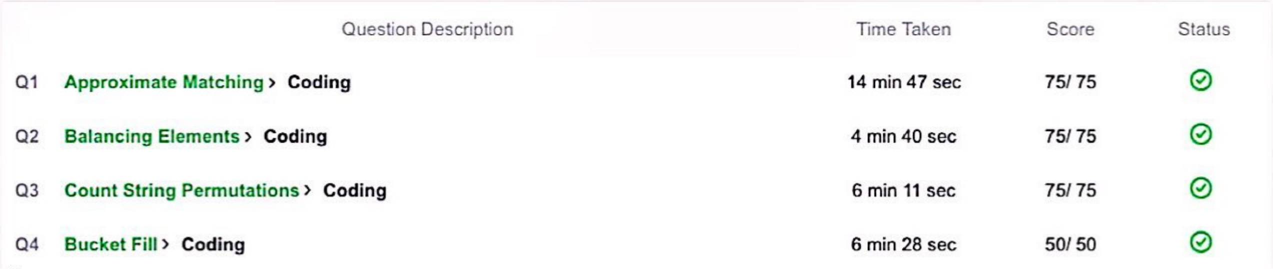
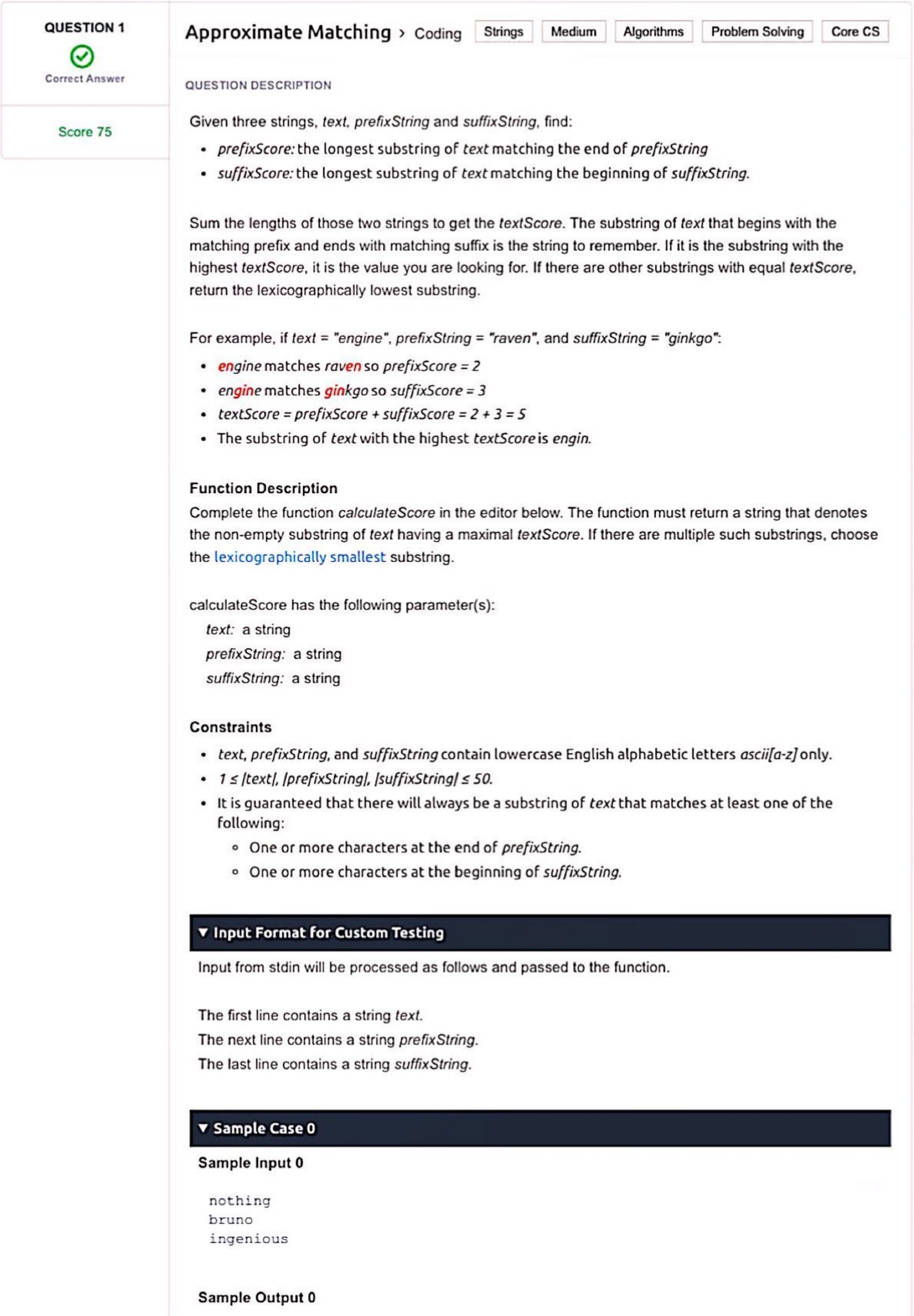


	Question Description	Time Taken	Score	Status
Q1	Approximate Matching > Coding	14 min 47 sec	75/ 75	✓
Q2	Balancing Elements > Coding	4 min 40 sec	75/ 75	✓
Q3	Count String Permutations > Coding	6 min 11 sec	75/ 75	✓
Q4	Bucket Fill > Coding	6 min 28 sec	50/ 50	✓



QUESTION 1

✓

Correct Answer

Score 75

Approximate Matching > Coding

QUESTION DESCRIPTION

Given three strings, *text*, *prefixString* and *suffixString*, find:

- *prefixScore:* the longest substring of *text* matching the end of *prefixString*
- *suffixScore:* the longest substring of *text* matching the beginning of *suffixString*.

Sum the lengths of those two strings to get the *textScore*. The substring of *text* that begins with the matching prefix and ends with matching suffix is the string to remember. If it is the substring with the highest *textScore*, it is the value you are looking for. If there are other substrings with equal *textScore*, return the lexicographically lowest substring.

For example, if *text* = *"engine"*, *prefixString* = *"raven"*, and *suffixString* = *"ginkgo"*:

- *engine* matches *raven* so *prefixScore = 2*
- *engine* matches *ginkgo* so *suffixScore = 3*
- *textScore = prefixScore + suffixScore = 2 + 3 = 5*
- The substring of *text* with the highest *textScore* is *engin*.

Function Description

Complete the function *calculateScore* in the editor below. The function must return a string that denotes the non-empty substring of *text* having a maximal *textScore*. If there are multiple such substrings, choose the lexicographically smallest substring.

calculateScore has the following parameter(s):

text: a string
prefixString: a string
suffixString: a string

Constraints

- *text*, *prefixString*, and *suffixString* contain lowercase English alphabetic letters *ascii[a-z]* only.
- *1 ≤ |text|, |prefixString|, |suffixString| ≤ 50.*
- It is guaranteed that there will always be a substring of *text* that matches at least one of the following:
  - One or more characters at the end of *prefixString*.
  - One or more characters at the beginning of *suffixString*.

▼ Input Format for Custom Testing

Input from stdin will be processed as follows and passed to the function.

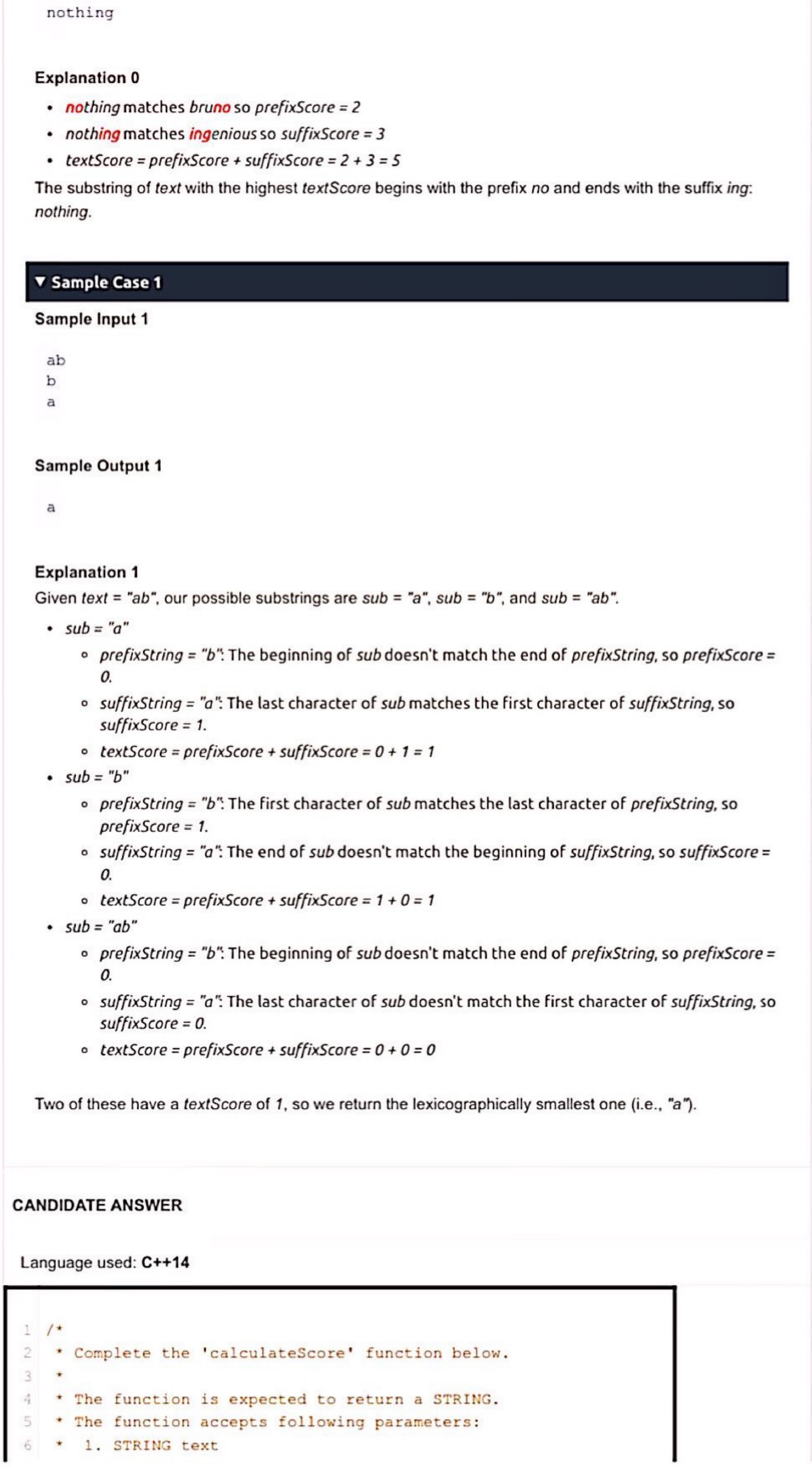
The first line contains a string *text*.
The next line contains a string *prefixString*.
The last line contains a string *suffixString*.

