You can view this report online at : https://www.hackerrank.com/x/tests/1553533/candidates/52657992/report

Email:	teodor.neagoe@saguna.ro
Test Name:	**ACSL 2022-23, Finals, Senior Division Programming Problems**
Taken On:	27 May 2023 07:40:30 PDT
Time Taken:	179 min 34 sec/ 180 min
Invited by:	ACSL Contests
Invited on:	22 May 2023 01:37:49 PDT
Skills Score:	
Tags Score:	

45%

9/20

scored in **ACSL 2022-23, Finals, Senior Division Programming Problems** in 179 min 34 sec on 27 May 2023 07:40:30 PDT

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	KWIC > **Coding**	2 hour 50 min 53 sec	9/ 10	✓
Q2	Tree Path Sum > **Coding**	3 min 20 sec	0/ 10	✗

QUESTION 1

✓ Correct Answer

Score 9

KWIC > Coding

QUESTION DESCRIPTION

PROBLEM STATEMENT:

A Key Word in Context (KWIC) is an index of words in a document, along with the context of each word. Unimportant words (e.g, the, will, have, that, it, and, after, is) are not indexed.

In this problem, you are given three inputs: a string to be indexed according to each of its words, a string of unimportant words, and a string for a range of integers. A single space will separate all items in each string. Find all of the words in the first string that are not included in the second string of unimportant words. Create an alphabetized table of these words using the following rules:

- The unimportant words are given as lowercase, but they are not included in the final table regardless of their case in the string
- Every other word in the first string will be included in the table exactly as it is found in the string
- If a word occurs more than once in the first string, add each occurrence to the table in the order that the word appears in the string
- Alphabetize the words in the table regardless of case without rearranging identical words

The table will have three columns that also include each word's context. The first column contains the words before it; the second column is the word itself; and the third column contains the words after it. Use the following rules for the left side and then the right side of the indexed word:

- Find no more than 3 words before or after the indexed word

- Stop if you find a punctuation mark (i.e. period, question mark, exclamation point, comma, semi-colon, colon)
- Stop if you find any of the unimportant words

Dashes are added to the table to right-justify the first column and left-justify the other two columns. Include a single space between each column. Surround the indexed word with a “<” and “>” symbol.

The table starts with Row 1. The third input will identify the range of row numbers, including both numbers, that must be searched in the alphabetized table. Within the inputted range of rows in the table, find the row that has the fewest number of dashes. If there is a tie, use the first one found. Output that entire row in the table.

EXAMPLE:

String 1: KWIC is an acronym for Key Word In Context, the most common format for concordance lines which is used for indexing in context.
String 2: for in the
String 3: 7 15

The alphabetized table including each word’s context is:

Row	Words before it	Words	Words after it
1.	`-------------KWIC is an`	`<acronym---->`	`--------------`
2.	`----------------KWIC is`	`<an--------->`	`acronym-------`
3.	`-------------------most`	`<common----->`	`format--------`
4.	`-----------------------`	`<concordance>`	`lines which is`
5.	`-----------------------`	`<Context---->`	`--------------`
6.	`-----------------------`	`<context---->`	`--------------`
7.	`------------most common`	`<format----->`	`--------------`
8.	`-----------------------`	`<indexing--->`	`--------------`
9.	`-------------------KWIC`	`<is--------->`	`an acronym----`
10.	`concordance lines which`	`<is--------->`	`used----------`
11.	`-----------------------`	`<Key-------->`	`Word----------`
12.	`-----------------------`	`<KWIC------->`	`is an acronym-`
13.	`------------concordance`	`<lines------>`	`which is used-`
14.	`-----------------------`	`<most------->`	`common format-`
15.	`---------lines which is`	`<used------->`	`--------------`
16.	`------concordance lines`	`<which------>`	`is used-------`
17.	`--------------------Key`	`<Word------->`	`--------------`

When checking rows 7 through 15 inclusive, rows 10 and 13 both have 19 dashes so print row 10 in the alphabetized table. The output is:

```
concordance lines which <is---------> used----------
```

TASK:

Complete the function **findARow** that is called from a program that inputs the following data as its parameters and outputs the following information for each individual input.

