Marcin Kubica
Treść zadania, Opracowanie

Marcin Michalski
Program

OI, Etap II, dzień 1, 9–02–2005

Sumy Fibonacciego

Liczby Fibonacciego to ciąg liczb całkowitych zdefiniowany następująco: $Fib_0 = 1$, $Fib_1 = 1$, $Fib_i = Fib_{i-2} + Fib_{i-1}$ *(dla* $i \geqslant 2$*). Oto kilka pierwszych wyrazów tego ciągu:* $(1, 1, 2, 3, 5, 8, \ldots)$.

Wielki informatyk Bajtazar konstruuje niezwykły komputer. Liczby w tym komputerze są reprezentowane **w układzie Fibonacciego**. *Liczby w takim układzie są reprezentowane jako ciągi zer i/lub jedynek (bitów), ciąg* $(b_1, b_2, \ldots, b_n)$ *reprezentuje liczbę* $b_1 \cdot Fib_1 + b_2 \cdot Fib_2 + \cdots + b_n \cdot Fib_n$. *(Zwróć uwagę, że nie korzystamy z* Fib_0*.) Taka reprezentacja liczb nie jest niestety jednoznaczna, tzn. tę samą liczbę można reprezentować na wiele sposobów. Na przykład, liczbę 42 można reprezentować jako:* $(0, 0, 0, 0, 1, 0, 0, 1)$, $(0, 0, 0, 0, 1, 1, 1, 0)$ *lub* $(1, 1, 0, 1, 0, 1, 1)$. *Dlatego też Bajtazar ograniczył się wyłącznie do reprezentacji spełniających następujące warunki:*

- *jeżeli* $n > 1$, *to* $b_n = 1$, *czyli reprezentacja liczby nie zawiera wiodących zer,*
- *jeżeli* $b_i = 1$, *to* $b_{i+1} = 0$ *(dla* $i = 1, \ldots, n-1$*), czyli reprezentacja liczby nie zawiera dwóch (lub więcej) jedynek obok siebie.*

Konstrukcja komputera okazała się trudniejsza, niż Bajtazar myślał. Ma on problemy z zaimplementowaniem dodawania. Pomóż mu!

Zadanie

Napisz program, który:

- *wczyta ze standardowego wejścia reprezentacje dwóch dodatnich liczb całkowitych,*
- *obliczy i wypisze reprezentację ich sumy na standardowe wyjście.*

Wejście

Na wejściu znajdują się reprezentacje Fibonacciego (spełniające podane powyżej warunki) dwóch dodatnich liczb całkowitych x *i* y *— jedna w pierwszym, a druga w drugim wierszu. Każda z tych reprezentacji jest zapisana jako ciąg nieujemnych liczb całkowitych, pooddzielanych pojedynczymi odstępami. Pierwsza liczba w wierszu to długość reprezentacji* n, $1 \leqslant n \leqslant 1\,000\,000$. *Po niej następuje* n *zer i/lub jedynek.*

Wyjście

W pierwszym i jedynym wierszu wyjścia Twój program powinien wypisać reprezentację Fibonacciego (spełniającą podane powyżej warunki) sumy $x + y$. *Tak jak to opisano dla wejścia, reprezentacja powinna mieć postać ciągu nieujemnych liczb całkowitych, pooddzielanych pojedynczymi odstępami. Pierwsza liczba w wierszu to długość reprezentacji* n, $1 \leqslant n \leqslant 1\,000\,000$. *Po niej następuje* n *zer i/lub jedynek.*

Przykład

Dla danych wejściowych:
```
4 0 1 0 1
5 0 1 0 0 1
```
poprawnym wynikiem jest:
```
6 1 0 1 0 0 1
```

Rozwiązanie

System Zeckendorfa

Opisany w zadaniu układ Fibonacciego jest znany jako system Zeckendorfa ([28], p. 6.6, [30], ćw. 1.2.8-34). System ten stanowi raczej ciekawostkę matematyczną i nie jest wykorzystywany w praktyce. Wynika to stąd, że implementacje operacji arytmetycznych są skomplikowane i mniej efektywne niż w przypadku klasycznego systemu dwójkowego.

