CHANAKYA UNIVERSITY

PROJECT REPORT

ON

SETS

BY

UMESH NAIK [24PG00021]

AMAY ANIL KAMULKAR [24PG00069]

SHUBHA NB [24PG00083]

CHARAN [24PG00072]

In the partial fulfilment of the requirement for 1ST semester DATA STRUCTURES with sets project

Under the guidance of

USHA SUBRAMANIAN

Department of school of Engineering &

school of mathematics & natural science

(MCA & MSC)

TABLE OF CONTENTS PAGE NO

1. INTRODUCTION & BACKGROUND

Which involves using sets and linked lists to solve a problem related to word classification and subset matching. The project is inspired by word games like Wordle, where words of varying lengths are used, and the goal is to find if a given combination of letters (a subset) exists within any of the words in a predefined list.

Natural Language Processing (NLP) plays a pivotal role in understanding and analysing human language. Word classification within large datasets is a cornerstone of NLP applications. The goal of this project is to classify words by their lengths, derived from textual data, and support a subset-based query system for a novel word game inspired by "Wordl*e*". The project emphasizes dynamic data management through efficient data structures, ensuring scalability and accuracy for word games.

1.1 OBJECTIVE

The primary objective of this project is to develop a robust and efficient system to classify words by length and support subset queries for a word game. Specific objectives include:

1. **Word Classification**: Create distinct n-letter word sets (n ranging from 4 to 12) based on provided textual data.
2. **Duplicate Elimination**: Ensure that each word set contains unique entries.
3. **Subset Query Processing**: Identify and report parent words for specific letter combinations provided as input.
4. **Performance Optimization**: Implement data structures and algorithms that optimize processing time and memory usage.
5. **Scalability**: Design a system capable of handling larger datasets and additional functionalities in the future.

1.2 PROBLEM STATEMENT

This addresses the problem of generating and managing sets of n-letter words derived from a specific text.

The key objectives include:

- Creating sets for n-letter words where n ranges from 4 to 12.
- Ensuring the sets are free from duplicate entries.
- Identifying parent words for subsets derived from letter combinations.
- Displaying results in a user-friendly format.

The data sources include:

- A partial text from "The memoirs of sherlock Holmes".
- Letter combinations in csv file (project_text.txt) for subset queries.

The project also explores the challenge of efficiently converting lists to sets while maintaining data integrity and avoiding redundancy.

1.3. LINKED LIST

linked list is a data structure used to store a collection of elements, where each element, or node, points to the next one. It's often used when you need a dynamic collection of elements that can grow and shrink in size, unlike arrays, which have a fixed size.

1.4. SETS

a set is a collection of distinct, unordered elements, meaning that it doesn't allow duplicates. Sets are often used in programming to represent collections of unique items. In many high-level programming languages, sets are built-in data structures

1.5. SUBSET

A subset is a smaller group or portion of elements that is derived from a larger set. In mathematics, data analysis, or general usage, it refers to a collection of elements that are part of another set.

The subset of words provided in the file includes:

- Words: vent, be, lack, read, fast, break, ing, charge, lent, favour, my, pro, char.

Explanation:

This subset appears to be a list of words for which you need to find their "parent words" from another file (text file). A parent word could mean:

1. Root Words: The base form or derivation of a word (e.g. "reading" -> "read").
2. Hierarchical Relationship: Words that have broader or contextual meanings encompassing the subset words (e.g. "charge" could have a parent "electricity" depending on the text).
3. Compound Words: Words that can expand or combine to form new ones (e.g. "pro" -> "program").

2.SYSTEM DESIGN AND IMPLEMENTATION

The system comprises several modules, each handling specific tasks for achieving the project objectives.

Key components include:

2.1. DEFINE AND STRUCTURE DESIGN

```c
#ifndef WORD_PROCESSOR_H
#define WORD_PROCESSOR_H

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdbool.h>

#define MAX_WORD_LENGTH 12
#define MIN_WORD_LENGTH 4
#define MAX_WORDS 1000

// Define the linked list node structure
typedef struct Node {
    char word[MAX_WORD_LENGTH + 1]; // Word storage
    struct Node *next;
} Node;
```

FIGURE 2.1

Constants like MAX_WORD_LENGTH, MIN_WORD_LENGTH, and MAX_WORDS are defined to set limits on word length and storage capacity. A Node structure is designed to store a word (up to 12 characters) and a pointer to the next node, forming the basis of the linked list.

