Vietnam National University, Ho Chi Minh City
UNIVERSITY OF SCIENCE
FACULTY OF INFORMATION TECHNOLOGY

# PROJECT 1 - SEARCH

## Let's chase Pac-Man!

### Introduction to Artificial Intelligence

**Instructors:**

Nguyen Thanh Tinh

23127047 - Luu Huy Hoang - 23CLC07
23127462 - Nguyen Minh Quang - 23CLC07
23127463 - Nguyen Vu Minh Quang - 23CLC07

**Ho Chi Minh City, 2025**

# Contents

# 1. Information

## 1.1. Project completion

| Implemetation | Percentage |
|---|---|
| Level 1 | 100% |
| Level 2 | 100% |
| Level 3 | 100% |
| Level 4 | 100% |
| Level 5 | 100% |
| Level 6 | 100% |

## 1.2. Project Planning and Task Distribution

| Student ID | Name | Task | Percentage |
|---|---|---|---|
| 23127047 | Luu Huy Hoang | - Implement search algorithms.<br>- Report - Algorithm Description.<br>- Record demo video. | 100% |
| 23127462 | Nguyen Minh Quang | - Implement GUI for the game.<br>- Report - Experiments. | 100% |
| 23127463 | Nguyen Vu Minh Quang | - Implement functions related to the behavior of Ghosts and Pac-Man.<br>- Report - Algorithm Description. | 100% |

## 1.3. Demo video

Link YouTube: https://youtu.be/VXsh8RUs3LE

# 2. Algorithm Description

## 2.1. Level 1 - BFS

- **Purpose**: Find the shortest path from Pac-Man to the target
- **Principle**:
  - Explore neighbors layer by layer, starting from the closest to the furthest.
  - Uses a **queue** to keep track of the cells to visit and the path to those cells.
  - Each cell is visited only once to avoid infinite loops.
  - Guarantees the shortest path in an **unweighted graph**.
- **Time Complexity**: $\mathbf{O}(\mathbf{V}+\mathbf{E})$, **V** is the number of vertices(cells) and **E** is the number of edges

## 2.2. Level 2 - DFS

- **Purpose**: Find any path from Pac-Man to the target (not guaranteed to be the shortest).
- **Principle**:
  - Explores as far as possible along each branch before backtracking.
  - Uses a **stack** or recursion to manage the exploration order.
  - Can skip already visited cells in the current path to avoid infinite loops.
- **Time Complexity**: $\mathbf{O}(\mathbf{V}+\mathbf{E})$, **V** is the number of vertices(cells) and **E** is the number of edges, but may be slower than BFS in graphs with many branches.

## 2.3. Level 3 - UCS

- **Purpose**: Find the shortest path from Pac-Man to the target when each movement has a cost. In the game, if Pacman goes straight, the cost of each move is 1. If Pacman makes a turn, the move cost is 3 and when Pacman goes in to the teleport gate on the middle road of the map, the move cost is 0.5.

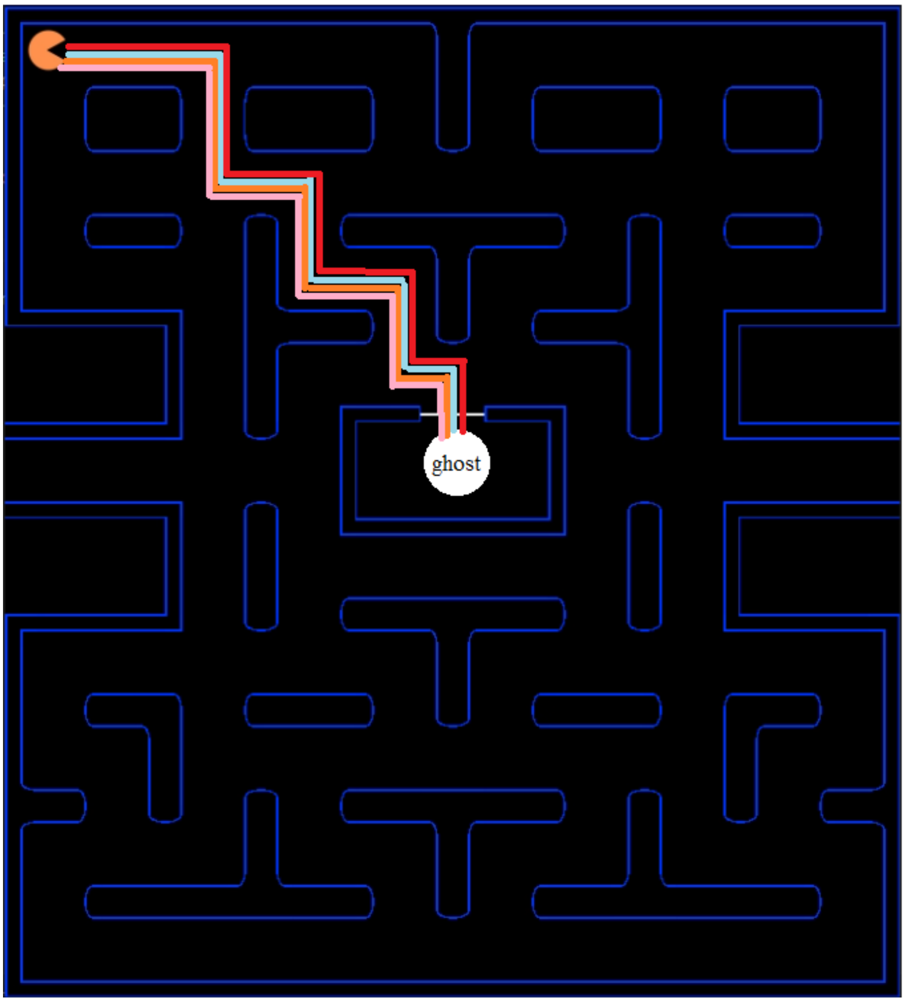
- **Principle**:
  - Expands the node with the lowest total cost from the starting point.
  - Uses a **priority queue** to keep track of the next cell to explore, where the priority is the total cost of reaching that cell.
  - If all costs are equal, UCS behaves exactly like BFS.
- **Time Complexity**: $\mathbf{O}(\mathbf{V} + \mathbf{E})$, **V** is the number of vertices(cells) and **E** is the number of edges, similar to BFS, but can be slower in graphs with varying costs.

