Arrays - DS

locked

by saikiran9194

Editorial by AllisonP

The 2 solutions provided below both follow this basic logic:

1. Write each element of input into an array.
2. Iterate through the array in reverse, printing each element as you go.

Tested by AllisonP

Problem Tester's code :

```
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int[] array = new int[scan.nextInt()];
        for(int i = 0; i < array.length; i++){
```

Statistics

Difficulty: Easy
Time Complexity: $O(n)$ Required
Knowledge: Arrays, Loops
Publish Date: May 02 2016

```
            array[i] = scan.nextInt();
        }
        scan.close();

        for(int i = array.length - 1; i >= 0; i--){
            System.out.print(array[i] + " ");
        }
    }
}



import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        int n = Integer.parseInt(scan.nextLine());
        String[] array = scan.nextLine().split(" ");
        scan.close();

        while(--n >= 0){
            System.out.print(array[n] + " ");
        }
    }
}
```

2D Array - DS

locked


by Shafaet


Editorial by jwpierce

Given the fixed small size of the problem, brute force is fine.

Points to note:-

1. Negative values possible.
2. Maximum sum can be less than zero.
3. Range of element value is -9 to 9.
4. Numbers to be summed for each hourglass = 7.
5. Minimum possible value for sum = $7 * -9 = -63$.

So we'll initialize our maxSum to -63. From there, we just calculate the sums of all hourglasses and return the maxSum. Here is an implementation of the function in Python:

```
def array2D(arr):

    # want to find the maximum hourglass sum
    # minimum hourglass sum = -9 * 7 = -63
```

Statistics

Difficulty: Easy
Publish Date: Oct 19 2015

```
    maxSum = -63

    for i in range(4):
        for j in range(4):

            # sum of top 3 elements
            top = sum(arr[i][j:j+3])

            # sum of the mid element
            mid = arr[i+1][j+1]

            # sum of bottom 3 elements
            bottom = sum(arr[i+2][j:j+3])

            hourglass = top + mid + bottom

            if hourglass > maxSum:
                maxSum = hourglass

    return maxSum
```

 Tested by AllisonP

Problem Tester's code :

```
public class Solution {
    private static final int _MAX = 6; // size of matrix
    private static final int _OFFSET = 2; // hourglass width
    private static int matrix[][] = new int[_MAX][_MAX];
    private static int maxHourglass = -63; // initialize to lowest possible
 sum (-9 x 7)

    /** Given a starting index for an hourglass, sets maxHourglass
    *   @param i row
    *   @param j column
    **/
    private static void hourglass(int i, int j){
        int tmp = 0; // current hourglass sum
```

```
        // sum top 3 and bottom 3 elements
        for(int k = j; k <= j + _OFFSET; k++){
            tmp += matrix[i][k];
            tmp += matrix[i + _OFFSET][k];
        }

        // sum middle element
        tmp += matrix[i + 1][j + 1];

        if(maxHourglass < tmp){
            maxHourglass = tmp;
        }
    }

    public static void main(String[] args) {
        // read inputs
        Scanner scan = new Scanner(System.in);
        for(int i=0; i < _MAX; i++){
            for(int j=0; j < _MAX; j++){
                matrix[i][j] = scan.nextInt();
            }
        }
        scan.close();

        // find maximum hourglass
        for(int i=0; i < _MAX - _OFFSET; i++){
            for(int j=0; j < _MAX - _OFFSET; j++){
                hourglass(i, j);
            }
        }

        // print maximum hourglass
        System.out.println(maxHourglass);
    }
}
```

Left Rotation

locked

by saikiran9194

Editorial by AllisonP

To solve this challenge, we perform the following basic steps:

1. Create a new n-element (where n is the length of arr) array named $rotated$ to hold the rotated items.
2. Copy the contents of arr over to the new array in two parts:
   - The d-element contiguous segment from arr_0 to arr_{d-1} must be copied over to the contiguous segment starting at $rotated_{n-d}$ and ending at $rotated_{n-1}$.
   - The $(n-d)$-element contiguous segment from arr_d to arr_{n-1} must be copied over to the contiguous segment starting at $rotated_0$ and ending at $rotated_d$.
3. Reassign the reference to arr so that it points to $rotated$ instead.

