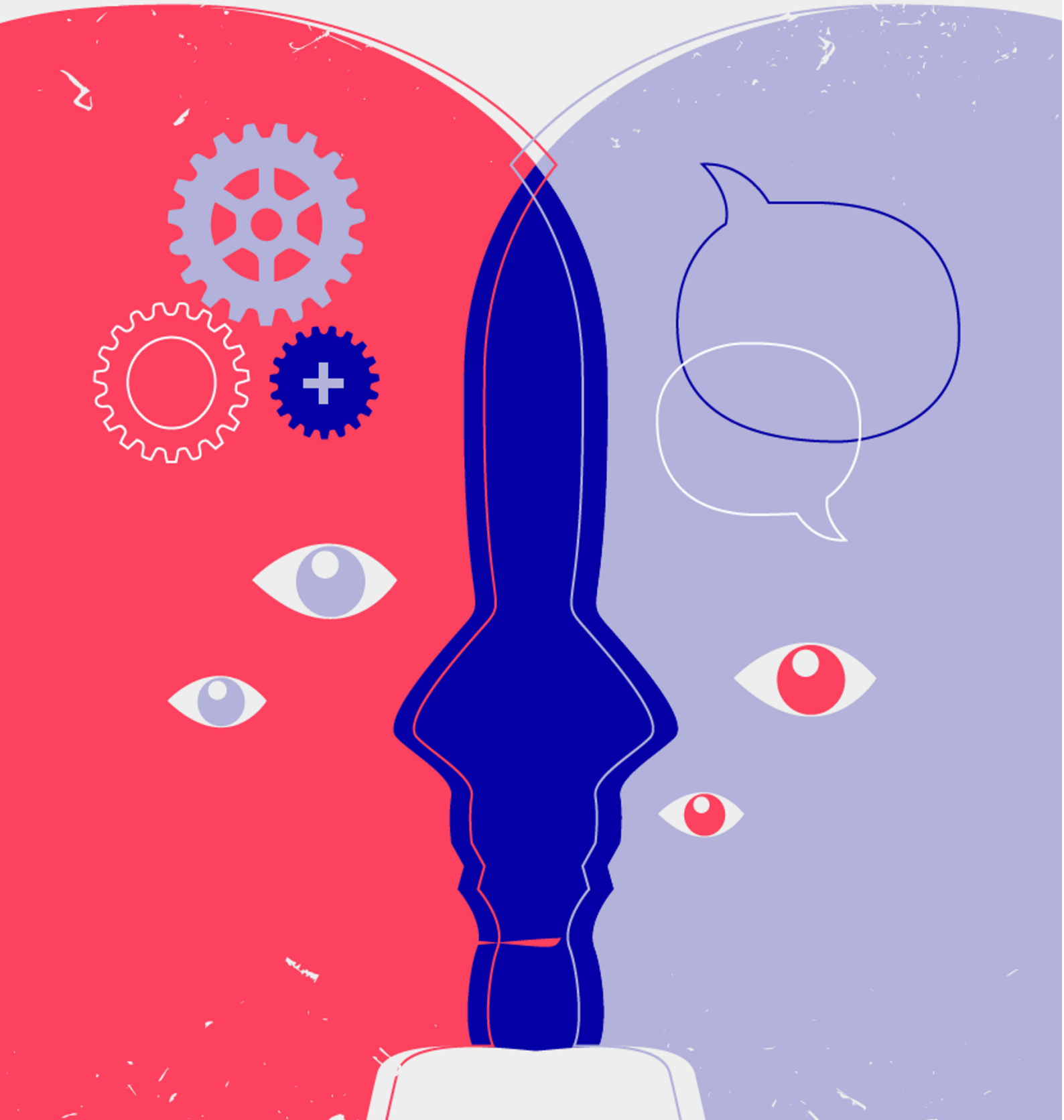
SUBMITTED BY:

TANISHQ DIXIT

➔ https://github.com/TanishqDixit/AI-Project_Cryptarithmetic_Alphametics



TITLE OF THE PROJECT

"Cryptarithmetic Puzzle Using Backtracking Algorithm & Using Brute-force search (Exhaustive search) Algorithm"

ABSTRACT OF THE PROJECT

Cryptarithmetic, also known as alphametics, is a type of mathematical game (or puzzle) consisting of a mathematical equation among unknown numbers, whose digits are represented by letters of the alphabet. The goal is to identify the value of each letter. Solving a cryptarithm by hand usually involves a mix of deductions and exhaustive tests of possibilities. To eliminate this human effort/problem, I have used backtracking paradigm of algorithm design. I have also considered using brute-force method, and algorithms that generate all permutations of m choices from n possibilities.

INTRODUCTION OF THE PROJECT

$$(C+R+Y+P-T)^4 = C\ R\ Y\ P\ T$$

What is C^R x $(Y - T) - R^R$?

Cryptarithmic puzzles are quite old and their inventor is unknown. The name "cryptarithm" was coined by puzzlist Minos in the May 1931 issue of Sphinx, a Belgian magazine of recreational mathematics, and was translated as "cryptarithmetic" by Maurice Kraitchik in 1942. The concept was first introduced by H.E. Dudeney and was first published in the July 1924 issue of *Strand Magazine* associated with the story of a kidnapper's ransom demand. In 1955, J. A. H. Hunter introduced the word "alphametic" to designate cryptarithms, such as Dudeney's, whose letters form meaningful words or phrases.

The classic example, which was published by Henry Dudeney, is:

```
  SEND
+ MORE
---------
 MONEY
```

Cryptarithmetic is a suitable example of the Constraint Satisfaction Problem. Instead of providing a description, a cryptarithmetic problem can be better described by some constraints.

Traditionally, the constraints of defining a cryptarithmetic problem are as follows:

1. Each letter should represent a different digit, and (as an ordinary arithmetic notation) the leading digit of a multi-digit number must not be zero,
2. The letters should make up a phrase (as in the example above).
3. When the digits replace letters or symbols, the resultant arithmetical operation must be correct.
4. A good puzzle should have a unique solution.

METHODOLOGY

Error Checking: Before we can apply any algorithm, we need to verify that the input entered by the user is proper.

- Total number of distinct letters should be less or equal to 10.
- Length of answer should not be lesser than the length of any operand.
- Length of answer can be only one more than any of the operands.

Since, we are going to solve cryptarithmetic puzzles using two approaches, namely, using Backtracking Algorithm & using Brute-force search (Exhaustive search) Algorithm. So let's brief with Backtracking Algorithm first and we will deal later with Brute-force search approach.

1.) Backtracking Algorithm

Backtracking is an algorithmic-technique for solving problems recursively by trying to build a solution incrementally, one piece at a time, removing those solutions that fail to satisfy the constraints of the problem at any point of time (by time, here, is referred to the time elapsed till reaching any level of the search tree).

Explanation of Algorithm used in the Code:

This is the "not-very-smart" version of cryptarithmetic solver. It takes the puzzle itself (with the 3 strings for the two addends and sum) and a string of letters as yet unassigned. If no more letters to assign then we've hit a base-case, if the current letter-to-digit mapping solves the puzzle, we're done, otherwise we return false to trigger backtracking If we have letters to assign, we take the first letter from that list, and try assigning it the digits from 0 to 9 and then recursively working through solving puzzle from here. If we manage to make a good assignment that works, we've succeeded, else we need to unassign that choice and try another digit. This version is easy to write, since it uses a simple approach (quite similar to permutations if you think about it) but it is not so smart because it doesn't take into account the structure of the puzzle constraints (for example, once the two digits for the addends have been assigned, there is no reason to try anything other than the correct digit for the sum) yet it tries a lot of useless combos regardless.

Explanation of the Code:

Each important step and explanation of the implementation of the code is explained using the comments and is clearly depicted in the code below.

The Code:

```
// Cryptarithmetic Puzzle using C++
// Using Backtracking Algorithm to look for One of the possible solns.
// Tanishq Dixit (2019IMT-120)
#include <bits/stdc++.h>
using namespace std;

// vector stores 1, corresponding to index num, when one char is assigned previo
usly; otherwise stores 0
vector<int> use(10);

// structure to store char and its corresponding integer
struct node
{
    char c;
    int v;
```

```
};

// function check for correct solution
int check(node* nodeArr, const int count, string s1,
                            string s2, string s3)
{
    int val1 = 0, val2 = 0, val3 = 0, m = 1, j, i;

    // calculate number corresponding to first string
    for (i = s1.length() - 1; i >= 0; i--)
    {
        char ch = s1[i];
        for (j = 0; j < count; j++)

        //when ch is present, break the loop
            if (nodeArr[j].c == ch)
                break;

        val1 += m * nodeArr[j].v;
        m *= 10;
    }
    m = 1;

    // calculate number corresponding to second string
    for (i = s2.length() - 1; i >= 0; i--)
    {
        char ch = s2[i];
        for (j = 0; j < count; j++)
            if (nodeArr[j].c == ch)
                break;

        val2 += m * nodeArr[j].v;
        m *= 10;
    }
    m = 1;

    // calculate number corresponding to third string
    for (i = s3.length() - 1; i >= 0; i--)
    {
        char ch = s3[i];
        for (j = 0; j < count; j++)
            if (nodeArr[j].c == ch)
                break;

        val3 += m * nodeArr[j].v;
        m *= 10;
    }

    // check whether the sum is same as third string or not; return true
    if (val3 == (val1 + val2))
        return 1;

    // else return false
    return 0;
}

// Recursive function to check solution for all permutations
bool permutation(const int count, node* nodeArr, int n,
                string s1, string s2, string s3)
{
    // Base case; when values are assigned for all characters
    if (n == count - 1)
    {
```

```
        // check for all numbers not used yet; assign value i
        for (int i = 0; i < 10; i++)
        {

            // if not used
            if (use[i] == 0)
            {

                // assign char at index n integer i
                nodeArr[n].v = i;

                // if solution found
                if (check(nodeArr, count, s1, s2, s3) == 1)
                {
                    cout << "\nSolution found: ";
                    for (int j = 0; j < count; j++)
                        cout << " " << nodeArr[j].c << " = "
                             << nodeArr[j].v;
                    return true;
                }
            }
        }
        return false;
    }

    for (int i = 0; i < 10; i++)
    {

        // if ith integer not used yet
        if (use[i] == 0)
        {

            // assign char at index n, int value i
            nodeArr[n].v = i;

            // mark it as not available for other char in future
            use[i] = 1;

            // call recursive function; go for next characters
            if (permutation(count, nodeArr, n + 1, s1, s2, s3))
                return true;

            // backtrack for all other possible solutions
            use[i] = 0;
        }
    }
    return false;
}

bool solveCryptographic(string s1, string s2,
                        string s3)
{
    // count to store number of unique characters
    int count = 0;

    // Length of all three strings
    int l1 = s1.length();
    int l2 = s2.length();
    int l3 = s3.length();

    // vector to store frequency of each char; there are 26 diff. char
    vector<int> freq(26);
```

```
    for (int i = 0; i < l1; i++)
        ++freq[s1[i] - 'A'];

    for (int i = 0; i < l2; i++)
        ++freq[s2[i] - 'A'];

    for (int i = 0; i < l3; i++)
        ++freq[s3[i] - 'A'];

    // count number of unique char
    for (int i = 0; i < 26; i++)
        if (freq[i] > 0)            //whose frequency is > 0, they are present
            count++;

    // soln. not possible for count > 10 as there are 10 digits in decimal syste
m
    if (count > 10)
    {
        cout << "Invalid strings";
        return 0;
    }

    // array of nodes
    node nodeArr[count];

    // store all unique char in nodeArr (in three strings)
    for (int i = 0, j = 0; i < 26; i++)
    {
        if (freq[i] > 0)
        {
            nodeArr[j].c = char(i + 'A');
            j++;
        }
    }
    return permutation(count, nodeArr, 0, s1, s2, s3);
}

// Driver function
int main()
{
    string s1 = "SUN";
    string s2 = "FUN";
    string s3 = "SWIM";

    if (solveCryptographic(s1, s2, s3) == false)
        cout << "No solution";
    return 0;
}
```

By using Backtracking Algorithm we find only one of the possible solutions. So let's approach Brute-force search to search through all of the possible permutations!

2.) Brute-force search (Exhaustive search) Algorithm

Brute-force search or exhaustive search, also known as generate and test, is a very general problem-solving technique and algorithmic paradigm that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement.

Brute-force Algorithms are exactly what they sound like – straightforward methods of solving a problem that rely on sheer computing power and trying every possibility rather than advanced techniques to improve efficiency.

Explanation of Algorithm used in the Code:

This algorithm solves a Cryptarithm by brute force. The "starting point" relationships are used to check every possible key permutation, obviating the need for rigorously solving the puzzle by hand.

➔ No. of tries reqd. can be found using the permutations

search space = permutation(10,n)

where n-> no. of unique char; since n items are being arranged out of 10

I have used the below permutation command in my code,

tries=perm(10,l4);

Next, variables are declared and the list of permutations is generated. In each iteration, one permutation is unpacked into variables representing the individual character symbols being used in the puzzle. These relationships are tested, and the program exits the loop when the solution has been found. The result is returned by printing the puzzle's

letters in the order. All that remains is to identify the three words given by the ordered string of letters.

