A* Search Algorithm

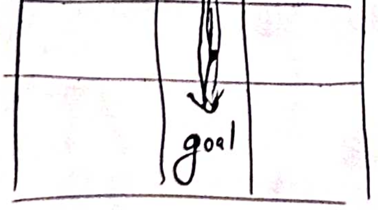
A* is a path finding and graph traversal algorithm that is often used to find the shortest path from a start node to a goal node. It uses a combination of actual cost from the start node ($g(n)$) and estimated cost to the goal ($h(n)$) - called the heuristic function.

Algorithm: ① Initialize open and closed lists. The open list contains the start node.

② While open-list is not empty.
- → Pick the node with lowest $f(n) = g(n) + h(n)$
- → If this node is the goal return the path
- → Otherwise expand this node by exploring its neighbours.
- → Update the costs and add the unvisited neighbours to the open list.

③ Move the current node to the closed list.

Example $h(n)$ is Manhattan distance is a grid search



3: Manhattan distance
'LRD' sequences length
or difference in X and Y absolutely added.

AO* Search (Decomposition) Algorithm

AO* algorithm is used for AND-OR graph searches, a problem that involves dependencies. It works by decomposing the problem into subproblems and solving each recursively

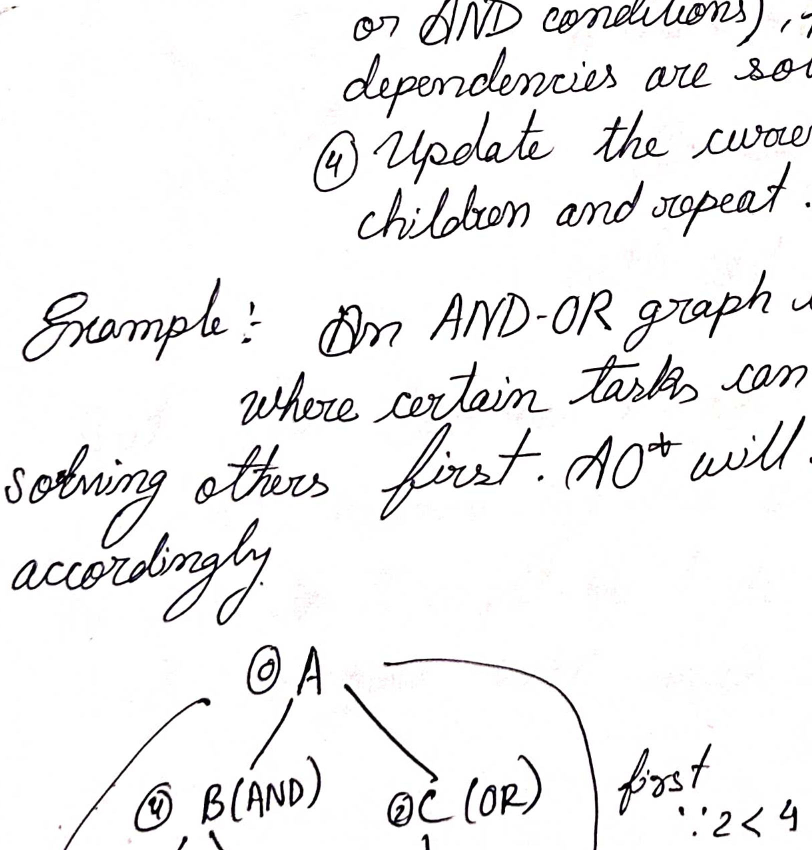
Algorithm ① Start from an arbitrary node, expand the most promising node

② If the expand leads to a solution return and mark as done.

③ If a problem has subproblems (dependencies or AND conditions), proceed when all dependencies are solved.

④ Update the current node based on its children and repeat.

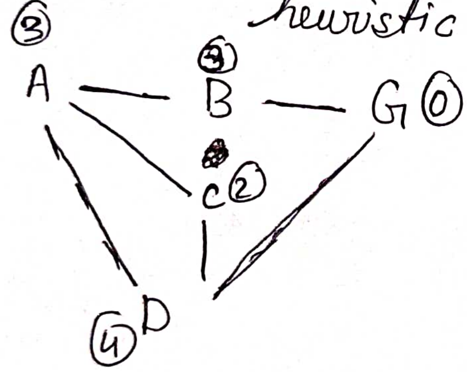
Example:- An AND-OR graph can represent tasks where certain tasks can only be solved by solving others first. AO* will explore such path accordingly



Greedy Search Algorithm

Greedy search selects the path that appears best at the current step without considering the overall path cost. It used only the heuristic $h(n)$

