OMI 2022 - Solución para *Frase Oculta*

SUBTAREA 1: Las palabras están formadas por letras iguales "*aaaa*" o "*ccc*".

El que una palabra aparezca en el texto depende únicamente de que el texto contenga al menos tantas letras de ese tipo como la palabra.

El siguiente código resuelve esta subtarea:

```
std::cin >> t >> n >> st;
for (int i = 0; i < t; ++i) letra[st[i] - 'a']++;

for (int i = 1; i <= n; ++i){
  std::cin >> largo >> palabra[i];
  if(letra[palabra[i][0] - 'a'] >= largo) std::cout << "1\n";
  else std::cout << "0\n";
}
```

SUBTAREA 2: Sólo hay una palabra y es *omi*.

Para esta subtarea basta con buscar en el texto que existan las tres letras en el orden adecuado.

```
bool o, m, i;
o = m = i = false;

// LETRA POR LETRA BUSCA QUE EL TEXTO TENGA PRIMERO UNA o, LUEGO UNA m Y
// FINALMENTE UNA i
for (int pos = 0; pos < t; ++pos) {
  if (!o && st[pos] == 'o')
    o = true;
  else if (o && !m && st[pos] == 'm')
    m = true;
  else if (o && m && !i && st[pos] == 'i')
    i = true;
}

if (o && m && i)
  std::cout << "1\n";
else
  std::cout << "0\n";
```

SUBTAREAS 3 y 4: $N \leq 10$ y $T, N \leq 1000$

Los límites de la subtarea permiten revisar cada palabra de forma independiente.

Observación: Una palabra aparece en el texto si todas sus letras existen en el texto en el orden correcto.

Por lo tanto basta con ir buscando en el texto letra por letra de la palabra y una vez que se encuentre, buscar la siguiente letra a partir de esa posición.

```
for (int p = 1; p <= n; ++p) {
  palabra[p] += '_';  // AGREGA UN MARCADOR AL FINAL PARA NO PASARTE
  posPalabra = 0;     // EMPIEZA EN LA PRIMERA LETRA DE LA PALABRA
  for (int posTexto = 0; posTexto < t; ++posTexto) {
    if (palabra[p][posPalabra] == st[posTexto]) {
      // SI LAS LETRAS SON IGUALES, AVANZA UNO LA POSICION EN LA PALABRA
      ++posPalabra;
      if (palabra[p][posPalabra] == '_') break;
    }
  }

  // SI LLEGO AL MARCADOR FINAL, SI LA ENCONTRO, ESCRIBE UNO Y TERMINA.
  if (palabra[p][posPalabra] == '_')
    std::cout << "1\n";
  else
    std::cout << "0\n";
```

Para cada palabra se requiere revisar todo el texto, por lo que la complejidad es de $O(TN)$

SUBTAREA 5: Sin restricciones adicionales.

Revisar palabra por palabra es lento para los límites generales del problema. Hay dos posibles caminos de optimización:

Optimizar el proceso de búsqueda de la próxima letra de la palabra en el texto.

Esto puede lograrse guardando todas las posiciones en las que aparece una letra en un `set`, de ese modo cuando se vayan buscando cada una de las letras de una palabra se puede encontrar la siguiente aparición de la letra deseada en el texto en un tiempo logarítmico.

```
for (int p = 1; p <= n; ++p) {
  bool ok = true;
  posTexto = -1;
  for (int l = 0; (std::size_t)l < palabra[p].size(); ++l) {
    letra = palabra[p][l] - 'a';
    auto it = posicionLetra[letra].upper_bound(posTexto);
    if (it == posicionLetra[letra].end()) {
      ok = false;
      break;
    }
    posTexto = (*it);  // AVANZA LA POSICION DEL TEXTO
  }
  if (ok)
    std::cout << "1\n";
  else
    std::cout << "0\n";
}
```

Esta solución por cada letra de las palabras hace una búsqueda logarítmica en el texto, de modo que la complejidad es $O(largo_total_de_palabras * logT)$

En vez de revisar con cada letra del texto todas las palabras, revisar únicamente las palabras a las que les puede servir.

Lo que se busca en esta solución es para cada letra del texto, afectar únicamente las palabras a las que esta letra les sirve y no invertir tiempo en las demás.

Crea un vector por cada letra. En ese vector estarán todas las palabras que *están buscando* esa letra. Recorre el

texto y con cada letra *avanza* todas las palabras que la estaban esperando. Cuando avances una palabra estará esperando una nueva letra, sácala del vector actual e insértala en el vector que corresponde.

Cada letra de cada palabra será insertada y extraída de un vector a lo más una vez. De igual modo cada letra del texto se revisa una vez, por lo tanto la complejidad de esta solución es $O(T + largo_total_palabras)$

📋

```
// PARA CADA PALABRA INSERTALA EN UNA COLA DEPENDIENDO DE CUAL
// ES LA SIGUIENTE LETRA QUE BUSCA
for (int i = 1; i <= n; ++i) {
  std::cin >> largo[i] >> palabra[i];
  posPalabra[i] = 0;
  palabrasConLetra[palabra[i][0] - 'a'].push_back(i);
}

// AHORA VE RECORRIENDO EL TEXTO COMPLETO Y CADA QUE ENCUENTRES UNA LETRA
// AVANZA TODAS LAS PALABRAS QUE ESTAN ESPERANDO ESA LETRA
for (int i = 0; i < t; ++i) {
  letra = st[i] - 'a';
  std::vector<int> tmp;
  while (palabrasConLetra[letra].size()) {
    int p = palabrasConLetra[letra].back();
    ++posPalabra[p];
    if (posPalabra[p] == largo[p])
      res[p] = 1;
    else
      tmp.push_back(p);
    palabrasConLetra[letra].pop_back();
  }

  for (auto p : tmp) {
    letra = palabra[p][posPalabra[p]] - 'a';
    palabrasConLetra[letra].push_back(p);
  }
}

// ESCRIBE LOS RESULTADOS
for (int i = 1; i <= n; ++i) std::cout << res[i] << "\n";
```