Przekonajmy się na początek, że jest to jednoznaczny i poprawny system pozycyjny, tzn. każdą liczbę naturalną można w nim zapisać w dokładnie jeden sposób. Przez $b_k b_{k-1} \ldots b_{1\,Fib}$ oznaczmy $\sum_{i=1}^{k} b_i \cdot Fib_i$ i niech x będzie dowolną liczbą naturalną. Możemy ją przekształcić do systemu Zeckendorfa w następujący sposób:

1. Jeśli $x \leqslant 1$, to reprezentacja x składa się tylko z jednej cyfry $x_{Fib} = x$.
2. Jeśli $x > 1$, to niech Fib_k będzie największą liczbą Fibonacciego nieprzekraczającą x. Wówczas reprezentacja x ma k cyfr, a najstarsza cyfra to $b_k = \lfloor x/Fib_k \rfloor$. Pozostałe cyfry $b_{k-1}, \ldots, b_1$ to reprezentacja liczby $x - b_k \cdot Fib_k$ uzupełniona do długości $k-1$ wiodącymi zerami.

Sprawdźmy, czy przedstawiony algorytm jest poprawny, to znaczy, czy dla każdej liczby naturalnej daje wynik i czy wynik ten spełnia warunki zadania.

Lemat 1 *Dla dowolnej liczby naturalnej x powyższy algorytm generuje taki ciąg k cyfr* 0 *i* 1, *że:*

(a) $b_k b_{k-1} \ldots b_{1\,Fib} = x$,

(b) $b_k = 1$,

(c) jeżeli $b_i = 1$, *to* $b_{i+1} = 0$ *(dla* $i = 1, \ldots, k-1$*).*

Dowód Reprezentacja wygenerowana przez algorytm oczywiście spełnia warunek (a). Dowód dla pozostałych warunków przeprowadzimy przez indukcję względem x.

1. Dla $x \leqslant 2$ można łatwo sprawdzić, że lemat jest prawdziwy.
2. Załóżmy, że $x > 2$ i lemat jest spełniony dla wartości mniejszych niż x.

   Wiemy, że Fib_k jest największą liczbą Fibonacciego nie przekraczającą x, czyli

$$Fib_k \leqslant x < Fib_{k+1} = Fib_{k-1} + Fib_k$$

i dalej

$$x - Fib_k < Fib_{k-1}. \tag{1}$$

Stąd widzimy, iż $b_k \leqslant 1$ (bo $Fib_{k-1} < Fib_k$), a z algorytmu dodatkowo wynika, iż $b_k \geqslant 1$. Mamy więc warunek (b).

Z nierówności (1) mamy także, że $b_{k-1} = 0$, a reprezentacja $x - Fib_k$ składa się z co najwyżej $k-2$ cyfr. Z założenia indukcyjnego możemy teraz wywnioskować, że ciąg cyfr $b_k \ldots b_1$ spełnia warunki (b) – (c).

Dodatkowo zauważamy, że wszystkie wygenerowane cyfry to zera i jedynki.

Na mocy zasady indukcji kończymy dowód. ■

Pokażemy teraz, że reprezentacja w systemie Zeckendorfa jest jednoznaczna.

Lemat 2 (Zeckendorf) *Dla każdej dodatniej liczby całkowitej x istnieje dokładnie jeden ciąg $b_k \ldots b_1$ cyfr 0 i/lub 1 spełniający warunki (b) – (c) oraz taki, że $b_k \ldots b_{1\,Fib} = x$.*

Dowód Dowód będzie przebiegał nie wprost. Załóżmy, że istnieje dodatnia liczba całkowita, która ma dwie różne reprezentacje spełniające warunki (b) – (c) i niech x będzie najmniejszą z takich liczb. Niech $b_k \ldots b_1$ i $b'_l \ldots b'_1$ będą dwiema różnymi reprezentacjami x spełniającymi warunki (b) – (c).

