**NAME:SAKCHI CHOUDHARY**

**REGISTRATION NO:RA1911042010035**

**BATCH:CSBS-R1**

**EXPERIMENT NO: 6**

# TOPIC: CONSTRAINTS AND SATISFACTION - CRYPT ARITHMETIC PROBLEMS

# CODE:

#include <bits/stdc++.h>

using namespace std;

// vector stores 1 corresponding to index

// number which is already assigned

// to any char, 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++)

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;

}

// sum of first two number equal to third 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

if (n == count - 1)

{

// check for all numbers not used yet

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 integer i

nodeArr[n].v = i;

// mark it as not available for other char

use[i] = 1;

// call recursive function

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 char

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

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)

count++;

// solution not possible for count greater than 10

if (count > 10)

{

cout << "Invalid strings";

return 0;

}

// array of nodes

node nodeArr[count];

// store all unique char in nodeArr

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 = "SEND";

string s2 = "MORE";

string s3 = "MONEY";

if (solveCryptographic(s1, s2, s3) == false)

cout << "No solution";

return 0;

}

**OUTPUT:**

**Text

Description automatically generated**