This is implemented by the following Java code:

```
public static int[] rotateArray(int[] arr, int d){
    // Because the constraints state d < n, we need not concern ourselves w
ith shifting > n units.
```

Statistics

Difficulty: Easy

Publish Date: Jul 06 2016

```
    int n = arr.length;

    // Create new array for rotated elements:
    int[] rotated = new int[n];

    // Copy segments of shifted elements to rotated array:
    System.arraycopy(arr, d, rotated, 0, n - d);
    System.arraycopy(arr, 0, rotated, n - d, d);

    return rotated;
}
```

 Tested by AllisonP

Problem Tester's code :

Java

```
import java.util.*;

public class Solution {

    public static int[] rotateArray(int[] arr, int d){
        // Because the constraints state d < n, we need not concern ourselv
es with shifting > n units.
        int n = arr.length;

        // Create a temporary d-element array to store elements shifted to
 the left of index 0:
        int[] temp_arr = Arrays.copyOfRange(arr, 0, d);

        // Shift elements from indices d through n to indices 0 through d -
 1:
        for(int i = d; i < n; i++) {
            arr[i - d] = arr[i];
        }

        // Copy the d shifted elements from the temporary array back to the
 original array:
```

```
        for(int i = n - d; i < n; i++) {
            arr[i] = temp_arr[i-n+d];
        }

        return arr;
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int d = scanner.nextInt();
        int[] numbers = new int[n];

        // Fill initial array:
        for(int i = 0; i < n; i++){
            numbers[i] = scanner.nextInt();
        }

        // Rotate array by d elements:
        numbers = rotateArray(numbers, d);

        // Print array's elements as a single line of space-separated value
s:
        for(int i : numbers) {
            System.out.print(i + " ");
        }
        System.out.println();

        scanner.close();
    }

}



import java.util.*;
import java.lang.System;

public class Solution {

    public static int[] rotateArray(int[] arr, int d){
        // Because the constraints state d < n, we need not concern ourselv
```

```
es with shifting > n units.
        int n = arr.length;

        // Create new array for rotated elements:
        int[] rotated = new int[n];

        // Copy segments of shifted elements to rotated array:
        System.arraycopy(arr, d, rotated, 0, n - d);
        System.arraycopy(arr, 0, rotated, n - d, d);

        return rotated;
    }

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);
        int n = scanner.nextInt();
        int d = scanner.nextInt();
        int[] numbers = new int[n];

        // Fill initial array:
        for(int i = 0; i < n; i++){
            numbers[i] = scanner.nextInt();
        }

        // Rotate array by d elements:
        numbers = rotateArray(numbers, d);

        // Print array's elements as a single line of space-separated value
s:
        for(int i : numbers) {
            System.out.print(i + " ");
        }
        System.out.println();

        scanner.close();
    }

}
```

Divisible Sum Pairs

locked

by wanbo

Problem Submissions Leaderboard Discussions Editorial

Editorial by wanbo

We can check every pair, one by one, and use $(a_i + a_j)\ \%\ k == 0$ to check if the pair is valid.

Set by wanbo

Problem Setter's code :

C++

```
#include <bits/stdc++.h>
using namespace std;

int n, k;
int a[N];

int main() {
```

Statistics

Difficulty: Easy
Time Complexity: $\mathcal{O}(n^2)$
Required Knowledge: Loops
Publish Date: May 20 2016

```
        cin >> n >> k;
        for(int i = 0; i < n; i++) cin >> a[i];

        int res = 0;
        for(int i = 0; i < n; i++)
                for(int j = i + 1; j < n; j++)
                        if((a[i] + a[j]) % k == 0) res++;
        cout << res << endl;
        return 0;
}
```

 Tested by shashank21j

Problem Tester's code :

Python 2

```
n, k = map(int,raw_input().split())
arr = map(int,raw_input().split())
count = 0
for i in range(0, n):
    for j in range(i+1, n):
        if (arr[i] + arr[j]) % k == 0:
            count += 1
print count
```

Minimum Distances

locked


by Shafaet


Editorial by Shafaet

Use two nested loops to choose two indices and check for matching elements, then find the distance between the two matching elements. Finally, select the minimum of the distances and print it on a new line.

