A picture containing text, metalware, gear

Description automatically generated



**LAB: Artificial Intelligence**

**TASK**

|  |  |
| --- | --- |
| **Lab Instructor** | Ms. Saba Aslam |
| **Department** | Computer Science |

**Lab No 5 : Informed Searches**

**Instructions**![Shape

Description automatically generated with medium confidence]()

1. Make a word document paste your code and output there.  2. Comments in the code explaining chunks of the code is important.  3. Plagiarism is strictly prohibited, 0 mark would be given to students who cheat.

**Objectives:**

• In this Lab we will study and implement basic algorithms of Informed  searches

• We will familiarize ourselves with Best First search and A\*

**Lab Tasks:**

Implement A\* on given graph, heuristic values are given in the table.Chart

Description automatically generatedTable

Description automatically generated

**CODE:**

import heapq  
  
  
class Graph:  
  
 def \_\_init\_\_(self):  
 self.edges = None  
  
 def init(self):  
 self.edges = {}  
  
 def neighbours(self, edge\_id):  
 return self.edges[edge\_id]  
  
  
class PriorityQueue:  
  
 def \_\_init\_\_(self):  
 self.elements = []  
  
 def empty(self):  
 return len(self.elements) == 0  
  
 def put(self, item, priority):  
 heapq.heappush(self.elements, (priority, item))  
  
 def get(self):  
 return heapq.heappop(self.elements)[1]  
  
  
def A\_Star\_Search(graph, source, destination, heuristic):  
 minHeap = PriorityQueue()  
 minHeap.put(source, 0)  
 came\_from = {}  
 cost\_so\_far = {}  
 came\_from[source] = None  
 cost\_so\_far[source] = 0  
  
 while not minHeap.empty():  
  
 N = minHeap.get()  
  
 if N == destination:  
 break  
  
 for node in graph.neighbours[N]:  
  
 next\_vertex = node[0]  
 edge\_weight = node[1]  
 new\_cost = cost\_so\_far[N] + edge\_weight  
  
 if next\_vertex not in cost\_so\_far or new\_cost < cost\_so\_far[next\_vertex]:  
 cost\_so\_far[next\_vertex] = new\_cost  
 priority = new\_cost + heuristic[next\_vertex]  
 minHeap.put(next\_vertex, priority)  
 came\_from[next\_vertex] = N  
  
 return came\_from, cost\_so\_far  
  
  
def print\_path(came\_from, Destination):  
 N\_node = Destination  
  
 if came\_from[N\_node] != None:  
 print\_path(came\_from, came\_from[N\_node])  
  
 if N\_node != 'G':  
  
 print(N\_node, "->", end=' ')  
  
 else:  
  
 print(N\_node)  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 inputGraph = Graph()  
  
 inputGraph.neighbours = {  
 'A': [('B', 11), ('D', 7), ('C', 14)],  
 'B': [('A', 11), ('E', 15)],  
 'D': [('A', 7), ('F', 25)],  
 'C': [('A', 14), ('E', 8), ('F', 10)],  
 'E': [('B', 15), ('C', 8), ('H', 9)],  
 'F': [('C', 10), ('D', 25), ('G', 20)],  
 'H': [('E', 9), ('G', 10)],  
 'G': [('H', 10), ('F', 20)],  
 }  
  
 heuristic = {  
 'A': 40,  
 'B': 32,  
 'C': 25,  
 'D': 35,  
 'E': 19,  
 'F': 17,  
 'H': 10,  
 'G': 0,  
 }  
  
 (came\_from, cost\_so\_far) = A\_Star\_Search(inputGraph, 'A', 'G', heuristic)  
  
 print("\nThe optimal cost to reach destination is : ", cost\_so\_far['G'])  
  
 print("\nThe optimal path is : ", end=" ")  
  
 print\_path(came\_from, 'G')

**OUTPUT:**

Text

Description automatically generated