Wojciech Rytter, Andrzej Walat
Treść zadania, Opracowanie

Szymon Acedański
Program

OI, Etap III, dzień drugi, 29.03.2007

Waga czwórkowa

Smok Bajtazar zamierza zorganizować ucztę, na którą chce zaprosić wielu gości. Dla uświetnienia uczty, Bajtazar chce każdemu z gości podarować pewną ilość złota. Przy tym każdy z gości powinien dostać tyle samo złota, aby nikt nie czuł się pokrzywdzony.

Smok będzie odważał złoto dla kolejnych gości wagą szalkową. Ma on do dyspozycji odważniki, których wagi w gramach są potęgami czwórki. Odważników każdego rodzaju ma dowolnie wiele. Bajtazar będzie zawsze kładł złoto na lewej szalce wagi, a odważniki na prawej lub na obu szalkach. Bajtazar przy każdym ważeniu chce użyć najmniejszej możliwej liczby odważników. Ponadto, aby gościom się nie nudziło, chce on każdemu z nich odważyć złoto w inny sposób (tzn. używając innych odważników lub inaczej je rozdzielając pomiędzy szalki).

Ponieważ smok nie umie za dobrze liczyć, polecił ci napisanie programu, który obliczy, ilu maksymalnie gości może zaprosić, czyli wyznaczy maksymalną liczbę sposobów, na jakie można odważyć n gramów złota, używając minimalnej liczby odważników. Jeśli dobrze się sprawisz, smok obsypie Cię złotem!

Zadanie

Napisz program który:

- *wczyta ze standardowego wejścia, ile złota (w gramach) Bajtazar zamierza dać każdemu z gości,*
- *obliczy, na ile sposobów można odważyć taką ilość złota za pomocą minimalnej liczby odważników,*
- *wypisze resztę z dzielenia wyniku przez 10^9 na standardowe wyjście.*

Wejście

W pierwszym i jedynym wierszu standardowego wejścia zapisania jest jedna liczba całkowita n, $1 \leqslant n < 10^{1000}$. Jest to ilość złota (w gramach), jaką Bajtazar chce dać każdemu z gości.

Wyjście

Na standardowe wyjście należy wypisać jedną liczbę całkowitą — resztę z dzielenia przez 10^9 maksymalnej liczby gości, których Bajtazar może zaprosić (czyli maksymalnej liczby sposobów, na jakie można odważyć n gramów złota, używając minimalnej możliwej liczby odważników).

Przykład

Dla danych wejściowych:	*poprawnym wynikiem jest:*
`166`	`3`

Do odważenia 166 gramów potrzeba użyć 7 odważników. Ważenie można wykonać na następujące sposoby:

1. *na lewej szalce złoto, na prawej odważniki 64, 64, 16, 16, 4, 1, 1;*
2. *na lewej szalce złoto i odważniki 64, 16, 16, na prawej odważniki 256, 4, 1, 1;*
3. *na lewej szalce złoto i odważniki 64, 16, 4, 4, 1, 1, na prawej odważnik 256.*

i

Rozwiązanie

Algorytm korzysta istotnie z reprezentacji czwórkowej liczby n. Niech

$$n = n_0 \cdot 4^k + n_1 \cdot 4^{k-1} + \cdots + n_k \cdot 4^0;$$

wówczas

$$n = [n_0, n_1, n_2, \ldots, n_k]_4$$

oznacza reprezentację czwórkową liczby n. Tradycyjnie przyjmujemy, że $n_0, n_1, \ldots, n_k$ są *cyframi czwórkowymi* i pochodzą ze zbioru $\{0,1,2,3\}$. Możemy jednak tę reprezentację uogólnić na system z cyframi ujemnymi $\{-3,-2,-1,0,1,2,3\}$. Nawiązując do treści zadania, możemy powiedzieć, że cyfra ujemna $-s$ oznacza umieszczenie s odważników o odpowiedniej wadze na lewej szalce — tej samej, co ważony przedmiot.

Przykład 1
$166 = [2,2,1,2]_4$, ponieważ $166 = 2 \cdot 4^3 + 2 \cdot 4^2 + 1 \cdot 4^1 + 2 \cdot 4^0$.
Z drugiej strony
$166 = [1,-1,-2,1,2]_4$, ponieważ $166 = 1 \cdot 4^4 - 1 \cdot 4^3 - 2 \cdot 4^2 + 1 \cdot 4^1 + 2 \cdot 4^0$.

