding shortest path.
ii) Geographical maps

classmate

PAGE

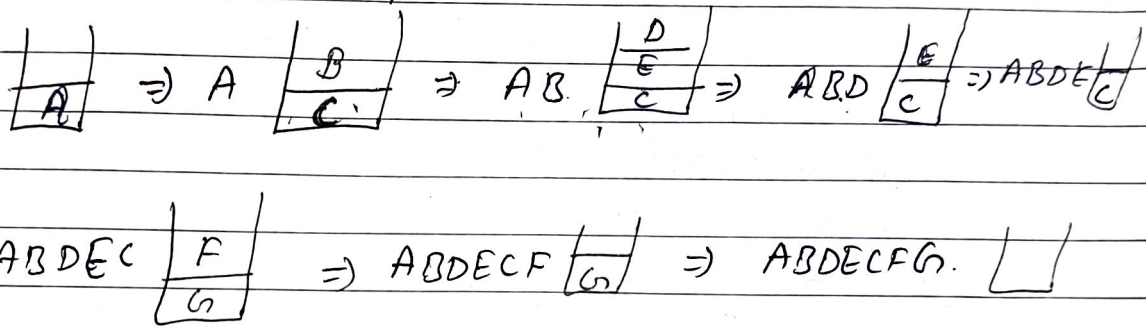
DATE

Ex:



Find DLS of following graph by using A as source & G as destination node using depth limit = 2.

Let's take a stack.



Hence, the path is A → B → D → E → C → F → G

* Performance Measure:

→ Time Complexity: $O(b^l)$

where, b = branching factor
l = depth limit.

→ Space complexity: $O(bl)$

→ Completeness:- It is not complete when $l < d$, in which is our DLS will never reach to goal.

→ Optimality:- It is not optimal even if $l > d$.

DATE

4) Iterative Deepening (Depth-first) Search

→ It is a combination of BFS and DFS, thus combining the advantages of each strategy, taking the completeness and optimality of BFS and modest memory requirement of DFS.

→ Here, depth-limited search is run repeadedly, increasing the depth limit with each iteration until it reaches d, the depth of the shallowest goal state.

→ On each iteration, IDDFS visits the node in same order as DFS, but cumulative order in which nodes are visited is effectively BFS.

→ Initially, IDDFS starts with the depth 0, and in every iteration it increases the depth limit by 1 if it doesn't find the goal node.

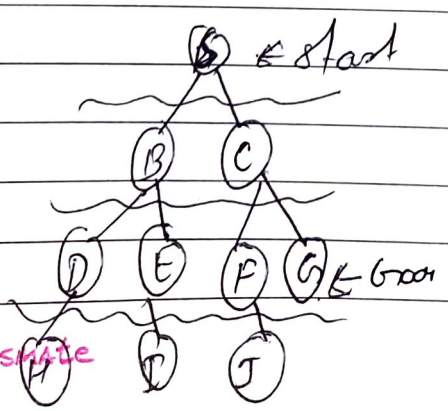
Advantages:

i) Combines both BFS & DFS strategy, so more efficient
ii) Uses much less memory

Disadvantages:

i) Time taken is exponential to reach the goal node
ii) The IDDFS might fail when the BFS fails

Ex:



PAGE

DATE

1st Iteration, depth (d) = 0 [S]
2nd Iteration, d = 0+1 = 1 [S → B → C]
3rd Iteration, d = 2 [S → B → D → E → F → G]

Hence, the path is: S → B → D → E → F → G.

* Performance measure:

→ Time Complexity:
- Node generation
  - Level d : once
  - level d-1 = 2
  - level d-2 = 3
  - ---
  - level 2 = d-1
  - level 1 = d

∴ Total nodes generated = $d \cdot b + (d-1)b^2 + (d-2) \cdot b^3 + \cdots + 1 \cdot b^d$
$= O(b^d)$

→ Space Complexity: $O(bd)$

→ Completeness: IDDFS is like BFS, so it is complete when the branching factor b is finite.

→ Optimality: IDDFS is also like BFS, so it is optimal when steps are of same cost.

PAGE

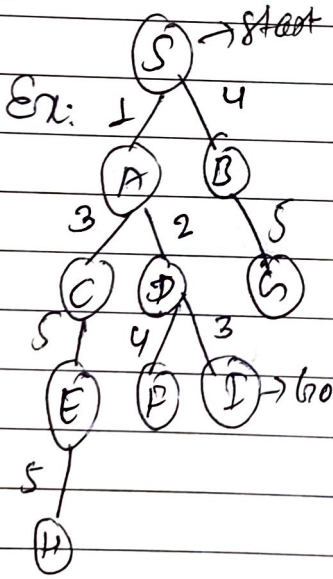
DATE

5) Uniform Cost Search

→ In the case of uniform cost search, we use weighted tree or graph for traversing.
→ It comes into play when a different cost is available for each edge.
→ The goal of ucs is to find a path to the goal node with the lowest cumulative cost.
→ It expands nodes according to their path costs from the root node.
→ It is implemented by the priority queue (lowest cost given highest priority)
→ It uses backtracking.

Advantages
→ Mostly gives optimal solution

Disadvantages:
→ May be stuck in an infinite loop.



Ex:

Path.

S → A → D → G.

Performance measure:

Completeness:- It is complete
Optimal:- Always optimal as it only selects path with lowest cost

Time Complexity: $O(b^{1+\lceil C^*/\varepsilon \rceil})$

Space: $O(b^{1+\lceil C^*/\varepsilon \rceil})$.

where
C^* = cost of optimal sol^n
ε = step to get closer to goal node

PAGE

DATE

6) Bidirectional cost.

→ As the name itself suggests, bidirectional search suggests to run two simultaneous search, one is forward search which is from the start/initial state to goal state and the other is backward search which is from the goal state to the initial state.

→ It replaces a single search graph with two smaller sub graphs

→ The search is terminated when both search meet at an intersection node.

→ Any search technique (BFS, DFS --) can be used for forward and backward search

Advantages

i) It is faster as it drastically reduces the amount of required exploration

ii) Also saves resources for users as it requires less memory capacity to store all the searches.

Disadvantages

i) Implementation is difficult

ii) Goal state should be known in advance.

Note: You can use BFS at forward & DFS at backward too.

Performance measures:

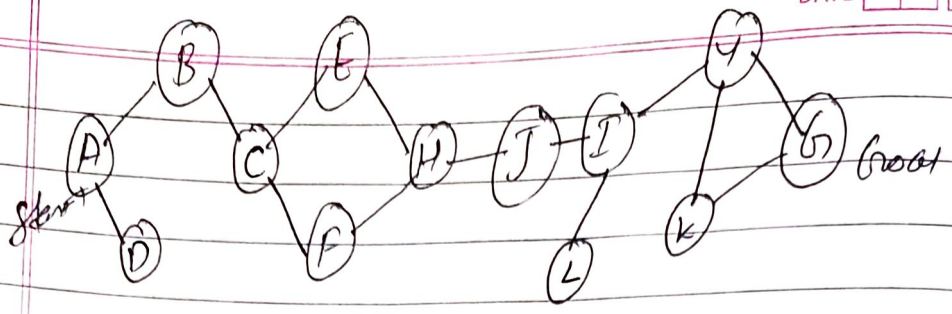
- Time Complexity: $O(b^{d/2})$
- Space Complexity: $O(b^{d/2})$
- Completeness: Complete when we use BFS in both searches
- Optimality: Optimal when used BFS & path are of same cost

Using other search may sacrifice completeness & optimality

CLASSMATE PAGE

DATE

Ex:



Let's use BFS in both forward & backward search

Let's setup queue in both sides

Forward	Backward
A	G
A [B D]	G [K Y]
AB [D C]	GK [Y]
ABD [C]	GKY [I]
ABDC [E F]	GKYI [L J]
ABCDE [F H]	GKYIL [J]
ABCDEF [H]	GKYILJ [I]
ABCDEFH [I]	GKYILJI [H]
ABCDEFHI	GKYILJIH

Meeting path is: A → B → D → C → E → F → H → J → L → I → Y → K → G

DATE

Drawbacks of uninformed search:-

i) Criterion to choose next node to expand depends only on a global criterion: level.
ii) Doesn't exploit the structure of the problem.
iii) Very often, we can select which rule to apply by comparing the current state & the desired state.

