A Project Report

on

WORD LADDER PUZZLE SOLVER

Submitted in partial fulfilment of requirements for the award of the course

of

**CGB1122 – DATA STRUCTURES**

Under the guidance of

### Mrs. K. MAKANYADEVI M.E.,

### Assistant Professor/CSE

Submitted By

**TARUNIKA C (927624BAD111)**

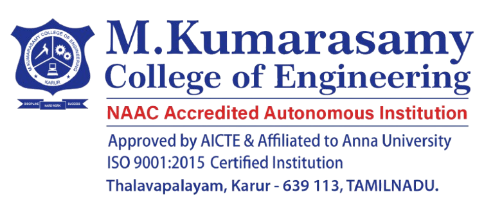
**DEPARTMENT OF FRESHMAN ENGINEERING**

**M.KUMARASAMY COLLEGE OF ENGINEERING**

(Autonomous)

**KARUR – 639 113**

MAY 2025

# M. KUMARASAMY COLLEGE OF ENGINEERING

**(Autonomous Institution affiliated to Anna University, Chennai)**

# KARUR – 639 113

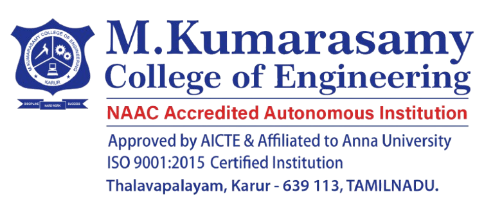
**BONAFIDE CERTIFICATE**

Certified that this project report on **“WORD LADDER PUZZLE SOLVER”** is the bonafide work of **TARUNIKA C (927624BAD111)** who carried out the project work during the academic year 2024- 2025 under my supervision.

|  |  |
| --- | --- |
| Signature | Signature |
| **Mrs. K.MAKANYADEVI M.E.,** | **Dr. K.CHITIRAKALA, M.Sc., M.Phil.,Ph.D.,** |
| **SUPERVISOR,** | **HEAD OF THE DEPARTMENT,** |
| Department of Computer Science and Engineering, | Department of Freshman Engineering, |
| M. Kumarasamy College of Engineering, | M. Kumarasamy College of Engineering, |
| Thalavapalayam, Karur -639 113. | Thalavapalayam, Karur -639 113. |

Submitted for the End Semester Review held on \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_



**DEPARTMENT OF ARTIFICIAL INTELLIGENCE AND DATA SCIENCE**

**VISION OF THE INSTITUTION**

To emerge as a leader among the top institutions in the field of technical education

**MISSION OF THE INSTITUTION**

* Produce smart technocrats with empirical knowledge who can surmount the global challenges
* Create a diverse, fully-engaged, learner-centric campus environment to provide quality education to the students
* Maintain mutually beneficial partnerships with our alumni, industry, and Professional associations

**VISION OF THE DEPARTMENT**

To produce competent industry relevant education, skillful research, technical and innovative computer science professionals acquaintance with managerial skills, human and social values.

**MISSION OF THE DEPARTMENT**

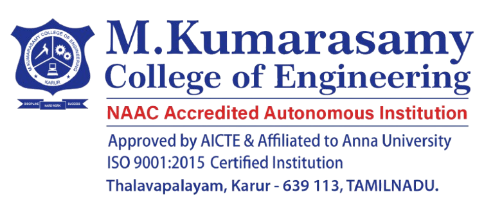
* To impart technical knowledge through innovative teaching, research, and consultancy.
* To develop and to promote student ability thereby to compete globally through excellence in education.
* To facilitate the development of academic-industry Collaboration.
* To produce competent engineers with professional ethics, technical competence and a spirit of innovation and managerial skills.

**PROGRAM EDUCATIONAL OBJECTIVES (PEOs)**

**PEO 1:** To acquire technical knowledge and proficiency required for the employment and higher education in the contemporary areas of computer science or management studies.

