Największy wspólny dzielnik
Algorytm Euklidesa (także rozszerzony)
Chińskie twierdzenie o resztach

Paweł Tarasiuk

Wybrane zagadnienia algorytmiki i programowania I

27 października 2010

Największy wspólny dzielnik - definicja

Największy wspólny dzielnik zbioru $S \subset R$ to taka liczba $d \in R$, że:

1. $\underset{s \in S}{\forall} \; d|s$
2. $\underset{g \in R}{\forall} \; \left(\left(\underset{s \in S}{\forall} \; g|s \right) \Longrightarrow g|d \right)$

Skoncentrujemy się na szukaniu największego wspólnego dzielnika podzbiorów zbioru liczb naturalnych. Zważywszy, że dla liczb naturalnych zachodzi "rozłączność" największego wspólnego dzielnika, czyli dla niepustych zbiorów $S_1, S_2 \subset \mathbb{N}$:

$$\mathrm{NWD}(S_1 \cup S_2) = \mathrm{NWD}(\mathrm{NWD}(S_1), \mathrm{NWD}(S_2))$$

Wystarczy zatem opracować rozwiązania problemu wyznaczania największego wspólnego dzielnika pary liczb naturalnych.

Największy wspólny dzielnik a Najmniejsza wspólna wielokrotność

Pojęciem pokrewnym z NWD, z którego skorzystamy przy tej prezentacji jest NWW - najmniejsza wspólna wielokrotność. Liczeniu jej nie poświęca się jednak tyle uwagi co NWD - gdyż najprostszy sposób wyznaczania NWW polega na skorzystaniu z zależności:

$$\mathrm{NWW}(a,\, b) \cdot \mathrm{NWD}(a,\, b) = a \cdot b$$

Czyli: największa wspólna wielokrotność pomnożona przez największy wspólny dzielnik dowolnego podzbioru liczb naturalnych daje w wyniku iloczyn wszystkich jego elementów. Dowód tej zależności zostanie pominięty, ale wystarczy przeanalizować, jak wygląda w zależności od argumentów rozbicie NWD oraz NWW na dzielniki pierwsze, aby potwierdzić prawdziwość tej zależności. NWW, podobnie jak NWD, jest łączne, zatem możliwość obliczania go dla zbioru dwuelementowego jest wystarczająca.

Największy wspólny dzielnik - poprzez faktoryzację

Dla każdego $n \in \mathbb{N}$ da się w sposób jednoznaczny zapisać:

$$n = p_1^{k_1} \cdot p_2^{k_2} \cdot p_3^{k_3} \cdot p_4^{k_4} \cdot p_5^{k_5} \cdot \ldots$$

Gdzie p_1, p_2, p_3, ... to kolejne liczby pierwsze, k_1, k_2, k_3, ... są pewnymi liczbami całkowitymi nieujemnymi, oraz $\exists_{Q \in \mathbb{N}} \forall_{q \in \mathbb{N},\, q > Q}\ k_q = 0$. Jeżeli zapiszemy drugą liczbę naturalną w ten sam sposób:

$$l = p_1^{m_1} \cdot p_2^{m_2} \cdot p_3^{m_3} \cdot p_4^{m_4} \cdot p_5^{m_5} \cdot \ldots$$

to:

$$\mathrm{NWD}(n, l) = p_1^{\min(k_1, m_1)} \cdot p_2^{\min(k_2, m_2)} \cdot p_3^{\min(k_3, m_3)} \cdot p_4^{\min(k_4, m_4)} \cdot p_4^{\min(k_5, m_5)} \cdot \ldots$$

Algorytm Euklidesa - podstawowe ujęcie

Praktyczną metodą obliczania NWD jest algorytm Euklidesa, czyli schemat postępowania oparty na twierdzeniu:

$$\forall_{a,\, b \in \mathbb{N},\, a>b} \; \mathrm{NWD}(a,\, b) = \mathrm{NWD}(a - b,\, b)$$

Powtarzanie powyższej operacji prowadzi do ciągłego malenia wartości argumentów naturalnych dla których chcemy poznać wartość NWD i kończy się, gdy nie da się wskazać większego spośród pary argumentów - wówczas wystarczy dokonać oczywistego spostrzeżenia, że $\mathrm{NWD}(a,\, a) = a$. Zatem niniejsze twierdzenie pozwala zawsze w sposób jednoznaczny poznać wartość NWD pary liczb. Pesymistyczna złożoność tego ujęcia algorytmu liczenia $\mathrm{NWD}(n,\, m)$ przystaje do $O(\max(n, m))$.

