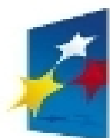
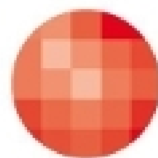
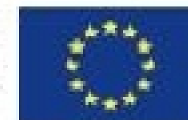
“Visiting Professor” Program







„”Projekt współfinansowany przez Unię Europejską w ramach Europejskiego Funduszu Społecznego”

“Visiting Professor” Program

Jesli Programowanie Jest Takie Łatwe To Dlaczego Jest Ono Takie Trudne?

Prof. Dr inż. Marek A. Suchenek

California State University
Dominguez Hills

WWSI, 30 maja 2011

Pierwszy program

Pierwszy program

Wczytaj a, $b \geq 0$ i wydrukuj wartosc a + b.

Pierwszy program

1: Wczytaj a, $b \geq 0$;

Pierwszy program

1: Wczytaj $a, b \geq 0$;

2: Jesli $b = 0$ to drukuj "a" i zatrzymaj sie;

Pierwszy program

1: Wczytaj a, $b \geq 0$;

2: Jesli $b = 0$ to drukuj “a” i zatrzymaj sie;

3: Dodaj 1 do a;

Pierwszy program

1: Wczytaj a, b $\geq$ 0;

2: Jesli b = 0 to drukuj “a” i zatrzymaj sie;

3: Dodaj 1 do a;

4: Odejmij 1 od b;

Pierwszy program

1: Wczytaj a, b ≥ 0;

2: Jesli b = 0 to drukuj “a” i zatrzymaj sie;

3: Dodaj 1 do a;

4: Odejmij 1 od b;

5: Skocz do 2;

Pierwszy program

1: Wczytaj a, $b \geq 0$;

2: Jesli $b = 0$ to drukuj “a” i zatrzymaj sie;

3: Dodaj 1 do a;

4: Odejmij 1 od b;

5: Skocz do 2;

A co sie stanie gdy $b < 0$?

Definicja pętli

Jesli pewna instrukcja programu moze byc wykonana wiecej niz raz w ciagu jednego przebiegu programu to program ten zawiera pętlę.

Klopoty z pętlą

Klopoty z pętlą

1: Skocz do 1

Klopoty z pętlą

1: Skocz do 1

Ta pętla nigdy się nie zakończy!

Klopoty z pętlą

1: Skocz do 1

Ta pętla nigdy się nie zakończy!

Ten program nigdy się nie zatrzyma!

Pierwsza miara trudnosci

Pierwsza miara trudnosci

Program:

Pierwsza miara trudnosci

Program: Czas przebiegu programu

Pierwsza miara trudnosci

Program: Czas przebiegu programu

Problem:

Pierwsza miara trudnosci

Program: Czas przebiegu programu

Problem: Czas przebiegu najszybszego programu rozwiazujacegp zadany problem

Pierwsza miara trudnosci

Program: Czas przebiegu programu

Problem: Czas przebiegu najszybszego programu rozwiazujacegp zadany problem

o ile taki najszybszy program istneje.

Programy bez pętli

Stosunkowo latwe do napisania

Programy bez pętli

Stosunkowo latwe do napisania, chociaz moga byc bardzo dlugie.

Programy bez pętli

Stosunkowo latwe do napisania, chociaz moga byc bardzo dlugie.

Analiza poprawnosci (wąsko rozumiana) jest stosunkowo prosta.

Programy bez pętli

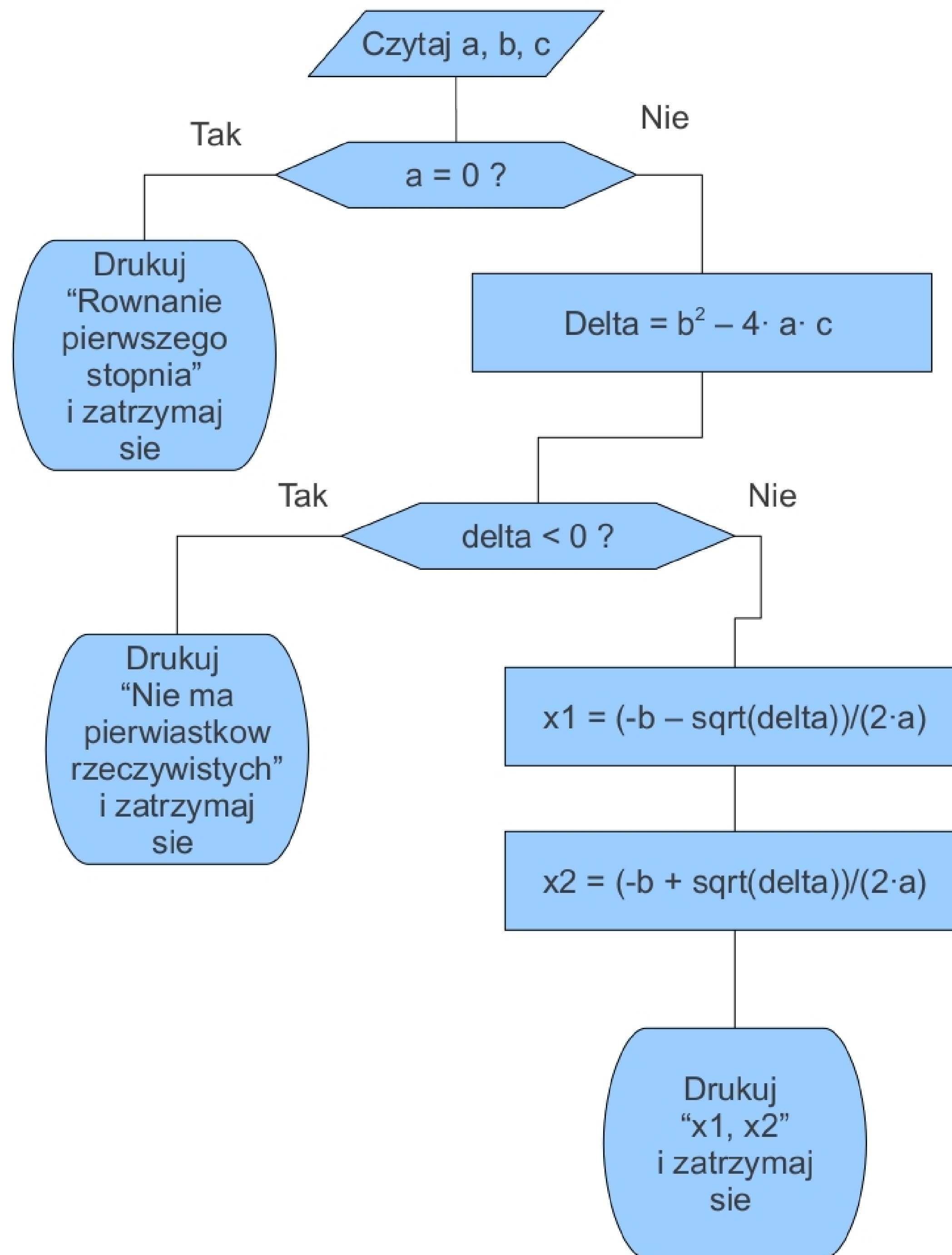
Przyklad:

Programy bez pętli

Przyklad:

Rozwiązać równanie

$$ax^2 + bx + c = 0$$


Czytaj a, b, c
Tak
Nie
a = 0 ?
Drukuj
“Rownanie
pierwszego
stopnia”
i zatrzymaj
sie
Delta = b2 – 4· a· c
Tak
Nie
delta < 0 ?
Drukuj
“Nie ma
pierwiastkow
rzeczywistych”
i zatrzymaj
sie
x1 = (-b – sqrt(delta))/(2·a)
x2 = (-b + sqrt(delta))/(2·a)
Drukuj
“x1, x2”
i zatrzymaj
sie

Programy bez pętli

Czas przebiegu ocenia sie latwo.