Informed (Heuristic) Search

Informed search

→ In informed search, we have information about the paths to the goal state which helps in more efficient searching.
→ It uses domain specific knowledge for searching
→ This information is obtained by a function called heuristic function that estimates how close a state is to the goal state. That's why it is called heuristic search

Heuristic function.

→ Heuristic function h(n) estimates the goodness of a node n. It estimates how close a state is to a goal state.
→ Using heuristic function, it might not always find the best solution but it is guaranteed to find the good soon
→ Heuristic function is an estimate based on domain specific information.

CLASSMATE PAGE

DATE

Why use heuristic searches?
- It may be too resource intensive (both time & space) to use a blind search
- It is more efficient

→ Types of Informed Search:

Best-first Search ┬ Greedy best first search
                  └ A* Search

Best-First Search

→ BFS and DFS blindly explore paths without considering any cost function.

→ The idea of BFS is to use an evaluation function to decide which adjacent is more promising and explore.

→ This "best first" behaviour is implemented with a priority queue.

→ Here we use an evaluation function $f(n)$ that gives an indication of which node to expand next for each node.
- Usually gives an estimate to the goal
- Node with the lowest value is expanded first

→ A key component of evaluation function $f(n)$ is a heuristic function, $h(n)$, which is the additional knowledge of the problem.

DATE

Special cases: based on evaluation function.
- Greedy best-first search
- A* Search.

1. Greedy best-first search.

→ The best-first search part of the name means that it uses an evaluation function to select which node is to be expanded next.

→ The node with the smallest/lowest evaluation (i.e. with lowest heuristic value) is selected for expansion because that is the best node, since it supposedly has the closest path to the goal (if the heuristic is good)

→ Greedy best first search only uses the heuristic, and not any link costs.

→ It uses both the combination of BFS and DFS

→ If the heuristic is not accurate, it can go down paths with high link cost.

Evaluation function $f(n) = h(n)$ = estimate of cost from n to goal.

Ex of heuristic: $h_{SLD}(n)$ = straight-line distance from n to goal.

→ It expands the node that appears to be closest to goal.

Advantages:
i) Can switch between BFS & DFS by gaining advantages of both.
ii) This algorithm is more efficient than BFS & DFS

Disadvantage.
i) It can get stuck in loops.
ii) Not optimal.

classmate

PAGE

Worst case: If heuristic function is bad, it acts as uninformed.

DATE

Performance measures:

→ Time Complexity: $O(b^d)$ or $O(b^m)$
where, b = average branching factor
d = maximum depth of the tree.

→ Space Complexity: $O(b^d)$

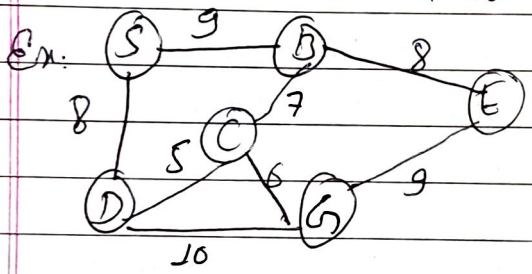
→ Completeness: It is not complete

→ Optimality: It is optimal when $h(n) \leq h^*(n)$
Otherwise it is not optimal
where $h(n)$ = heuristic cost
$h^*(n)$ = actual cost

Note: $h(n) < h^*(n) \Rightarrow$ underestimates
$h(n) > h^*(n) \Rightarrow$ overestimates

For proper optimality, we use A* algorithm but it always gives good solution.

Ex:



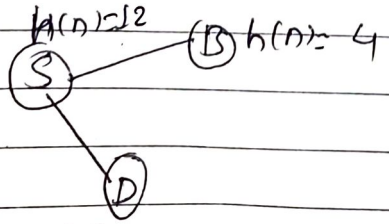
S = Source (start)
G = Goal

Straight line distance to node G (goal node) from other nodes is

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

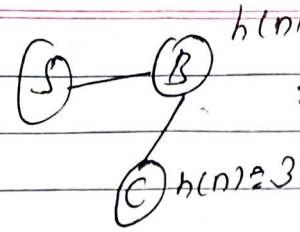
Let, $h(n)$ = Straight line distance

Step-1:

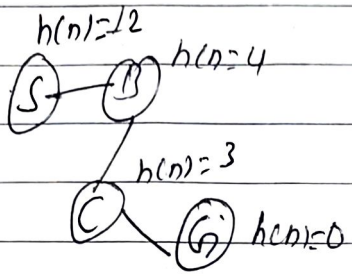


DATE

Step-2:-

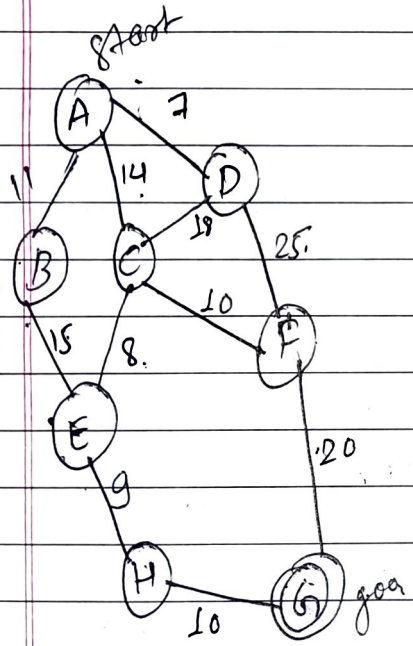


Step-3:



Hence path: S → B → C → G

Eg:



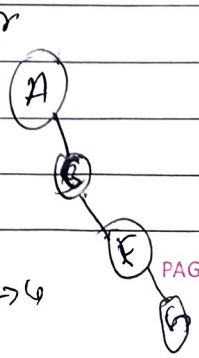
Straight Line distance (heuristic values)

A → G = 40
B → G = 32
C → G = 25
D → G = 35
E → G = 19
F → G = 17
H → G = 10
G → G = 0

⇒

Open	Closed (Visited)
A	
C B D	A
B D	A C
F E B D	A C
G E B D	A C F
G E B D	A C F

Or

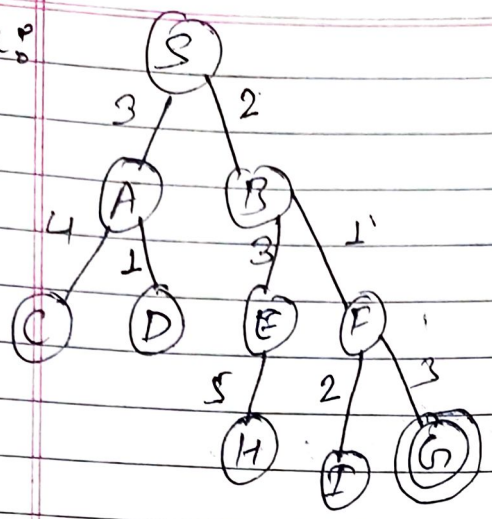


path: A → C → F → G

PAGE

DATE

Ex:



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

Open

S

B | A

F | E | A

~~H~~ | E | A

G | E | A

Close

S

S | B

S | B | F

S | B | F | E

Talk about optimality

Open

S

B | A

F | E | A

G | I | E | A

I | E | A

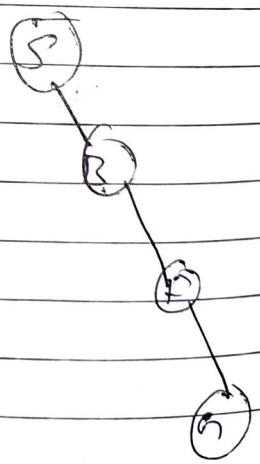
Close

S

S | B

S | B | F

S | B | F | G



Path: S → B → F → G

classmate

PAGE

DATE

2. A* Search:

First Let's talk about:

Admissible Heuristics

→ A heuristic function is said to be admissible if it is no more than the lowest-cost path to the goal. In other words, a heuristic is admissible if it never overestimates the cost of reaching the goal. It is also known as optimistic heuristic

→ It is used to estimate the cost of reaching the goal node in an informed search algorithm.

