A* Algorithm:

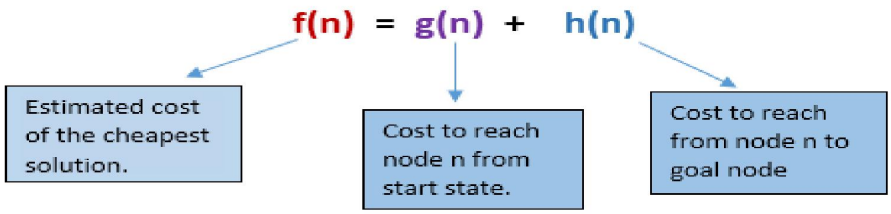
- A* Algorithm is one of the best and popular techniques used for path finding and graph traversals.
- A lot of games and web-based maps use this algorithm for finding the shortest path efficiently.
- It is essentially a best first search algorithm.

Working Procedure:

A* Algorithm works as follows:

- It maintains a tree of paths originating at the start node.
- It extends those paths one edge at a time.
- It continues until its termination criterion is satisfied.

A* Algorithm extends the path that minimizes the following function-



Here,

- 'n' is the last node on the path
- g(n) is the cost of the path from start node to node 'n'
- h(n) is a heuristic function that estimates cost of the cheapest path from node 'n' to the goal node

Algorithm:

- The implementation of A* Algorithm involves maintaining two lists- OPEN and CLOSED.
- OPEN contains those nodes that have been evaluated by the heuristic function but have not been expanded into successors yet.
- CLOSED contains those nodes that have already been visited.

Heuristics Value Calculation

The calculation of h(n) can be done in various ways: The Manhattan Distance from node to the goal is often used. This is a standard heuristic for a grid. If $h(n) = 0$, A* becomes Dijkstra's algorithm, which is guaranteed to find a shortest path. The heuristic function must be admissible, which means it can never overestimate the cost to reach the goal. Both the Manhattan distance and $h(n) = 0$ are admissible.

Heuristics

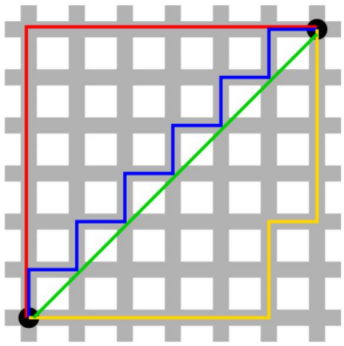
Using a good heuristic is important in determining the performance of $A*$. The value of $h(n)$ would ideally equal the exact cost of reaching the destination. This is, however, not possible because we do not even know the path. We can, however, choose a method that will give us the exact value some of the time, such as when traveling in a straight line with no obstacles. This will result in a perfect performance of $A*$ in such a case.

We want to be able to select a function $h(n)$ that is less than the cost of reaching our goal. This will allow h to work accurately, if we select a value of that is greater; it will lead to a faster but less accurate performance. Thus, it is usually the case that we choose an $h(n)$ that is less than the real cost.

The Manhattan Distance Heuristic:

This method of $h(n)$ computing is called the Manhattan method because it is computed by calculating the total number of squares moved horizontally and vertically to reach the target square from the current square. We ignore diagonal movement and any obstacles that might be in the way.

$$\boldsymbol{h = |x_{start} - x_{destination}| + |y_{start} - y_{destination}|}$$

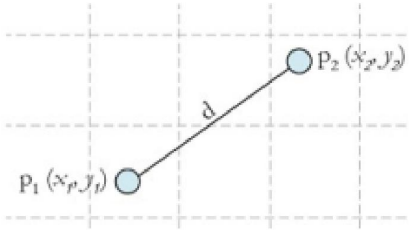


This heuristic is exact whenever our path follows a straight line. That is, will find paths that are combinations of straight line movements. Sometimes we might prefer a path that tends to follow a straight line directly to our destination.

