CAPSTONE PROJECT

CRACKING THE SAFE

CSA0695- DESIGN ANALYSIS AND ALGORITHMS FOR OPEN ADDRESSING TECHNIQUES

SAVEETHA SCHOOL OF ENGINEERING

SIMATS ENGINEERING

Supervisor

Dr. R. Dhanalakshmi

Done by

Gnanesh P C (192210492)

CRACKING THE SAFE

PROBLEM STATEMENT:

The objective of this project is to determine the shortest sequence of digits that can unlock a safe. The safe is protected by a password consisting of n digits, each of which can be in the range [0, k-1]. The sequence is checked by examining the most recent n digits after every digit entered. The challenge is to find the shortest possible sequence that contains every possible combination of the n-digit password.

ABSTRACT:

This project focuses on developing a solution to efficiently unlock a password-protected safe. The password consists of n digits, where each digit can take values in the range [0, k-1]. The safe verifies the most recent n digits entered in the sequence. To unlock the safe, we need to determine the shortest sequence that contains all possible combinations of the n-digit password. This problem can be tackled using combinatorial methods, particularly De Bruijn sequences, which ensure that every possible password combination is covered in the shortest sequence. This capstone project explores the algorithmic approach and demonstrates its implementation in C.

INTRODUCTION:

In modern security systems, password protection is a fundamental method for safeguarding important data and systems. This project presents a unique challenge related to unlocking a password-protected safe using the shortest sequence of digits. The safe checks the most recent n digits entered and compares them to the correct password. Our goal is to generate a sequence that guarantees the safe will unlock at some point during the sequence input.

This problem is closely related to combinatorics and graph theory. Specifically, it can be solved using the concept of De Bruijn sequences, which are sequences that contain every possible combination of symbols of a given length at least once. By leveraging De Bruijn sequences, we can minimize the length of the sequence required to unlock the safe.

In this project, we implement a C code solution to this problem and explain how it works. We also analyze the time and space complexity of the solution, discuss various test cases, and provide insight into the future applications of this approach.

CODING:

The code generates the shortest possible sequence of digits that contains all possible n-digit combinations. This is achieved by constructing a De Bruijn sequence using recursive backtracking. The De Bruijn sequence is the shortest string that includes all possible combinations of n digits in the range [0, k-1].

C-programming

```
#include <stdio.h>
#include <string.h>

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

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

int main() {
    int n = 2, k = 2;
    printf("Shortest sequence: %s\n", crackSafe(n, k));
    return 0;
}
```

Code Explanation:

Recursive Function: The generateSequence() function generates the De Bruijn sequence by recursively visiting all possible states of the n-digit sequence using depth-first search.

Visited Array: To ensure we don't repeat any combination, we maintain a visited array to mark which n-digit sequences have been explored.

Result Array: The result array stores the final sequence that contains all possible combinations.

Main Function: In the main() function, we call crackSafe() with the values of n and k. The function returns the shortest sequence, which is printed as output.

OUTPUT:

This sequence contains all possible 2-digit combinations: 00, 01, 11, and 10.

COMPLEXITY ANALYSIS:

Time Complexity: The time complexity of this approach is O(k^n), where k is the number of digits available (i.e., the range) and n is the length of the password. This is because we need to generate and traverse all possible combinations of n digits in the range [0, k-1].

Space Complexity:

The space complexity is O(k^n) as well, as we need to store the visited states and maintain the sequence generated.

BEST CASE:

In the best case scenario, the safe unlocks after inputting the sequence without backtracking or revisiting any combination. The time taken will be minimal in this scenario, and the sequence will quickly contain the correct password.

WORST CASE:

In the worst case, the safe unlocks only after the entire sequence has been entered, and the sequence contains all possible combinations of the n-digit password. The time taken will be maximal in this case, as every combination must be checked.

AVERAGE CASE:

In the average case, the safe unlocks somewhere in the middle of the sequence. The time taken will depend on the distribution of the possible combinations and when the correct password appears in the sequence.

FUTURE SCOPE:

The concept of generating De Bruijn sequences has applications beyond unlocking safes. It can be applied in various fields like cryptography, coding theory, and data compression, where it is essential to represent all possible combinations of a sequence. In real-world scenarios, such algorithms can enhance security systems, generate unique identifiers, or optimize sequence generation problems.

CONCLUSION:

This project demonstrated an efficient solution for unlocking a password-protected safe by generating the shortest sequence of digits containing all possible combinations of the password. The solution uses a recursive approach to generate a De Bruijn sequence and ensures optimal length. We analyzed the complexity of the algorithm and discussed its best, worst, and average case performances. In the future, similar techniques can be applied to more complex combinatorial problems in various fields of computing.