Zauważmy, że k musi być różne od l, w przeciwnym przypadku usuwając b_k i b_l (i ewentualnie wiodące zera) uzyskalibyśmy dwie różne reprezentacje liczby $x - Fib_k$ mniejszej niż x. Przyjmijmy, że $k > l$.

Spróbujmy oszacować wartość x od góry i od dołu wiedząc, że ma reprezentację k-cyfrową i l-cyfrową. Skoro $b_k = 1$, to $x \geqslant Fib_k$. Największa liczba mająca reprezentację długości l spełniającą warunki (b) – (c), to $101010\ldots10_{Fib}$ (dla l parzystego) lub $101010\ldots01_{Fib}$ (dla l nieparzystego). Stąd, dla l parzystego mamy

$$\begin{aligned}
x &< x+1 \leqslant \\
&\leqslant 101010\ldots1010_{Fib} + 1 = \\
&= 101010\ldots1011_{Fib} = \\
&= 101010\ldots1100_{Fib} = \\
&\quad\vdots \\
&= 101100\ldots0000_{Fib} = \\
&= 110000\ldots0000_{Fib} = \\
&= 1000000\ldots0000_{Fib} = \\
&= Fib_{l+1} \leqslant Fib_k \leqslant x
\end{aligned}$$

i podobnie dla l nieparzystego mamy

$$\begin{aligned}
x &< x+1 \leqslant \\
&\leqslant 101010\ldots101_{Fib} + 1 = \\
&= 101010\ldots110_{Fib} =
\end{aligned}$$

$$\begin{aligned} &\vdots \\ &= 101100\ldots0000_{Fib} = \\ &= 110000\ldots0000_{Fib} = \\ &= 1000000\ldots0000_{Fib} = \\ &= Fib_{l+1} \leqslant Fib_k \leqslant x \end{aligned}$$

W ten sposób doszliśmy do sprzeczności: $x < x$. Tak więc założenie, że x może mieć dwie różne reprezentacje spełniające warunki (b) – (c) jest fałszywe, co kończy dowód. ■

Prosty algorytm dodawania

Najprostszy sposób dodawania liczb w systemie Zeckendorfa polega na dodawaniu do pierwszej liczby kolejno liczb Fibonacciego odpowiadających jedynkom w reprezentacji drugiej liczby. Aby do liczby x dodać liczbę Fib_i, zwiększamy i-tą cyfrę reprezentacji x o 1. W wyniku możemy dostać:

- reprezentację spełniającą warunki (b) – (c),
- reprezentację zawierającą dwie lub trzy sąsiadujące ze sobą jedynki,
- reprezentację zawierającą dwójkę, która może sąsiadować tylko z zerami.

Eliminacja dwójek (1) Jeżeli wynik zawiera dwójkę, to możemy ją wyeliminować stosując następujące przekształcenia począwszy od najbardziej znaczących cyfr:

$$x0200y_{Fib} = x1001y_{Fib} \quad (2)$$

$$x0201y_{Fib} = x1002y_{Fib} \quad (3)$$

$$x02_{Fib} = x10_{Fib} \quad (4)$$

Zauważmy, że w przekształceniu 3 dwójka nie znika, lecz przesuwa się w prawo. Jednak przeglądając ciąg w podanym kierunku wyeliminujemy ją (lub przesuniemy dalej w prawo) w kolejnym kroku. Warto przy tym zauważyć, że nowopowstała dwójka jest również otoczona zerami lub jest ostatnią cyfrą sumy, gdyż jedynka przed operacją również była otoczona zerami. Cały proces może wymagać czasu liniowego. Na koniec otrzymujemy reprezentację, która nie zawiera dwójek, ale może zawierać sąsiadujące ze sobą jedynki.

Eliminacja sąsiednich jedynek (2) Sąsiadujące ze sobą jedynki możemy wyeliminować przekształcając reprezentację począwszy od najmniej znaczących cyfr i stosując następujące tożsamości:

$$x0\underbrace{11\ldots11}_{2l \text{ cyfr}}y_{Fib} = x\underbrace{10\ldots10}_{2l \text{ cyfr}}0y_{Fib} \quad (5)$$

$$x0\underbrace{11\ldots11}_{2l+1 \text{ cyfr}}y_{Fib} = x\underbrace{10\ldots10}_{2l \text{ cyfr}}01y_{Fib} \quad (6)$$

Całe przekształcenie może wymagać czasu liniowego.

