Piotr Chrząstowski-Wachtel
Uniwersytet Warszawski

Al Chwarizmi i trzy algorytmy Euklidesa

Algorytmika

- Najważniejsza część informatyki
- Opisuje jak rozwiązywać problemy algorytmiczne, jakie struktury danych dobierać, jak analizować zachowanie się programów.
- Pozwala na osiągnięcie znacznie bardziej spektakularnych wyników, niż samo przyspieszanie działania sprzętu

Skąd się wzięło słowo „algorytm”?

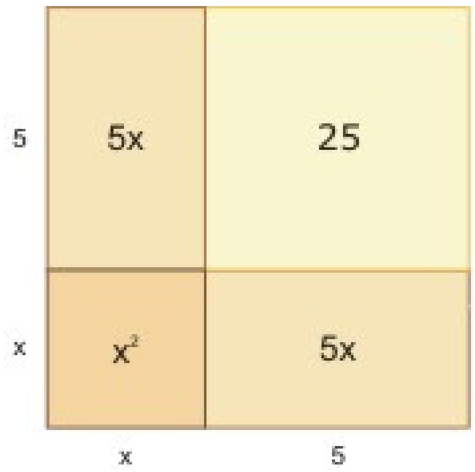
- Musi to być stare słowo, bo w większości języków brzmi ono podobnie.

Al Chwarizmi

- **Abu Ja'far Muhammad ibn Musa Al-Chuwarizmi (ok. 780 – ok. 850)**
- ***Hisab al-jabr w'al-muqabala***
- **Opisał ciekawe algorytmy, między innymi rozwiązywania równań, w tym kwadratowych!**

Przykład z Al Chwarizmiego

- **Rozwiązujemy równanie kwadratowe $x^2+10x=39$**
- ***Co nam wolno?***
  - używać tylko liczb dodatnich (długości odcinków)
  - dodawać i odejmować odcinki
  - mnożyć (tworzyć prostokąty)
  - pierwiastkować (wyznaczać bok kwadratu o zadanym polu)



S=64

więc x=3

Przykład z Euklidesa

Euklides

- Chcemy wyznaczyć Największy Wspólny Dzielnik dwóch liczb naturalnych NWD(m,n)
- ***Co nam wolno? To samo co poprzednio, a w szczególności:***
  - używać tylko liczb dodatnich (długości odcinków)
  - dodawać i odejmować odcinki

NWD(m,n)

- Okazuje się, że nie ma wzoru na NWD(m,n), który działałby dla każdych m i n.
- Euklides zauważył pewien (wcale nieoczywisty) fakt: Jeśli od większej liczby odejmiemy mniejszą, ich NWD się nie zmieni.

NWD(m,n) dla $0 \le m \le n$, $n>0$

- Pierwszy pomysł:

Euklides 1

- *Jeśli $m=0$ to $NWD(m,n)=n$*
- *Jeśli $m>0$ to $NWD(m,n)=NWD(n-m,m)$*

- Powyższy algorytm można zaprogramować za pomocą następującego kodu:

```
Wczytaj(m,n);
While m>0 do
   begin
    if m>n then Zamien(m,n);
    n:=n-m
   end;
Wypisz(n)
```

Czy widzimy problem?

- Wykonajmy krótkie szacowanie: ile zajęłoby komputerowi wykonanie tego algorytmu dla $n=10^{30}$, $m=1$ na najszybszych obecnie dostępnych komputerach świata (~1THz)?
  - Komputer ten byłby ok. 400-krotnie szybszy od mojego laptopa, więc z 10^9 uporałby się w 1/40 s.
  - Ale $10^{21}/40$s = ponad 790 miliardów lat ! To kilkadziesiąt razy więcej, niż trwa nasz Wszechświat!
  - W szyfrowaniu RSA stosuje się liczby rzędu 10^{200}

Czy pora się poddać?

- Przyjrzyjmy się nieco dokładniej, jak działa nasz algorytm.
  - Zauważmy, że wielokrotne odejmowanie mniejszej liczby od większej kończy się wtedy, gdy ta większa stopnieje poniżej tej odejmowanej.
  - To co zostaje z większej, to nic innego, niż reszta z dzielenia n przez m.
  - W tym momencie są one zamieniane i odejmowanie zaczynamy z nową, już mniejszą wartością, bo reszta jest zawsze ostro mniejsza od dzielnika.
  - Czyli całe odejmowanie jednej wartości ma na celu jedynie wyznaczenie reszty z dzielenia!

Czy nie ma lepszej metody obliczania reszty z dzielenia?

