Code: (A*)

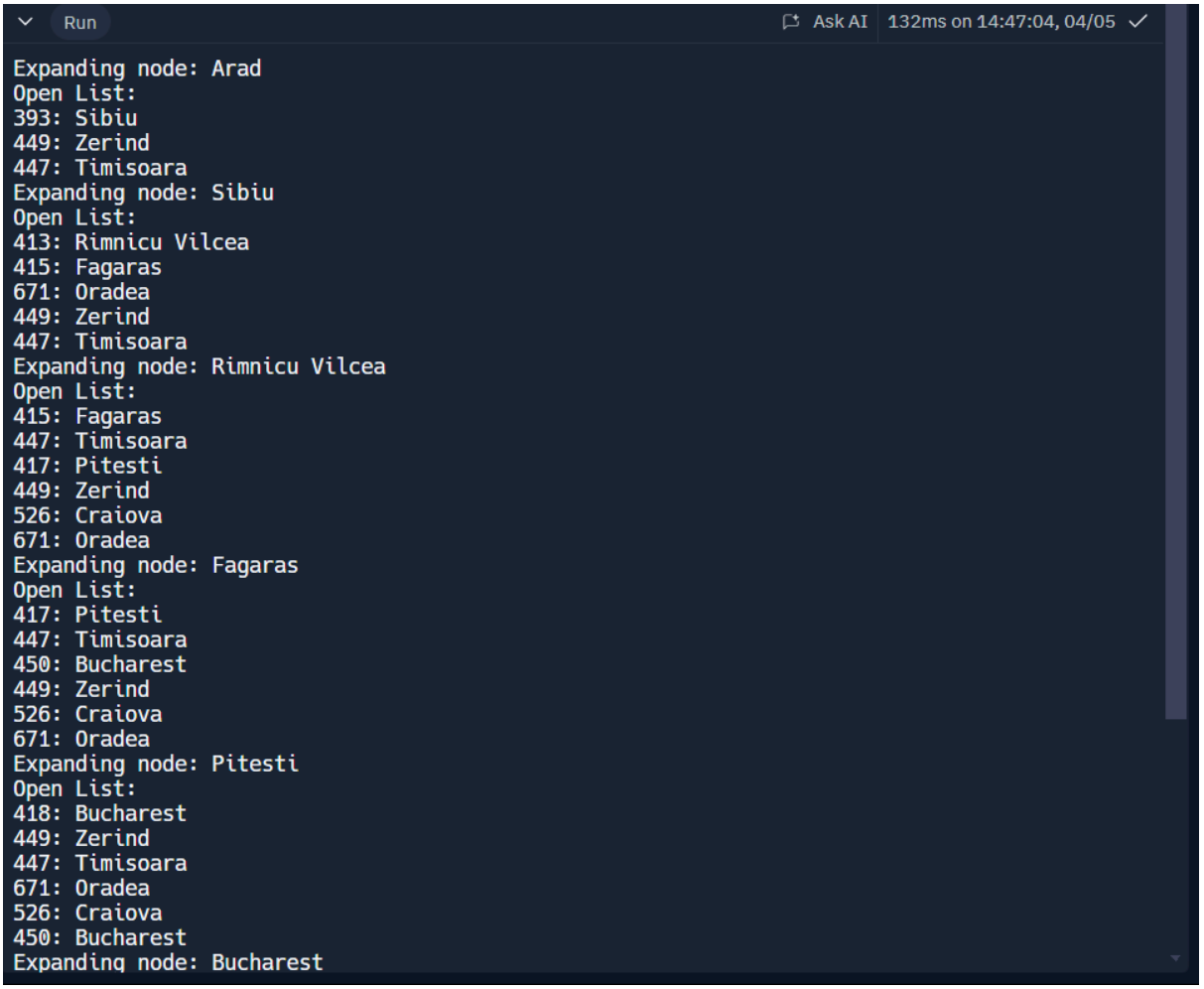
```
import heapq

romania_graph = {
    '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,
```

