NWD - algorytm Euklidesa

Problem

Dla danych dwóch liczb naturalnych *a* i *b* znaleźć największą liczbę naturalną *c*, która dzieli bez reszty liczbę *a* i dzieli bez reszty liczbę *b*.

Liczba *c* o powyższej własności nosi nazwę **NWD - największego wspólnego dzielnika** *a* i *b* (ang. GCD - greatest common divisor). NWD znajdujemy za pomocą znanego algorytmu Euklidesa, będącego jednym z najstarszych algorytmów, ponieważ pojawił się on w dziele *Elementy* napisanym przez Euklidesa około 300 p.n.e. Właściwie Euklides nie podał algorytmu dla liczb, lecz dla dwóch odcinków. Chodziło w nim o znalezienie wspólnej miary (czyli odcinka jednostkowego), która mogłaby posłużyć do zmierzenia obu danych odcinków - wspólna miara odkłada się w każdym z odcinków całkowitą liczbę razy.

Rozwiązanie 1

Euklides wykorzystał prosty fakt, iż NWD liczb *a* i *b* dzieli również ich różnicę. Zatem od większej liczby odejmujemy w pętli mniejszą dotąd, aż obie liczby się zrównają. Wynik to NWD dwóch wyjściowych liczb.

Algorytm Euklidesa

Wejście

a, b - liczby naturalne, których NWD poszukujemy, *a*, *b* ∈ N

Wyjście:

NWD liczb *a* i *b*

Lista kroków:

K01: **Dopóki** *a* ≠ *b* **wykonuj** krok K02

K02: **Jeśli** *a* < *b*, **to** *b* ← *b* - *a* *; od większej liczby odejmujemy mniejszą aż się zrównają*
inaczej *a* ← *a* - *b*

K03: **Pisz** *a* *; wtedy dowolna z nich jest NWD*

K04: **Zakończ**

Rozwiązanie 2

Pierwsze rozwiązanie problemu znajdowania NWD jest złe z punktu widzenia efektywności. Wyobraźmy sobie, iż *a* jest równe 4 miliardy, a *b* jest równe 2. Pętla odejmująca będzie wykonywana dotąd, aż zmienna *a* zrówna się ze zmienną *b*, czyli w tym przypadku 2 miliardy razy - trochę dużo. Tymczasem można wykorzystać operację reszty z dzielenia. Mniejszą liczbę można odjąć od większej liczby tyle razy, ile wynosi iloraz całkowity tych liczb. Po odejmowaniu pozostaje reszta z dzielenia - a Euklides właśnie zauważył, iż NWD dzieli również różnicę danych liczb, czyli:

NWD(*a*,*b*) = NWD(*a* **mod** *b*,*b*)

Ponieważ reszta zawsze jest mniejsza od dzielnika, wymieniamy *a* z *b*, a *b* z *a* **mod** *b*. Jeśli otrzymamy wynik *b* = 0, to w *a* jest ostatni dzielnik dzielący bez reszty różnicę.

Algorytm Euklidesa

Wejście

a, *b* - liczby naturalne, których NWD poszukujemy, *a*, *b* $\in N$

Wyjście:

NWD liczb *a* i *b*

Zmienne pomocnicze

t - tymczasowo przechowuje dzielnik, *t* $\in N$

Lista kroków:

K01:	**Dopóki** *b* ≠ 0 **wykonuj** kroki K02...K04	
K02:	*t* ← b	*; zapamiętujemy dzielnik*
K03:	*b* ← *a* **mod** *b*	*; wyznaczamy resztę z dzielenia, która staje się dzielnikiem*
K04:	*a* ← *t*	*; poprzedni dzielnik staje teraz się dzielną*
K05:	**Pisz** *a*	*; NWD jest ostatnią dzielną*
K06:	**Zakończ**	

Rozwiązanie 3

Istnieje algorytm znajdowania NWD, w którym nie wykonuje się dzieleń - są one kłopotliwe dla małych procesorów, np. dla kontrolerów jednoukładowych, i zajmują stosunkowo dużo czasu procesora. Algorytm ten wykorzystuje fakt, iż wszystkie liczby są przechowywane w komputerze w postaci ciągu bitów. Operacje dzielenia z resztą zastępuje się przesunięciami bitów, które są proste w realizacji i wykonywane szybko, nawet na najprostszym sprzęcie komputerowym. W efekcie otrzymujemy około 60% przyrost szybkości wyznaczania NWD w stosunku do standardowego algorytmu Euklidesa. Opisywany algorytm nosi nazwę **binarnego algorytmu NWD** (ang. binary GCD algorithm).