Implementacja algorytmu Euklidesa - popularna heurystyka

Niech a, b, $k \in \mathbb{N}$ oraz $a > k \cdot b$. Wówczas posługując się klasycznym ujęciem algorytmu Euklidesa (z poprzedniego slajdu) do obliczenia NWD(a, b), wykonywalibyśmy obliczenia:

$$\text{NWD}(a, b) = \text{NWD}(a-b, b) = \text{NWD}(a-2b, b) = \ldots = \text{NWD}(a-k\cdot b, b)$$

Proces ten można uprościć zastępując wielokrotne odejmowanie poprzez operację reszty z dzielenia (zapożyczając ze składni języka C, oznaczać ją będziemy poprzez %). Korzystać zatem będziemy z nieco zmienionej, także poprawnej wersji twierdzenia:

$$\forall_{a,\, b \in \mathbb{N},\, a>b} \; NWD(a, b) = NWD(a\%b, b)$$

Złożoność algorytmu maleje wówczas do $O(\log(\max(n, m)))$.

Implementacja algorytmu Euklidesa - prosta implementacja

Warunek stopu postaci NWD(a, a) = a można zmodyfikować - gdyby nie zatrzymać obliczeń, następnym krokiem byłoby pytanie o NWD(a, 0) - przyjmijmy zatem, że implementacja powinna wskazywać wynik NWD(a, 0) = a, a otrzymamy prosty kod:

```
# Najwiekszy wspolny dzielnik – wersja rekurencyjna
def nwd(a, b):
    assert a != 0
    if b == 0:
        return a
    else:
        return nwd(b, a % b)
```

Czyli: jeżeli b jest zerem, to NWD(a, b) wynosi a, w przeciwnym wypadku - NWD(a, b) jest równe tyle, co NWD(b, $a\%b$). Jeżeli w danych wejściowych a nie jest zerem, to nigdy w wyniku wykonywania obliczeń do tego nie dojdzie, bo zawsze $a > a\%b$. Linijkę z asercją oczywiście w praktyce (np. na SPOJu) pomijamy, ufając sobie że nie wywołamy funkcji nwd z pierwszym argumentem równym zero.

Implementacja algorytmu Euklidesa - zwięzła implementacja w C++

Poniższa rekurencyjna implementacja jest niezwykle krótka, gdy zapisze się ją w języku C++ (lub C), z wykorzystaniem operatora trójargumentowego. Przykład poniżej:

```
unsigned int nwd(unsigned int const &a, unsigned int const &b) {
    assert(a != 0);
    return b ? a : nwd(b, a % b);
}
```

Po usunięciu asercji, zostałoby nam proste, jednolinijkowe ciało funkcji.
Aby nie zredukować rozwiązywania zadań do przepisania kodu przygotowanego od razu w C++, w dalszej części prezentacji pokazywane jednak będą wyłącznie pseudokody oparte na składni Pythona, takie jak na poprzednim slajdzie.

Implementacja algorytmu Euklidesa - po derekursywacji

Dokonując derekursywacji mini-implementacji zaproponowanej na poprzednich slajdach, możemy zapisać:

```
# Najwiekszy wspolny dzielnik — wersja iteracyjna
def nwd(a, b):
    assert a != 0
    while b != 0:
        t = b
        b = a % b
        a = t
    return a
```

Ta wersja jest już zupełnie optymalnym rozwiązaniem w dziedzinie pojedynczego obliczania NWD pary liczb naturalnych - złożoność obliczeniowa to $O(\log(\max(n, m)))$, po usunięciu rekurencji mamy także zagrawantowaną złożoność pamięciową $O(1)$ - i nic asymptotycznie lepszego nie uda się zaproponować. Ale...

