2021 상반기 ICT 인턴십 국내과정 코딩테스트 > sickal2812@naver.com

You can view this report online at : https://www.hackerrank.com/x/tests/1018642/candidates/21936845/report

| | |
|---|---|
| Full Name: | 안일훈 |
| Email: | sickal2812@naver.com |
| Test Name: | **2021 상반기 ICT 인턴십 국내과정 코딩테스트** |
| Taken On: | 15 Feb 2021 14:01:59 JST |
| Time Taken: | 646 min 12 sec/ 720 min |
| Personal Email Address: | sickal2812@naver.com |
| Univ/College Name: | 한국항공대학교 |
| Contact Number: | 01027442812 |
| Major: | 소프트웨어학과 |
| Invited by: | Young-Guk |
| Invited on: | 15 Feb 2021 11:02:04 JST |
| Skills Score: | Problem Solving (Advanced) 154/175<br>Problem Solving (Basic) 100/100<br>Problem Solving (Intermediate) 75/75 |
| Tags Score: | Algorithms 204/225<br>Arrays 75/75<br>Combinatorics 75/75<br>Data Structures 200/200<br>Dynamic Programming 154/175<br>Easy 100/100<br>Hard 100/100<br>Hash Map 50/50<br>Implementation 75/75<br>Interviewer Guidelines 200/200<br>Linked Lists 50/50<br>Loops 50/50<br>Medium 129/150<br>Problem Solving 329/350<br>Strings 104/125<br>Theme: Automotive 100/100 |

**94%**

**329/350**

scored in **2021 상반기 ICT 인턴십 국내과정 코딩테스트** in 646 min 12 sec on 15 Feb 2021 14:01:59 JST

Recruiter/Team Comments:

*No Comments.*

| | Question Description | Time Taken | Score | Status |
|---|---|---|---|---|
| Q1 | Binary Number in a Linked List > **Coding** | 30 min 51 sec | 50/ 50 | ✓ |

| | | | | | |
|---|---|---|---|---|---|
| Q2 | Fun with Anagrams › | Coding | 42 min 24 sec | 50/ 50 | ✓ |
| Q3 | Inversions › | Coding | 1 hour 48 min 33 sec | 75/ 75 | ✓ |
| Q4 | Perfect Substring › | Coding | 5 hour 41 min 2 sec | 54/ 75 | ✓ |
| Q5 | Paths in a Warehouse › | Coding | 2 hour 1 min 3 sec | 100/ 100 | ✓ |

QUESTION 1

✓

Correct Answer

Score 50

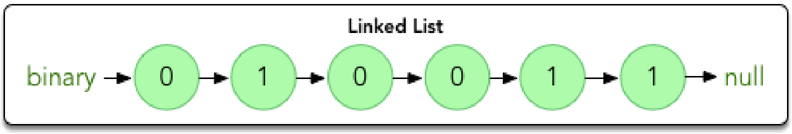
## Binary Number in a Linked List › Coding

Easy | Data Structures | Algorithms | Problem Solving | Linked Lists | Loops | Interviewer Guidelines

QUESTION DESCRIPTION

A binary number is represented as a series of *0*'s and *1*'s. In this challenge, the series will be in the form of a singly-linked list. Each node instance, a *LinkedListNode,* has a value, *data*, and a pointer to the next node, *next*. Given a reference to the head of a singly-linked list, convert the binary number represented to a decimal number.

**Example**



Linked list corresponding to the binary number $(010011)_2$ or $(19)_{10}$.

**Function Description**

Complete the function *getNumber* in the editor below.

getNumber has the following parameter(s):

*binary:* reference to the head of a singly-linked list of binary digits

Returns:

int: a (long integer)$_{10}$ representation of the binary number

**Constraints**

- $1 \le n \le 64$
- *All LinkedListNode.data* $\in \{01\}$
- *The described (integer)*$_2 < 2^{64}$

### ▼ Input Format for Custom Testing

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

The first line contains an integer *n*, the size of the linked list *binary*.

Each of the next *n* lines contains an integer *LinkedListNode.data[i]* where $0 \le i < n$.

### ▼ Sample Case 0

**Sample Input**

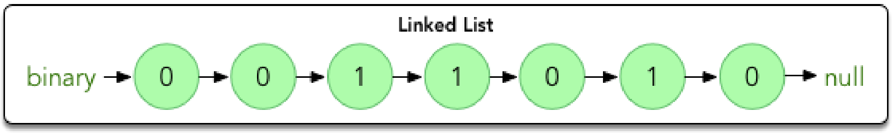
```
STDIN     Function
-----     -----
7      →  binary[] size n = 7
0      →  binary LinkedListNode.data = [0, 0, 1, 1, 0, 1, 0]
0
1
```

```
1
0
1
0
```

**Sample Output**

```
26
```

**Explanation**



The linked list is given as input.