**PEO 2:** To apply their competency in design and development of innovative solutions for real-world problems.

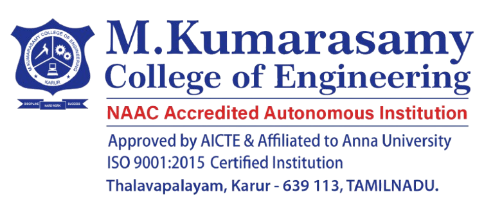
**PEO 3:** To demonstrate leadership qualities with high ethical standards and collaborated with other industries for the socio-economical growth of the country.



**PROGRAM OUTCOMES (POs)**

Engineering students will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems:** Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.
8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

1. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one’s own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
2. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

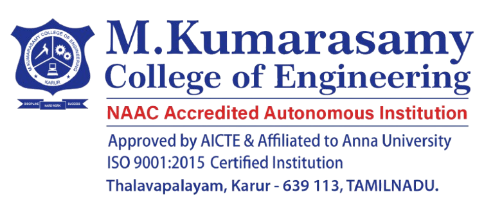
**PROGRAM SPECIFIC OUTCOMES (PSOs)**

1. **PSO1:** Ability to apply the analytical and business skills to provide sustainable solutions as an engineer/researcher for the real-time applications using Machine Learning, Internet of Things and Data analytics.
2. **PSO2:** Ability to practice ethical and human values with soft-skills qualities in computer science and business disciplines to emerge as an entrepreneur for the growth and development of the society.

# ABSTRACT

This project presents an efficient Word Ladder puzzle solver designed to compute the shortest transformation sequence between two words by altering one letter at a time, where each intermediate word is valid and belongs to a predefined dictionary. The application utilizes a breadth-first search (BFS) algorithm, ensuring optimal pathfinding within a dynamic word network. By modeling words and their relationships as graph nodes and edges, the system delivers accurate results with minimal computational overhead. Exception handling mechanisms are integrated to address invalid input cases and dictionary constraints, enhancing the system’s robustness. The interface is intuitively crafted, allowing users to input parameters and visualize the transformation path seamlessly. This project demonstrates core competencies in algorithm design, data structure application, and user-oriented functionality within a puzzle-solving framework.

# 

# ABSTRACT WITH POs AND PSOs MAPPING

|  |  |  |
| --- | --- | --- |
| **ABSTRACT** | **POs MAPPED** | **PSOs MAPPED** |
| This project presents an efficient Word Ladder puzzle solver designed to compute the shortest transformation sequence between two words by altering one letter at a time, where each intermediate word is valid and belongs to a predefined dictionary. The application utilizes a breadth-first search (BFS) algorithm, ensuring optimal pathfinding within a dynamic word network. By modeling words and their relationships as graph nodes and edges, the system delivers accurate results with minimal computational overhead. Exception handling mechanisms are integrated to address invalid input cases and dictionary constraints, enhancing the system’s robustness. The interface is intuitively crafted, allowing users to input parameters and visualize the transformation path seamlessly. This project demonstrates core competencies in algorithm design, data structure application, and user-oriented functionality within a puzzle-solving framework.  . | **PO1(2)**  **PO2(3)**  **PO3(2)**  **PO4(2)**  **PO5(3)**  **PO6(1)**  **PO7(3)**  **PO8(2)**  **PO9(3)**  **PO10(3)**  **PO11(2)**  **PO12(2)** | **PSO1(3)**  **PSO2(2)** |

Note: 1- Low, 2-Medium, 3- High

**SUPERVISOR HEAD OF THE DEPARTMENT**

# TABLE OF CONTENTS

|  |  |  |
| --- | --- | --- |
| **CHAPTER** No. | TITLE | **PAGE** No. |
| 1 | Introduction |  |
|  | 1.1 Introduction |  |
|  | 1.2 Objective |  |
|  | 1.3 Data Structure Choice |  |
| 2 | Project Methodology |  |
|  | 2.1 Graph:BFS(Breadth First Search) |  |
|  | 2.2 Block Diagram |  |
| 3 | Modules |  |
|  | 3.1 Input Module |  |
|  | 3.2 Comparison Module |  |
|  | 3.3 BFS Module3.4 Path Reconstruction Module3.5 Data Structure Module3.6 Output Module |  |
| 4 | Results and Discussion |  |
| 5 | Conclusion |  |
|  | References |  |
|  | Appendix |  |

# CHAPTER 1

# INTRODUCTION

# 1.1 Introduction

The Word Ladder puzzle, popularized by Lewis Carroll in 1878, is a classic word game that involves transforming a given start word into a target word by changing only one letter at a time, with each intermediate step forming a valid English word. This seemingly simple puzzle presents an interesting challenge in computational problem solving and can be effectively modeled using graph theory.