▼ Sample Case 0

Sample Input 0

```
nothing
bruno
ingenious
```

Sample Output 0



```
nothing
```

Explanation 0

- *nothing* matches *bruno* so *prefixScore = 2*
- *nothing* matches *ingenious* so *suffixScore = 3*
- *textScore = prefixScore + suffixScore = 2 + 3 = 5*

The substring of *text* with the highest *textScore* begins with the prefix *no* and ends with the suffix *ing*: *nothing*.

▼ Sample Case 1

Sample Input 1

```
ab
b
a
```

Sample Output 1

```
a
```

Explanation 1

Given *text* = *"ab"*, our possible substrings are *sub* = *"a"*, *sub* = *"b"*, and *sub* = *"ab"*.

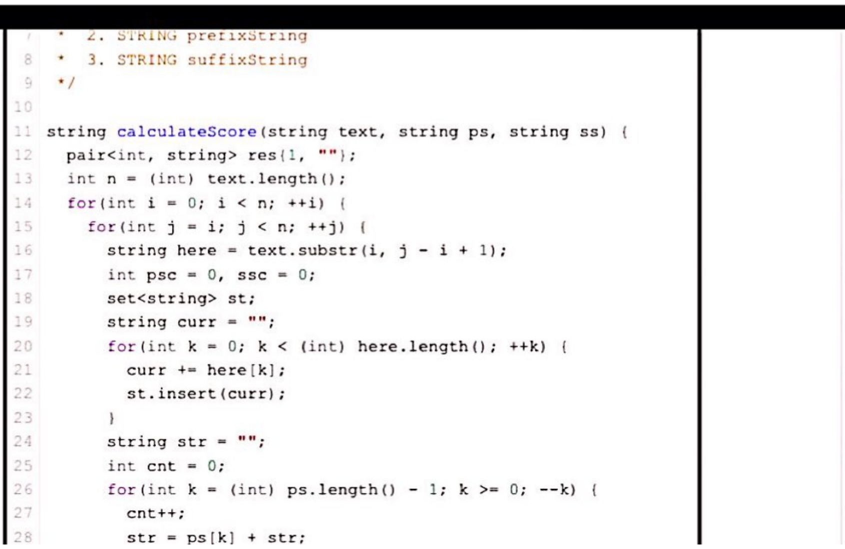
- *sub = "a"*
  - *prefixString = "b"*: The beginning of *sub* doesn't match the end of *prefixString*, so *prefixScore = 0*.
  - *suffixString = "a"*: The last character of *sub* matches the first character of *suffixString*, so *suffixScore = 1*.
  - *textScore = prefixScore + suffixScore = 0 + 1 = 1*
- *sub = "b"*
  - *prefixString = "b"*: The first character of *sub* matches the last character of *prefixString*, so *prefixScore = 1*.
  - *suffixString = "a"*: The end of *sub* doesn't match the beginning of *suffixString*, so *suffixScore = 0*.
  - *textScore = prefixScore + suffixScore = 1 + 0 = 1*
- *sub = "ab"*
  - *prefixString = "b"*: The beginning of *sub* doesn't match the end of *prefixString*, so *prefixScore = 0*.
  - *suffixString = "a"*: The last character of *sub* doesn't match the first character of *suffixString*, so *suffixScore = 0*.
  - *textScore = prefixScore + suffixScore = 0 + 0 = 0*

Two of these have a *textScore* of *1*, so we return the lexicographically smallest one (i.e., *"a"*).

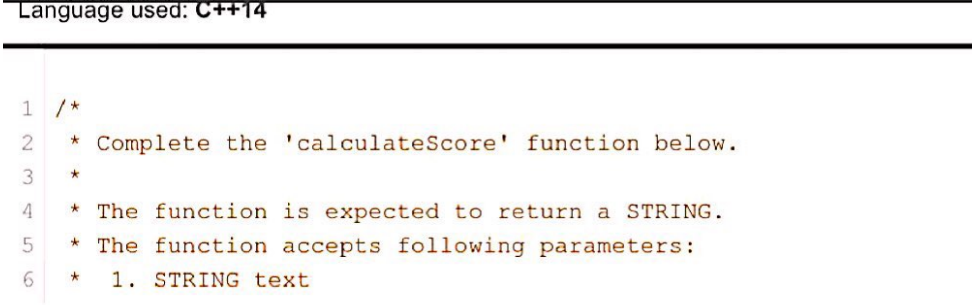
CANDIDATE ANSWER

Language used: **C++14**

```
/*
 * Complete the 'calculateScore' function below.
 *
 * The function is expected to return a STRING.
 * The function accepts following parameters:
 *  1. STRING text
```

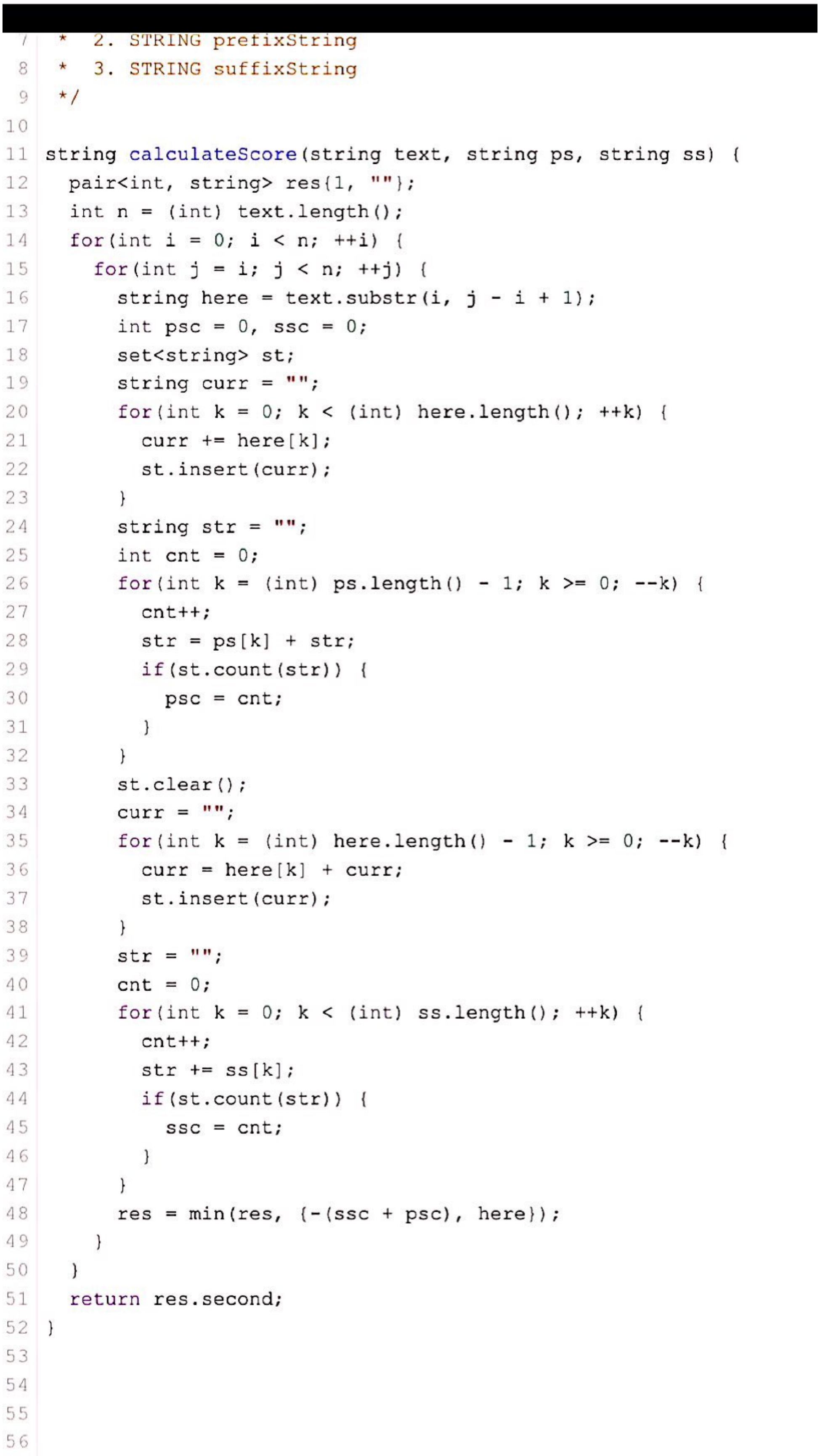


```
 *  2. STRING prefixString
 *  3. STRING suffixString
 */

string calculateScore(string text, string ps, string ss) {
  pair<int, string> res{1, ""};
  int n = (int) text.length();
  for(int i = 0; i < n; ++i) {
    for(int j = i; j < n; ++j) {
      string here = text.substr(i, j - i + 1);
      int psc = 0, ssc = 0;
      set<string> st;
      string curr = "";
      for(int k = 0; k < (int) here.length(); ++k) {
        curr += here[k];
        st.insert(curr);
      }
      string str = "";
      int cnt = 0;
      for(int k = (int) ps.length() - 1; k >= 0; --k) {
        cnt++;
        str = ps[k] + str;
```