The linked list forms the binary number *0011010* → *$(0011010)_2$* = *$(26)_{10}$*

### ▼ Sample Case 1

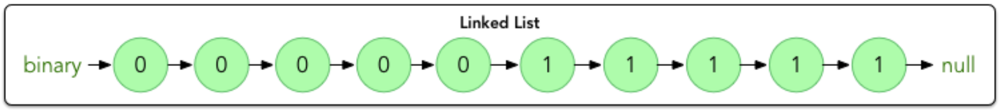
**Sample Input**

```
STDIN     Function
-----     -----
10     →  binary[] size n = 10
0      →  binary LinkedListNode.data = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1]
0
0
0
0
1
1
1
1
1
```

**Sample Output**

```
31
```

**Explanation**



The linked list given as input.

The linked list forms the binary number *0000011111* → *$(0000011111)_2$* = *$(31)_{10}$*

INTERVIEWER GUIDELINES

### ▼ Hint 1

The problem can be solved in a single pass of the list.

### ▼ Hint 2

For each node multiply the current answer by 2 then add the value stored in the current node.

### ▼ Solution

**Concepts covered: This problem covers concepts of work with loops, linked lists and conversion of binary to decimal.**

**Optimal Solution:**

### Optimal Solution:

Initialize the variable *ans* to 0. Step through the list and at each node, multiply *ans* by 2 and add *binary.data*.

For example, in case of 1->0->1->0:

At each iteration, *ans* = *ans**2 + *binary.data*

ans = 0

node 0 = 1, 0*2 + 1 = 1

node 1 = 0, 1*2 + 0 = 2

node 2 = 1, 2*2 + 1 = 5

node 3 = 0, 5*2 +0 = 10

$1010_b = 10_{10}$

```
def getNumber(binary):

    ans = 0
    # repeat through the last node
    while binary != None:
        binary.data
        ans *= 2
        ans += binary.data
        binary = binary.next

    return ans
```

**Error Handling:**

1. The loop needs to be repeated while the current node is not None. Changing this condition to "while next != None" will result in a wrong answer.

**▼ Complexity Analysis**

**Time Complexity** - O(N), where N is the length of the list.

**Space Complexity** - O(1) - only space for a single variable is required.

**CANDIDATE ANSWER**

Language used: **C++**

```
/*
 * Complete the 'getNumber' function below.
 *
 * The function is expected to return a LONG_INTEGER.
 * The function accepts INTEGER_SINGLY_LINKED_LIST binary as parameter.
 */

/*
 * For your reference:
 *
 * SinglyLinkedListNode {
 *     int data;
 *     SinglyLinkedListNode* next;
 * };
 *
 */
```

```
long getNumber(SinglyLinkedListNode* binary) {
    SinglyLinkedListNode* temp = binary;
    long count = 0;
    while(temp != NULL) {
        count = (count << 1) + temp->data;
        temp = temp->next;
    }
    return count;
}
```

| TESTCASE | DIFFICULTY | TYPE | STATUS | SCORE | TIME TAKEN | MEMORY USED |
|---|---|---|---|---|---|---|
| TestCase 0 | Easy | Sample case | ✓ Success | 1 | 0.0219 sec | 8.83 KB |
| TestCase 1 | Easy | Sample case | ✓ Success | 1 | 0.0195 sec | 8.96 KB |
| TestCase 2 | Easy | Sample case | ✓ Success | 1 | 0.0241 sec | 8.86 KB |
| TestCase 3 | Easy | Hidden case | ✓ Success | 6 | 0.0215 sec | 8.89 KB |
| TestCase 4 | Easy | Hidden case | ✓ Success | 6 | 0.0189 sec | 8.94 KB |
| TestCase 5 | Easy | Hidden case | ✓ Success | 6 | 0.027 sec | 9.08 KB |
| TestCase 6 | Easy | Hidden case | ✓ Success | 6 | 0.023 sec | 9.03 KB |
| TestCase 7 | Medium | Sample case | ✓ Success | 5 | 0.0178 sec | 8.98 KB |
| TestCase 8 | Medium | Hidden case | ✓ Success | 5 | 0.0309 sec | 9.03 KB |
| TestCase 9 | Medium | Hidden case | ✓ Success | 5 | 0.0235 sec | 8.99 KB |
| TestCase 10 | Hard | Sample case | ✓ Success | 2 | 0.0348 sec | 9.03 KB |
| TestCase 11 | Hard | Hidden case | ✓ Success | 2 | 0.0179 sec | 9.02 KB |
| TestCase 12 | Hard | Hidden case | ✓ Success | 2 | 0.0261 sec | 8.98 KB |
| TestCase 13 | Hard | Hidden case | ✓ Success | 2 | 0.022 sec | 8.93 KB |

No Comments

**QUESTION 2**

✓ Correct Answer

Score 50

## Fun with Anagrams › Coding

Data Structures | Strings | Problem Solving | Easy | Interviewer Guidelines | Hash Map

### QUESTION DESCRIPTION

Two strings are anagrams if they are permutations of each other. In other words, both strings have the same size and the same characters. For example, "aaagmnrs" is an anagram of "anagrams". Given an array of strings, remove each string that is an anagram of an earlier string, then return the remaining array in sorted order.