In this context, each word is treated as a node in a graph, and an edge connects two words if they differ by exactly one letter. To solve the puzzle efficiently, this project employs the Breadth-First Search (BFS) algorithm. BFS is particularly well-suited for this task because it explores all neighboring nodes (words) level by level, guaranteeing that the first time the target word is encountered, the shortest possible path has been found. This makes BFS an ideal choice for shortest-path problems in unweighted graphs such as the Word Ladder.

This project not only provides a working solution to the puzzle but also demonstrates key concepts in data structures, algorithms, and graph traversal. It is designed to be both educational and practical, offering insights into real-world applications of computer science fundamentals**.**

**1.2 Objective**

Our objective is to develop an efficient and user-friendly application to solve the Word Ladder puzzle by computing the shortest transformation sequence between two valid English words. The solution employs a breadth-first search (BFS) algorithm to guarantee optimal pathfinding, supported by a well-structured dictionary of valid words. A clean and intuitive user interface will facilitate easy input and clear result visualization.

In addition to solving the puzzle, the project serves as an educational tool to demonstrate key computer science concepts, including graph theory, string manipulation, and algorithm optimization. Robust error handling ensures reliability against invalid inputs, unreachable transformations, and scalability with large word sets, making the application both practical and pedagogical.

**1.3 Data Structure Choice**

When selecting a data structure for solving the Word Ladder puzzle, the choice plays a crucial role in optimizing the search for valid word transformations. A commonly preferred structure for this application is the graph. Graphs are well-suited for representing words as nodes, with edges connecting words that differ by a single letter. This structure enables efficient traversal using algorithms like breadth-first search (BFS), which guarantees the discovery of the shortest transformation path between two words.

Graphs offer clear advantages in handling large datasets by supporting dynamic relationships between words and allowing fast lookups of neighbors during traversal. However, the primary trade-off involves the overhead of constructing and maintaining the graph, especially in terms of preprocessing time and memory usage for dense word sets. Despite this, the benefits in pathfinding efficiency and scalability make graphs an ideal choice for the Word Ladder **s**olver

.**CHAPTER 2**

**PROJECT METHODOLOGY**

**2.1 Breadth-First Search (BFS)**

Breadth-First Search (BFS) is a fundamental graph traversal algorithm used to explore nodes and edges systematically in a level-by-level manner. Starting from a given source node, BFS visits all neighboring nodes before moving to the next level of neighbors. This structured approach makes BFS ideal for finding the shortest path in unweighted graphs.

In the context of the Word Ladder puzzle, BFS is particularly effective because it guarantees the shortest transformation sequence between two words. By exploring all one-letter transformations (neighbors) at each level before moving deeper, BFS ensures that the first time the target word is reached, it is thro-ugh the shortest possible path.

Key Features of BFS:

1.\*\* Shortest Path Guarantee\*\*: BFS ensures that the shortest transformation path (in terms of steps or edges) is found in unweighted graphs, making it ideal for the Word Ladder puzzle.

