SIMATS SCHOOL OF ENGINEERING

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

CHENNAI-602105

Cracking the Safe

A CAPSTONE PROJECT REPORT

Submitted in the partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

IN COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE AND DATA SCIENCE

Submitted by
Nithyanandhan R (192210692)

Under the Supervision of
Dr. T.Sangeetha

DECLARATION

I R. Nithyanandhan **,** student of **Bachelor of Engineering in Computer Science Engineering and Artificial Intelligence and Data Science** at Saveetha Institute of Medical and Technical Sciences, Saveetha University, Chennai, hereby declare that the work presented in this Capstone Project Work entitled **"Title"** is the outcome of my own bonafide work. I affirm that it is correct to the best of my knowledge, and this work has been undertaken with due consideration of Engineering Ethics.

R. Nithyanandhan
(192210692)

Date: 23-09-2024.

Place: Saveetha School of Engineering, Thandalam.

CERTIFICATE

This is to certify that the project entitled Cracking the Safe submitted by Nithyanandhan R has been carried out under my supervision. The project has been submitted as per the requirements in the current semester of B.E Computer science engineering and B.Tech Artificial Intelligence in Data science.

Faculty-in-charge

Dr. T.Sangeetha

ABSTRACT

The problem of cracking a safe with a specific password checking mechanism involves finding a sequence of digits that unlocks the safe. The safe verifies the password by checking the most recent 'n' digits entered, where 'n' is a given parameter. The password itself is a sequence of 'n' digits in a specified range. The goal is to design an algorithm that determines the shortest sequence of digits that guarantees the safe's unlocking. The algorithm needs to account for the safe's unique validation process and identify a sequence that triggers a successful match with the hidden password. The problem presents a challenge in finding an optimal strategy that minimizes the number of digits required to unlock the safe.

Keywords:

Cracking a safe
Password checking mechanism
Sequence of digits
Unlocks the safe
Most recent 'n' digits
Specified range
Hidden password
Algorithm design
Shortest sequence
Safe's validation process
Successful match
Optimal strategy
Minimizing digits required

INTRODUCTION

In this problem, you are tasked with unlocking a safe that is protected by a password. The password consists of a sequence of nnn digits, where each digit can be in the range [0,k−1][0, k-1][0,k−1]. The safe has a unique method for checking the password: it continuously checks the most recent nnn digits that were entered.

To illustrate, consider the example where the correct password is "345" and the sequence entered is
"012345":

- After typing 0, the most recent 3 digits are "0", which is incorrect.
- After typing 1, the most recent 3 digits are "01", which is incorrect.
- After typing 2, the most recent 3 digits are "012", which is incorrect.
- After typing 3, the most recent 3 digits are "123", which is incorrect.
- After typing 4, the most recent 3 digits are "234", which is incorrect.
- After typing 5, the most recent 3 digits are "345", which is correct, and the safe unlocks.

Your objective is to return a string of minimum length that will unlock the safe at some point during
the entry.

Example:

- **Input:** n=1,k=2n = 1, k = 2n=1,k=2
- **Output:** "10"
- **Explanation**: The password is a single digit, so entering each digit will unlock the safe. The sequence "10" ensures that both "0" and "1" are tried.

The challenge lies in finding the shortest possible sequence that ensures every possible combination of nnn digits within the range [0,k−1][0, k-1][0,k−1] is tested. This problem is a fascinating exercise in combinatorics and algorithm design.

CODING

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void generateSequence(char *result, int n, int k) {
 int total = 1;
 for (int i = 0; i < n; i++) {
 total *= k;
 }

 for (int i = 0; i < total; i++) {
 int num = i;
 for (int j = 0; j < n; j++) {
 result[i * n + j] = '0' + (num % k);
 num /= k;
 }
 }
}
char* crackSafe(int n, int k) {
 int length = n * (1 << (n * (k - 1)));
 char result = (char)malloc((length + 1) * sizeof(char));
 memset(result, 0, (length + 1) * sizeof(char));
 generateSequence(result, n, k);
 return result;
}
int main() {
 int n;
 int k;
 printf("Enter n:");
 scanf("%d",&n);
 printf("Enter k:");
 scanf("%d",&k);
 char *result = crackSafe(n, k);
 printf("Generated sequence: %s\n", result);
 free(result);
 return 0;
}
```

OUTPUT

Complexity Analysis

Best Case: The complexity is always the same as it must generate and store all combinations, so:

O(k^n×n)

Worst Case: The complexity remains the same because the process is deterministic and does not depend on the input values:

O(k^n×n)

Average Case: As the number of combinations is fixed, the average case is also the same as the best and worst cases:

O(k^n×n)

Overall Complexity:

Time Complexity: O(k^n×n)
Space Complexity: O(k^n×n)

CONCLUSION

The "Cracking the Safe" problem can be effectively solved using the concept of De Bruijn sequences. This mathematical approach ensures that all possible nnn-length sequences over a given alphabet are covered in the shortest possible string.

1. **Mathematical Insight: By** generating a De Bruijn sequence, we ensure that every possible combination of nnn digits appears exactly once within the sequence. This guarantees that the correct password will be tested efficiently.

2. **Feasibility Check:** The De Bruijn sequence method ensures uniqueness, correctness, and efficiency. It provides a minimal-length sequence that includes all potential passwords, ensuring the safe will unlock.

3. **Implementation**: The implementation involves generating the De Bruijn sequence using a depth-first search (DFS) approach. The result is a sequence of length **O(k^n×n)** that contains all possible nnn-length combinations.

Complexity Analysis:

- **Time Complexity**: The generation of the De Bruijn sequence has a time complexity of **O(k^n×n)** , as each of the knk^nkn sequences is visited once.
- **Space Complexity:** The space required to store the visited sequences and the resultant sequence is also **O(k^n×n)**, ensuring that all necessary combinations are checked within the sequence.

This approach is optimal and provides a clear, structured solution to the problem, ensuring both efficiency and correctness in unlocking the safe with the minimum length sequence