Featured Solutions

Ruby

```ruby
#!/bin/ruby

n = gets.strip.to_i
a = gets.strip
a = a.split(' ').map(&:to_i)

last ={}
min = -1
n.times do |i|
```

Statistics

Difficulty: Easy
Time Complexity: $O(n^2)$
Required Knowledge: Array
Publish Date: Dec 13 2015

```
    x = a[i]
    l = last[x]
    if l and (min < 0 or i - l < min)
        min = i - l
    end
    last[x] = i
end

puts min
```

 Set by Shafaet

Problem Setter's code :

Python 2

```
n = int(raw_input())
A = map(int, raw_input().split())

ans = 9999999999
for i in range(n):
    for j in range(n):
        if A[i] == A[j] and i != j:
            ans = min(ans, abs(i - j))

if ans == 9999999999:
    ans = -1
print ans
```

 Tested by Wanbo

Problem Tester's code :

C++

```cpp
#include <bits/stdc++.h>
using namespace std;

int n;
int a[1000+1];
#define inf 1000000000
int main() {
        cin>>n;
        for(int i = 0; i < n; i++) cin>>a[i];

        int res = inf;
        for(int i = 0; i < n; i++)
                for(int j = 0; j < i; j++)
                        if(a[i] == a[j]) res = min(res, i - j);

        if(res == inf) res = -1;
        cout << res << endl;

        return 0;
}
```

Equalize the Array

locked

by muratekici

Problem Submissions Leaderboard Discussions Editorial

Editorial by muratekici

Count the number of occurrences of each element. Find the element with maximum occurrences and subtract that number of occurrences from the initial array size to determine the minimum number of elements that must be deleted. The total complexity for this is $O(n)$. Take some time to review the code below for more detail.

Statistics

Difficulty: Easy
Time Complexity: $O(n)$ Required
Knowledge: Implementation
Publish Date: Jul 06 2016

Set by muratekici

Problem Setter's code :

```
#include <bits/stdc++.h>

using namespace std;

int n, x, h[105];
```

```c
int main() {
    scanf("%d", &n);
    for(int i = 0; i < n; i++) {
        scanf("%d", &x);
        h[x]++;
    }
    printf("%d\n", n - *max_element(h + 1, h + 100 + 1));
}
```

The Maximum Subarray

locked

by sh4d0wkn1ght

Editorial by kevinsogo

In this editorial, we use 0-indexing, so we refer to the elements of the array as $a_0, a_1, \ldots, a_{n-1}$.

First, understand that there are two questions being asked here:

1. What is the maximum sum of any nonempty *subarray*?
2. What is the maximum sum of any nonempty *subsequence*?

Here, *subarray* means *contiguous subsequence*. They sound similar, but they are solved differently.

We will discuss the solutions to both. We will begin with the second, since it turns out to be easier.

Maximum sum of any nonempty subsequence

Statistics

Difficulty: Medium
Time Complexity: $O(N)$ Required
Knowledge: Dynamic programming, greedy
Publish Date: Nov 19 2014

Let's find the maximum sum of *any* nonempty subsequence, not necessarily contiguous. This is much easier since we have more freedom to choose our subsequence; in fact, to form a subsequence, for each element we get to choose whether to include it or not, *independently of the others*. But since we want to make the sum as large as possible, it follows that we should include those (and only those) elements which contribute positively to the sum.

Specifically,

- Each positive element increases the sum, so we should add them to the subsequence.
- Each negative element decreases the sum, so we should *not* add them.
- The zero elements don't affect the sum, so it doesn't matter a lot whether we include them or not. (Actually, we don't want to end up with an empty subsequence, so it makes sense to include the zeroes just to make the subsequence nonempty as much as possible.)

This gives rise to the following algorithm: *Take the subsequence consisting solely of the positive and zero elements, and print the sum.*

This definitely produces the largest sum of *any* subsequence. This only works, though, if there is at least one positive or zero element. If all elements are negative, then this algorithm produces an *empty* subsequence. But since the empty subsequence is not allowed, we're forced to take at least one negative element. In this case, we shouldn't take as much as we're forced to, so we should just take one negative value. Furthermore, we want to maximize the sum, so we should take the maximum negative value, i.e., the one with the smallest absolute value.

The following high-level pseudocode details this algorithm:

```
def max_sum_subsequence(a):
    Let m be the maximum of a
    Let t be the sum of all positive and zero elements of a
    return (if m >= 0 then t else m)
```

In the return statement, we check whether there is at least one zero or positive element with the expression `m >= 0`. If there is, then we just return `t`, the sum of such elements, otherwise we just return `m`, the largest overall (negative) element.