Łączny koszt dodania liczby Fib_i do liczby x jest więc liniowy, stąd opisany algorytm dodawania dwóch liczb działa w czasie kwadratowym.

Efektywny algorytm dodawania

Efektywny algorytm dodawania w systemie Zeckendorfa jest bardziej skomplikowany. Najpierw dodajemy odpowiadające sobie cyfry sumowanych liczb. W rezultacie uzyskujemy reprezentację, w której:

- mogą występować cyfry 0, 1 i 2,
- jedynki mogą sąsiadować ze sobą,
- dwójki mogą sąsiadować tylko z zerami.

Ciąg ten musimy tak przekształcić, by spełniał warunki (b) – (c).

Eliminacja sąsiednich jedynek (3) W poprzednim punkcie opisaliśmy procedurę eliminacji sąsiadujących ze sobą jedynek (Eliminacja 2). Możemy ją rozszerzyć wykorzystując również następujące tożsamości:

$$x0210y_{Fib} = x1100y_{Fib} \quad (7)$$
$$x0211y_{Fib} = x1101y_{Fib} \quad (8)$$

Operacja 7 jest konieczna, gdyż w trakcie operacji 5 i 6 skrajnie lewa nowopowstała jedynka może być koło dwójki, tworząc układ 210. Natomiast w operacji 7 powstają dwie jedynki obok siebie, które mogą utworzyć układ 211. Taki układ jest eliminowany przez operację 8, która może znów wygenerować układ 211, jednak przesunięty dalej o dwa miejsca.

W ten sposób wyeliminujemy sąsiadujące ze sobą jedynki również z reprezentacji zawierającej dwójki sąsiadujące początkowo tylko z zerami. Po eliminacji dostajemy ciąg, który nie zawiera sąsiadujących jedynek, ale nadal może zawierać dwójki — sąsiadujące tylko z zerami. Procedura wymaga jednorazowego przejrzenia ciągu począwszy od najmniej znaczących cyfr i działa w czasie liniowym.

W następnym kroku przekształcimy reprezentację tak, by nie zawierała dwójek. Efektem ubocznym przekształcenia może być ponowne pojawienie się w reprezentacji sąsiadujących ze sobą jedynek. Możemy jednak sobie na to pozwolić, bo potrafimy je wyeliminować w czasie liniowym (Eliminacja 2).

Eliminacja dwójek (4) Przekształcenie reprezentacji zaczynamy od wyeliminowania sekwencji cyfr 201 i 200. Zauważmy, iż skoro wyeliminowaliśmy sekwencje jedynek, sekwencją 201 jest w rzeczywistości sekwencją 2010 lub występuje na końcu liczby.

Przeglądamy ciąg począwszy od najbardziej znaczących cyfr i stosujemy następujące dwie tożsamości:

$$x201y_{Fib} = x112y_{Fib} \quad (9)$$
$$x200y_{Fib} = x111y_{Fib} \quad (10)$$

Tę fazę algorytmu możemy wykonać w czasie liniowym, a po jej zakończeniu dostajemy ciąg, w którym:

- mogą występować sąsiadujące jedynki,

- po lewej stronie każdej dwójki musi być zero lub ciąg przynajmniej dwóch jedynek poprzedzony zerem;
- trzy najmniej znaczące cyfry reprezentacji są różne od 201.