### 2.4. Level 4 - A*

- **Purpose**: Find the shortest path efficiently by combining the UCS with a heuristic function (Manhattan Distance).
- **Principle**:
  - Expands nodes similarly to UCS, but prioritizes them based on:
    * $\mathbf{f(n) = g(n) + h(n)}$
      $\mathbf{g(n)}$: Cost to reach the current cell from the starting point.
      $\mathbf{h(n)}$: Estimated cost to reach the target from the current cell (heuristic) based on Manhattan Distance .
  - The heuristic function should never overestimate the actual cost of A* to guarantee optimality.
- **Guarantee**: If the heuristic is admissible (never underestimates) and consistent (following the triangle inequality), A* finds the optimal path faster than UCS.
- **Time Complexity**: $\mathbf{O}(\mathbf{V} + \mathbf{E})$, **V** is the number of vertices(cells) and **E** is the number of edges, but often faster due to the heuristic that guides the search.

# 3. Experiments

## 3.1. Test case 1




### 3.1.1 Search Time comparison

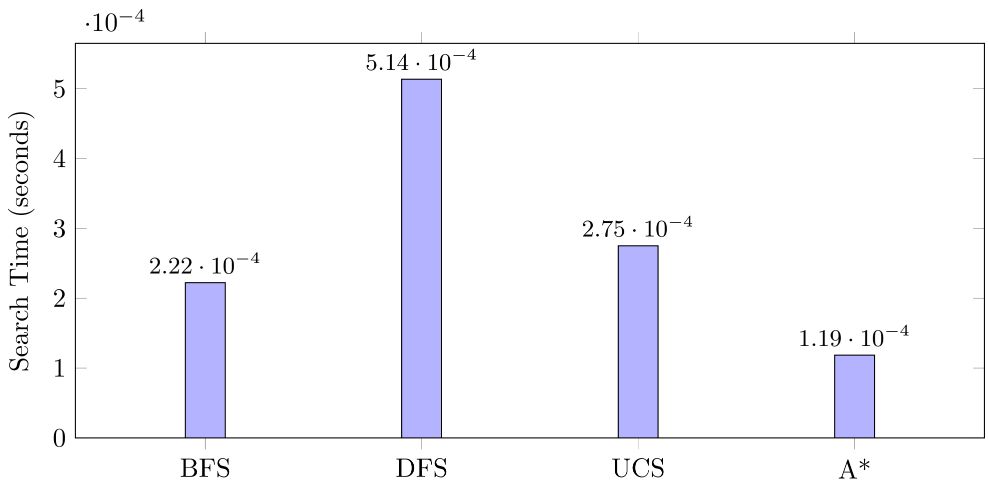



Figure 3.1: Search time comparison for different algorithms. Lower values indicate better performance.

### 3.1.2 Memory Usage comparison

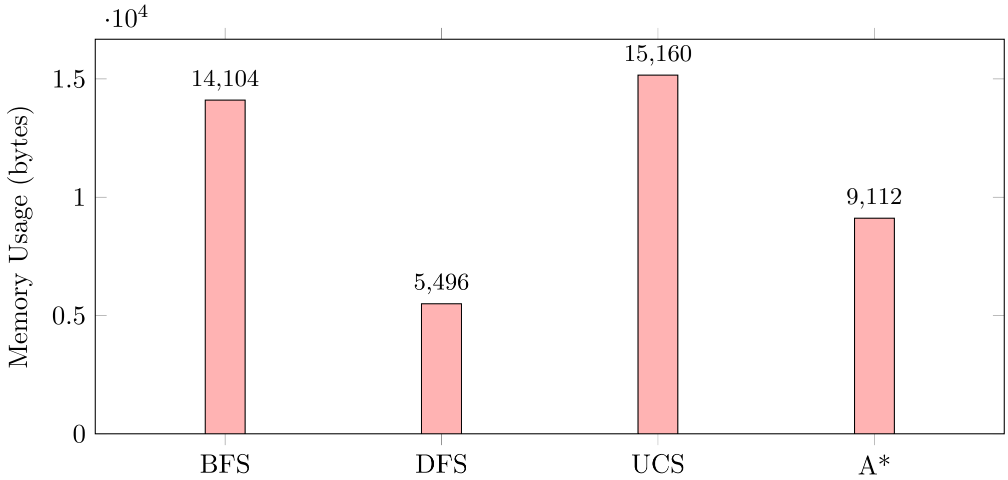



Figure 3.2: Memory usage comparison for different algorithms. Lower values indicate better performance.