Explanation of the Code:

Each important step and explanation of the implementation of the code is explained using the comments and is clearly depicted in the code below.

The Code:

```
// Cryptarithmetic Puzzle using C++
// Using Brute-
force search (Exhaustive search) Algorithm to search through all of the possible
 permutations
// Tanishq Dixit (2019IMT-120)
#include<iostream>
#include<string.h>
using namespace std;
int l4;
int values[10];
char letters[10];

void assign(char str[])
{
    int i,j,l;
    l=strlen(str);
    for(i=0;i<l;i++)
    {
        for(j=l4-1;j>=0;j--)
        {
            if(letters[j]==str[i])
                break;
        }
        if(j==-1)
        {
            letters[l4]=str[i];
            l4++;
        }
    }
}

int pos(char str[], char x)
{
    int i,l;
    l=strlen(str);
    for(i=0;i<l;i++)
    {
        if(str[i]==x)
            return i;
    }
    return 0;
}

void findnext()
```

```
{
    int i=l4-1;
    values[i]++;
    while(values[i]==10)
    {
        values[i]=0;
        i--;
        values[i]++;
    }
}

int fact(int n)
{
    if(n==0)
        return 1;
    else
        return n*fact(n-1);
}

int perm(int n, int r)
{
    return fact(n)/fact(n-r);
}

int main()
{
    int i,j,k,n1,n2,n3,l1,l2,l3,p,sol=0,tries;
    char w1[20],w2[20],w3[20];

    cout<<"Enter the words for Cryptarithmetic puzzle: \n";
    gets(w1);
    cout<<"+";
    gets(w2);
    cout<<"=";
    gets(w3);

    l1=strlen(w1); l2=strlen(w2); l3=strlen(w3); l4=0;
    assign(w1); assign(w2); assign(w3);

    if(l4>10)       //there can't be more than 10 unique char
    {
        cout<<"Input is wrong, try again.";
        return 0;
    }
    cout<<letters;

    //initialize values array to 0
    for(i=0;i<l4;i++)
        values[i]=0;

    p=0;
    tries=perm(10,l4);     //exhaustive search; no. of tries reqd. can be found
using the permutations
    //search space = permutation(10,n) where n-
> no. of unique char; since n items are being arranged out of 10

    for(i=1;i<=tries;i++)
    {
        findnext();
        for(j=0;j<l4;j++)
        {
            for(k=j+1;k<l4;k++)
            {
```

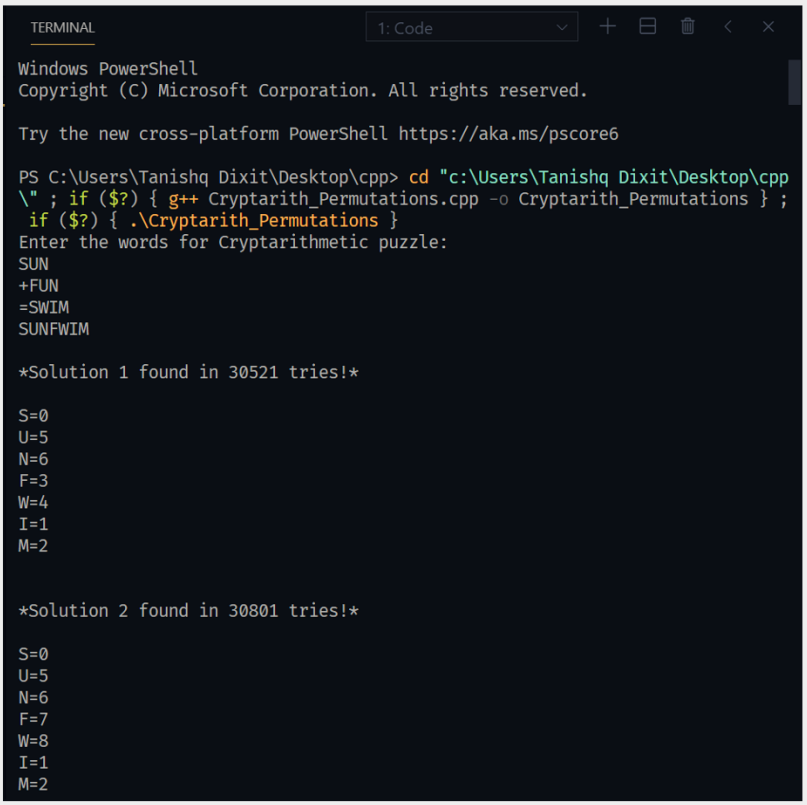
```
                if(values[j]==values[k])
                {
                    findnext();
                    j=-1;
                    break;
                }
            }
        }
        n1=0; n2=0; n3=0;
        for(j=0;j<l1;j++)
            n1=(n1*10)+values[pos(letters,w1[j])];
        for(j=0;j<l2;j++)
            n2=(n2*10)+values[pos(letters,w2[j])];
        for(j=0;j<l3;j++)
            n3=(n3*10)+values[pos(letters,w3[j])];

        if(n1+n2==n3)     //if sum of nos. formed by first two words is equal to
 the no. formed by third word

        {
            sol++;
            cout<<"\n\n*Solution "<<sol<<" found in "<<i<<" tries!*\n\n";
            for(j=0;j<l4;j++)
            {
                cout<<letters[j]<<"="<<values[j]<<"\n";
            }
        }
        if(i==tries)
        {
            if(sol==0)
                cout<<"\n\nNo solution found.";
            else
                cout<<"\n"<<sol<<" solutions found.";
        }
    }
    return 0;
}
```