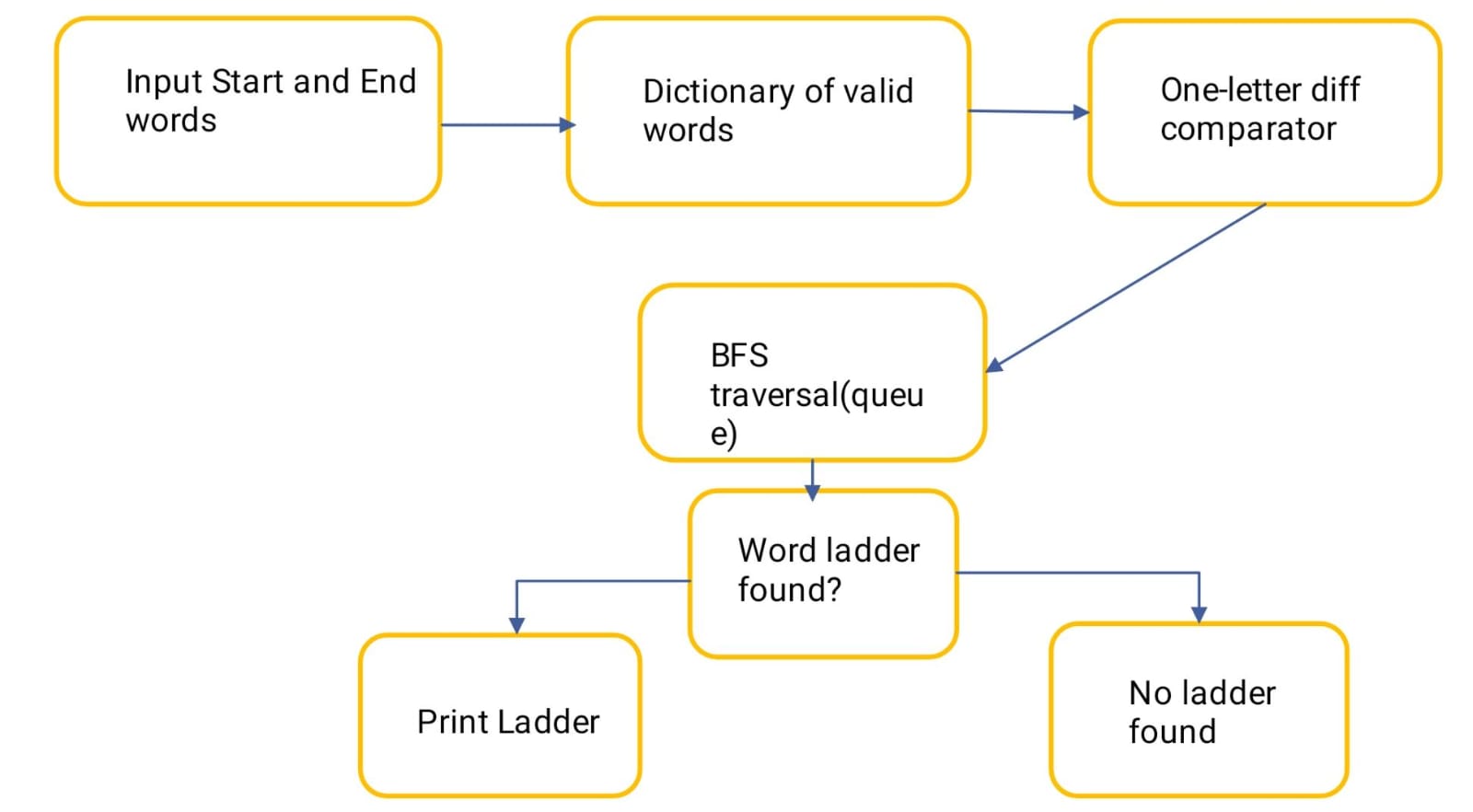
2.\*\* Queue-Based Traversal\*\*: BFS employs a queue data structure to manage the order in which nodes are visited. This first-in, first-out (FIFO) approach ensures level-by-level exploration.

3. \*\*Visited Tracking\*\*: BFS maintains a visited set or array to prevent revisiting nodes. This avoids cycles and redundant processing, improving efficiency and correctness.

4.\*\* Path Reconstruction\*\*: During traversal, BFS can store parent or predecessor information, which allows the shortest path to be reconstructed once the target node is reached.

1. \*\*Scalability and Efficiency\*\*: BFS performs reliably even with large datasets, such as extensive dictionaries used in Word Ladder, due to its organized traversal pattern and memory-efficient design.

**2.2 Block Diagram**

****

**CHAPTER 3**

**MODULES**

## Input Module

## To gather and validate all necessary data from the user to run the algorithm.Prompt the user to enter the start word and end word.Ask for the number of dictionary words.Read each dictionary word from user input and store them in a 2D array.Ensure the format and lengths of the words are appropriate.Valid input is essential for the BFS module to work correctly.It ensures words are stored in a consistent format for comparison.