Programy bez pętli

Czas przebiegu ocenia sie latwo.

Mozna pomnozyc liczbe wszystkich instrukcji programu przez czas wykonania jednej instrukcji.

Programy bez pętli

Czas przebiegu ocenia sie latwo.

Mozna pomnozyc liczbe wszystkich instrukcji programu przez czas wykonania jednej instrukcji.

Powyzszy program zakonczy sie < 10 µsec

Programy z pętlami

- Czas przebiegu programu z petla moze byc dowolnie dlugi

Programy z pętlami

- Czas przebiegu programu z petla moze byc dowolnie dlugi, nawet nieskonczony.

Programy z pętlami

Czas przebiegu programu z petla moze byc dowolnie dlugi, nawet nieskonczony.

1: Skocz do 1;

Programy z pętlami

Czas przebiegu programu z petla moze byc dowolnie dlugi, nawet nieskonczony.

1: Skocz do 1;

Najlepiej byloby ich w ogole uniknac!

Programy z pętlami

Czas przebiegu programu z petla moze byc dowolnie dlugi, nawet nieskonczony.

1: Skocz do 1;

Najlepiej byloby ich w ogole uniknac!

Ale wtedy nie wiele daloby sie zaprogramowac.

Programy z pętlami

Na przyklad, do napisania skonczonej dlugosci programu testowania pierwszosci liczb naturalnych niezbedna jest petla.

Programy z pętlami

Na przyklad, do napisania skonczonej dlugosci programu testowania pierwszosci liczb naturalnych niezbedna jest petla.

Innymi slowy, test pierwszosci bez petli wymagalby nieskonczonej liczby instrukcji.

Programy z pętlami

Na przyklad, do napisania skonczonej dlugosci programu testowania pierwszosci liczb naturalnych niezbedna jest petla.

Innymi slowy, test pierwszosci bez petli wymagalby nieskonczonej liczby instrukcji.

Taki program nie zmiescilby sie w pamieci.

Programy z pętlami

```
52
53    private boolean isPrime()
54    {
55        for (int i = 2; i <= sqrt(n); i++)
56        {
57            if (n % i == 0) return false;
58        }
59        return true;
60    }
```

Programy z pętlami

```
52
53    private boolean isPrime()
54    {
55        for (int i = 2; i <= sqrt(n); i++)
56        {
57            if (n % i == 0) return false;
58        }
59        return true;
60    }
```

Przebieg tego programu wymaga $< c \times \sqrt{n}$ operacji, gdzie c jest pewna stala.

Programy z pętlami

```
52
53    private boolean isPrime()
54    {
55        for (int i = 2; i <= sqrt(n); i++)
56        {
57            if (n % i == 0) return false;
58        }
59        return true;
60    }
```

Przebieg tego programu wymaga < c × √n operacji, gdzie c jest pewna stala.

A to jest bardzo dlugo (powoli).

Programy z pętlami

Oto szybszy program testowania pierwszosci odkryty stosunkowo niedawno (ca. 2002):

Programy z pętlami

Test pierwszosci AKS (Agrawal–Kayal–Saxena; tlumaczenie z Wikipedii):

1. Czytaj calkowite $n > 1$.
2. Jesli $n = a^b$ dla pewnego calkowitego $a > 0$ and $b > 1$ to drukuj "zlozona" i zatrzymaj.
3. Znajdz najmniejsze r take ze $O_r(n) > (\log_2 n)^2$.
4. Jesli $1 < \mathrm{NWP}(a,n) < n$ dla pewnego calkowitego $a \leq r$ to drukuj "zlozona" i zatrzymaj.
5. Jesli $n \leq r$ to drukuj "pierwsza" i zatrzymaj.
6. Dla $a = 1$ az do $\lfloor \sqrt{\varphi(r)} \log(n) \rfloor$ powtarzaj:

   Jesli $(X+a)^n \neq X^n + a \pmod{X^r - 1, n}$ to drukuj "zlozona" i zatrzymaj.

7. Drukuj "pierwsza" i zatrzymaj.

Czas wykonania nie wiekszy niz $c \times (\log_2 n)^{12}$.

(Petla w instrukcji 6 nigdy sie nie wykonuje do konca.)

Programy z pętlami

Test pierwszosci AKS (Agrawal–Kayal–Saxena; tlumaczenie z Wikipedii):

Oznaczenia:

Dla dowolnej liczby calkowitej n dodatniej liczby calkowitej r takich ze NWP(n, r) = 1, multyplikatywny rzad n modulo r jest to najmniejsza dodatnia liczba calkowita k taka, ze

$$a^k \equiv 1 \pmod{n}.$$

Rzad n modulp r oznacza sie przez $O_r(n)$.

Funkcja Eulera: Dla dowolnej dodatniej liczby calkowitej n, $\varphi(n)$ jest to liczba tych dodatnich liczb calkowitych $r \leq n$ dla ktorych NWP(n, r) = 1.

Programy z pętlami

Przebieg programu AKS wymaga $< c \times (\log_2 n)^{12}$ operacji, gdzie c jest pewna stala.

Programy z pętlami

Przebiebieg programu AKS wymaga $< c \times (\log_2 n)^{12}$ operacji, gdzie c jest pewna stala.

I to jest poprawa w stosunku do poprzedniego testu pierwszosci.

Hierarchia czasow przebiegu

Hierarchia czasow przebiegu

Czas T przebiegu programu jest funkcja rzeczywista wielkosci m jego danych na wejsciu.

Hierarchia czasow przebiegu

Czas T przebiegu programu jest funkcja rzeczywista wielkosci m jego danych na wejsciu.

