**UNIVERSITY OF SCIENCE, VNU-HCMC**

**FALCUTY INFORMATION TECHNOLOGY**

**CSC14003 - Introduction to Artificial Intelligence**

****

**Lab 1  
Searching**

**Instructors**: Nguyễn Trần Duy Minh, Nguyễn Ngọc Thảo,   
 Nguyễn Thanh Tình, Nguyễn Hải Đăng

**Class**: 22CLC06

**Student**: 22127154 – Nguyễn Gia Huy

# Table of contents

[Table of contents 2](#_Toc171757450)

[1. Student information 3](#_Toc171757451)

[2. Self-evaluation 3](#_Toc171757452)

[3. Detailed algorithm description 4](#_Toc171757453)

[3.1. Breadth-first search (BFS) 4](#_Toc171757454)

[3.2. Depth-first search (DFS) 5](#_Toc171757455)

[3.3 Uniform-cost search (UCS) 5](#_Toc171757456)

[3.4. Iterative deepening search (IDS) 6](#_Toc171757457)

[3.5. Greedy best-first search (GBFS) 6](#_Toc171757458)

[3.6. A\* search 7](#_Toc171757459)

[3.7. Hill climbing search 7](#_Toc171757460)

[4. Implementation and experimentation 8](#_Toc171757461)

[4.1 Test cases 8](#_Toc171757462)

[4.1.1 Test case 1: Simple Weighted Undirected Graph with Admissible Heuristics 9](#_Toc171757463)

[4.1.2 Test case 2: Simple Weighted Undirected Graph with Inadmissible Heuristics 10](#_Toc171757464)

[4.1.3 Test case 3: Simple Weighted Directed Graph with Admissible Heuristics 11](#_Toc171757465)

[4.1.4 Test case 4: Simple Weighted Undirected Graph with Inadmissible Heuristics 12](#_Toc171757466)

[4.1.5 Test case 5: Complex Graph with Admissible Heuristics 13](#_Toc171757467)

[4.2. Implementation 14](#_Toc171757468)

[4.2.1 External libraries 14](#_Toc171757469)

[4.3 Experimental results 15](#_Toc171757470)

[4.3.1 Execution time 15](#_Toc171757471)

[4.3.2 Memory usage 15](#_Toc171757472)

[4.3.3 Discussion 16](#_Toc171757473)

[5. Conclusion 17](#_Toc171757474)

[6. References 17](#_Toc171757475)

# 1. Student information

- Student ID: 22127154  
- Full name: Nguyen Gia Huy  
- Class: 22CLC06

# 2. Self-evaluation

|  |  |  |
| --- | --- | --- |
| **No.** | **Details** | **Completion Rate (%)** |
| 1 | Implement BFS | 100% |
| 2 | Implement DFS | 100% |
| 3 | Implement UCS | 100% |
| 4 | Implement IDS | 100% |
| 5 | Implement GBFS | 100% |
| 6 | Implement A\* | 100% |
| 7 | Implement Hill-climbing | 100% |
| 8 | Generate at least 5 test cases for all algorithms with different attributes. Describe them in the experiment section | 100% |
| 9 | Report algorithm, experiment with some reflection or comments | 100% |

# 3. Detailed algorithm description

In this lab, we will implement the following search algorithms:

* Breadth-first search (BFS)
* Depth-first search (DFS)
* Uniform-cost search (UCS)
* Iterative deepening search (IDS)
* Greedy best-first search (GBFS)
* A\* search
* Hill climbing search

The following sections will describe the algorithms in detail.