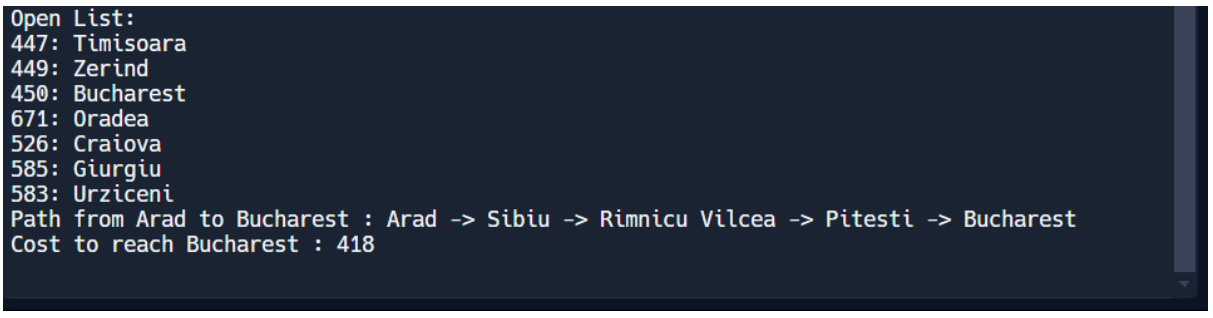
```
        'Pitesti': 138
    },
    'Rimnicu Vilcea': {
        'Sibiu': 80,
        'Craiova': 146,
        'Pitesti': 97
    },
    'Fagaras': {
        'Sibiu': 99,
        'Bucharest': 211
    },
    'Pitesti': {
        'Rimnicu Vilcea': 97,
        'Craiova': 138,
        'Bucharest': 101
    },
    '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 = {
    '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
}


def a_star(graph, start, goal):
  open_nodes = []
  closed_nodes = set()

  heapq.heappush(open_nodes, (0 + heuristic[start], 0, start, []))

  while open_nodes:

    f, g, current_node, path = heapq.heappop(open_nodes)

    print("Expanding node:", current_node)
    for neighbor, cost in graph[current_node].items():
```

```
      neighbor_g = g + cost
      neighbor_f = neighbor_g + heuristic[neighbor]

      if neighbor in closed_nodes:
        continue

      neighbor_in_open_list = False
      for i, (f_val, g_val, node, _) in enumerate(open_nodes):
        if node == neighbor:
          neighbor_in_open_list = True
          break

      if neighbor_in_open_list and neighbor_g >= g_val:
        continue

      heapq.heappush(open_nodes,
                     (neighbor_f, neighbor_g, neighbor, path +
[current_node]))

    closed_nodes.add(current_node)

    print("Open List:")
    for f_val, _, node, _ in open_nodes:
      print(f"{f_val}: {node}")

    if current_node == goal:
      return path + [current_node], g
  return None, None


start_city = 'Arad'
goal_city = 'Bucharest'
path, cost = a_star(romania_graph, start_city, goal_city)
if path:
  print("Path from", start_city, "to", goal_city, ":", " ->
".join(path))
  print("Cost to reach", goal_city, ":", cost)
else:
  print("No path found from", start_city, "to", goal_city)
```

Output:


Run
Ask AI
132ms on 14:47:04, 04/05
Expanding node: Arad
Open List:
393: Sibiu
449: Zerind
447: Timisoara
Expanding node: Sibiu
Open List:
413: Rimnicu Vilcea
415: Fagaras
671: Oradea
449: Zerind
447: Timisoara
Expanding node: Rimnicu Vilcea
Open List:
415: Fagaras
447: Timisoara
417: Pitesti
449: Zerind
526: Craiova
671: Oradea
Expanding node: Fagaras
Open List:
417: Pitesti
447: Timisoara
450: Bucharest
449: Zerind
526: Craiova
671: Oradea
Expanding node: Pitesti
Open List:
418: Bucharest
449: Zerind
447: Timisoara
671: Oradea
526: Craiova
450: Bucharest
Expanding node: Bucharest


Open List:
447: Timisoara
449: Zerind
450: Bucharest
671: Oradea
526: Craiova
585: Giurgiu
583: Urziceni
Path from Arad to Bucharest : Arad -> Sibiu -> Rimnicu Vilcea -> Pitesti -> Bucharest
Cost to reach Bucharest : 418

Code:

```
import heapq

def gbfs(graph, start, goal, heuristic):
  visited = set()
  priority_queue = [(heuristic[start], start)]
  path = {start: None}

  while priority_queue:
    _, current_node = heapq.heappop(priority_queue)
    if current_node == goal:
      return construct_path(path, start, goal)
    visited.add(current_node)

    for neighbor in graph[current_node]:
      if neighbor not in visited:
        heapq.heappush(priority_queue, (heuristic[neighbor], neighbor))
        path[neighbor] = current_node

  return None


def construct_path(path, start, goal):
  current_node = goal
  path_sequence = []

  while current_node:
    path_sequence.insert(0, current_node)
    current_node = path[current_node]

  return path_sequence


graph = {
    'A': ['S', 'T', 'Z'],
    'S': ['A', 'F', 'O', 'R'],
    'T': [],
    'Z': [],
    'F': ['S', 'B']
}

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

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

heuristic = {
    'A': 366,
    'S': 253,
    'F': 176,
    'T': 329,
    'O': 380,
    'Z': 374,
    'R': 193,
    'B': 0
}

gbfs_path = gbfs(graph, start_node, goal_node, heuristic)

if gbfs_path:
  print('Path:', gbfs_path)
  print(f"Goal '{goal_node}' found using GBFS.")
else:
  print(f"Goal '{goal_node}' not found using GBFS.")
```

Output: