* **A\* Search and Recursive Best-First Search:**

**Code:-**

import heapq

def astar(graph, start, goal):

open\_set = []

heapq.heappush(open\_set, (0, start))

came\_from = {}

while open\_set:

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

if current\_node == goal:

path = reconstruct\_path(came\_from, start, goal)

print("Path found:", ' -> '.join(path))

return

for neighbor in graph[current\_node]:

if neighbor not in came\_from:

heapq.heappush(open\_set, (0, neighbor))

came\_from[neighbor] = current\_node

print("No path found")

def reconstruct\_path(came\_from, start, goal):

current = goal

path = [current]

while current != start:

current = came\_from[current]

path.append(current)

return path[::-1]

city\_graph = {

'Delhi': {'Mumbai', 'Jaipur', 'Lucknow'},

'Mumbai': {'Delhi', 'Chennai', 'Bangalore'},

'Chennai': {'Mumbai', 'Bangalore'},

'Bangalore': {'Mumbai', 'Chennai', 'Hyderabad'},

'Hyderabad': {'Bangalore', 'Chennai'},

'Jaipur': {'Delhi', 'Lucknow'},

'Lucknow': {'Delhi', 'Jaipur'}

}

start\_city = input("Enter the start city in India: ")

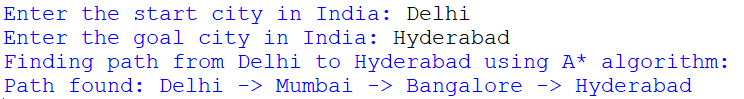
goal\_city = input("Enter the goal city in India: ")

print(f"Finding path from {start\_city} to {goal\_city} using A\* algorithm:")

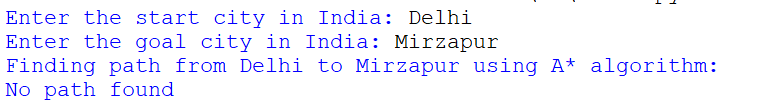
astar(city\_graph, start\_city, goal\_city)

**Output:-**

**Path found:-**



**Path not Found;-**

****