### 3.1.3 Expanded Nodes comparison

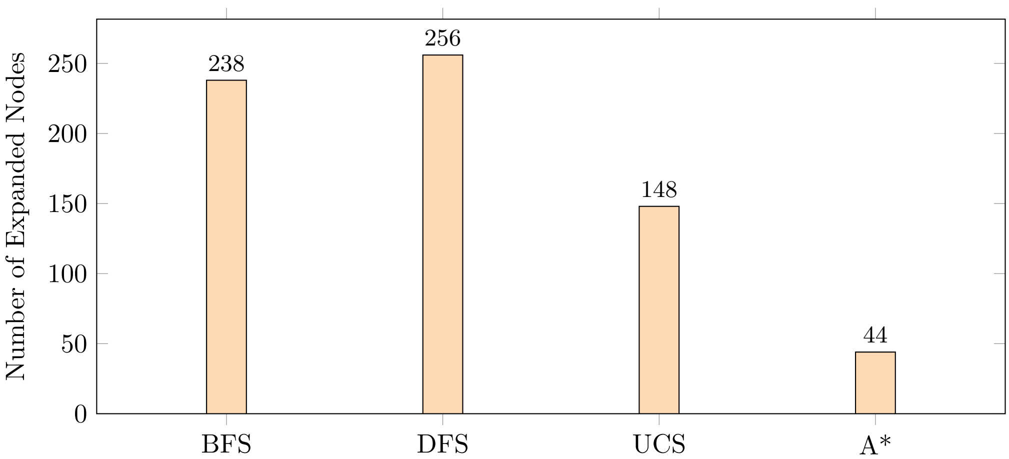
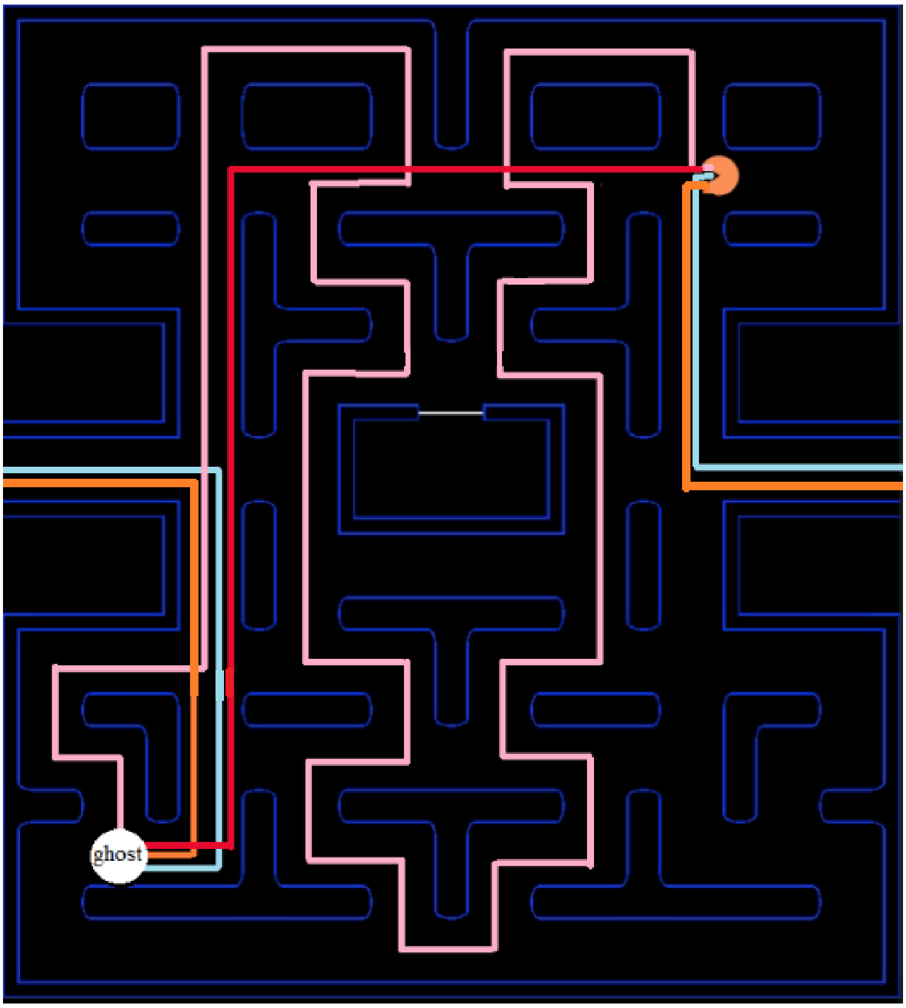



Figure 3.3: Expanded nodes comparison for different algorithms.

## Insights:

- A* is the best overall: fastest, least expanded nodes, and moderate memory usage.
- BFS is faster than UCS but expands more nodes.
- UCS performs similarly to BFS but uses more memory.
- DFS is the worst: slowest, most expanded nodes, but least memory usage.

## 3.2. Test case 2




### 3.2.1 Search Time comparison

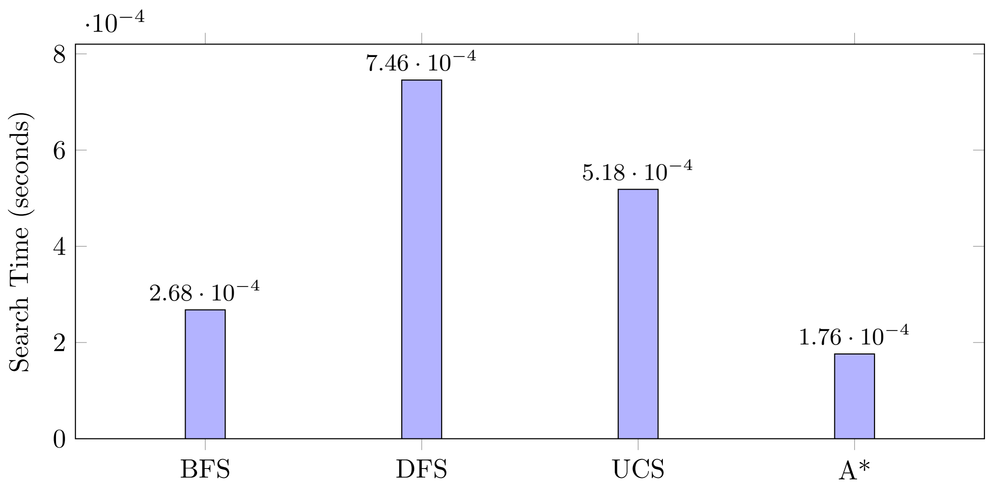



Figure 3.4: Search time comparison for different algorithms. Lower values indicate better performance.