Algorytm Steina

Zauważyliśmy już, że wielokrotne odejmowanie jest nieoptymalne. Jednakże, większość procesorów lepiej radzi sobie z operacjami bitowymi i dodawaniem, niż z obliczaniem reszty z dzielenia. Stąd rodzi się pomysł na implementację nazywany "binarnym NWD" lub "algorytmem Steina". Opiera się on na sposrzeżeniach:

1. Jeżeli przyjmiemy, że $\mathrm{NWD}(a, 0) = \mathrm{NWD}(0, a) = a$, oraz $\mathrm{NWD}(0, 0) = 0$, obliczenia pozostaną poprawne.
2. Dla parzystych a, b mamy $\mathrm{NWD}(a, b) = 2 \cdot \mathrm{NWD}(a/2, b/2)$ (co łatwo jest zaimplementować przesunięciami bitowymi).
3. Jeżeli a jest parzyste, zaś b jest nieparzyste (pamiętajmy też o przemienności NWD), to $\mathrm{NWD}(a, b) = \mathrm{NWD}(a/2, b)$ (można zredukować wszystkie zera z końca zapisu binarnego liczby a).
4. Dla nieparzystych a, b mamy $\mathrm{NWD}(a, b) = \mathrm{NWD}(|a - b|/2, \min(a, b))$.

Algorytm Steina nie poprawia asymptotycznej złożoności, lecz pozwala zmniejszyć praktyczny czas obliczeń, nawet o 62%.

Rozszerzony algorytm Euklidesa - czego dotyczy?

Nieznacznie rozszerzając postępowanie wykonywane przy liczeniu NWD za pomocą algorytmu Euklidesa (także z heurystyką polegającą na obliczaniu reszty z dzielenia), możemy uzyskać metodę szukania liczb całkowitych p i q takich, że dla zadanych liczb naturalnych a, b:

$$p \cdot a + q \cdot b = \text{NWD}(a, b)$$

Postępowanie polegające na jednoczesnym obliczaniu liczb p, q, oraz NWD(a, b) nazywamy rozszerzonym algorytmem Euklidesa.

Rozszerzony algorytm Euklidesa - przykład

Zauważmy, jak postępujemy obliczając NWD(162, 60). W ujęciu klasycznym:

$\mathrm{NWD}(160,\ 62) = \mathrm{NWD}(102,\ 60)$	$102 = 162 - 60$
$\mathrm{NWD}(102,\ 60) = \mathrm{NWD}(42,\ 60)$	$42 = 102 - 60 = (162 - 60) - 60 = 162 - 2 \cdot 60$
$\mathrm{NWD}(60,\ 42) = \mathrm{NWD}(18,\ 42)$	$18 = 60 - 42 = 60 - (162 - 2 \cdot 60) = -162 + 3 \cdot 60$
$\mathrm{NWD}(42,\ 18) = \mathrm{NWD}(24,\ 18)$	$24 = 42 - 18 = (162 - 2 \cdot 60) - (-162 + 3 \cdot 60) = 2 \cdot 162 - 5 \cdot 60$
$\mathrm{NWD}(24,\ 18) = \mathrm{NWD}(6,\ 18)$	$6 = 24 - 18 = (2 \cdot 162 - 5 \cdot 60) - (-162 + 3 \cdot 60) = 3 \cdot 162 - 8 \cdot 60$

$$\mathrm{NWD}(18,\ 6) = \mathrm{NWD}(12,\ 6) = \mathrm{NWD}(6,\ 6) = 6$$

Wykonując typowe dla algorytmu Euklidesa operacje, przy okazji zauważyliśmy, że $6 = 3 \cdot 162 - 8 \cdot 60$.

Rozszerzony algorytm Euklidesa - uproszczenie przykładu

Znacznie mniej obliczeń, a ten sam efekt, otrzymamy stosując heurystykę polegającą na zastąpieniu odejmowania resztami z dzielenia.

$$\begin{aligned} \mathrm{NWD}(160,\,62) &= \mathrm{NWD}(42,\,60) & 42 &= 162 - 2 \cdot 60 \\ \mathrm{NWD}(60,\,42) &= \mathrm{NWD}(18,\,42) & 18 &= 60 - 42 = 60 - (162 - 2 \cdot 60) = -162 + 3 \cdot 60 \\ \mathrm{NWD}(42,\,18) &= \mathrm{NWD}(6,\,18) & 6 &= 42 - 2 \cdot 18 = (162 - 2 \cdot 60) - 2 \cdot (-162 + 3 \cdot 60) = 3 \cdot 162 - 8 \cdot 60 \end{aligned}$$

$$\mathrm{NWD}(18,\,6) = \mathrm{NWD}(0,\,6) = 6$$

Zatem rozszerzenie algorytmu Euklidesa doskonale komponuje się ze stosowaniem reszt z dzielenia - znacznie szybciej, bez specjalnego dostosowywania się do nowej metody, otrzymaliśmy ten sam rezultat co poprzednio. Pesymistyczna złożoność tej wersji jest taka sama, jak bez rozszerzenia: $O(\log(\max(n, m)))$ - jedynie stała jest kilkukrotnie większa.