**Example**

*str = ['code', 'doce', 'ecod', 'framer', 'frame']*

- "*code*" and "*doce*" are anagrams. Remove "doce" from the array and keep the first occurrence *"code"* in the array.
- *"code"* and "*ecod*" are anagrams. Remove "ecod" from the array and keep the first occurrence *"code"* in the array.
- *"code"* and "*framer*" are not anagrams. Keep both strings in the array.
- "*framer*" and "*frame*" are not anagrams due to the extra *'r'* in *'framer'*. Keep both strings in the array.
- Order the remaining strings in ascending order: *[ "code","frame","framer"]*.

**Function Description**

### Function Description

Complete the function *funWithAnagrams* in the editor below.

*funWithAnagrams* has the following parameters:
*string text[n]:* an array of strings
Returns:
*string[m]:* an array of the remaining strings in ascending alphabetical order,.

### Constraints

- *0 ≤ n ≤ 1000*
- *0 ≤ m ≤ n*
- *1 ≤* length of *text[i] ≤ 1000*
- Each string *text[i]* is made up of characters in the range ascii[a-z].

### ▼ Input Format For Custom Testing

The first line contains an integer, *n*, that denotes the number of elements in *text*.

Each line *i* of the *n* subsequent lines (where *0 ≤ i < n*) contains a string that describes *text[i]*.

### ▼ Sample Case 0

**Sample Input For Custom Testing**

```
STDIN       Function
-----       --------
4        →  n = 4
code     →  text = ["code","aaagmnrs","anagrams","doce"]
aaagmnrs
anagrams
doce
```

**Sample Output**

```
aaagmnrs
code
```

**Explanation**

- "*code"* and "*doce"* are anagrams. Remove "*doce*" and keep the first occurrence *"code"* in the array.
- *"aaagmnrs"* and "*anagrams"* are anagrams. Remove "a*nagrams*" and keep the first occurrence *"aaagmnrs"* in the array.
- Order the remaining strings in ascending order: *["aaagmnrs", "code"].*

### ▼ Sample Case 1

**Sample Input For Custom Testing**

```
STDIN      Function
-----      --------
4       →  n = 4
poke    →  text = ["poke","pkoe","okpe","ekop"]
pkoe
okpe
ekop
```

**Sample Output**

```
poke
```

**Explanation**

- *"poke*" and *"pkoe"* are anagrams. Remove *"pkoe"* and keep the first occurrence *"poke"* in the array.
- *"poke"* and *"okpe"* are anagrams. Remove *"okpe"* and keep the first occurrence *"poke"* in the array.
- *"poke"* and *"ekop"* are anagrams. Remove *"ekop"* and keep the first occurrence *"poke"* in the array.
- Order the remaining strings in ascending order: *["poke"]*.

INTERVIEWER GUIDELINES

### ▼ Hint 1

What is an efficient way of comparing mixed up characters between 2 strings? Answer: Sort the characters before comparing.

### ▼ Hint 2

What is an efficient data structure for checking whether the sorted characters has been seen? Answer: A hash map of some kind. The best from a memory standpoint is a set that only allows one occurrence of a value.

### ▼ Solution

**Concepts covered: Sorting, data type conversions, use of hash maps**

**Optimal Solution:**

For each string, convert it to a hashable sorted list of characters. See if it has already been seen. If not, store the string to the answer array and the sorted list to the hash table. Finally, sort the resulting list of strings alphabetically.

```
def funWithAnagrams(text):
    # Write your code here
    # a set of words as sorted character tuples
    cs = set()
    # words remaining
    ans = []
    for t in text:
        # store text as a tuple of sorted characters
        # hash map requires immutable type
        tt = tuple(sorted(list(t)))
        # if the character tuple has not been seen
        if not tt in cs:
            ans.append(t)
            cs.add(tt)
    # the results are sorted alphabetically
    return sorted(ans)
```

**Error Handling:** Hash tables require immutable types. The sorted list must be cast as a valid type for hashing.

### ▼ Complexity Analysis

**Time Complexity** - O(N log N).

All characters must be sorted, so N is the sum of the lengths of all strings.

**Space Complexity** - O(N)

Space is required for a hash map. The worst case is that there are no anagrams, so all strings will be stored in the hash map.

## CANDIDATE ANSWER

Language used: **C++**

```
/*
 * Complete the 'funWithAnagrams' function below.
 *
 * The function is expected to return a STRING_ARRAY.
 * The function accepts STRING_ARRAY text as parameter.
 */

vector<string> funWithAnagrams(vector<string> text) {
```

```
vector<string> funWithAnagrams(vector<string> text) {
    vector<string> temp;
    vector<string> ans;
    vector<string>::iterator it = text.begin();

    string sorted;
    for(int i=0; i<text.size(); i++) {
        sorted = *it;
        sort(sorted.begin(),sorted.end());
        temp.push_back(sorted);
        it++;
    }
    sort(temp.begin(),temp.end());
    temp.erase(unique(temp.begin(),temp.end()),temp.end());
    // text 요소 추출후 정렬후 temp에 집어넣고 중복제거

    it = text.begin();
    vector<string>::iterator it2 = temp.begin();

    for(int i=0; i<temp.size(); i++) {

        it = text.begin();
        for(int j=0; j<text.size(); j++) {
            string org = *it;
            sort(org.begin(),org.end());
            if(org == *it2) {
                ans.push_back(*it);
                break;
            }
            it++;
        }
        it2++;
    }
    sort(ans.begin(),ans.end());

    return ans;
}
```

| TESTCASE | DIFFICULTY | TYPE | STATUS | SCORE | TIME TAKEN | MEMORY USED |
|---|---|---|---|---|---|---|
| TestCase 0 | Easy | Sample case | ✓ Success | 2 | 0.0187 sec | 9.06 KB |
| TestCase 1 | Easy | Sample case | ✓ Success | 2 | 0.02 sec | 8.89 KB |
| TestCase 2 | Easy | Sample case | ✓ Success | 2 | 0.0193 sec | 9.09 KB |
| TestCase 5 | Easy | Sample case | ✓ Success | 4 | 0.0218 sec | 8.95 KB |
| TestCase 6 | Medium | Hidden case | ✓ Success | 6 | 0.024 sec | 9.06 KB |
| TestCase 7 | Medium | Sample case | ✓ Success | 8 | 0.0256 sec | 9.09 KB |
| TestCase 9 | Hard | Hidden case | ✓ Success | 12 | 0.021 sec | 9.05 KB |
| TestCase 11 | Hard | Hidden case | ✓ Success | 14 | 0.0205 sec | 9.12 KB |

No Comments

**QUESTION 3**

✓ Correct Answer

Score 75

## Inversions › Coding

Implementation | Medium | Arrays | Problem Solving | Combinatorics

QUESTION DESCRIPTION

A subsequence is created by deleting zero or more elements from a list while maintaining the order. For example, the subsequences of [1,2,3] are [1], [2], [3], [1,2], [1,3], [2,3], [1,2,3]. An *inversion* is a strictly example, the subsequences of *[1,2,3]* are *[1]*, *[2]*, *[3]*, *[1,2]*, *[1,3]*, *[2,3]*, *[1,2,3]*. An *inversion* is a strictly decreasing subsequence of length *3*. More formally, given an array, *p = p[n]* an *inversion* in the array is any time some *p[i] > p[j] > p[k]* and *i < j < k*.

Determine the number of inversions within a given array.

**Example**
*n = 5*
*arr = [5,3,4,2,1]*.

The array inversions are:

```
        [5,3,2]
        [5,3,1]
        [5,4,2]
        [5,4,1]
        [5,2,1]
        [3,2,1]
        [4,2,1]
```

**Example 2**
*n = 4*
*prices = [4,2,2,1]*.

The only inversion is [4, 2, 1] and there are two instances: indices 0, 1, 3 and indices 0, 2, 3. The arrays [4, 2, 2] and [ 2, 2, 1] are not considered inversions because they are not strictly decreasing.

**Function Description**
Complete the function *maxInversions* in the editor below.

maxInversions has the following parameter(s):
*int prices[n]:* an array of integers

**Returns**
*long:* a long integer denoting the number of inversions in the array.

**Constraints**

- $1 \le n \le 5000$
- $1 \le arr[i] \le 10^6$, where $0 \le i < n$

**▼ Input Format for Custom Testing**

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

The first line contains an integer *n*, the size of the array arr.
Each of the next *n* lines contains an integer *arr[i]*.

**▼ Sample Case 0**

**Sample Input 0**

```
STDIN   Function Parameters
-----   -------------------
5    →  arr[] size n = 5
4    →  arr = [4, 1, 3, 2, 5]
1
3
2
5
```

**Sample Output 0**

```
1
```

**Explanation 0**

There is only one inversion in the array: *(4, 3, 2)*.

**▼ Sample Case 1**

**Sample Input 1**

```
STDIN   Function Parameters
-----   --------------------
5     → arr[] size n = 5
15    → arr = [15, 10, 1, 7, 8]
10
1
7
8
```

**Sample Output 1**

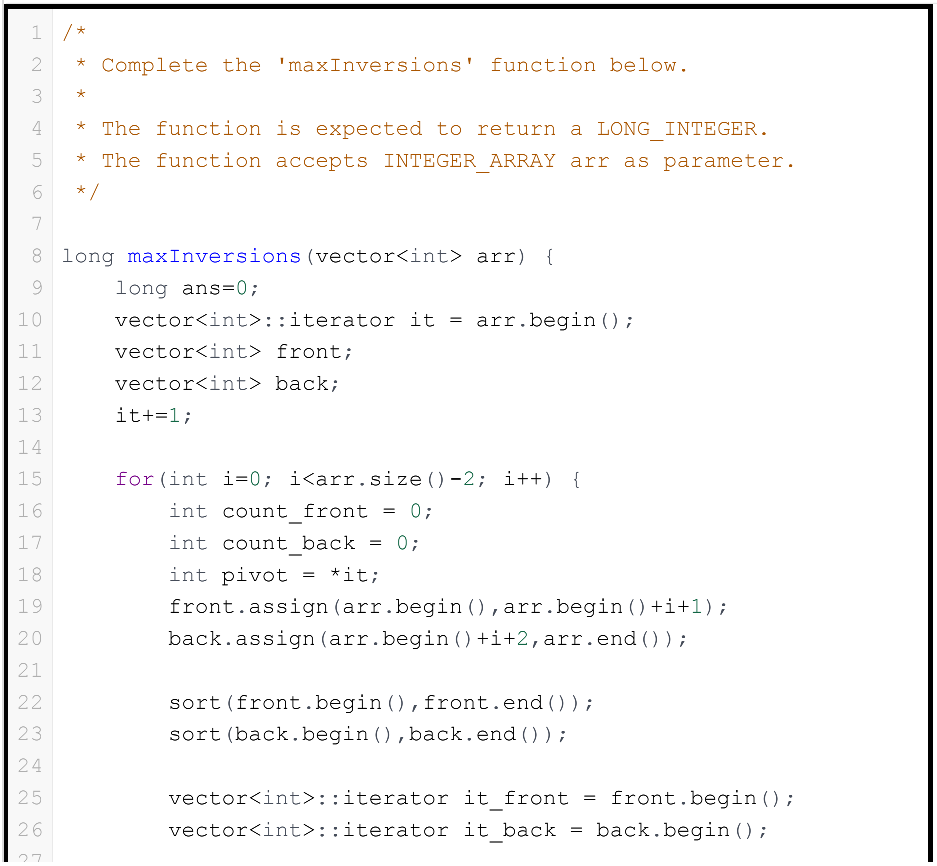
```
3
```

**Explanation 1**

There are three inversions in the array: *(15, 10, 1)*, *(15, 10, 7)*, and *(15, 10, 8)*.

**CANDIDATE ANSWER**

Language used: **C++**



```cpp
/*
 * Complete the 'maxInversions' function below.
 *
 * The function is expected to return a LONG_INTEGER.
 * The function accepts INTEGER_ARRAY arr as parameter.
 */

long maxInversions(vector<int> arr) {
    long ans=0;
    vector<int>::iterator it = arr.begin();
    vector<int> front;
    vector<int> back;
    it+=1;

    for(int i=0; i<arr.size()-2; i++) {
        int count_front = 0;
        int count_back = 0;
        int pivot = *it;
        front.assign(arr.begin(),arr.begin()+i+1);
        back.assign(arr.begin()+i+2,arr.end());

        sort(front.begin(),front.end());
        sort(back.begin(),back.end());

        vector<int>::iterator it_front = front.begin();
        vector<int>::iterator it_back = back.begin();
        for(int i=0; i<front.size(); i++) {
            if(*it_front > pivot) {
                count_front +=1;
            }
            it_front++;
        }
        for(int i=0; i<back.size(); i++) {
            if(pivot > *it_back) {
                count_back +=1;
            }
            it_back++;
        }
        ans += count_back * count_front;
        it++;
    }


    return ans;

}
```

| TESTCASE | DIFFICULTY | TYPE | STATUS | SCORE | TIME TAKEN | MEMORY USED |
|---|---|---|---|---|---|---|
| TestCase 0 | Easy | Sample case | ✓ Success | 1 | 0.0182 sec | 9.02 KB |
| TestCase 1 | Easy | Sample case | ✓ Success | 1 | 0.0215 sec | 8.86 KB |
| TestCase 2 | Easy | Hidden case | ✓ Success | 2.5 | 0.0188 sec | 9.02 KB |
| TestCase 3 | Easy | Hidden case | ✓ Success | 2.5 | 0.02 sec | 9 KB |
| TestCase 4 | Medium | Sample case | ✓ Success | 7 | 0.0191 sec | 8.92 KB |
| TestCase 6 | Medium | Hidden case | ✓ Success | 7 | 0.0183 sec | 9.11 KB |
| TestCase 8 | Medium | Sample case | ✓ Success | 7 | 0.0692 sec | 9.04 KB |
| TestCase 9 | Medium | Hidden case | ✓ Success | 7 | 0.0676 sec | 9.25 KB |
| TestCase 11 | Hard | Hidden case | ✓ Success | 10 | 0.0715 sec | 9.09 KB |
| TestCase 12 | Hard | Hidden case | ✓ Success | 10 | 0.0656 sec | 9.04 KB |
| TestCase 13 | Hard | Hidden case | ✓ Success | 10 | 1.5285 sec | 9.27 KB |
| TestCase 14 | Hard | Hidden case | ✓ Success | 10 | 1.5048 sec | 9.09 KB |

No Comments

**QUESTION 4**

✓ Correct Answer

Score 54

## Perfect Substring › Coding

Medium | Problem Solving | Algorithms | Dynamic Programming | Strings

**QUESTION DESCRIPTION**

A string *s* comprised of digits from *0* to *9* contains a perfect substring if all the elements within a substring occur exactly *k* times. Calculate the number of perfect substrings in *s*.

**Example**

*s = 1102021222*

*k = 2*

The *6* perfect substrings are:

1. *s[0:1] = 11*

2. *s[0:5] = 110202*
3. *s[1:6] = 102021*
4. *s[2:5] = 0202*
5. *s[7:8] = 22*
6. *s[8:9] = 22*

**Function Description**

Complete the function *perfectSubstring* in the editor below.

perfectSubstring has the following parameters:

*str s*: a string where the value of each element *s[i]* is described by the character at position *i* (where 0 ≤ i < n)

*int k*: an integer that denotes the required frequency of the substring

**Output**

*int:* an integer that represents the number of perfect substrings in the given string

**Constraints**

- $1 \le sizeof(s) \le 10^5$
- $0 \le s[i] \le 9$ (where $0 \le i < n$)
- $1 \le k \le 10^5$

### ▼ Input Format For Custom Testing

The first line will contain a string, *s*.

The second line will contain an integer, *k*, the required frequency of the characters in a perfect substring.

### ▼ Sample Case 0

**Sample Input For Custom Testing**

```
STDIN          Function
-----          --------
1020122   →    s = '1020122'
2         →    k = 2
```

**Sample Output**

```
2
```

**Explanation**

Perfect substrings are:

*s[0:5] = 102012*

*s[5:6] = 22*

### ▼ Sample Case 1

**Sample Input For Custom Testing**

```
STDIN          Function
-----          --------
1221221121 →   s = '1221221121'
3          →   k = 3
```

**Sample Output**

```
3
```

**Explanation**

Perfect substrings are:

*s[2:7] = 212211*

*s[3:8] = 122112*

*s[4:9] = 221121*

## CANDIDATE ANSWER

Language used: **C++**

```
/*
 * Complete the 'perfectSubstring' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts following parameters:
 *  1. STRING s
 *  2. INTEGER k
 */

int perfectSubstring(string s, int k) {
    int ans = 0;

    for(int i=0; i<s.size(); i++) {
        int check[10];
        int num =0;
        for(int j=0; j<10; j++) {
            check[j] = 0;
        }
        for(int j=i; j<s.size(); j++) {
            check[(s[j])-48] +=1;
            if(check[s[j]-48] > k) {
                break;
            } else {
                for(int q=0; q<10; q++) {
                    cout << check[q] ;
                    if(check[q] != 0) {
                        if(k != check[q]) {
                            break;
                        }
                        if(q==9) {
                            ans++;
                        }
                    } else {
                        if(q==9) {
                            ans++;
                        }
                    }
                } cout << endl;
            }

        }
    }


    return ans;
}

```

| TESTCASE | DIFFICULTY | TYPE | STATUS | SCORE | TIME TAKEN | MEMORY USED |
|---|---|---|---|---|---|---|
| TestCase 0 | Easy | Sample case | ✓ Success | 3 | 0.0198 sec | 8.97 KB |

| | | | | | | |
|---|---|---|---|---|---|---|
| TestCase 1 | Easy | Sample case | ✓ Success | 3 | 0.0229 sec | 8.95 KB |
| TestCase 2 | Easy | Sample case | ✓ Success | 3 | 0.0194 sec | 8.92 KB |
| TestCase 3 | Easy | Sample case | ✓ Success | 6 | 0.0512 sec | 9.06 KB |
| TestCase 4 | Medium | Sample case | ✓ Success | 9 | 0.7022 sec | 9.16 KB |
| TestCase 5 | Medium | Hidden case | ✓ Success | 12 | 0.1838 sec | 9.04 KB |
| TestCase 8 | Hard | Hidden case | ✓ Success | 18 | 0.5045 sec | 9.15 KB |
| TestCase 11 | Hard | Hidden case | ⊗ Terminated due to timeout | 0 | 2.0042 sec | 8.59 KB |

No Comments

**QUESTION 5**

✓ Correct Answer

Score 100

## Paths in a Warehouse › Coding

Dynamic Programming | Hard | Algorithms | Data Structures | Problem Solving | Theme: Automotive | Interviewer Guidelines

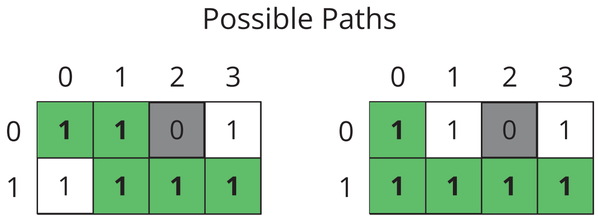
### QUESTION DESCRIPTION

A forklift worker moves products from one place to the other in an automotive parts warehouse. There a map in the dashboard that shows, in real time, the open and blocked sections inside the warehouse. The map is displayed as an *n x m* matrix of 1's and 0's which represent open and blocked sections respectively. A forklift driver always starts at the upper left corner of the map at *warehouse[0][0]* and tries to reach the bottom right section of the map or *warehouse[n-1][m-1]*. Each movement must be in increasing value along a row or column but not both. Given the warehouse map, determine the number of distinct paths to get from *warehouse[0][0]* to *warehouse[n-1][m-1]*. The number may be large, so return the value modulo ($10^9$+7).

**Example**

*warehouse = [1, 1, 0, 1], [1, 1, 1, 1]*

The matrix below is drawn from the *warehouse* array showing open and blocked sections of the warehouse. *1* indicates an open section and *0* indicates a blocked section. It is only possible to travel through open sections, so no path can go through the section at *(0, 2)*.



Possible Paths

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | **1** | **1** | 0 | 1 |
| 1 | 1 | **1** | **1** | **1** |

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| 0 | **1** | 1 | 0 | 1 |
| 1 | **1** | **1** | **1** | **1** |

There are *2* possible paths from *warehouse[0][0]* to *warehouse[1][3]* and *2 modulo ($10^9$+7) = 2*.

**Function Description**

Complete the function *numPaths* in the editor below.

numPaths has the following parameter(s):

*warehouse[n][m]:* a two dimensional array of integers of *n* rows and *m* columns

Returns:

*int:* the number of paths through the matrix, modulo *($10^9$ + 7)*.

Constraints

- *1 ≤ n, m ≤ 1000*
- Each cell in matrix *a* contains either a *0* or a *1*.

### ▼ Input Format for Custom Testing

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

The first line contains an integer *n*, the number of rows in the matrix *warehouse*.
The next line contains an integer *m*, the number of columns in the matrix *warehouse*.
The next *n* lines each contain a string *warehouse[i]* where 0 ≤ i < n and |*warehouse*[i]| = m.

### ▼ Sample Case 0

**Sample Input 0**

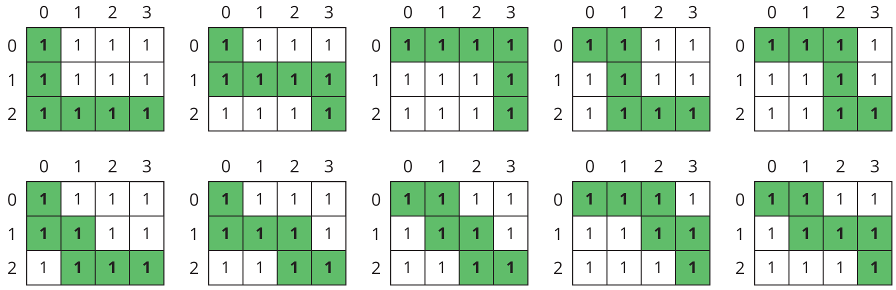
```
STDIN       Function
-----       --------
3       →   warehouse[][] size n=3 m=4
4
1 1 1 1 →   warehouse = [[1, 1, 1, 1], [1, 1, 1, 1], [1, 1, 1, 1]]
1 1 1 1
1 1 1 1
```

**Sample Output 0**

```
10
```

**Explanation 0**

Possible Paths



There are 10 possible paths from *warehouse[0][0]* to *warehouse[2][3]* and *10 modulo ($10^9$ +7) = 10*.

### ▼ Sample Case 1

**Sample Input 1**

```
STDIN      Function
-----      --------
2      →   warehouse[][] size n=2 m=2
2
1 1    →   warehouse = [[1, 1], [0, 1]]
0 1
```

**Sample Output 1**

```
1
```

**Explanation 1**

Possible Path

|   | 0 | 1 |
|---|---|---|
| 0 | **1** | **1** |
| 1 | 0 | **1** |

There is *1* possible path from *warehouse[0][0]* to *warehouse[1][1]* and *1 modulo ($10^9$ + 7) = 1.*

INTERVIEWER GUIDELINES

### ▼ Hint 1

First, try coming up with a brute force approach. We know that for each cell there are two options: go right or go down. Build a recursive solution for the problem.

### ▼ Hint 2

You may notice that many subproblems are redundant. Can you memoize the results? Try coming up with dynamic programming approach.