### 3.2.2 Memory Usage comparison

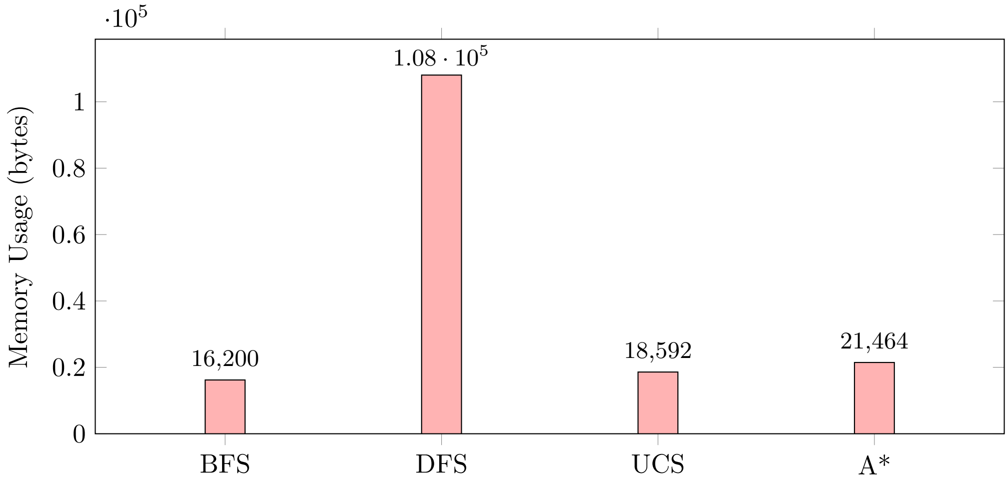



Figure 3.5: Memory usage comparison for different algorithms. Lower values indicate better performance.

### 3.2.3 Expanded Nodes comparison

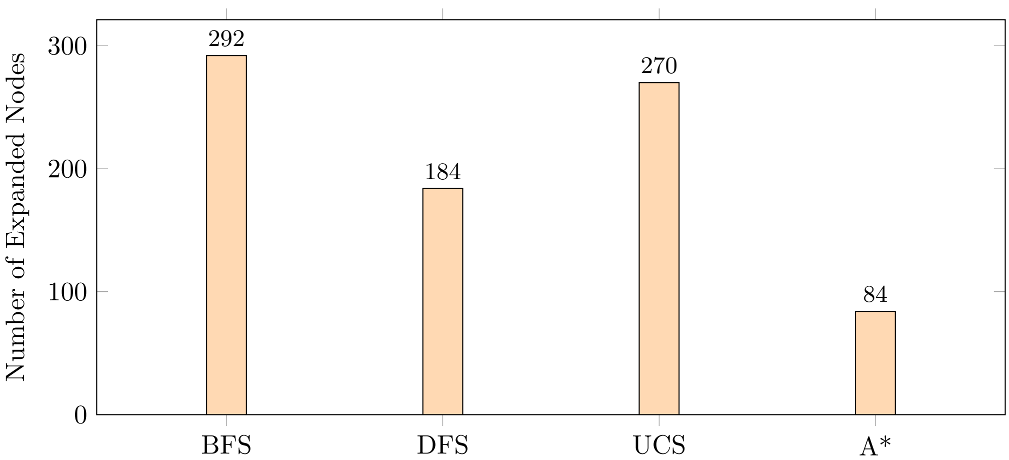
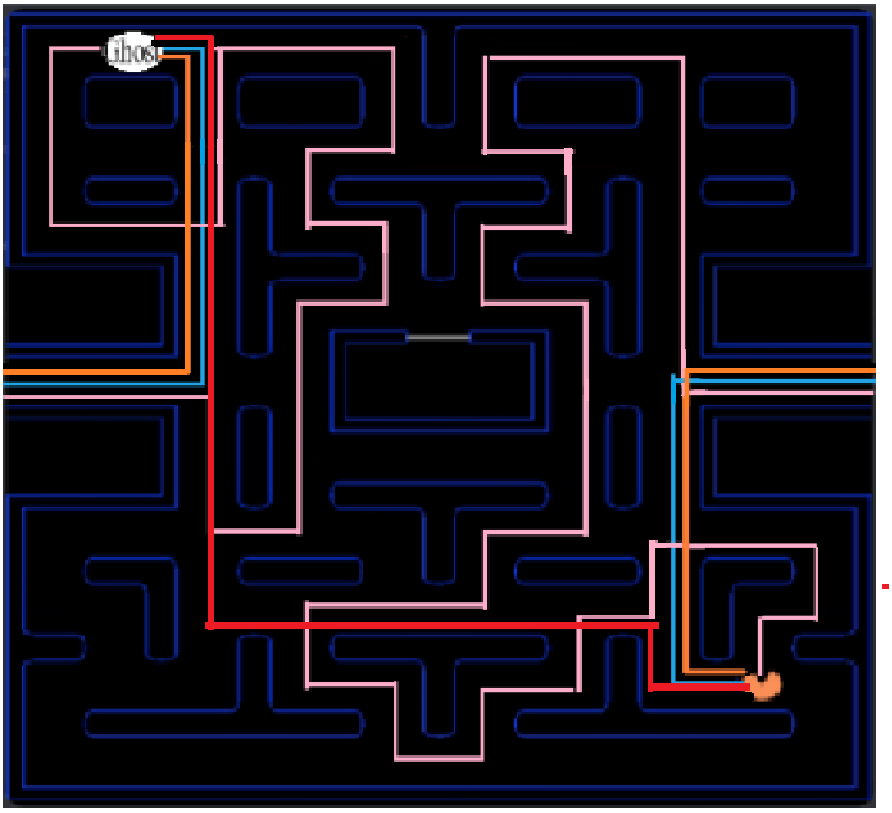



Figure 3.6: Expanded nodes comparison for different algorithms.

