Zygzak (rozwiązanie)

Autor zadania: **Karol Pokorski**
Opracowanie: **Michał Górniak, Alicja Kluczek**
Opis rozwiązania: **Alicja Kluczek, Bartosz Kostka**

Olimpiada Informatyczna Juniorów

Pierwszym krokiem rozwiązania jest określenie rozmiaru prostokąta, który narysowała Bajtosia. Aby otrzymać te wartości, możemy zliczyć ile razy Bajtosia narysowała kreskę w prawo (to będzie szerokość prostokąta) oraz ile razy narysowała kreskę w górę (to będzie wysokość prostokąta). Oznaczmy te wartości odpowiednio przez p i q, tak jak w treści zadania.

Jeśli potrafimy skrócić nasz ułamek, to część rozwiązania za nami. Znaną ze szkoły metodą jest obliczenie największego wspólnego dzielnika p i q, a następnie skrócenie ułamka dzieląc licznik i mianownik przez tę wartość. Jak zatem znaleźć największy wspólny dzielnik (będziemy często używać skrótu *gcd*, pochodzącego od nazwy *greatest common divisor*) dwóch liczb całkowitych? Najprostszą metodą, którą mogliśmy użyć w tym zadaniu było sprawdzenie każdej liczby całkowitej od 1 do $\min(p, q)$. Największa spośród sprawdzonych liczb, która dzieliła zarówno p i q jest liczbą szukaną. Poniższe dwa kody przedstawiają implementację funkcji `naiwneNWD`, która pokazuje tą metodę liczenia największego wspólnego dzielnika.

naive_gcd.cpp

```cpp
int naiwneNWD(int p, int q) {
  int d = 1;
  for (int i = 1; i <= min(p, q); i++) {
    if (p % d == 0 and q % d == 0) d = i;
  }
  return d;
}
```

naive_gcd.py

```python
def naiwneNWD(p, q):
  d = 1
  for i in range(1, min(p, q) + 1):
    if p % d == 0 and q % d == 0: d = i
  return d
```

Inną metodą jest użycie algorytmu Euklidesa[1], który korzysta z faktu, że $\gcd(a, b) = \gcd(b, a \mod b)$, gdzie $\mod$ oznacza operację wzięcia reszty z dzielenia. Możemy zatem napisać rekurencyjną funkcję, która wyliczy nam największy wspólny dzielnik.

euclid_gcd.cpp

```cpp
int algorytmEuklidesa(int p, int q) {
  if (q == 0) return p;
  return algorytmEuklidesa(q, p % q);
}
```

euclid_gcd.py

```python
def algorytmEuklidesa(p, q):
  if q == 0: return p
  return algorytmEuklidesa(q, p % q)
```

Na dłuższą metę, algorytm Euklidesa jest znacznie lepszą metodą obliczania największego wspólnego dzielnika – działa szybko nawet dla bardzo dużych liczb. Wcześniej przedstawiona, „naiwna" metoda jest w tym zadaniu możliwa dlatego, że dane p i q nie przekraczają 10^6.

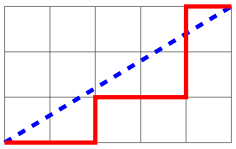
Mamy jeszcze możliwość skorzystania z gotowych implementacji w standardowych bibliotekach. Ta metoda bardzo przydaje się przy startach w konkursach, jako że nie musimy się martwić o potencjalne błędy przy samodzielnej implementacji. W języku C++ można znaleźć funkcję `gcd(p,q)` w bibliotece `numeric`: `https://en.cppreference.com/w/cpp/numeric/gcd`, natomiast w języku Python w bibliotece `math`: `https://docs.python.org/3.7/library/math.html#math.gcd`. Po dodaniu odpowiednich nagłówków, możemy korzystać z tej funkcji.

Jednak samo skrócenie ułamka nie jest końcem zadania (chociaż samo wypisanie ułamka w postaci skróconej dawało nam już 22 punkty). Nie zawsze napis reprezentuje ułamek, tak jak w teście przykładowym z treści zadania:

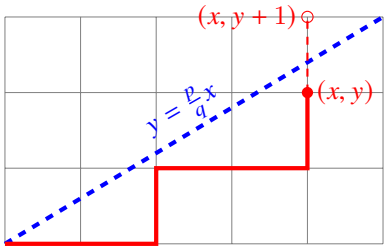
[1] `https://pl.wikipedia.org/wiki/Algorytm_Euklidesa`

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".

```
PPGPPGGP
```



Znając wymiary prostokąta, możemy sami stworzyć napis spełniający wymogi Bajtosi i porównać go z tym, który wczytaliśmy. Zasada jest prosta – jeśli ruch w górę nie przekroczy przekątnej, to ruszamy się w górę. Jak sprawdzić, czy ruch w górę jest możliwy? Jeśli lewy dolny wierzchołek prostokąta oznaczymy jako punkt $(0, 0)$ na osi współrzędnych, to przekątna należy do prostej o równaniu $y = \frac{p}{q}x$. Jeśli jesteśmy w punkcie (x, y) i zastanawiamy się, czy punkt $(x, y + 1)$ na odtwarzanej przez nas trasie będzie wciąż pod przekątną (lub ewentualnie dotykał przekątnej), to tak naprawdę chcemy wiedzieć, czy prosta przechodząca przez $(x, y+1)$ i $(0, 0)$ będzie miała mniejsze „nachylenie" od tej, która zawiera przekątną. A nachylenie tej prostej jest niczym innym jak ułamkiem $\frac{y+1}{x}$.



