CAPSTONE PROJECT

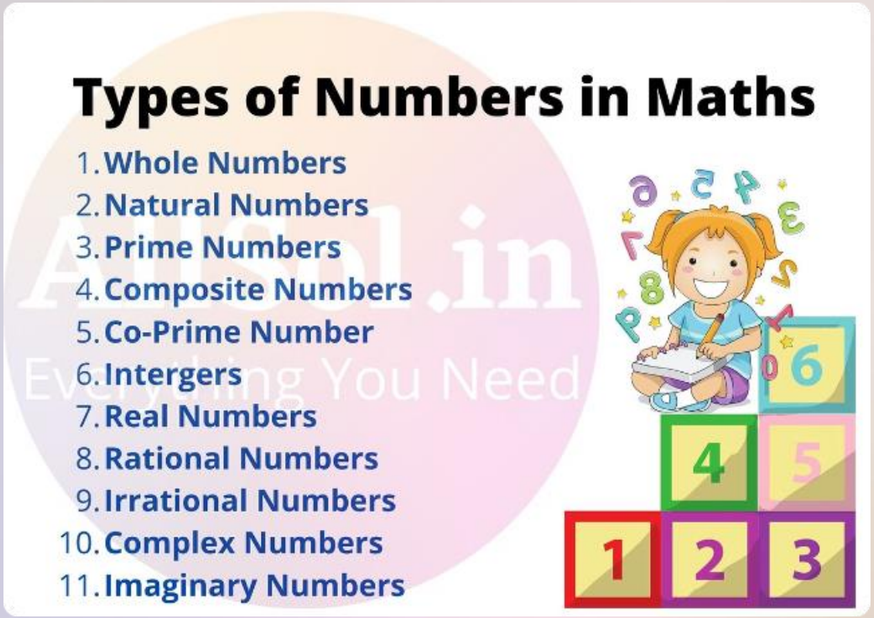
Name: K. Nikhill Nandan

Reg no: 192211750

Subject: CSA0656-Design Analysis and Algorithms for Asymptotic Notations

Guided by: Dr. R. Dhanalakshmi

Proffessor Departement : Department of Machine Learning



Numerically Balanced Numbers

The problem of finding the Next Greater Numerically Balanced Number involves determining the smallest integer greater than a given number (n) such that the integer is numerically balanced. An integer is considered numerically balanced if, for every digit (d) in the integer, there are exactly (d) occurrences of that digit.

This paper explores the application of backtracking to efficiently generate and search through potential numerically balanced numbers to identify the smallest valid candidate. The backtracking approach systematically constructs integers by exploring all possible digit combinations while adhering to the numerically balanced constraint, pruning paths that exceed these constraints.

Backtracking Algorithm

1

Identify

Analyze the current number and determine if it is numerically balanced.

