Algorytmy szkolne, część 1

dr Marcin Ziółkowski

Instytut Matematyki i Informatyki
Akademia im. Jana Długosza w Częstochowie

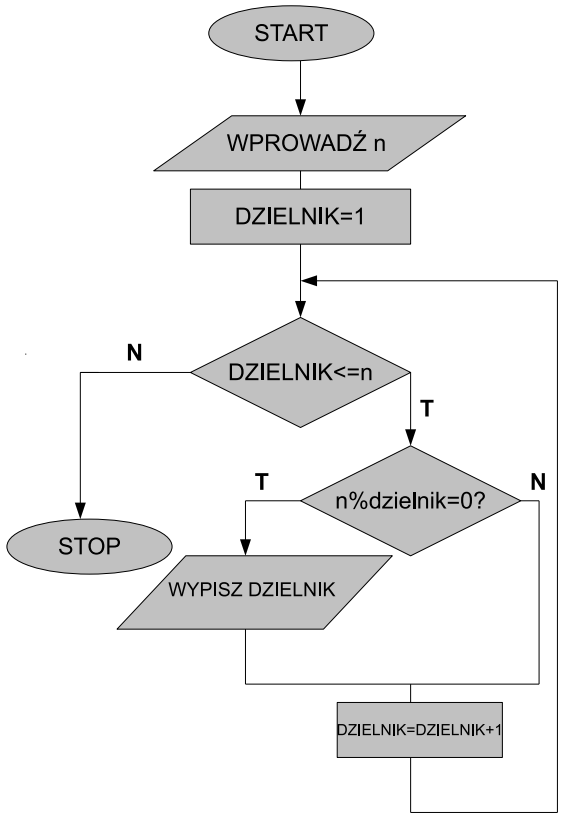
11 kwietnia 2017 r.

ALGORYTMY SZKOLNE

Nauczyciele matematyki oraz informatyki w swojej praktyce szkolnej mają często do czynienia z pewnymi standardowymi algorytmami. Celem tego wykładu jest zapoznanie z najważniejszymi algorytmami wykorzystywanymi na lekcjach matematyki oraz informatyki oraz przedstawieniem pewnych możliwości ich udoskonalania w celu uzyskania algorytmów szybszych, które mają mniejszą złożoność obliczeniową. Do takich algorytmów należą m.in. wypisywanie dzielników, znajdowanie NWD oraz NWW dwóch lub większej ilości liczb, rozkład danej liczby naturalnej na czynniki pierwsze, sprawdzanie, czy dana liczba naturalna jest pierwsza, wypisywanie listy liczb pierwszych, sprawdzanie czy dana liczba jest doskonała, wypisywanie listy liczb doskonałych, wypisywanie par liczb zaprzyjaźnionych, a także odczytywanie liczb zapisanych w innym systemie liczenia oraz zapisywanie liczb w systemie o danej podstawie. Algorytmy te będą również ilustrowane programami w języku PYTHON.

WYPISYWANIE DZIELNIKÓW

Już w szkole podstawowej uczniowie wypisują dzielniki podanej liczby. Problem nie wydaje się trudny dla liczb małych, jednak dla większych liczb staje się skomplikowany nawet dla dorosłych ludzi i w celu jego rozwiązania wykorzystujemy często komputer. Na poprzednim wykładzie przedstawiony został prosty algorytm rozwiązujący ten problem:



WYPISYWANIE DZIELNIKÓW

A oto program, wykorzystujący ten algorytm, napisany w języku PYTHON:

WYPISYWANIE DZIELNIKÓW - PROSTY PROGRAM

```
print("Podaj liczbe")
a=int(input())
for i in range(1,a+1):
    if a%i==0:
        print(i)
input()
```