## Comparison Module (Word Adjacency Check)

## int one\_letter\_diff(const char \*a, const char \*b); Determines whether two words can be connected in the word ladder (i.e., they differ by exactly one letter).Compares two strings character by character.Counts how many characters differ.Returns true only if there’s exactly one differing character.This forms the edge condition in the graph model.Helps build an implicit graph where each node (word) is connected only to valid one-letter neighbors.

## BFS Module (Breadth-First Search)

## int bfs(char \*start, char \*end, char dictionary[][WORD\_LEN], int dict\_size);Performs the BFS algorithm to explore all possible transformations and find the shortest valid path from the start word to the end word.Uses a queue of Node structures to explore transformations.Tracks visited words to prevent cycles and redundant checks.Enqueues neighboring words (based on one-letter difference).If the end word is reached, it triggers the path reconstruction.BFS ensures that the shortest transformation sequence is found.Efficient and guarantees optimal results in unweighted graphs like this.

## Path Reconstruction Module

## Integrated within bfs(). Once the target word is found by BFS, this module traces the path back to the start using the parent links stored in each nodee.It Starts from the index of the end word in the BFS queue.Follows the .parent pointers backwards to the start.Stores the indices in reverse order, then prints the transformation sequence from start to end.It turns the abstract search result into a clear, user-visible path showing how to transform the start word into the end word

## 3.5 Data Structure Module

## This module Represents each entry in the BFS queue, storing both the word and a pointer (index) to its parent word in the transformation path.word: The current word being examined or enqueued.parent: Index of the parent word in the queue used for backtracking the path.Enables path reconstruction and proper traversal of the transformation graph.

## **3.6 Output Module**

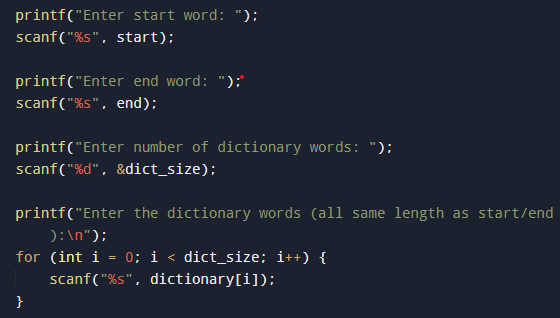
## To display the final transformation path from the start word to the end word, or a message indicating that no valid path was found.Transforms the internal logic and computation into a clear result for the user.

**CHAPTER 4**

**RESULTS AND DISCUSSION**

**4.1 Results**

**4.1.1 Input Module**



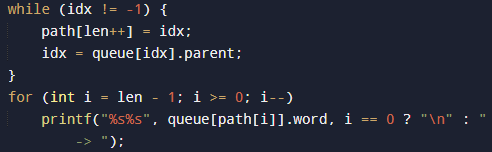
**4.1.2 Comparison Module(Word Adjacency Check)**