→ In order for a heuristic to be admissible to the search problem, the estimated cost must always be lower than the actual cost of reaching the goal state.

→ Ex; in A* Search the evaluation function is:

$$f(n) = g(n) + h(n)$$

Where, $f(n)$ = the evaluation function.

$g(n)$ = Cost from start node to current node.

$h(n)$ = estimated cost from current node to goal.

→ $h(n)$ is calculated using the heuristic function.

→ With non-admissible heuristic, the A* algorithm could overlook the optimal solution

DATE

2. A* Search

→ A* is a best first, informed graph search algorithm
→ It is different from other best first search algorithms because it uses a heuristic function h(x) as well as the path cost to the node g(x), in computing the cost f(x) = h(x) + g(x) for the node.
→ The h(x) part of the f(x) must be an admissible heuristic ie. it must not overestimate the distance to the goal.

→ It finds a minimal cost-path joining the start node and a goal node for node n.
Evaluation function : f(n) = g(n) + h(n)
Where,
g(n) = cost so far to reach n from root
h(n) = estimated cost to goal from n.

→ As it traverses the graph, it follows a path of lowest known path keeping a sorted priority queue of alternate path segments along the way.
→ If, at any point, a segment of the path being traversed has a higher cost than another encountered path segment, it abandons the higher cost path segment & traverses the lower-cost path segment instead.
→ This process continues until the goal is reached.

Advantages: i) It is optimal search algorithm in terms of heuristic
ii) It is used to solve complex search problems

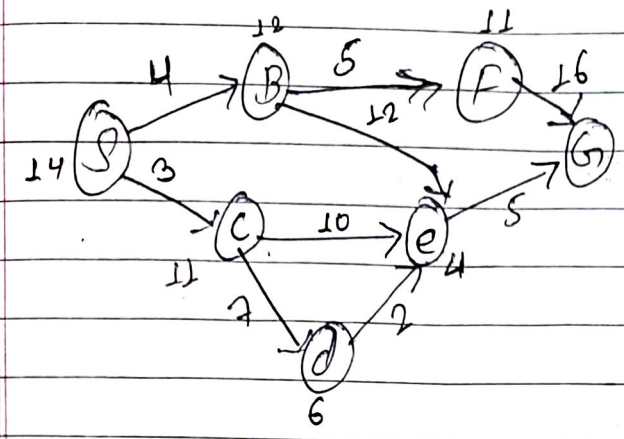
Disadvantages
i) It is complete if the branching factor is finite & every action has fixed cost.

PAGE

DATE

ii) Its performance is bad if heuristic algorithm is bad.

Ex:



We know,

$f(n) = h(n) + g(n)$

↓ Estimation cost ↓ Actual cost

Let's start

$f(S) = 0 + 14 = 14$.

$S \to B$	$S \to C$
$= 4 + 12$	$= 3 + 11$
$= 16$	$= 14$.

Since, $14 < 11$, so we move towards $S \to C$.

Now,

$SC \to e$	$SC \to d$
$= 13 + 4$	$= 10 + 6$
$= 17$	$= 16$

Since, $16 < 17$, so we move towards SCd.

Here, since, $S \to B$ cost is also 16. So, we can check path S→B in parallel.

$SB \to F$	$SB \to e$
$= 4 + 5 + 11$	$= 4 + 12 + 4$
$= 20$	$= 20$

Now,

$scd \to e$

$= 3 + 7 + 2 + 4$

$= 16$

Since, $16 < 20$, so we choose path scde

$scde \to G$

$= 17 + 0$

$= 17$ which is our answer

So, path is $S \to C \to d \to e \to G$

classmate

PAGE

DATE

Performance measures:

→ Time Complexity: $O(b^d)$ Where, d = length/depth of goal node from start node
→ Space Complexity: $O(b^d)$:- It keeps all generated nodes in memory
→ Completeness: Yes A* search always gives solution
→ Optimality: Gives optimal solution when the heuristic function is admissible heuristic

Admissibility and Optimality

→ A* is admissible and considers fewer nodes than any other admissible search algorithm with the same heuristic.

→ It is admissible if, for any graph, it always terminates in an optimal path (if exists), from initial state to goal state.
→ Thus, A* search algorithm is said to be admissible if it is guaranteed to return an optimal solution.
→ A heuristic "h(n)" is admissible, if for every node n,

$$h(n) \leq h^*(n)$$

Where, $h^*(n)$ = True cost to reach the goal state from n.
ie. it underestimates the cost to reach the goal.
→ An admissible heuristic never overestimates the cost to reach the goal. So, it is optimistic.

DATE

3) Hill Climbing Search

→ This Search technique can be used to solve problems that have many solutions, some of which are better than others

→ It starts with a random solution, and iteratively makes small changes to the solution, each time improving a little. When the algorithm cannot see any improvements anymore, it terminates

→ Ideally, at that point the current solution is close to optimal, but is not guaranteed that hill climbing will ever close to the optimal solution.

The hill climbing can be described as follows:

- Evaluate the initial state, if it is goal state then Terminate. Otherwise, current state is final state.
- Select the new operator for this state & generate a new state.
- Evaluate the new state
  - * If it is closer to goal, make it current state
  - * If it is no better, ignore.
- If the current state is goal state or no new operators available, Terminate.
  Otherwise repeat from 2.

→ It is basically depth-first search with a heuristic measurement that orders choices as nodes are expanded.

→ It always selects the most promising successor of the node last expanded

DATE

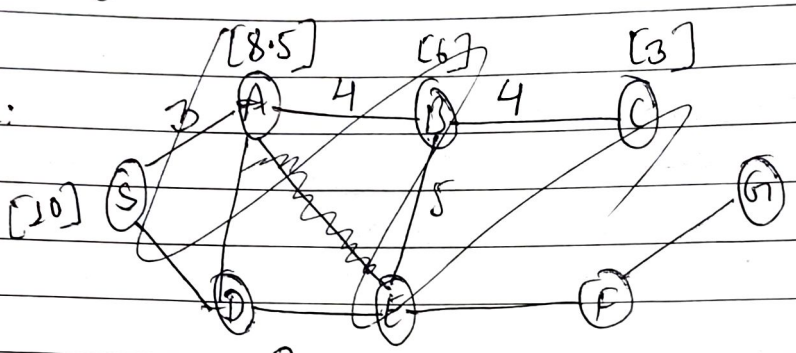
Problems

i) Gets stuck at local maxima. It is not guaranteed that we have found the best solution

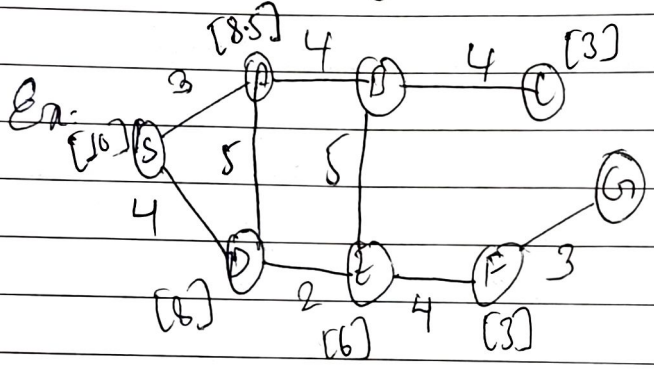
ii) Ridge is a sequence of local maxima.

iii) We may find a plateau. (It is an area so flat that all neighbours return the same evaluation)

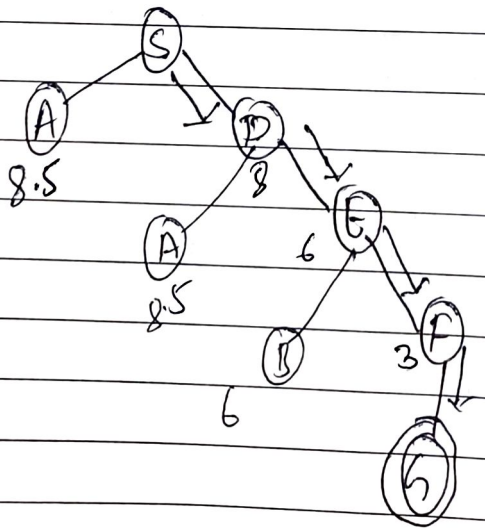
Ex:



Ex:



S: source, G: goal



classmate

PAGE

DATE

4) Simulated Annealing Search

→ This search is motivated by a physical annealing process in which material is heated and slowly cooled into a uniform structure.