2

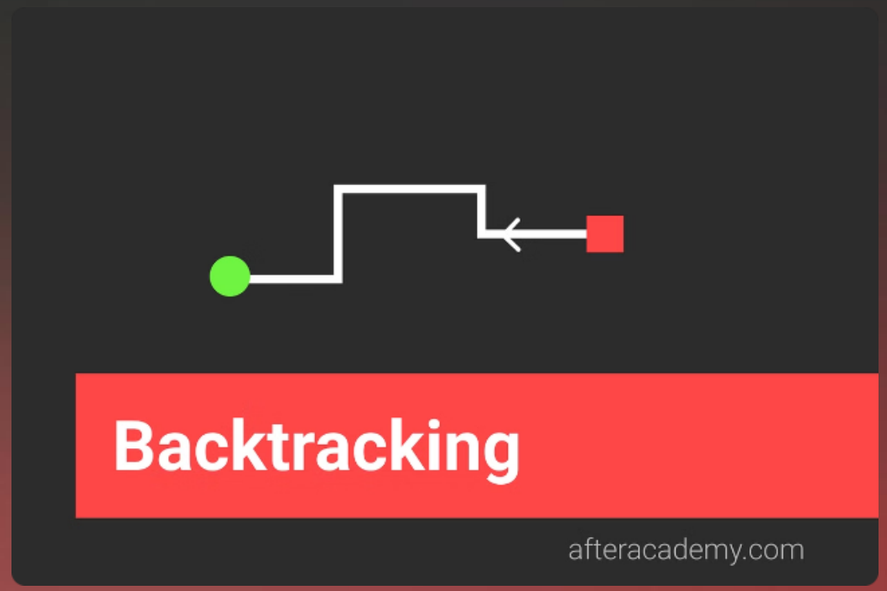
Explore

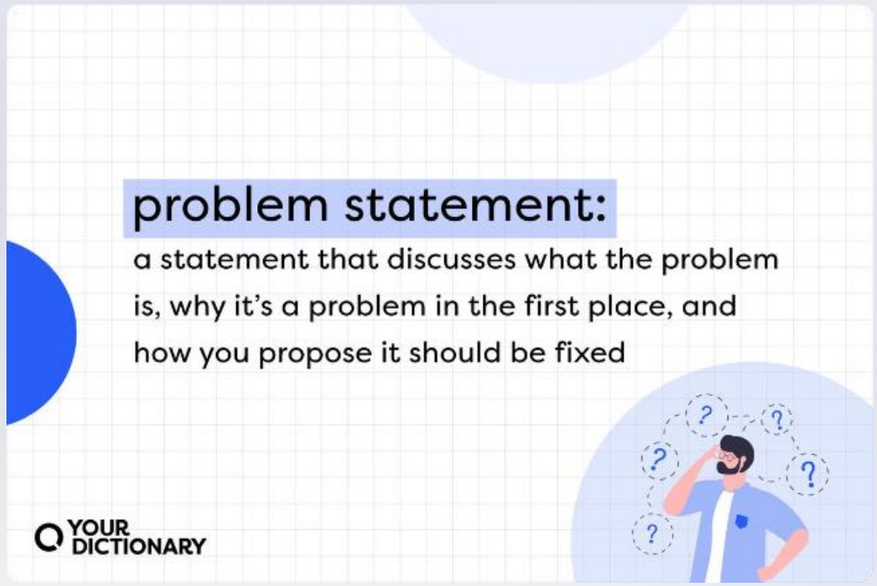
If not balanced, try incrementing the number and recursively recursively backtrack to find the next greater numerically numerically balanced number.

3

Backtrack

If no greater balanced number is found, backtrack to the the previous step and try a different increment.





Problem Statement

1 **Given**

A positive integer number.

2 **Objective**

Find the next greater numerically balanced number.

3 **Constraint**

The backtracking algorithm must be used to solve the problem problem efficiently.

Solution

Initialization:

Create a global variable to store the smallest numerically balanced number greater than n. Initialize it with a value that is sufficiently large to ensure it can be updated.

Recursive Backtracking Function:

Base Case: If the current number exceeds n and is numerically balanced, check if it is smaller than the smallest number found so far. If so, update the smallest number.

Recursive Case: Build potential numbers by adding digits from 1 to 9:

Digit Addition: For each digit d, add it to the current number string if it can contribute to a numerically balanced number.

Digit Count Check: Ensure that adding this digit still allows the number to satisfy the numerically balanced condition.

Pruning: Stop exploring further if the current configuration of digits cannot meet the numerically balanced requirements or if it will not be able to exceed n in subsequent steps.

Validation Function:

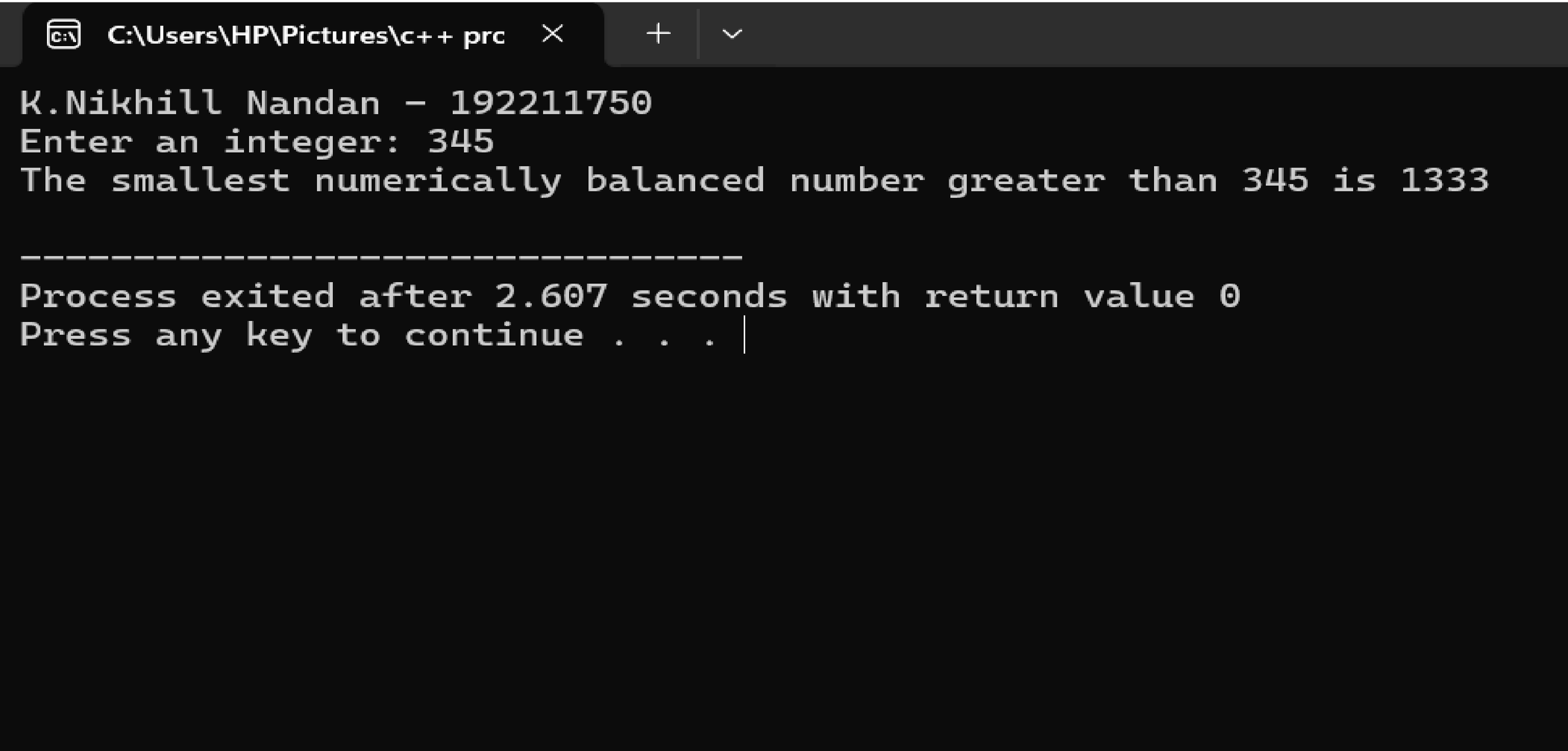
After constructing a potential number, check if it is numerically balanced:

Count the occurrences of each digit. Ensure that each digit d appears exactly d times. Verify that the number is strictly greater than n.