WYPISYWANIE DZIELNIKÓW

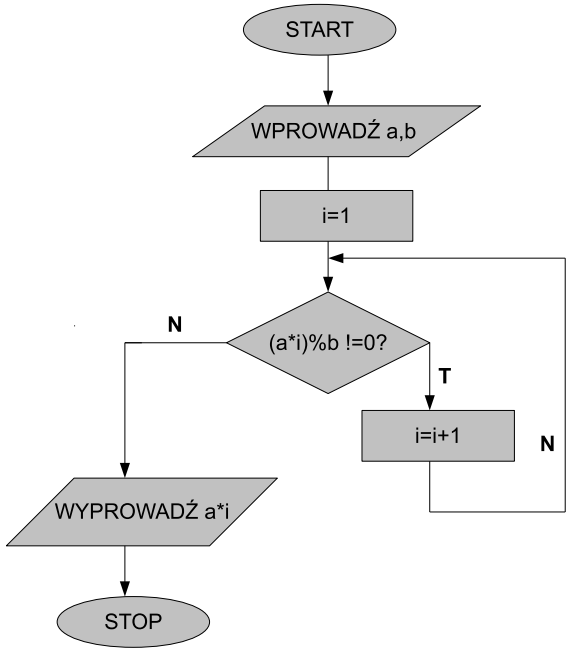
Dla małych liczb program działa bardzo szybko, dla większych jego działanie jest wolne. Dzieje się tak dlatego, że sprawdzamy po kolei wszystkie liczby począwszy od 1 do liczby n. Okazuje się, że jeśli dzielniki będziemy wypisywać parami (np. dla liczby 40: $1, 40, 2, 20, 4, 10, 5, 8$), to możemy sprawdzać dzielniki tylko do liczby całkowitej nie większej od $\sqrt{n}$, co znacznie poprawia jego szybkość, otrzymujemy złożoność rzędu $O(\sqrt{n})$ zamiast $O(n)$. Na przykład dla liczby 1000000 wystarczy sprawdzić 1000, a nie 1000000 dzielników. Oto modyfikacja:

WYPISYWANIE DZIELNIKÓW - PROGRAM SZYBSZY

```
print("Podaj liczbe")
import math
a=int(input())
for i in range(1,math.floor(math.sqrt(a))+1):
    if i==a/i:
        print(i,",")
    elif a%i==0:
        print(i,",",int(a/i))
input()
```

OBLICZANIE NWW

Kolejnym problemem jest znajdowanie NWW dwóch lub większej ilości liczb naturalnych. W szkole często wykorzystuje się do tego rozkład liczby naturalnej na czynniki pierwsze. Jednak znajdowanie NWW może być oparte na algorytmie nie wykorzystującym tego rozkładu. Algorytm taki jest szybszy, a jego idea była również przedstawiona na poprzednim wykładzie:



OBLICZANIE NWW

Program realizujący ten algorytm jest przedstawiony poniżej. Aby program działał jeszcze szybciej zapewniamy, aby mnożona była większa z liczb (instrukcja warunkowa IF).

ZNAJDOWANIE NWW - PROGRAM

```
print("Podaj dwie liczby naturalne")
a,b=int(input()),int(input())
if a<b:
   x=a
   a=b
   b=x
i=1
while (a*i)%b !=0:
   i=i+1
print("NWW liczb", a, "oraz", b, "wynosi",a*i)
input()
```

OBLICZANIE NWW

Zauważmy, że ta sama idea może mieć zastosowanie przy obliczaniu NWW większej ilości liczb. Jedną z tych liczb (najlepiej największą, ale trzeba tu stosować wówczas algorytmy sortujące) mnożymy przez kolejne dodatnie liczby naturalne do momentu, aż uzyskany iloczyn będzie podzielny przez wszystkie pozostałe liczby. Możemy też napisać program rekurencyjny (ale o tym na jednym z kolejnych wykładów), ponieważ $NWW(a, b, c) = NWW(NWW(a, b), c)$. Oto przykład programu dla trzech liczb:

OBLICZANIE NWW TRZECH LICZB - PROGRAM

```
print("Podaj trzy liczby naturalne")
a,b,c=int(input()),int(input()), int(input())
i=1
while (a*i)%b !=0 or (a*i)%c!=0:
   i=i+1
print("NWW liczb", a, "oraz", b, "oraz", c, "wynosi",a*i)
input()
```

