A string S consisting only of the letters "A", "B" and "C" is given. This string can be transformed according to one of the following rules:

1. substitute one occurrence of "AB" with "AA",
2. substitute one occurrence of "BA" with "AA",
3. substitute one occurrence of "CB" with "CC",
4. substitute one occurrence of "BC" with "CC",
5. substitute one occurrence of "AA" with "A",
6. substitute one occurrence of "CC" with "C".

The rule is *useful* if we can transform the string by using it. If there is at least one useful rule, one of the useful rules is picked at random and the string is transformed according to that rule (exactly one occurrence should be substituted). As long as there are useful rules, the above process should be repeated.

Write a function:

```
def solution(s)
```

that, given a string S consisting of N characters, returns any string that can result from a sequence of transformations as described above.

For example, given string S = "ABBCC" the function may return "AC", because this is one of the possible sequences of transformations:

- "ABBCC": useful rules are 1, 4 and 6. We pick rule number 1;
- "AABCC": useful rules are 1, 4, 5 and 6. We pick rule number 5;
- "ABCC": useful rules are 1, 4 and 6. We pick rule number 4;
- "ACCC": we pick rule number 6;
- "ACC": we pick rule number 6;
- '"AC"': there are no useful rules, so no further rule can be applied.

Assume that:

- the length of S is within the range [0..50,000];
- string S consists only of the following characters: "A", "B" and/or "C".

Complexity:

- expected worst-case time complexity is O(N);
- expected worst-case space complexity is O(N) (not counting the storage required for input arguments).

An anagram is a word that can be obtained by rearranging the letters of another word, using all the constituent letters exactly once. For example, the words "`admirer`" and "`married`" are anagrams of one another. However, the words "`rather`" and "`harder`" are not anagrams, since "`rather`" does not contain the letter '`d`' and "`harder`" does not contain the letter '`t`'. Two identical words are anagrams, too. You may also notice that the anagram of any word must be of the same length as the word itself.

Bob has written two words, A and B, consisting of N and M letters, respectively. He would like them to be anagrams of each other. Bob has decided to add some letters to these words to achieve his goal.

For example, given the words "`rather`" and "`harder`", Bob can add the letter '`d`' to the first word and the letter '`t`' to the second word. Having done this, the words are now anagrams.

Bob would like to add as few letters as possible. In the example above, Bob added two letters, which is the minimum possible. Your task is to tell him the minimum number of letters that he needs to add to make the given words anagrams in this way.

Write a function:

```
def solution(a, b)
```

```
def solution(a, b)
```

that, given two non-empty strings, A and B, consisting of N and M letters respectively, returns the minimum number of letters that Bob has to add to the words specified in A and B to make them anagrams.

For example, given the words "`rather`" and "`harder`" your function should return 2, as explained above.

For the given words "`apple`" and "`pear`" your function should return 3, since you can add the letter '`r`' to the first word and the letters '`p`' and '`l`' to the second word to form anagrams.

And for the given words "`lemon`" and "`melon`", your function should return 0, since the given words are already anagrams.

Assume that:

- N and M are integers within the range [1..100,000];
- string ('A', 'B') consists only of lowercase letters (`a` - `z`).

Complexity:

- expected worst-case time complexity is O(N+M);
- expected worst-case space complexity is O(1) (not counting the storage required for input arguments).

A non-empty zero-indexed array A consisting of N integers is given. The array consists only of integers in the range [0..N−1]. Each element of the array can be treated as a pointer to another element of the array: if A[K] = M then element A[K] points to A[M].

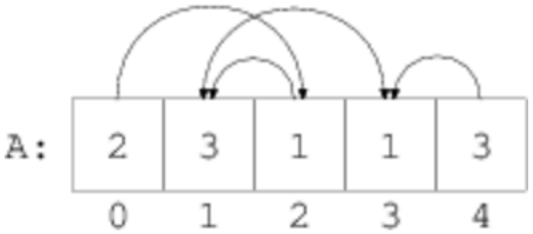
The array defines a sequence of jumps of a pawn as follows:

- initially, the pawn is located at position 0;
- on each jump the pawn moves from its current position K to A[K];
- the pawn jumps forever.

Since the number of possible positions of the pawn is finite, eventually, after some initial sequence of jumps, the pawn enters a cycle. Compute the length of this cycle.

For example, for the following array A:

```
A[0] = 2    A[1] = 3    A[2] = 1
A[3] = 1    A[4] = 3
```



consecutive positions of the pawn are: 0, 2, 1, 3, 1, 3, 1, 3, ..., and the length of the cycle is 2.

Write a function:

```
def solution(a)
```

that, given a non-empty zero-indexed array A consisting of N integers in the range [0..N−1], returns the length of the cycle that the pawn eventually enters, as described above. For example, given the array shown above, the function should return 2.

Assume that:

- N is an integer within the range [1..200,000];
- each element of array A is an integer within the range [0..N−1].

Complexity:

- expected worst-case time complexity is O(N);
- expected worst-case space complexity is O(1), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.

A zero-indexed array A consisting of N integers is given. A triplet (P, Q, R) is called a *triangle* if $0 \leq P < Q < R < N$ and:

- A[P] + A[Q] > A[R],
- A[Q] + A[R] > A[P],
- A[R] + A[P] > A[Q].

The perimeter of such a triangle equals A[P] + A[Q] + A[R].

For example, consider the following array A:

```
A[0] = 10
A[1] = 2
A[2] = 5
A[3] = 1
A[4] = 8
A[5] = 20
```

Triplet (0, 2, 4) is a triangle and its perimeter equals 10 + 5 + 8 = 23. There is no other triangle in this array with a larger perimeter.

Write a function:

```
def solution(a)
```

that, given a zero-indexed array A of N integers, returns the maximum perimeter of any triangle in this array. The function should return −1 if there are no triangles.

For example, given:

```
A[0] = 10
A[1] = 2
A[2] = 5
A[3] = 1
A[4] = 8
A[5] = 20
```

the function should return 23, as explained above.

Given array A such that:

```
A[0] = 5
A[1] = 10
A[2] = 18
A[3] = 7
A[4] = 8
A[5] = 3
```

the function should return 25: the triangle with the maximum perimeter is (1, 3, 4).

While for an array A:

```
A[0] = 10
A[1] = 20
A[2] = 30
```

the function should return −1, as it is impossible to create a triangle.

Assume that:

- N is an integer within the range [0..100,000];

Assume that:

- N is an integer within the range [0..100,000];
- each element of array A is an integer within the range [1..100,000,000].

Complexity:

- expected worst-case time complexity is O(N*log(N));
- expected worst-case space complexity is O(N), beyond input storage (not counting the storage required for input arguments).

Elements of input arrays can be modified.