### ▼ Solution

**Concepts covered:** Dynamic Programming

**Optimal Solution:**

Let's define dp[i][j] to be the number of ways to reach cell (i,j). If the value in cell (i,j) is 0, dp[i][j] = 0. Now, we have two choices:

We came to cell (i, j) from (i-1, j) (Only if warehouse[i-1][j] == 1)

We came to cell (i, j) from (i, j-1) (Only if warehouse[i][j-1] == 1)

The recurrence to build this subproblem is as follows:
dp[i][j] = warehouse[i-1][j] * dp[i-1][j] + warehouse[i][j-1] * dp[i][j-1]

The base case is dp[0][0] = 0. The final answer is dp[n-1][m-1].

```
def numPaths(warehouse):
    n = len(warehouse)
    m = len(warehouse[0])
    dp = [[0] * m for _ in range(n)]
    dp[0][0] = warehouse[0][0]
    for i in range(n):
        for j in range(m):
            if warehouse[i][j] and i > 0:
                dp[i][j] += dp[i - 1][j]
            if warehouse[i][j] and j > 0:
                dp[i][j] += dp[i][j - 1]
            dp[i][j] %= (10**9 + 7)
    return dp[n-1][m-1]
```

**Brute Force Approach:** Passes 6 of 10 test cases

```
def dfs(i, j, n, m, warehouse):
    if i == n or j == m or not warehouse[i][j]:
        return 0
    if i == n-1 and j == m-1:
        if warehouse[i][j]:
            return 1
        else:
            return 0
    return dfs(i+1,j,n,m,warehouse) + dfs(i,j+1,n,m,warehouse) % (10**9 +
7)
def numPaths(warehouse):
    return dfs(0,0,len(warehouse),len(warehouse[0]),warehouse)
```

**Error Handling:** Watch for integer overflows. Do modulo from time to time.

### ▼ Complexity Analysis

**Time Complexity** - O(nm).

Since we are iterating over each element of the grid exactly once, the time complexity is O(nm).

**Space Complexity** - O(nm).

The dp array has the same dimensions as the grid.

### ▼ Follow up Question

**What if the size of grid was huge, n,m of the order of $10^6$, but the number of cells with value 0(obstacles) is relatively smaller, of the order of $10^3$?**

What if there were no obstacles? We have C(n+m-2,n-1) ways to reach (n,m) by combinatorics. But, here we have few obstacles. We can again solve the problem using dynamic programming but the subproblem will be different. The subproblem here will be dp[i] which denotes the number of ways to reach the $i^{th}$ obstacle. It's easy to calculate dp[i] as follows:

dp[i] = C(x+y-2,x-1) - $\Sigma^{j < i}$ dp[j] * (ways to reach from j to i) where $j^{th}$ obstacle lies within (1,1) and (i,j).

## CANDIDATE ANSWER

Language used: **C++**

```
/*
 * Complete the 'numPaths' function below.
 *
 * The function is expected to return an INTEGER.
 * The function accepts 2D_INTEGER_ARRAY warehouse as parameter.
 */

int numPaths(vector<vector<int>> warehouse) {
    int row = warehouse.size();
    vector<int> temp = warehouse.front();
    int col = temp.size();
    int arr[row][col];

    vector<vector<int>>::iterator it = warehouse.begin();
    for(int i=0; i<row; i++) {
        temp = *it;
        vector<int>::iterator it1 = temp.begin();
        for(int j=0; j<col; j++) {
            arr[i][j] = *it1;
            it1++;
        }
        it++;
    }

    for(int i=0; i<row; i++) {
        for(int j=0; j<col; j++) {
            if(i==0 && j != 0 && arr[i][j-1]==0) {
                arr[i][j]=0;
            }
            if(j==0 && i != 0 &&arr[i-1][j]==0) {
                arr[i][j]=0;
            }
            if(i!=0 && j!=0 && arr[i-1][j] == 0 && arr[i][j-1] == 0) {
                arr[i][j]=0;
            }
            if(i!=0 && j!=0 && arr[i][j] == 1) {
                arr[i][j] = arr[i-1][j]%1000000007 + arr[i][j-1]%1000000007;
            }
        }
    }

    return arr[row-1][col-1]%1000000007;

}
```

| TESTCASE | DIFFICULTY | TYPE | STATUS | SCORE | TIME TAKEN | MEMORY USED |
|---|---|---|---|---|---|---|
| Testcase 0 | Easy | Sample case | ✓ Success | 1 | 0.0192 sec | 8.96 KB |
| Testcase 1 | Easy | Sample case | ✓ Success | 1 | 0.0216 sec | 9.03 KB |
| Testcase 2 | Easy | Sample case | ✓ Success | 9 | 0.0327 sec | 8.93 KB |
| Testcase 3 | Easy | Sample case | ✓ Success | 9 | 0.0192 sec | 8.96 KB |
| Testcase 4 | Medium | Sample case | ✓ Success | 15 | 0.0204 sec | 8.96 KB |
| Testcase 5 | Medium | Hidden case | ✓ Success | 15 | 0.0218 sec | 9 KB |
| Testcase 6 | Hard | Hidden case | ✓ Success | 15 | 0.1457 sec | 19.6 KB |
| Testcase 7 | Hard | Hidden case | ✓ Success | 15 | 0.0508 sec | 11.3 KB |
| Testcase 8 | Hard | Hidden case | ✓ Success | 10 | 0.0443 sec | 9.99 KB |
| Testcase 9 | Hard | Hidden case | ✓ Success | 10 | 0.0312 sec | 9.52 KB |

No Comments

**PDF generated at: 15 Feb 2021 15:49:48 UTC**