Rozszerzony algorytm Euklidesa - pseudokod

Postępowanie przedstawione na przykładzie, na poprzednich dwóch slajdach, jest bardzo intuicyjne. Oto przykład zapisania tego rozumowania w naszym pseudokodzie:

```
# Rozszerzony algorytm Euklidesa
# wynikiem sa trzy liczby: [r, p, q] takie, ze
# NWD(a, b) = r = p * a + q * b
def rozszerzonyEuklides(a, b):
    assert a != 0
    # a = pa * a + qa * b
    pa = 1
    qa = 0
    # b = pb * a + qb * b
    pb = 0
    qb = 1
    while b != 0:
        # dzielenie bez reszty: "ile razy a sie miesci w b"
        quot = a / b
        # zapamietanie wartosci a i pasujacych do niej wspolczynnikow
        old_a = a
        old_pa = pa
        old_qa = qa
        # przypisanie b do a (wraz ze wspolczynnikami)
        a = b
        pa = pb
        qa = qb
        # przypisanie do b reszty z dzielenia a przez b, obl. wspolczynnikow
        b = old_a - r * b
        pb = old_pa - r * pb
        qb = old_qa - r * qb
    return [a, pa, qa]
```

Odwrotność w pierścieniach $\mathbb{Z}_n$ - definicja

"Odwrotnością" przeważnie nazywa się element odwrotny względem działania mnożenia, czyli taki element a^{-1}, który dla danego a spełnia zależność $a^{-1} \cdot a = a \cdot a^{-1} = 1$, gdzie 1 to element neutralny mnożenia. Tak jest też w tym przypadku. Mnożenie w pierścieniu $\mathbb{Z}_n$ (jego elementami są liczby całkowite od 0 do $n-1$) oznacza wyznaczanie reszty z dzielenia iloczynu pary liczb przez n, np. w pierścieniu $\mathbb{Z}_7$ otrzymujemy, że 5 pomnożone przez 4 daje 6 (gdyż w zwykłym zbiorze liczb całkowitych $\mathbb{Z}$ jest to liczba 20, która przy dzieleniu przez 7 daje resztę 6).
Odwrotność elementu w pierścieniu $\mathbb{Z}_n$ nie zawsze istnieje (za chwilę to pokażemy). Jeżeli jednak akurat istnieje odwrotność liczby a w pierścieniu $\mathbb{Z}_n$ (oznaczmy ją jako p), to oznacza to, że:

$$\exists_{k \in \mathbb{Z}} \; a \cdot p = k \cdot n + 1$$

Odwrotność w pierścieniach $\mathbb{Z}_n$ - wyznaczanie

Okazuje się, że warunkiem koniecznym istnienia odwrotności liczby a w $\mathbb{Z}_n$ jest to, aby liczby a i n były względnie pierwsze, czyli $\text{NWD}(a, n) = 1$. Przypomnijmy, co nam może dać rozszerzony algorytm Euklidesa:

$$p \cdot a + q \cdot n = \text{NWD}(a, n)$$

$$p \cdot a + q \cdot n = 1$$

$$p \cdot a = (-q) \cdot n + 1$$

$$p \cdot a \equiv (1 \mod n)$$

Jeżeli $\text{NWD}(a, n) = 1$, to odwrotnością liczby a w pierścieniu $\mathbb{Z}_n$ jest liczba p, którą wyznaczamy za pomocą rozszerzonego algorytmu Euklidesa. Oznaczamy $p \equiv (a^{-1} \mod n)$.

Wreszcie - chińskie twierdzenie o resztach!