OBLICZANIE NWD

Następny problem to znajdowanie NWD dwóch lub większej ilości liczb naturalnych. W szkole również wykorzystuje się do tego rozkład liczby naturalnej na czynniki pierwsze. Znajdowanie NWD może być jednak również dużo łatwiejsze. Możemy na przykład od mniejszej z liczb odejmować liczbę jeden dopóki większa liczba nie podzieli się przez uzyskaną różnicę i jednocześnie różnica ta nie będzie dzielnikiem mniejszej z liczb. Oto przykład takiego programu:

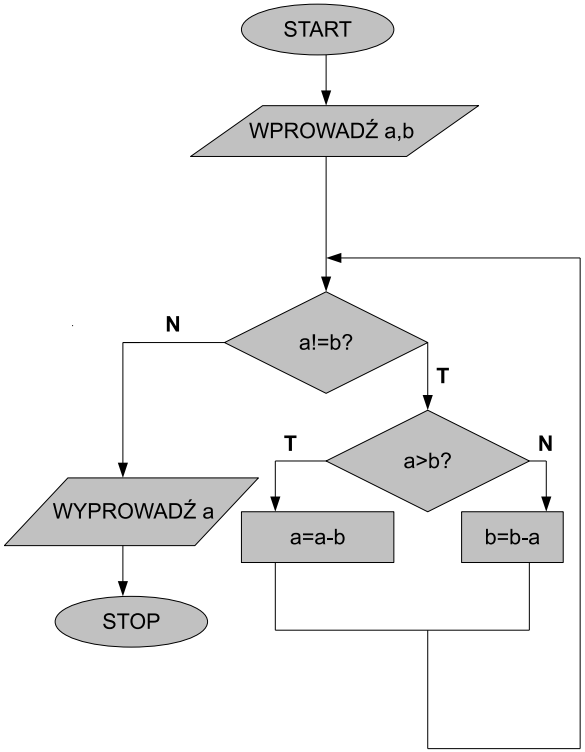
OBLICZANIE NWD

ZNAJDOWANIE NWD - PROGRAM

```
print("Podaj dwie liczby naturalne")
a,b=int(input()),int(input())
if a>b:
   x=a
   a=b
   b=x
temp=a
while b%temp!=0 or a%temp!=0:
   temp=temp-1
print("NWD wynosi",temp)
input()
```

OBLICZANIE NWD - ALGORYTM EUKLIDESA

Innym ze znanych sposobów jest na przykład algorytm Euklidesa wykorzystujący jedynie operację odejmowania przedstawiony na poprzednim wykładzie:



OBLICZANIE NWD - ALGORYTM EUKLIDESA

Program realizujący ten algorytm jest przedstawiony poniżej.

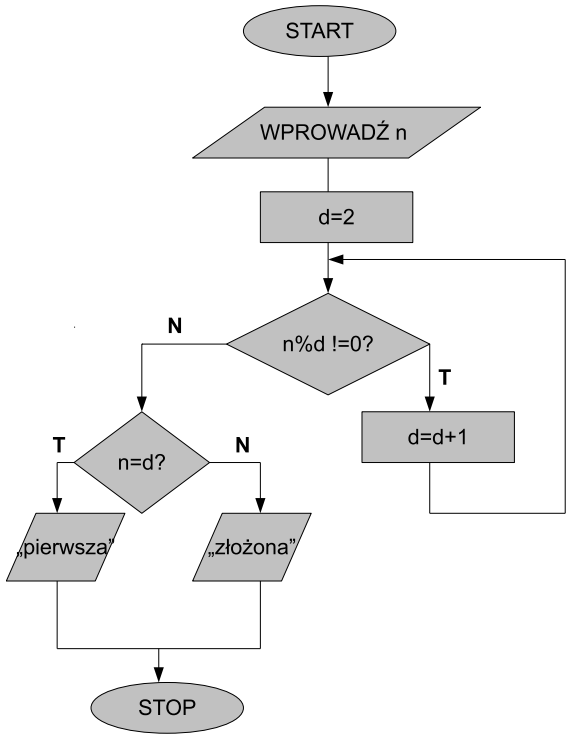
ZNAJDOWANIE NWD, ALGORYTM EUKLIDESA - PROGRAM

```
print("Podaj dwie liczby naturalne")
a,b=int(input()),int(input())
while a!=b:
    if(a>b):
        a=a-b
    else:
        b=b-a
print("NWD wynosi",a)
input()
```

Ideę tę również możemy rozszerzyć dla większej ilości liczb lub wykorzystać rekurencję, ponieważ dla największego wspólnego dzielnika mamy również zależność: $NWD(a, b, c) = NWD(NWD(a, b), c)$.

