# CO6008 Knowledge-Based Systems Coursework 1

Title: A Simple Knowledge-Based Expert System for Pathfinding Using Breadth-First Search (BFS)

Author: Munir Mohammed

Date: October 2025

GitHub Repository: https://github.com/munirm9786/CO6008-BFS-Expert-System

## Table of Contents

1. Introduction  
2. Knowledge Extraction  
3. Knowledge Representation  
4. Inference Mechanism  
5. Implementation  
6. Testing and Improvements  
7. Essay Question 1: Knowledge Representation  
8. Essay Question 2: Search Algorithms  
9. Conclusion  
10. Appendix: Python Code

## 1. Introduction

This report explains how I designed and built a simple expert system that finds the shortest path between two points using the Breadth-First Search (BFS) algorithm. The project is part of the CO6008 Knowledge-Based Systems module.  
  
The system acts like a small artificial intelligence that can reason through a set of connections (called a graph) to find the best route. It works like a human trying to plan a journey — checking all possible routes step by step until the destination is reached. The goal of this project is to show how simple reasoning and search can be used to build a knowledge-based system.

## 2. Knowledge Extraction

Before building the system, I needed to decide what “knowledge” it would use. In this case, knowledge is made up of nodes (places) and the connections between them. Each connection shows that you can travel from one node to another.  
  
For example, if there is a path from A to B and A to C, then the system should know that from A, it can go to B or C. This information is simple but powerful — it can be used to reason how to move from one place to another.   
  
The information was taken from examples used in class when learning about search algorithms. It is like how AI programs represent maps or problem spaces. I converted this information into Python so it could be stored as a dictionary, which acts as the system’s knowledge base.

## 3. Knowledge Representation

The expert system represents its knowledge as a graph. The graph is stored as a Python dictionary, where each key is a node, and its values are the nodes directly connected to it.   
  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': ['G'],  
 'E': ['F', 'H'],  
 'F': ['I'],  
 'G': [],  
 'H': ['I'],  
 'I': []  
}  
  
Each line in the graph can be seen as a rule: “IF at A, THEN you can go to B or C.”  
This way, the graph works like a set of rules — just like an expert system in Prolog.

## 4. Inference Mechanism

The inference mechanism is the part of the expert system that reasons with the knowledge base to find answers. In this project, the inference mechanism is the Breadth-First Search (BFS) algorithm.  
  
BFS explores all possible moves to one level at a time until it finds the goal. Because BFS explores level by level, it always finds the shortest path if one exists. It uses a queue to manage which paths to explore next.

## 5. Implementation

The system was implemented in Python because it’s easy to use for AI tasks. Python’s built-in data structures like lists, sets, and dictionaries make it perfect for representing knowledge and writing algorithms clearly.  
  
Main parts of the program:  
1. Knowledge Base – the graph that defines which nodes connect to each other.  
2. Inference Engine (BFS Function) – the logic that searches through the graph step by step.  
3. User Interface – lets the user enter a start and goal node and see the result.

## 6. Testing and Improvements

I tested the expert system with six different input cases in Visual Studio Code.  
For each test, I entered a start and goal node and checked whether the system returned the correct shortest path or showed that no path existed.  
Screenshots of each test were saved in a folder called *testing* and uploaded to GitHub as evidence.

**Test Results:**

1. When starting at A and aiming for I, the system correctly found the path A → C → F → I.
2. When starting at A and aiming for H, it found the correct route A → B → E → H.
3. When starting at B and aiming for G, it correctly returned B → D → G.
4. When starting at C and aiming for H, the system correctly stated that there was no path between those nodes.
5. When lowercase letters (a and i) were entered, the program still worked after converting them to uppercase, finding the path A → C → F → I.
6. When a node not in the graph (Z) was entered, the program displayed the message “Start node 'Z' is not in the graph,” showing that the validation worked.

All tests produced the expected results after refinement.

**Problems and Fixes:**

* Lowercase input did not work at first, so I added .upper() to convert inputs to uppercase.
* Invalid node entries caused crashes (KeyError). I fixed this by adding input validation before running the BFS algorithm.
* The output looked unclear, so I added plain text messages such as “Shortest path found” and “No path found” to make results easier to understand.

## 7. Essay Question 1: Knowledge Representation

Knowledge representation means how information and rules are stored in an AI system so that the computer can reason with them.  
  
There are different ways to represent knowledge:  
- Rule-based systems: Use IF–THEN rules.  
- Semantic networks: Use links between concepts.  
- Frames: Use structured data.  
- Logic-based systems: Use facts and reasoning.  
  
My system uses a rule-based approach — the graph itself is a list of small rules. Each connection can be read as: IF at A THEN go to B or C.

## 8. Essay Question 2: Search Algorithms

Search algorithms are how AI systems find answers by exploring possible states or paths.  
  
There are two main types:  
- Uninformed search (BFS, DFS): Don’t use extra info.  
- Informed search (A\*, Greedy): Use hints or heuristics.  
  
BFS is an uninformed search that checks all paths level by level, always finding the shortest one if it exists.

## 9. Conclusion

This project showed how a small expert system can use search and reasoning to solve a problem. The BFS algorithm acted as the inference engine, exploring a knowledge base made of connected nodes. Testing proved that the system works correctly and is robust after improvements.

## 10. Appendix: Python Code

from collections import deque  
  
graph = {  
 'A': ['B', 'C'],  
 'B': ['D', 'E'],  
 'C': ['F'],  
 'D': ['G'],  
 'E': ['F', 'H'],  
 'F': ['I'],  
 'G': [],  
 'H': ['I'],  
 'I': []  
}  
  
def bfs\_path(graph, start, goal):  
 queue = deque([[start]])  
 visited = set()  
 while queue:  
 path = queue.popleft()  
 node = path[-1]  
 if node == goal:  
 return path  
 if node not in visited:  
 for neighbour in graph.get(node, []):  
 new\_path = list(path)  
 new\_path.append(neighbour)  
 queue.append(new\_path)  
 visited.add(node)  
 return None  
  
print("=== PATHFINDING EXPERT SYSTEM ===")  
start = input("Enter start node (e.g., A): ").strip().upper()  
goal = input("Enter goal node (e.g., I): ").strip().upper()  
  
if start not in graph:  
 print(f"❌ Start node '{start}' is not in the graph.")  
elif goal not in graph:  
 print(f"❌ Goal node '{goal}' is not in the graph.")  
else:  
 path = bfs\_path(graph, start, goal)  
 if path:  
 print("\n✅ Shortest path found:")  
 print(" → ".join(path))  
 else:  
 print("\n❌ No path found between the given nodes.")