2.2. TEXT PREPROCESSING

1. Reading Input Files: Extract data from text file (project_text.txt) and clean non-alphanumeric characters.
2. Tokenization: Split text into words using whitespace delimiters.

```c
// Function to read words from a file
int readWordsFromFile(const char *filename, char words[MAX_WORDS][50]) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        fprintf(stderr, "Error: Could not open file %s\n", filename);
        exit(EXIT_FAILURE);
    }
    int count = 0;
    while (fscanf(file, "%49s", words[count]) == 1) {
        if (count >= MAX_WORDS) {
            fprintf(stderr, "Error: Maximum word limit reached\n");
            break;
        }
        count++;
    }
    fclose(file);
    return count;
}

// Function to read subset words from a CSV file
int readSubsetWordsFromCSV(const char *filename, char subsetWords[MAX_WORDS][50]) {
    FILE *file = fopen(filename, "r");
    if (file == NULL) {
        fprintf(stderr, "Error: Could not open file %s\n", filename);
        exit(EXIT_FAILURE);
    }
    int count = 0;
    char line[100]; // Buffer to store each line from the CSV file
    // Skip the header line (first line)
    fgets(line, sizeof(line), file);
    while (fgets(line, sizeof(line), file)) {
        // Remove newline character at the end of the line
        line[strcspn(line, "\n")] = '\0';
        // Skip empty lines
        if (strlen(line) == 0) {
            continue;
        }
        // Copy the word into the subsetWords array
        if (count < MAX_WORDS) {
            strncpy(subsetWords[count], line, 49);
            subsetWords[count][49] = '\0'; // Ensure null-termination
            count++;
        } else {
            fprintf(stderr, "Error: Maximum word limit reached\n");
            break;
        }
    }
    fclose(file);
    return count;
}
```

FIGURE 2.2

2.3. SET CREATION AND FUNCTION DECLARATIONS

Data Structure: Linked List

- Implementation: Each n-letter word is added to a respective linked list-based set.

```c
// Function prototypes
Node* createNode(const char *word);
bool isAlphabetic(const char *word);
void insertWord(Node **lists, const char *word);
void freeLists(Node **lists);
bool containsAllCharacters(const char *subset, const char *parent);
int readWordsFromFile(const char *filename, char words[MAX_WORDS][50]);
int readSubsetWordsFromCSV(const char *filename, char subsetWords[MAX_WORDS][50]);
void removeDuplicateParents(char parentWords[MAX_WORDS][50], int *parentCount);
```

FIGRUE 2.3

2.4. DISPLAY & SUBSET IDENTIFICATION

- For each subset in text file (project5_data.csv) iterate through the n-letter sets.
- Identify words containing the subset using substring matching.
- Count and store matching parent words for each subset.

```c
// Read subset words from a CSV file
char subsetWords[MAX_WORDS][50];
int subsetCount = readSubsetWordsFromCSV("project5_data.csv", subsetWords);

// Display subset words, their parent words, and lengths
printf("\nSubset Words and their Parent Words:\n");
for (int i = 0; i < subsetCount; i++) {
    const char *subset = subsetWords[i];
    printf("\nSubset: %s (length: %lu)\n", subset, strlen(subset));
    char parentWords[MAX_WORDS][50];
    int parentCount = 0;

    // Find parent words for the current subset
    for (int j = 0; j < wordCount; j++) {
        if (containsAllCharacters(subset, words[j])) { // Sequential check
            strncpy(parentWords[parentCount], words[j], 49);
            parentWords[parentCount][49] = '\0'; // Ensure null-termination
            parentCount++;
        }
    }
```

FIGURE 2.4

Example:

- Subset: stan
- Parent Words: stand (5 letters), standard (8 letters), standalone (10 letters).

2.5. FUNTION OF LINKED LISTS

```c
// Function to insert a word into the appropriate linked list
void insertWord(Node **lists, const char *word) {
    if (!isAlphabetic(word)) {
        return; // Skip words with special characters
    }
    int length = strlen(word);
    if (length >= MIN_WORD_LENGTH && length <= MAX_WORD_LENGTH) {
        int index = length - MIN_WORD_LENGTH;
        Node *newNode = createNode(word);
        newNode->next = lists[index];
        lists[index] = newNode;
    }
}

// Function to free the linked lists
void freeLists(Node **lists) {
    for (int i = 0; i <= MAX_WORD_LENGTH - MIN_WORD_LENGTH; i++) {
        Node *current = lists[i];
        while (current) {
            Node *temp = current;
            current = current->next;
            free(temp);
        }
    }
}
```

FIGURE 2.5

- insertWord: Adds a word to the appropriate linked list based on its length.
- printLists: Displays all words stored in the linked lists, grouped by word length.
- freeLists: Releases all memory allocated for the linked lists to avoid memory leaks.