$$T: N \rightarrow R^+$$

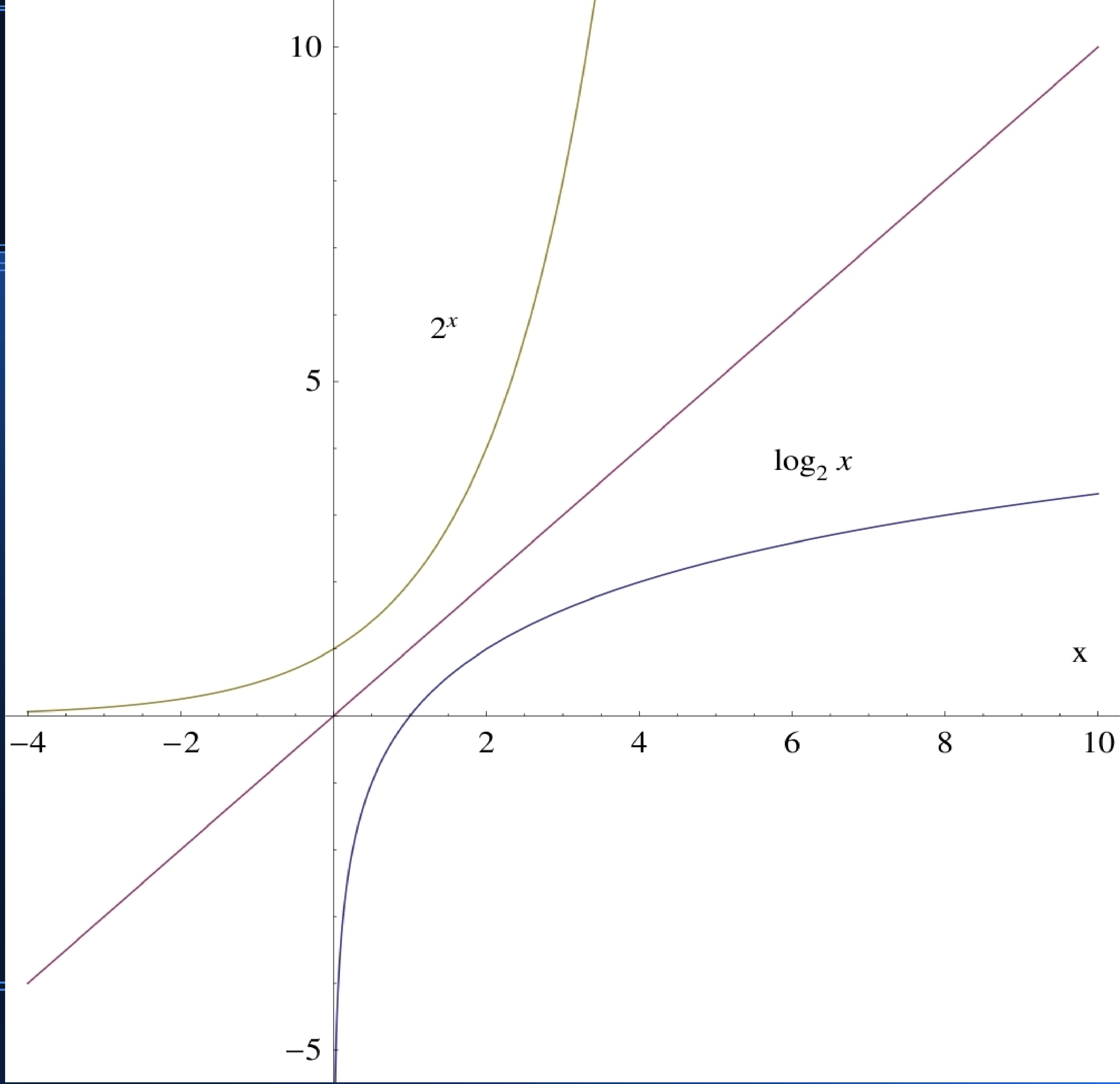
Hierarchia czasow przebiegu

Czas T przebiegu programu jest funkcja rzeczywista wielkosci m jego danych na wejsciu.

$$T: N \rightarrow R^{+}$$

Jesli jest jedna dana n na wejsciu to jej wielkosc jest (w przyblizeniu) proporcjonalna do:

$$m = \log_2 n$$



10
5
−5
2^x
$\log_2 x$
x
−4
−2
2
4
6
8
10

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

T(m) =

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n)$

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$$T(m) = T(\log_2 n) = c \times \sqrt{n}$$

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$$T(m) = T(\log_2 n) = c \times \sqrt{n} = c \times \sqrt{2^m}$$

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times \sqrt{n} = c \times \sqrt{2^m} = c \times 2^{m/2}$

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times \sqrt{n} = c \times \sqrt{2^m} = c \times 2^{m/2}$

AKS test pierwszosci mial czas przebiegu

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times \sqrt{n} = c \times \sqrt{2^m} = c \times 2^{m/2}$

AKS test pierwszosci mial czas przebiegu

T(m)

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times \sqrt{n} = c \times \sqrt{2^m} = c \times 2^{m/2}$

AKS test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n)$

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times \sqrt{n} = c \times \sqrt{2^m} = c \times 2^{m/2}$

AKS test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times (\log_2 n)^{12}$

Hierarchia czasow przebiegu

Pierwszy test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times \sqrt{n} = c \times \sqrt{2^m} = c \times 2^{m/2}$

AKS test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times (\log_2 n)^{12} = c \times m^{12}$

Hierarchia czasow przebiegu

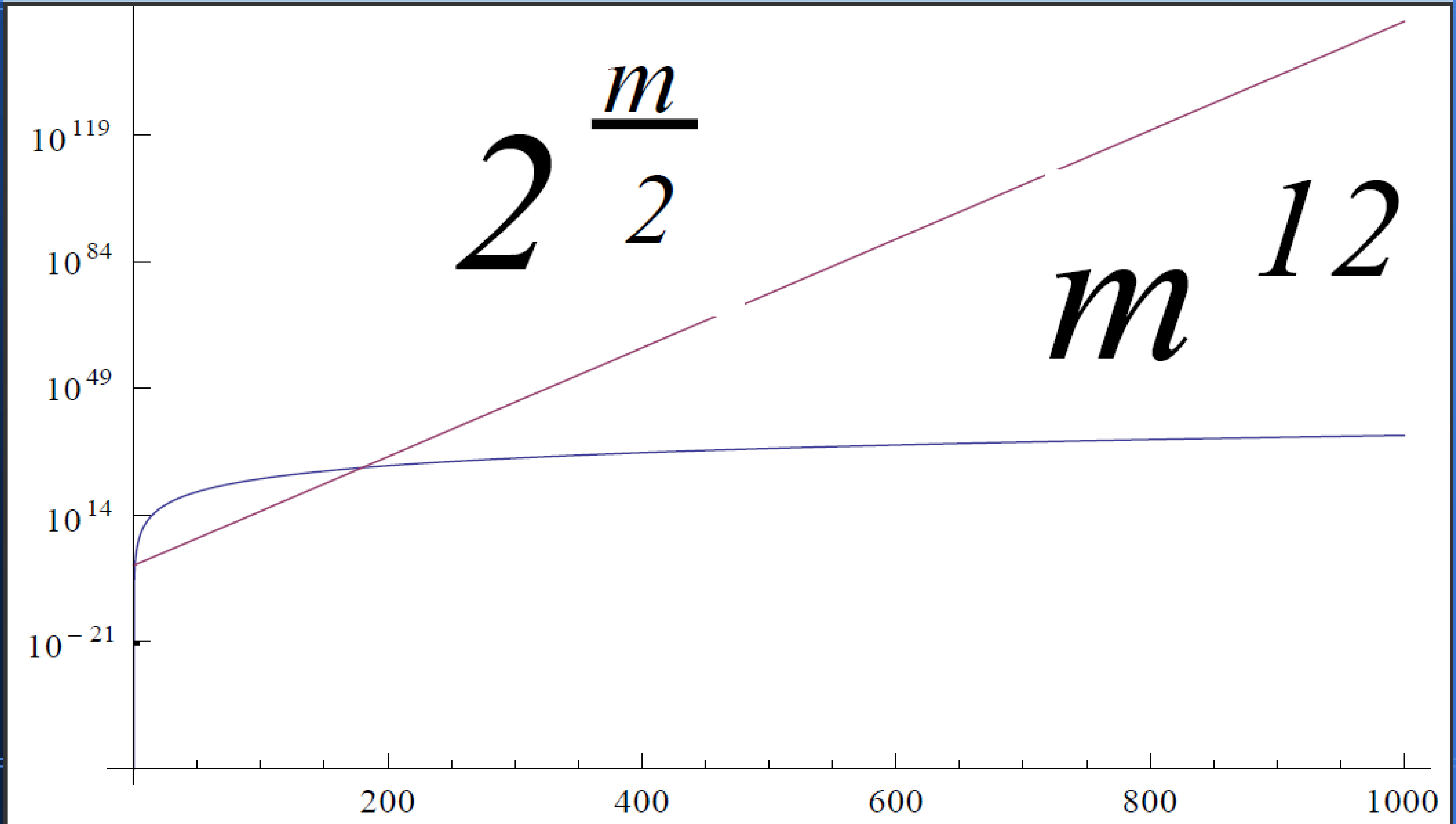
Pierwszy test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times \sqrt{n} = c \times \sqrt{2^m} = c \times \mathbf{2^{m/2}}$

AKS test pierwszosci mial czas przebiegu

$T(m) = T(\log_2 n) = c \times (\log_2 n)^{12} = c \times \mathbf{m^{12}}$

Hierarchia czasow przebiegu



Hierarchia czasow przebiegu

Funkcja

$$2^{m/2}$$

rosnie do nieskonczonosci znacznie szybciej niz funkcja

$$m^{12}.$$

Hierarchia czasow przebiegu

A zatem

nasz pierwszy test pierwszosci

jest znacznie wolniejszy niz

test AKS.

