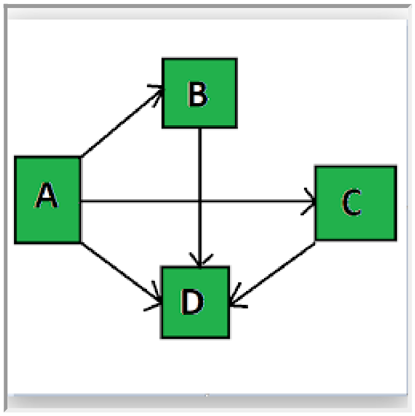
```
%%html
<iframe src="https://drive.google.com/file/d/1Ndszq_bmdkBF9Ltqds377atj9Dm6WZ9R/preview" width
```




```
def aStarAlgo(start_node, stop_node):

        open_set = set(start_node)
        closed_set = set()
        g = {} #store distance from starting node
        parents = {}# parents contains an adjacency map of all nodes
         #ditance of starting node from itself is zero
        g[start_node] = 0
        #start_node is root node i.e it has no parent nodes
        #so start_node is set to its own parent node
        parents[start_node] = start_node


        while len(open_set) > 0:
            n = None
            print("n=", n)
            print("open_set",open_set)
            print("closed_set",closed_set)

            #node with lowest f() is found
            for v in open_set:
                if n == None or g[v] + heuristic(v) < g[n] + heuristic(n):
                    n = v
            print("n=", n)

            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
                    #print("m=",m)
                    #print("weight",weight)

                    if m not in open_set and m not in closed_set:
```

```
                if m not in open_set and m not in closed_set:
                    #print("inside")
                    open_set.add(m)
                    parents[m] = n
                    g[m] = g[n] + weight


                #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:
                            printf("\ninside if******",m)
                            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("\nHi")
                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
            print("open_set *** ",open_set)
            print("closed_set ***",closed_set)
            print("add")
            open_set.remove(n)
            closed_set.add(n)

        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:
        print("Graph_nodes[v]",Graph_nodes[v])
        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': 1,
            'B': 1,
            'C': 1,
            'D': 1,
        }

        return H_dist[n]

#Describe your graph here
Graph_nodes = {'A': [('B', 1), ('C', 3), ('D', 7)],'B': [('D', 5)],'C': [('D', 12)]}
aStarAlgo('A', 'D')
```

```
n= None
open_set {'A'}
closed_set set()
n= A
Graph_nodes[v] [('B', 1), ('C', 3), ('D', 7)]
open_set ***  {'D', 'A', 'C', 'B'}
closed_set *** set()
add
n= None
open_set {'D', 'C', 'B'}
closed_set {'A'}
n= B
Graph_nodes[v] [('D', 5)]
open_set ***  {'D', 'C', 'B'}
closed_set *** {'A'}
add
n= None
open_set {'D', 'C'}
closed_set {'A', 'B'}
n= C
Graph_nodes[v] [('D', 12)]
open_set ***  {'D', 'C'}
closed_set *** {'A', 'B'}
add
n= None
open_set {'D'}
closed_set {'A', 'C', 'B'}
n= D

Hi
```

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