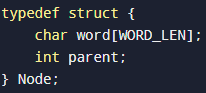
**4.1.3 BFS Module**



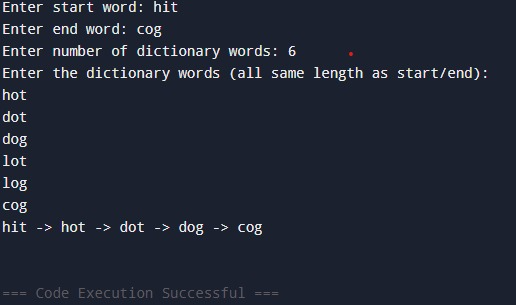
**4.1.4 Path Reconstruction Module**



**4.1.5 Data Sturcture Module**



**4.1.6 Output Module**



**4.2 Discussion**

The Word Ladder Puzzle Solver, developed using a graph-based approach with the Breadth-First Search (BFS) algorithm, offers an efficient and educational solution for solving word transformation challenges. The choice of BFS ensures the discovery of the shortest transformation path between two valid words, highlighting the reliability and optimality of the solution. The application effectively models words as graph nodes and uses string manipulation to establish edges based on one-letter differences, thereby reflecting fundamental principles of graph theory and algorithm design. The system incorporates key functionalities such as dynamic input of start and end words, dictionary validation, and transformation tracking, ensuring accuracy and responsiveness. Exception handling mechanisms are implemented to address invalid inputs and unreachable end conditions, promoting robustness and graceful failure management. The user interface is kept clear and interactive, guiding users through the transformation process and providing immediate visual feedback. This project not only demonstrates strong algorithmic proficiency but also reinforces the practical application of theoretical computer science concepts through an intuitive and user-centric design.

**CHAPTER 5**

**CONCLUSION**

In conclusion, This project demonstrates the effective use of graph traversal, specifically Breadth-First Search (BFS), to solve word transformation puzzles like the Word Ladder. It successfully converts a start word into a target word using valid dictionary entries in the fewest possible steps, showcasing the practicality of algorithm design in solving pathfinding problems. While the current implementation is limited to small datasets and returns only one shortest path, it provides a solid foundation for more advanced features. Future enhancements could include dynamic dictionary loading, support for finding all shortest transformation paths, and performance improvements using data structures like hash maps or tries. Overall, the project illustrates how core computer science concepts can be applied to build efficient and purposeful tools.

**REFERENCES**

1. Levitin, A. (2011). Introduction to the Design and Analysis of Algorithms (3rd ed.). Pearson Education.

2. Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to Algorithms (3rd ed.). MIT Press.

3. Knuth, D. E. (1997). The Art of Computer Programming, Volume 1: Fundamental Algorithms (3rd ed.). Addison-Wesley.

4. Norvig, P. (2001). Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp. Morgan Kaufmann.

1. Miller, B. N., & Ranum, D. L. (2014). Problem Solving with Algorithms and Data Structures Using Python (2nd ed.). Franklin, Beedle & Associates.

6. Wikipedia contributors. (2023). Word ladder. Wikipedia. https://en.wikipedia.org/wiki/Word\_ladder

**APPENDIX**

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#define MAX\_WORDS 1000

#define WORD\_LEN 10

typedef struct {

char word[WORD\_LEN];

int parent;

} Node;

int one\_letter\_diff(const char \*a, const char \*b) {

int diff = 0;

for (int i = 0; a[i]; i++) {

if (a[i] != b[i]) diff++;

if (diff > 1) return 0;

}

return diff == 1;

}

int bfs(char \*start, char \*end, char dictionary[][WORD\_LEN], int dict\_size) {

Node queue[MAX\_WORDS];

int visited[MAX\_WORDS] = {0};

int front = 0, rear = 0;

strcpy(queue[rear].word, start);

queue[rear].parent = -1;

rear++;

while (front < rear) {

Node current = queue[front];

if (strcmp(current.word, end) == 0) {

int path[MAX\_WORDS], len = 0, idx = front;

while (idx != -1) {

path[len++] = idx;

idx = queue[idx].parent;

}

for (int i = len - 1; i >= 0; i--)

printf("%s%s", queue[path[i]].word, i == 0 ? "\n" : " -> ");

return 1;

}

for (int i = 0; i < dict\_size; i++) {

if (!visited[i] && one\_letter\_diff(current.word, dictionary[i])) {

visited[i] = 1;

strcpy(queue[rear].word, dictionary[i]);

queue[rear].parent = front;

rear++;

}

}

front++;

}

return 0;

}

int main() {

char dictionary[MAX\_WORDS][WORD\_LEN];

char start[WORD\_LEN], end[WORD\_LEN];

int dict\_size;

printf("Enter start word: ");

scanf("%s", start);

printf("Enter end word: ");

scanf("%s", end);

printf("Enter number of dictionary words: ");

scanf("%d", &dict\_size);

printf("Enter the dictionary words (all same length as start/end):\n");

for (int i = 0; i < dict\_size; i++) {

scanf("%s", dictionary[i]);

}

if (!bfs(start, end, dictionary, dict\_size))

printf("No ladder found.\n");

return 0;

}