Assignment 8

Name : Sanket Kele- SE-B-B2-34

Title : A* Algorithm using Graph Search with Python GUI

---

1. Aim

To implement the A* (A-star) algorithm for solving AI search problems using the Graph Search method with a Python-based GUI visualization.

---

2. Objectives

- To understand the working of heuristic search in AI.
- To explore the use of the A* algorithm in pathfinding and graph traversal problems.
- To analyze the efficiency of informed search strategies compared to uninformed search.
- To visualize the step-by-step working of A* using a graphical interface.

---

3. Theory

Artificial Intelligence (AI) employs search algorithms to find solutions efficiently. The *A (A-star) algorithm** is an **informed search** technique that combines the strengths of **Uniform Cost Search (UCS)** and **Greedy Best-First Search**.

The evaluation function is defined as:

f(n)=g(n)+h(n)f(n) = g(n) + h(n)

Where:

- **g(n):** Actual cost from the start node to the current node.
- **h(n):** Estimated heuristic cost from the current node to the goal.
- **f(n):** Estimated total cost of the path through the current node.

Applications of A*:

- Pathfinding in maps (e.g., GPS navigation).
- Game AI (shortest pathfinding for NPCs).
- Robotics (autonomous navigation).

---

4. Algorithm (Steps of A*)

1. Initialize the **open list** with the start node.
2. Initialize the **closed list** as empty.
3. Repeat until the goal is found or the open list is empty:
    - Select the node with the **lowest f(n)** from the open list.
    - If this node is the **goal**, return success (trace back the path).
    - Otherwise, expand the node:
        - For each successor, calculate **g(n), h(n), f(n)**.
        - If the successor is not in the open/closed lists, add it to the open list.
        - If it is already present with a higher cost, update its values.
    - Move the expanded node to the **closed list**.
4. If the open list becomes empty and the goal is not found → return failure.