LICZBY PIERWSZE

Problem znajdowania liczb pierwszych jest problemem bardzo ważnym z punktu widzenia matematyki i informatyki. Liczby pierwsze są bowiem wykorzystywane m.in. w kryptografii (szyfrowanie danych) oraz mają wiele innych zastosowań. Test sprawdzania, czy dana liczba jest pierwsza może być oparty na przykład o poniższy algorytm:



Program napisany w oparciu o ten algorytm wygląda następująco:

TEST PIERWSZOŚCI - PROGRAM

```
print("Podaj liczbe naturalna")
a=int(input())
d=2
while a%d!=0:
   d=d+1
if(a==d):
   print("Liczba", a, "jest pierwsza")
else:
   print("Liczba", a, "jest zlozona")
input()
```

Program ten można przyspieszyć, szukając dzielników tylko do liczby całkowitej nie większej od $\sqrt{n}$. Daje to złożoność rzędu $O(\sqrt{n})$ zamiast liniowej $O(n)$.

TEST PIERWSZOŚCI - PROGRAM SZYBSZY

```
print("Podaj liczbe naturalna")
a=int(input())
d=2
import math
while a%d!=0 and d<=math.floor(math.sqrt(a)):
   d=d+1
if(d==1+math.floor(math.sqrt(a))):
   print("Liczba", a, "jest pierwsza")
else:
   print("Liczba", a, "jest zlozona")
input()
```

LICZBA DOSKONAŁA - DEFINICJA

Liczbą doskonałą nazywamy liczbę naturalną, która jest równa sumie mniejszych od siebie dzielników.

Liczby doskonałe to wielka zagadka teorii liczb, ponieważ:

1. Nie wiadomo, czy liczb doskonałych jest skończenie czy nieskończenie wiele
2. Wszystkie znalezione do tej pory liczby doskonałe są parzyste, nie wiadomo czy istnieje nieparzysta liczba doskonała
3. Nie ma żadnego efektywnego algorytmu znajdowania liczb doskonałych, istnieje jednak związek pomiędzy liczbami doskonałymi i pewnymi liczbami pierwszymi (dokładniej liczbami pierwszymi Mersene'a, czyli liczbami pierwszymi, które da się przedstawić w postaci $1 + 2 + \ldots + 2^n$)

Oto kilka znanych liczb doskonałych:
$6, 28, 496, 8128, 33550336, 8589869056, 137438691328.$

LICZBY DOSKONAŁE

W oparciu o definicję liczby doskonałej można łatwo skonstruować algorytm oraz napisać program, który sprawdza, czy dana liczba jest liczbą doskonałą. Oto przykład takiego programu:

TEST DOSKONAŁOŚCI - PROGRAM

```
print("Podaj liczbe naturalna")
a=int(input())
suma=0
for d in range(1,a):
    if a%d==0:
        suma=suma+d
if(suma==a and a>0):
    print("Liczba", a, "jest doskonala")
else:
    print("Liczba", a, "nie jest doskonala")
input()
```

LICZBY DOSKONAŁE

I jeszcze program szybszy:

TEST DOSKONAŁOŚCI - PROGRAM SZYBSZY

```
print("Podaj liczbe naturalna")
a=int(input())
import math
suma=1
for d in range(2,math.floor(math.sqrt(a))+1):
    if a%d==0 and a!=a/d:
        suma=suma+d+a/d
    elif a%d==0:
        suma=suma+d
if(suma==a and a>1):
    print("Liczba", a, "jest doskonala")
else:
    print("Liczba", a, "nie jest doskonala")
input()
```

LICZBY ZAPRZYJAŹNIONE

LICZBY ZAPRZYJAŹNIONE - DEFINICJA

Dwie różne liczby naturalne nazywamy zaprzyjaźnionymi, jeśli jedna z nich jest równa sumie dzielników drugiej liczby mniejszych od niej i na odwrót.

