Vim

For any string s, let $f(s)$ denote the minimum number of key presses Victor needs to do to remove all "e"s from s.

For any string s with some letters underlined, let $g(s)$ denote the minimum number of key presses Victor needs to do so that the cursor is at every underlined letter at least once.

Lemma 1. *$f(l_1 s_1 \underbrace{\text{e}\dots\text{e}}_{n_2} l_2 s_2 \dots \underbrace{\text{e}\dots\text{e}}_{n_k} l_k s_k) = 2(n_2+\dots+n_k)+g(\underline{l_1} s_1 \dots \underline{l_k} s_k)$ if $l_1, \dots, l_k$ are letters different from "e" and $n_2, \dots, n_k \geq 1$.*

Proof. Let $t = l_1 s_1 \underbrace{\text{e}\dots\text{e}}_{n_2} l_2 s_2 \dots \underbrace{\text{e}\dots\text{e}}_{n_k} l_k s_k$ and $s = \underline{l_1} s_1 \dots \underline{l_k} s_k$.

"$\leq$" Let $o_1, \dots, o_p$ be some operations on s that visit each letter at least once. At the first point, when the cursor is at the letter $\underline{l_i}$ (for $2 \leq i \leq k$), insert n_i times the operations hx. This yields a sequence of $2(n_2 + \dots + n_k) + p$ operations on t that delete all the "e"s.

"$\geq$" Victor has to do the operation x exactly $n_2 + \dots + n_k$ times (once for each "e"). To move the cursor to some letter "e", he has to move it to the following letter and then issue h. Therefore, the cursor visits the letters $l_1, \dots, l_k$ at least once.

□

Using Lemmas 1 and 2, it only remains to find $g(s)$ for strings s not containing the letter "e" (with some letters underlined). This is an instance of the travelling salesman problem, but the corresponding graph has some additional structure described by the following two lemmas.

Lemma 2. *If the cursor is at some position p and Victor does some sequence of operations* h, $o_1, \dots, o_k$ *with $o_1 \neq$* h *(or $k = 0$), then Victor can instead do a shorter sequence of operations which visit the same letters, apart from at most the letter at position $p-1$.*

Proof. Let l be the letter at position p. If $o_1 \neq$ fl (or $k = 0$), then Victor can do the operations $o_1, \dots, o_k$. Otherwise, he can do the operations $o_2, \dots, o_k$.

□

Lemma 3. *Assume Victor does the minimum number of key presses to visit each underlined letter in s at least once. Then Victor will not do operation* h *more than once with the cursor at the same position.*

Proof. Otherwise, Victor does the operation h two times with the cursor at the same position and such that one of the following both operations is different from h. Therefore, Victor could shorten his sequence of operations according to Lemma 3. □

It follows, that Victor gets the minimum number of key presses like in the following picture where each underlined character (represented by a dot in the picture) is contained in one of the intervals spanned by the horizontal arrows from right to left (which can have length 0) and each such arrow ends at an underlined letter.

This observation yields an $\mathcal{O}(N^2S)$ solution: It is easy to find a suitable recurrence relation for the minimum number $r(a, b)$ of key presses to visit all underlined letters up to position a with the cursor ending up at position b.

An $\mathcal{O}(NS^2)$ solution works roughly as follows:

For simplicity, assume that operation fc moves the cursor infinitly far away to the right if the character c does not appear anywhere to the right of the current cursor position. Apparently, this does not change the answer.

Let $p(a, c)$ be the minimum number of key presses before or after which the cursor is to the left of a such that the cursor lands on or passes position a exactly once and with operation fc.

Let $q(a, c, d)$ be the minimum number of key presses before or after which the cursor is to the left of a such that the cursor lands on or passes position a exactly three times: the first time with operation fc, the second time with operation h and the third time with operation fd.

Let s_i be the i-th character of the string, let

$$u_i = \begin{cases} \infty, & \text{if the } i\text{-th character is underlined} \\ 0, & \text{otherwise} \end{cases}$$

and let

$$\kappa_{c,d} = \begin{cases} \infty, & c = d \\ 0, & c \neq d \end{cases}$$

Then the following recurrence relations hold:

$$\begin{aligned} p(a+1, c) = \min(&p(a, c) + \kappa_{c,s_a} + u_a, \\ &p(a, s_a) + 2, \\ &q(a, s_a, c) + \kappa_{c,s_a}, \\ &q(a, s_a, s_a) + 2) \end{aligned}$$

$$\begin{aligned} q(a+1, c, d) = \min(&p(a, c) + 3 + \kappa_{c,s_a}, \\ &p(a, s_a) + 5, \\ &q(a, c, d) + 1 + \kappa_{c,s_a} + \kappa_{d,s_a}, \\ &q(a, c, s_a) + 3 + \kappa_{c,s_a}, \\ &q(a, s_a, d) + 3 + \kappa_{d,s_a}, \\ &q(a, s_a, s_a) + 5) \end{aligned}$$

Then, we have $g(s) = p(N, k) - 2$ where N is an (imaginary) position directly to the right of the last character of s and k is some (imaginary) letter not contained in s.

Remark. The condition that the first character of Victor's document ist not an "e" is in fact not necessary due to the following lemma.

Lemma 4. *$f(\underbrace{\text{e}\ldots\text{e}}_{n \text{ times}}s) = n + f(s)$ for any $n \geq 0$.*

Source code

```
#include <stdio.h>
#include <algorithm>
using namespace std;

const int NMAX = 70000;
const int S = 11;

char line[NMAX+10]; // the input
int Nin, N = 0, rewritecost = 0;
// input with characters converted to numbers from 0 to S-1 and 'e' removed
int text[NMAX+2];
bool must[NMAX+2]; // whether we have to reach this character

int dp1[NMAX][S];
int dp2[NMAX][S][S];

int main() {
        scanf("%d %s", &Nin, line);
        bool nextmust = false;
        for (int i = 0; i < Nin; i++) {
                if (line[i] == 'e') {
                        rewritecost++;
                        if (N) {
                                rewritecost++;
                                nextmust = true;
                        }
                } else {
                        must[N] = nextmust;
                        nextmust = false;
                        text[N] = line[i]-'a';
                        N++;
                }
        }
        for (int s = 0; s < S; s++) {
                dp1[0][s] = 1E9;
                for (int t = 0; t < S; t++)
                        dp2[0][s][t] = 1E9;
        }
        dp1[0][text[0]] = 0;
        for (int i = 1; i <= N; i++) {
                for (int s = 0; s < S; s++) {
                        int h = 1E9;
                        int p = text[i-1];
                        if (p != s && !must[i-1])
                                h = min(h, dp1[i-1][s]);
                        h = min(h, dp1[i-1][p]+2);
                        if (p != s)
                                h = min(h, dp2[i-1][p][s]);
                        h = min(h, dp2[i-1][p][p]+2);
                        dp1[i][s] = h;
                        for (int t = 0; t < S; t++) {
                                h = 1E9;
```

```
                                        if (p != s)
                                                h = min(h, dp1[i-1][s]+3);
                                        h = min(h, dp1[i-1][p]+5);
                                        if (p != s && p != t)
                                                h = min(h, dp2[i-1][s][t]+1);
                                        if (p != s)
                                                h = min(h, dp2[i-1][s][p]+3);
                                        if (p != t)
                                                h = min(h, dp2[i-1][p][t]+3);
                                        h = min(h, dp2[i-1][p][p]+5);
                                        dp2[i][s][t] = h;
                                }
                        }
                }
        printf("%d\n", dp1[N][S-1]+rewritecost-2);
        return 0;
}
```