Traktując reprezentację czwórkową $[n_0, n_1, \ldots, n_k]$ jako sposób zważenia sztabki o wadze n, widzimy, że liczba użytych odważników to suma wartości bezwzględnych cyfr w reprezentacji. Minimalną wartość tej sumy oznaczymy przez $M(n)$. *Minimalna reprezentacja* to taka, w której suma cyfr wynosi $M(n)$.

W zadaniu chodzi o wyznaczenie liczby, oznaczanej tutaj przez $ILE(n)$, minimalnych reprezentacji liczby n.

Przykład 2 Dla rozważanego przykładu zachodzi:
$M(166) = 7$ (suma minimalna cyfr wynosi 7) oraz $ILE(166) = 3$ (mamy 3 minimalne reprezentacje):

$$166 = [2,2,1,2]_4 = [1,-1,-1,-2,-2]_4 = [1,-1,-2,1,2]_4.$$

Obserwacja 1 *Zauważmy, że w minimalnej reprezentacji nigdy nie wystąpią cyfry o wartości bezwzględnej większej niż* 2*. Jest oczywiste, że użycie czterech takich samych odważników jest nieoszczędne — zawsze można je zastąpić jednym większym odważnikiem. Natomiast trzy odważniki o wadze* 4^i *zawsze można zastąpić jednym odważnikiem o wadze* 4^{i+1} *oraz jednym odważnikiem o wadze* 4^i *postawionym na przeciwnej szalce.*

Przyjmijmy, że $[n_0,n_1,\ldots,n_k]_4$ jest *tradycyjną reprezentacją czwórkową* n, tzn. $n_i \in \{0,1,2,3\}$ dla $i = 0,1,\ldots,k$. Zdefiniujmy także $n^{<i>} = [n_0,n_1,\ldots,n_{i-1}]_4$ — jest to wartość liczby, której reprezentacją jest prefiks ciągu $[n_0,n_1,n_2,\ldots,n_k]$ długości i (złożony z i najbardziej znaczących cyfr). Dodatkowo wprowadźmy oznaczenia:

- $x_i = M(n^{<i>})$;
- $y_i = M(n^{<i>}+1)$;
- $X_i = ILE(n^{<i>})$;
- $Y_i = ILE(n^{<i>}+1)$.

Łatwo zauważyć, że poszukiwanym wynikiem jest liczba X_{k+1}. Skonstruujemy algorytm, który będzie wyznaczał kolejno wszystkie czwórki x_i,y_i,X_i,Y_i dla $i = 1,\ldots,k+1$.

Pierwszy sukces — wyznaczenie liczb x_i, y_i

Na początku policzymy tylko sumę cyfr w minimalnej reprezentacji, czyli minimalną liczbę odważników. Kluczem do sukcesu jest wykorzystanie obu wartości — x_i oraz y_i — do wyznaczenia kolejnej pary x_{i+1} oraz y_{i+1}. Zauważmy, że $n^{<i+1>} = n^{<i>} \cdot 4 + n_i$. W zależności od wartości n_i poszukiwane liczby x_{i+1} oraz y_{i+1}, wyznaczamy następująco:

Oblicz(x_{i+1},y_{i+1})

$$\begin{array}{lll} n_i = 0 & \Longrightarrow & (x_{i+1},y_{i+1}) = (x_i,x_i+1) \\ n_i = 1 & \Longrightarrow & (x_{i+1},y_{i+1}) = (x_i+1,\min(x_i+2,y_i+2)) \\ n_i = 2 & \Longrightarrow & (x_{i+1},y_{i+1}) = (\min(x_i+2,y_i+2),y_i+1) \\ n_i = 3 & \Longrightarrow & (x_{i+1},y_{i+1}) = (y_i+1,y_i) \end{array}$$

Uzasadnijmy, że nasza procedura jest poprawna. Rozważmy w tym celu wyznaczenie wartości x_{i+1} we wszystkich czterech przypadkach. Wartości y_{i+1} to wartości x_{i+1} z „sąsiedniego" przypadku, więc uzasadnienia poprawności obliczeń dla nich pominiemy.