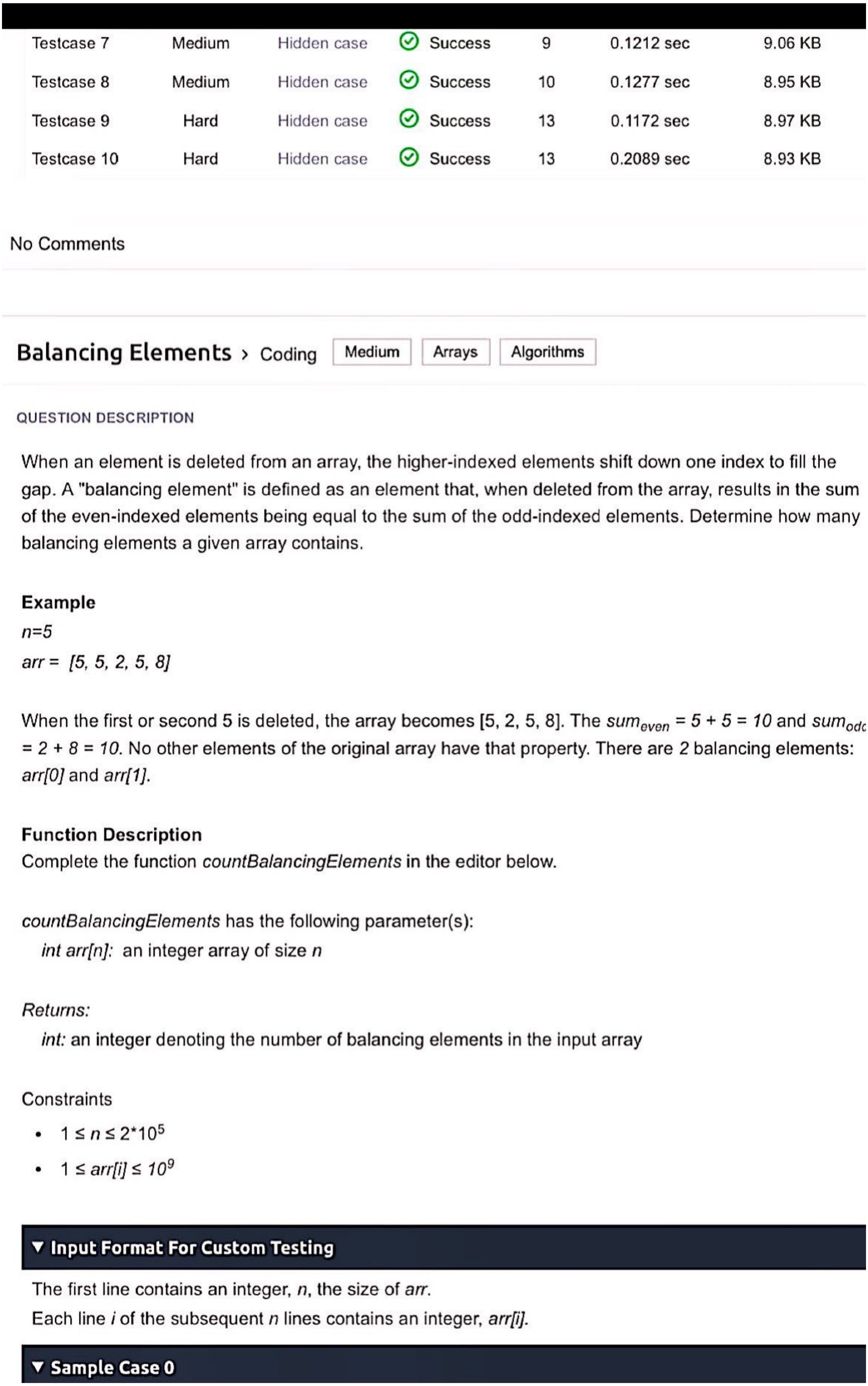


Language used: **C++14**

```
/*
 * Complete the 'calculateScore' function below.
 *
 * The function is expected to return a STRING.
 * The function accepts following parameters:
 *  1. STRING text
```



```
 *  2. STRING prefixString
 *  3. STRING suffixString
 */

string calculateScore(string text, string ps, string ss) {
  pair<int, string> res{1, ""};
  int n = (int) text.length();
  for(int i = 0; i < n; ++i) {
    for(int j = i; j < n; ++j) {
      string here = text.substr(i, j - i + 1);
      int psc = 0, ssc = 0;
      set<string> st;
      string curr = "";
      for(int k = 0; k < (int) here.length(); ++k) {
        curr += here[k];
        st.insert(curr);
      }
      string str = "";
      int cnt = 0;
      for(int k = (int) ps.length() - 1; k >= 0; --k) {
        cnt++;
        str = ps[k] + str;
        if(st.count(str)) {
          psc = cnt;
        }
      }
      st.clear();
      curr = "";
      for(int k = (int) here.length() - 1; k >= 0; --k) {
        curr = here[k] + curr;
        st.insert(curr);
      }
      str = "";
      cnt = 0;
      for(int k = 0; k < (int) ss.length(); ++k) {
        cnt++;
        str += ss[k];
        if(st.count(str)) {
          ssc = cnt;
        }
      }
      res = min(res, {-(ssc + psc), here});
    }
  }
  return res.second;
}
```



Testcase 7	Medium	Hidden case	✓ Success	9	0.1212 sec	9.06 KB
Testcase 8	Medium	Hidden case	✓ Success	10	0.1277 sec	8.95 KB
Testcase 9	Hard	Hidden case	✓ Success	13	0.1172 sec	8.97 KB
Testcase 10	Hard	Hidden case	✓ Success	13	0.2089 sec	8.93 KB

No Comments

Balancing Elements › Coding

Medium Arrays Algorithms

QUESTION DESCRIPTION

When an element is deleted from an array, the higher-indexed elements shift down one index to fill the gap. A "balancing element" is defined as an element that, when deleted from the array, results in the sum of the even-indexed elements being equal to the sum of the odd-indexed elements. Determine how many balancing elements a given array contains.

Example

n=5

arr = [5, 5, 2, 5, 8]

When the first or second 5 is deleted, the array becomes [5, 2, 5, 8]. The sum_{even} = *5 + 5 = 10* and sum_{odc} = *2 + 8 = 10*. No other elements of the original array have that property. There are *2* balancing elements: *arr[0]* and *arr[1]*.

Function Description

Complete the function *countBalancingElements* in the editor below.

countBalancingElements has the following parameter(s):

int arr[n]: an integer array of size *n*

Returns:

int: an integer denoting the number of balancing elements in the input array

Constraints

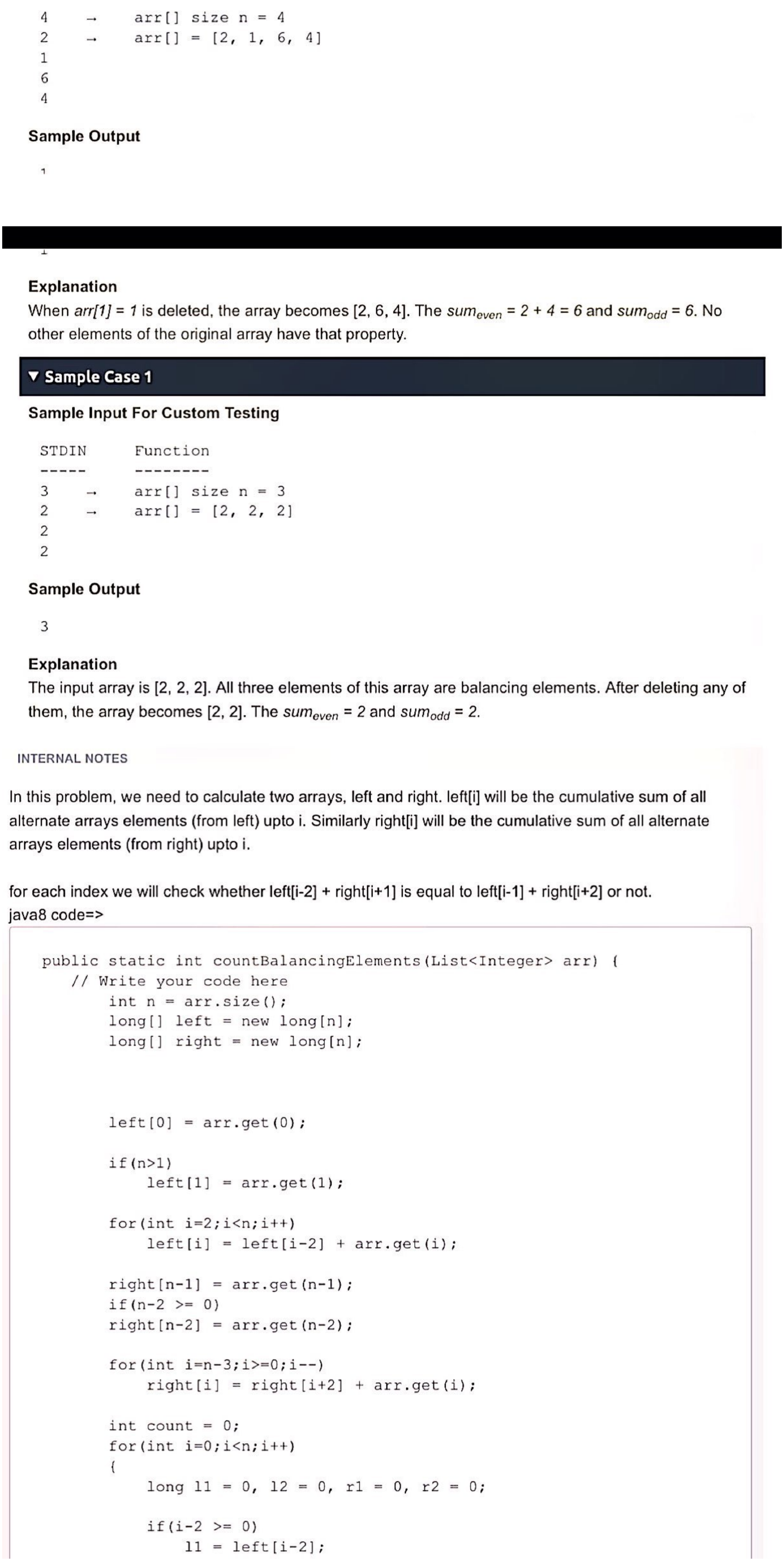
- $1 \le n \le 2*10^5$
- $1 \le arr[i] \le 10^9$

▼ Input Format For Custom Testing

The first line contains an integer, *n*, the size of *arr*.