Hierarchia czasow przebiegu

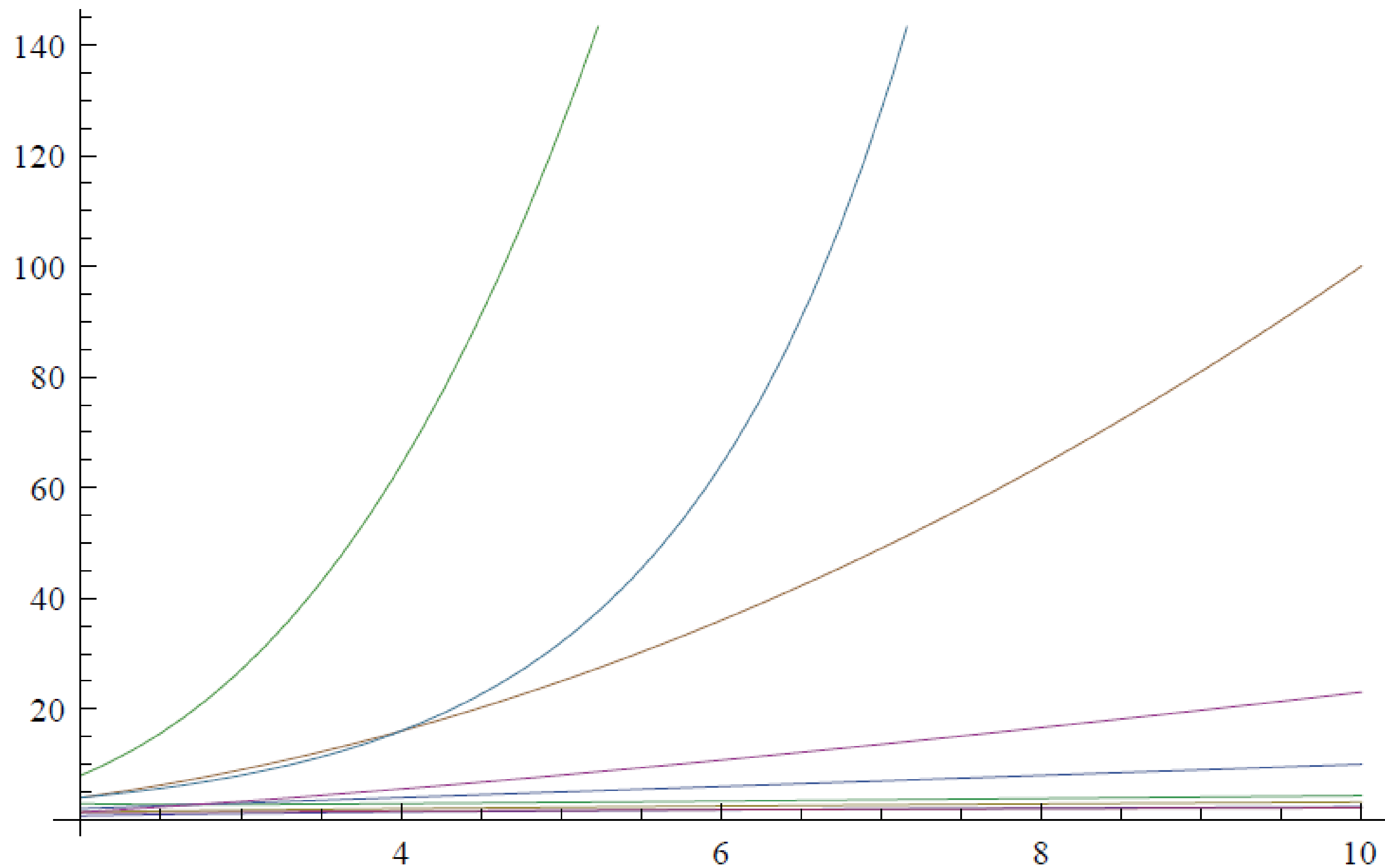
Klasa P: Problemy ktorych rozwiazanie wymaga czasu co najwyzej wielomianowego w funkcji wielkosci danych wejsciowych.

Hierarchia czasow przebiegu

Klasa **P**: Problemy ktorych rozwiazanie wymaga czasu co najwyzej wieloomianowego w funkcji wielkosci danych wejsciowych.

Problem rozstrzygania pierwszosci jest zatem w klasie **P**.

Hierarchia czasow przebiegu



Hierarchia czasow przebiegu

Klasa funkcji pierwotnie rekurencyjnych Prek.

Hierarchia czasow przebiegu

Klasa funkcji pierwotnie rekurencyjnych Prek.

Wszystkie funkcje obliczalne przez programy ktorych jedynymi petlami sa petle for.

Hierarchia czasow przebiegu

Klasa funkcji pierwotnie rekurencyjnych Prek.

Wszystkie funkcje obliczalne przez programy ktorych jedynymi petlami sa petle for.

Przebiegi takich programow zawsze sie koncza

Hierarchia czasow przebiegu

Klasa funkcji pierwotnie rekurencyjnych Prek.

Wszystkie funkcje obliczalne przez programy ktorych jedynymi petlami sa petle for.

Przebiegi takich programow zawsze sie koncza – nie ma tam petli nieskonczonych.

Hierarchia czasow przebiegu

Klasa funkcji pierwotnie rekurencyjnych Prek.

Wszystkie funkcje obliczalne przez programy ktorych jedynymi petlami sa petle for.

Przebiegi takich programow zawsze sie koncza – nie ma tam petli nieskonczonych.

Nie ma tu 1: Skocz do 1;

Hierarchia czasow przebiegu

Klasa funkcji rekurencyjnych Rek.

Hierarchia czasow przebiegu

Klasa funkcji rekurencyjnych Rek.

Wszystkie funkcje obliczalne przez programy ktore nie wpadaja w petle nieskonczone.

Petle for tu nie wystarczaja.

Funkcja Ackermanna

Funkcja Ackermanna

$A(1, n, k) = n + k$

Funkcja Ackermanna

$A(1, n, k) = n + k = n + 1 + \ldots + 1$

Funkcja Ackermanna

$A(1, n, k) = n + k = n + 1 + \ldots + 1$

$A(2, n, k) = n \times k$

Funkcja Ackermanna

$A(1, n, k) = n + k = n + 1 + \ldots + 1$

$A(2, n, k) = n \times k = n + n + \ldots + n$

Funkcja Ackermanna

$A(1, n, k) = n + k = n + 1 + \ldots + 1$

$A(2, n, k) = n \times k = n + n + \ldots + n$

$A(3, n, k) = n^k$

. . .

Funkcja Ackermanna

$A(1, n, k) = n + k = n + 1 + \ldots + 1$

$A(2, n, k) = n \times k = n + n + \ldots + n$

$A(3, n, k) = n^k = n \times n \times \ldots \times n$

. . .