Przypadek 1: $n_i = 0$**.** Jeśli do dotychczas rozpatrywanej wartości $n^{<i>}$ dopisujemy na końcu zero, to optymalny sposób jej zważenia polega na zastąpieniu wszystkich odważników czterokrotnie cięższymi. Gdyby bowiem istniał lepszy dobór $x_{i+1} < x_i$ odważników, wówczas także liczbę $n^{<i>}$ dałoby się odważyć przy pomocy x_{i+1} odważników. Wśród odważników tworzących minimalną reprezentację $n^{<i+1>}$ nie ma bowiem odważnika o wadze jeden i wszystkie odważniki tworzące tę reprezentację możemy zastąpić czterokrotnie lżejszymi.

Przypadek 2: $n_i = 1$**.** Aby zważyć sztabkę o wadze $n^{<i+1>}$, możemy zmienić wszystkie odważniki używane w minimalnej reprezentacji $n^{<i>}$ na czterokrotnie cięższe i dołożyć odważnik o wadze jednostkowej na prawej szalce, przeciwnej niż sztabka. Ważymy w ten sposób, używając $x_i + 1$ odważników. Dlaczego nie można lepiej? W optymalnym zestawie odważników dla sztabki o wadze $n^{<i+1>}$ musi być dokładnie jeden odważnik o wadze 1 (w optymalnym zestawie nie używamy bowiem więcej niż dwóch odważników tego samego typu i tylko zestawy z jednym odważnikiem jednostkowym pozwalają nam odważyć wartości nieparzyste). Wyrzucając go z zestawu, dostajemy zestaw o wadze $4 \cdot n^{<i>}$ złożony z $x_{i+1} - 1$ odważników. Możemy je zastąpić czterokrotnie lżejszymi, otrzymując rozwiązanie dla $n^{<i>}$, a zatem $x_{i+1} - 1 \leqslant x_i$, z optymalności x_i mamy również nierówność przeciwną $x_{i+1} - 1 \geqslant x_i$, czyli $x_{i+1} - 1 = x_i$.

Przypadek 3: $n_i = 2$**.** Jeśli do $n^{<i>}$ dopisujemy na końcu dwójkę, to nową sztabkę $n^{<i+1>}$ możemy zważyć:

- w zestawie x_i odważników służącym do zważenia $n^{<i>}$ zamieniając wszystkie odważniki na czterokrotnie cięższe i dokładając dwa o wadze jeden lub
- biorąc optymalny zestaw y_i odważników dla $n^{<i>} + 1$, zamieniając w nim wszystkie odważniki na czterokrotne cięższe i dokładając dwa o wadze jeden na tę samą szalkę, co sztabka.

W pierwszym przypadku użyjemy $x_i + 2$ odważniki, w drugim przypadku wykorzystamy $y_i + 2$ odważniki. Za x_{i+1} przyjmujemy lepszą z tych możliwości. Dlaczego inny sposób nie może być lepszy? Załóżmy, że uda nam się zważyć $n^{<i+1>}$ lepiej, używając $x_{i+1} < \min(x_i + 2, y_i + 2)$ odważników. Ponieważ wartość $n^{<i+1>}$ daje resztę 2 z dzielenia przez 4, więc w optymalnym zestawie odważników muszą być dwa odważniki o wadze 1 i muszą stać na tej samej szalce. Usuwając je, zmieniamy odważoną liczbę o -2 (jeśli stały na przeciwnej szalce niż sztabka) albo o 2 (jeśli stały na tej samej szalce, co sztabka). W uzyskanym zestawie wszystkie odważniki możemy zastąpić czterokrotnie lżejszymi. Taka operacja daje nam zestaw o wadze $n^{<i>}$ lub $n^{<i>} + 1$ złożony z $x_{i+1} - 2$ odważników, co pozwalałoby zważyć $n^{<i>}$ lub $n^{<i>} + 1$ lepiej niż optymalnie!

Przypadek 4: $n_i = 3$**.** Aby ustawić odważniki o łącznej wadze $n^{<i+1>}$, możemy skomponować zestaw dla wagi $n^{<i>} + 1$, następnie zamienić wszystkie odważniki na czterokrotnie cięższe i potem na szalce ze sztabką położyć odważnik jednostkowy. Dostaniemy w ten sposób zestaw złożony z $y_i + 1$ odważników. Czy jest on optymalny? Załóżmy, że nie jest i że można sztabkę o wadze $n^{<i+1>}$ zważyć, używając $x_{i+1} < y_i + 1$ odważników. Wśród nich musi być jeden odważnik jednostkowy (bo sztabka ma nieparzysty ciężar) — usuńmy go. Dostaliśmy w ten sposób zestaw bez odważników jednostkowych, gotowy do odważenia sztabki o wadze równej (w zależności od tego, z której szalki usunęliśmy odważnik):

- $n^{<i+1>} + 1 = n^{<i>} \cdot 4 + 4$, w którym możemy zamienić wszystkie odważniki na czterokrotnie lżejsze, uzyskując zestaw do odważenia sztabki $n^{<i>} + 1$ przy pomocy mniej niż y_i odważników, co jest niemożliwe;

- $n^{<i+1>} - 1 = n^{<i>} \cdot 4 + 2$, co jest niemożliwe, bo w zestawie nie ma już odważników jednostkowych, więc można odważyć nim tylko wielokrotności liczby 4.