→ Compared to hill climbing the main difference is that SA allows downwards steps.

→ It also differs from hill climbing in that a move is selected at random & then decides whether to accept it.

→ If the move is better than its current position then SA will always take it. If the move is worse then it will be accepted based on some probability.

→ The probability of accepting a worse state is:

$$P = \text{exponential}(-c/t) > r$$

Where.

c = the change in evaluation function

t = the current value.

r = a random number between 0 and 1.

→ The most common way to implement an SA algorithm is to implement hill climbing with an accept function and modify it for SA

DATE

* Game playing.

- Game playing is an important domain of AI.
- Games don't require much knowledge; The only knowledge we need to provide is the rules, legal moves and the conditions of winning or losing the game.
- Both players try to win the game. So both of them try to make the best move possible at each turn.
- Searching techniques like BFS are not accurate for this, as the branching factor is very high, so searching will take a lot of time.
- So, we need another search procedures that improves searching time.
- Games are a form of multi-agent environment and arise questions like. What do other agents do and how they affect our success? Competitive multiagent environments give rise to games

* Adversial Search Techniques.

Adversial Search is also known as Minimax Search used in calculating the best move in two player games where all the information is available, such as Chess or tic tac toe. It consists of navigating through a tree which captures all the possible moves in the game,

DATE

where each move is represented in terms of loss & gain for one of the players

Game - adversary

→ Solution is strategy (strategy specifies move for every possible opponent reply)
→ Time limits force an approximate solution
→ Evaluation function: evaluate "goodness" of game position.
→ Examples: Chess, Checkers, backgammon.

1. The Min-Max Search.

- It is a recursive or backtracking algorithm which is used in decision-making & game theory.
- It provides an optimal move for the player assuming that opponent is also playing optimally.
- It is mostly used for game playing in AI such as Chess, Checkers, tic-tac-toe.
- Here, two players play the game, one is called MAX and other is called MIN.
- Both players are opponent to each other, where MAX will select the maximized value & MIN will select the minimized value.
- It performs a DFS algorithm where it proceeds all the way to the terminal node of the game tree, then backtrack the tree as the recursion.

DATE

Performance measures:

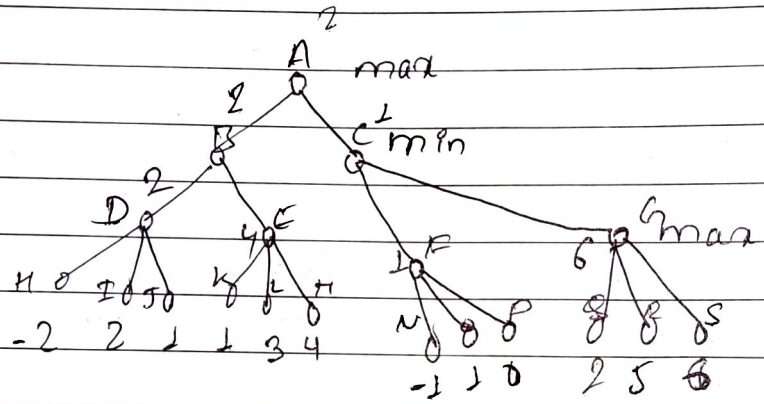
(1) It is complete
(2) It is optimal.
(3) Time & space complexity: $O(b^m)$ & $O(bm)$

b = branching factor of gametree
m = maximum depth of the

Limitations

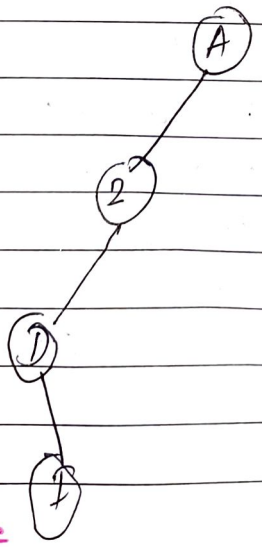
- Slow for complex Games such as chess. It can be improved from alpha-beta prunning.

Ex:



Initially, MAX = $-\infty$, MIN = ∞

Path for benefit of MAX is:

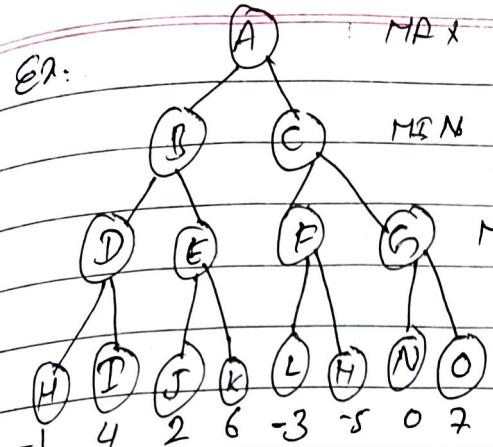


classmate

PAGE

DATE

Eg:



Initially, value of $Max = -\infty$, $MIN = \infty$] worst case.

Here, 1st turn for MAX

For node D $\max(-1, -\infty) \Rightarrow \max(-1, 4) = 4$

For NODE E, $\max(2, \infty) \Rightarrow \max(2, 6) = 6$

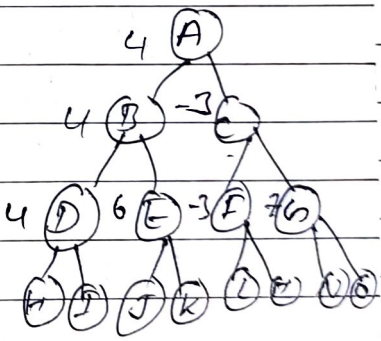
For NODE F, $\max(-\infty, -3) \Rightarrow \max(-3, -5) = -3$

For NODE G, $\max(-\infty, 0) \Rightarrow \max(0, 7) = 7$

Next step, its turn for MIN.

For NODE B, $\min(4, 6) = 4$

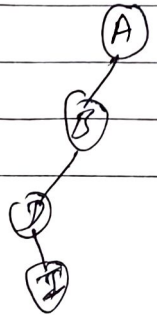
For NODE C, $\min(-3, 7) = -3$



Now, its turn for MAX,

For NODE A, $\max(B, C) = \max(4, -3) = 4$.

Hence, the best path for maximum benefit for MAX is:



classmate

PAGE

DATE

2. Alpha - Beta Pruning

- It is a modified version of the minimax algorithm. It is an optimization technique for the minimax algorithm.
- Here, without checking each node of the game tree we can compute the correct minimax decision, and this technique is called pruning.
- It involves two threshold parameter Alpha & beta for future expansion, so it is called alpha-beta pruning. It is also called as Alpha-Beta Algorithm
- It can be applied at any depth of a tree, and sometimes it not only prune the tree leaves but also entire sub-tree.

Two parameters can be defined as:

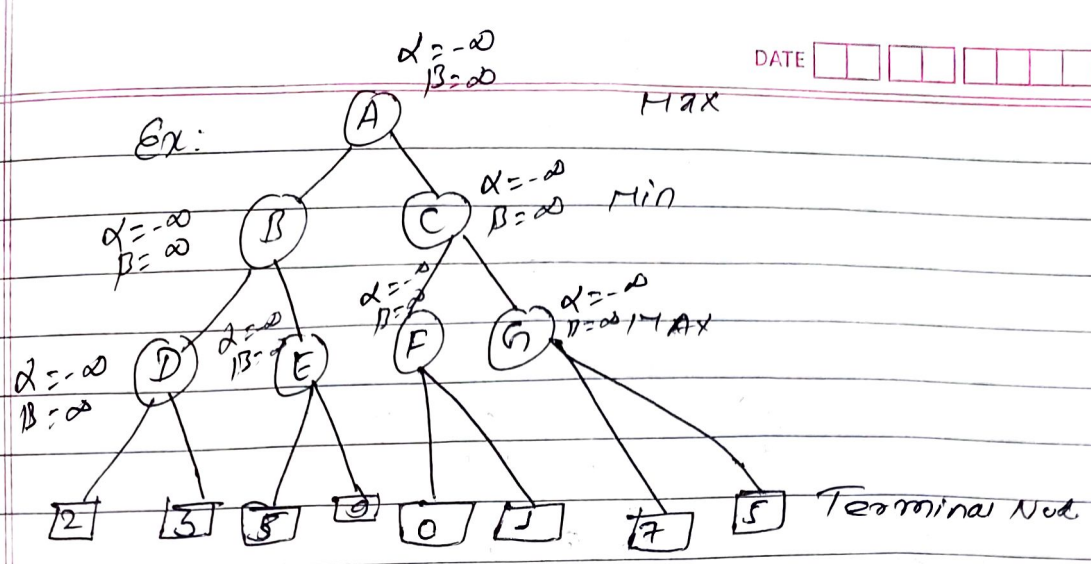
Alpha: The best (highest value) choice we have found so far at any point along the path of Maximizer. Its initial value is $-\infty$.