5. Python Implementation with GUI

```
import tkinter as tk
from tkinter import messagebox
import heapq

# --- A* Algorithm Implementation ---
def a_star_steps(graph, start, goal, heuristics):
    frontier = [(heuristics[start], 0, start, [start])]
    explored = set()
    steps = []

    while frontier:
        f, g, node, path = heapq.heappop(frontier)
        steps.append(("expand", node, list(path), g))

        if node == goal:
            steps.append(("goal", node, path, g))
            return path, g, steps

        if node in explored:
            continue
        explored.add(node)

        for neighbor, cost in graph[node].items():
            if neighbor not in explored:
                g_new = g + cost
                f_new = g_new + heuristics[neighbor]
                heapq.heappush(frontier, (f_new, g_new, neighbor, path + [neighbor]))
                steps.append(("discover", neighbor, path + [neighbor], g_new))

    return None, float("inf"), steps


# --- GUI Class ---
class AStarGUI:
    def __init__(self, root):
        self.root = root
        self.root.title("A* Search Visualization (Home → University)")
        self.root.geometry("950x700")
        self.root.configure(bg="white")
```

```
# --- Graph with given distances ---
self.graph = {
    'Home': {'School': 50, 'Garden': 40, 'Bank': 45},
    'School': {'Home': 50, 'Post Office': 59, 'Railway Station': 75},
    'Garden': {'Home': 40, 'Railway Station': 72},
    'Bank': {'Home': 45, 'Police Station': 60},
    'Police Station': {'Bank': 60, 'University': 28},
    'Post Office': {'School': 59},
    'Railway Station': {'School': 75, 'Garden': 72, 'University': 40},
    'University': {'Police Station': 28, 'Railway Station': 40}
}

# --- Tuned heuristic values ---
self.heuristics = {
    'Home': 150,
    'School': 110,
    'Garden': 100,
    'Bank': 85,
    'Police Station': 25,
    'Post Office': 160,
    'Railway Station': 35,
    'University': 0
}

# --- Node positions for GUI layout ---
self.positions = {
    'Home': (100, 250),
    'School': (250, 120),
    'Garden': (250, 380),
    'Bank': (250, 250),
    'Police Station': (450, 250),
    'Post Office': (450, 100),
    'Railway Station': (450, 400),
    'University': (700, 250),
}

self.start = "Home"
self.goal = "University"
self.current_step_index = -1
self.steps = []
self.final_path = None
self.final_cost = None

# --- UI Setup ---
```

```
        self.canvas = tk.Canvas(root, width=900, height=500, bg="white")
        self.canvas.pack(pady=10)

        btn_frame = tk.Frame(root, bg="white")
        btn_frame.pack(pady=5)

        self.prev_btn = tk.Button(btn_frame, text="Previous", command=self.prev_step, width=12)
        self.prev_btn.grid(row=0, column=0, padx=5)

        self.next_btn = tk.Button(btn_frame, text="Next", command=self.next_step, width=12)
        self.next_btn.grid(row=0, column=1, padx=5)

        self.reset_btn = tk.Button(btn_frame, text="Reset", command=self.reset, width=12)
        self.reset_btn.grid(row=0, column=2, padx=5)

        # Progress log at bottom
        self.progress = tk.Text(root, height=8, width=100, bg="white", fg="black", font=("Courier",
10))
        self.progress.pack(pady=10)

        # Run algorithm
        self.final_path, self.final_cost, self.steps = a_star_steps(self.graph, self.start, self.goal,
self.heuristics)
        self.update_canvas()

    def update_canvas(self):
        self.canvas.delete("all")

        # Draw edges
        for node, neighbors in self.graph.items():
            x1, y1 = self.positions[node]
            for neighbor, cost in neighbors.items():
                x2, y2 = self.positions[neighbor]
                self.canvas.create_line(x1, y1, x2, y2, fill="gray", width=2)
                mid_x, mid_y = (x1 + x2) // 2, (y1 + y2) // 2
                self.canvas.create_text(mid_x, mid_y, text=str(cost), fill="blue")

        # Draw nodes
        for node, (x, y) in self.positions.items():
            color = "lightgray"
            if node == self.start:
                color = "lightgreen"
            elif node == self.goal:
                color = "lightcoral"
```

```
            self.canvas.create_oval(x-25, y-25, x+25, y+25, fill=color, outline="black")
            self.canvas.create_text(x, y, text=node, font=("Arial", 10, "bold"))

        # Highlight explored path so far
        if 0 <= self.current_step_index < len(self.steps):
            step_type, node, path, cost = self.steps[self.current_step_index]
            for i in range(len(path) - 1):
                x1, y1 = self.positions[path[i]]
                x2, y2 = self.positions[path[i+1]]
                self.canvas.create_line(x1, y1, x2, y2, fill="orange", width=4)

        # Highlight final path
        if self.final_path and self.current_step_index == len(self.steps) - 1:
            for i in range(len(self.final_path) - 1):
                x1, y1 = self.positions[self.final_path[i]]
                x2, y2 = self.positions[self.final_path[i+1]]
                self.canvas.create_line(x1, y1, x2, y2, fill="green", width=5)

    def next_step(self):
        if self.current_step_index < len(self.steps) - 1:
            self.current_step_index += 1
            self.update_canvas()
            self.update_progress()

    def prev_step(self):
        if self.current_step_index > -1:
            self.current_step_index -= 1
            self.update_canvas()
            self.update_progress()

    def reset(self):
        self.current_step_index = -1
        self.update_canvas()
        self.progress.delete(1.0, tk.END)

    def update_progress(self):
        self.progress.delete(1.0, tk.END)
        if 0 <= self.current_step_index < len(self.steps):
            step_type, node, path, cost = self.steps[self.current_step_index]
            self.progress.insert(tk.END, f"Step {self.current_step_index+1}: {step_type.upper()}
{node}\n")
            self.progress.insert(tk.END, f"Current Path: {' -> '.join(path)}\n")
            self.progress.insert(tk.END, f"Current Cost: {cost}\n")
```

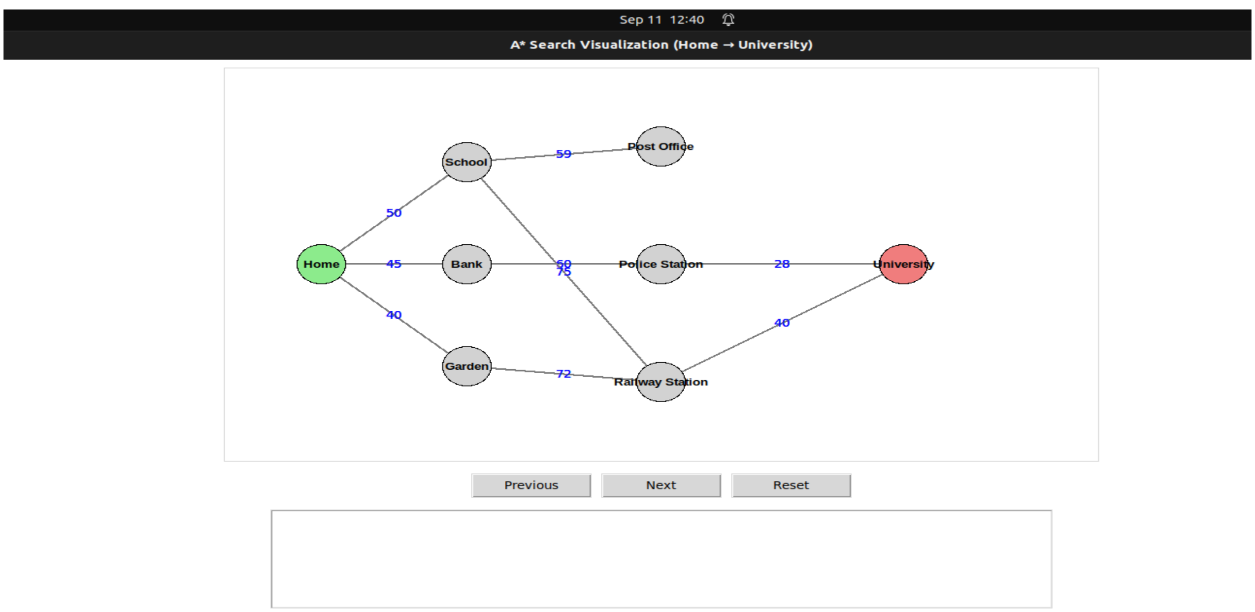
```
        if self.final_path and self.current_step_index == len(self.steps) - 1:
            self.progress.insert(tk.END, f"\nOptimal Path: {' -> '.join(self.final_path)}\n")
            self.progress.insert(tk.END, f"Total Cost: {self.final_cost}\n")


# --- Run Program ---
if __name__ == "__main__":
    root = tk.Tk()
    app = AStarGUI(root)
    root.mainloop()
```

