A*: It is best known from Best first search. It avoids expanding paths that already expensive but expanding most promising path first.

In this algorithm we use a formula:

F(n) = g(n) + h(n).

Where, g(n) = The cost to reach the node.

h(n) = Heuristic value.

F(n) = Estimated total cost to reach the goal.

Algorithm:

Insert the root node into the queue
While the queue is not empty
Dequeue the element with the highest priority
(If priorities are same, alphabetically smaller path is chosen)
If the path is ending in the goal state, print the path and exit
Else
Insert all the children of the dequeued element, with f(n) as the priority

IDA*: Iterative deepening A* (IDA*) is a graph traversal and path search algorithm that can find the shortest path between a designated start node and any member of a set of goal nodes in a weighted graph. It is a variant of iterative deepening depth-first search that borrows the idea to use a heuristic function to evaluate the remaining cost to get to the goal from the A* search algorithm.

Algorithm:

IDA* Algorithm

- In the first iteration, we determine a **"f-cost limit" – cut-off value**

  $f(n_0) = g(n_0) + h(n_0) = h(n_0)$, where n_0 is the start node.

- We expand nodes using the **depth-first algorithm** and backtrack whenever f(n) for an expanded node n exceeds the cut-off value.
- If this search does not succeed, determine the **lowest f-value** among the nodes that were visited but not expanded.
- Use this f-value as the **new limit value – cut-off value** and do another depth-first search.
- Repeat this procedure until a goal node is found.

SMA*: Simplified Memory Bounded A* is a shortest path algorithm based on the A* algorithm. The main advantage of SMA* is that it uses a bounded memory, while the A* algorithm might need exponential memory. All other characteristics of SMA* are inherited from A*.

Properties:

SMA* has the following properties

- It works with a heuristic, just as A*
- It is complete if the allowed memory is high enough to store the shallowest solution
- It is optimal if the allowed memory is high enough to store the shallowest optimal solution, otherwise it will return the best solution that fits in the allowed memory
- It avoids repeated states as long as the memory bound allows it
- It will use all memory available
- Enlarging the memory bound of the algorithm will only speed up the calculation
- When enough memory is available to contain the entire search tree, then calculation has an optimal speed

SMA* Algorithm

- SMA* makes use of all available memory M to carry out the search
- It avoids repeated states as far as memory allows
- Forgotten nodes: shallow nodes with high f-cost will be dropped from the fringe
- A forgotten node will only be regenerated if all other child nodes from its ancestor node have been shown to look worse
- Memory limitations can make a problem intractable