Beta: The best (lowest-value) choice we have found so far at any point along the path of Minimizer. Its initial value is $+\infty$.

- It basically reduces/removes all the nodes which are not really affecting the final decision but making algorithm slow. Hence by pruning these nodes, it makes the algorithm fast.

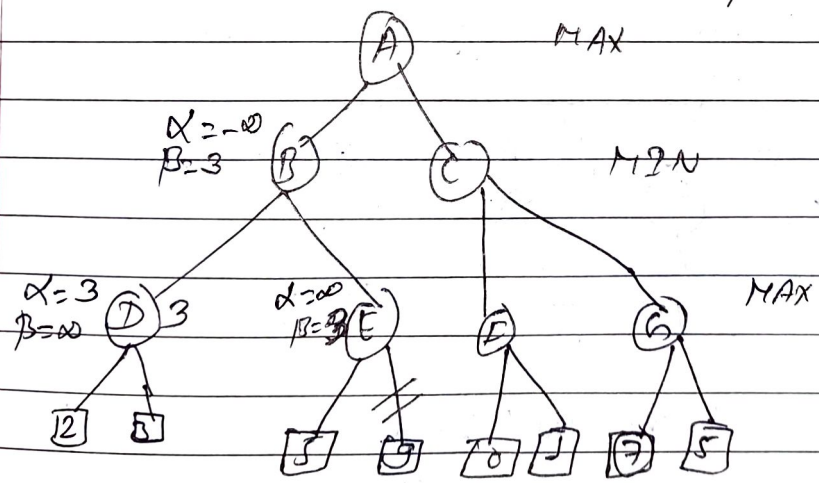
Condition for pruning: $\alpha >= \beta$

DATE



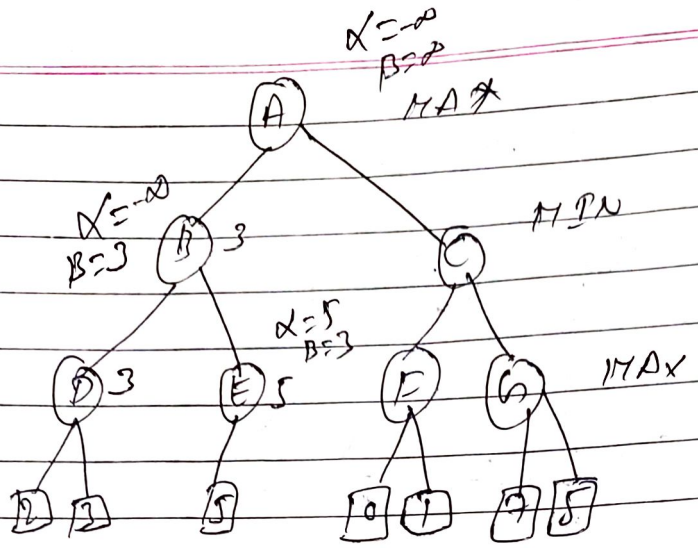
At node D, it's turn for MAX. So, the value of alpha will be calculated.
So, max(2,3) = 3 will be the value of α at node D and node value will also be 3.

Now, we backtrack to node B, where it's turn for MIN. Now, β = +∞. So, we will compare with available child nodes value i.e. MIN(∞, 3) = 3
Here, at node B now, α = -∞, β = 3.



Now, we travel next successor node of B, which is E.
Here, MAX will take his turn, max(5, 3) = 5 so, α = 5, β = 3, since α > β, so we prun the right successor

DATE



Now, backtrack from B to A.

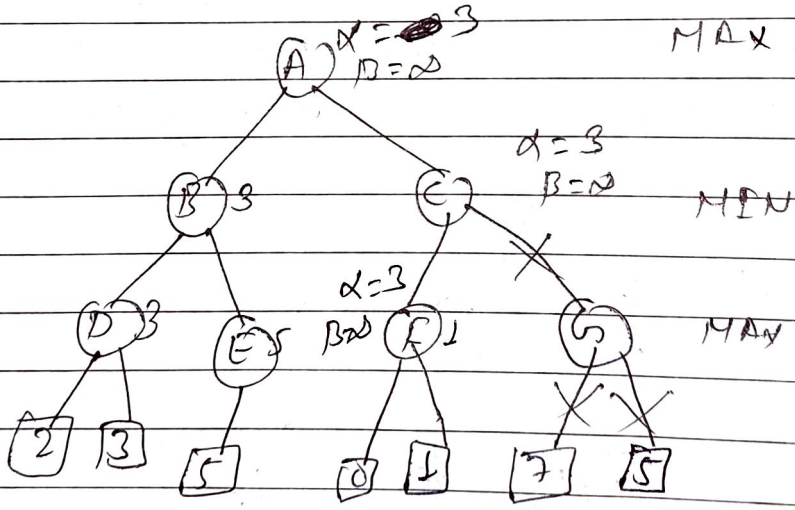
max. (-∞, 3) = 3 ∴ α = 3, β = ∞

At node C, α = 3, β = ∞

~~Next~~ At node F, α = 3, β = ∞

Now, At node F, MAX (3, 0) => max(3,0) = 3.

So, α = 3, β = ∞ and node value is 1

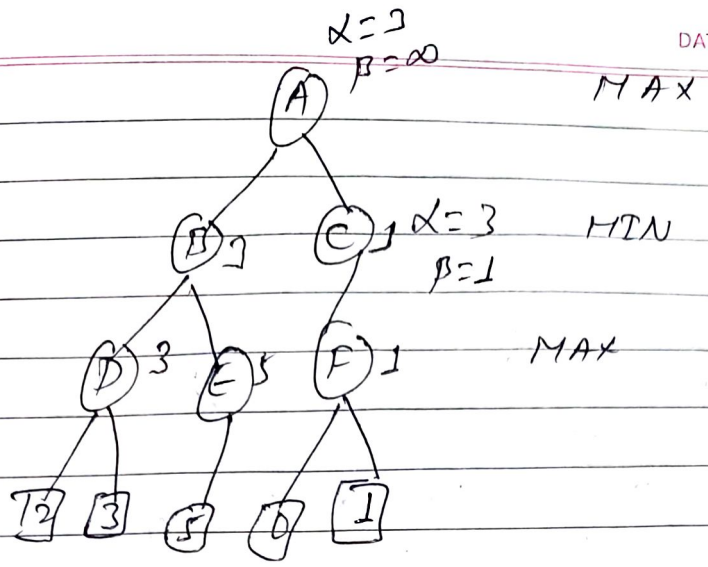


Now, F will return node value 1 to C. At C α = 3, β = ∞.

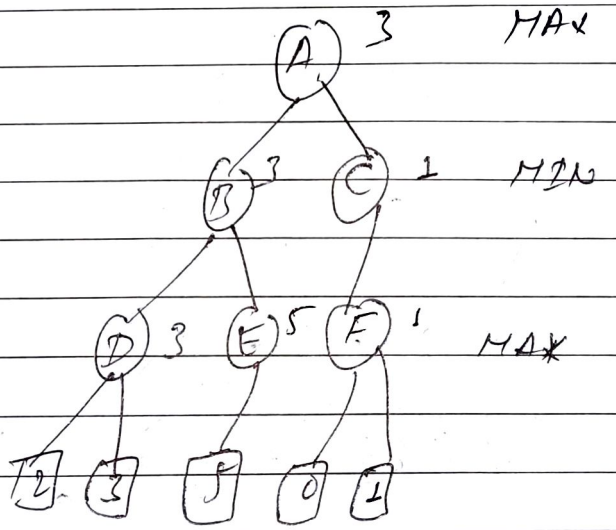
Here, At C, the value of β will be min(∞, 1) = 1.

