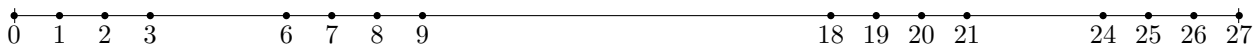
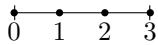


# Pinezki (rozwiązanie)

Autor zadania: **Karol Pokorski**  
Opracowanie: **Bartosz Kostka, Jacek Salata**  
Opis rozwiązania: **Bartosz Kostka**



W zadaniu tym kluczowe jest zanalizowanie jak wyglądają odcinki dla kolejnych  $N$ . Rozważmy kolejno  $N = 1, 2$  oraz  $3$  i narysujmy te odcinki:



Ponumerujemy te odcinki kolejno 1, 2, oraz 3. Można zauważyć, że odcinek  $N$  powstaje poprzez skopiowanie dwóch odcinków  $N - 1$ . Nazwiemy te kopie odpowiednio "lewą" i "prawą".

Odnotujmy jeszcze kilka rzeczy. Odcinek o numerze  $N$  ma długość  $3^N$  (co jest powiedziane w treści zadania). Możemy także podać liczbę pinezek który ten odcinek zawiera – jest to  $2^{N+1}$ . Wynika to z faktu, że odcinek dla  $N = 1$  ma  $4 = 2^2$  pinezek, a każdy kolejny składa się z dwóch kopii mniejszych odcinków. Możemy więc łatwo stwierdzić, kiedy należy odpowiedzieć NIE.

Aby rozwiązać teraz nasz problem, popatrzymy na niego rekurencyjnie. Mając dany numer pinezki  $K$  na danym odcinku o długości  $3^N$  (lub numerze  $N$ ), zastanowimy się czy znajduje się on w jego lewej lub prawej części. Możemy to łatwo stwierdzić, sprawdzając czy  $K \leq 2^N$ . Jeżeli tak, to pinezka znajduje się w lewej części odcinka (bo mamy łącznie  $2^{N+1}$  pinezek i połowa z nich znajduje się po lewej stronie). W przeciwnym wypadku pinezka znajduje się po prawej stronie. Zauważmy, że wtedy możemy ten sam problem rozważyć dla mniejszego odcinka (o numerze  $N - 1$ ).

Jedną kwestią, którą trzeba wspomnieć jest fakt, że gdy odkrywamy, że pinezka jest po prawej stronie, to w odcinku o numerze  $N - 1$  nie szukamy już pinezki o numerze  $K$ , ale pinezki o numerze  $K - 2^N$  – musimy odjąć wszystkie pinezki, które znajdowały się po lewej stronie. Analogicznie, wynik (pozycję ten pinezki) musimy dostosować o przesunięcie prawego odcinka względem lewego, czyli dodać  $2 \cdot 3^{N-1}$ .

Finalnie dojdziemy do odcinka o numerze  $N = 1$ , w którym łatwo już wyznaczyć pozycję każdej pinezki (patrz rysunek powyżej).

Całe zadanie sprowadza się zatem do napisania odpowiedniej funkcji rekurencyjnej, która zacznie od odcinka  $N$  i będzie znajdowała pozycję tej pinezki w coraz mniejszych odcinkach.

pin.cpp

```
1 #include <iostream>
2
3 using namespace std;
4
5 long long WyznaczPozycjePinezki(
6     long long K,
7     long long dlugosc_odcinka,
8     long long liczba_pinezek) {
9     // Jeżeli odcinek ma długość 1, to znaleźliśmy odpowiedź, zwróć K.
10    if (dlugosc_odcinka == 1) return K;
11    // W przeciwnym wypadku musimy zejść poziom niżej,
12    // zatem odpowiednio zmniejszamy liczbę pinezek i długość odcinka.
13    long long pinezki_w_czesci = liczba_pinezek / 2;
14    long long dlugosc_czesci_odcinka = dlugosc_odcinka / 3;
15    // Sprawdzamy w której części jesteśmy, jeżeli po lewej to po prostu
16    // wywołujemy się rekurencyjnie z mniejszymi wartościami.
17    if (K < pinezki_w_czesci)
```



```

18     return WyznaczPozycjePinezki(K, dlugosc_czesci_odcinka, pinezki_w_czesci);
19     // Natomiast jeżeli po prawej, to musimy dodać pinezki z lewej strony.
20     else
21         return 2*dlugosc_czesci_odcinka +
22             WyznaczPozycjePinezki(K-pinezki_w_czesci, dlugosc_czesci_odcinka,
23                                     pinezki_w_czesci);
24 }
25
26 int main() {
27     int N, Q;
28     cin >> N >> Q;
29     long long dlugosc_odcinka = 1;
30     for (int i=0; i<N; i++) dlugosc_odcinka *= 3;
31     long long liczba_pinezek = 2;
32     for (int i=0; i<N; i++) liczba_pinezek *= 2;
33     while(Q-->0) {
34         long long K;
35         cin >> K;
36         // Jeżeli K jest większe od liczby pinezek na odcinku, zwróć NIE.
37         if (K > liczba_pinezek) cout << "NIE\n";
38         else {
39             long long ret = WyznaczPozycjePinezki(K-1, dlugosc_odcinka,
40                                                     liczba_pinezek);
41             cout << ret << "\n";
42         }
43     }
44 }

```



```
1 def WyznaczPozycjePinezki(K, dlugosc_odcinka, liczba_pinezek):
2     # Jeżeli odcinek ma długość 1, to znaleźliśmy odpowiedź, zwróć K.
3     if dlugosc_odcinka == 1: return K
4     # W przeciwnym wypadku musimy zejść poziom niżej,
5     # zatem odpowiednio zmniejszamy liczbę pinezek i długość odcinka.
6     pinezki_w_czesci = liczba_pinezek // 2
7     dlugosc_czesci_odcinka = dlugosc_odcinka // 3
8     # Sprawdzamy w której części jesteśmy, jeżeli po lewej to po prostu
9     # wywołujemy się rekurencyjnie z mniejszymi wartościami.
10    if K < pinezki_w_czesci:
11        return WyznaczPozycjePinezki(K, dlugosc_czesci_odcinka, pinezki_w_czesci)
12    # Natomiast jeżeli po prawej, to musimy dodać pinezki z lewej strony.
13    else:
14        return 2*dlugosc_czesci_odcinka + \
15            WyznaczPozycjePinezki(K-pinezki_w_czesci, dlugosc_czesci_odcinka,
16                pinezki_w_czesci)
17
18    N = int(input())
19    Q = int(input())
20    dlugosc_odcinka = 3**N
21    liczba_pinezek = 2**(N+1)
22
23    for _ in range(Q):
24        K = int(input())
25        # Jeżeli K jest większe od liczby pinezek na odcinku, zwróć NIE.
26        if K > liczba_pinezek: print("NIE")
27        else: print(WyznaczPozycjePinezki(K-1, dlugosc_odcinka, liczba_pinezek))
```