AI-7

Searching is the universal technique of problem solving in AI. There are some single-player games such as tile games, Sudoku, crossword, etc. The search algorithms help you to search for a particular position in such games.

## Single Agent Pathfinding Problems

The games such as 3X3 eight-tile, 4X4 fifteen-tile, and 5X5 twenty four tile puzzles are single-agent-path-finding challenges. They consist of a matrix of tiles with a blank tile. The player is required to arrange the tiles by sliding a tile either vertically or horizontally into a blank space with the aim of accomplishing some objective.

The other examples of single agent pathfinding problems are Travelling Salesman Problem, Rubik’s Cube, and Theorem Proving.

## Search Terminology

* Problem Space − It is the environment in which the search takes place. (A set of states and set of operators to change those states)
* Problem Instance − It is Initial state + Goal state.
* Problem Space Graph − It represents problem state. States are shown by nodes and operators are shown by edges.
* Depth of a problem − Length of a shortest path or shortest sequence of operators from Initial State to goal state.
* Space Complexity − The maximum number of nodes that are stored in memory.
* Time Complexity − The maximum number of nodes that are created.
* Admissibility − A property of an algorithm to always find an optimal solution.
* Branching Factor − The average number of child nodes in the problem space graph.
* Depth − Length of the shortest path from initial state to goal state.

## Brute-Force Search Strategies

They are most simple, as they do not need any domain-specific knowledge. They work fine with small number of possible states.

Requirements −

* State description
* A set of valid operators
* Initial state
* Goal state description

### Breadth-First Search

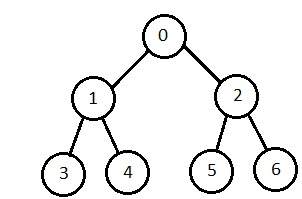
It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

If branching factor (average number of child nodes for a given node) = b and depth = d, then number of nodes at level d = bd.

The total no of nodes created in worst case is b + b2 + b3 + … + bd.

Disadvantage − Since each level of nodes is saved for creating next one, it consumes a lot of memory space. Space requirement to store nodes is exponential.

Its complexity depends on the number of nodes. It can check duplicate nodes.



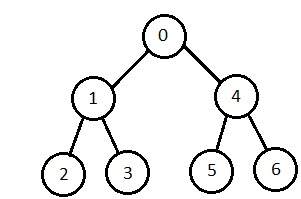
### Depth-First Search

It is implemented in recursion with LIFO stack data structure. It creates the same set of nodes as Breadth-First method, only in the different order.

As the nodes on the single path are stored in each iteration from root to leaf node, the space requirement to store nodes is linear. With branching factor *b* and depth as *m*, the storage space is *bm.*

Disadvantage − This algorithm may not terminate and go on infinitely on one path. The solution to this issue is to choose a cut-off depth. If the ideal cut-off is *d*, and if chosen cut-off is lesser than *d*, then this algorithm may fail. If chosen cut-off is more than *d*, then execution time increases.

Its complexity depends on the number of paths. It cannot check duplicate nodes.



### Bidirectional Search

It searches forward from initial state and backward from goal state till both meet to identify a common state.

The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

### Uniform Cost Search

Sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost.

It explores paths in the increasing order of cost.

Disadvantage − There can be multiple long paths with the cost ≤ C\*. Uniform Cost search must explore them all.

### Iterative Deepening Depth-First Search

It performs depth-first search to level 1, starts over, executes a complete depth-first search to level 2, and continues in such way till the solution is found.

It never creates a node until all lower nodes are generated. It only saves a stack of nodes. The algorithm ends when it finds a solution at depth *d*. The number of nodes created at depth *d* is bd and at depth *d-1* is bd-1.