Kolejny krok polega na takim przekształceniu reprezentacji, żeby dwie najmniej znaczące cyfry były różne od dwójki. W tym celu wystarczy do ciągów niespełniających tego warunku zastosować najpierw tożsamość:

$$x20_{Fib} = x12_{Fib} \tag{11}$$

a następnie:

$$x0\underbrace{11\ldots11}_{2l \text{ cyfr}}2_{Fib} = x\underbrace{10\ldots10}_{2l+2 \text{ cyfr}}{}_{Fib} \tag{12}$$

$$x0\underbrace{11\ldots11}_{2l+1 \text{ cyfr}}2_{Fib} = x\underbrace{10\ldots10}_{2l+2 \text{ cyfr}}1_{Fib} \tag{13}$$

Krok ten wymaga czasu najwyżej liniowego.

Zastanówmy się jakie może być sąsiedztwo dwójek w powstałej reprezentacji. Na lewo od dwójki wciąż może być zero albo sekwencja $01\ldots1$. Zaś na prawo od dwójki:

- może powstać jedynka (w wyniku operacji 12 i 13), jednak wówczas zawsze otrzymamy sekwencję 210,
- nie mogą znajdować się dwa zera, gdyż zostały wyeliminowane w wyniku operacji 10 i nie mogły powstać w wyniku późniejszych operacji,
- może znajdować się sekwencja 011, ale nie 010 — w wyniku operacji 9 i 10 wyeliminowano wszystkie wcześniej istniejące sekwencje 201, lecz mogły powstać sekwencje 2011 (np. z sekwencji 20200 lub 20201); późniejsze operacje nie mogły wprowadzić sekwencji 2010, gdyż do tego potrzebna by była sekwencja 200, a wszystkie takie sekwencje zostały wyeliminowane przez operacje 10.

Wobec tego w wyniku dostajemy reprezentację, w której:

- na lewo od każdej dwójki może występować zero albo $01\ldots1$,
- na prawo od każdej dwójki może wystąpić sekwencja 02 albo 011, albo 10.

Z takiego ciągu możemy ostatecznie wyeliminować dwójki przeglądając cyfry od najmniej znaczącej i stosując następujące tożsamości:

$$x2011y_{Fib} = x2100y_{Fib} \tag{14}$$

$$x021y_{Fib} = x110y_{Fib} \tag{15}$$

$$x121y_{Fib} = x210y_{Fib} \tag{16}$$

W wyniku otrzymujemy reprezentację, w której nie występują dwójki, lecz mogą występować sąsiadujące jedynki. Stosujemy do niej Eliminację 2 i otrzymujemy postać spełniającą warunki (b) – (c).

Sumaryczny czas wykonania wszystkich przekształceń jest liniowo zależny od długości reprezentacji. Implementacja opisanego algorytmu znajduje się w pliku `suma.pas` na płycie CD załączonej do książki.

Testy

Rozwiązania zawodników były sprawdzane na 10 testach. Cztery spośród testów, to testy poprawnościowe, w których sumowane składniki mają do 101 cyfr. Pozostałe sześć testów, oprócz poprawności sprawdzało również wydajność rozwiązań. Zostały one tak dobrane, żeby nieefektywne algorytmy dodawania wymagały przynajmniej czasu kwadratowo zależnego od długości reprezentacji. Rozmiar danych w tych testach zwiększa się stopniowo, tak aby za ich pomocą dało się odróżnić rozwiązania o różnych złożonościach czasowych. Wszystkie testy można znaleźć na płycie CD.
W poniższej tabeli l_1 oraz l_2 oznaczają długość liczb, które należy dodać.

Nazwa	**l_1**	**l_2**	**Opis**
sum1.in	1	5	mały test
sum2.in	1	1	test sprawdzający poprawność stanów „końcowych”
sum3.in	50	51	kolejny prosty test sprawdzający poprawność
sum4.in	101	101	test, w którym po zsumowaniu dostajemy naprzemienny ciąg 2 i 0
sum5.in	1 001	1 001	duży test losowy
sum6.in	3 750	3 750	„złośliwy” test losowy
sum7.in	50 005	50 005	test „złośliwy”— na zmianę występują grupy 1 i 2
sum8.in	250 005	250 005	test analogiczny do poprzedniego, ale większy
sum9.in	500 005	500 005	test, w którym po zsumowaniu najpierw występują grupy 1, a później 2
sum10.in	975 005	975 005	największy test