Rozważenie wszystkich przypadków pozwoliło nam upewnić się, że mamy szybką i prostą metodę wyznaczania wartości x_i oraz y_i.

Ostateczny sukces — wyznaczenie liczb X_i, Y_i

Przypomnijmy, że X_i oznacza liczbę optymalnych sposobów zważenia sztabki o wadze $n^{<i>}$, a Y_i — liczbę optymalnych sposobów dla sztabki o wadze $n^{<i>} + 1$. Ponadto X_{k+1} jest poszukiwanym wynikiem. Do wyznaczenia X_i oraz Y_i wykorzystamy następujące zależności:

Oblicz(X_{i+1}, Y_{i+1})

$$\begin{aligned}
n_i = 0 &\implies (X_{i+1}, Y_{i+1}) = (X_i, X_i) \\
n_i = 1, x_i < y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i, X_i) \\
n_i = 1, x_i = y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i, X_i + Y_i) \\
n_i = 1, x_i > y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i, Y_i) \\
n_i = 2, x_i < y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i, Y_i) \\
n_i = 2, x_i = y_i &\implies (X_{i+1}, Y_{i+1}) = (X_i + Y_i, Y_i) \\
n_i = 2, x_i > y_i &\implies (X_{i+1}, Y_{i+1}) = (Y_i, Y_i) \\
n_i = 3 &\implies (X_{i+1}, Y_{i+1}) = (Y_i, Y_i)
\end{aligned}$$

Zastanówmy się, czy zaproponowana metoda jest poprawna. W uzasadnieniu — podobnie jak poprzednio — ograniczymy się do wartości X_i. Przypomnijmy, że w poprzednim rozdziale pokazaliśmy, jak rozwiązania optymalne dla $n^{<i>}$ przekształcać w rozwiązania optymalne dla $n^{<i+1>}$ i *vice versa*.

Przypadek 1: $n_i = 0$**.** Każde rozwiązanie optymalne dla $n^{<i>}$ możemy przekształcić w inne rozwiązanie optymalne dla $n^{<i+1>}$ — wystarczy zastąpić wszystkie odważniki czterokrotnie cięższymi. To dowodzi, że $X_i \leqslant X_{i+1}$. Także każde rozwiązanie optymalne dla $n^{<i+1>}$ możemy przekształcić w inne rozwiązanie optymalne dla $n^{<i>}$ — wystarczy zastąpić wszystkie odważniki czterokrotnie lżejszymi. Stąd $X_{i+1} \leqslant X_i$.

Przypadek 2: $n_i = 1$**.** Jak poprzednio, każde rozwiązanie optymalne dla $n^{<i>}$ przekształcamy w jedno rozwiązanie optymalne dla $n^{<i+1>}$ i *vice versa*. Ponownie więc mamy równość $X_i = X_{i+1}$.

Przypadek 3: $n_i = 2$**.** Tym razem może być różnie. Tworząc rozwiązanie optymalne dla $n^{<i+1>}$, wybieramy lepszą z dwóch możliwości. Jeśli $x_i < y_i$, to przekształcamy rozwiązanie optymalne dla $n^{<i>}$ i łatwo zauważyć, że jest to przekształcenie różnowartościowe w obie strony (a więc w tym przypadku $X_i = X_{i+1}$). Jeśli $x_i > y_i$, to przekształcamy rozwiązanie optymalne dla $n^{<i>} + 1$ (i analogicznie jak poprzednio mamy $Y_i = X_{i+1}$). Jeśli $x_i = y_i$, to każde rozwiązanie dla $n^{<i>}$ *oraz* każde rozwiązanie dla $n^{<i>} + 1$ można przekształcić w optymalne rozwiązanie dla $n^{<i+1>}$. Co ważne, łatwo zauważyć, że rozwiązania te są różne. To dowodzi, że w tym przypadku $X_{i+1} \geqslant X_i + Y_i$. Ponieważ przekształcenie odwrotne pozwala nam z każdego

rozwiązania optymalnego dla $n^{<i+1>}$ otrzymać rozwiązanie optymalne dla $n^{<i>}$ *albo* rozwiązanie optymalne dla $n^{<i>}+1$, więc także $X_{i+1} \leqslant X_i + Y_i$.