Here's a more detailed pseudocode:

```
def max_sum_subsequence(a, n):
    m = a[0]
    t = 0
    for i from 0 to n-1:
        m = max(m, a[i])
        if a[i] >= 0: t += a[i]
    return (m >= 0 ? t : m)
```

Maximum sum of any nonempty subarray

Let's now find the maximum sum of a nonempty *contiguous* subarray. This time, we can't do the same greedy approach as before since our first few choices influence what we can take later on.

Of course, we can try to brute-force the solution; however, there are $n(n-1)/2 = O(n^2)$ contiguous subarrays, so enumerating them all won't pass the time limit. Let's try to find a better approach.

How do we form a subarray with a large sum? This time, we can't just greedily choose the positive elements since they may not be contiguous. Instead, let's first select one element, say the last. Suppose we want the ith element, a_i, to be the last element. Then there are two cases. Let's figure out the maximum sum we can form in both cases:

- a_i is the only element of the subarray, i.e., our subarray is just $[a_i]$. In this case, the sum of the subarray will be a_i, and we can't really do anything to change this.
- a_i is not the only element. In this case, our subarray consists of two parts: some *nonempty* subarray ending at a_{i-1}, and a_i itself. Now, we want to maximize the sum, which means we want to maximize the sum of each part. The second part is just a_i which we can't change, but we can do something about the first part.

So, what's the maximum sum of the first part, i.e., the maximum sum of any nonempty subarray ending at a_{i-1}? Note that this is basically the same problem, except we're considering $i-1$, not i!

More formally, if we let $f(k)$ be the maximum sum of any nonempty subarray ending at a_k, then the answer to the last question is $f(i-1)$, hence the largest sum for this case is $f(i-1)+a_i$.

We can now compute $f(i)$ as the larger result of both cases, i.e., $f(i) = \max(a_i, f(i-1)+a_i)$. This works for $i > 0$, but for $i = 0$, only the first case applies, so we have $f(0) = a_0$.

This is now enough to compute the answer quickly! Note that the answer is just $\max(f(0), f(1), \ldots, f(n-1))$, i.e., the maximum sum of any nonempty subarray ending at any position. So we can use the recurrence and base case above to compute $f(i)$ for all i, and just print the maximum!

In pseudocode,

```
def max_sum_subarray(a, n):
    f[0..n-1]
    f[0] = a[0]
    ans = f[0]
    for i from 1 to n-1:
        f[i] = max(a[i], f[i-1] + a[i])
        ans = max(ans, f[i])
    return ans
```

This is called **dynamic programming**, and in this case, it runs very fast; specifically, in $O(n)$ time!

We can improve this algorithm even further by noticing that we don't need to store the whole `f` array all the time because at any point, we only the last element `f[i-1]` and and current element `f[i]`. Hence, we can do the following:

```
def max_sum_subarray(a, n):
    f = a[0]
    ans = f
    for i from 1 to n-1:
        f = max(a[i], f + a[i])
        ans = max(ans, f)
    return ans
```

We can interpret this algorithm in the following way: rewrite $f(i) = \max(a_i, f(i-1) + a_i)$ into

$$f(i) = \max(0, f(i-1)) + a_i = \begin{cases} f(i-1) + a_i & \text{if } f(i-1) \geq 0 \\ a_i & \text{if } f(i-1) < 0. \end{cases}$$

This means that at any step, we're calculating $f(i)$, the largest sum of any subarray ending at a_i, and we do that by trying to append it to the previous maximum $f(i-1)$, but if the previous maximum subarray happens to be negative, we ignore it instead and begin a new subarray $[a_i]$.

This algorithm is more popularly known as Kadane's algorithm.

 Tested by shashank21j

Problem Tester's code :

Python 2

```
for _ in range(input()):
    n = input()
    arr = map(int,raw_input().split())
    max_sum = -999999999
    c_sum = 0
    sum1 = 0
    for i in range(n):
        sum1+=arr[i]
```

```
        if arr[i]>0:
            c_sum += arr[i]
        if sum1>max_sum:
            max_sum = sum1
        if sum1<0:
            sum1 = 0
    if c_sum == 0:
        c_sum = max(arr)
    print max_sum,c_sum
```

CamelCase

locked

by nabila_ahmed

Problem Submissions Leaderboard Discussions Editorial

Editorial by Nabila_ahmed

Each word begins with a capital letter, so you can solve this problem by counting the number of capitalized characters and adding 1 to that number. This works because each capital letter signifies the start of a word, with the exception of the very first word which is lowercase (hence the $+1$).

