CSA0695 DESIGN AND ANALYSIS OF ALGORITHMS FOR OPEN ADDRESSING TECHNIQUES

-CAPSTONE PROJECT

DIFFERENCE BETWEEN MAXIMUM AND MINIMUM PRICE SUM

PRESENTED BY: GNANESH P.C

192210492

GUIDED BY: DR. R. DHANALAKSHMI

DIFFERENCE BETWEEN MAXIMUM AND MINIMUM PRICE SUM

Problem Statement & Abstract

Problem Statement:

Find the shortest sequence of digits to unlock a safe.
Password has n digits, each in the range [0, k-1].
The sequence should contain every possible combination of the n-digit password.

Abstract:

The goal is to generate a sequence using combinatorics (De Bruijn sequences) to ensure every password combination is covered in the shortest sequence possible.
Implemented using C, the project explores algorithmic methods and efficiency.

ABSTRACT

. Use De Bruijn sequences, which are the shortest sequences containing all n-digit combinations.

This method minimizes the sequence length, reducing the number of inputs required.

INTRODUCTION

Password-protected safes check the most recent n digits entered.

We need a sequence that guarantees unlocking.

SOLUTION APPROACH:

•Algorithm:

Generate a **De Bruijn sequence** using a **recursive backtracking** method to visit all possible n-digit combinations.

•Steps:

- Start at an initial state and recursively explore all combinations of digits in the range [0, k-1].
- Keep track of visited combinations to avoid repetition.
- Construct the final sequence by appending digits as combinations are explored.

•Code Implementation:

Implemented in C using:

- A recursive function to generate the sequence.
- Arrays to store visited combinations and the result sequence.

CODE IMPLEMENTATION (KEY PARTS):

.

INPUT:

```c
#include <stdio.h>

int visited[10000], idx;
char result[100000];

void generateSequence(int node, int k, int total) {
    for (int i = 0; i < k; ++i) {
        int nextNode = node * 10 + i;
        if (!visited[nextNode % total]) {
            visited[nextNode % total] = 1;
            generateSequence(nextNode % total, k, total);
            result[idx++] = '0' + i;
        }
    }
}

char* crackSafe(int n, int k) {
    int total = 1;
    for (int i = 0; i < n; ++i) total *= k;
    idx = 0;
    generateSequence(0, k, total);
    for (int i = 0; i < n - 1; ++i) result[idx++] = '0';
    result[idx] = '\0';
    return result;
}
```

OUTPUT:

```mathematica
Shortest sequence: 00110
```

Complexity Analysis

•**Time Complexity**:
O(k^n) — the algorithm explores all possible combinations of n digits from k options.

•**Space Complexity**:
O(k^n) — memory is required to store visited combinations and the resulting sequence.

•**Cases:**

- **Best Case**:
- Unlocks early in the sequence.
- **Worst Case**:
- Unlocks only after the entire sequence.
- **Average Case**:
- Unlocks somewhere in the middle of the sequence.

Conclusion & Future Scope

•**Conclusion**:
The project efficiently solves the problem using a recursive algorithm to generate the shortest sequence containing all possible password combinations. This approach minimizes the sequence length required to unlock the safe.

•**Future Scope**:
Applications in fields like **cryptography**, **data compression**, and **coding theory** where it's crucial to represent all possible combinations of sequences.

Thank you