Przypadek 4: $n_i = 3$**.** W tym przypadku nie mamy wyboru, podobnie jak w dwóch pierwszych przypadkach. Rozwiązanie optymalne dla $n^{<i+1>}$ otrzymujemy z rozwiązania optymalnego dla $n^{<i>}+1$ i *vice versa*. Ponadto oba przekształcenia są różnowartościowe, więc $X_{i+1} = Y_i$.

Algorytm

Przeprowadzone rozważania pozwalają nam skonstruować bardzo prosty algorytm rozwiązania zadania:

```
1: x_0 := 0; y_0 := 1; X_0 := 1; Y_0 := 1;
2: for i := 0 to k-1 do
3: begin
4:    Oblicz(x_{i+1},y_{i+1});
5:    Oblicz(X_{i+1},Y_{i+1});
6: end
7: return X_{k+1};
```

Jeżeli pominąć koszt implementacji własnej arytmetyki dużych liczb, to powyższe rozwiązanie ma złożoność czasową $O(k) = O(\log n)$ — taka jest liczba wykonywanych operacji porównań i dodawań. Jeśli uwzględnimy, że wszystkie te operacje są wykonywane na liczbach długości $O(\log n)$, to złożoność algorytmu wzrasta do $O(\log^2 n)$. Dodatkowo, sama zamiana danych na czwórkowy układ pozycyjny zajmuje czas $O(\log^2 n)$, co jest składnikiem decydującym o złożoności czasowej programu.

Implementacja rozwiązania wzorcowego znajduje się w plikach `wag.c`, `wag1.pas` i `wag2.cpp`.

Rozwiązania nieoptymalne

Istnieje szeroka gama rozwiązań nieoptymalnych; pokrótce wymienimy najważniejsze z nich.

Zauważmy, że ustaliwszy łączną masę odważników znajdujących się na lewej szalce, można łatwo obliczyć, jaką masę odważników trzeba ustawić na szalce prawej. Z kolei daną masę odważników m można zawsze ustawić na dokładnie jeden sposób, zakładając konieczność użycia minimalnej liczby odważników — liczbę tę można wyznaczyć jako sumę cyfr w rozwinięciu czwórkowym m (dowód tego prostego faktu pozostawiamy Czytelnikowi). Istnieje kilka różnych rozwiązań, które na wszystkie sensowne sposoby (czyli takie, które nie wykorzystują odważników istotnie cięższych niż n) próbują wybierać masę m odważników na lewej szalce i sprawdzać, czy otrzymany układ wykorzystuje nie więcej odważników niż dotychczasowe. Przykładowe implementacje znajdują się w plikach `wags1.c` (złożoność czasowa $O(5^{\log_4 n})$) i `wags2.c` (złożoność $O(n)$).

Istnieją także szybsze rozwiązania nieoptymalne, które częściowo opierają się na obserwacjach z rozwiązania wzorcowego. W rozwiązaniu zaimplementowanym w pliku

`wags3.cpp` wykonujemy operację, w której próbujemy dopełniać aktualną wartość n do podzielnej przez 4 na dwa sposoby:

a) umieszczamy $(n \bmod 4)$ odważników jednostkowych na prawej szalce,

b) umieszczamy $(4-n \bmod 4)$ odważników jednostkowych na lewej szalce.

Dla każdego z przypadków następuje następnie wywołanie rekurencyjne, w którym konstruujemy rozwiązanie dla $\frac{n}{4}$ w przypadku a), natomiast w przypadku b) — dla $\frac{n}{4}+1$. A zatem w każdej fazie n zostaje podzielone mniej więcej przez 4, skąd można wywnioskować, że liczba faz będzie rzędu $O(\log_4 n)$. Ponieważ każda faza powoduje dwukrotne zwiększenie liczby wywołań rekurencyjnych, to złożoność czasowa całego algorytmu to (na mocy własności logarytmów):

$$O(2^{\log_4 n}) = O(2^{\frac{\log_2 n}{\log_2 4}}) = O(2^{0.5\log_2 n}) = O(\sqrt{n}).$$

