# Bahria University,

## Karachi Campus



LAB EXPERIMENT NO.

**\_04\_**

LIST OF TASKS

|  |  |
| --- | --- |
| **TASK NO** | **OBJECTIVE** |
| 01 | Implement A\* Search and find the optimal path for finding the goal  Start = S  Goal = E |
|  |  |
|  |  |
|  |  |
|  |  |

Submitted On:

Date: 02/04/2024

**Task No 01:** Implement A\* Search and find the optimal path for finding the goal

Start = S and Goal = E

**Solution:**

import heapq

def heuristic(node, goal):

x1, y1 = node

x2, y2 = goal

return abs(x1 - x2) + abs(y1 - y2)

def astar\_search(start, goal, graph):

frontier = []

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

came\_from = {}

cost\_so\_far = {}

came\_from[start] = None

cost\_so\_far[start] = 0

while frontier:

current\_cost, current\_node = heapq.heappop(frontier)

if current\_node == goal:

break

for next\_node in graph[current\_node]:

new\_cost = cost\_so\_far[current\_node] + graph[current\_node][next\_node]

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

cost\_so\_far[next\_node] = new\_cost

priority = new\_cost + heuristic(next\_node, goal)

heapq.heappush(frontier, (priority, next\_node))

came\_from[next\_node] = current\_node

path = []

current\_node = goal

while current\_node != start:

path.append(current\_node)

current\_node = came\_from[current\_node]

path.append(start)

path.reverse()

return path

graph = {

'S': {'A': 1, 'B': 3},

'A': {'C': 3, 'D': 1},

'B': {'D': 3, 'E': 2},

'C': {'E': 3},

'D': {'E': 1},

'E': {}}

start = 'S'

goal = 'E'

optimal\_path = astar\_search(start, goal, graph)

print("Optimal path found:", optimal\_path)