- Pamiętamy choćby z III klasy szkoły podstawowej dzielenie w słupkach, które dałoby się jakoś pewnie zaimplementować cyfra po cyfrze.
- Zatem możemy nasz algorytm zmodyfikować:

Euklides 2

- *Jeśli m=0 to NWD(m,n)=n*
- *Jeśli m>0 to NWD(m,n)=NWD(n mod m, m)*,
  - gdzie mod oznacza resztę z dzielenia

Kod algorytmu Euklides 2

- Powyższy algorytm można zaprogramować za pomocą następującego kodu:

```
Wczytaj(m,n);
While m>0 do
   begin
    r := n mod m;
    n := m;
    m := r
   end;
Wypisz(n)
```

Czy cokolwiek zyskaliśmy?

- Tym razem na pewno złośliwymi danymi nie będą największa liczba z badanego zakresu oraz jedynka. Dla takich danych algorytm wykona jedno dzielenie i wyjdzie reszta zero, która zakończy całą zabawę.
- Żeby zorientować się, co nas może w najgorszym razie czekać, spróbujmy wygenerować dane, dla których algorytm *Euklides 2* będzie działać najdłużej
- Jakie zatem dane są najbardziej złośliwe dla liczb- powiedzmy – 30-cyfrowych?

Złożoność pesymistyczna

- ... to liczba operacji, które algorytm wykona dla najbardziej złośliwych danych, czyli takich, które będą powodem możliwie długiej pracy algorytmu.
- Oczywiście musimy określić zakres, w jakim szukamy takich danych.
- Może spuśćmy nieco z tonu. Postarajmy się wygenerować najgorsze dane w zakresie 1..99. Dla jakiej pary liczb nasza pętla wykona się najwięcej razy?

Złożoność algorytmu *Euklides 2*

- Odwróćmy kota ogonem. Zamiast pytać się, dla jakich danych w określonym zakresie program będzie działał najdłużej, zadajmy pytanie, jaki musi być co najmniej zakres, żeby wymusić konkretną liczbę obrotów pętli.
- Zacznijmy od małych wartości. Jakie najmniejsze dane spowodują 1 obrót pętli? Oczywiście (1,1).
- Dla dwóch obrotów pętli potrzebujemy co najmniej liczb (2,3), bo dzielnik musi być większy od 1, a dzielna od dzielnika.

..Złożoność algorytmu *Euklides* 2

- Kolejne wartości:
  - 3 obroty pętli: (3,5), bo żeby dostać „najtańsze” 2 obroty, czyli parę (2,3), dzielnikiem musi być 3, a możliwie mała dzielna dająca przy tym dzielniku resztę 2 i od niego większa, to 5.
  - 4 obroty pętli: (5,8); dzielnik 5 i dzielna 5+3=8
  - 5 obrotów pętli: (8,13); bo 13=8+5
  - ...itd
- Zawsze chodzi o to, żeby wynik dzielenia był równy 1

..Złożoność algorytmu *Euklides* 2

- Podsumujmy rekordową parę (55,89) – 9 obrotów:
  - (55,89)
  - (34,55)
  - (21,34)
  - (13,21)
  - (8,13)
  - (5, 8)
  - (3, 5)
  - (2, 3)
  - (1, 2)
  - (0, 1)

Przyglądając sie drugiej kolumnie widzimy, że:

•nie da się 2 obrotów wykonać za pomocą liczb mniejszych od 3,

•nie da się 3 obrotów wykonać za pomocą liczb mniejszych od 5
...
•nie da się 9 obrotów wykonać za pomocą liczb mniejszych od 89

itd...

Liczby Fibonacciego

- Widzimy więc, że najzłośliwsze pary, to takie liczby (licząc od dołu), że mniejsza z nich jest większą z liczb poprzedniej pary, a większa jest ich sumą.
  - (55,89)
  - (34,55)
  - (21,34)
  - (13,21)
  - (8,13)
  - (5, 8)
  - (3, 5)
  - (2, 3)
  - (1, 2)
  - (0, 1)

Liczby w jednej kolumnie tworzą ciąg Fibonacciego

$F_0 = 0,\ F_1 = 1,$
$F_n = F_{n-1} + F_{n-2}$ dla $n>1$

Leonardo Fibonacci

0 1 1 2 3 5 8 13 21 34 55 89 144 233 377 ...

Liczby Fibonacciego

- Oto liczby obrotów pętli uzyskane dla możliwie złośliwych argumentów:
  - (F_{10},F_{11}) 9 obrotów
  - (F_9,F_{10}) 8 obrotów
  - (F_8,F_9) 7 obrotów
  - (F_7,F_8) 6 obrotów
  - (F_6,F_7) 5 obrotów
  - (F_5,F_6) 4 obroty
  - (F_4,F_5) 3 obroty
  - (F_3,F_4) 2 obroty
  - (F_2,F_3) 1 obrót
  - (F_0,F_1) 0 obrotów

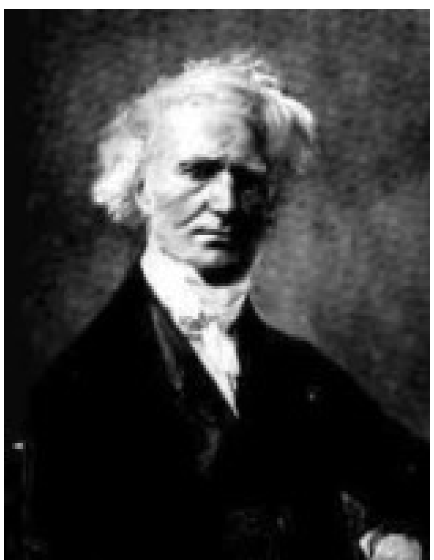
Z pary (F_{n-1},F_n) dla $n>2$ uzyskujemy $n-2$ obroty pętli.