Each line *i* of the subsequent *n* lines contains an integer, *arr[i]*.

▼ Sample Case 0



```
4     →    arr[] size n = 4
2     →    arr[] = [2, 1, 6, 4]
1
6
4
```

Sample Output

```
1
```

[redacted]

```
1
```

Explanation

When *arr[1] = 1* is deleted, the array becomes [2, 6, 4]. The *sum*$_{even}$ = *2 + 4 = 6* and *sum*$_{odd}$ = *6*. No other elements of the original array have that property.

▼ Sample Case 1

Sample Input For Custom Testing

```
STDIN      Function
-----      --------
3     →    arr[] size n = 3
2     →    arr[] = [2, 2, 2]
2
2
```

Sample Output

```
3
```

Explanation

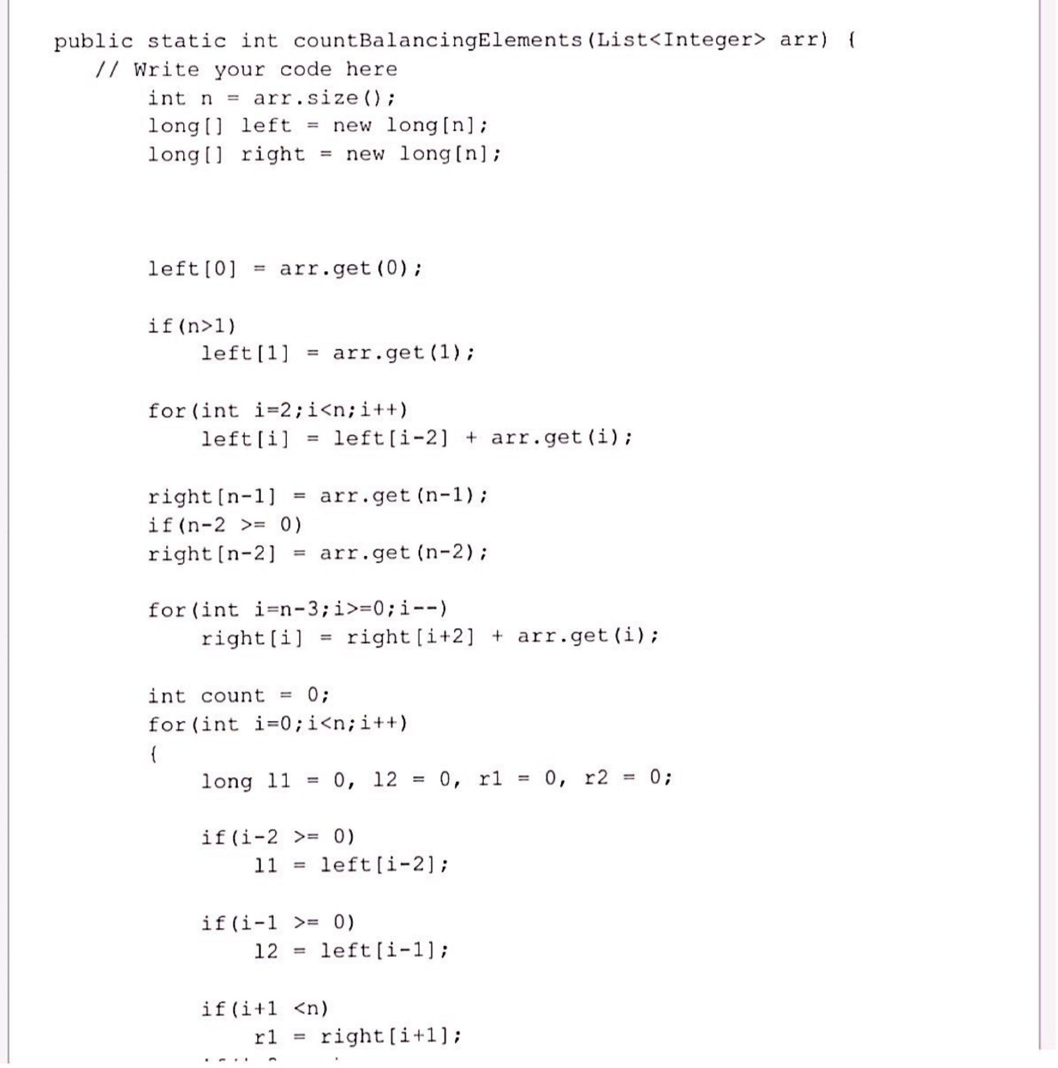
The input array is [2, 2, 2]. All three elements of this array are balancing elements. After deleting any of them, the array becomes [2, 2]. The *sum*$_{even}$ = *2* and *sum*$_{odd}$ = *2*.

INTERNAL NOTES

In this problem, we need to calculate two arrays, left and right. left[i] will be the cumulative sum of all alternate arrays elements (from left) upto i. Similarly right[i] will be the cumulative sum of all alternate arrays elements (from right) upto i.

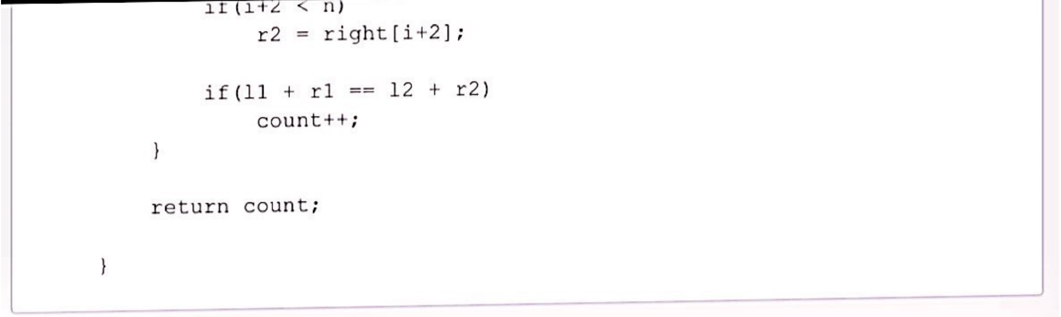
for each index we will check whether left[i-2] + right[i+1] is equal to left[i-1] + right[i+2] or not.
java8 code=>

```
public static int countBalancingElements(List<Integer> arr) {
   // Write your code here
       int n = arr.size();
       long[] left = new long[n];
       long[] right = new long[n];



       left[0] = arr.get(0);

       if(n>1)
           left[1] = arr.get(1);

       for(int i=2;i<n;i++)
           left[i] = left[i-2] + arr.get(i);

       right[n-1] = arr.get(n-1);
       if(n-2 >= 0)
       right[n-2] = arr.get(n-2);

       for(int i=n-3;i>=0;i--)
           right[i] = right[i+2] + arr.get(i);

       int count = 0;
       for(int i=0;i<n;i++)
       {
           long l1 = 0, l2 = 0, r1 = 0, r2 = 0;

           if(i-2 >= 0)
               l1 = left[i-2];
```

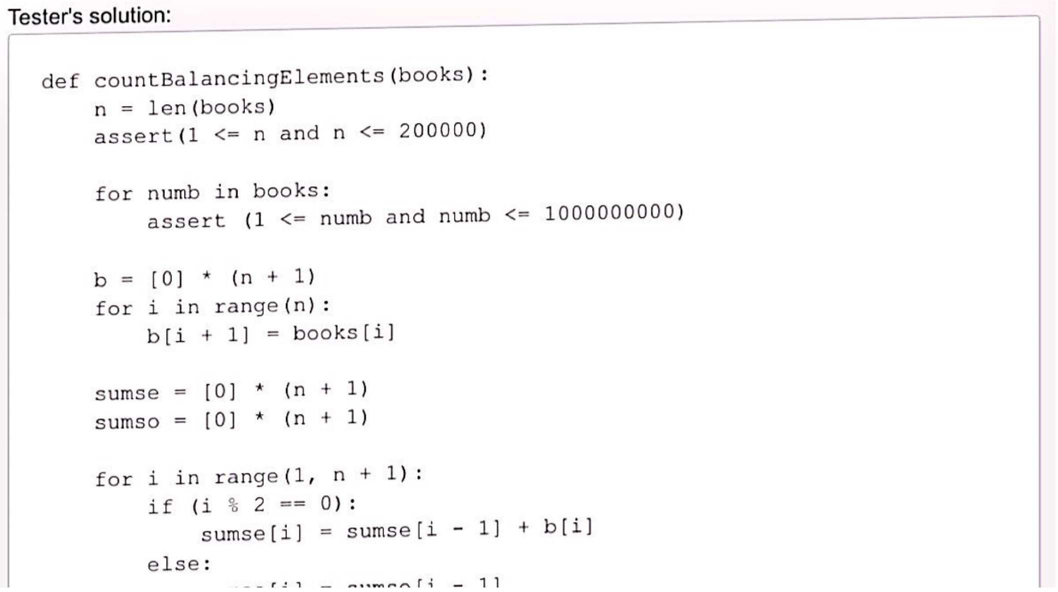


```
public static int countBalancingElements(List<Integer> arr) {
   // Write your code here
       int n = arr.size();
       long[] left = new long[n];
       long[] right = new long[n];



       left[0] = arr.get(0);

       if(n>1)
           left[1] = arr.get(1);

       for(int i=2;i<n;i++)
           left[i] = left[i-2] + arr.get(i);

       right[n-1] = arr.get(n-1);
       if(n-2 >= 0)
       right[n-2] = arr.get(n-2);

       for(int i=n-3;i>=0;i--)
           right[i] = right[i+2] + arr.get(i);

       int count = 0;
       for(int i=0;i<n;i++)
       {
           long l1 = 0, l2 = 0, r1 = 0, r2 = 0;

           if(i-2 >= 0)
               l1 = left[i-2];

           if(i-1 >= 0)
               l2 = left[i-1];

           if(i+1 <n)
               r1 = right[i+1];
```

[redacted]