∴ At C, α = 3, β = 1. Again it satisfies α ≥ β.

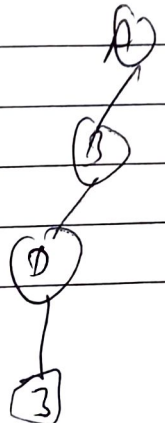
So, we prune next child of C.



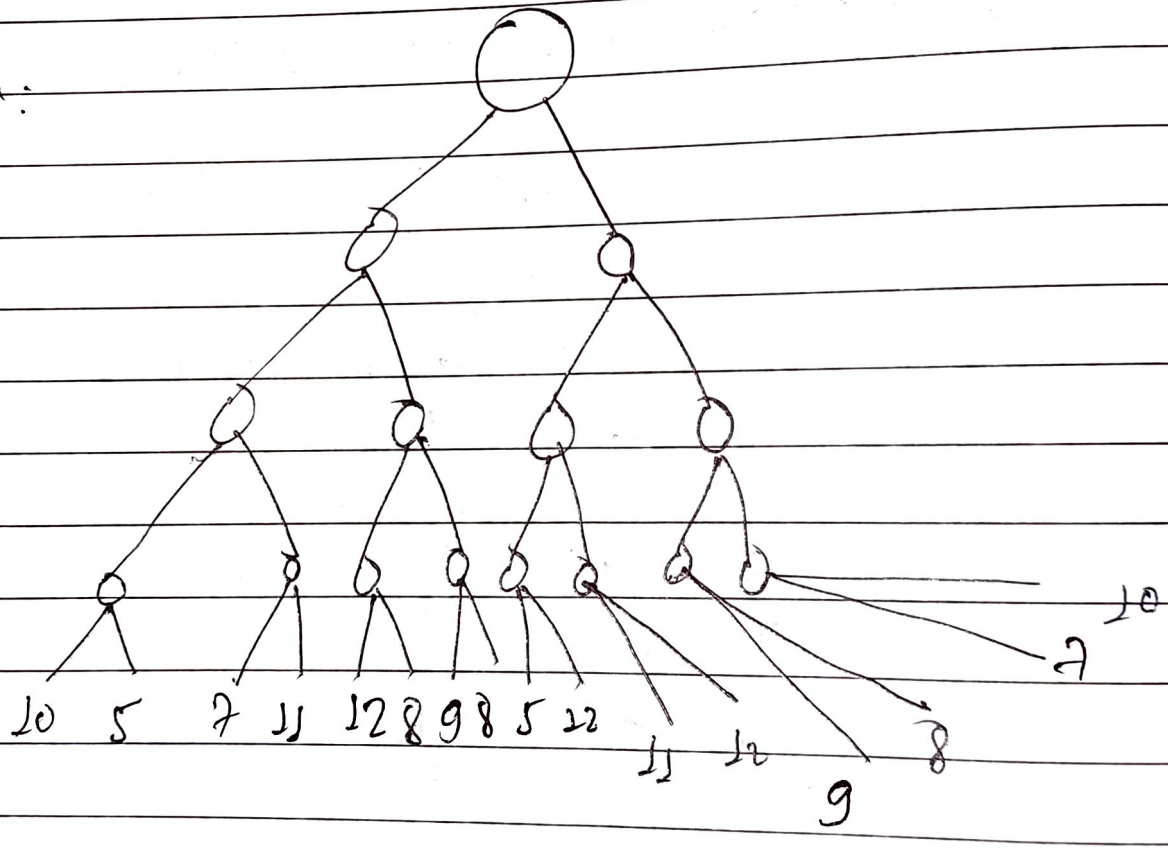
Now, C returns the value of 1 to A here the best value for A is max(3,1) = 3



Hence, best path for MAX will be



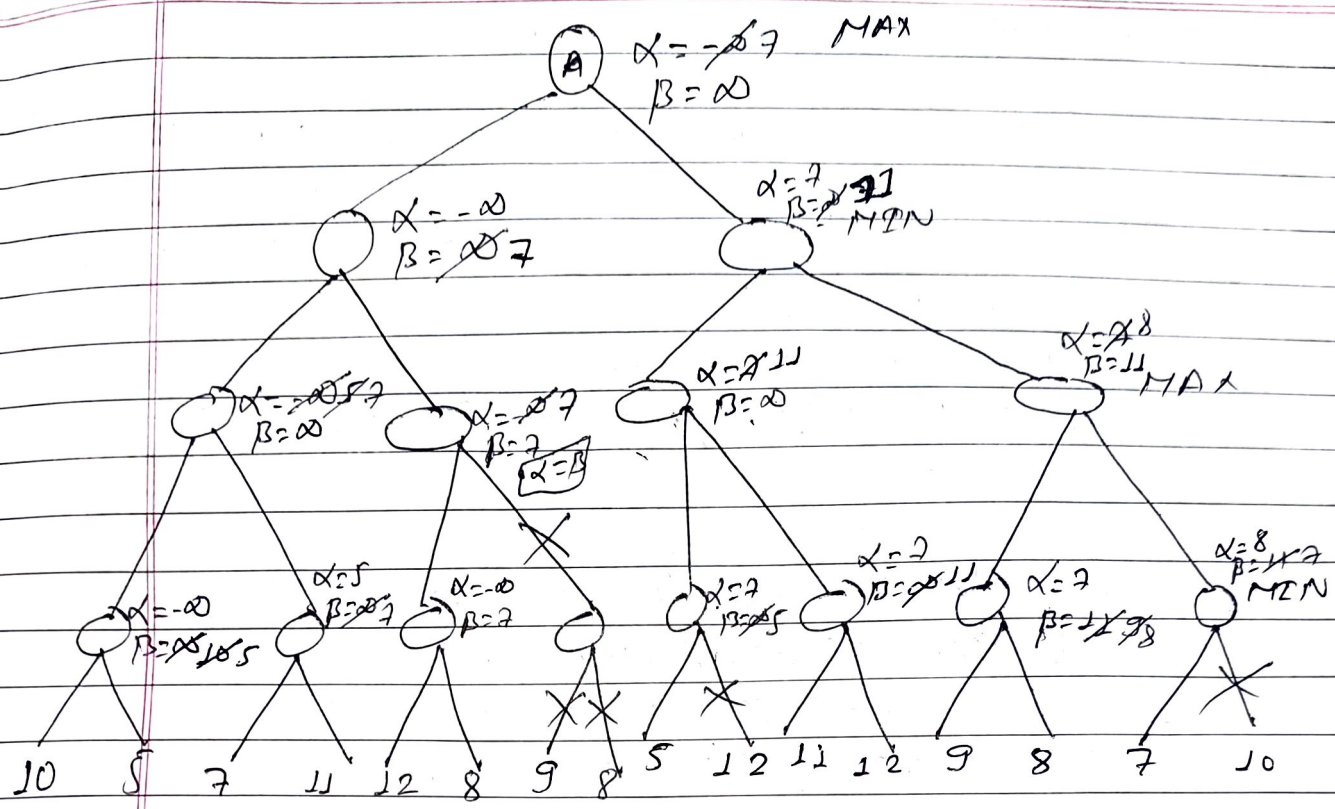
Ex:



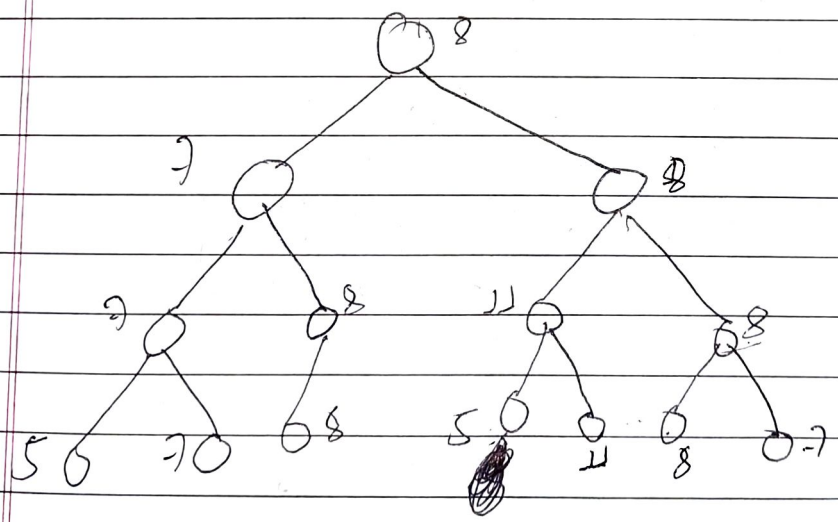
Initially $\alpha = -\infty$

$\beta = +\infty$

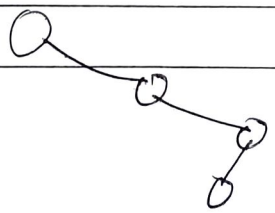
DATE



Hence, final result is.



best path is



DATE

* Constraint Satisfaction Problems.

