Batch: B - 1 Roll No.: 16014022050

Experiment No.: 4

Title: Implementation of informed search algorithm (Greedy Best First search/A*)

Course Outcome: Implementation of informed search algorithm (Greedy Best First search/A*)

CO2: Describe & implement AI Search and optimization techniques.

Books/ Journals/ Websites referred:

1. **"Artificial Intelligence: a Modern Approach" by Russell and Norving, Pearson education Publications**
2. **"Artificial Intelligence" By Rich and knight, Tata McGraw Hill Publications**

Introduction:

In Artificial Intelligence (AI), search algorithms are used to find paths or solutions in a problem space. Unlike uninformed search (like BFS, DFS, UCS), which only uses the structure of the graph, informed search algorithms make use of heuristics (extra knowledge or estimates about the problem domain) to guide the search process more efficiently.
Two commonly used informed search strategies are:

Greedy Best-First Search (GBFS)

- Definition:
  Greedy Best-First Search is an informed search algorithm that selects the next node to expand based only on the heuristic function h(n) (an estimate of the cost from the current node to the goal).
- Working Principle:
  At each step, the algorithm chooses the node that appears closest to the goal according to the heuristic value, without considering the path cost so far.
- Advantages:
  - Faster than uninformed search.
  - Requires less memory in some cases.
  - Often finds a solution quickly.
- Disadvantages:
  - Not guaranteed to find the optimal (cheapest) solution.
  - Can get stuck in long or costly paths if the heuristic is misleading.

A* Search Algorithm

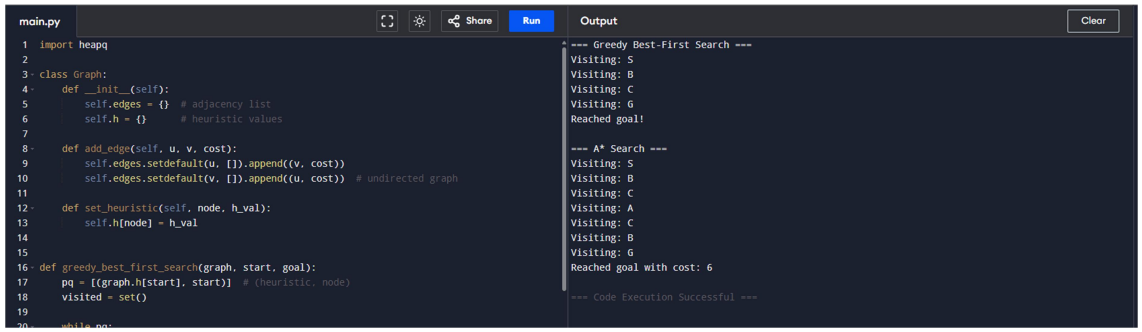
- Definition:
  A* is an informed search algorithm that uses both the actual cost so far g(n) and the heuristic estimate h(n) to select which node to expand. It evaluates nodes using the function:
  f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)
  where:
  - g(n) = cost from the start node to n
  - h(n) = estimated cost from n to the goal
- Working Principle:
  A* expands nodes in the order of increasing f(n). This ensures that both the distance already traveled and the estimated distance to the goal are considered.
- Advantages:
  - Guaranteed to find the optimal solution if the heuristic is admissible (never overestimates).
  - Efficiently balances between exploring the cheapest path so far and moving toward the goal.
- Disadvantages:
  - Requires more memory than Greedy search.
  - Performance depends on the quality of the heuristic.