RESULT

1.) Backtracking Algorithm:

```
TERMINAL                                1: Code

Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\Tanishq Dixit\Desktop\cpp> cd "c:\Users\Tanishq Dixit\Desktop\cpp
\" ; if ($?) { g++ CryptArithmetic_Puzzle.cpp -o CryptArithmetic_Puzzle } ; i
f ($?) { .\CryptArithmetic_Puzzle }

Solution found:  F = 1 I = 3 M = 4 N = 7 S = 0 U = 6 W = 2
PS C:\Users\Tanishq Dixit\Desktop\cpp>
```

2.) Brute-force Algorithm:



...

...

...

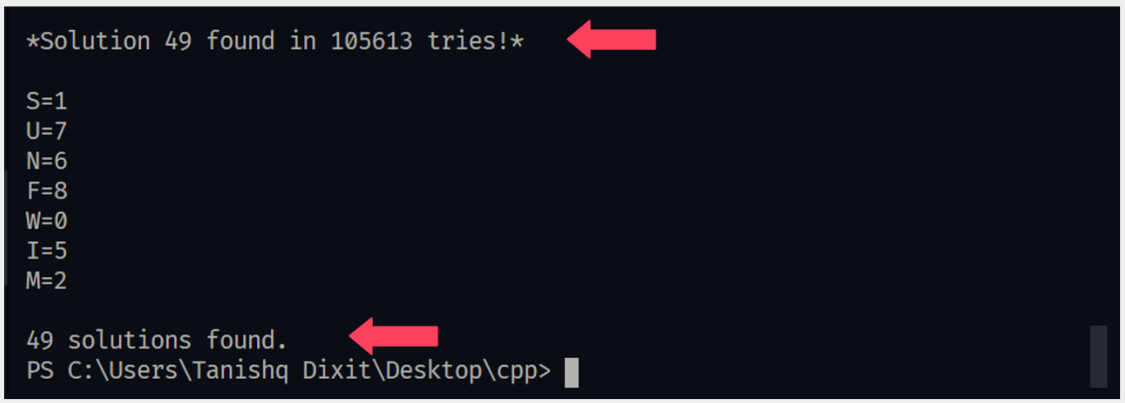
...

...

...

DISCUSSION

As you can see due to ease of presenting and reading I have attached only three result/outputs which is only due to the factor of showing that all the permutations where achieved for: *SUN + FUN = SWIM*



Solution found in total no. of tries = 105613

Total no. of solutions or permutations achieved = 49

CONCLUSION

By this project I was clearly able to understand this fun interesting problem of Cryptarithmetic. I was able to understand the optimality solutions and the no. of permutations for each given three args. through C++ implementation.

- ➔ Learned what Backtracking is and how it works especially in the field of Constraint Satisfaction Problem through Cryptarithmetic.
- ➔ Learned what the Brute-force search approach is and how it takes on the algorithm and searches through all the permutations checking the no. of tries and whether the solution is optimal or not.
- ➔ Learned about the history of Cryptarithmetic (alphametics) on behalf of the study part in AI.

LINK TO GITHUB REPOSITORY FOR THIS PROJECT

https://github.com/TanishqDixit/AI-Project_Cryptarithmetic_Alphametics

RESOURCES

https://www.geeksforgeeks.org/

https://en.wikipedia.org/

01 02 03 04

https://developers.google.com/optimization

https://www.codeproject.com/