**Insights**:

- A* expands fewer nodes than BFS/UCS but uses more memory.
- DFS consumes significantly more memory (even more than A*) and is the slowest.

## 3.3. Test case 3




### 3.3.1 Search Time comparison

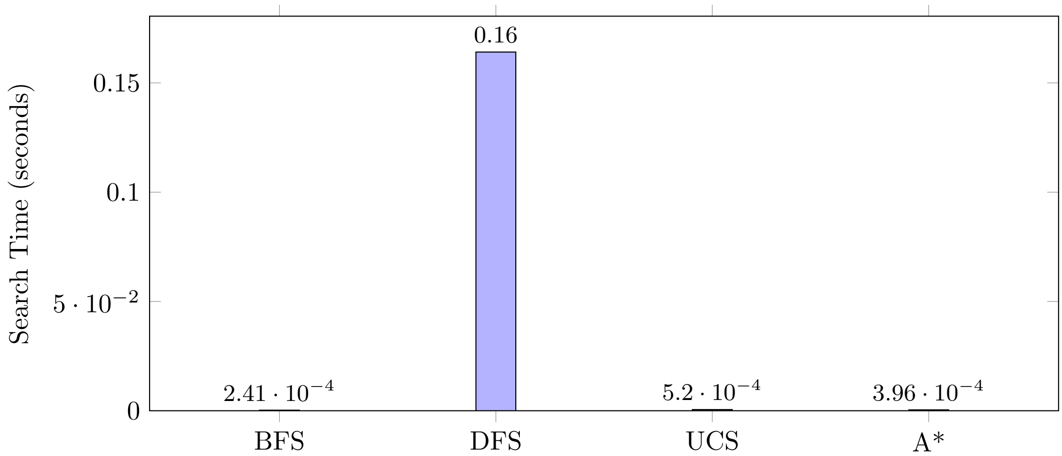



Figure 3.7: Search time comparison for different algorithms. Lower values indicate better performance.

### 3.3.2 Memory Usage comparison

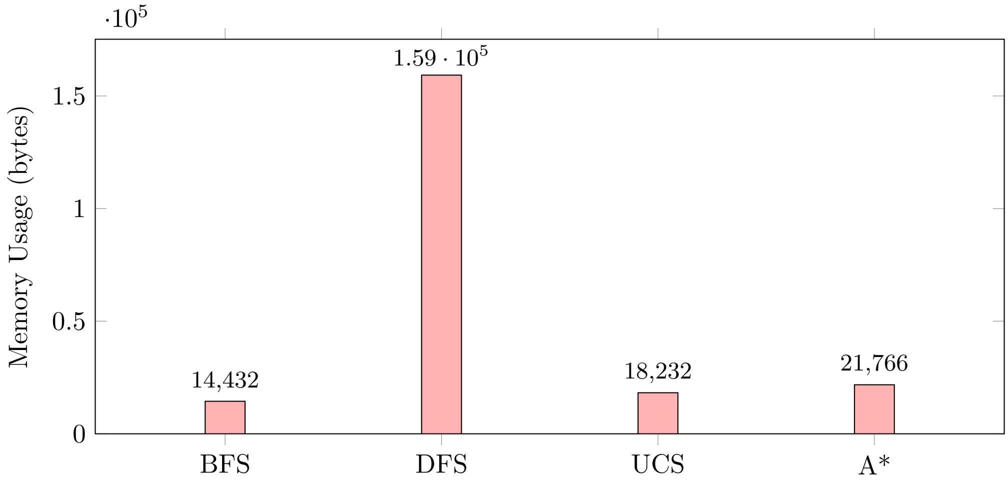



Figure 3.8: Memory usage comparison for different algorithms. Lower values indicate better performance.