Statistics

Difficulty: Easy
Time Complexity: O(n)
Required Knowledge: String
Publish Date: Jul 02 2016

Problem Setter's code :

C++

```
#include <bits/stdc++.h>
#include<assert.h>

using namespace std;
```

```
void solution() {

   string str;
   cin >> str;
   int len = str.size();
   int ans = 1;
   for(int i = 0; i < len; i++){
       if(str[i] >= 'A' && str[i] <= 'Z') {
           ans++;
       }
   }
   cout<<ans<<endl;
}


int main() {

        solution();

    return 0;
}
```

 Tested by Shafaet

Problem Tester's code :

Python 2

```
# Enter your code here. Read input from STDIN. Print output to STDOUT
s = raw_input()
ans = 1
assert len(s) >= 1 and len(s) <= 100000
for c in s:
    if ord(c) >= ord('A') and ord(c) <= ord('Z'):
        ans = ans + 1

print ans
```

Java (AllisonP)

```
import java.util.*;

public class Solution {

    public static void main(String[] args) {
        Scanner scan = new Scanner(System.in);
        String s = scan.next();
        scan.close();

        // use a regex matching to split the string on capital letters
        // the resulting array contains contiguous sections of lowercase le
tters
        String[] words = s.split("[A-Z]");
        // this works because the problem states that each word has at leas
t 2 characters, and we know that the first character of each word is always
 capitalized.

        System.out.println(words.length);
    }
}
```

Caesar Cipher

locked


by vatsalchanana

To encrypt the string, we need to rotate the alphabets in the string by a number k. If k is a multiple of 26, then rotating the alphabets in the string by k has no effect on the string. Rotation by k is same as rotation by k+26. So by taking modulo of k with 26, we get the number of times we need to rotate the alphabets of the string. To rotate the alphabets in a string by a value k, we add k to the character. If this value exceeds the range of the alpabets, we need to wrap it back. The range of uppercase characters is from 65 ('A') to 90 ('Z'). The range of lowercase characters is from 97 ('a') to 122 ('z').
Example: For the string : "middle-Outz" and k=2
We add 2 to 'm'. 'm' becomes 'o'. This value is within the ascii range so we don't need to wrap it. '-' remains unaltered. 'z' on adding 2 becomes 124. This value lies outside the range of lowercase characters. We need to wrap this value. By taking the modulo of this value with 122, and adding this value to 96('a'-1) we get the rotated character.

For lowercase characters,

Statistics

Difficulty: Easy
Time Complexity: O(N) Required
Knowledge: Modulo Operator
Publish Date: Jun 08 2015

```
// Let char c = s[i] be a lowercase character in the string.
k = k % 26;
c += k;
if(c > 122) {
    c = 96 + (c % 122);
}
```

Featured Solutions

Python 2 shashank21j

```
n = input()
s = list(raw_input())
k = input() % 26
temp = map(lambda x: ord(x), s)
for i in range(len(s)):
    if 65 <= temp[i] <= 90:
        temp[i] = 65 + ((temp[i] - 65) + k) % 26
    elif 97 <= temp[i] <= 122:
        temp[i] = 97 + ((temp[i] - 97) + k) % 26
print "".join(map(chr, temp))
```

Problem Setter's code :

C++

```
#include <iostream>
#include <string>

using namespace std;

int main() {
    int n;
    string s;
    cin >> n;
    cin >> s;
```

```
    int k;
    cin >> k;

    k %= 26;
    for (int i = 0;i < n; i++) {
        int c = s[i];
        if (c >= 'a' && c <= 'z') { // Lowercase characters
            c += k;
            if (c > 'z') { // C exceeds the ascii range of lowercase charac
ters.
               c = 96 + (c % 122); // wrapping from z to a
            }
        } else if(c >= 'A' && c <= 'Z') { // Uppercase characters
            c += k;
            if(c > 'Z') { // C exceeds the ascii range of uppercase charact
ers.
                c = 64 + (c % 90); //wrapping from Z to A
            }
        }
        cout << (char)c;
    }
    cout << endl;
    return 0;
}
```

Alternating Characters

locked

by amititkgp

Problem Submissions Leaderboard Discussions Editorial

Editorial by amititkgp

It is given in the question that the resultant string shouldn't have two adjacent matching characters.