Funkcja Ackermanna

$A(1, n, k) = n + k = n + 1 + \ldots + 1$

$A(2, n, k) = n \times k = n + n + \ldots + n$

$A(3, n, k) = n^k = n \times n \times \ldots \times n$

$A(4, n, k) = n^{n^{\cdots^{n}}}$

Funkcja Ackermanna

$A(1, n, k) = n + k = n + 1 + \ldots + 1$

$A(2, n, k) = n \times k = n + n + \ldots + n$

$A(3, n, k) = n^k = n \times n \times \ldots \times n$

$A(4, n, k) = n^{n^{\cdots^{n}}}$

. . .

Funkcja Ackermanna

Zdefiniujmy

$$B(n) = A(n, n, n)$$

Funkcja Ackermanna

Zdefiniujmy

$$B(n) = A(n, n, n)$$

$B(1) = 1 + 1 = 2$

Funkcja Ackermanna

Zdefiniujmy

$$B(n) = A(n, n, n)$$

$B(1) = 1 + 1 = 2$

$B(2) = 2 \times 2 = 4$

Funkcja Ackermanna

Zdefiniujmy

$$B(n) = A(n, n, n)$$

$B(1) = 1 + 1 = 2$

$B(2) = 2 \times 2 = 4$

$B(3) = 3^3 = 27$

Funkcja Ackermanna

Zdefiniujmy

$$B(n) = A(n, n, n)$$

$B(1) = 1 + 1 = 2$

$B(2) = 2 \times 2 = 4$

$B(3) = 3^3 = 27$

$B(4) = 4^{4^{4^{4}}}$

Funkcja Ackermanna

Zdefiniujmy

$$B(n) = A(n, n, n)$$

$B(1) = 1 + 1 = 2$

$B(2) = 2 \times 2 = 4$

$B(3) = 3^3 = 27$

$B(4) = 4^{4^{4^4}} = 4^{4^{256}} =$

Funkcja Ackermanna

Zdefiniujmy

$$B(n) = A(n, n, n)$$

$B(1) = 1 + 1 = 2$

$B(2) = 2 \times 2 = 4$

$B(3) = 3^3 = 27$

$B(4) = 4^{4^{4^4}} = 4^{4^{256}} = \ldots$ HUGE !!!

Funkcja Ackermanna

$\log_{10} B(4) =$

Funkcja Ackermanna

$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}}$

Funkcja Ackermanna

$$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}} = \log_{10} 4^{4^{256}} =$$

Funkcja Ackermanna

$$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}} = \log_{10} 4^{4^{256}} =$$

$$4^{256} \log_{10} 4 =$$

Funkcja Ackermanna

$$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}} = \log_{10} 4^{4^{256}} =$$

$$4^{256} \log_{10} 4 = 2^{512} \log_{10} 4$$

Funkcja Ackermanna

$$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}} = \log_{10} 4^{4^{256}} =$$

$$4^{256} \log_{10} 4 = 2^{512} \log_{10} 4 = 0.3 \times 2^{512}$$

Funkcja Ackermanna

$$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}} = \log_{10} 4^{4^{256}} =$$

$$4^{256} \log_{10} 4 = 2^{512} \log_{10} 4 = 0.3 \times 2^{512} = 0.3 \times 10^{154}$$

Funkcja Ackermanna

$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}} = \log_{10} 4^{4^{256}} =$

$4^{256} \log_{10} 4 = 2^{512} \log_{10} 4 = 0.3 \times 2^{512} =$ **0.3×10^{154}**

Funkcja Ackermanna

$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}} = \log_{10} 4^{4^{256}} =$

$4^{256} \log_{10} 4 = 2^{512} \log_{10} 4 = 0.3 \times 2^{512} =$ **0.3×10^{154}**

Ale to jest **liczba cyfr** B(4)

Funkcja Ackermanna

$\log_{10} B(4) = \log_{10} 4^{4^{4^{4}}} = \log_{10} 4^{4^{256}} =$

$4^{256} \log_{10} 4 = 2^{512} \log_{10} 4 = 0.3 \times 2^{512} =$ **0.3×10^{154}**

Ale to jest **liczba cyfr** B(4)

Nie ma tylu atomow w calym wszechswiecie!

Funkcja Ackermanna

A ile to bedzie $B(4^{4^{4^{4}}})$?

Funkcja Ackermanna

Program obliczajacy funkcje Ackermanna jest zdumiewajaco prosty!

Funkcja Ackermanna

```
40        public static long A(long k, long m, long n)
41        {
42                if (n == 0) return m + k;
43                else
44                {
45                        if (m == 0)
46                        {
47                                if (n == 1) return 0;
48                                else
49                                {
50                                     if (n == 2) return 1;
51                                     else return k;
52                                }
53                        }
54                        else return A(k, A(k, m-1, n), n-1);
55                }
56        }
```

Funkcja Ackermanna

Ale zaden program obliczajacy funkcje Ackermanna nie moze byc skonstruowany w taki sposob aby wszystkie jego petle byly postaci **for**.

Funkcja Ackermanna

Ale zaden program obliczajacy funkcje Ackermanna nie moze byc skonstruowany w taki sposob aby wszystkie jego petle byly postaci **for**.

W szczegolnosci, kazdy program obliczajacy funkcje Ackermanna jest powolniejszy niz najwolniejszy program ktorego wszystkie jego petle sa postaci **for**.

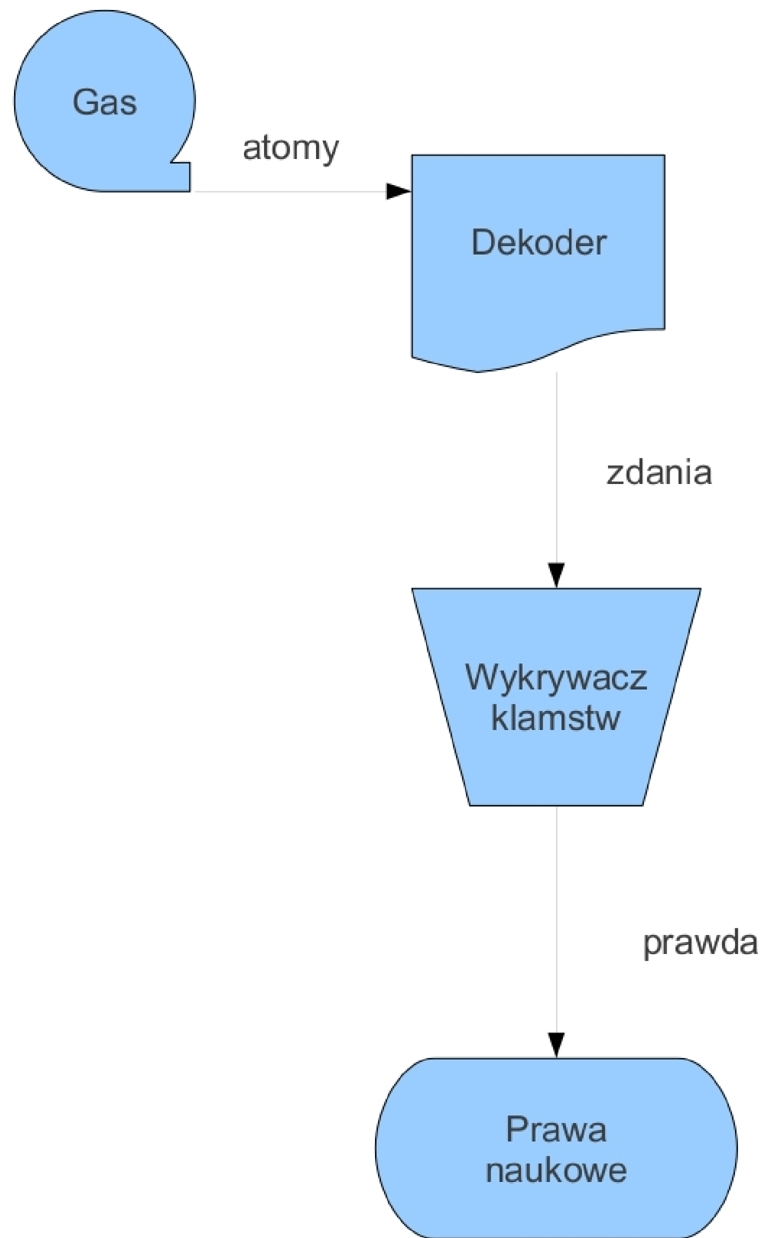
Funkcja Ackermanna

Funkcja Ackermanna jest przykladem ograniczenia na (praktyczna) obliczalnosc za pomoca programu.