Liczby Fibonacciego

- Wśród liczb 30-cyfrowych zatem najgorsze będą te, które są możliwie dużymi liczbami Fibonacciego mieszczącymi się w tym zakresie.
- Numer większej z nich pomniejszony o 2 będzie szukaną liczbą obrotów pętli
- Kluczowe jest zatem pytanie, jak szybko rosną liczby Fibonacciego?
- Jeśli szybko, to dobrze! Bo numer liczby będzie nieduży.

Liczby Fibonacciego

- Sam Fibonacci nie umiał wyznaczyć wzoru na „swoją” n-tą liczbę.
- Dokonał tego w XVIII wieku wielki Leonard Euler, a ponownie odkrył ten wzór Jacques Binet w XIX w.



Jacques Binet

Wzór Eulera-Bineta

$$F_n = \frac{1}{\sqrt{5}}\left(\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n\right)$$

Leonard Euler

Wnioski ze wzoru Eulera-Bineta

- Wprowadźmy oznaczenia

$$\varphi = \frac{1+\sqrt{5}}{2}, \hat{\varphi} = \frac{1-\sqrt{5}}{2}$$

- Zatem liczby te są równe odpowiednio ok. 1.618... oraz -0.618..., a wzór Eulera-Bineta przyjmuje postać

$$F_n = \frac{1}{\sqrt{5}}(\varphi^n - \hat{\varphi}^n)$$

- Teraz pomijając drugi składnik różnicy, dążący szybko do zera otrzymujemy prostszy wzór określający liczby Fibonacciego:

$$F_n = [\frac{1}{\sqrt{5}}\varphi^n]$$

Liczby Fibonacciego rosną wykładniczo szybko!

- Zatem od pewnego momentu każda kolejna liczba Fibonacciego jest od poprzedniej większa o ponad 60%
- Ponieważ chcemy znaleźć największą liczbę Fibonacciego nieprzekraczającą 10^{30}, więc chcemy rozwiązać nierówność
- $\varphi^n/\sqrt{5} < 10^{30}$, co po zlogarytmowaniu przy podstawie φ daje $n-2 < 30\log_\varphi 10 = 148{,}33...$
- Ostatecznie $n \leq 150$.

Wygraliśmy!

- Okazuje się, że nawet dla tak ogromnych danych, jak liczby 30-cyfrowe, liczba obrotów pętli nie przekroczy 150 (a nawet 148).
- Jest jednak łyżka dziegciu: operację mod się trudniej programuje, niż odejmowanie (i nieco dłużej wykonuje), tym niemniej naprawdę warto!

Czy nie ma zatem metody szybkiej, ale też łatwwej do zaprogramowania?

- Spróbujmy zastanowić się, jak na NWD wpływa parzystość argumentów Dla m≤n NWD(m,n) wynosi

Euklides 3: NWD(m,n)=

n Jeśli m=0

*2*NWD(m/2,n/2)* *jeśli* $m,n \in P$

NWD(m,n/2) *jeśli* $m \notin P$ $n \in P$

NWD(m/2,n) *jeśli* $m \in P$ $n \notin P$

NWD(n-m,m), *jeśli* $m,n \notin P$

Analiza złożoności algorytmu Euklides 3

- W każdym obrocie pętli wykonujemy dzielenie parzystych argumentów przez 2 lub odejmowanie nieparzystych
- Odjęcie jednej liczby nieparzystej od drugiej daje wynik parzysty.
- Zatem przynajmniej raz na dwa kroki przynajmniej jeden z argumentów podzielimy przez 2.

Analiza złożoności algorytmu Euklides 3 -cd.

- Zatem co najmniej raz na dwa obroty pętli wykonamy dzielenie przynajmniej jednego z argumentów przez 2.
- Zatem łączna liczba kroków algorytmu nie przekroczy 2(log m + log n).

Implementacja Euklides 3

- Algorytm ten wykorzystuje łatwe do zaimplementowania operacje porównania liczb, odejmowania oraz dzielenia przez 2 i mnożenia przez 2. Każda z nich ma złożonośc liniową ze względu na długość zapisu pozycyjnego liczby, więc łączna złożoność jest rzędu $(\log n)^2$; mamy bowiem logarytmicznie dużo obrotów pętli, każda z nich wykonuje logarytmicznie dużo działań na cyfrach.