Implementation:

```
import heapq

class Graph:
    def __init__(self):
        self.edges = {}  # adjacency list
        self.h = {}      # heuristic values

    def add_edge(self, u, v, cost):
        self.edges.setdefault(u, []).append((v, cost))
        self.edges.setdefault(v, []).append((u, cost))  # undirected graph

    def set_heuristic(self, node, h_val):
        self.h[node] = h_val


def greedy_best_first_search(graph, start, goal):
    pq = [(graph.h[start], start)]  # (heuristic, node)
    visited = set()

    while pq:
        _, current = heapq.heappop(pq)
        print("Visiting:", current)
```

```
        if current == goal:
            print("Reached goal!")
            return

        visited.add(current)
        for neighbor, cost in graph.edges.get(current, []):
            if neighbor not in visited:
                heapq.heappush(pq, (graph.h[neighbor], neighbor))


def a_star_search(graph, start, goal):
    pq = [(graph.h[start], 0, start)]  # (f = g+h, g, node)
    visited = {}

    while pq:
        f, g, current = heapq.heappop(pq)
        print("Visiting:", current)

        if current == goal:
            print("Reached goal with cost:", g)
            return

        if current in visited and visited[current] <= g:
            continue
        visited[current] = g

        for neighbor, cost in graph.edges.get(current, []):
            new_g = g + cost
            f_val = new_g + graph.h[neighbor]
            heapq.heappush(pq, (f_val, new_g, neighbor))


# Example usage
g = Graph()
g.add_edge("S", "A", 1)
g.add_edge("S", "B", 4)
g.add_edge("A", "C", 2)
g.add_edge("B", "C", 1)
g.add_edge("C", "G", 3)

# heuristic values (straight-line distance estimates)
g.set_heuristic("S", 7)
g.set_heuristic("A", 6)
g.set_heuristic("B", 2)
g.set_heuristic("C", 1)
g.set_heuristic("G", 0)

print("=== Greedy Best-First Search ===")
```

```
greedy_best_first_search(g, "S", "G")

print("\n=== A* Search ===")
a_star_search(g, "S", "G")
```




```
=== Greedy Best-First Search ===
Visiting: S
Visiting: B
Visiting: C
Visiting: G
Reached goal!

=== A* Search ===
Visiting: S
Visiting: B
Visiting: C
Visiting: A
Visiting: C
Visiting: B
Visiting: G
Reached goal with cost: 6

=== Code Execution Successful ===
```

Solving

Greedy Best-First Search (GBFS)

Rule: At each step, pick the node with the smallest heuristic h(n).

Step-by-step:

Start at S (h=7).

Neighbors:

A (h=6)

B (h=2)

Pick B (smallest h).
At B (h=2).
Neighbors:
S (visited)
C (h=1)

Pick C.
At C (h=1).
Neighbors:
A (h=6, visited via S)
B (h=2, visited)
G (h=0)

Pick G.

Path (by heuristic only):
S → B → C → G

Cost = 4 (S→B) + 1 (B→C) + 3 (C→G) = 8
(but GBFS never checked cost — it only followed heuristics).

A* Search
Rule: At each step, compute f(n) = g(n) + h(n)
g(n) = cost so far from start
h(n) = heuristic

Pick node with smallest f(n).

Step-by-step:

Start at S
g(S) = 0
f(S) = 0 + 7 = 7
Neighbors:
A: g=1 → f=1+6=7
B: g=4 → f=4+2=6

Pick B (f=6 is smaller).
At B
g(B)=4, f(B)=6
Neighbors:

C: g=5 → f=5+1=6

Pick C.
At C
g(C)=5, f(C)=6
Neighbors:
G: g=8 → f=8+0=8

Pick G.
At G
g(G)=8, f(G)=8

Goal reached!

Path: S → B → C → G
Cost = 8 (this matches the real shortest path cost).

Post Lab Descriptive Questions:

1. **How does Greedy Best-First Search differ from A* in terms of the cost function used?**
   Greedy Best-First Search:
   Uses only the heuristic function h(n) to guide the search.
   - Cost function:
     f(n)=h(n)f(n) = h(n)f(n)=h(n)
     It ignores the actual path cost so far.

   A*:
   Uses both the actual cost g(n) and the heuristic estimate h(n).
   - Cost function:
     f(n)=g(n)+h(n)f(n) = g(n) + h(n)f(n)=g(n)+h(n)
     This ensures both path cost and goal estimate are considered, guaranteeing optimality (if h is admissible).

2. **Suggest a real-world application where A* would be preferred over Greedy Best-**
   First Sea Application: GPS Navigation Systems (like Google Maps).
   A GPS system must not only guide you *towards* the destination but also ensure the shortest/fastest route is chosen.
   - Greedy Best-First might send you on a seemingly short path that actually takes much longer.
   - A* guarantees the optimal path by considering both travel distance so far and the estimated remaining distance.

3. **What is the role of the heuristic $h(n)$ in A* search?**
   - The heuristic h(n) provides an estimate of the cost from node n to the goal.
   - It helps guide the search in the right direction and reduce unnecessary exploration.
   - If the heuristic is admissible (never overestimates the true cost), A* is guaranteed to find the optimal solution.
   - In practice, h(n) acts like a GPS compass → it points toward the goal, making the search efficient