Próba automatyzacji programowania

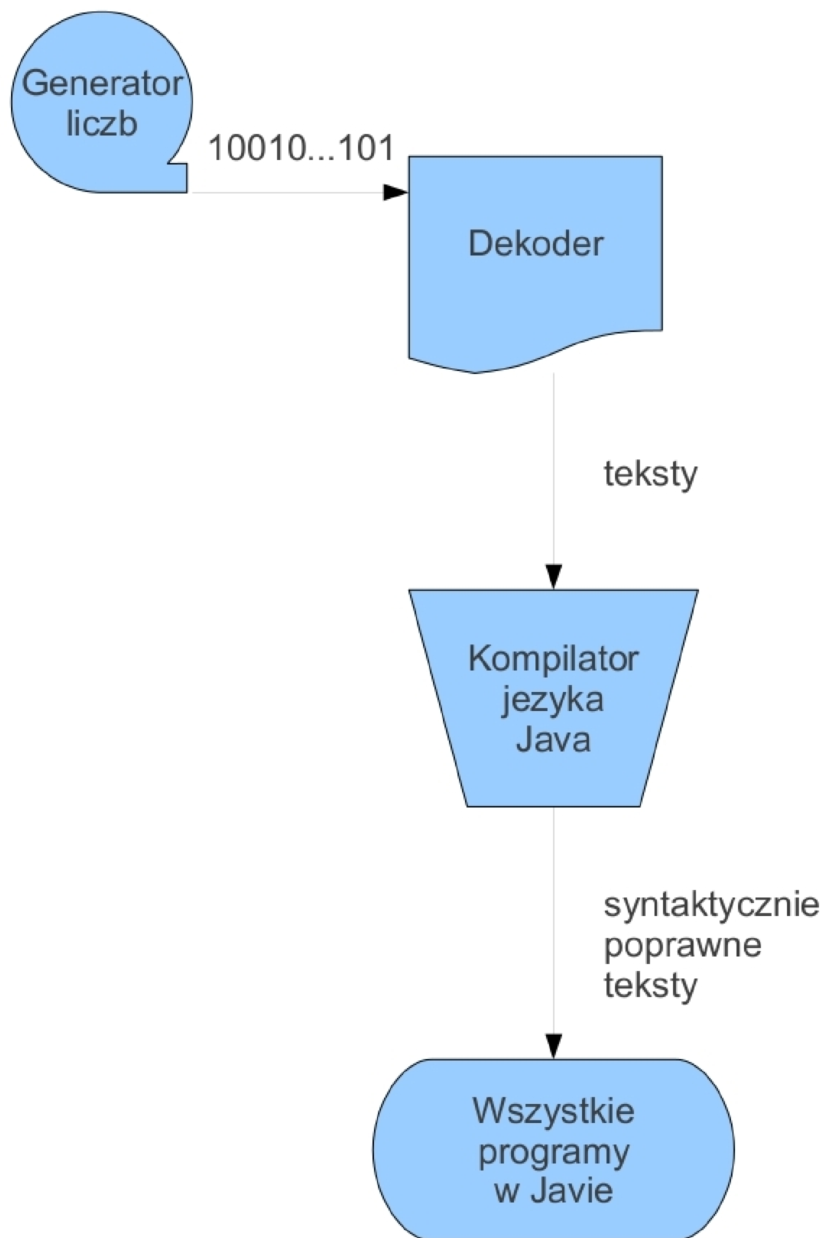
Próba automatyzacji programowania

A oto pomysl Trurla i Klapaucjusza na odkrywanie wszelkiej prawdy naukowej wziety z “Cyberiady” Stanislawa Lema:


Gas
atomy
Dekoder
zdania
Wykrywacz
klamstw
prawda
Prawa
naukowe

Próba automatyzacji programowania

Podobnie mozna by zorganizowac automatyczne konstruowanie wszystkich syntaktycznie poprawnych programow w Javie:


Generator
liczb
10010...101
Dekoder
teksty
Kompilator
jezyka
Java
syntaktycznie
poprawne
teksty
Wszystkie
programy
w Javie

Próba automatyzacji programowania

Ten schemat dziala.

Próba automatyzacji programowania

Ten schemat dziala.

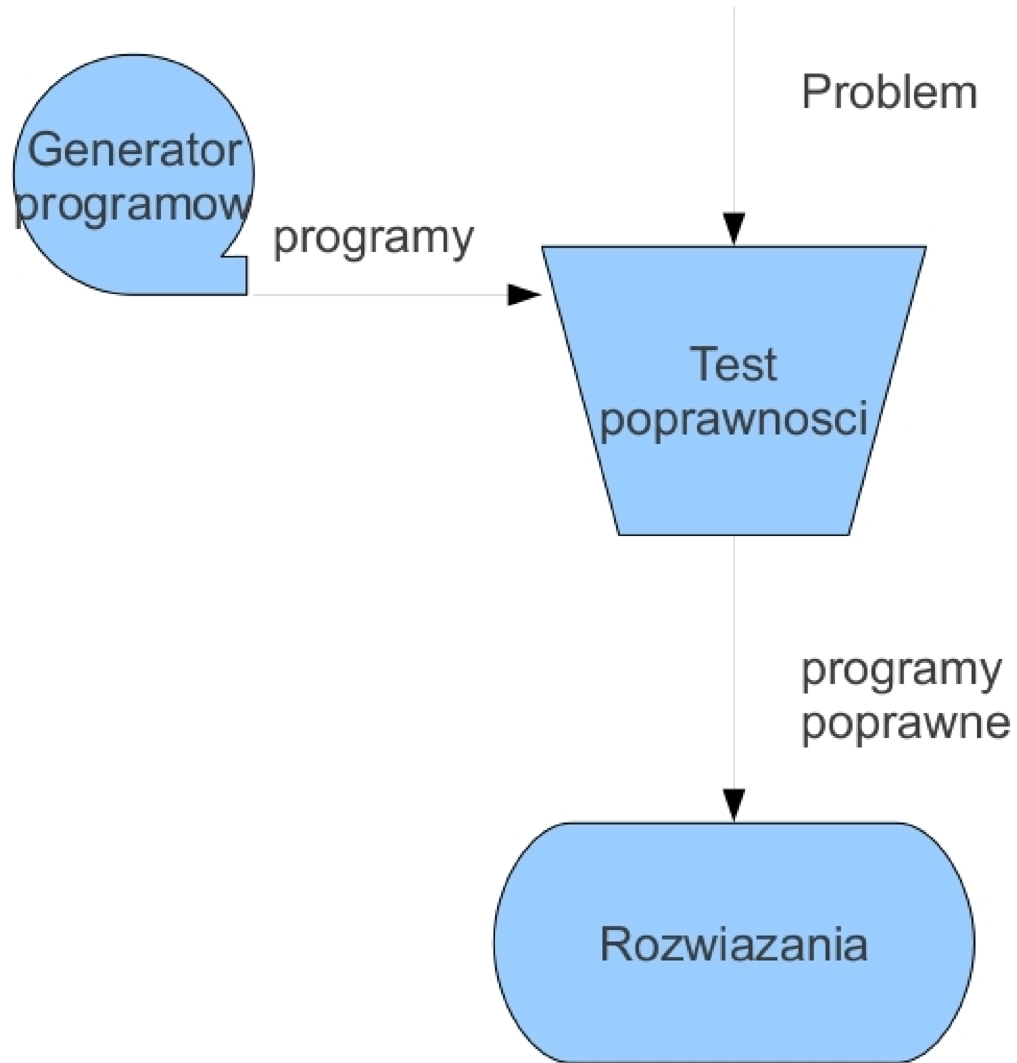
Moze wiec da sie go wzmocnic tak, zeby produkowal on wszystkie rozwiazania zadanego problemu?