### 3.3.3 Expanded Nodes comparison

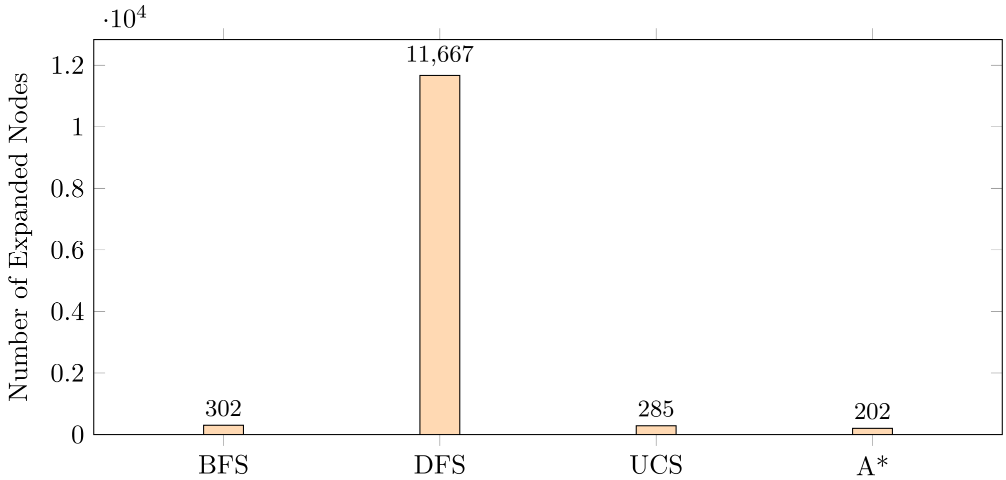
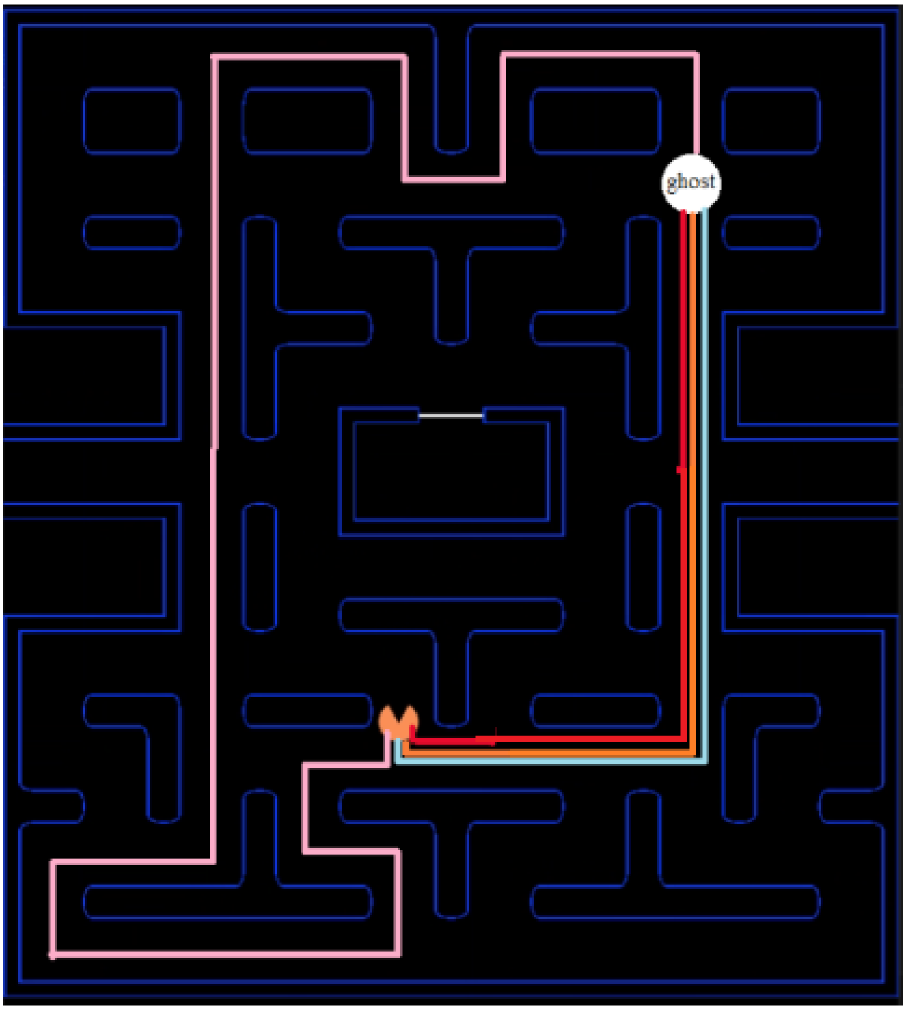



Figure 3.9: Expanded nodes comparison for different algorithms.

**Insights**:

- BFS/UCS run fast but expand many nodes.
- DFS is highly inefficient, taking 0.16s and expanding 11,667 nodes!
- A* maintains fewer expanded nodes and fast search time.

## 3.4. Test case 4




### 3.4.1 Search Time comparison

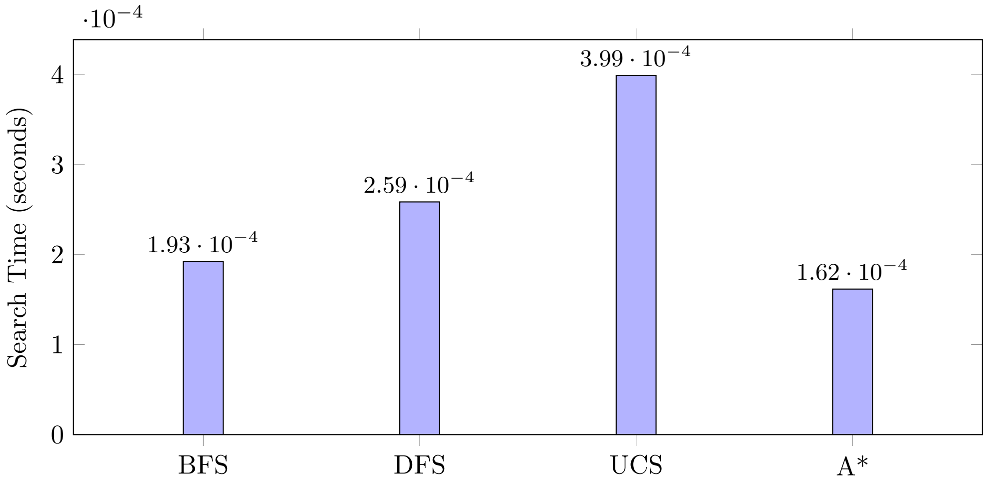



Figure 3.10: Search time comparison for different algorithms. Lower values indicate better performance.

