

EXP NO. 4 A*

AIM:- To write the A* program and execute successfully.

PSEUDOCODE:

Function a_star (graph, start, goal, heuristics):

 CREATE an empty priority queue open_set

 ENQUEUE (0, start) INTO open_set

 CREATE g_costs DICTIONARY with each ^{node} ~~other~~ initialized to infinity

 SET g_costs [start] = 0

 CREATE an empty dictionary came_from

 WHILE open_set IS NOT EMPTY:

 DEQUEUE current_node with the lowest priority
 (cost + heuristic) from open_set

 IF current_node == goal:

 CREATE an empty list path

 WHILE current_node IN came_from:

 ADD current_node To path

 SET current_node = came_from[current_node]

 ADD start to path

 REVERSE path

 PRINT path

 RETURN

 FOR each neighbor, cost IN graph [current_node]:

 set ~~g_costs~~ tentative_g_cost = g_costs [current_node] + cost

```

IF tentative_g_cost < g_costs[neighbor]:
    set g_costs[neighbor] = tentative_g_cost
    set f_cost = tentative_g_cost + heuristic[neighbor]
    ENQUEUE (f_cost, neighbor) INTO open_set.
    SET came_from[neighbor] = current_node
print "No path found"

```

FUNCTION create_graph():

```

CREATE an empty dictionary graph
PROMPT "Enter no. of nodes: " AND READ num_nodes
FOR i FROM 1 TO num_nodes:
    PROMPT "Enter the node: " AND READ node
    PROMPT "Enter the neighbors of node i with cost
    (format: neighbor cost): " AND READ neighbors
    SPLIT neighbors INTO a list.
    SET graph[node] = [neighbors[i], int(neighbors[i+1])]
    FOR i FROM 0 to length(neighbors)-1 STEP 2]
RETURN graph.

```

FUNCTION create_heuristics():

```

CREATE an empty dictionary heuristics
PROMPT "Enter the no of nodes with heuristic value: "
AND READ num_nodes

```

FOR i FROM 1 TO num_nodes:

PROMPT "Enter the node: " AND READ node

PROMPT "Enter hvalue of node i: " AND READ h.

Set heuristics[node] = heuristics

Return heuristics

Function main():

SET graph = call create_graph()

SET heuristics = call create_heuristics()

PROMPT "Enter the starting node: " AND READ start_node

PROMPT "Enter the goal node: " AND READ goal_node

IF start_node IN graph AND goal_node IN graph:

PRINT "Searching for path from '{start_node}' to '{goal_node}' using A* algorithm:"

CALL a_star(graph, start_node, goal_node, heuristics)

ELSE:

PRINT "Start or goal node not found in the graph."

CALL main()

PROGRAM:-

from queue import PriorityQueue

def a_star(graph, start, goal, heuristics):

open_set = PriorityQueue()

open_set.put((0, start))

g_costs = {node: float('inf') for node in graph}

g_costs[start] = 0

came_from = {}

while not open_set.empty():

current_cost, current_node = open_set.get()

If current_node == goal:


```
path = []
```

```
while current_node in came_from:
```

```
    path.append(current_node)
```

```
    current_node = came_from[current_node]
```

```
path.append(start)
```

```
path.reverse()
```

```
print ("Path found:", '→'.join(path))
```

```
return
```

```
for neighbor, cost in graph[current_node]:
```

```
    tentative_g_cost = g_costs[current_node] + cost
```

```
    if tentative_g_cost < g_costs[neighbor]:
```

```
        g_costs[neighbor] = tentative_g_cost
```

```
        f_cost f_cost = tentative_g_cost + heuristic[neighbor]
```

```
        open_set.put ((f_cost, neighbor))
```

```
        came_from[neighbor] = current_node
```

```
print ("No path found")
```

```
def create_graph():
```

```
    graph = {}
```

```
    num_nodes = int(input("Enter the no. of nodes: "))
```

```
    for i in range(num_nodes):
```

```
        node = input("Enter node: ")
```

```
        neighbors = input("Enter neighbors of node with cost
```

```
        (format: neighbor cost): ").split()
```

```
        graph[node] = [neighbors[i], int(neighbors[i+1])]
```

```
    for i in range(0, len(neighbors), 2):
```

```
return graph
```

```
def create_heuristics():
```

```
    heuristics = {}
```

```
    num_node = int(input("Enter no of nodes with heuristics:"))
```

```
    for _ in range(num_node):
```

```
        node = input("Enter node: ")
```

```
        heuristic = int(input("Enter the value of heuristic: "))
```

```
        heuristics[node] = heuristic
```

```
    return heuristics
```

```
def main():
```

```
    graph = create_graph()
```

```
    heuristics = create_heuristics()
```

```
    start_node = input("Enter the starting node: ")
```

```
    goal_node = input("Enter the goal node: ")
```

```
    if start_node in graph and goal_node in graph:
```

```
        print(f"Searching for path from '{start_node}' to  
'{goal_node}' using A* algorithm: ")
```

```
        a_star(graph, start_node, goal_node, heuristics)
```

```
    else:
```

```
        print("Start or goal node not found in the graph")
```

```
if __name__ == "__main__":
```

```
    main()
```

OUTPUT:-

Enter the no. of nodes: 4

Enter node: A

Enter the neighbors of A with costs (format: neighbor cost): B 1 C 4

Enter node: B

Enter the neighbors of B with cost (format: neighbor cost):

C 2 D 5

Enter the node: C

Enter the " " : D 1

Enter node: D

Enter " " :

Enter the no. of nodes with hvalue: 4

Enter the node: A

Enter hvalue of A: 7

Enter the node: B

" " B: 6

Enter node: C

Enter " " C: 2

Enter the node: D

Enter the hvalue of D: 0

Enter the ~~starting~~ node: A

Enter the ~~goal~~ node: D

Searching for path from 'A' to 'D' using A* algorithm:

Path found: $A \rightarrow B \rightarrow C \rightarrow D$

RESULT:-

Therefore the program is successfully executed and output is verified.