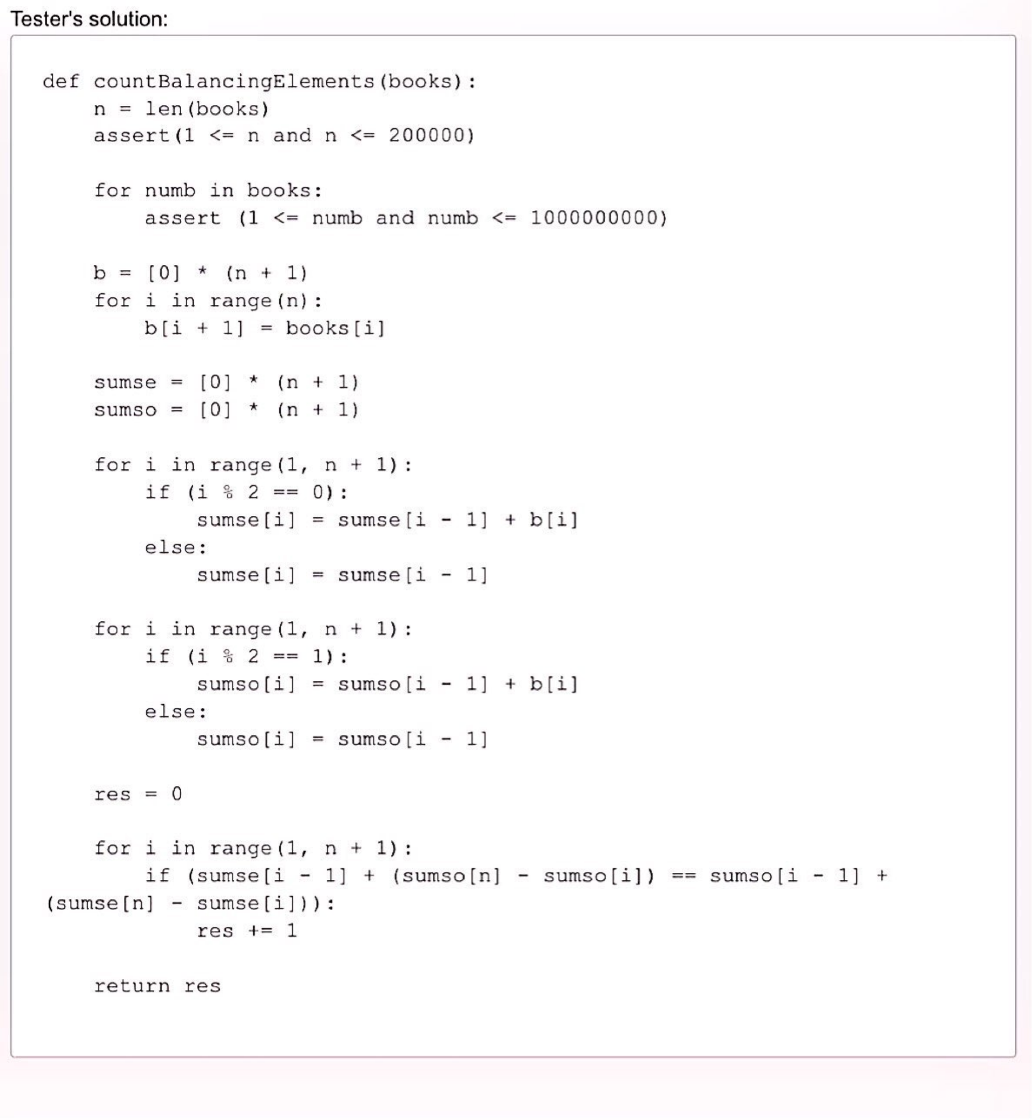
```
               r2 = right[i+2];

           if(l1 + r1 == l2 + r2)
               count++;
       }

       return count;

   }
```



Tester's solution:

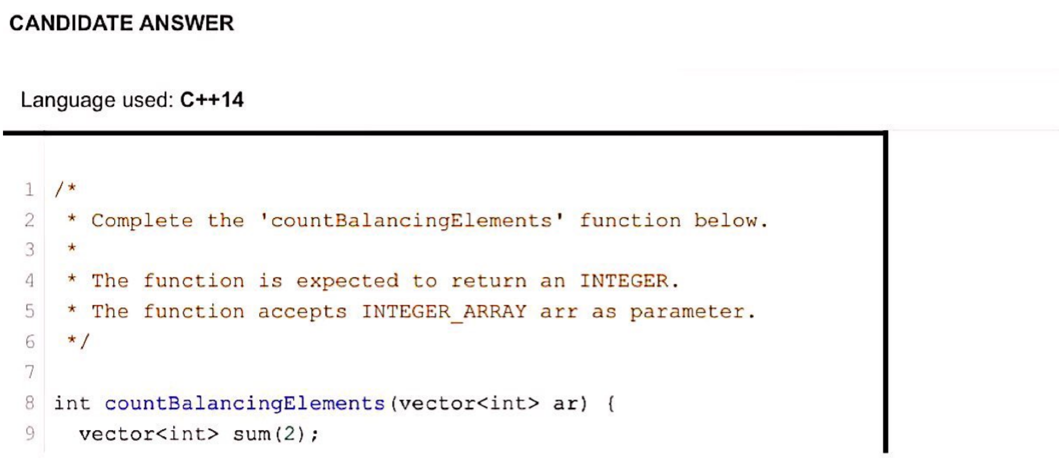
```
def countBalancingElements(books):
    n = len(books)
    assert(1 <= n and n <= 200000)

    for numb in books:
        assert (1 <= numb and numb <= 1000000000)

    b = [0] * (n + 1)
    for i in range(n):
        b[i + 1] = books[i]

    sumse = [0] * (n + 1)
    sumso = [0] * (n + 1)

    for i in range(1, n + 1):
        if (i % 2 == 0):
            sumse[i] = sumse[i - 1] + b[i]
        else:
            sumso[i] = sumso[i - 1]
```

Scanned with CamScanner



Tester's solution:

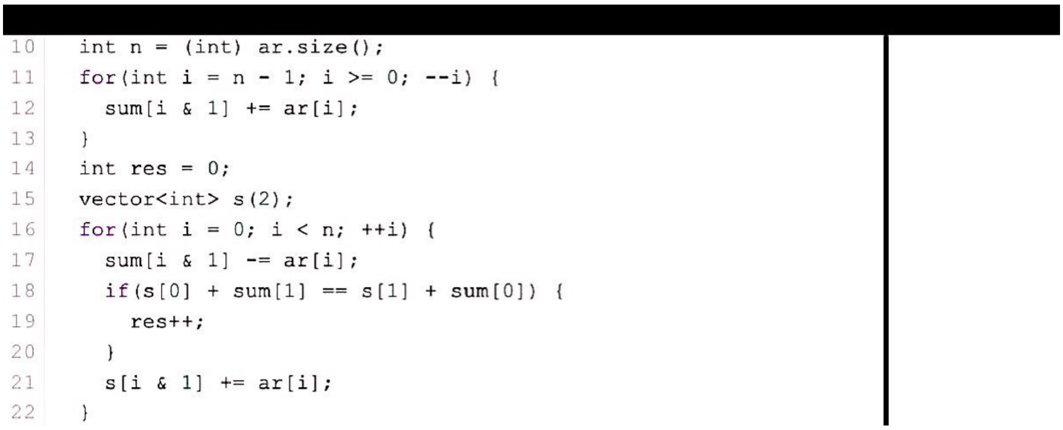
```
def countBalancingElements(books):
    n = len(books)
    assert(1 <= n and n <= 200000)

    for numb in books:
        assert (1 <= numb and numb <= 1000000000)

    b = [0] * (n + 1)
    for i in range(n):
        b[i + 1] = books[i]

    sumse = [0] * (n + 1)
    sumso = [0] * (n + 1)

    for i in range(1, n + 1):
        if (i % 2 == 0):
            sumse[i] = sumse[i - 1] + b[i]
        else:
            sumse[i] = sumse[i - 1]

    for i in range(1, n + 1):
        if (i % 2 == 1):
            sumso[i] = sumso[i - 1] + b[i]
        else:
            sumso[i] = sumso[i - 1]

    res = 0

    for i in range(1, n + 1):
        if (sumse[i - 1] + (sumso[n] - sumso[i]) == sumso[i - 1] +
(sumse[n] - sumse[i])):
            res += 1

    return res
```



CANDIDATE ANSWER

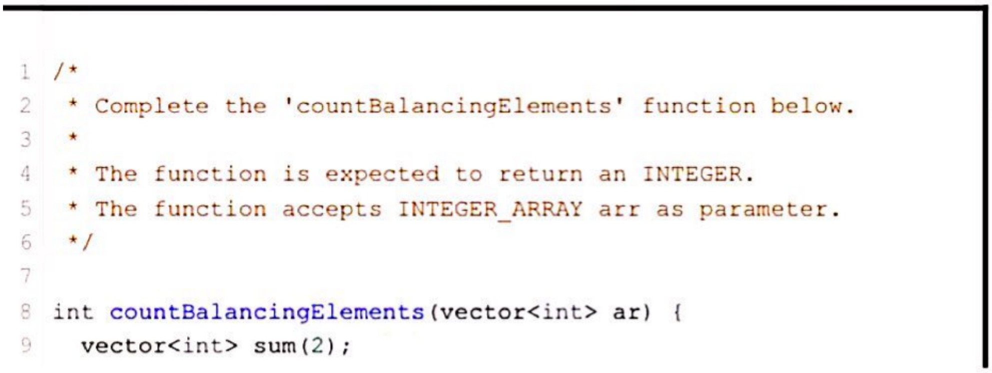
Language used: **C++14**

```
/*
 * Complete the 'countBalancingElements' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

int countBalancingElements(vector<int> ar) {
  vector<int> sum(2);
```

