|  |  |
| --- | --- |
| **Name** | Manish Shashikant Jadhav |
| **UID no.** | 2023301005 |

|  |  |
| --- | --- |
| **Experiment 3** | |
| **AIM :** | Implement a the Road Map problem using the Informed searching technique using A\* search. The start node is Arad and Goal Node is Bucharest.  Analyze the algorithm with respect to Completeness, Optimality, time and space Complexity. |
| **CODE:** | *import* heapq  class Node:      def \_\_init\_\_(*self*, *state*, *parent*=None, *g*=0, *h*=0):  *self*.state = *state*  *self*.parent = *parent*  *self*.g = *g*  *self*.h = *h*  *self*.f = *g* + *h*      def \_\_lt\_\_(*self*, *other*):  *return* *self*.f < *other*.f  def astar\_search(*graph*, *start*, *goal*, *heuristic*):      start\_node = Node(*start*, None, 0, *heuristic*[*start*])      frontier = []      heapq.heappush(frontier, start\_node)      explored = set()      step = 1  *while* frontier:          current\_node = heapq.heappop(frontier)          print(f"\nStep {step}:")          print(f"Current node: {current\_node.state}")          print(f"f(n) = {current\_node.f}, g(n) = {current\_node.g}, h(n) = {current\_node.h}")  *if* current\_node.state == *goal*:              path = []  *while* current\_node:                  path.append(current\_node.state)                  current\_node = current\_node.parent  *return* list(reversed(path))          explored.add(current\_node.state)          print(f"Explored set: {explored}")  *for* neighbor, cost *in* *graph*[current\_node.state].items():  *if* neighbor not in explored:                  g = current\_node.g + cost                  h = *heuristic*[neighbor]                  new\_node = Node(neighbor, current\_node, g, h)  *if* new\_node not in frontier:                      heapq.heappush(frontier, new\_node)                      print(f"Added to frontier: {neighbor} (f={new\_node.f}, g={g}, h={h})")  *else*:  *for* i, node *in* enumerate(frontier):  *if* node.state == neighbor and node.g > g:                              frontier[i] = new\_node                              heapq.heapify(frontier)                              print(f"Updated in frontier: {neighbor} (f={new\_node.f}, g={g}, h={h})")  *break*          print("Frontier:", [(node.state, node.f) *for* node *in* frontier])          step += 1  *return* None  *# Romania map values*  romania\_map = {      'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},      'Zerind': {'Arad': 75, 'Oradea': 71},      'Oradea': {'Zerind': 71, 'Sibiu': 151},      'Sibiu': {'Arad': 140, 'Oradea': 151, 'Fagaras': 99, 'Rimnicu Vilcea': 80},      'Timisoara': {'Arad': 118, 'Lugoj': 111},      'Lugoj': {'Timisoara': 111, 'Mehadia': 70},      'Mehadia': {'Lugoj': 70, 'Drobeta': 75},      'Drobeta': {'Mehadia': 75, 'Craiova': 120},      'Craiova': {'Drobeta': 120, 'Rimnicu Vilcea': 146, 'Pitesti': 138},      'Rimnicu Vilcea': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},      'Fagaras': {'Sibiu': 99, 'Bucharest': 211},      'Pitesti': {'Rimnicu Vilcea': 97, 'Craiova': 138, 'Bucharest': 101},      'Bucharest': {'Fagaras': 211, 'Pitesti': 101, 'Giurgiu': 90, 'Urziceni': 85},      'Giurgiu': {'Bucharest': 90},      'Urziceni': {'Bucharest': 85, 'Vaslui': 142, 'Hirsova': 98},      'Hirsova': {'Urziceni': 98, 'Eforie': 86},      'Eforie': {'Hirsova': 86},      'Vaslui': {'Urziceni': 142, 'Iasi': 92},      'Iasi': {'Vaslui': 92, 'Neamt': 87},      'Neamt': {'Iasi': 87}  }  *# heuristic values*  heuristic = {      'Arad': 366, 'Bucharest': 0, 'Craiova': 160, 'Drobeta': 242, 'Eforie': 161,      'Fagaras': 176, 'Giurgiu': 77, 'Hirsova': 151, 'Iasi': 226, 'Lugoj': 244,      'Mehadia': 241, 'Neamt': 234, 'Oradea': 380, 'Pitesti': 100, 'Rimnicu Vilcea': 193,      'Sibiu': 253, 'Timisoara': 329, 'Urziceni': 80, 'Vaslui': 199, 'Zerind': 374  }  *#Taking user input for start and goal cities*  print("Available cities:", ', '.join(romania\_map.keys()))  start = input("Enter the start city: ")  goal = input("Enter the goal city: ")  *if* start not in romania\_map or goal not in romania\_map:      print("Invalid start or goal city. Please choose from the available cities.")  *else*:      path = astar\_search(romania\_map, start, goal, heuristic)  *if* path:          print(f"\nPath found: {' -> '.join(path)}")          print(f"Total cost: {sum(romania\_map[path[i]][path[i+1]] *for* i *in* range(len(path)-1))}")  *else*:          print("No path found") |
| **OUTPUT:** |  |
| **Analysis of Algorithm** | 1. **Completeness:** A\* search is complete, meaning it will always find a solution if one exists, provided that: 2. The branching factor is finite 3. All edge costs are positive 4. The heuristic function is admissible (never overestimates the cost to the goal) 5. **Optimality:** A\* search is optimal if the heuristic function is admissible and consistent (monotonic). This means it will always find the least-cost path to the goal if such a path exists. For the Romania map problem, if the heuristic values provided are admissible and consistent, the algorithm will find the optimal path from Arad to Bucharest. 6. **Time Complexity:** The time complexity of A\* search depends on the heuristic function. In the worst case, when the heuristic is poor, the time complexity can be exponential, O(b^d), where b is the branching factor and d is the depth of the solution. For the Romania map problem, the branching factor is relatively small, and with a good heuristic (like straight-line distance to Bucharest), the algorithm should perform well, exploring significantly fewer nodes than uninformed search methods. 7. **Space Complexity:** The space complexity of A\* search is O(b^d), as it needs to store all generated nodes in memory. This is because A\* keeps track of the frontier (nodes to be explored) and the explored set.   For the Romania map problem, the space requirements are manageable due to the limited number of cities. However, for larger graphs, space can become a limiting factor. |
| **CONCLUSION:** | Hence by completing this experiment I came to know about Implementation a the Road Map problem using the Informed searching technique using A\* search. |