A* Search

Aim:

To find the shortest path between a start node & goal node in a graph or grid, exploring only the most promising paths.

code:

```
from Queue import priority Queue
def a_star_search (graph, start, goal):
    open_list = priority Queue()
    open_list.put((0, start)).
    came_from = {}.
    g_score = {start :0}
    f_score = {start : heuristic (start, goal)}
    came_from [start] = none
    while not open_list.empty():
        current = open_list.get() [1].
        if current == goal:
            return
        reconstruct_path (come_from, current)
        for neighbor, cost in graph [current]:
        tentative_g_score = g score [current] + cost
    if neighbor not in g_score or tentative_g_
        score < g_score [neighbor]:
            come_from [neighbor] = current
    g_score [neighbor] = tentative_g_score
    f_score [neighbor] = tentative_g_score +
        heuristic (neighbor, goal)
    open_list.put ((f_score [neighbor], neighbor))
        return none.
```

```
def heuristic(node, goal):
    return abs (node [0] - goal [0]) + abs(node[1] -
goal [1])

def reconstruct - path (come - from, current):
    total path = [current]
    while current in come - from and come -
    from [current ] is not none :
        current = come - from [current]
            total - path . append (current)
        total - path . reverse ().
        return total - path.


graph = {(0,0): [((0,1), 1), ((1,0), 1)]
         (0,1): [((0,0), 1), ((1,1), 1)]
         (1,0): [((0,0), 1), ((1,1), 1)]
         (1,1): [((1,0), 1), ((0,1), 1),
         ((2,2), 1)].
             (2,2): [].

        }
    Start = (0,0)
     goal = (2,2)
    Path = a_star_search (graph, start,
                 goal)
    print (f"path found: {path}")
```

Algorithm of A* Search.

* Start & Initialize.
  * Create two sets → open list & closed list

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

* $g(n)$ → cost from start to current node
  $f(n)$ → total estimated cost
  h → start.

* Select the node from the openlist with lowest $f(n)$
* node is the goal, reconstruct and return the path
* otherwise add the node to the closed list.
* neighbor node is the closed list and the new $g(n)$ is higher and skip it.

Scanned with CamScanner

* If low.
  - → Update the neighbors g(n) & f(n)
  - → Set the current node as the neighbors parent
  - → neighbor not in open list add it
* Terminate, if the goal is reach.
* Stop.

Result:

Thus the program is successfully executed and output is verified.