Szybkie potęgowanie z użyciem schematu Hornera

W 1819 roku W.G Horner podał sposób obliczenia wartości wielomianu, nazywany dzisiaj jego nazwiskiem, ale 150 lat wcześniej Isaac Newton stosował podobny sposób obliczenia wartości wielomianu, który występował w jego rachunkach fizycznych. Te fakty z historii jeszcze raz dowodzą, że wiele algorytmów stosowanych dzisiaj w obliczeniach komputerowych pochodzi z czasów, gdy jeszcze nie było komputerów. A. Borodin udowodnił w 1971 roku, że schemat Hornera jest najszybszym sposobem obliczenia wartości wielomianu. Jest to więc przykład algorytmu optymalnego.

Szybkie potęgowanie

Binarna reprezentacja liczb naturalnych jest podstawą dla szybkich metod obliczenia wartości potęgi x^n, gdzie *n* jest liczbą naturalną, zaś x dowolną liczbą rzeczywistą (wartość podstawy *x* nie ma znaczenia dla naszych rozważań). Metody te znajdują zastosowanie w algorytmach kryptograficznych, w których są obliczane wartości potęg o bardzo dużych wykładnikach.

Posłużymy się najpierw przykładem. Przypuśćmy, że chcemy obliczyć wartość potęgi x^{22}.
Proste wymnożenie podstawy przez siebie $x^{22} = x*x*...*x$ to *21* mnożeń. Ale skorzystajmy z binarnej reprezentacji wykładnika potęgi. Można go przedstawić jako sumę potęg liczby *2*:

$22 = 2 + 4 + 16 = 2^1 + 2^2 + 2^4$

wtedy nasza potęga przyjmuje postać

$x^{22} = x^{2+4+16} = x^2x^4x^{16}$

i można ją obliczyć wielokrotnie, podnosząc do kwadratu podstawę x i mnożąc przez siebie odpowiednie czynniki. W naszym przypadku obliczamy: x^2, $x^4 = (x^2)^2$, $x^8 = (x^4)^2$, $x^{16} = (x^8)^2$ i mnożymy przez siebie $x^2x^4x^{16}$. W sumie wykonujemy *6* mnożeń.

Korzystając z przedstawienia wykładnika w reprezentacji binarnej $22 = (10110)_2$ w postaci schematu Hornera, wykładnik można zapisać $22 = 1*2^4 + 0*2^3 + 1*2^2 + 1*2^1 + 0*2^0 = (((2 +0)2 +1)2 +1)2 +0$, a stąd otrzymujemy:

$x^{(((2+0)2+1)2+1)2+0} = x^{(((2+0)2+1)2+1)2} = (x^{((2+0)2+1)2+1})^2 = (x^{((2+0)2+1)^2}x)^2 =$
$= ((x^{((2+0)2+1)^2}x)^2 = ((x^{(2+0)^2}x)^2x)^2 = (((x^{(2+0)^2}x)^2x)^2 = (((x^2)^2x)^2x)^2.$

Są dwie metody potęgowania korzystające z binarnego rozwinięcia wykładnika, ale różnią się tym, że w pierwszym przypadku, to od najbardziej znaczącego bitu (czyli metodą od lewej do prawej), a w drugim – rozwinięcie jest przeglądane od najmniej znaczącego (mówimy o metodzie od prawej do lewej).

Binarny algorytm potęgowania „od lewej do prawej"
Na tej podstawie powyższych przekształceń można wyprowadzić następującą zależność między postacią wykładnika w układzie binarnym a działaniami wykonywanymi przy obliczaniu potęgi:

1) najbardziej znaczący bit w rozwinięciu wykładnika (który jest zawsze równy 1) odpowiada rozpoczęciu obliczeń od przyjęcia liczby za początkową wartość potęgi;
2) każda następna pozycja w rozwinięciu odpowiada podniesieniu częściowego wyniku do kwadratu i ewentualnie pomnożeniu przez x, jeśli bit rozwinięcia na tej pozycji jest równy 1.