Wreszcie w pliku `wags4.cpp` jest opisane rozwiązanie o pesymistycznie takiej samej złożoności czasowej, ale w praktyce zachowujące się znacznie lepiej. Zauważamy mianowicie, że w przypadku, kiedy $(n \bmod 4) \in \{0,1,3\}$, wiemy dokładnie, na którą szalkę należy stawiać odważniki jednostkowe, żeby wynikowa liczba odważników była najmniejsza możliwa: dla wartości 0 i 1 będzie to szalka prawa, natomiast dla 3 — lewa. Dowód tego faktu wynika wprost z dowodu poprawności rozwiązania wzorcowego, lecz można go także wywnioskować prostszymi metodami. To pokazuje, że w przypadku reszt $0,1,3$ wystarczy jedno wywołanie rekurencyjne powyższego algorytmu, a jedynie dla reszty 2 musimy rozpatrzyć dwie możliwości. Opisane spostrzeżenie pozwalało na przejście praktycznie wszystkich testów dla $n \leqslant 10^{19}$, czyli w zakresie liczb całkowitych 64-bitowych.

Testy

Rozwiązania zawodników były sprawdzane na 12 zestawach testów, z których połowa była testami poprawnościowymi, odróżniającymi od siebie gorsze rozwiązania nieoptymalne, a połowa — wydajnościowymi, które przechodziło jedynie rozwiązanie wzorcowe. W poniższej tabelce n oznacza ilość złota do odważenia (lub liczbę cyfr liczby n, w przypadku gdy n jest bardzo duże), a w — resztę z dzielenia przez 10^9 wynikowej liczby sposobów odważenia masy n na szalce. W przypadku testów poprawnościowych wartości parametru w są dokładne (a nie tylko modulo 10^9).

Nazwa	n	w	Opis
wag1a.in	6582	4	mały test poprawnościowy
wag1b.in	6566	5	mały test poprawnościowy
wag1c.in	1	1	mały test poprawnościowy
wag1d.in	4	1	mały test poprawnościowy
wag2a.in	42394	7	mały test poprawnościowy
wag2b.in	42395	5	mały test poprawnościowy

Nazwa	n	w	Opis
wag3a.in	42406	8	średni test poprawnościowy
wag3b.in	681382	11	średni test poprawnościowy
wag4a.in	9857418	10	duży test poprawnościowy
wag4b.in	9869466	9	duży test poprawnościowy
wag5a.in	17-cyfrowe	63	(bardzo) duży test poprawnościowy
wag5b.in	17-cyfrowe	48	(bardzo) duży test poprawnościowy
wag6a.in	1000-cyfrowe	927630336	test wydajnościowy
wag6b.in	1000-cyfrowe	22068224	test wydajnościowy
wag7a.in	1000-cyfrowe	750522880	test wydajnościowy
wag7b.in	1000-cyfrowe	263792640	test wydajnościowy
wag8a.in	1000-cyfrowe	667363840	test wydajnościowy
wag8b.in	1000-cyfrowe	252758528	test wydajnościowy
wag9a.in	1000-cyfrowe	324093440	test wydajnościowy
wag9b.in	1000-cyfrowe	78606848	test wydajnościowy
wag10a.in	998-cyfrowe	928779529	test wydajnościowy
wag10b.in	998-cyfrowe	771686912	test wydajnościowy
wag11a.in	9854718	1	duży test poprawnościowy
wag11b.in	9864966	2	duży test poprawnościowy
wag12a.in	1000-cyfrowe	20610560	test wydajnościowy
wag12b.in	1000-cyfrowe	523009536	test wydajnościowy

Testy były generowane w sposób losowy, z dodatkowym założeniem, żeby w większości zestawów wartości parametru n w obu testach były zbliżone do siebie. Dodatkowo, sposób generowania uwzględniał to, żeby wyniki dla testów (wartości parametru w) były duże.

Rozwiązania o złożoności $\Omega(n)$ przechodziły odpowiednio pierwsze 2 i 3 testy. Rozwiązanie o złożoności czasowej $O(\sqrt{n})$, ale bez optymalizacji, przechodziło wszystkie testy poprawnościowe poza bardzo dużymi, natomiast rozwiązanie z optymalizacjami — wszystkie testy poprawnościowe. 50% testów przechodziła także implementacja rozwiązania wzorcowego nieużywająca arytmetyki dużych liczb.

XVIII Międzynarodowa Olimpiada Informatyczna

Mérida, Meksyk 2006