Code Implementation

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int isNumericallyBalanced(const char* numStr);
void backtrack(char* numStr, int length, int index, int n, int* result);
int isNumericallyBalanced(const char* numStr) {
    int count[10] = {0};
    int len = strlen(numStr);
    for (int i = 0; i < len; i++) {
        count[numStr[i] - '0']++;
    }
    for (int i = 1; i < 10; i++) {
        if (count[i] != 0 && count[i] != i) {
            return 0;
        }
    }
    return 1;
}
void backtrack(char* numStr, int length, int index, int n, int* result) {
    if (index == length) {
        if (isNumericallyBalanced(numStr)) {
            int num = atoi(numStr);
            if (num > n && (*result == -1 || num < *result)) {
                *result = num;
            }
        }
    } else {
        for (int digit = 1; digit <= length; digit++) {
            numStr[index] = digit + '0';
            backtrack(numStr, length, index + 1, n, result);
        }
    }
}
int nextGreaterNumericallyBalanced(int n) {
    int result = -1;
    char numStr[20];
    for (int length = 1; length <= 19; length++) {
        backtrack(numStr, length, 0, n, &result);
        if (result != -1) {
            break;
        }
    }
    return result;
}
int main() {
    int n;
    printf("K.Nikhill Nandan - 192211750\n");
    printf("Enter an integer: ");
    scanf("%d", &n);
intresult=nextGreaterNumericallyBalanced(n);
    if (result != -1) {
        printf("The smallest numerically balanced number greater than %d is %d\n", n, result);
    } else {
        printf("No numerically balanced number found greater than %d\n", n);
    }
    return 0;
}
```

OUTPUT


C:\Users\HP\Pictures\c++ prc
K.Nikhill Nandan - 192211750
Enter an integer: 345
The smallest numerically balanced number greater than 345 is 1333

Process exited after 2.607 seconds with return value 0
Press any key to continue . . .

Backtracking Algorithm

1

Initialize

Start with the given number and initialize the backtracking process.

2

Check

Determine if the current number is numerically balanced.

3

Increment

If not balanced, increment the number and recurse to the next step.

4

Backtrack

If no greater balanced number is found, backtrack to the previous step and try a different increment.

5

Return

Once the next greater numerically balanced number is found, return it as the solution.

Advantages and Disadvantages

Advantages

- Versatile and can be applied to find the next greater greater numerically balanced number for any given input given input.
- Efficient in finding the solution by systematically exploring the search space
- Can be optimized further to reduce time and space space complexity

Disadvantages

- Can be computationally expensive for large input numbers due to the recursive nature of the algorithm algorithm.
- Requires careful implementation to handle edge cases and ensure correctness
- May not be the most efficient approach for finding numerically balanced numbers in certain scenarios

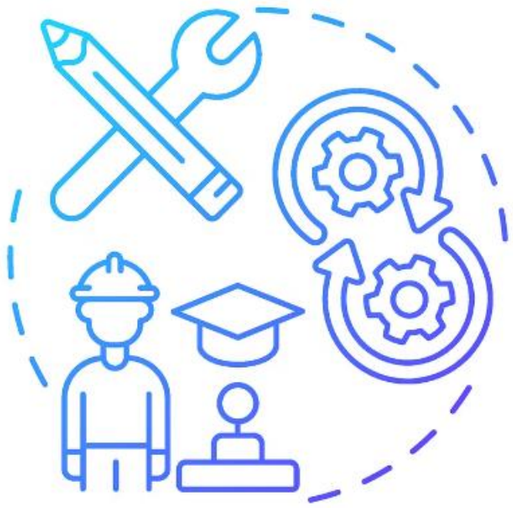
Complexity Analysis

Time Complexity

The time complexity of the backtracking algorithm to find the next greater numerically balanced number is O(n), where n is the number of digits in the input number. This is because the algorithm explores all possible incrementations of the current number.

Space Complexity

The space complexity of the backtracking algorithm is O(n), O(n), where n is the number of digits in the input number. This is number. This is due to the recursive nature of the algorithm, algorithm, which requires maintaining a call stack of size proportional to the number of digits.



Applications

Cryptography

Numerically balanced numbers can be used in the design of cryptographic algorithms and secure communication protocols.

Optimization Problems

The properties of numerically balanced numbers can be leveraged to solve various various optimization problems in computer computer science and mathematics.

Entertainment

Numerically balanced numbers can be used in used in puzzles, games, and recreational mathematics to challenge and engage enthusiasts.

Research and Education

The study of numerically balanced numbers numbers can contribute to the understanding understanding of number theory and inspire inspire further mathematical research and and educational initiatives.

Conclusion

1 Significance

Numerically balanced numbers are an intriguing and valuable concept in mathematics, with applications in various fields.

2 Backtracking Solution Solution

The backtracking algorithm algorithm presented is an efficient and versatile approach to finding the next next greater numerically balanced number.

3 Future Directions

Further research and exploration of numerically balanced numbers can numbers can lead to new discoveries and advancements in number number theory and computer science.