

TITLE: Implement A* Algorithm for any game search problem

AIM: Aim of this practical is to apply algorithmic logic for any game

OBJECTIVES: Based on above main aim following are the objectives

1. To study A* algorithm
2. To study different game search problems
3. To determine performance A* algorithm in various games

MOTIVATION: To approximate the shortest path in real-life situations, like- in maps, games where there can be many hindrances.

INTRODUCTION

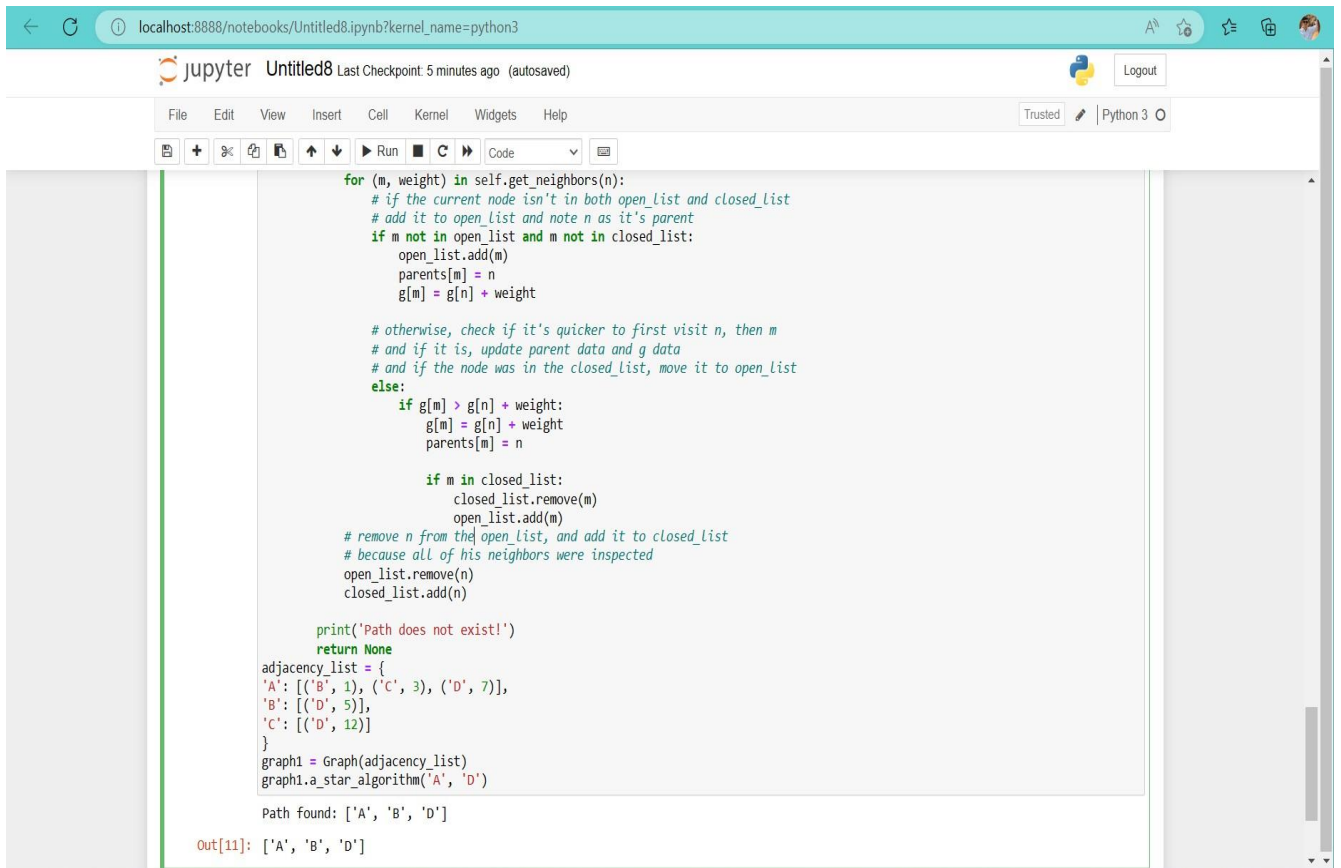
A* Search Algorithm:

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

Program :-



```
for (m, weight) in self.get_neighbors(n):
    # if the current node isn't in both open_list and closed_list
    # add it to open_list and note n as it's parent
    if m not in open_list and m not in closed_list:
        open_list.add(m)
        parents[m] = n
        g[m] = g[n] + weight

    # otherwise, check if it's quicker to first visit n, then m
    # and if it is, update parent data and g data
    # and if the node was in the closed_list, move it to open_list
    else:
        if g[m] > g[n] + weight:
            g[m] = g[n] + weight
            parents[m] = n

            if m in closed_list:
                closed_list.remove(m)
                open_list.add(m)

    # remove n from the open_list, and add it to closed_list
    # because all of his neighbors were inspected
    open_list.remove(n)
    closed_list.add(n)

    print('Path does not exist!')
    return None

adjacency_list = {
    'A': [('B', 1), ('C', 3), ('D', 7)],
    'B': [('D', 5)],
    'C': [('D', 12)]
}

graph1 = Graph(adjacency_list)
graph1.a_star_algorithm('A', 'D')

Path found: ['A', 'B', 'D']

Out[11]: ['A', 'B', 'D']
```

CONCLUSIONS: Thus we have studied A* algorithm in detail and implemented it in game search problem