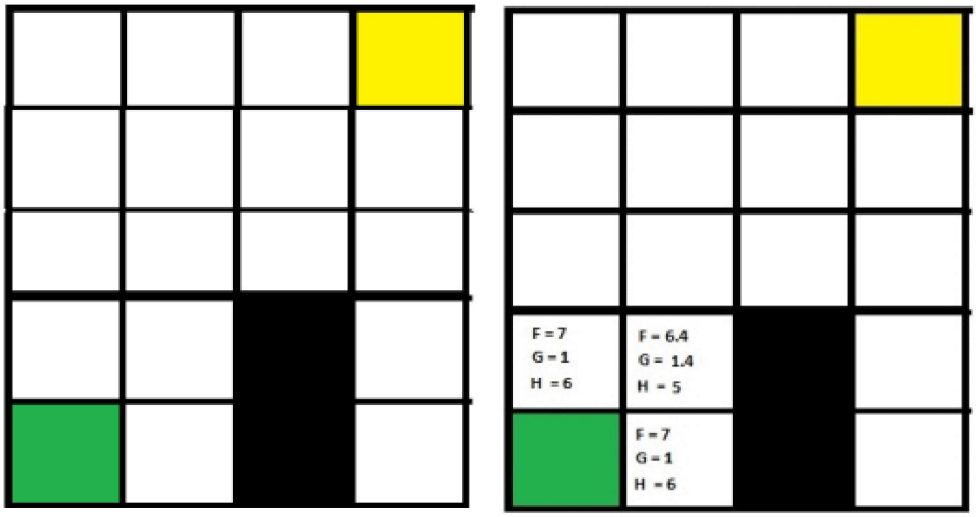
The Euclidean Distance Heuristic:

This heuristic is slightly more accurate than its Manhattan counterpart. If we try run both simultaneously on the same maze, the Euclidean path finder favors a path along a straight line. This is more accurate but it is also slower because it has to explore a larger area to find the path.

$$\boldsymbol{h = \sqrt{(x_{start} - x_{destination})^2 + (y_{start} - y_{destination})^2}}$$

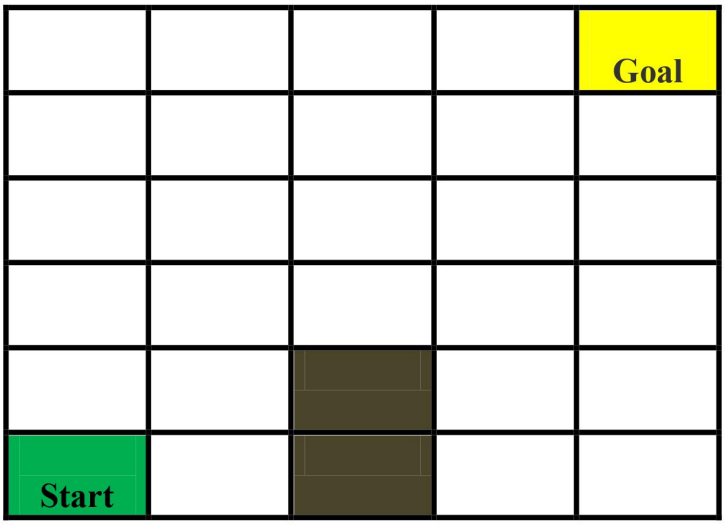


Let us start by choosing an admissible heuristic. For this case, we can use the Manhattan heuristic. We then proceed to the starting cell. We call it our current cell and then we proceed to look at all its neighbors and compute *f*, *g*, *h* for each of them. We then select the neighbor with the lowest *f* cost. This is our new current cell and we then repeat the process above (populates neighbors and compute *f*, *g*, *h* and choose the lowest). We do this until we are at the goal cell. The image below demonstrates how the search proceeds. In each cell the respective *f*, *g* and *h* values are shown. **Remember**, *g* is the cost that has been accrued in reaching the cell and *h* is the Manhattan distance towards the yellow cell while *f* is the sum of *g* and *h*.



Problem-01(For Odd ID Students): Solve the following Maze Game using A* Algorithm.

Find the most cost-effective path to reach the final state from initial state using A* Algorithm.
The agent will avoid the obstacles during traversing.



Problem-02(For Even ID Students): Solve the following Maze Game using A* Algorithm.

Find the most cost-effective path to reach the final state from initial state using A* Algorithm.
The agent will avoid the obstacles during traversing.