## 3.1. Breadth-first search (BFS)

Breadth-first search is a simple graph traversal algorithm that starts at the root node and explores all the neighboring nodes. Then, it moves to the next level of nodes and explores their neighbors, and so on. The algorithm uses a queue to keep track of the nodes to be explored. The algorithm is implemented as follows:

A screen shot of a computer code

Description automatically generated

## 3.2. Depth-first search (DFS)

Depth-first search is another graph traversal algorithm that starts at the root node and explores as far as possible along each branch before backtracking. The algorithm uses a stack to keep track of the nodes to be explored. The algorithm is implemented as follows:

A screen shot of a computer code

Description automatically generated

## 3.3 Uniform-cost search (UCS)

Uniform-cost search is a variant of Dijkstra's algorithm that finds the shortest path from the start node to the goal node. The algorithm uses a priority queue to keep track of the nodes to be explored. The algorithm is implemented as follows:

A screenshot of a computer code

Description automatically generated

## 3.4. Iterative deepening search (IDS)

Iterative deepening search is a combination of depth-first search and breadth-first search that uses a depth-first search strategy with a depth limit. The algorithm is implemented as follows:

A screenshot of a computer code

Description automatically generated

## 3.5. Greedy best-first search (GBFS)

Greedy best-first search is a graph traversal algorithm that uses a heuristic function to determine the best node to explore next. The algorithm uses a priority queue to keep track of the nodes to be explored. The algorithm is implemented as follows:

A screen shot of a computer code

Description automatically generated

## 3.6. A\* search

A\* search is a graph traversal algorithm that uses a combination of the cost to reach a node and the estimated cost to reach the goal node to determine the best node to explore next. The algorithm uses a priority queue to keep track of the nodes to be explored. The algorithm is implemented as follows:

A screenshot of a computer code

Description automatically generated

## 3.7. Hill climbing search

Hill climbing search is a local search algorithm that starts at the initial state and moves to the neighboring state with the highest heuristic value. The algorithm is implemented as follows:

A computer screen shot of text

Description automatically generated

# 4. Implementation and experimentation

This section describes the implementation of the search algorithms and presents the experimental results on 5 test cases. The implementation details, test cases, and results are presented below.

## 4.1 Test cases

I have implemented the search algorithms in Python and tested them on 5 test cases.

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.

I used [*Visualgo*](https://visualgo.net/en/graphds) generate the visual representation graphs for the test cases.

I have referenced test cases and solutions for running the algorithm and checking its correctness [here](https://drive.google.com/drive/folders/1BBs84NGgreHz1Xl20xG_4x86fU36Vu98?usp=drive_link).

The details of the test cases are as follows:

### 4.1.1 Test case 1: Simple Weighted Undirected Graph with Admissible Heuristics

- Initial state: 0  
- Goal state: 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Vertex** | 0 | 1 | 2 | 3 | 4 |
| **Heuristic** | 6 | 4 | 2 | 2 | 0 |

A screenshot of a computer

Description automatically generated

Figure 1: Input file of [test case 1](https://drive.google.com/file/d/1DCODkX6cKgsEpYlLWGZpXqk6wdGcUVNx/view?usp=drive_link)  
[A screenshot of a computer

Description automatically generated](https://drive.google.com/file/d/1DCODkX6cKgsEpYlLWGZpXqk6wdGcUVNx/view?usp=drive_link)

Figure 2: Detailed illustration graph of [test case 1](https://drive.google.com/file/d/1DCODkX6cKgsEpYlLWGZpXqk6wdGcUVNx/view?usp=drive_link)

### 4.1.2 Test case 2: Simple Weighted Undirected Graph with Inadmissible Heuristics

- Initial state: 0  
- Goal state: 4

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Vertex** | 0 | 1 | 2 | 3 | 4 |
| **Heuristic** | 3 | 5 | 4 | 7 | 3 |

A screenshot of a computer

Description automatically generated  
Figure 3: Input file of [test case 2](https://drive.google.com/file/d/1DCODkX6cKgsEpYlLWGZpXqk6wdGcUVNx/view?usp=drive_link)