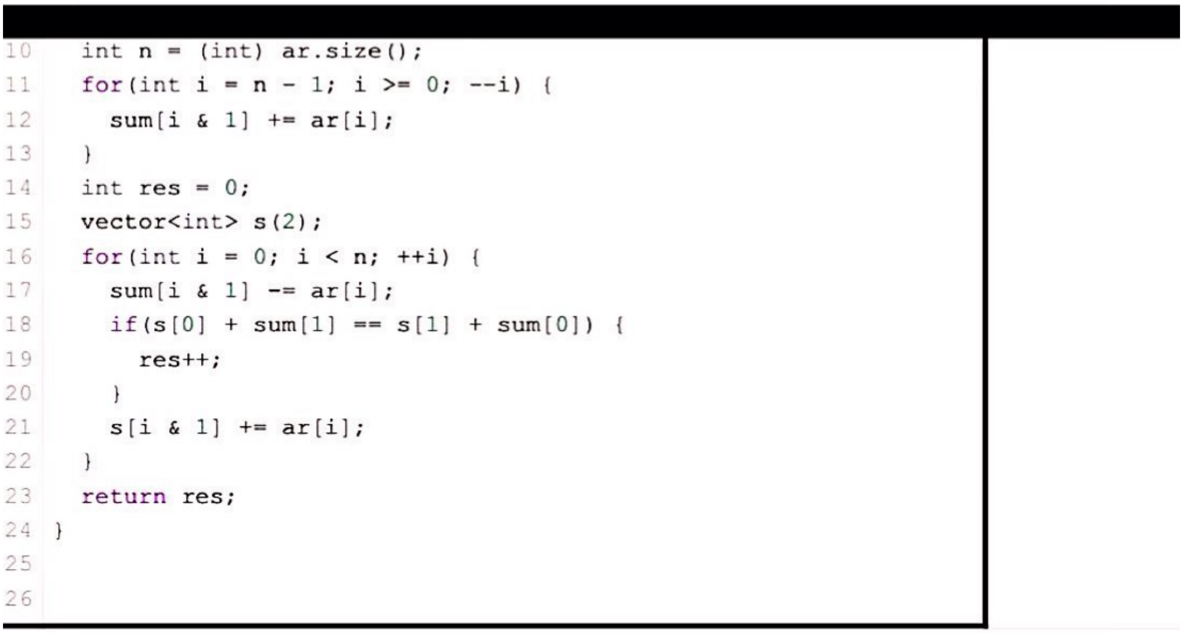


```
  int n = (int) ar.size();
  for(int i = n - 1; i >= 0; --i) {
    sum[i & 1] += ar[i];
  }
  int res = 0;
  vector<int> s(2);
  for(int i = 0; i < n; ++i) {
    sum[i & 1] -= ar[i];
    if(s[0] + sum[1] == s[1] + sum[0]) {
      res++;
    }
    s[i & 1] += ar[i];
  }
```

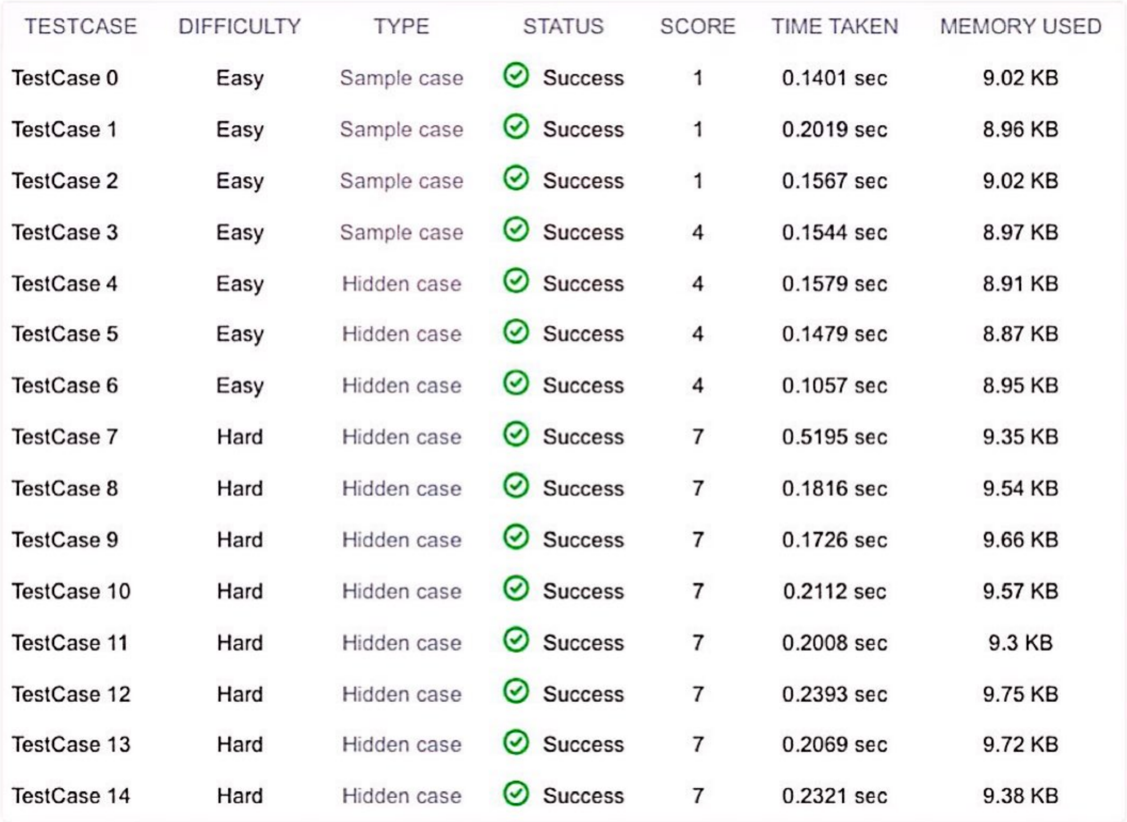
Language used: **C++14**



```
/*
 * Complete the 'countBalancingElements' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

int countBalancingElements(vector<int> ar) {
  vector<int> sum(2);
```

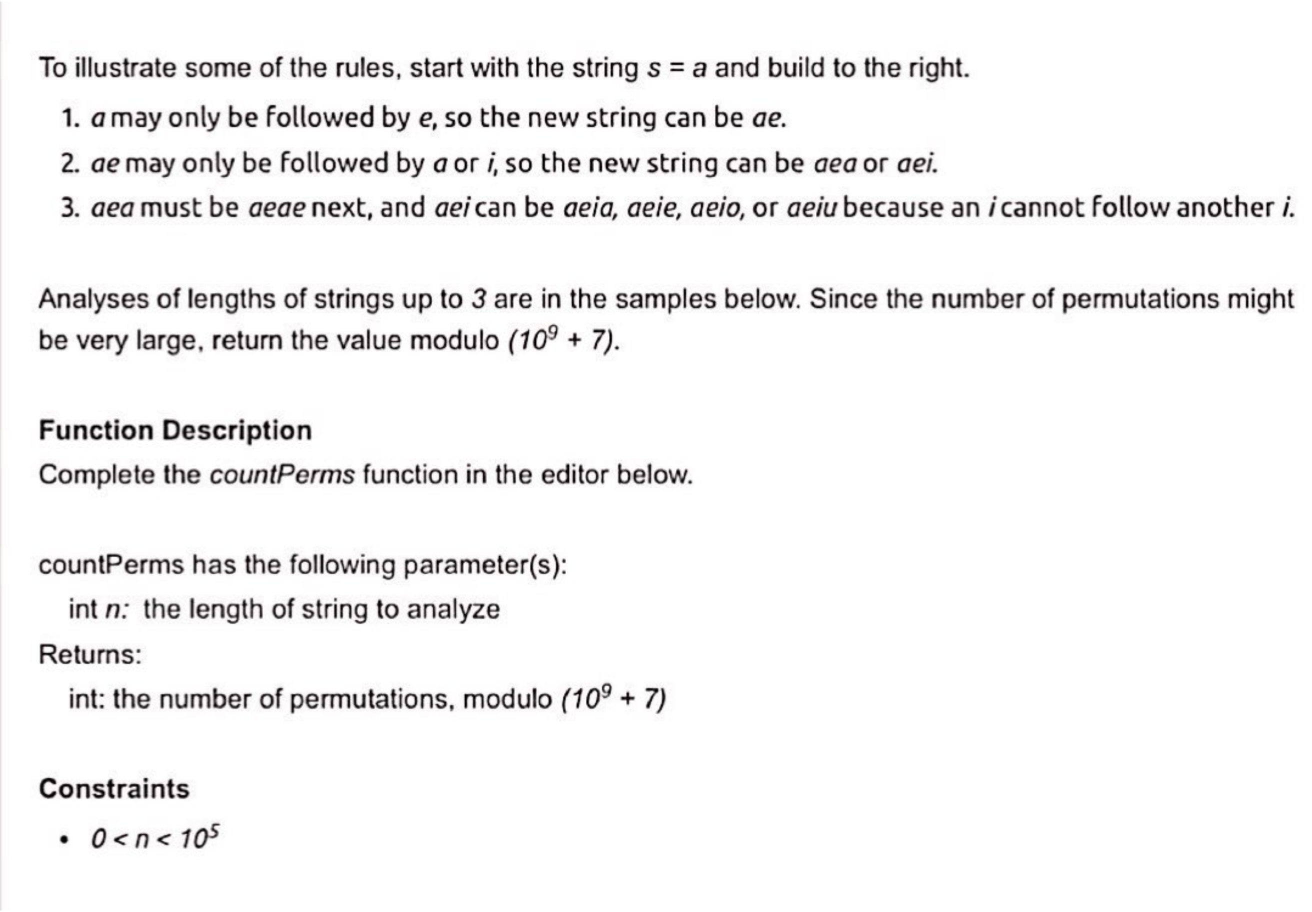


```
  int n = (int) ar.size();
  for(int i = n - 1; i >= 0; --i) {
    sum[i & 1] += ar[i];
  }
  int res = 0;
  vector<int> s(2);
  for(int i = 0; i < n; ++i) {
    sum[i & 1] -= ar[i];
    if(s[0] + sum[1] == s[1] + sum[0]) {
      res++;
    }
    s[i & 1] += ar[i];
  }
  return res;
}
```



TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	✓ Success	1	0.1401 sec	9.02 KB
TestCase 1	Easy	Sample case	✓ Success	1	0.2019 sec	8.96 KB
TestCase 2	Easy	Sample case	✓ Success	1	0.1567 sec	9.02 KB
TestCase 3	Easy	Sample case	✓ Success	4	0.1544 sec	8.97 KB
TestCase 4	Easy	Hidden case	✓ Success	4	0.1579 sec	8.91 KB
TestCase 5	Easy	Hidden case	✓ Success	4	0.1479 sec	8.87 KB
TestCase 6	Easy	Hidden case	✓ Success	4	0.1057 sec	8.95 KB
TestCase 7	Hard	Hidden case	✓ Success	7	0.5195 sec	9.35 KB
TestCase 8	Hard	Hidden case	✓ Success	7	0.1816 sec	9.54 KB
TestCase 9	Hard	Hidden case	✓ Success	7	0.1726 sec	9.66 KB
TestCase 10	Hard	Hidden case	✓ Success	7	0.2112 sec	9.57 KB
TestCase 11	Hard	Hidden case	✓ Success	7	0.2008 sec	9.3 KB
TestCase 12	Hard	Hidden case	✓ Success	7	0.2393 sec	9.75 KB
TestCase 13	Hard	Hidden case	✓ Success	7	0.2069 sec	9.72 KB
TestCase 14	Hard	Hidden case	✓ Success	7	0.2321 sec	9.38 KB

No Comments



To illustrate some of the rules, start with the string $s = a$ and build to the right.