Dane: Liczba naturalna n w postaci binarnej $(n_n n_{n-1} \ldots n_1 n_0)_2$ i dowolna liczba x.
Wynik: Wartość potęgi x^n
Krok 1. {Ustalenie początkowej wartości potęgi} w=x
Krok 2. Dla i=n-1,...,1,0 wykonaj: jeśli n_i==1, to w=w*w*x, w innym razie w=w*w

Powyższy algorytm, wykorzystujące schemat Hornera pozostawiamy do samodzielnego wykonania.

Obliczmy, ile mnożeń wykonujemy w tym algorytmie. Wszystkie te działania są wykonywane w kroku 2, dla kolejnych bitów w binarnej reprezentacji wykładnika, z wyjątkiem pierwszego, najbardziej znaczącego bitu. Zatem liczba mnożeń w całym algorytmie jest równa liczbie wszystkich bitów w binarnej reprezentacji wykładnika, minus jeden, plus liczba jedynek w tej reprezentacji.

Rozwiązanie rekurencyjne:

Potęgowanie (Globalny zasięg)

```
1  #include<iostream>
2  using namespace std;
3  long long pot_szybkie_rek(long long x, unsigned int n)
4  {
5      long long w;
6      if (n == 0)
7          return 1;
8      if (n % 2 == 1) //gdy n jest nieparzyste
9      {   //żeby dwa razy nie wchodzić w tą samą rekurencję
10         w = pot_szybkie_rek(x, (n - 1) / 2);
11         return w * w * x;
12     }
13
14     else
15     {
16         w = pot_szybkie_rek(x, n / 2);
17         return w * w;
18     }
19 }
20 int main()
21 {
22     unsigned int n;
23     long long x;
24     cout << "Podaj podstawe: ";
25     cin >> x;
26     cout << "Podaj wykladnik: ";
27     cin >> n;
28     cout << x << " do potegi " << n << " wynosi " << pot_szybkie_rek(x, n);
29     cin.get(); //czekamy na wciśnięcie klawisza
30     return 0;
31 }
```

Algorytm potęgowania "od-lewej-do-prawej" ma pewną wadę. Rozwinięcie binarne liczby jest naturalnie tworzone od najmniej znaczącego bitu, czyli "od-prawej-do-lewej. Można odeprzeć ten argument stwierdzeniem, że liczby naturalne są pamiętane w komputerze w postaci binarnej, więc binarna reprezentacja wykładnika jest dana. Jednak, aby z niej skorzystać musimy i tak dotrzeć do poszczególnych jej bitów. Najłatwiej to zrobić, stosując algorytm, który tworzy taką właśnie reprezentację bit po bicie.

Poniższy algorytm obliczania wartości potęgi wykonuje potęgowanie rozkładając wykładnik na postać binarną (postać ta jednak nie jest zachowywana w algorytmie).

Binarny algorytm potęgowania „od prawej do lewej”

Dane: Liczba naturalna n i dowolna liczba x.

Wynik: Wartość potęgi $w = x^n$.

Krok 1. {Ustalenie początkowych wartości potęgi} w = 1;

Krok 2. Jeśli n == 0, to zakończ algorytm – wynikiem jest bieżąca wartość w.

Krok 3. Jeśli n % 2 == 1 (czyli n jest liczbą nieparzystą), to w = w * x;

Krok 4. x = x * x;

Krok 5. n = n / 2; wróć do kroku 2.

Potęgowanie | (Globalny zasięg)

```
1  #include<iostream>
2  using namespace std;
3  long long pot_szybkie_ite(long long x, unsigned int n)
4  {
5      long long w = 1;
6      while (n > 0)
7      {
8          if (n % 2 == 1) //jeśli bit jest równy 1
9              w = w * x;
10         x = x * x;
11         n = n / 2; //skrócenie o jeden bit
12     }
13     return w;
14 }
15 int main()
16 {
17     unsigned int n;
18     long long x;
19     cout << "Podaj podstawe: ";
20     cin >> x;
21     cout << "Podaj wykladnik: ";
22     cin >> n;
23     cout << x << " do potegi " << n << " wynosi " << pot_szybkie_ite(x, n);
24     cin.get(); //czekamy na wciśnięcie klawisza
25     return 0;
26 }
```