### 3.4.2 Memory Usage comparison

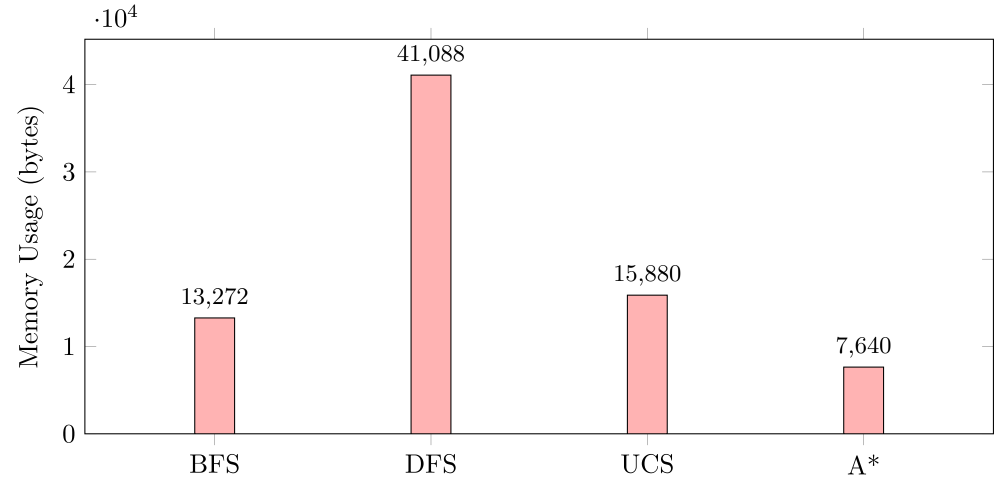



Figure 3.11: Memory usage comparison for different algorithms. Lower values indicate better performance.

### 3.4.3 Expanded Nodes comparison

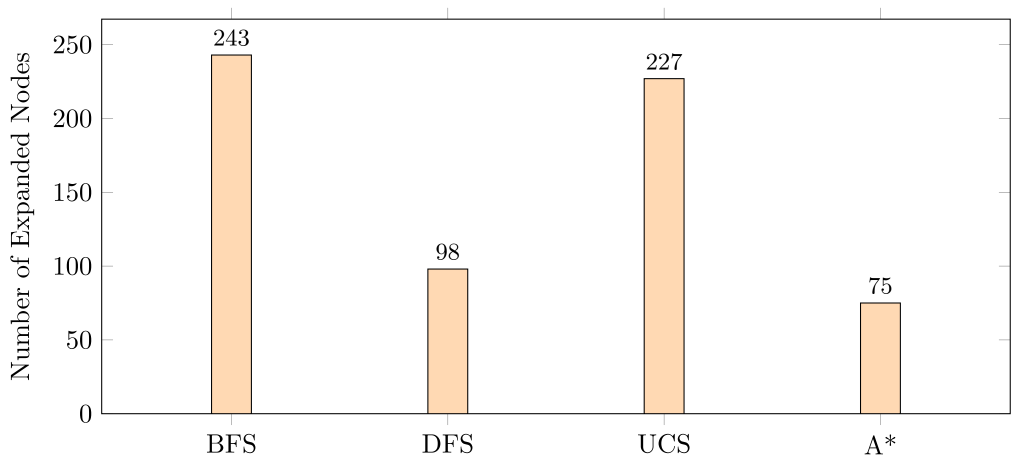
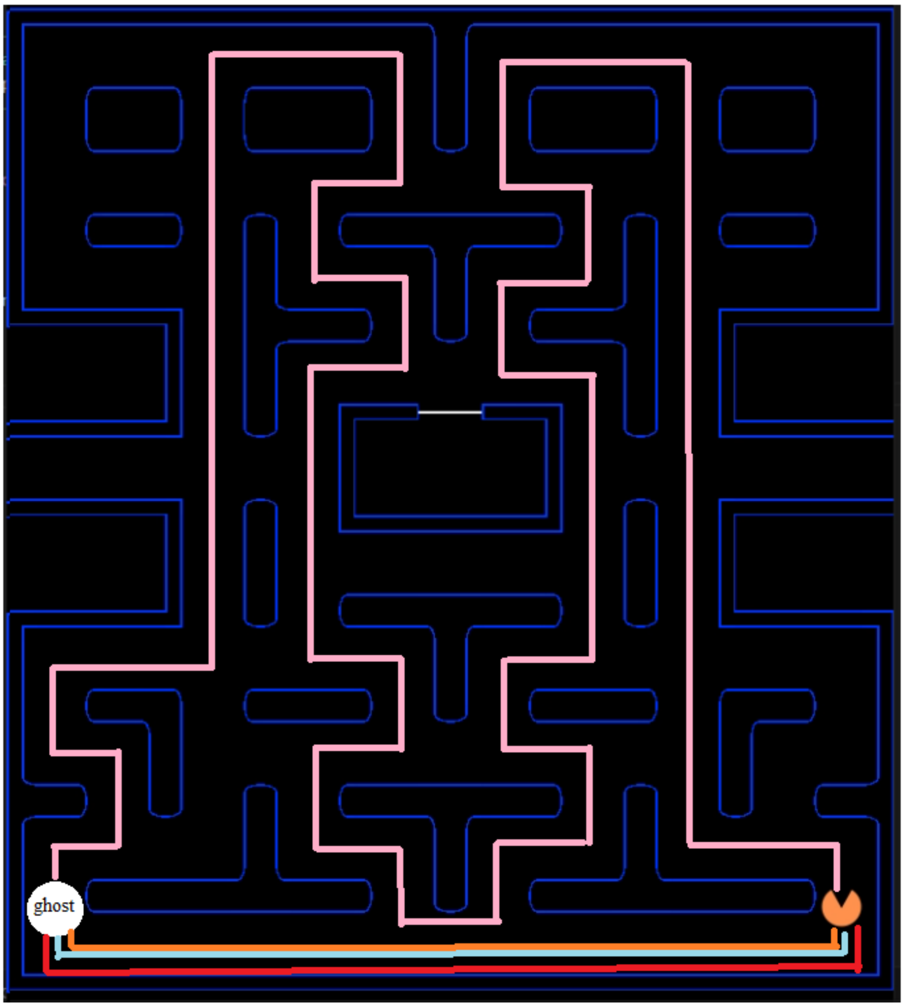



Figure 3.12: Expanded nodes comparison for different algorithms.

**Insights**:

- DFS expands almost the least nodes but still consumes a lot of memory.
- A* continues to have low expanded nodes with a moderate search time.
- BFS and UCS perform similarly in memory usage and the number of expanded nodes, but BFS is much faster.

