**A\* Algorithm**

import heapq

class AStar:

def \_\_init\_\_(self, graph, heuristic\_function):

self.graph = graph

self.heuristic\_function = heuristic\_function

def search(self, start, goal):

frontier = []

heapq.heappush(frontier, (0, start))

came\_from = {start: None}

cost\_so\_far = {start: 0}

while frontier:

\_, current = heapq.heappop(frontier)

if current == goal:

break

for neighbor, edge\_cost in self.graph[current].items():

new\_cost = cost\_so\_far[current] + edge\_cost

if neighbor not in cost\_so\_far or new\_cost < cost\_so\_far[neighbor]:

cost\_so\_far[neighbor] = new\_cost

priority = new\_cost + self.heuristic\_function(neighbor, goal)

heapq.heappush(frontier, (priority, neighbor))

came\_from[neighbor] = current

return came\_from, cost\_so\_far

# Example usage:

graph = {

'A': {'B': 1, 'C': 4},

'B': {'A': 1, 'C': 2, 'D': 5},

'C': {'A': 4, 'B': 2, 'D': 1},

'D': {'B': 5, 'C': 1}

}

def heuristic\_function(node, goal):

return abs(ord(node) - ord(goal))

a\_star = AStar(graph, heuristic\_function)

came\_from, cost\_so\_far = a\_star.search('A', 'D')

path = []

current = 'D'

while current is not None:

path.append(current)

current = came\_from.get(current)

path.reverse()

print(path)

Output:

['A', 'B', 'C', 'D']