A constraint satisfaction problem (CSP) consists of a set of variables, a domain of values for each variable and a set of constraints. The objective is to assign a value for each variable such that all constraints are satisfied.
Examples: The n-Queen problem, Crossword puzzle, Sudoku, etc.

A constraint satisfaction problem is characterized by
→ a set of variables $\{x_1, x_2, x_3, \ldots, x_n\}$.
→ for each variable x_i, a domain D_i with the possible values for that variable, and a set of constraints that are assumed to hold between the value of variables.

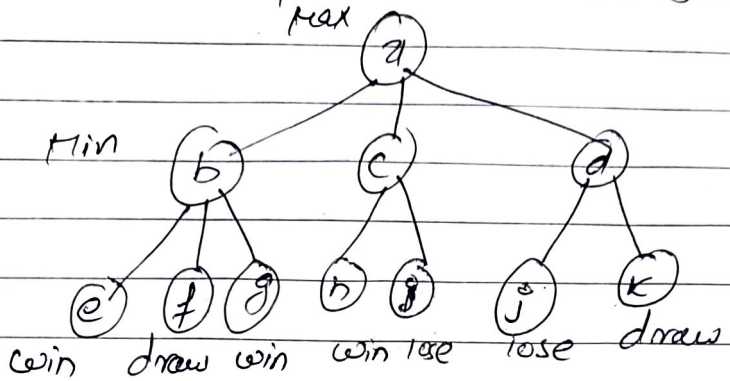
A constraint satisfaction problem is to find, for each i from 1 to n, a value in D_i for x_i so that all constraints are satisfied.
This problem can be easily stated as a sentence in first order logic of the form.

$$(\exists x_1) \ldots (\exists x_n).(D_1(x_1) \& \ldots D_n(x_n)) \Rightarrow (C_1 \ldots C_m)$$

A constraint satisfaction problem is usually represented as an undirected graph called constraint Graph where the nodes are the variables & the edges are the binary constraints

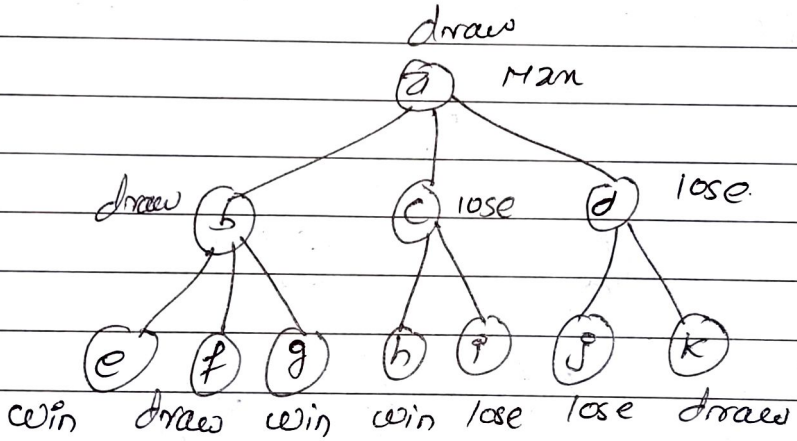
DATE

Ex: Consider the following game tree (drawn from the point of view of the Maximizing player)



What move should be chosen by the Max player, & what should be the response of the Min player, assuming that both are using the minimax procedure?

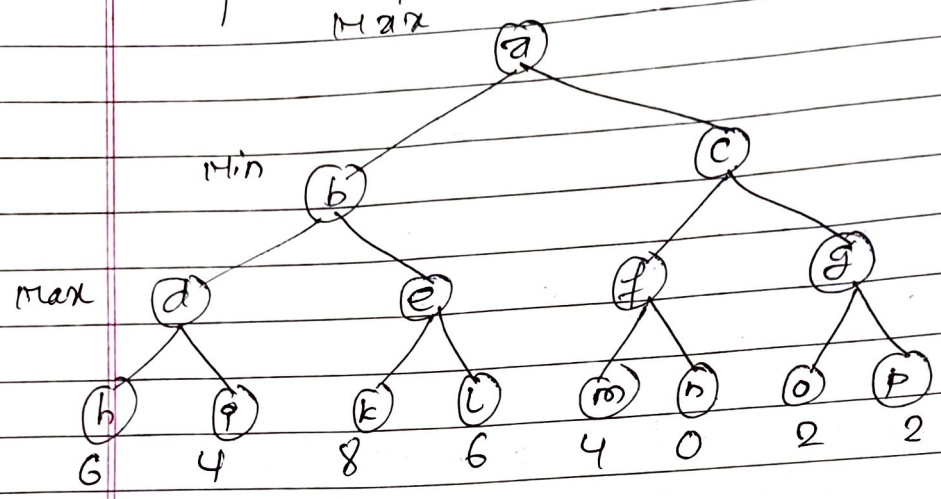
=>



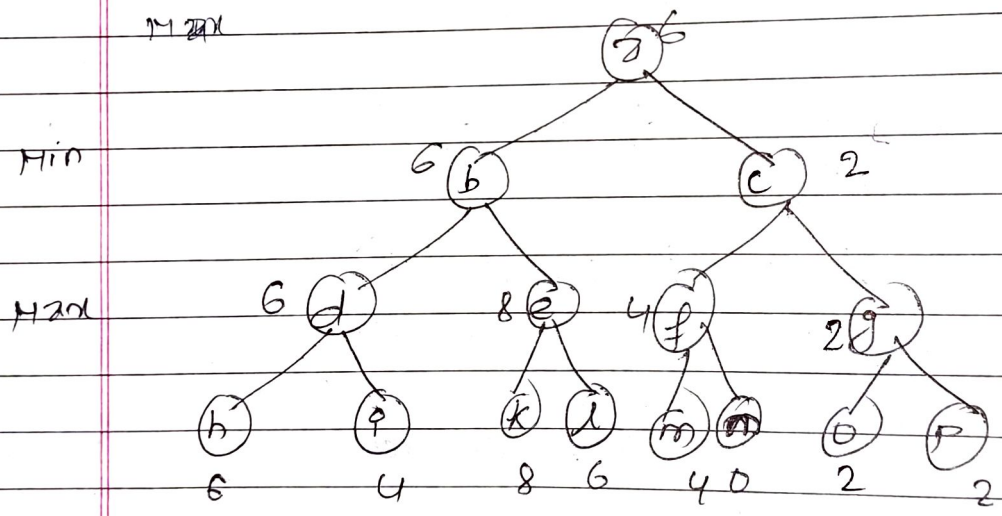
Max will move to b and Min will respond by moving to f.

DATE

Ex: Consider the following game tree (drown from the point of view of the Maximizing player):



a) Use the mini-max procedure and show what moves should be chosen by the two players.