Zatem porównujemy, czy ułamek $\frac{y+1}{x}$ jest niewiększy od ułamka $\frac{p}{q}$. Jeśli tak, to punkt $(x, y+1)$ znajduje się pod przekątną, jeśli zaś ułamki są równe, punkt leży na przekątnej. Implementując rozwiązanie warto pamiętać, że typy zmiennoprzecinkowe, takie jak `float` i `long double` mają ograniczoną *precyzję*, czyli nie potrafią zapisywać ułamków dokładnie. Aby uniknąć problemów z niedokładnością, możemy wykonać małą sztuczkę i wymnożyć stronami mianowniki, to jest zamiast sprawdzać czy

$$\frac{p}{q} \leq \frac{y+1}{x},$$

będziemy sprawdzać czy

$$x \cdot p \leq q \cdot (y + 1).$$

Wówczas wszystkie operacje wykonujemy na liczbach całkowitych. Korzystając z tej sztuczki w języku C++ należy pamiętać o skorzystaniu z typu `long long`.

Po wykonaniu $p + q$ kroków powyższego algorytmu generującego trasę porównujemy otrzymany napis z napisem wczytanym. Jeśli są takie same, to znaczy, że trasa była poprawna i wypisujemy skrócony ułamek. W przeciwnym wypadku wypisujemy `NIE`.

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".

zyg.cpp

```cpp
#include "bits/stdc++.h"

using namespace std;

// Pomocnicza funkcja, która generuje opis prawidłowego zygzaka dla ułamka p/q.
string generujZygzak(int p, int q) {
  // (x,y) będzie aktualną pozycją końca zygzaka.
  int x = 0;
  int y = 0;
  string poprawny_zygzak = "";
  while (x < q or y < p) {
    // Sprawdzam czy mogę ruszyć w górę (preferowane) bez przekroczenia przekątnej.
    // Uwaga, przy sprawdzaniu iloczyny te mogą wykraczać poza typ int.
    if ((long long)(y+1)*q <= (long long)p*x) {
      y += 1;
      poprawny_zygzak += 'G';
    // Jeżeli nie, to ruszam się w prawo.
    } else {
      x += 1;
      poprawny_zygzak += 'P';
    }
  }
  return poprawny_zygzak;
}

int main() {
  // Wczytujemy opis zygzaka.
  string zygzak;
  cin >> zygzak;

  // Zliczamy liczbę znkaków G i P w opisie zygzaku.
  int p = 0;
  int q = 0;
  for (char znak : zygzak) {
    if (znak == 'G') p += 1;
    else if (znak == 'P') q += 1;
  }

  // Sprawdzenie, czy trasa wzorcowa pokrywa się z trasą na wejściu.
  if (zygzak != generujZygzak(p, q)) {
    cout << "NIE\n";
  } else {
    // Jeżeli tak, to chcemy znaleźć jego postać nieskracalną.
    // Aby to zrobić, liczymy największy wspólny dzielnik (gcd) p i q.
    int d = gcd(p, q);
    // Po czym dzielimy p i q przez ten dzielnik.
    int licznik = p / d;
    int mianownik = q / d;
    // Na końcu wypisujemy ułamek w skróconej formie.
    cout << licznik << "/" << mianownik << "\n";
  }
}
```


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".

zyg.py

```python
from math import gcd


# Pomocnicza funkcja, która generuje opis prawidłowego zygzaka dla ułamka p/q.
def generujZygzak(p, q):
  # (x,y) będzie aktualną pozycją końca zygzaka.
  x = 0
  y = 0
  poprawny_zygzak = ""
  # Dopóki nie doszedłem/doszłam do końca.
  while x < q or y < p:
    # Sprawdzam czy mogę ruszyć w górę (preferowane) bez przekroczenia przekątnej.
    if (y+1)*q <= p*x:
      y += 1
      poprawny_zygzak += 'G'
    # Jeżeli nie, to ruszam się w prawo.
    else:
      x += 1
      poprawny_zygzak += 'P'
  return poprawny_zygzak


def main():
  # Wczytujemy opis zygzaka.
  zygzak = input()

  # Zliczamy liczbę znkaków G i P w opisie zygzaku.
  p = 0
  q = 0
  for znak in zygzak:
    if znak == 'G': p += 1
    elif znak == 'P': q += 1

  # Sprawdzenie, czy trasa wzorcowa pokrywa się z trasą na wejściu.
  if zygzak != generujZygzak(p,q):
    print("NIE")
  else:
    # Jeżeli tak, to chcemy znaleźć jego postać nieskracalną.
    # Aby to zrobić, liczymy największy wspólny dzielnik (gcd) p i q.
    d = gcd(p, q)
    # Po czym dzielimy p i q przez ten dzielnik.
    licznik = p // d
    mianownik = q // d
    # Na końcu wypisujemy ułamek w skróconej formie.
    print("%d/%d" % (licznik, mianownik))


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".