<u>Algorithm</u> ① Initalize the start node
② Expand the node with the smallest heuristic value.
③ Move to the next node with the best heuristic until the goal is reached



$A \rightarrow \cancel{B}\, \overset{\text{lowest in nbr[A]}}{\overset{\downarrow}{C}} \rightarrow D \rightarrow G$

On applying greedy search

with cost $3+2+4+0$
$= 9$

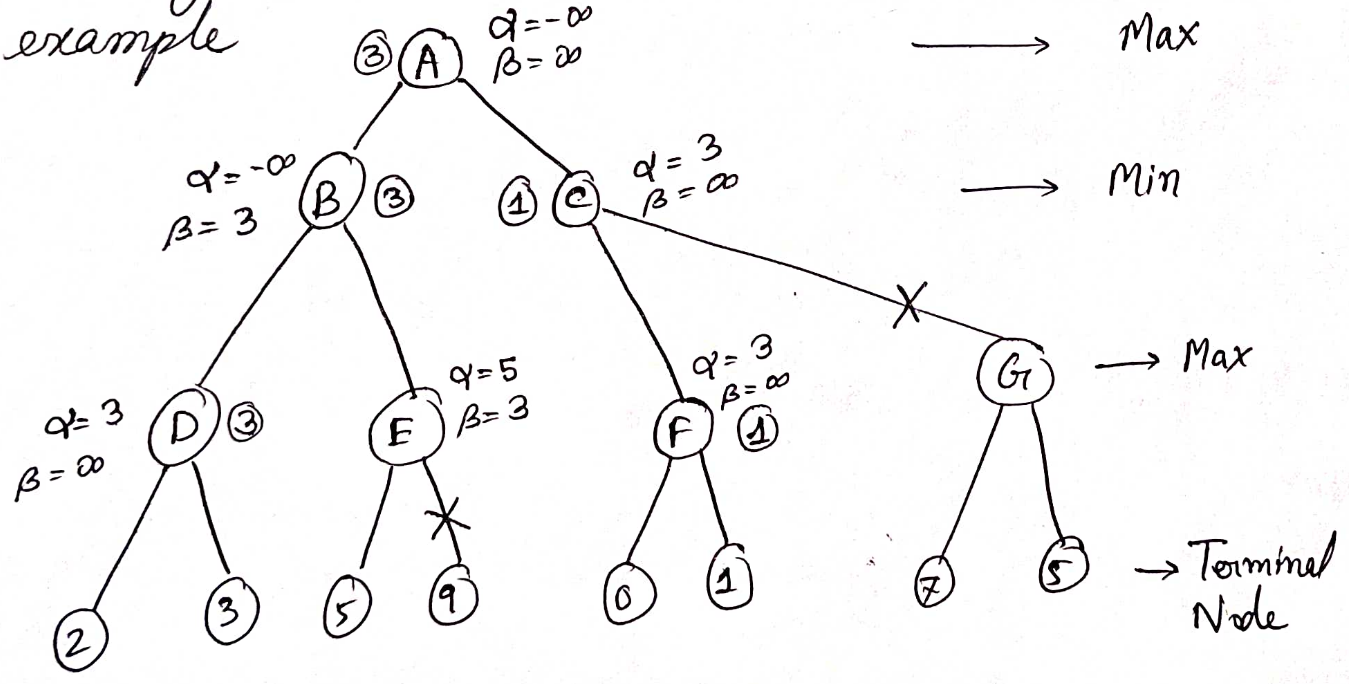
But this does not always lead to optimal solution. Proven by the above example while the cost to B might be high but the extra overhead leads to goal with much lower cost.

Alpha-Beta Pruning

It is an optimization over the minimax algorithm.

① $\alpha = -\infty$ $\beta = \infty$ ② only max player updates α
③ Only minplayer updates β ④ While backtracking node values are parsed and not the value of α and β
⑤ α, β is only passed downwards towards the child nodes. ⑥ Prune condition $\alpha >= \beta$.

Keeping these features / ~~@~~ rules lets apply this on an example



③ A $\alpha = -\infty$ $\beta = \infty$
$\alpha = -\infty$ $\beta = 3$ B ③
① C $\alpha = 3$ $\beta = \infty$
$\alpha = 3$ $\beta = \infty$ D ③
E $\alpha = 5$ $\beta = 3$
F $\alpha = 3$ $\beta = \infty$ ①
G
2
3
5
9
0
1
7
5
Max
Min
Max
Terminal Node

Worst ordering $O(b^m)$

Ideal ordering $O(b^{m/2})$

m = depth

b = branching factor