Keyboard (keyboard)

BY ANDRES UNT (ESTONIA)

Let's think of Juku's alphabet as an undirected, unweighted graph. Every letter is a vertex, and if two of them appear consecutively in one of the common words, there is an edge between them. It is possible to divide the letters between the left and the right half of the keyboard iff this graph is *bipartite*. This property can be checked easily: Start a depth-first search from each vertex that was not visited yet, and color the nodes alternatingly either *red* or *blue*. If this does not work, because some vertex needs to be assigned both colors simultaneously (i.e. there is a cycle of odd length in the graph), just print `impossible`. Otherwise, for 10% of the achievable score, it is allowed to print any non-negative integer and to terminate at this point.

N, the number of characters in the alphabet, is at the same time the number of vertices. Let's denote by E the number of edges, which is at most 1 000 000 – M (to simplify things, let's just think of E as being bounded by 1 000 000). The running time of the algorithm so far is $O(N + E)$.

Subtask 1 (40 points). $N \leq 5\,000$

In this subtask, the number of vertices is bounded by 5 000. Let's assume the algorithm discussed above has already been executed and it answered that it is *not impossible* to build an ergonomic keyboard. Therefore, the remaining task is to partition the vertices of the graph into two sets, such that the two endpoints of no edge are both in the same set and the absolute difference between the sizes of the sets is minimal.

In addition to whether the graph is bipartite or not, we need some more information: How the graph is partitioned into connected components. For each of these components, we also need to know how many of its vertices have been colored red, and how many have been colored blue. Assume a component had r red and b blue vertices. Now, Juku can either put the r red vertices on the left side of the keyboard and the b blue ones on the right side or the other way round. Thus, the difference between the sizes of the two sides either changes by $r - b$ or by $b - r$ (this way, the difference can become negative, although we are interested in the minimum *absolute* difference in the end).

Now we can reformulate the problem: For each component, we insert the value $abs(r - b)$ into a multiset S (the actual number of red or blue vertices is irrelevant, only the *difference* in the sizes of the two bipartition parts in a connected component matters). Choose a sign (either positive or negative) for each element in S, such that the sum of all (signed) elements is as close as possible to zero. The absolute value of this sum is the expected output. You can think of an element having a positive sign as a connected component of which the larger part of its bipartition is located on the left side of the keyboard. If the sign is negative, the larger part of the bipartition is on the right side.

To solve this problem, an algorithm similar to a Dynamic Programming solution for Knapsack can be used. Let's denote the elements of S by $S = \{s_1, \ldots, s_{|S|}\}$. Then $\mathrm{dp}[i][w]$ is a boolean that is true if it is possible to assign signs to the first i elements in S, such that these signed elements sum up to exactly w. The Dynamic Programming values can be calculated as

$$\mathrm{dp}[i][w] = \mathrm{dp}[i-1][w - s_i] \text{ or } \mathrm{dp}[i-1][w + s_i].$$

Since the sum of elements in S is bounded by N, w is in the interval $[-N, N]$. Thus, to make the indices for the DP array non-negative, in an implementation for this, it may be necessary to add a fixed offset of N to each w.

When all these values have been calculated, the result is the minimum $|w|$ for which $\mathrm{dp}[|S| - 1][w]$ is true. The running time of this algorithm is $O(|S| \cdot N) \leq O(N^2)$.

Subtask 2 (30 points). $N \leq 100\,000$

Two possible optimizations would solve the second subtask where N may be up to 100 000.

The first one is to use bitsets to "parallelize" the computation of DP. The easiest way to implement this is by using the bitset class from the C++ standard library. If you want to implement it yourself, it works like this: Encode the 64 bits of $\mathrm{dp}[i][w], \ldots, \mathrm{dp}[i][w+63]$ in a single 64-bit integer. This has to be done for w that are multiples of 64 only. To calculate this integer, just take the *bitwise or* of $\mathrm{dp}[i][w-s_i], \ldots, \mathrm{dp}[i][w-s_i+63]$ and $\mathrm{dp}[i][w+s_i], \ldots, \mathrm{dp}[i][w+s_i+63]$. If s_i itself is not a multiple of 64, each of these two integers has to be constructed by sticking two of the previously calculated integers together. Although the asymptotic running time does not change, this speed-up with a factor of 64 suffices to solve the second subtask. Internally, the bitsets of the C++ standard library are implemented in a similar way and give the same speed-up factor.

Another optimization that solves this subtask is to de-duplicate the values in S. As long as a value s occurs at least three times in S, replace these three with one occurrence of s and one of $2s$. This way, all four values $-3s$, $-s$, s, and $3s$, which can be constructed by summing the three occurrences of s with arbitrary signs, can still be constructed by choosing appropriate signs for s and $2s$. Therefore, the replacement does not change anything in the result. However, it does change the asymptotic running time to $O(\sqrt{N} \cdot N)$ because doing this optimization makes S have size at most $O(\sqrt{N})$: Even if the elements of S are as small as possible, these have to be $1, 1, 2, 2, 3, 3, \ldots$ (because no number may occur more than twice). By taking the Gaussian sum, this means that the sum of all elements in S is at least $\Omega(|S|^2)$. But since this sum is bounded by N, $|S|$ can not be larger than $O(\sqrt{N})$.

Subtask 3 (30 points). No further constraints.

The complete problem can be solved by combining the two mentioned optimizations. The asymptotic running time changes to $O(\sqrt{N} \cdot N)$ by doing the second one, and the solution speeds up by a factor of 64 through the first one.