A\* Search Algorithm is a Path Finding Algorithm. It is similar to Breadth First Search(BFS). It will search shortest path using heuristic value assigned to node and actual cost from Source\_node to Dest\_node

#### **Real-life Examples**

* Maps
* Games

#### **Formula for A\* Algorithm**

h(n) **=** heuristic\_value

g(n) **=** actual\_cost

f(n) **=** actual\_cost **+** heursitic\_value

f(n) **=** g(n) **+** h(n)

Code:

**def** aStarAlgo(start\_node, stop\_node):

open\_set **=** set(start\_node) *# {A}, len{open\_set}=1*

closed\_set **=** set()

g **=** {} *# store the distance from starting node*

parents **=** {}

g[start\_node] **=** 0

parents[start\_node] **=** start\_node *# parents['A']='A"*

**while** len(open\_set) **>** 0 :

n **=** **None**

**for** v **in** open\_set: *# v='B'/'F'*

**if** n **==** **None** **or** g[v] **+** heuristic(v) **<** g[n] **+** heuristic(n):

n **=** v *# n='A'*

**if** n **==** stop\_node **or** Graph\_nodes[n] **==** **None**:

**pass**

**else**:

**for** (m, weight) **in** get\_neighbors(n):

*# nodes 'm' not in first and last set are added to first*

*# n is set its parent*

**if** m **not** **in** open\_set **and** m **not** **in** closed\_set:

open\_set**.**add(m) *# m=B weight=6 {'F','B','A'} len{open\_set}=2*

parents[m] **=** n *# parents={'A':A,'B':A} len{parent}=2*

g[m] **=** g[n] **+** weight *# g={'A':0,'B':6, 'F':3} len{g}=2*

*#for each node m,compare its distance from start i.e g(m) to the*

*#from start through n node*

**else**:

**if** g[m] **>** g[n] **+** weight:

*#update g(m)*

g[m] **=** g[n] **+** weight

*#change parent of m to n*

parents[m] **=** n

*#if m in closed set,remove and add to open*

**if** m **in** closed\_set:

closed\_set**.**remove(m)

open\_set**.**add(m)

**if** n **==** **None**:

print('Path does not exist!')

**return** **None**

*# if the current node is the stop\_node*

*# then we begin reconstructin the path from it to the start\_node*

**if** n **==** stop\_node:

path **=** []

**while** parents[n] **!=** n:

path**.**append(n)

n **=** parents[n]

path**.**append(start\_node)

path**.**reverse()

print('Path found: {}'**.**format(path))

**return** path

*# remove n from the open\_list, and add it to closed\_list*

*# because all of his neighbors were inspected*

open\_set**.**remove(n)*# {'F','B'} len=2*

closed\_set**.**add(n) *#{A} len=1*

print('Path does not exist!')

**return** **None**

*#define fuction to return neighbor and its distance*

*#from the passed node*

**def** get\_neighbors(v):

**if** v **in** Graph\_nodes:

**return** Graph\_nodes[v]

**else**:

**return** **None**

*#for simplicity we ll consider heuristic distances given*

*#and this function returns heuristic distance for all nodes*

**def** heuristic(n):

H\_dist **=** {

'A': 10,

'B': 8,

'C': 5,

'D': 7,

'E': 3,

'F': 6,

'G': 5,

'H': 3,

'I': 1,

'J': 0

}

**return** H\_dist[n]

*#Describe your graph here*

Graph\_nodes **=** {

'A': [('B', 6), ('F', 3)],

'B': [('C', 3), ('D', 2)],

'C': [('D', 1), ('E', 5)],

'D': [('C', 1), ('E', 8)],

'E': [('I', 5), ('J', 5)],

'F': [('G', 1),('H', 7)] ,

'G': [('I', 3)],

'H': [('I', 2)],

'I': [('E', 5), ('J', 3)],

}

aStarAlgo('A', 'J')

Output:

Path found: ['A', 'F', 'G', 'I', 'J']