## 3.5. Test case 5




### 3.5.1 Search Time comparison

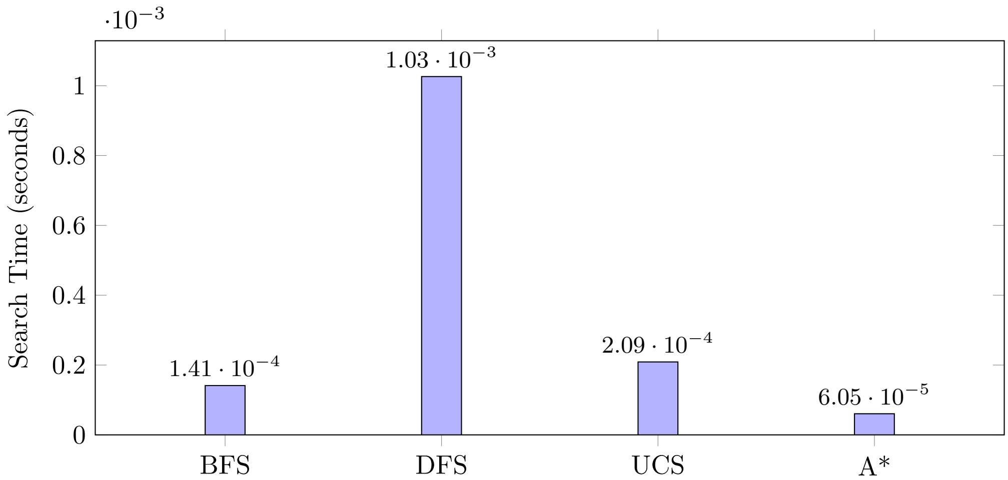



Figure 3.13: Search time comparison for different algorithms. Lower values indicate better performance.

### 3.5.2 Memory Usage comparison

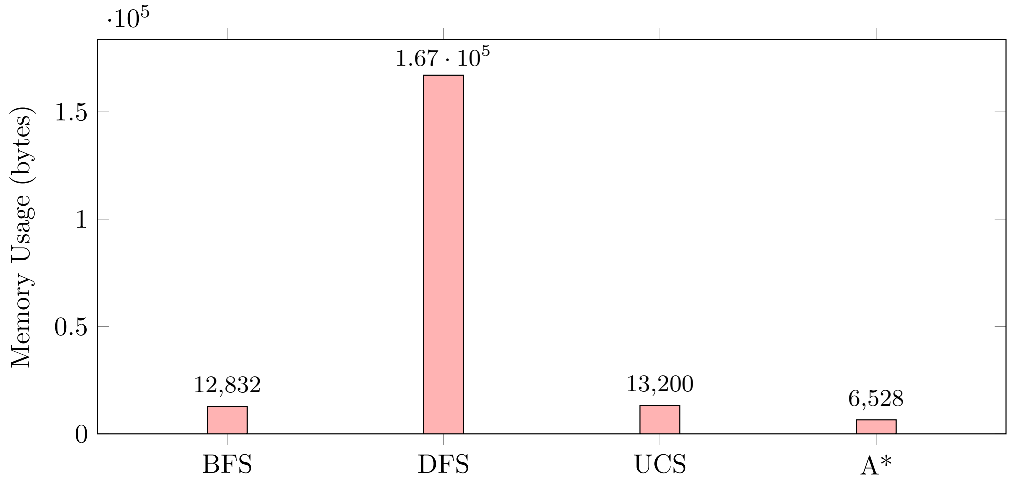



Figure 3.14: Memory usage comparison for different algorithms. Lower values indicate better performance.

### 3.5.3 Expanded Nodes comparison

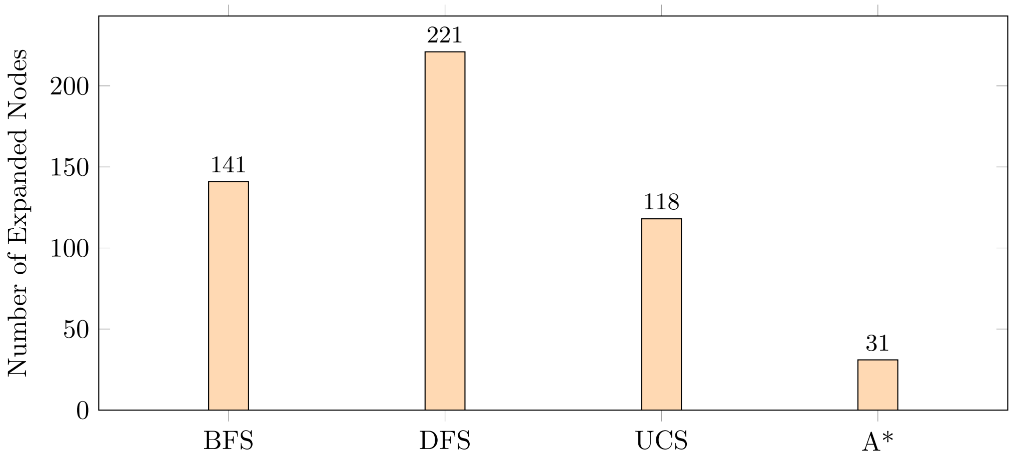



Figure 3.15: Expanded nodes comparison for different algorithms.

**Insights**:

- A* is still the best choice — fastest and expands the fewest nodes.
- Both BFS and UCS reduced memory usage and improved speed.
- DFS remains extremely memory-intensive (167080 bytes).

## 3.6. Conclusion

- DFS consistently has high memory usage and is not efficient.
- A* performs the best across speed, memory, and expanded nodes.
- BFS/UCS are sometimes fast but tend to expand many nodes.

# 4. Reference

1. Drawing the game map: Pacman - DevinLeamy
2. Pacman-AI - nxhawk