[A screenshot of a computer

Description automatically generated](https://drive.google.com/file/d/1DCODkX6cKgsEpYlLWGZpXqk6wdGcUVNx/view?usp=drive_link)  
Figure 4: Detailed illustration graph of [test case 2](https://drive.google.com/file/d/1DCODkX6cKgsEpYlLWGZpXqk6wdGcUVNx/view?usp=drive_link)

### 4.1.3 Test case 3: Simple Weighted Directed Graph with Admissible Heuristics

- Initial state: 0  
- Goal state: 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vertex** | 0 | 1 | 2 | 3 | 4 | 5 |
| **Heuristic** | 6 | 5 | 5 | 2 | 1 | 0 |

A screenshot of a computer

Description automatically generated

Figure 5: Input file of [test case 3](https://drive.google.com/file/d/1DD7wJJPGcKfOoGbfupcsfFoHlx09jA2U/view?usp=drive_link)

[A screenshot of a computer

Description automatically generated](https://drive.google.com/file/d/1DD7wJJPGcKfOoGbfupcsfFoHlx09jA2U/view?usp=drive_link)  
Figure 6: Detailed illustration graph of [test case 3](https://drive.google.com/file/d/1DD7wJJPGcKfOoGbfupcsfFoHlx09jA2U/view?usp=drive_link)

### 4.1.4 Test case 4: Simple Weighted Undirected Graph with Inadmissible Heuristics

- Initial state: 0  
- Goal state: 5

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Vertex** | 0 | 1 | 2 | 3 | 4 | 5 |
| **Heuristic** | 7 | 5 | 9 | 6 | 3 | 5 |

A screenshot of a computer program

Description automatically generated

Figure 7: Input file of [test case 4](https://drive.google.com/file/d/1DD7wJJPGcKfOoGbfupcsfFoHlx09jA2U/view?usp=drive_link)

[A screenshot of a computer

Description automatically generated](https://drive.google.com/file/d/1DD7wJJPGcKfOoGbfupcsfFoHlx09jA2U/view?usp=drive_link)  
Figure 8: Detailed illustration graph of [test case 4](https://drive.google.com/file/d/1DD7wJJPGcKfOoGbfupcsfFoHlx09jA2U/view?usp=drive_link)

### 4.1.5 Test case 5: Complex Graph with Admissible Heuristics

- Initial state: 0  
- Goal state: 7

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Vertex** | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| **Heuristic** | 8 | 6 | 6 | 5 | 7 | 9 | 5 | 0 |

A screenshot of a computer code

Description automatically generated

Figure 9: Input file of [test case 5](https://drive.google.com/file/d/1DDNNoNjk4IBLPE0bOS30UxK3w9wvi-3P/view?usp=drive_link)

[A screenshot of a computer

Description automatically generated](https://drive.google.com/file/d/1DDNNoNjk4IBLPE0bOS30UxK3w9wvi-3P/view?usp=drive_link)  
Figure 10: Detailed illustration graph of [test case 5](https://drive.google.com/file/d/1DDNNoNjk4IBLPE0bOS30UxK3w9wvi-3P/view?usp=drive_link)

## 4.2. Implementation

I used Python to implement the search algorithms. The complete implementation and detailed documentation of code can be found in the src folder.

Here is the structure of the src folder:

A screenshot of a computer

Description automatically generated

### 4.2.1 External libraries

I used the following external libraries in the implementation:

- ***collections***: Used for implementing the queue and stack data structures  
- ***heapq***: Used for implementing the priority queue data structure  
- ***time***: Used for measuring the execution time of the algorithms  
- ***tracemalloc***: Used for measuring the memory usage of the algorithms

## 4.3 Experimental results

I tested the search algorithms on the 5 test cases and measured the execution time and memory usage. The results are presented in the following tables:

### 4.3.1 Execution time

The execution time (second) of the search algorithms on the test cases is as follows:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test case** | **Nodes** | **Edges** | **Admissible Heuristic** | **BFS** | **DFS** | **UCS** | **IDS** | **GBFS** | **A\*** | **HC** |
| 1 | 5 | 7 | Yes | 0.0015 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 2 | 5 | 7 | No | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 3 | 6 | 10 | Yes | 0.0 | 0.0 | 0.01 | 0.0 | 0.0 | 0.0 | 0.0 |
| 4 | 6 | 10 | No | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| 5 | 8 | 14 | Yes | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |

### 4.3.2 Memory usage

The memory usage (KB) of the search algorithms on the test cases is as follows:

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test case** | **Nodes** | **Edges** | **Admissible Heuristic** | **BFS** | **DFS** | **UCS** | **IDS** | **GBFS** | **A\*** | **HC** |
| 1 | 5 | 7 | Yes | 2.0 | 0.91 | 1.02 | 0.59 | 0.56 | 1.24 | 0.36 |
| 2 | 5 | 7 | No | 1.75 | 0.73 | 0.48 | 0.48 | 0.50 | 0.43 | 0.30 |
| 3 | 6 | 10 | Yes | 1.97 | 0.88 | 1.10 | 0.55 | 0.46 | 0.69 | 0.36 |
| 4 | 6 | 10 | No | 1.97 | 0.88 | 1.10 | 0.55 | 0.52 | 0.55 | 0.36 |
| 5 | 8 | 14 | Yes | 2.27 | 0.76 | 1.29 | 0.46 | 0.52 | 0.67 | 0.36 |

### 4.3.3 Discussion

The results show that the search algorithms have different performance characteristics in terms of execution time, memory usage, and correctness. The following observations can be made:

- **Execution time**: Because the number of nodes and edges in the graph is small, the execution time of the algorithms is relatively low and not significantly different. However, the algorithms that use heuristics (GBFS, A\*) tend to be faster than the other algorithms because they can make more informed decisions about which nodes to explore next. The hill climbing search algorithm has the highest execution time because it only considers the neighboring nodes without any heuristic information.

- **Memory usage**: The memory usage of the algorithms is also relatively low due to the small size of the graph. The hill climbing search algorithm has the lowest memory usage because it only needs to store the current node and its neighbors. The algorithms that use heuristics (GBFS, A\*) have higher memory usage because they need to store additional information about the estimated cost to reach the goal node. The algorithms such as DFS, IDS and GBFS have higher memory usage because they need to store the frontier nodes in a stack or queue. The UCS and A\* algorithms have the higher memory usage because they need to store the priority queue of nodes to be explored based on the path cost and heuristic value respectively. The BFS algorithm has the highest memory usage because it needs to store all the nodes in the frontier queue.

- **Correctness**: All the algorithms except the hill climbing search algorithm are complete and optimal. The hill climbing search algorithm is not complete because it can get stuck in a local maximum and not reach the goal state. The hill climbing search algorithm is also not optimal because it only considers the neighboring nodes and may not find the shortest path to the goal state.

Overall, the choice of algorithm depends on the specific problem and the desired trade-offs between time and space complexity.

# 5. Conclusion

In this lab, we implemented and tested several search algorithms, including breadth-first search, depth-first search, uniform-cost search, iterative deepening search, greedy best-first search, A\* search, and hill climbing search. We presented the detailed algorithm descriptions, the implementation details, and the experimental results on 5 test cases. The results show that the algorithms have different performance characteristics in terms of execution time and memory usage. The choice of algorithm depends on the specific problem and the desired trade-offs between time and space complexity.

# 6. References

* Lecture slides and materials from the course.
* GeeksforGeeks. (2023, March 22). Search algorithms in AI.   
  GeeksforGeeks. <https://www.geeksforgeeks.org/search-algorithms-in-ai/>
* GeeksforGeeks. (2023b, April 20). *Introduction to hill climbing artificial intelligence*.  
  GeeksforGeeks. <https://www.geeksforgeeks.org/introduction-hill-climbing-artificial-intelligence/>