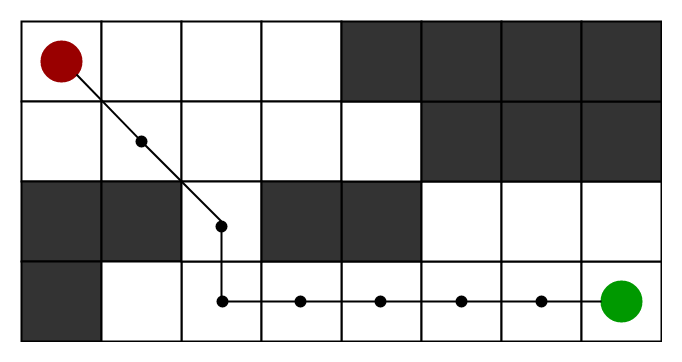
**A\* Search Algorithm**

**Motivation**   
To approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances.  
We can consider a 2D Grid having several obstacles and we start from a source cell (colored red below) to reach towards a goal cell (colored green below)



**What is A\* Search Algorithm?**   
A\* Search algorithm is one of the best and popular technique used in path-finding and graph traversals.

**Why A\* Search Algorithm?**   
Informally speaking, A\* Search algorithms, unlike other traversal techniques, it has “brains”. What it means is that it is really a smart algorithm which separates it from the other conventional algorithms. This fact is cleared in detail in below sections.   
And it is also worth mentioning that many games and web-based maps use this algorithm to find the shortest path very efficiently (approximation).

**Explanation**   
Consider a square grid having many obstacles and we are given a starting cell and a target cell. We want to reach the target cell (if possible) from the starting cell as quickly as possible. Here A\* Search Algorithm comes to the rescue.  
What A\* Search Algorithm does is that at each step it picks the node according to a value-‘**f**’ which is a parameter equal to the sum of two other parameters – ‘**g**’ and ‘**h**’. At each step it picks the node/cell having the lowest ‘**f**’, and process that node/cell.  
We define ‘**g**’ and ‘**h**’ as simply as possible below  
**g** = the movement cost to move from the starting point to a given square on the grid, following the path generated to get there.   
**h** = the estimated movement cost to move from that given square on the grid to the final destination. This is often referred to as the heuristic, which is nothing but a kind of smart guess. We really don’t know the actual distance until we find the path, because all sorts of things can be in the way (walls, water, etc.). There can be many ways to calculate this ‘h’ which are discussed in the later sections.

**Algorithm**   
We create two lists – Open List and Closed List (just like Dijkstra Algorithm)

// A\* Search Algorithm

1. Initialize the open list

2. Initialize the closed list

put the starting node on the open

list (you can leave its **f** at zero)

3. while the open list is not empty

a) find the node with the least **f** on

the open list, call it "q"

b) pop q off the open list

c) generate q's 8 successors and set their

parents to q

d) for each successor

i) if successor is the goal, stop search

ii) else, compute both **g** and **h** for successor

successor.**g** = q.**g** + distance between

successor and q

successor.**h** = distance from goal to

successor (This can be done using many

ways, we will discuss three heuristics-

Manhattan, Diagonal and Euclidean

Heuristics)

successor.**f** = successor.**g** + successor.**h**

iii) if a node with the same position as

successor is in the OPEN list which has a

lower **f** than successor, skip this successor

iV) if a node with the same position as

successor is in the CLOSED list which has

a lower **f** than successor, skip this successor

otherwise, add the node to the open list

end (for loop)

e) push q on the closed list

end (while loop)

So suppose as in the below figure if we want to reach the target cell from the source cell, then the A\* Search algorithm would follow path as shown below. Note that the below figure is made by considering Euclidean Distance as a heuristics.