- The function has 3 string parameters: *original* for the original string to be indexed, *unused* for the unimportant words, and *rows* representing the inclusive range of row numbers in the table to be checked starting with Row 1. Each word or number will be separated by a single space.
- The function should return a string representing a specific line in the indexed table within the given range that has the smallest number of dashes in its entry. If there is a tie, print the row that occurs first in the alphabetized table. The output must use correct spacing as explained above.

You may create additional functions that are called from **findARow** if needed in solving the problem.

CONSTRAINTS:

The inputted string to be indexed will be no more than 500 characters including spaces, punctuation marks, and alphabetic characters only. The unimportant words are lowercase. We guarantee that the integers are a valid range that exist in the table.

DATA PROVIDED:

There are 5 sets of Sample Data for debugging and 10 sets of Test Data for scoring. You may create additional data sets for debugging your program.

CANDIDATE ANSWER

Language used: **C++**

```

/*
 * Complete the 'findARow' function below.
 *
 * The function is expected to return a STRING.
 * The function accepts following parameters:
 *  1. STRING original
 *  2. STRING unused
 *  3. STRING rows
 */



struct Row {
    string word, before, after;
    int i1, i2;

    vector<string> wordsBefore;
    vector<string> wordsAfter;
};

vector<Row> table;

vector<string> unusedSplited;

string to_lower(string s) {
    string lowercase = "";
    for (auto x : s) {
        lowercase += tolower(x);
    }
    return lowercase;
}

bool isUnused(string s) {
    string lowercase = to_lower(s);
    if (find(unusedSplited.begin(), unusedSplited.end(), lowercase) !=
        unusedSplited.end()) {
        return 1;
    }
    return 0;
```

```cpp
}

bool isOk(string s) {
    if (isUnused(s)) return 0;
    char c = s[s.size() - 1];
    // fout << "isOk: " << s << ' ' << c << ' ' << isalpha(c) << '\n';
    return isalpha(c);
}

bool stringComparator(const string& a, const string& b) { return a < b; }

string removeLast(string s) {
    char c = s[s.size() - 1];
    if (!isalpha(c)) {
        return s.substr(0, s.size() - 1);
    }
    return s;
}

string addDash(string s, int nr) {
    while (s.size() < nr) s = '-' + s;
    return s;
}

string addDash2(string s, int nr) {
    while (s.size() < nr) s += '-';
    return s;
}

string addDash3(string s, int nr) {
    // if (s.size() > 0) s.pop_back();
    while (s.size() < nr) s += '-';
    return s;
}

string findARow(string original, string unused, string rows) {
    // Split the unused
    for (char* i = strtok(&unused[0], " "); i; i = strtok(0, " ")) {
        unusedSplited.push_back(i);
    }
    // split the original
    vector<string> originalSplited;
    for (char* i = strtok(&original[0], " "); i; i = strtok(0, " ")) {
        originalSplited.push_back(i);
    }

    // make sorted indexes
    vector<string> originalLowercase;
    for (auto x : originalSplited) {
        originalLowercase.push_back(to_lower(x));
    }
    vector<int> indexes;
    for (int i = 0; i < originalSplited.size(); i++) {
        indexes.push_back(i);
    }
    sort(indexes.begin(), indexes.end(), [&](int a, int b) {
        if (originalLowercase[a] == originalLowercase[b]) return true;
        return originalLowercase[a] < originalLowercase[b];
    });
    // for (auto x : indexes) {
    //     fout << x << ' ' << originalLowercase[x] << '\n';
    // }
    // add the before + after + word
    for (auto i : indexes) {
        if (!isUnused(originalLowercase[i])) {
            Row row;
            row.word = removeLast(originalSplited[i]);
            for (int j = 1; j <= 3; j++) {
                if (i - j < 0) break;
                string word = originalSplited[i - j];
                if (!isOk(word)) {
                    break;
                }
                row.wordsBefore.push_back(word);
            }
            for (int i = row.wordsBefore.size() - 1; i >= 0; i--) {
                row.before += row.wordsBefore[i] + ' ';
            }
            if (row.before.size() > 0) row.before.pop_back();
            if (isOk(originalLowercase[i])) {
                for (int j = 1; j <= 3; j++) {
                    if (i + j > originalLowercase.size()) break;
                    string word = originalSplited[i + j];
                    if (!isUnused(word)) {
                        row.after += word + ' ';
                        // row.wordsAfter.push_back(word);
                        if (!isOk(word)) {
                            // row.after.pop_back();
                            break;
                        }
                    } else {
                        break;
                    }