1. *a* may only be followed by *e*, so the new string can be *ae*.
2. *ae* may only be followed by *a* or *i*, so the new string can be *aea* or *aei*.
3. *aea* must be *aeae* next, and *aei* can be *aeia, aeie, aeio,* or *aeiu* because an *i* cannot follow another *i*.

Analyses of lengths of strings up to *3* are in the samples below. Since the number of permutations might be very large, return the value modulo $(10^9 + 7)$.

Function Description

Complete the *countPerms* function in the editor below.

countPerms has the following parameter(s):

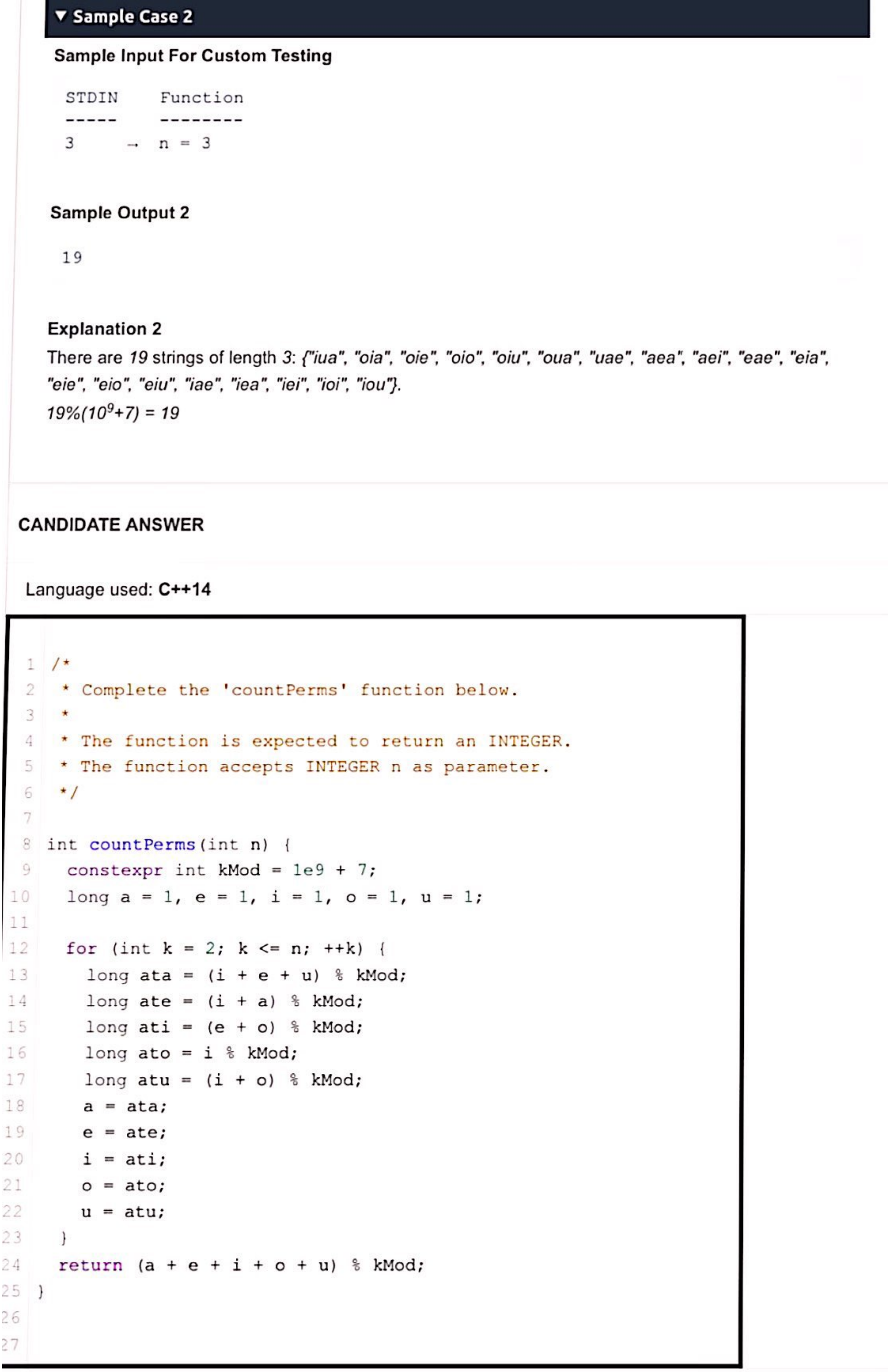
int *n:* the length of string to analyze

Returns:

int: the number of permutations, modulo $(10^9 + 7)$

Constraints

- $0 < n < 10^5$



▼ Sample Case 2

Sample Input For Custom Testing

```
STDIN     Function
-----     --------
3      →  n = 3
```

Sample Output 2

```
19
```

Explanation 2

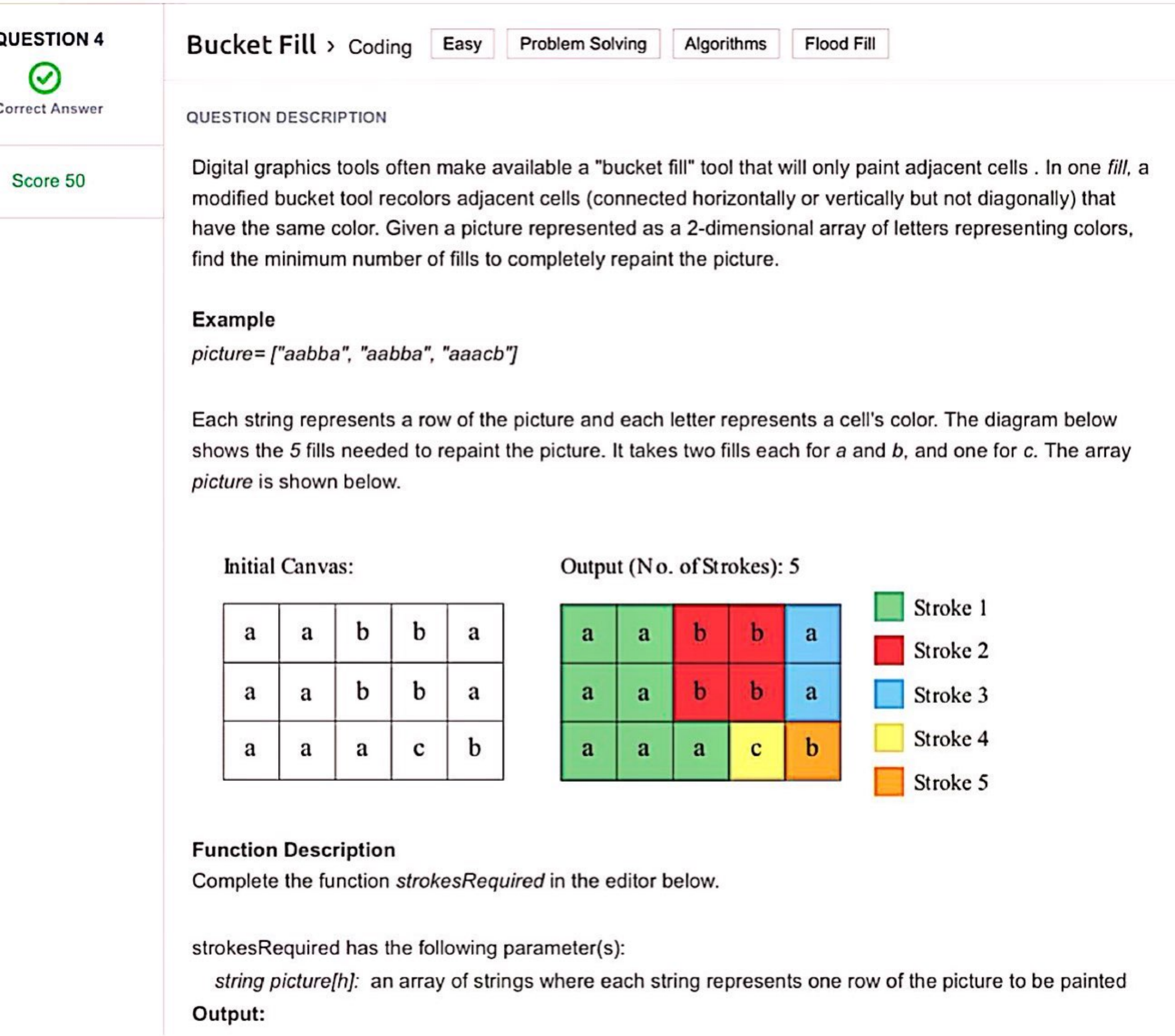
There are *19* strings of length *3*: *{"iua", "oia", "oie", "oio", "oiu", "oua", "uae", "aea", "aei", "eae", "eia", "eie", "eio", "eiu", "iae", "iea", "iei", "ioi", "iou"}*.