Chińskie twierdzenie o resztach, w ujęciu formalnym, mówi o tym, mając dany układ kongruencji:

$$\begin{cases} x \equiv (m_1 \mod n_1) \\ x \equiv (m_2 \mod n_2) \\ \dots \\ x \equiv (m_k \mod n_k) \end{cases}$$

Można jednoznacznie wskazać, ile wynosi reszta z dzielenia liczby x przez NWW $\left(\bigcup_{i=1}^{k} \{n_i\}\right)$. Innymi słowy, istnieje dokładnie jedna taka liczba $1 \leqslant x_0 \leqslant \text{NWW}\left(\bigcup_{i=1}^{k} \{n_i\}\right)$, która spełnia wskazany układ kongruencji. Warto zauważyć, że jeżeli liczby n_i (dla $i = 1 \dots k$) są parami względnie pierwsze, to $\text{NWW}\left(\bigcup_{i=1}^{k} \{n_i\}\right) = n_1 \cdot n_2 \cdot \ldots \cdot n_k$.

Chińskie twierdzenie o resztach - uproszczenie

Aby rozwiązywać takie układy kongruencji, wystarczy jednak posiadać umiejętność sprowadzenia dwóch kongruencji do jednej równoważnej z nimi. Zapiszmy zatem:

$$\begin{cases} x \equiv (m_1 \mod n_1) \\ x \equiv (m_2 \mod n_2) \end{cases}$$

Co więcej, wystarczy, abyśmy potrafili rozwiązać przypadek względnie pierwszych n_1 i n_2. Jeżeli n_1 i n_2 nie są względnie pierwsze, to zamiast pary reszt z dzielenia przez n_1 i n_2 możemy rozważać reszty z dzielenia przez n_1 oraz $n_2/\mathrm{NWD}(n_1, n_2)$, a sprowadzimy problem do składania kongruencji ze względnie pierwszymi dzielnikami (i w wyniku otrzymamy resztę z dzielenia przez $n_1 \cdot (n_2/\mathrm{NWD}(n_1, n_2)) = \mathrm{NWW}(n_1, n_2)$).

Chińskie twierdzenie o resztach - wyznaczenie wzoru

Zatem szukamy reszty z dzielenia liczby x przez $n_1 n_2$, wiedząc, że liczby n_1 oraz n_2 są względnie pierwsze, oraz:

$$\begin{cases} x \equiv (m_1 \mod n_1) \\ x \equiv (m_2 \mod n_2) \end{cases}$$

Z rozszerzonego algorytmu Euklidesa, możemy wskazać takie p_1, p_2, że:

$$p_1 \cdot n_1 + p_2 \cdot n_2 = \text{NWD}(n_1, n_2) = 1$$

Wówczas $p_1 n_1$ daje resztę 0 przy dzieleniu przez n_1 i resztę 1 przy dzieleniu przez n_2. Analogicznie jest z $p_2 n_2$ - otrzymujemy resztę 0 przy dzieleniu przez n_2 i resztę 1 przy dzieleniu przez n_1. Zatem liczba $\mathbf{x_0 = p_1 n_1 m_2 + p_2 n_2 m_1}$ musi spełniać obie kongruencje. Pozostałe rozwiązania różnią się o całkowitą wielokrotność $n_1 n_2$. Mamy zatem całą klasę rozwiązań, co należało wyznaczyć.

Chińskie twierdzenie o resztach - pseudokod

Algorytm postępowania z układem kongruencji jest obudową na rozszerzony algorytm Euglidesa, a złożoność wynosi $O(\log(\max(n_1, n_2)))$. Łączenie większej liczby kongruencji naraz nie poprawia tego wyniku, zatem wystarczy nam "doklejać" kongruencje z układu po jednej, aż do uzyskania wyniku końcowego.

```
# przyjmuje jako argumenty dwie pary: [ni, mi]
# oznaczajace, ze szukane x przystaje do mi modulo ni
# wynikiem jest jedna para [n, m] taka, ze x przystaje do n modulo m
# i wynik jest rownowazny koniunkcji argumentow
def zlaczKongruencje([n1, m1], [n2, m2]):
    # zapewniamy wzgledna pierwszosc n1 i n2
    n2 = n2 / nwd(n1, n2)
    # "naprawiamy" m2, jesli potrzeba (reszta z reszty dzielenia)
    m2 = m2 % n2
    # odczytujemy wynik rozszerzonego algorytmu Euklidesa
    [u, p1, p2] = rozszerzonyEuklides(n1, n2)
    # NWD musi byc rowne jeden, bo przeciez to zapewnialismy
    assert u == 1
    # ustalamy parametry wyniku, zgodnie z poprzednim slajdem
    n = n1 * n2
    m = p1 * n1 * m2 + p2 * n2 * m1
    # zapewniamy, ze m nalezy do Z_n
    m = m % n
    return [n, m]
```