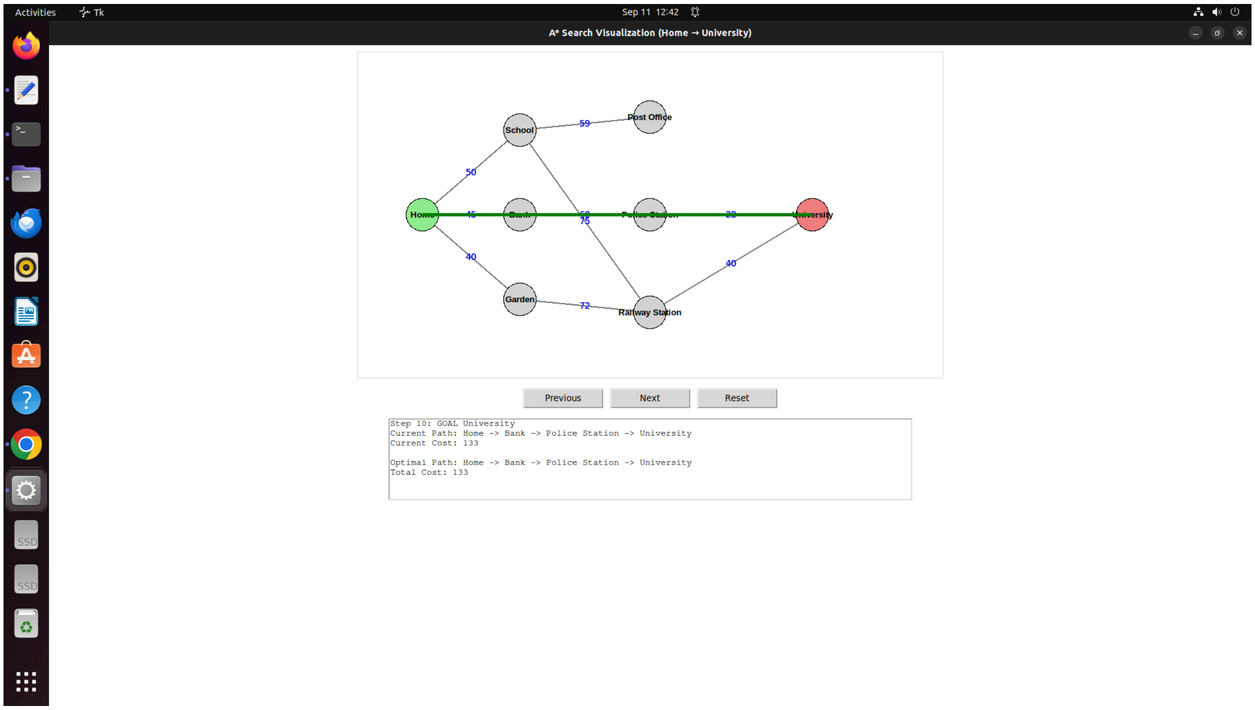
---

6. Output Screenshots

Screenshot 1: Initial Graph Interface



Screenshot 2: Pathfinding Progress & Final Result



---

7. Conclusion

In this assignment, the A* algorithm was successfully implemented using the **Graph Search method** in Python with a **Tkinter GUI**.

- The algorithm efficiently found the shortest path from the start node to the goal node.
- The GUI visualization helped to clearly understand how nodes are expanded and how the algorithm progresses step by step.
- The progress panel displayed the current and optimal paths, making the process transparent and easy to follow.

Thus, the experiment demonstrated how **informed search strategies like A*** outperform uninformed ones by reducing the search space and providing an optimal solution efficiently.