If there are n adjacent matching characters, delete $n - 1$ of those characters and repeat this process to the end of the string.

See the implementation of the setter for more details.

Statistics

Difficulty: Easy

Time Complexity: O(|s|) per test case

Required Knowledge: Basics of any language

Publish Date: Sep 03 2014

Set by amititkgp

Problem Setter's code :

C++

```cpp
#include <cmath>
#include <cstdio>
#include <vector>
#include <iostream>
#include <algorithm>
using namespace std;

int main() {
    int t;
    cin >> t;
    while(t--) {
        string str;
        cin >> str;
        int ans = 0;
        for (int i = 0 ; i < str.length() - 1; i++) {
            if (str[i] == str[i + 1]) {
                // If two consecutive characters are the same, delete one o
f them.
                ans++;
            }
        }
        cout << ans << endl;
    }
    return 0;
}
```

 Tested by shashank21j

Problem Tester's code :

Python 2

```python
t = input()
for _ in range(t):
    s = raw_input()
    count = 0
    for i in range(1,len(s)):
```

```
        if s[i] == s[i-1]:
            count += 1
    print count
```

HackerRank in a String!

locked


by shashank21j


Editorial by StefanK

Observation

In order to determine if the characters in `hackerrank` are in an input string, we've got to go through the input string, character by character, and check off the characters in `hackerrank` when they're found. If the character in the input string doesn't match the current character we're looking for, we skip that input string's character and go to the next one. Don't forget that have to check off the characters of `hackerrank` in order too.

Solution

First we'll create a string or character array containing `hackerrank` called $hackerrank$. We'll initialize an index variable to 0. Next we'll iterate through the query string until we find $hackerrank[0] = h$ or reach the end of the string. From the next position in s, find $hackerrank[1] = a$ or end of string, and so on. If we find $hackerrank[9] = k$ by the

Statistics

Difficulty: Easy
Required Knowledge: implementation
Publish Date: Feb 06 2017

end of the string, we print `YES`. Otherwise, `hackerrank` is not a subsequence of s, so we print `NO`.

 Set by shashank21j

Problem Setter's code :

Python 2

```
n = input()
t = 'hackerrank'
for _ in range(n):
    s = raw_input()
    cnt = 0
    ctr = 0
    for i in t:
        while(ctr < len(s)):
            if s[ctr] ==  i:
                cnt += 1
                ctr += 1
                break
            ctr += 1
    if cnt == 10:
        print "YES"
    else:
        print "NO"
```

 Tested by AllisonP

Problem Tester's code :

Java (RegEx)

```java
import java.util.*;
import java.util.regex.*;

public class Solution {
    public static boolean containsHackerRank(String s) {
        // Check if string contains 'hackerrank' with
        // 0 or more other chars spaced around each char
        Pattern p = Pattern.compile(".*h.*a.*c.*k.*e.*r.*r.*a.*n.*k.*");
        Matcher m = p.matcher(s);

        return m.matches();
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int t = in.nextInt();
        for(int a0 = 0; a0 < t; a0++){
            String s = in.next();
            System.out.println((containsHackerRank(s)) ? "YES" : "NO");
        }
        in.close();
    }
}
```

Java (Iterative)

```java
import java.io.*;
import java.util.*;
import java.text.*;
import java.math.*;
import java.util.regex.*;

public class Solution {

    public static boolean containsHackerRank(String s) {
        boolean contains = false;
        char[] hackerrank = {'h', 'a', 'c', 'k', 'e', 'r', 'r', 'a', 'n',
'k'};
        int index = 0;
```

```
        // Iterate through each character in hackerrank
        for (int i = 0; i < hackerrank.length; i++) {
            // Set contains to false
            contains = false;

            // Search for current char in the rest of the string
            while (index < s.length()) {
                // if found, break
                if (hackerrank[i] == s.charAt(index)) {
                    contains = true;
                    index++;
                    break;
                }
                // else, check next char
                else {
                    index++;
                }
            }
            // stop checking characters if current char doesn't exist in re
st of string
            if (!contains) {
                return false;
            }
        }

        // Returns true if subsequence 'hackerrank' was found
        return contains;
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        int t = in.nextInt();
        for(int a0 = 0; a0 < t; a0++){
            String s = in.next();
            System.out.println((containsHackerRank(s)) ? "YES" : "NO");
        }
        in.close();
    }
}
```

