1307. Verbal Arithmetic Puzzle

Problem Explanation:

This is a problem about mapping numbers to words, similar to the classic problem of cryptarithms where mathematical expressions involve letters instead of actual numbers. The idea is to assign each letter in the words and the result a unique digit, form numbers by these assignments, and check if the equality is maintained. In all these assignments, no leading zeroes are allowed, i.e., no word can start with a zero.

For example, with words = `["SEND", "MORE"]` and result = `"MONEY"`, we can assign 'S'-> 9, 'E'->5, 'N'->6, 'D'->7, 'M'->1, 'O'->0, 'R'->8, 'Y'->'2'. This forms: "SEND"(9567) + "MORE"(1085) = "MONEY"(10652), which is a valid equation.

Solution Approach:

The solution uses depth-first search (DFS) to fill the paths from all digits (0-9), tracking the ones already used. At each step, we check the current position in all words and the resulting word. We start from the least significant positions, i.e., from the rightmost characters, and go to the leftmost characters.

For each remaining character in this column for each word, we either fill in a digit that has not been used before, or if this character has been assigned a digit before, we just continue to the next character.

If we have looked at all the characters in this column for all the words, we then check if the sum of the assigned digits for this column matches the digit in the corresponding result column. If yes, we continue to the next column; if not, we stop and backtrack.

Let's walk through this in the example words = `["SEND", "MORE"]` and result = `"MONEY"`:

1. Start with the rightmost column, 'D' in 'SEND' and 'E' in 'MORE'. Let's assign 'D' -> 7 and 'E' -> 5.
2. Next, the 'N' in 'SEND' and 'R' in 'MORE'. We assign 'N' -> 6 and 'R' -> 8.
3. For the third column, 'E' in 'SEND' has been assigned, then we assign 'O' in 'MORE' -> 0, 'S' in 'SEND' -> 9, 'M' in 'MORE' -> 1.
4. The last column 'M' in 'MONEY' -> 1.
5. Since 'S', 'E', 'N', 'D', 'M', 'O', 'R', 'Y' in 'MONEY' have all been assigned and the formed numbers give a valid equation, we stop and return True.

C++ Solution:

```
cpp
class Solution {
 public:
  bool isSolvable(vector<string>& words, string result) {
    vector<bool> usedDigit(10);
    words.push_back(result);
    int rows = words.size();
    int cols = 0;
    for (const string& word : words) cols = max(cols, int(word.length()));
    return dfs(words, 0, 0, 0, cols, usedDigit);
  }

  bool dfs(vector<string>& words, int row, int col, int sum, const int cols, vector<bool>& usedDigit) {
    unordered_map<char, int> letterToDigit;

    if (col == cols) return sum == 0;
    if (row == rows) return sum % 10 == 0 && dfs(words, 0, col + 1, sum / 10, cols, usedDigit);

    string word = words[row];
    if (col >= word.length()) return dfs(words, row + 1, col, sum, cols, usedDigit);

    char letter = word[word.length() - col - 1];
    int sign = row == rows - 1 ? -1 : 1;

    if (const auto it = letterToDigit.find(letter); it != letterToDigit.end() && (it->second > 0 || col < word.length
      return dfs(words, row + 1, col, sum + sign * letterToDigit[letter], cols,  usedDigit);

    for (int digit = 0; digit < 10; ++digit)
      if (!usedDigit[digit] && (digit > 0 || col + 1 < word.length())) {
        letterToDigit[letter] = digit;
        usedDigit[digit] = true;
        if (dfs(words, row + 1, col, sum + sign * digit, cols,  usedDigit)) return true;
        usedDigit[digit] = false;
        letterToDigit.erase(letter);
      }

    return false;
  }
};
```

Note:

Due to the time constraint, solutions in other languages could not be provided.# Python Solution:

```
python
class Solution:
    def isSolvable(self, words, result):
        n = max(map(len, words + [result]))
        words = [word.rjust(n, '#') for word in words]
        result = result.rjust(n, '#')

        used = [0]*10
        dict_ = {}
        self.words = words
        self.result = result
        return self.dfs(n-1, 0, used, dict_)

    def dfs(self, i, carry, used, dict_):
        m, n = len(self.words), len(self.result)
        if i < 0:
            return carry == 0

        s = carry
        for j in range(m):
            if self.words[j][i] == '#': continue
            if self.words[j][i] not in dict_:
                for k in range(10):
                    if used[k]: continue
                    if self.words[j][i-1] != '#' and k == 0: continue
                    dict_[self.words[j][i]] = k
                    used[k] = 1
                    s += dict_[self.words[j][i]]
                    if self.dfs(i-1, s//10, used, dict_): return True
                    s -= dict_[self.words[j][i]]
                    used[k] = 0
                    del dict_[self.words[j][i]]
                return False

            s += dict_[self.words[j][i]]

        if self.result[i] not in dict_:
            if used[s%10] or (self.result[i-1] != '#' and s%10 == 0): return False
            dict_[self.result[i]] = s%10
            used[s%10] = 1
            if self.dfs(i-1, s//10, used, dict_): return True
            used[s%10] = 0
            del dict_[self.result[i]]
            return False

        return s%10 == dict_[self.result[i]] and self.dfs(i-1, s//10, used, dict_)


# Test the solution
words = ["SEND", "MORE"]
result = "MONEY"
sol = Solution()
print(sol.isSolvable(words, result))  # Should print True
```

JavaScript Solution

```
javascript
var isSolvable=function(words, result) {
    const map = new Map()
    const used = new Set()
    const isFirst = new Set()
    for (let word of words.concat([result])) for (let i = 0; i < word.length; ++i) {
        if (i === 0 && word.length > 1) isFirst.add(word[i])
        if (!map.has(word[i])) map.set(word[i], [])
        map.get(word[i]).push([words.indexOf(word), word.length - 1 - i])
    }
    words = words.map(word => [...word].reverse().join(''))

    result = [...result].reverse().join('')
    const dfs = (str, i, sum) => {
        if (i === 10) return sum === 0
        if (sum % (1 << 10) != (str[i] || 0) - '0'.charCodeAt()) return false
        if (i === str.length) return sum === 0
        for (let j = (isFirst.has(result[i]) ? 1 : 0); j < 10; ++j) {
            if (used.has(j)) continue
            const next = []
            let carry = sum >> 10
            for (let [idx, exp] of map.get(result[i]) || []) {
                if (words[idx][exp] === undefined) next.push(j << exp)
                else {
                    if ((carry & 1) != (words[idx][exp].charCodeAt() + j >> exp & 1)) return false
                    words[idx] = words[idx].substring(0, exp) + String.fromCharCode((words[idx][exp].charCodeAt() + j
                }
                carry >>= 1
            }
            next.push(j << i)
            used.add(j)
            if (dfs(str, i + 1, sum + (carry << 10) + next.reduce((a, b) => a + b, 0))) return true
            used.delete(j)
            for (let k = next.length - 1; k >= 0; --k) {
                let [idx, exp] = map.get(result[i])[k] || []
                if (words[idx] === undefined) continue
                words[idx] = words[idx].substring(0, exp) + String.fromCharCode((words[idx].substring(exp, exp + 1).c
            }
        }
        return false
    }
    return dfs(result, 0, 0)
};

words = ["ACCA", "DDA"];
result = "ADDC";
console.log(words, result, isSolvable(words, result));
```