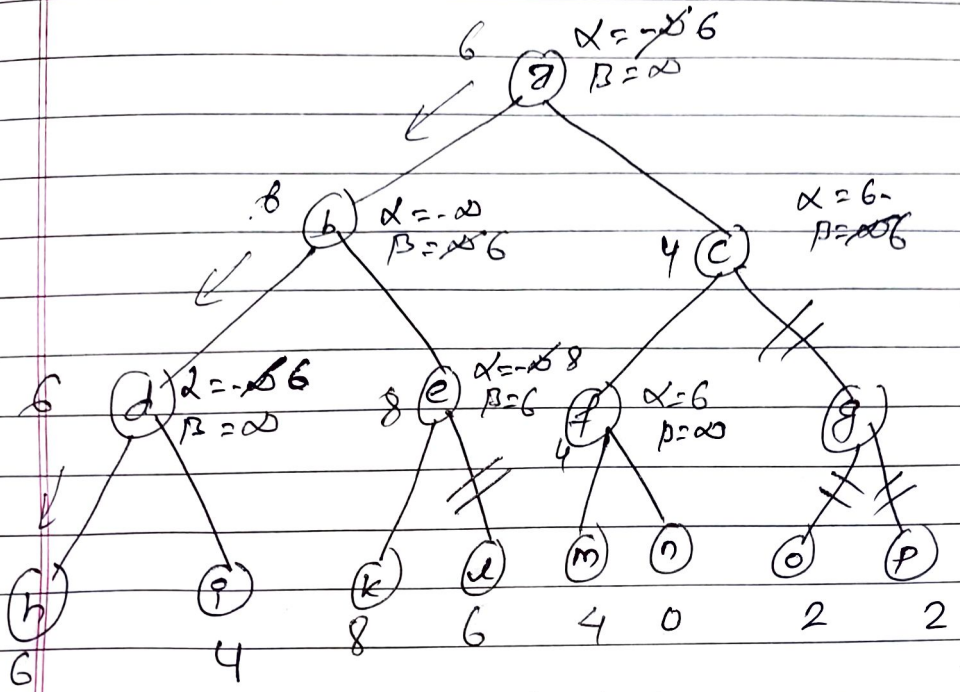


Path is a → b → d → h

Here, Max will move to b then min will move to d then max will move to h.

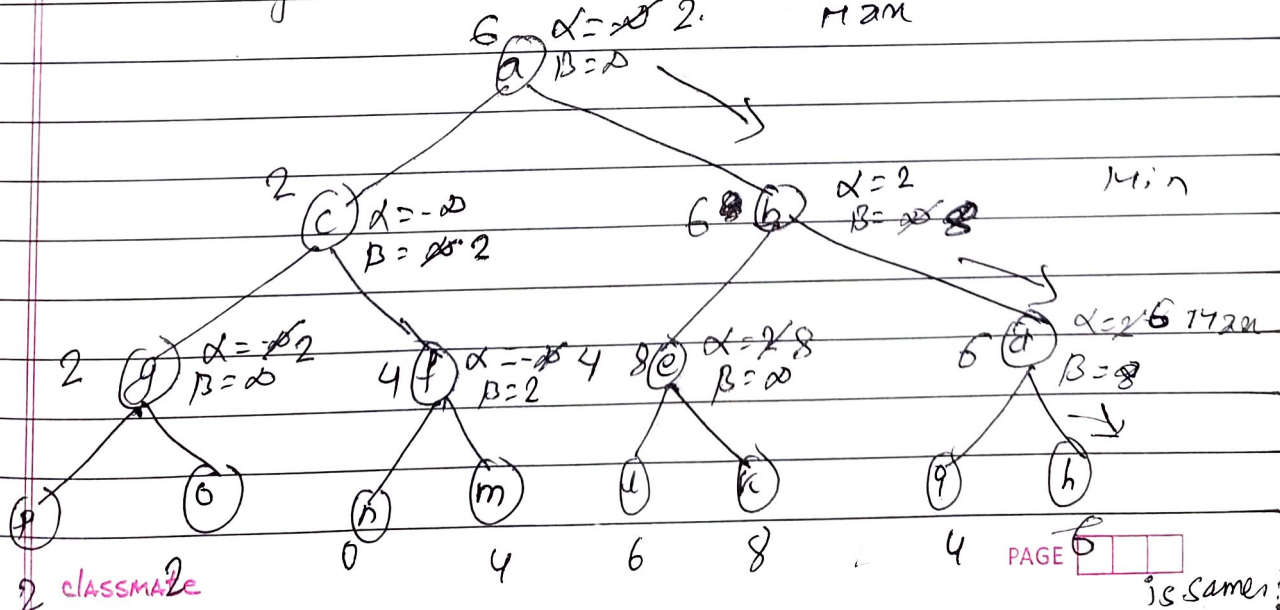
DATE

b) Use alpha-beta pruning procedure and show what nodes would not need to be examined



Hence nodes e, g, o, p would not need to be examined.

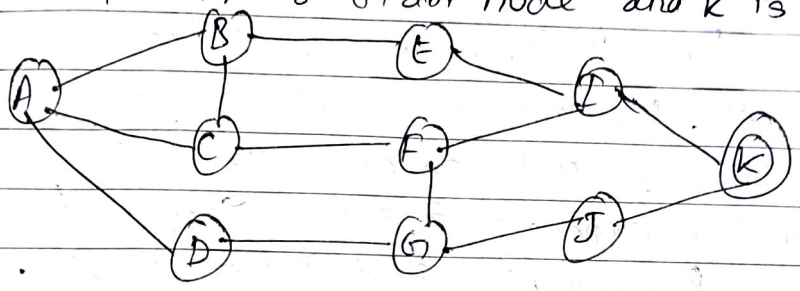
c) Consider the mirror image of the above tree and apply again the alpha-beta pruning procedure. What do you notice?



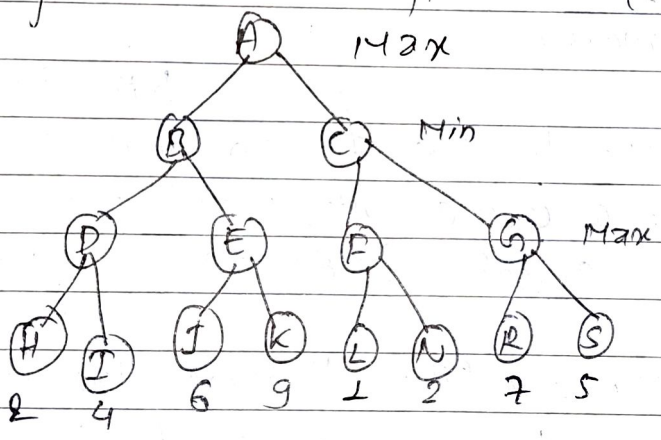
It does not produce any savings since path: a-b-d-h is same

Questions asked from this Chapter.

Q.1 How informed search are different than uninformed? Given following state space, illustrate how depth limited search and iterative deepning Search works? Use your own assumption for depth limit. Here, A is start node and K is goal node. (2078 - 10 marks)



Q.2 Given following search space, determine if there exists any alpha & beta cutoff. (2078 - 5 marks)



Q.3 Illustrate with an example, how uniform cost search algorithm can be used for finding goal in a state space. (2076 - 5 marks)

Q.4 Consider a following state space representing a game. Use minimax search to find solution & perform alpha beta pruning, if exists. (2076 - 5 marks)

fig. Same as Q.2. except terminal node values are

13 15 16 19 11 12 0 9

DATE

Q. Construct a state space with appropriate heuristic and local costs. Show that Greedy Best First Search is not complete for the state space. Also illustrate A* is complete and guarantees solution for the same state space. (2076 - 10 marks)

Q. Justify the Searching is one of the important parts of AI. Explain in detail about DFS & BFS techniques with an example. (2073 - 10 marks)
How can you expand it to informed Search? (2074 - 6 marks)
Explain in detail about informed search (2069 - 6 marks)

Q. In problem solving, what is the concept of state space, state, successor function, goal test and path cost? Illustrate each with suitable example. (2076 - 6 marks)

Q. If we set the heuristic function $h(n) = g(n)$ for both greedy as well as A*. What will be the effect in the algorithms? Explain? (2070 - 6 marks) (2068 - 6 marks)
$h(n) = -g(n)$ and
NOR: we get BFS when $h(n) = g(n)$

Q. The minimax algorithm returns the best move for MAX under the assumption that MIN play optimally. What happens when MIN plays suboptimally? (2070 - 6 marks)

PAGE