Making Anagrams

locked

by amititkgp

Problem | Submissions | Leaderboard | Discussions | Editorial

Editorial by amititkgp

Two strings, a and b, will be anagrams of one another if they share all of the same characters and each character has the same frequency in both strings. Keep a count array for each of them that stores the number of occurrences of each of character. Suppose character c occurs x times in string a and y times in string b; in this case, we'll have to perform $max(x, y) - min(x, y)$ deletions for all of the characters.

Featured Solutions

Python 2

```
from collections import *
a = Counter(raw_input())
b = Counter(raw_input())
c = a - b
d = b - a
```

Statistics

Difficulty: Easy
Success Rate: 96.97%
Time Complexity: O(length of string)
Required Knowledge: Counting
Publish Date: Feb 26 2014

Of the 2769 contest participants, 1585(57.24%)submitted code for this challenge.

```
e = c + d
print len(list(e.elements()))
```

 Set by amititkgp

Problem Setter's code :

```
#include<bits/stdc++.h>

using namespace std;

int main() {
    string str1,str2;
    getline(cin,str1);
    getline(cin,str2);

    int A[26],B[26],i;
    for(i=0 ; i< 26 ; i++)
        A[i] = B[i] = 0;
    for(i = 0 ; i< str1.length() ; i++)
        A[(int)(str1[i] - 'a')]++;
    for(i = 0 ; i< str2.length() ; i++)
        B[(int)(str2[i] - 'a')]++;
    int outp = 0;
    for(i=0 ; i< 26 ; i++)
    {
        outp = outp + A[i] + B[i] - 2*min(A[i],B[i]);
    }
    cout<<outp<<endl;
    return 0;
}
```

 Tested by shashank21j

Problem Tester's code :

Haskell

```
import Data.List
main :: IO ()
main = getContents >>= print. parse
parse :: String -> Int
parse input =
  case lines input of
    [a, b] -> validate (a, b)
    _      -> error "there are not two lines"
validate :: (String, String) -> Int
validate (aa, bb)
  | length aa < 1 || length aa > 10000 = error "length of a is out of rang
e"
  | length bb < 1 || length bb > 10000 = error "length of b is out of rang
e"
  | otherwise = foldr (\(p, q) acc -> abs(p-q) + acc) 0 $ zip a b
    where
      a = map (length). group. sort $ aa ++ ['a'..'z']
      b = map (length). group. sort $ bb ++ ['a'..'z']
```

The Love-Letter Mystery

locked

 by amititkgp

 Editorial by amititkgp

Given a string, you have to convert it into a palindrome using a certain set of operations.

Let's assume that the length of the string is L, and the characters are indexed from 0 to $L-1$. The string will be a palindrome only if the characters at index i and the characters at index $(L-1-i)$ are the same. So, we need to ensure that the characters for all such indices are the same.

Let's assume that one such set of indices can be denoted by A' and B'. It is given that it takes one operation to decrease the value of a character by 1. So, the total number of operations will be $|A'-B'|$. We need to calculate this for all such pairs of indices, which can be done in a single *for* loop. Take a look at the setter's code.

 Set by amititkgp

Statistics

Difficulty: Easy
Success Rate: 97.41%
Time Complexity: $\mathcal{O}(N)$ Required
Knowledge: Implementation
Publish Date: May 15 2014

Of the 1601 contest participants, 771(48.16%)submitted code for this challenge.

Problem Setter's code :

C++

```
#include<bits/stdc++.h>
using namespace std;
int main()
{
    int t,i;
    cin>>t;
    while(t--)
    {
        string str;
        cin>>str;
        int s = 0;
        for(i=0 ; i< str.length()/2 ; i++)
        {
              s += abs(str[i] - str[str.length()-i-1]);
        }
        cout<<s<<endl;
    }
    return 0;
}
```

 Tested by shashank21j

Problem Tester's code :

Python 2

```
t = input()
assert t<=10 and t>=1
for _ in range(t):
    s = list(raw_input())
    assert len(s) >=1 and len(s)<= 10000
    su = 0
    for i in range(0,len(s)/2):
```

```
        su+= abs(ord(s[i]) - ord(s[-1-i]))
    print su
```

String Construction

locked

by ma5termind

Problem Submissions Leaderboard Discussions Editorial

Editorial by ma5termind

There is only a cost for each new *distinct* character being copied from string s_i to string p_i. Once a character is copied to p_i, it can be copied from p_i as a substring of length 1 and appended to the end of p_i at no cost. Therefore, the minimum cost to copy each string will always be the number of distinct characters in string s.