Z liczbami zaprzyjaźnionymi jest podobny problem. Nie wiadomo, czy takich par liczb jest nieskończenie wiele i czy istnieją liczby zaprzyjaźnione, z których jedna byłaby parzysta, a druga nieparzysta.
Oto przykłady takich par liczb:

LICZBY ZAPRZYJAŹNIONE

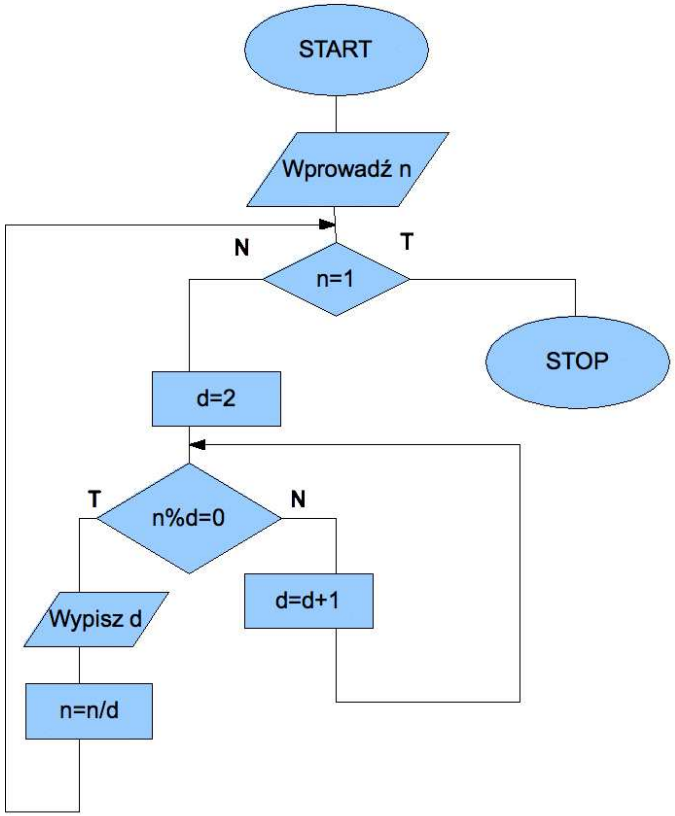
220 i 284 1184 i 1210 2620 i 2924 5020 i 5564 6232 i 6368
10744 i 10856 12285 i 14595

LICZBY ZAPRZYJAŹNIONE - PROGRAM SPRAWDZAJĄCY

```
print("Podaj dwie liczby naturalne")
a,b=int(input()), int(input())
import math
suma1=suma2=1
for d in range(2,math.floor(math.sqrt(a))+1):
   if a%d==0 and a!=a/d:
      suma1=suma1+d+a/d
   elif a%d==0:
      suma1=suma1+d
for d in range(2,math.floor(math.sqrt(b))+1):
   if b%d==0 and b!=b/d:
      suma2=suma2+d+b/d
   elif b%d==0:
      suma2=suma2+d
if(suma1==b and suma2==a and a!=b and a>1 and b>1):
   print("Liczby", a, " oraz ", b, "sa zaprzyjaznione")
else:
   print("Liczby", a, " oraz ", b, "nie sa zaprzyjaznione")
input()
```

ROZKŁAD LICZBY NATURALNEJ NA CZYNNIKI PIERWSZE

Na koniec omówimy algorytm rozkładu liczby na czynniki pierwsze. Przedstawiony poniżej algorytm opiera się o intuicyjne rozumienie rozkładu i rozkład ten jest wykonywany w oparciu o dzielenie przez kolejne liczby naturalne. Na przykład $60 = 2 \cdot 30 = 2 \cdot 2 \cdot 15 = 2 \cdot 2 \cdot 3 \cdot 5$.



ROZKŁAD LICZBY NATURALNEJ NA CZYNNIKI PIERWSZE

Program napisany w oparciu o ten algorytm przedstawia się następująco:

ROZKŁAD NA CZYNNIKI PIERWSZE - PROGRAM

```
print("Podaj liczbe naturalna:")
n=int(input())
while n!=1:
    d=2
    while n%d!=0:
        d=d+1
    print(d,",")
    n=n/d
input()
```