



Zadanie „Dziwne sumy”

```
#typedef
#potęga_dwójki
```

W zadaniu *Dziwne sumy* problem polega na wyliczeniu wartości sumy kolejnych liczb naturalnych, przy czym pewne liczby – całkowite potęgi liczby 2 – mają znak zmieniony na minus.

Najprostsze podejście do rozwiązania, to obliczanie kolejnych wyrazów sumy (problemem jest tylko ich znak) – zwyczajnie w pętli. Ponieważ jednak z treści zadania wynika, że ilość składników sumy może sięgać 10^9 , zatem takie „naiwne” podejście nie może być optymalne. Być może uzyskamy jakiś ułamek punktów, ale na pewno nie dostaniemy 100%, czyli tak zwanego *maksa*.

Jak przyspieszyć działanie programu? Trzeba przyjrzeć się strukturze obliczanej sumy i zastosować zwięzły wzór na sumę szeregu, a dokładniej: dwóch szeregów – arytmetycznego i geometrycznego. Weźmy dla przykładu $n = 10$:

$$S_{10} = -1 - 2 + 3 - 4 + 5 + 6 + 7 - 8 + 9 + 10 = T_{10} - 2P_{10},$$

gdzie:

$$T_n = 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2},$$

jest sumą kolejnych liczb naturalnych (ciąg arytmetyczny), zaś:

$$P_n = 2^0 + 2^1 + 2^2 + \dots + 2^k = 2^{k+1} - 1,$$

jest sumą kolejnych potęg liczby 2 (ciąg geometryczny). Wykładnik k jest dobrany tak, by była to największa liczba, dla której zachodzi $2^k \leq n$. Dla $n = 10$ mamy $k = 3$.

Nasz program-rozwiązanie może wyglądać na przykład tak:

```
#include <bits/stdc++.h>
using namespace std;

typedef long long ll;

int main()
{
    int Z;
    cin >> Z;
```

```
while(Z--)  
{  
    ll n;  
    cin >> n;  
    ll p = 1;  
    while(p <= n)  
        p *= 2;  
    cout << n*(n+1)/2 - 2*(p-1) << endl;  
}  
return 0;  
}
```

Zmienna p oznacza 2^k . Jej wartość po zakończeniu wewnętrznej pętli `while` jest za duża, czyli wynosi tak naprawdę 2^{k+1} , ale właśnie takiej wartości potrzebujemy w końcowym wzorze.

Użycie typu `long long` (nazwaliśmy go dla wygody `ll`) jest konieczne, ponieważ n może osiągnąć wartość 10^9 , a więc iloczyn $n(n+1)$ może znacząco wykroczyć poza zakres typu `int`.

Zauważmy, że iloczyn $n(n+1)$ jest na pewno parzysty, zatem możemy bezpiecznie dzielić go przez 2. Działania w tym przypadku są wykonywane od lewej do prawej, ponieważ mają równy priorytet.

Uwagi o potęgowaniu

Z obliczaniem wartości potęg w języku C++ jest generalnie pewien problem, gdyż nie ma w nim dedykowanego operatora potęgowania (jak na przykład operator `**` w Pythonie).^{*} Znany z Excela operator `^` oznacza w C++ działanie bitowe zwane *alternatywą rozłączną* (`xor`).

Istnieje natomiast metoda szybkiego (nawet bardzo szybkiego!) obliczania potęg liczby 2 poprzez przesunięcie bitowe (w lewo) liczby 1. I tak wyrażenie:

```
1 << k
```

oznacza liczbę 1 przesuniętą w lewo o k pozycji, zatem jest to po prostu 2^k . Używając powyższej formuły warto dla bezpieczeństwa wziąć ją w nawiasy okrągłe, aby nie interferowała z identycznym operatorem przy strumieniu `cout`.

Oczywiście nie można przesadzać z wartością k , aby nie „wysunąć” jedynek poza granicę zajmowaną przez daną liczbę. W przypadku zwykłej jedynek (domyślnie typu `int`) musimy ograniczyć się do $k \leq 31$, jeśli natomiast użyjemy jedynek „długiej”, czyli `1LL` (typ `long long`), wtedy możemy użyć $k \leq 63$.

^{*}Biblioteczna funkcja `power()` zwraca rezultat typu zmiennoprzecinkowego, co nie jest wskazane przy obliczeniach stricte całkowitoliczbowych.