Próba automatyzacji programowania

Ten schemat dziala.

Moze wiec da sie go wzmocnic tak, zeby produkowal on wszystkie rozwiazania zadanego problemu?

Sprobujmy:


Problem
Generator programow
programy
Test poprawnosci
programy poprawne
Rozwiazania

Próba automatyzacji programowania

Pierwsza trudnosc:

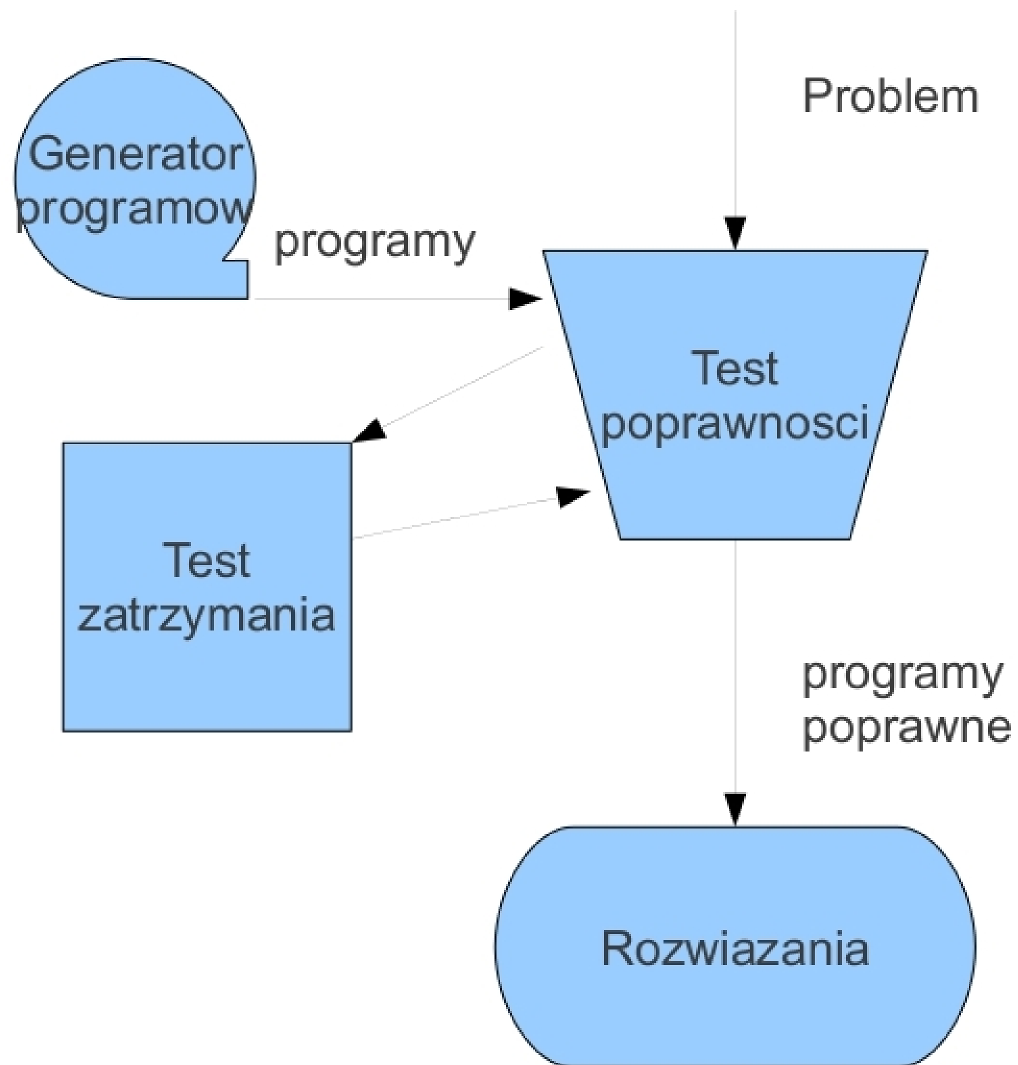
Nalezy wykluczyc the syntaktyczne programy ktore
wpadaja w nieskonczone petle.

Próba automatyzacji programowania

Pierwsza trudnosc:

Nalezy wykluczyc the syntaktyczne programy ktore wpadaja w nieskonczone petle.

Potrzebujemy zatem skonstruowac test zatrzymywania jako czesc calego systemu.

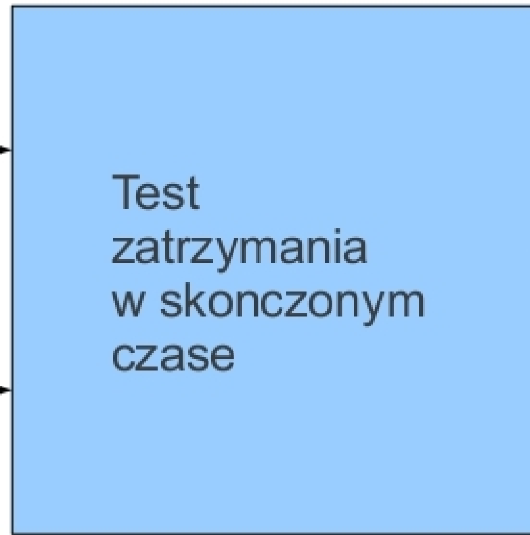

Problem
Generator programow
programy
Test poprawnosci
Test zatrzymania
programy poprawne
Rozwiazania

Próba automatyzacji programowania

Taki podsystem wygladalby nastepujaco:

Program

Dane do
programu



Zatrzymal sie

Zapetlil sie

Testowanie zatrzymywania

Skonstruowanie go przekracza nasze mozliwosci!

Testowanie zatrzymywania

Skonstruowanie go przekracza nasze mozliwosci!

Na przyklad, przez ostatnie 80 lat nie udalo sie nikomu rozstrzygnac, czy ponizszy pozornie prosty program moze wpasc w nieskonczona petle czy nie!

Testowanie zatrzymywania

```
public static int f(int n)
{
    if (n <= 1) return 1;
    if (n % 2 == 0) return (f(n/2) + 1);
            else return (f(3*n + 1) + 1);
}
```

Testowanie zatrzymywania

Testowanie zatrzymywania

Tak wiec testowanie zatrzymywania jest w ogolnosci **niewykonalne**.

