In [ ]:

```
Github Link: https://github.com/Aaenoor/B208
```

In [1]:

```
class Node:
    def __init__(self, state, parent, actions, totalCost, heuristic):
        self.state = state
        self.parent = parent
        self.actions = actions
        self.totalCost = totalCost
        self.heuristic = heuristic
```

In [ ]:

```
import math
def findMin(frontier):
    minV = math.inf
    node = ""
    for i in frontier:
        if minV > frontier[i][1]:
            minV = frontier[i][1]
            node = i
    return node
```

In [20]:

```
def actionSequence(graph, goalState):
    solution = [goalState]
    currentParent = graph[goalState].parent
    while currentParent != None:
        solution.append(currentParent)
        currentParent = graph[currentParent].parent
    solution.reverse()
    return solution
```

In [24]:

```
def Astar(graph, initialState, goalState):
    frontier = dict()
    heuristicCost = math.sqrt(((graph[goalState].heuristic[0] - graph[initialState].heuristic[0])**2) + ((graph
    frontier[initialState] = (None, heuristicCost)
    explored = dict()
    while len(frontier) != 0:
        currentNode = findMin(frontier)
        del frontier[currentNode]
        if graph[currentNode].state == goalState:
            return actionSequence(graph, goalState)
        heuristicCost = math.sqrt(((graph[goalState].heuristic[0] - graph[currentNode].heuristic[0])**2) + ((gr
        currentCost = graph[currentNode].totalCost
        explored[currentNode] = (graph[currentNode].parent, heuristicCost + currentCost)
        for child in graph[currentNode].actions:
            currentCost = child[1] + graph[currentNode].totalCost
            heuristicCost = math.sqrt(((graph[goalState].heuristic[0] - graph[child[0]].heuristic[0]) ** 2) + (
            if child[0] in explored:
                if graph[child[0]].parent == currentNode or child[0] == initialState or explored[child[0]][1] <
                    continue
            if child[0] not in frontier:
                graph[child[0]].parent = currentNode
                graph[child[0]].totalCost = currentCost
                frontier[child[0]] = (graph[child[0]].parent, currentCost + heuristicCost)
            else:
                if frontier[child[0]][1] < currentCost + heuristicCost:
                    graph[child[0]].parent = frontier[child[0]][0]
                    graph[child[0]].totalCost = frontier[child[0]][1] - heuristicCost
                else:
                    frontier[child[0]] = (currentNode, currentCost + heuristicCost)
                    graph[child[0]].parent = frontier[child[0]][0]
                    graph[child[0]].totalCost = currentCost
```

In [25]:

```
graph = {
    "A" : Node("A", None, [("F", 1)], 0, (0, 0)),
    "B" : Node("B", None, [("G", 1), ("C", 1)], 0, (2, 0)),
    "C" : Node("C", None, [("B", 1), ("D", 1)], 0, (3, 0)),
    "D" : Node("D", None, [("C", 1), ("E", 1)], 0, (4, 0)),
    "E" : Node("E", None, [("D", 1)], 0, (5, 0)),
    "F" : Node("F", None, [("A", 1), ("H", 1)], 0, (0, 1)),
    "G" : Node("G", None, [("B", 1), ("J", 1)], 0, (2, 1)),
    "H" : Node("H", None, [("F", 1), ("I", 1), ("M", 1)], 0, (0, 2)),
    "I" : Node("I", None, [("H", 1), ("J", 1), ("N", 1)], 0, (1, 2)),
    "J" : Node("J", None, [("G", 1), ("I", 1)], 0, (2, 2)),
    "K" : Node("K", None, [("L", 1), ("P", 1)], 0, (4, 2)),
    "L" : Node("L", None, [("K", 1), ("Q", 1)], 0, (5, 2)),
    "M" : Node("M", None, [("H", 1), ("N", 1), ("R", 1)], 0, (0, 3)),
    "N" : Node("N", None, [("I", 1), ("M", 1), ("S", 1)], 0, (1, 3)),
    "O" : Node("O", None, [("P", 1), ("U", 1)], 0, (3, 3)),
    "P" : Node("P", None, [("O", 1), ("Q", 1)], 0, (4, 3)),
    "Q" : Node("Q", None, [("L", 1), ("P", 1), ("V", 1)], 0, (5, 3)),
    "R" : Node("R", None, [("M", 1) ,("S", 1)], 0, (0, 4)),
    "S" : Node("S", None, [("N", 1), ("R", 1), ("T", 1)], 0, (1, 4)),
    "T" : Node("T", None, [("S", 1), ("U", 1), ("W", 1)], 0, (2, 4)),
    "U" : Node("U", None, [("O", 1), ("T", 1)], 0, (3, 4)),
    "V" : Node("V", None, [("Q", 1), ("Y", 1)], 0, (5, 4)),
    "W" : Node("W", None, [("T", 1)], 0, (2, 5)),
    "X" : Node("X", None, [("Y", 1)], 0, (4, 5)),
    "Y" : Node("Y", None, [("V", 1), ("X", 1)], 0, (5, 5))
}
```

In [26]:

```
solution = Astar(graph, "A", "Y")
print(solution)
```

```
['A', 'F', 'H', 'M', 'R', 'S', 'T', 'U', 'O', 'P', 'Q', 'V', 'Y']
```

In [1]:

```
citiesGraph = {
    "Marketplatz": Node("Marketplatz", None, [("S", 87)], 0),
    "TrainStation": Node("TrainStation", None, [("KK", 87)], 0),
    "S": Node("S", None, [("Marketplatz", 87), ("St", 142), ("KK", 98)], 0),
    "KK": Node("KK", None, [("TrainStation", 87), ("S", 98), ("KKN", 85)], 0),
    "St": Node("St", None, [("S", 142), ("KKN", 86), ("Dia", 83)], 0),
    "KKN": Node("KKN", None, [("KK", 85), ("St", 86)], 0),
    "Dia": Node("Dia", None, [("St", 83)], 0)
}
solution = Astar(citiesGraph, "Marketplatz", "Dia")
print(solution)
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Cell In[1], line 2
      1 citiesGraph = {
----> 2     "Marketplatz": Node("Marketplatz", None, [("S", 87)], 0),
      3     "TrainStation": Node("TrainStation", None, [("KK", 87)], 0),
      4     "S": Node("S", None, [("Marketplatz", 87), ("St", 142), ("KK", 98)], 0),
      5     "KK": Node("KK", None, [("TrainStation", 87), ("S", 98), ("KKN", 85)], 0),
      6     "St": Node("St", None, [("S", 142), ("KKN", 86), ("Dia", 83)], 0),
      7     "KKN": Node("KKN", None, [("KK", 85), ("St", 86)], 0),
      8     "Dia": Node("Dia", None, [("St", 83)], 0)
      9 }
     10 solution = Astar(citiesGraph, "Marketplatz", "Dia")
     11 print(solution)

NameError: name 'Node' is not defined
```

In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js