Set by ma5termind

Problem Setter's code :

C++

```
#include <bits/stdc++.h>
using namespace std;

int main() {
```

Statistics

Difficulty: Easy
Time Complexity: $O(m \cdot n)$
Required Knowledge: String, Implementation, Observation
Publish Date: May 17 2016

```
    int t; cin >> t;
    while( t-- ) {
        string s; cin >> s;
        int M[26] = {0}, ans = 0;
        for( auto it: s ) {
            if(!M[it-'a']) {
                ans ++;
            }
            M[it-'a'] = 1;
        }
        cout << ans << "\n";
    }
    return 0;
}
```

Highest Value Palindrome

locked

by nabila_ahmed

Problem Submissions Leaderboard Discussions Editorial

Consider the given number, S, as an array of characters. To make S palindromic:

1. Create 2 pointers, $left$ and $right$, where $left$ initially points to position 0 and $right$ initially points to position $n-1$.
2. Compare the digits referenced by $left$ and $right$; if they do not match, then overwrite the smaller value with the larger value. For every modified digit, decrement k.
3. Increment $left$ (moving it one position to the right), and decrement $right$ (moving it one position to the left).

Repeat steps 2 and 3 until $left$ and $right$ meet (i.e., the number is a palindrome). If k is negative, then it's not possible to make S a palindrome and you must print `-1`.

In the event that you do not use all k moves, then you can further maximize the number. To do this, you again perform step 1 (above). Then you overwrite the values for both $left$ and $right$ for any digit < 9 until you fully deplete your k moves (observe that each time

Statistics

Difficulty: Medium
Time Complexity: $O(N)$ Required
Knowledge: String, palindromes
Publish Date: Apr 29 2016

you perform this action, you are changing 2 digits). If you only have 1 move left and n is odd, you can use this move to increase the middle digit to 9.

Problem Setter's code :

C++

```
#include <bits/stdc++.h>
#include<assert.h>

using namespace std;

char ans[100005] = {'\0'};
bool mark[100005];

void solution() {

   memset(mark,0,sizeof(mark));
   int n, len, l, r, k;
   string str;

   cin>>n>>k;
   cin>>str;
   len = str.size();

   assert(n>0 && n<=100000);
   assert(k>=0 && k<=100000);
   assert(len>0 && len<=100000);

   //Making palindrome
   l=0; r=len-1;
   while(l<=r)
   {
           assert(str[l]>='0' && str[l]<='9');
           assert(str[r]>='0' && str[r]<='9');

           if(l==r)
           {
```

```
                ans[l] = str[l];
                break;
        }
        if(str[l] == str[r])
        {
                ans[l] = str[l];
                ans[r] = str[r];
        }
        else
        {
                if(str[l]>= str[r])
                {
                        mark[r] = 1;
                        k--;
                        ans[l] = ans[r] = str[l];
                }
                else
                {
                        mark[l] = 1;
                        k--;
                        ans[l] = ans[r] = str[r];
                }
        }
        l++;
        r--;
}

if(k<0)
{
        printf("-1\n");
        return;
}

//Maximizing number
l=0; r=len-1;
while(l<=r)
{
        if(l==r)
        {
                if(ans[l]<'9' && k>=1)
                     ans[l] = '9';
                break;
```

```
            }
            if(ans[l]<'9')
            {
                    if(mark[l] == 0 && mark[r] == 0 && k>=2) //not touch bef
ore
                    {
                            k-=2;
                            ans[l] = ans[r] = '9';
                    }
                    else if((mark[l]==1 || mark[r]==1) && k>=1)
                    {
                            k-=1;
                            ans[l] = ans[r] = '9';
                    }
            }
            l++;
            r--;
    }
    ans[len] = '\0';
    printf("%s\n", ans);
}

int main () {

                solution();
        return 0;
}
```