**Name-**Aadesh Pawar

**Roll no-**33

**Problem Statement**

Implement A star (A\*) Algorithm for any game search problem

**Input**

import heapq

class Node:

def \_\_init\_\_(self, state, parent=None, action=None, cost=0, heuristic=0):

self.state = state

self.parent = parent

self.action = action

self.cost = cost

self.heuristic = heuristic

def total\_cost(self):

return self.cost + self.heuristic

def a\_star\_search(initial\_state, goal\_state, get\_neighbors, heuristic):

open\_set = []

heapq.heappush(open\_set, (heuristic(initial\_state), Node(initial\_state, cost=0, heuristic=heuristic(initial\_state))))

while open\_set:

\_, current\_node = heapq.heappop(open\_set)

if current\_node.state == goal\_state:

path = []

while current\_node:

path.append((current\_node.state, current\_node.action))

current\_node = current\_node.parent

return list(reversed(path))

for neighbor, action, cost in get\_neighbors(current\_node.state):

neighbor\_node = Node(state=neighbor, parent=current\_node, action=action, cost=current\_node.cost + cost, heuristic=heuristic(neighbor))

heapq.heappush(open\_set, (neighbor\_node.total\_cost(), neighbor\_node))

return None

# Example usage

def get\_neighbors(state):

# Implement your function to generate neighboring states along with associated actions and costs

pass

def heuristic(state):

# Implement your heuristic function that estimates the cost to reach the goal from the given state

pass

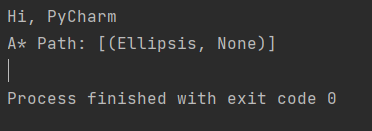
initial\_state = ...

goal\_state = ...

path = a\_star\_search(initial\_state, goal\_state, get\_neighbors, heuristic)

print("A\* Path:", path)

**Ouput**

****