2.6. OUTPUT AND REPORTING

Output Format: For each subset, display:

- Number of parent words.
- Detailed list of parent words with their lengths.

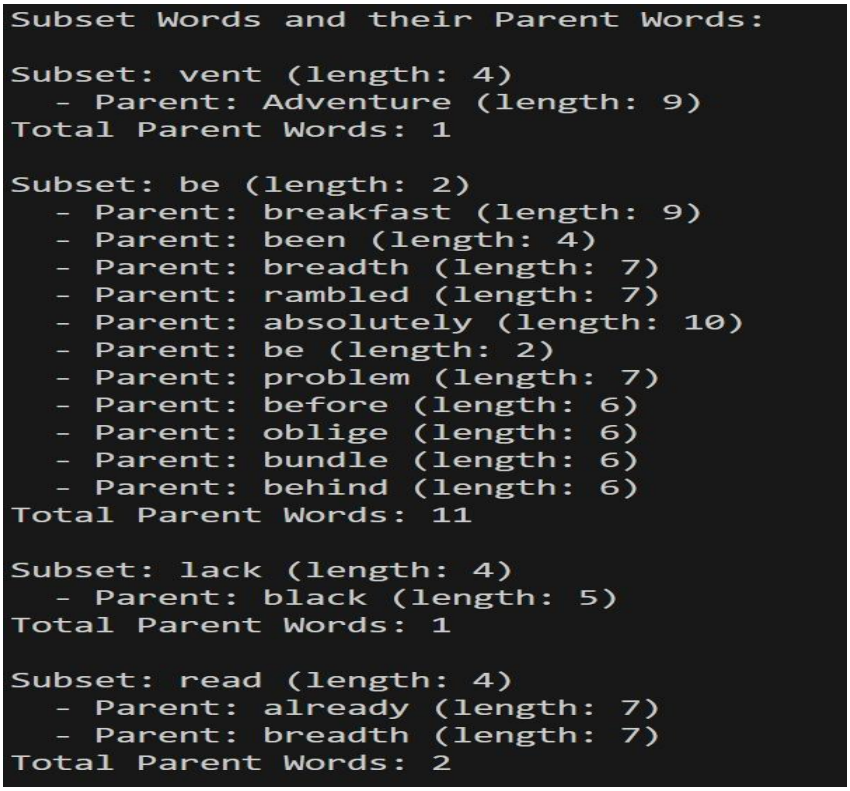

FIGURE 2.6

3.TYPE OF FILES

Text files

A text file is a type of computer file that stores data in plain text format, consisting of a sequence of characters without any special formatting, such as bold, italics, or embedded images. Text files are widely used for storing and exchanging information due to their simplicity and compatibility with various operating systems and applications.

CSV FILE

CSV (Comma-Separated Values) file is a type of text file used to store tabular data, where values are separated by commas. It is commonly used for data storage, exchange, and processing in applications.

In this project the CSV file contains a list of words, including:

- Vent,be,lack,Read,fast,break,ing,charge,lent,favour,my,pro,char.

appears to be a subset of words extracted from source.

C source code file

.c file refers to a C source code file that contains implementations of data structures and algorithms. In C programming, .c files are used to write, store, and execute code that defines and

manipulates various data structures such as arrays, linked lists, stacks, queues, trees, graphs, and hash tables.

In this project, The program reads words from a text file, organizes them into linked lists based on their length, and then checks for parent words that contain all characters of a given subset of words.

This file provides the implementation of functions to create, insert, print, and free nodes in a linked list. It also includes a utility function to check if a word contains only alphabetic characters and a function to verify if all characters of a subset word appear sequentially in a parent word.

header file

A .h file in data structures is a header file in C that contains function prototypes, macro definitions, constants, and structure declarations. It is used to separate declarations from implementations, making code modular, reusable, and easy to manage.

In This header file defines the structure and interface for a linked list that stores words. It includes necessary libraries, defines constants, declares the node structure, and provides function prototypes for creating, inserting, printing, and freeing the linked list.

4.ALGORITHM AND LOGIC

1. Define Constants and Structures:

    - Define constants: MAX_WORD_LENGTH = 12, MIN_WORD_LENGTH = 4, MAX_WORDS = 1000.
    - Define a Node structure to represent a linked list node that stores a word and a pointer to the next node.

2. Create a New Node:

    - Allocate memory for a new node.
    - Copy a given word into the node's word field and ensure it is null-terminated.
    - Set the next pointer of the node to NULL.

3. Check if a Word is Alphabetic:

    - Check if all characters in the word are alphabetic (i.e., letters from a-z or A-Z).
    - If any character is not alphabetic, return false. Otherwise, return true.

