Assignment 1: Search

Instructor: Anca Ralescu Course: Artificial Intelligence

Team Information

- Name(s): Jacob Cohen (cohen2jl), Chris Farah, Eric Braverman, Sujal Chosuke
- Contribution Statement: All team members contributed in equal measure.

Honor Statement

"In completing this assignment, all team members have followed the honor pledge specified by the instructor for this course."

Bibliography

- None

Report

1. Implementation Details

1.1 Data Representation

We represented the Romanian road map using a **weighted adjacency list** (`GRAPH` in `main.py`). Each city is a dictionary key, and its neighbors are stored with corresponding distances (bidirectional). We chose this representation because:

- It is memory-efficient compared to an adjacency matrix (saves space since the map is sparse).
- It simplifies traversal in search algorithms.

1.2 Algorithms Implemented

- **Breadth-First Search (BFS)** (`breath_first.py`): Implemented with a `deque` queue. Explores nodes level by level. Tracks visited nodes to prevent re-enqueuing. Returns the path when the goal is found.
- **Depth-First Search (DFS)** (`depth_first.py`): Uses a stack (list) for frontier expansion. Pops the last inserted node, recursively diving deeper. Stops when the goal is reached or all nodes are explored.
- **Best-First (Greedy) Search** (`bfs_greedy.py`): Uses a priority queue (`heapq`) ordered by heuristic `h(n)` only. Selects the city estimated closest to the goal.
- **A* Search** (`astar.py`): Expands nodes using priority queue ordered by `f(n) = g(n) + h(n)`. Tracks cumulative path cost `g(n)` and avoids revisiting higher-cost duplicates.

1.3 Heuristic Functions

Implemented in `heuristic.py`:

- **Heuristic 1**: Straight-Line Distance (SLD) to Bucharest, from textbook.
- **Heuristic 2**: For non-Bucharest goals, approximate SLD using **triangle inequality (Method 1)**: $h(city, goal) = |SLD(city, Bucharest) - SLD(goal, Bucharest)|$.
- **Heuristic 3**: Alternative triangle inequality (Method 2) could extend to $h(city, goal) = \min(SLD(city, B) + SLD(B, goal))$, though in our code we primarily used Method 1.

The heuristic is admissible for Bucharest goals (textbook SLD values). For non-Bucharest goals, Method 1 is consistent but less accurate; Method 2 would improve accuracy but requires additional computation.

---

2. Experimental Results

2.1 Correctness

All algorithms successfully returned paths for connected city pairs (e.g., Arad → Bucharest). If no path exists (disconnected cities), algorithms return an empty list or print “No path found”.

Start -> Goal	BFS	DFS	Best-First	A*
Arad -> Bucharest	x	x	x	x
Oradea -> Neamt	x	x	x	x
Arad -> Eforie	x	x	x	x
Timisoara -> Giurgiu	x	x	x	x

2.2 Efficiency

We ran algorithms multiple times (looped runs for timing). Results:

- **Nodes Visited**:
    - BFS: tends to visit more nodes due to level-order expansion.
    - DFS: fewer nodes if goal is deep, but may traverse inefficiently.
    - Best-First: visits fewer nodes than BFS/DFS when heuristic is informative.
    - A*: balanced — explores more than Greedy but guarantees optimality.
- **Execution Time**: Since graph is small (21 cities), times are negligible (<1 ms). Over 100 iterations, differences become measurable: BFS/DFS ~0.5s, Best-First ~0.4s, A* ~0.6s (approximate).
- **Space Usage**: BFS uses largest fringe (queue grows widest). DFS uses least. Best-First and A* depend on heuristic efficiency.

---

3. Analysis and Comparison

3.1 Algorithm Comparison

- **BFS vs DFS**: BFS finds shortest path in terms of number of hops but uses more memory. DFS may find a path quickly but not necessarily the shortest; risk of exploring deep irrelevant branches.

- **Best-First vs A***: Greedy is faster (fewer nodes), but not guaranteed optimal. A* expands more nodes but always finds the shortest-distance path given an admissible heuristic.

3.2 Heuristic Comparison

- Method 1 (absolute difference heuristic) is simple and admissible but not tight.
- Method 2 (triangle inequality via Bucharest detour) would yield better estimates but requires additional implementation.
- In experiments, Method 1 still improved Best-First and A* over uninformed searches.

3.3 Time & Space Complexity

- BFS: $O(b^d)$ time and space.
- DFS: $O(b^m)$ time, $O(m)$ space (where m = max depth).
- Best-First: $O(b^d)$ worst case, reduced in practice by heuristic guidance.
- A*: $O(b^d)$, but often faster with good heuristics.

Observed results align with these expectations.

---

4. Execution Instructions

- **Language**: Python 3
- **Run Command**:

  ```
  python3 main.py <start_city> <goal_city> <algorithm>
  ```

  Algorithms: `bfs`, `dfs`, `bfs_greedy`, `a_star`

- **Dependencies**: `argparse`, `heapq`, `collections` (all standard library).

Example:

```
python3 main.py Arad Bucharest a_star
```

Appendix (Optional)

- Example Run:

  ```
  python3 main.py Arad Bucharest dfs
  DFS path: Arad -> Sibiu -> Fagaras -> Bucharest
  ```

- BFS outputs the shortest-hop path; A* outputs the least-cost path with respect to distances.