**Problem Name:** Verbal Arithmetic Puzzle

**Tags:** Array, Math, String, Backtracking

**Company:** Atlassian

**Level:** Hard

**Language:** C++

**Problem Statement:** Given an equation, represented by words on the left side and the result on the right side.

You need to check if the equation is solvable under the following rules:

* Each character is decoded as one digit (0 - 9).
* Every pair of different characters must map to different digits.
* Each words[i] and result are decoded as one number **without** leading zeros.
* Sum of numbers on the left side (words) will equal to the number on the right side (result).

**Input Format:**

The first line of each input contains integer n (size of list)

Second line contain n different strings

Third line contain result string

2

SEND MORE

MONEY

# Output Format:

A Boolean value “true” or “false”

# Constraints:

* 2 <= words.length <= 5
* 1 <= words[i].length, result.length <= 7
* words[i], result contain only uppercase English letters.
* The number of different characters used in the expression is at most 10.

# Example:

**Input:** words = ["SEND","MORE"], result = "MONEY"

**Output:** true

**Explanation:** Map 'S'-> 9, 'E'->5, 'N'->6, 'D'->7, 'M'->1, 'O'->0, 'R'->8, 'Y'->'2'

Such that: "SEND" + "MORE" = "MONEY", 9567 + 1085 = 10652

# Solution:

**Explanation:** We can reduce the problem to just working with the (max) ten distinct characters in the words. Each characters maps to a unique digit, so we try all possible combinations. If the sum of each digit\*base for a given combination is zero, we know that we have the answer. we count characters from words as positive and characters from result as negative: we know a combination works if the "contributions" from words and result balance out to 0.

 It is a pretty standard depth-first search/backtrack where we try matching each index of the array with a unique digit. Used digits at each stage in DFS are recorded using a bit mask (we set bit n to 1 if digit n is already used). Also, we put all the characters that can't be zero first, so whenever we are considering the digit 0 in the DFS, we can simply check if we are in the right part of the array.

For each DFS call, we check if the curSum variable is greater than sums[curIndex]. If so, we can just return false because we are looking to make curSum = 0, and no matter what happens we can no longer reach 0 with the values left in the array.

# Code:

#include <bits/stdc++.h>

using namespace std;

bool dfs(vector<int> &chars, int zeroInd, int ind, int used, int curSum, vector<int>& sums) {

    if (ind == chars.size())

        return curSum == 0;

    else if (abs(curSum) > sums[ind])

        return false;

    for (int i = 0; i < 10; i++) {

        if (((used >> i) & 1) == 1 || (i == 0 && ind < zeroInd))

            continue;

        if (dfs(chars, zeroInd, ind + 1, used | (1 << i), curSum + chars[ind]\*i, sums))

            return true;

    }

    return false;

}

bool isSolvable(vector<string>& words, string result) {

    unordered\_map<char, int> charMap;

    unordered\_map<char, bool> firstChars;

    for (auto &word : words) {

        int base = 1;

        for (int i = word.length() - 1; i >= 0; i--) {

            charMap[word[i]] += base;

            base \*= 10;

        }

        if (word.length() > 1)

            firstChars[word[0]] = true;

    }

    int base = 1;

    for (int i = result.length() - 1; i >= 0; i--) {

        charMap[result[i]] -= base;

        base \*= 10;

    }

    if (result.length() > 1)

        firstChars[result[0]] = true;

    vector<int> chars(charMap.size());

    int nonZeroInds = 0;

    int zeroInds = charMap.size() - 1;

    for (auto &p : charMap) {

        if (firstChars[p.first])

            chars[nonZeroInds++] = p.second;

        else

            chars[zeroInds--] = p.second;

    }

    sort(chars.begin(), chars.begin() + nonZeroInds);

    sort(chars.begin() + nonZeroInds, chars.end());

    vector<int> sums = vector<int>(1, 0);

    for (int i = chars.size() - 1; i >= 0; i--)

        sums.push\_back(abs(chars[i])\*9 + sums.back());

    reverse(sums.begin(), sums.end());

    return dfs(chars, nonZeroInds, 0, 0, 0, sums);

}

int main(){

 vector<string> words(5);

    string result;

    int n;

    cin>>n;

    for(int i=0;i<n; i++)

        cin>>words[i];

    cin>>result;

    cout<<isSolvable(words, result)<<"\n";

}

**Time Complexity:** O(10n)

**Space Complexity:** O(N)