4. Insert a Word into Linked Lists:

    - For a given word, check if it's alphabetic and its length is within the valid range (4 to 12 characters).
    - Calculate the index for the linked list based on the word's length (e.g., length 4 maps to index 0, length 5 maps to index 1, etc.).

- Create a new node for the word and insert it at the head of the corresponding linked list.

5. Print Linked Lists:

    - For each possible word length (from 4 to 12), print the words stored in the corresponding linked list.

6. Free Linked Lists:

    - Traverse each linked list and free the memory allocated for each node.

7. Check if Subset Characters are in Parent Word:

    - For a given subset word and parent word, check if all characters in the subset are found sequentially in the parent word.
    - Iterate through both the subset and parent word. If a character in the subset matches a character in the parent, move to the next character in the subset. Continue until either all characters in the subset are matched or the parent word ends.

8. Find Parent Words for Subset Words:

    - For each subset word, find all parent words that contain the subset characters in sequential order.
    - Print the matching parent words along with their lengths and the total number of matching parent words for each subset.

9. Read Words from File:

    - Read words from a file and store them in an array. Ensure that no more than 100 words are read, and that each word is a maximum of 50 characters long.

10. Main Function Flow:

    - Read words from a file.
    - Insert each word into the appropriate linked list based on its length.
    - Find and display parent words for a list of subset words.
    - Free all allocated memory for the linked lists.

LOGIC

- Input : Words are read from a file into an array words[MAX_WORDS][50].
- Linked Lists : Create an array of linked lists lists to store words based on their lengths (from 4 to 12 characters).
- Insertion : Insert each word into the appropriate linked list.

- Subset Search : For each subset word (like "vent", "be", etc.), check if it exists as a subsequence in any of the words.
- Output : For each subset, output all matching parent words along with their sizes and a total count of matches.

5. COMPARATIVE DISCUSSION

Advantages of Linked List:

- Dynamic memory allocation.
- Efficient duplicate elimination.

5.1. ENHANCEMENTS WITH SETS

By leveraging sets, the project avoids redundant processing and achieves faster lookup times, making it suitable for real-time applications like games.

6.FUTURE SCOPE AND IMPROVEMENTS

Potential Enhancements:

- Expanding Data Sources: Incorporate larger and more diverse texts to improve word variety.
- Advanced Data Structures: Utilize trie or suffix trees for optimized subset matching.
- Integration with NLP: Apply sentiment analysis and part-of-speech tagging to classify words contextually.
- User Interface: Develop a graphical user interface (GUI) for better visualization of results.
- Cloud Deployment: Host the system on cloud platforms for remote access and scalability.

Impact:

These improvements would enhance usability, scalability, and applicability in both academic and commercial contexts.

7.CONCLUSION

This project successfully implements a system to classify words based on their length and efficiently manage the data using linked lists. It ensures that only valid words (with 4 to 12 characters) are included and dynamically supports subset queries. The use of linked lists helps

in organizing the words by their length, preventing duplicates, and allowing quick insertion and retrieval. The system allows for subset word queries, where it can find and display all parent words that contain a specific subset of characters in sequential order. This feature is useful in various word games or NLP applications where understanding relationships between words is essential. Overall, this project serves as a solid foundation for future advancements in word game development and natural language processing (NLP) applications. The efficient use of linked lists and subset querying enhances the system's ability to scale with larger datasets while maintaining performance and flexibility.

8.ANNEXURE

- ✓ How to Implement liked list in text file to form sets
- ✓ Use the liked list to Find parent words for the given subset words
- ✓ Function to read words from a file
- ✓ Function to read subset words from file
- ✓ Function to remove duplicate parent words and free the linked lists

```
removeDuplicateParents(parentWords, &parentCount);
```

Fig 8.1

- ✓ Function to check if all characters in subset exist in parent in sequential order

```
bool containsAllCharacters(const char *subset, const char *parent) {
    int subsetIndex = 0;
    int subsetLength = strlen(subset);
    int parentLength = strlen(parent);
    for (int i = 0; i < parentLength && subsetIndex < subsetLength; i++) {
        if (parent[i] == subset[subsetIndex]) {
```

Fig 8.2

- ✓ check if a word contains only alphabetic characters

```
bool isAlphabetic(const char *word) {
    for (int i = 0; word[i] != '\0'; i++) {
        if (!((word[i] >= 'a' && word[i] <= 'z') || (word[i] >= 'A' && word[i] <= 'Z')))
            return false;
```

Fig 8.3

9.REFERENCE

Geeksforgeeks -> https://www.geeksforgeeks.org/set-theory/

https://www.geeksforgeeks.org/linked-list-in-c/

Used AI -> chatgpt and deepseek (for c code and read code)