$19\%(10^9+7) = 19$

CANDIDATE ANSWER

Language used: **C++14**

```
/*
 * Complete the 'countPerms' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts INTEGER n as parameter.
 */

int countPerms(int n) {
  constexpr int kMod = 1e9 + 7;
  long a = 1, e = 1, i = 1, o = 1, u = 1;

  for (int k = 2; k <= n; ++k) {
    long ata = (i + e + u) % kMod;
    long ate = (i + a) % kMod;
    long ati = (e + o) % kMod;
    long ato = i % kMod;
    long atu = (i + o) % kMod;
    a = ata;
    e = ate;
    i = ati;
    o = ato;
    u = atu;
  }
  return (a + e + i + o + u) % kMod;
}


```



QUESTION 4

✓

Correct Answer

Score 50

Bucket Fill › Coding

Easy | Problem Solving | Algorithms | Flood Fill

QUESTION DESCRIPTION

Digital graphics tools often make available a "bucket fill" tool that will only paint adjacent cells . In one *fill*, a modified bucket tool recolors adjacent cells (connected horizontally or vertically but not diagonally) that have the same color. Given a picture represented as a 2-dimensional array of letters representing colors, find the minimum number of fills to completely repaint the picture.

Example

picture= ["aabba", "aabba", "aaacb"]

Each string represents a row of the picture and each letter represents a cell's color. The diagram below shows the *5* fills needed to repaint the picture. It takes two fills each for *a* and *b*, and one for *c*. The array *picture* is shown below.

Initial Canvas:

a	a	b	b	a
a	a	b	b	a
a	a	a	c	b

Output (No. of Strokes): 5

🟩 a	🟩 a	🟥 b	🟥 b	🟦 a
🟩 a	🟩 a	🟥 b	🟥 b	🟦 a
🟩 a	🟩 a	🟩 a	🟨 c	🟧 b

🟩	Stroke 1
🟥	Stroke 2
🟦	Stroke 3
🟨	Stroke 4
🟧	Stroke 5

Function Description

Complete the function *strokesRequired* in the editor below.

strokesRequired has the following parameter(s):

string picture[h]: an array of strings where each string represents one row of the picture to be painted

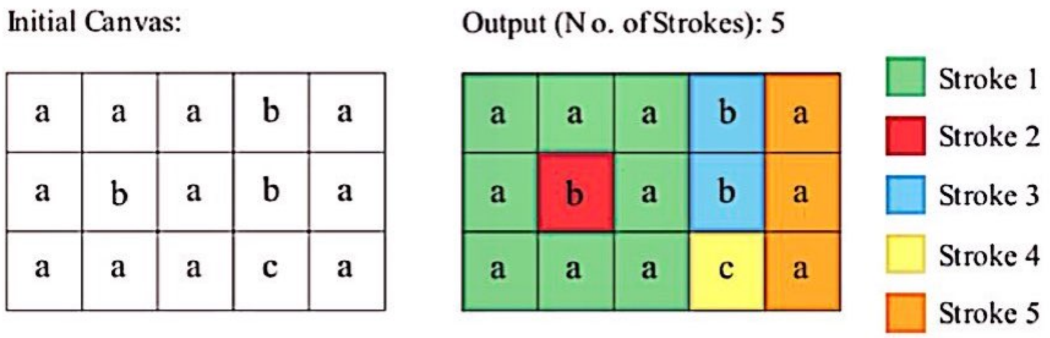
Output:

```
3      →    picture[] size h = 3
aaaba  →    picture = [ "aaaba" , "ababa" , "aaaca" ]
ababa
aaaca
```

Sample Output

```
5
```

Explanation



Initial Canvas:

a	a	a	b	a
a	b	a	b	a
a	a	a	c	a

Output (No. of Strokes): 5

a	a	a	b	a
a	b	a	b	a
a	a	a	c	a

🟩 Stroke 1

🟥 Stroke 2

🟦 Stroke 3

🟨 Stroke 4

🟧 Stroke 5

Letter *a* takes *2* fills, *b* takes *2* fills and *c* takes *1* fill for a total of *5*.

▼ Sample Case 1

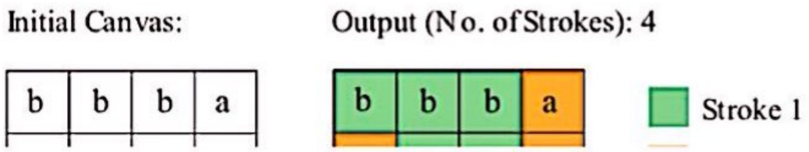
Sample Input For Custom Testing

```
STDIN       Function
-----       --------
4      →    picture[] size h = 4
bbba   →    picture = [ "bbba", "abba", "acaa" , "aaac" ]
abba
acaa
aaac
```

Sample Output

```
4
```

Explanation



Initial Canvas:

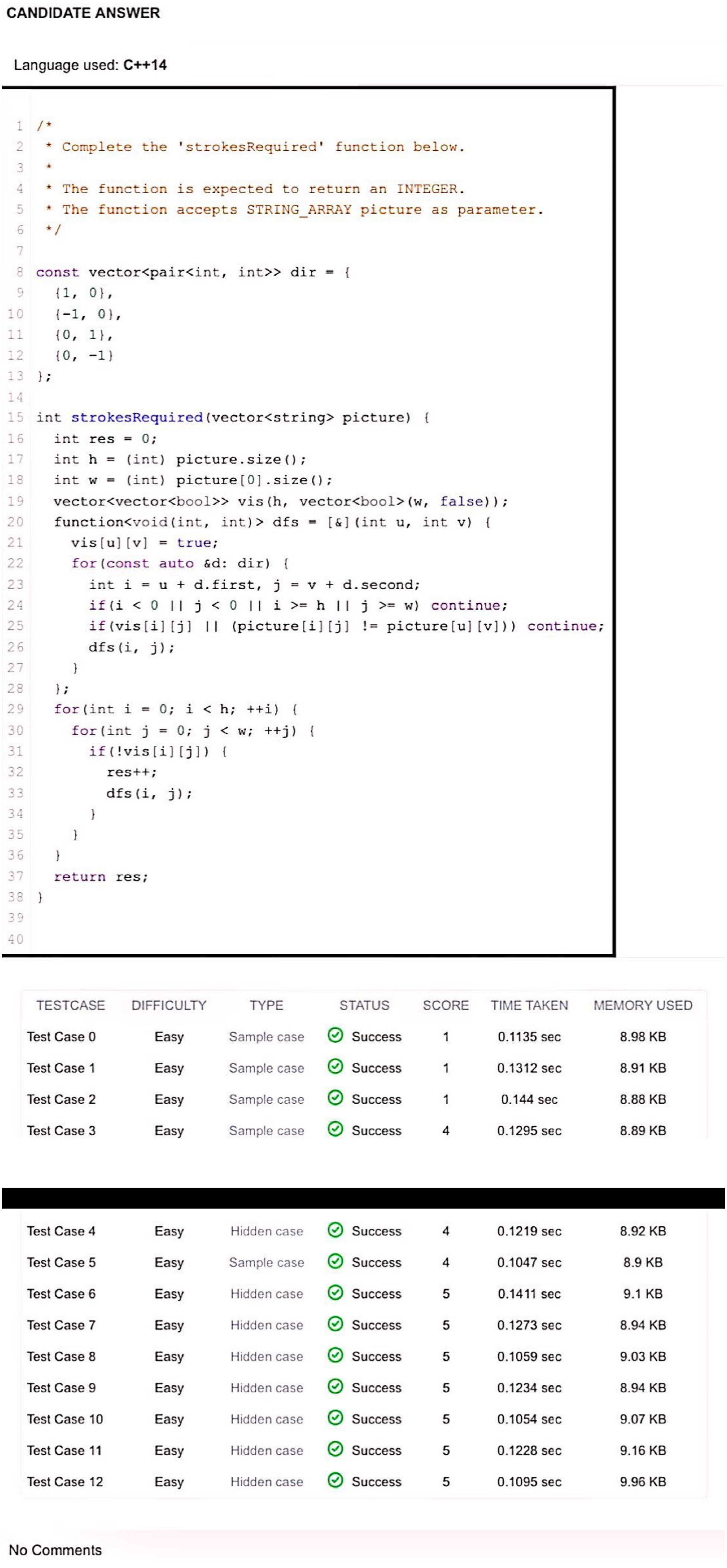
b	b	b	a

Output (No. of Strokes): 4

b	b	b	a

🟩 Stroke 1

Scanned with CamScanner



CANDIDATE ANSWER

Language used: **C++14**

```cpp
/*
 * Complete the 'strokesRequired' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts STRING_ARRAY picture as parameter.
 */

const vector<pair<int, int>> dir = {
  {1, 0},
  {-1, 0},
  {0, 1},
  {0, -1}
};

int strokesRequired(vector<string> picture) {
  int res = 0;
  int h = (int) picture.size();
  int w = (int) picture[0].size();
  vector<vector<bool>> vis(h, vector<bool>(w, false));
  function<void(int, int)> dfs = [&](int u, int v) {
    vis[u][v] = true;
    for(const auto &d: dir) {
      int i = u + d.first, j = v + d.second;
      if(i < 0 || j < 0 || i >= h || j >= w) continue;
      if(vis[i][j] || (picture[i][j] != picture[u][v])) continue;
      dfs(i, j);
    }
  };
  for(int i = 0; i < h; ++i) {
    for(int j = 0; j < w; ++j) {
      if(!vis[i][j]) {
        res++;
        dfs(i, j);
      }
    }
  }
  return res;
}


```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Test Case 0	Easy	Sample case	✓ Success	1	0.1135 sec	8.98 KB
Test Case 1	Easy	Sample case	✓ Success	1	0.1312 sec	8.91 KB
Test Case 2	Easy	Sample case	✓ Success	1	0.144 sec	8.88 KB
Test Case 3	Easy	Sample case	✓ Success	4	0.1295 sec	8.89 KB
Test Case 4	Easy	Hidden case	✓ Success	4	0.1219 sec	8.92 KB
Test Case 5	Easy	Sample case	✓ Success	4	0.1047 sec	8.9 KB
Test Case 6	Easy	Hidden case	✓ Success	5	0.1411 sec	9.1 KB
Test Case 7	Easy	Hidden case	✓ Success	5	0.1273 sec	8.94 KB
Test Case 8	Easy	Hidden case	✓ Success	5	0.1059 sec	9.03 KB
Test Case 9	Easy	Hidden case	✓ Success	5	0.1234 sec	8.94 KB
Test Case 10	Easy	Hidden case	✓ Success	5	0.1054 sec	9.07 KB
Test Case 11	Easy	Hidden case	✓ Success	5	0.1228 sec	9.16 KB
Test Case 12	Easy	Hidden case	✓ Success	5	0.1095 sec	9.96 KB

No Comments