Algorytm redukuje problem znajdowania NWD przez stosowanie poniższych równoważności:

1. NWD(0, b) = b, ponieważ każda liczba naturalna dzieli zero, a b jest największą liczbą dzielącą b. Podobnie NWD(a, 0) = a. Natomiast NWD(0, 0) nie jest zdefiniowane.
2. Jeśli liczby a i b są parzyste, to NWD(a, b) = 2NWD($^{a}/_{2}$, $^{b}/_{2}$), ponieważ obie posiadają wspólny podzielnik 2.
3. Jeśli liczba a jest parzysta a b jest nieparzysta, to NWD(a, b) = NWD($^{a}/_{2}$, b), ponieważ 2 nie jest wspólnym podzielnikiem i można go pominąć. Podobnie jest w przypadku odwrotnym, gdy a jest nieparzyste a b jest parzyste, wtedy NWD(a, b) = NWD(a, $^{b}/_{2}$).
4. Jeśli obie liczby a i b są nieparzyste, a $a \geq b$, to NWD(a, b) = NWD($^{(a-b)}/_{2}$, b), inaczej jeśli obie są nieparzyste i $a < b$, to NWD(a, b) = NWD($^{(b-a)}/_{2}$, a). Takie same operacje wykonuje w pętli podstawowy algorytm Euklidesa - od większej liczby odejmuje mniejszą. Podzielenie różnicy przez 2 daje zawsze liczbę całkowitą, ponieważ odejmowane są dwie liczby nieparzyste.
5. Kroki 3 i 4 należy powtarzać aż do otrzymania b = 0. Wtedy NWD jest równy $2^{k}a$, gdzie k jest liczbą wspólnych czynników 2 wyeliminowanych w kroku 2. Mnożenie 2^{k} wykonujemy przez przesunięcie bitów zmiennej a o k pozycji w lewo.

Algorytm Euklidesa

Wejście

a, *b* - liczby naturalne, których NWD poszukujemy, *a*, *b* $\in C$

Wyjście:

NWD liczb *a* i *b*

Zmienne pomocnicze

k - przechowuje liczbę wspólnych podzielników 2, *k* $\in C$

r - wykorzystywane do przechowywania różnicy *a* i *b*, *r* $\in C$

Lista kroków:

K01:	**Jeśli** *a* = 0, **to pisz** *b* i **zakończ**	*; NWD(0,b) = b*
K02:	**Jeśli** *b* = 0, **to pisz** *a* i **zakończ**	*; NWD(a,0) = a*
K03:	*k* ← 0	*; inicjujemy liczbę wspólnych podzielników 2*
K04:	**Dopóki** *a* i *b* parzyste **wykonuj** kroki K05...K07	*; usuwamy z a i b wspólne czynniki 2, zapamiętując ich liczbę w k*
K05:	*a* ← *a* **shr** 1	*; przesuwamy bity a o 1 w prawo*
K06:	*b* ← *b* **shr** 1	*; przesuwamy bity b o 1 w prawo*
K07:	*k* ← *k* + 1	
K08:	**Jeśli** *a* = 0, **to** *a* ← *b* i **idź do** K18	*; NWD(0,b) = b*
K09:	**Dopóki** *a* parzyste **wykonuj** *a* ← *a* **shr** 1	*; eliminujemy podzielniki 2 z a*
K10:	**Dopóki** *b* parzyste **wykonuj** *b* ← *b* **shr** 1	*; eliminujemy podzielniki 2 z b, teraz a i b są nieparzyste*
K11:	**Jeśli** *a* ≥ *b*, **to idź do** K16	
K12:	*r* ← (*b* - *a*) **shr** 1	*; NWD(a,b) = NWD($^{(b-a)}/_2$,a)*
K13:	*b* ← *a*	*; zamieniamy b z a*
K14:	*a* ← *r*	*; a z różnicą r*
K15:	**Idź do** K17	
K16:	*a* ← (*a* - *b*) **shr** 1	*; NWD(a,b) = NWD($^{(a-b)}/_2$,b)*
K17:	**Jeśli** *b* ≠ 0, **to idź do** K08	
K18:	**Jeśli** *k* > 0, **to** *a* ← *a* **shl** *k*	*; przesuwamy bity a o k pozycji w lewo → mnożenie przez 2^k*
K19:	**Pisz** *a*	
K20:	**Zakończ**	