OIJ (rozwiązanie)

Autorzy zadania: **Bartosz Łukasiewicz, Jacek Tomasiewicz**
Opracowanie: **Tomasz Nowak, Bartosz Podkanowicz**
Opis rozwiązania: **Tomasz Nowak**

Olimpiada Informatyczna Juniorów

Najpierw skupmy się na znalezieniu najmniejszego początkowego fragmentu, który zawiera napis `oij` jako podciąg. Zaczynając od początku słowa, można znaleźć pierwsze wystąpienie litery `o`, następnie znaleźć pierwsze wystąpienie `i` po znalezionym `o`, następnie znaleźć pierwsze wystąpienie `j` po znalezionym `i`. Pozwala to już rozwiązać pierwsze podzadanie (jeżeli da się znaleźć podciąg `oij` należy wypisać 1, zaś kiedy nie uda się znaleźć kolejno litery `o`, `i` lub `j` to wiadomo, że ciąg na wejściu nie zawiera żadnego `oij` jako podciąg, więc należy wypisać `NIE`), jednak to nas nie zadowala – chcemy znaleźć minimalną liczbę fragmentów, na jaką można podzielić ciąg z wejścia, by każdy zawierał jako podciąg słowo `oij`.

Pierwszy fragment można zakończyć po znalezionej literce `j` – na pewno nie jest to możliwe wcześniej, a zarazem nie ma potrzeby przedłużania go dalej. Wtedy o tym fragmencie można zapomnieć i ponownie szukać pierwszego wystąpienia `oij` jako podciąg. Taki algorytm powtarzamy tak długo, jak się da, a wynik jest liczbą jego udanych kroków. Jeżeli na końcu okaże się, że powstał niezamknięty fragment, można go doczepić do poprzedniego fragmentu (przedłużenie jakiegoś fragmentu zawierającego słowo `oij` jako podciąg nie sprawi, że ten podciąg zniknie, więc można bezpiecznie doczepić).

Na przykładzie, ten algorytm zachłanny podzieli ciag `zjobjoijsjooojjjjiiojojijjbwa`
na fragmenty `zjobjoij`, `sjooojjjjiioj`, `ojijjbwa` (zwróć uwagę na to, że ostatni fragment został przedłużony do końca słowa).

Opisany algorytm zachłanny działa w złożoności $O(N)$ (zarówno czasowej, jak i pamięciowej, przy czym N to długość ciągu z wejścia).

oij.py

```
1  def main():
2    napis = input()
3
4    # Wynikiem będzie liczba fragmentów,
5    # szukana literka będzie oznaczać, którą kolejno
6    # literkę w napisie oij szukamy, tj.
7    # szukana_literka=0 oznacza, że szukamy 'o',
8    # szukana_literka=1 oznacza, że szukamy 'i',
9    # a szukana_literka=2 oznacza, że szukamy 'j'.
10   wynik, szukana_literka = 0, 0
11   # Idziemy kolejno po literkach słowa.
12   for litera in napis:
13     # Jeżeli szukamy 'o' i znaleźliśmy tą literę,
14     # to dalej będziemy szukali 'i'.
15     if szukana_literka == 0 and litera == 'o':
16       szukana_literka += 1
17     # Szukamy 'i' i znaleźliśmy 'i'.  Dalej szukamy 'j'.
18     elif szukana_literka == 1 and litera == 'i':
19       szukana_literka += 1
20     # Znaleźliśmy 'j', czyli mamy całe słowo.  Zwiększamy wynik
21     # i szukamy kolejnej literki 'o'.
22     elif szukana_literka == 2 and litera == 'j':
23       szukana_literka = 0
24       wynik += 1
25
26   # Jeżeli wynikiem jest 0, wypisz NIE.
27   if wynik == 0: print("NIE")
28   # W przeciwnym wypadku, wypisz wynik.
29   else: print(wynik)
30
31 main()
```


Olimpiada finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2019/2020, 2020/2021, 2021/2022".

oij.cpp

```cpp
#include <iostream>
using namespace std;

int main() {
  // Wczytywanie wejścia.
  string napis;
  cin >> napis;

  // Wynikiem będzie liczba fragmentów,
  // szukana literka będzie oznaczać, którą kolejno
  // literkę w napisie oij szukamy, tj.
  // szukana_literka=0 oznacza, że szukamy 'o',
  // szukana_literka=1 oznacza, że szukamy 'i',
  // a szukana_literka=2 oznacza, że szukamy 'j'.
  int wynik = 0, szukana_literka = 0;
  // Idziemy kolejno po literkach słowa.
  for (int i = 0; i < int(napis.size()); ++i) {
    // Jeżeli szukamy 'o' i znaleźliśmy tą literę,
    // to dalej będziemy szukali 'i'.
    if (szukana_literka == 0 and napis[i] == 'o')
      ++szukana_literka;
    // Szukamy 'i' i znaleźliśmy 'i'.  Dalej szukamy 'j'.
    else if (szukana_literka == 1 and napis[i] == 'i')
      ++szukana_literka;
    // Znaleźliśmy 'j', czyli mamy całe słowo.  Zwiększamy wynik
    // i szukamy kolejnej literki 'o'.
    else if (szukana_literka == 2 and napis[i] == 'j') {
      szukana_literka = 0;
      ++wynik;
    }
  }

  // Jeżeli wynikiem jest 0, wypisz NIE.
  if (wynik == 0) cout << "NIE" << "\n";
  // W przeciwnym wypadku, wypisz wynik.
  else cout << wynik << "\n";
}
```

Takie rozwiązanie zachłanne wypada zwykle udowodnić formalnie. Poniżej przytoczymy taki dowód opierający się na tradycyjnym dowodzie algorytmów zachłannych. Załóżmy, że nasz algorytm zachłanny podzielił ciąg na fragmenty kończące się w miejscach $z_1 < z_2 < \cdots < z_p$ (przy czym załóżmy tutaj, że z_p nie musi być równe długości ciągu na wejściu, bo można ten ostatni fragment po prostu przedłużyć do końca ciągu), oraz istnieje inny, lepszy algorytm, który podzielił ciąg na fragmenty kończące się w miejscach $l_1 < l_2 < \cdots < l_q$ (czyli $q < p$). Udowodnimy teraz, że zachodzi sprzeczność, czyli ten „lepszy" algorytm jest nie gorszy od naszego zachłannego. Zacznijmy od przyjrzenia się liczbom z_1 oraz l_1. Musi zachodzić $z_1 \leq l_1$, ponieważ nasz algorytm zachłanny znajduje minimalne możliwe z_1. Jeżeli zachodzi $l_1 < z_1$, to można bezpiecznie koniec pierwszego fragmentu w lepszym algorytmie przesunąć w lewo (wtedy dalej ciąg końców fragmentów lepszego algorytmu będzie poprawny, pomimo tego, że został lekko zmieniony) do momentu, aż $l_1 = z_1$. Takie rozumowanie można indukcyjnie kontynuować na z_2 oraz l_2, potem z_3 oraz l_3, aż do z_q oraz l_q. Wtedy okaże się, że $z_i = l_i$ dla $1 \leq i \leq q$. Wychodzi sprzeczność, ponieważ założyliśmy, że ciąg lepszego algorytmu jest krótszy niż ciąg naszego zachłannego algorytmu, więc nasz algorytm daje poprawny wynik.

Olimpiada finansowana jest ze środków Ministerstwa Edukacji Narodowej w ramach zadania publicznego „Organizacja i przeprowadzenie olimpiad i turniejów w latach szkolnych 2019/2020, 2020/2021, 2021/2022".