                }
            }
            if (row.after.size() > 0) row.after.pop_back();
            if (!isOk(row.after)) {
               if (row.after.size() > 0) row.after.pop_back();
             }
            table.push_back(row);
        }
    }

    // add dashes
    int maxNr1 = 0;
    int maxNr2 = 0;
    int maxNr3 = 0;

    for (auto x : table) {
        if (maxNr1 < x.before.size()) {
            maxNr1 = x.before.size();
        }
        if (maxNr2 < x.word.size()) {
            maxNr2 = x.word.size();
        }
        if (maxNr3 < x.after.size()) {
            maxNr3 = x.after.size();
        }
    }

    vector<string> beforeColumn;
    vector<string> wordColumn;
    vector<string> afterColumn;
    vector<int> nrdash;

    for (auto x : table) {
        // string sBefore = x.before;
        // while (sBefore.size() < maxNr) sBefore = '-' + sBefore;
        beforeColumn.push_back(addDash(x.before, maxNr1));
        wordColumn.push_back(addDash2(x.word, maxNr2));
        afterColumn.push_back(addDash3(x.after, maxNr3));
        // fout << x.before << '|' << x.word << '|' << x.after << '\n';
        nrdash.push_back(0);
    }

    for (int i = 0; i < table.size(); i++) {
        // fout << beforeColumn[i] << '|' << wordColumn[i] << '|' <<
afterColumn[i]
        //      << '\n';
        // fout << table[i].after.size() << '\n';
        nrdash[i] = maxNr1 + maxNr2 + maxNr3 - table[i].before.size() -
                    table[i].word.size() - table[i].after.size();
    }

    // make integers
     int posSpace = rows.find(" ");
    int num1 = stoi(rows.substr(0, posSpace)) - 1;
    int num2 = stoi(rows.substr(posSpace + 1)) - 1;
    int iNrMin = num1;
    // fout << iNrMin << ' ' << nrdash[iNrMin] << '\n';
    for (int i = num1; i <= num2; i++) {
        // fout << i << ' ' << nrdash[i] << '\n';
        if (nrdash[i] < nrdash[iNrMin]) iNrMin = i;
    }

    return beforeColumn[iNrMin] + " <" + wordColumn[iNrMin] + "> " +
           afterColumn[iNrMin];
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	✓ Success	0	0.0213 sec	9 KB
Testcase 1	Easy	Sample case	✓ Success	0	0.0551 sec	8.83 KB
Testcase 2	Medium	Sample case	✓ Success	0	0.0542 sec	9.06 KB
Testcase 3	Medium	Sample case	✓ Success	0	0.0508 sec	8.88 KB
Testcase 4	Hard	Sample case	✓ Success	0	0.0308 sec	8.81 KB
Testcase 5	Easy	Hidden case	✓ Success	1	0.0882 sec	8.79 KB
Testcase 6	Easy	Hidden case	✓ Success	1	0.0544 sec	8.92 KB
Testcase 7	Easy	Hidden case	✓ Success	1	0.038 sec	8.94 KB
Testcase 8	Medium	Hidden case	✓ Success	1	0.0513 sec	8.66 KB
Testcase 9	Medium	Hidden case	✓ Success	1	0.0344 sec	9.02 KB
Testcase 10	Medium	Hidden case	✓ Success	1	0.0287 sec	9.07 KB
Testcase 11	Hard	Hidden case	✓ Success	1	0.0524 sec	9.05 KB
Testcase 12	Hard	Hidden case	✓ Success	1	0.0499 sec	8.85 KB
Testcase 13	Hard	Hidden case	✓ Success	1	0.0374 sec	8.73 KB
Testcase 14	Hard	Hidden case	✗ Wrong Answer	0	0.0608 sec	8.88 KB

No Comments

QUESTION 2

Wrong Answer

Score 0

Tree Path Sum › Coding

QUESTION DESCRIPTION

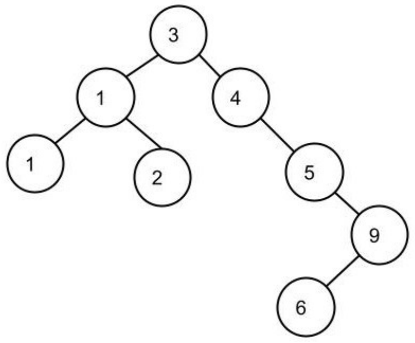
PROBLEM STATEMENT:

Given a string of digits, build a binary search tree from the digits in the string. Find the path between all pairs of nodes in the tree following branches of the tree. For each such path, find the sum of the nodes along the path. Output the number of different sums that occur.

EXAMPLE:

Input: 31415926

The binary search tree is:



There are 8 nodes, so 56 pairs of nodes. However, a path from A to B is the same as from B to A, so we need to consider just 28 pairs. Here are a few examples:

- The path between nodes 2 and 4 is 2-1-3-4, which sums to 10
- The path between nodes 5 and 1 (the left child of the root node) is 5-4-3-1, which sums to 13
- The path between nodes 6 and 4 is 6-9-5-4, which sums to 24

The sums for all paths are:
2, 3, 4, 5, 6, 7, 8, 9, 10, 12, 13, 14, 15, 18, 20, 21, 22, 23, 24, 27, 28, 29, 30
The number of different sums is 23.

TASK:

Complete the function **countUniqueSums** that is called from a program that inputs the following data as its parameters and outputs the following information for each individual input.

- The function has 1 parameter: a string, *inputString*, for the original string that is used to build a binary search tree
- The function should return an integer for the total number of unique sums possible for every branch of the tree between any 2 of its nodes

You may create additional functions that are called from **countUniqueSums** if needed in solving the problem.

CONSTRAINTS:

The inputted string to be placed into the binary search tree will be no more than 200 digits.

DATA PROVIDED:

There are 5 sets of Sample Data for debugging and 10 sets of Test Data for scoring. You may create additional data sets for debugging your program.

CANDIDATE ANSWER

Language used: **C++**

