Lab 1

Searching

1 Description

You are required to implement and compare various graph search algorithms. The algorithms to be included in this lab are as follows:

1. Breadth-first search (BFS)
2. Depth-first search (DFS)
3. Uniform-cost search (UCS)
4. Iterative deepening search (IDS)
5. Greedy best-first search (GBFS)
6. Graph-search A* (A*)
7. Hill-climbing (HC) variant

You will:

- Read input from a file.
- Perform path search from the start node to the goal node on a given graph.
- Write the results to an output file.
- Compare and evaluate the algorithms based on runtime and memory usage.

Please note that:

- BFS, DFS and GBFS algorithms stop when the goal node is generated, not when the goal node is expanded.
- UCS and A* algorithms stop when the goal node is expanded.
- If Hill Climbing algorithm gets stuck, it is considered as having no path.

2 Requirements

2.1 Programming language

- The source code must be written in Python.
- You can use supporting any libraries, but the main algorithms directly related to the search process must be implemented by your own.

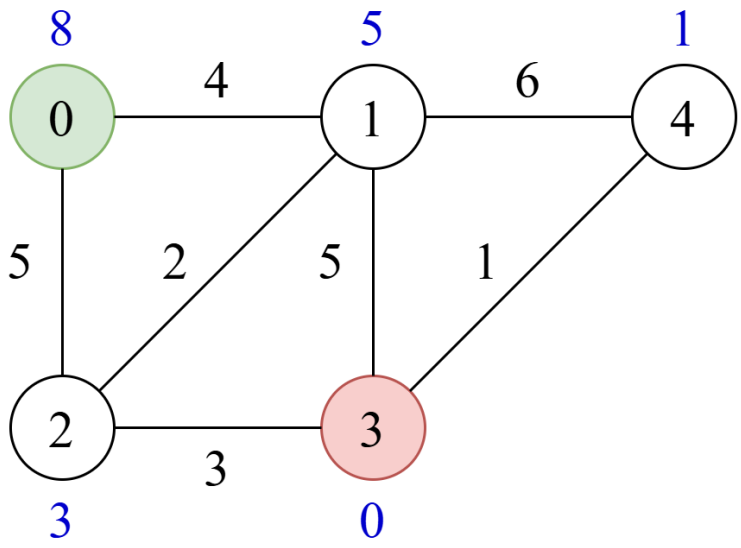
2.2 Input

The input file contains information about the graph and weights, formatted as follows:

- The first line contains the number of nodes in the graph.
- The second line contains two integers representing the start and goal nodes.
- The subsequent lines contain the adjacency matrix of the graph.
- The last line contains the heuristic weights for each node (for algorithms that use heuristics).

Note that the graph can be either directed or undirected.

Example input file:



input.txt

```
5
0 3
0 4 5 0 0
4 0 2 5 6
5 2 0 3 0
0 5 3 0 1
0 6 0 1 0
8 5 3 0 1
```

Figure 1: Example input file for a graph.

2.3 Output

The output file should contain:

- The path from the start node to the goal node for each algorithm.

  If there is no path, the output is -1.

- The runtime and memory usage of each algorithm.

Example output file:

```
BFS:
Path: 0 -> 1 -> 3
Time: 0.0000003 seconds
Memory: 8 KB

...

Hill-climbing:
Path: 0 -> 2 -> 3
Time: 0.0000003 seconds
Memory: 8 KB
```

To compare and evaluate the algorithms, you need to measure the runtime and the memory usage. Here are the detailed instructions:

- **Runtime Measurement:**
  - Use a high-resolution timer to measure the time taken by each algorithm.
  - Ensure the timer starts just before the algorithm begins processing and stops right after the path is found or determined to be nonexistent.
  - Record the time in seconds (or milliseconds if more appropriate).
- **Memory Usage Measurement:**
  - Track the peak memory usage during the execution of each algorithm.
  - Use a library or tool appropriate to measure memory consumption.
  - Record the memory usage in kilobytes (KB) or megabytes (MB).

2.4 Report

The report must fully give the following sections:

- Your information (student ID, full name, etc.).
- Self-evaluation of the completion rate of the lab and other requirements.
- Detailed algorithm description. Illustrative images are encouraged.
- Describe the test cases and performance metrics of the algorithm. Provide a complete and detailed review of your system.
- The report needs to be well-formatted and exported to PDF. Note that for editors like Jupyter notebook, you need to find a way to format it well before exporting to PDF.
- If there are figures cut off by the page break, etc., points will be deducted.
- References (if any).

2.5 Submission

- Your report, source code and test cases must be contributed in the form of a compressed file (.zip, .rar, .7z) and named according to the format **StudentID.zip/.rar/.7z**.
- If the compressed file is larger than 25MB, prioritize compressing the report and program. Test cases, may be uploaded to the Google Drive and shared via a link.

3 Notices

Please pay attention to the following notices:

- This is a **INDIVIDUAL** assignment.
- You can refer to the provided prototype source code, or you can implement it yourself.
- Duration: about 3 weeks.
- Any plagiarism, any tricks, or any lie will have a 0 point for the course grade.

4 Assessment

No.	Details	Score
1	Implement BFS correctly.	10%
2	Implement DFS correctly.	10%
3	Implement UCS correctly.	10%
4	Implement IDS correctly.	10%
5	Implement GBFS correctly.	10%
6	Implement A* correctly.	10%
7	Implement Hill-climbing correctly.	10%
8	Generate at least 5 test cases for all algorithm with different attributes. Describe them in the experiment section of your report.	10%
9	Report your algorithm, experiment with some reflection or comments.	20%
	Total	**100%**

The End.