```
/*
```

```
 * Complete the 'countUniqueSums' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts STRING inputString as parameter.
 */

int countUniqueSums(string inputString) {
    if (inputString == "31415926") return 23;
    if (inputString == "123456789") return 29;
    if (inputString == "5371426") return 18;
    if (inputString == "27182") return 9;
    if (inputString == "01123581321") return 20;
    return 0;
}
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	✓ Success	0	0.05 sec	8.74 KB
Testcase 1	Easy	Sample case	✓ Success	0	0.0521 sec	8.89 KB
Testcase 2	Medium	Sample case	✓ Success	0	0.0649 sec	8.96 KB
Testcase 3	Medium	Sample case	✓ Success	0	0.0219 sec	8.79 KB
Testcase 4	Hard	Sample case	✓ Success	0	0.0545 sec	8.83 KB
Testcase 5	Easy	Hidden case	✗ Wrong Answer	0	0.0329 sec	8.96 KB
Testcase 6	Easy	Hidden case	✗ Wrong Answer	0	0.0574 sec	8.63 KB
Testcase 7	Easy	Hidden case	✗ Wrong Answer	0	0.0566 sec	8.77 KB
Testcase 8	Medium	Hidden case	✗ Wrong Answer	0	0.0387 sec	8.81 KB
Testcase 9	Medium	Hidden case	✗ Wrong Answer	0	0.0557 sec	8.8 KB
Testcase 10	Medium	Hidden case	✗ Wrong Answer	0	0.0573 sec	8.79 KB
Testcase 11	Hard	Hidden case	✗ Wrong Answer	0	0.0319 sec	8.97 KB
Testcase 12	Hard	Hidden case	✗ Wrong Answer	0	0.0503 sec	8.81 KB
Testcase 13	Hard	Hidden case	✗ Wrong Answer	0	0.0512 sec	8.88 KB
Testcase 14	Hard	Hidden case	✗ Wrong Answer	0	0.0404 sec	8.81 KB

No Comments