Testowanie zatrzymywania

Tak wiec testowanie zatrzymywania jest w ogolnosci **niewykonalne**.

Nie istnieje program ktory by zawsze w skonczonej liczbie krokow poprawnie rozsitrzygal czy zadany inny program wpadnie w nieskonczona petle na zadanym wejsciu.

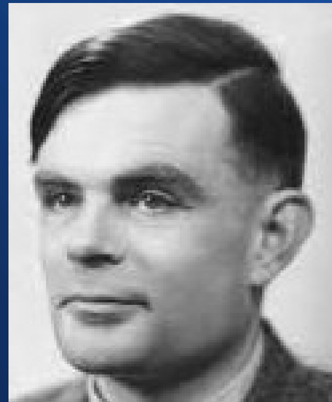
Testowanie zatrzymywania

Tak wiec testowanie zatrzymywania jest w ogolnosci **niewykonalne**.

Nie istnieje program ktory by zawsze w skonczonej liczbie krokow poprawnie rozsitrzygal czy zadany inny program wpadnie w nieskonczona petle na zadanym wejsciu.

[Alan Turing, ca. 1930]

Testowanie zatrzymywania



Próba automatyzacji programowania

Nie da sie tego zrobic!

Próba automatyzacji programowania

Nie da się tego zrobic!

Pętle to wszystko popsuly.

Próba automatyzacji programowania

Nie da się tego zrobic!

Pętle to wszystko popsuly.

Ale są jeszcze inne przeszkody.

Próba automatyzacji programowania

Dla dowolnej funkcji rekurencyjnej F, nie istnieje program ktory by poprawnie rozpoznawal wszystkie te i tylko the programy ktore obliczja F.

Próba automatyzacji programowania

Dla dowolnej funkcji rekurencyjnej F, nie istnieje program ktory by poprawnie rozpoznawal wszystkie te i tylko the programy ktore obliczja F.

Na przyklad, nie istnieje program poprawnie rozpoznajacy programy obliczajace

$$F(n) = 2 \times n$$

Próba automatyzacji programowania

- A zatem nawet jesli uda sie nam mapisac program rozwiazujacy poprawnie zadany problem, na ogol nie bedziemy potrafili sie o tym przekonac.

Koronny przyklad

- Problem: Czy Ziemia I Ksiezyc kiedys wpadna na siebie?

Koronny przyklad

Problem: Czy Ziemia I Ksiezyc kiedys wpadna na siebie?

Program A 1: Drukuj “Tak” I zatrzymaj sie.

Koronny przyklad

Problem: Czy Ziemia I Ksiezyc kiedys wpadna na siebie?

Program A 1: Drukuj “Tak” I zatrzymaj sie.

Program B 1: Drukuj “Nie” I zatrzymaj sie.

Koronny przyklad

Problem: Czy Ziemia I Ksiezyc kiedys wpadna na siebie?

Program A 1: Drukuj “Tak” I zatrzymaj sie.

Program B 1: Drukuj “Nie” I zatrzymaj sie.

A lub B jest rozwiazaniem, ale nie wiemy ktory.

Niemozliwosc

Na dodatek zlego, pewnych problemow w ogole nie da sie rozwiazac przy pomocy programow komputerowych.

Niemozliwosc

Na dodatek zlego, pewnych problemow w ogole nie da sie rozwiazac przy pomocy programow komputerowych.

Przykladami byly:

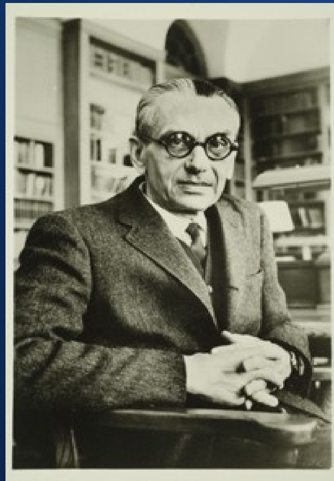
- rozstrzyganie zatrymywania
- rozstrzyganie poprawnosci

Niemozliwosc

- Kurt Gödel wymyslil jeszcze bardziej nierozstrzygalny problem ktorego uzyl w dowodzie twierdzenia o nieaksjomatyzowalnosci arytmetyki.

Niemozliwosc

- Kurt Gödel wymyslil jeszcze bardziej nierozstrzygalny problem ktorego uzyl w dowodzie twierdzenia o nieaksjomatyzowalnosci arytmetyki.



Niemozliwosc

Profesor Marchewka nie moze udowodnic tego zdania bez zaprzeczania sobie.

Niemozliwosc

Profesor Marchewka nie moze udowodnic tego zdania bez zaprzeczania sobie.

Udowodnimy, ze powyzsze zdanie jest prawdziwe.

Niemozliwosc

Profesor Marchewka nie moze udowodnic tego zdania bez zaprzeczania sobie.

Udowodnimy, ze powyzsze zdanie jest prawdziwe.

Zalozmy ze jest ono falszywe.

Niemozliwosc

Profesor Marchewka nie moze udowodnic tego zdania bez zaprzeczania sobie.

Udowodnimy, ze powyzsze zdanie jest prawdziwe.

Zalozmy ze jest ono falszywe.

Ale wtedy jego negacja jest prawdziwa.

Niemozliwosc

Zatem prawda jest, ze

Profesor Marchewka moze udowodnic to zdanie bez zaprzeczania sobie.

Ale skoro tak to do zdanie jest prawdiwe (bo ma dowod) – przeciwnie do naszego zalozenia ze jest ono falszywe.

Niemozliwosc

Zatem prawda jest, ze

Profesor Marchewka moze udowodnic to zdanie bez zaprzeczania sobie.

Ale skoro tak to do zdanie jest prawdiwe (bo ma dowod) – przeciwnie do naszego zalozenia ze jest ono falszywe. **Sprzecznosc!**

Niemozliwosc

Zatem prawda jest, ze

Profesor Marchewka moze udowodnic to zdanie bez zaprzeczania sobie.

Ale skoro tak to do zdanie jest prawdiwe (bo ma dowod) – przeciwnie do naszego zalozenia ze jest ono falszywe. **Sprzecznosc!**

To konczy dowod prawdziwosci zdania.

Niemozliwosc

Czyli w istocie,

Profesor Marchewka nie moze udowodnic tego zdania bez zaprzeczania sobie.

Ale kazdy z Was potrafi to zdanie udowodnic bez zaprzeczania sobie.

Niemozliwosc

Czyli w istocie,

Profesor Marchewka nie moze udowodnic tego zdania bez zaprzeczania sobie.

Ale kazdy z Was potrafi to zdanie udowodnic bez zaprzeczania sobie.

Stad powazne ograniczenie rozwiazywalnosci.

Podsumowanie

Podsumowanie

1. Petle ponosza cala wine za to ze programowanie jest trudne.

Podsumowanie

1. Petle ponosza cala wine za to ze programowanie jest trudne.
2. Szczegolnie winne sa te wszystke petle ktore nie sprowadzaja sie do petli for.

Podsumowanie

1. Petle ponosza cala wine za to ze programowanie jest trudne.
2. Szczegolnie winne sa te wszystke petle ktore nie sprowadzaja sie do petli for.
3. Dwa elementy uniemozliwiajace automatyzacje programowania to:

Podsumowanie

1. Petle ponosza cala wine za to ze programowanie jest trudne.
2. Szczegolnie winne sa te wszystke petle ktore nie sprowadzaja sie do petli for.
3. Dwa elementy uniemozliwiajace automatyzacje programowania to:
   a. nie dajace sie poprawnie rozpoznac nieskonczone petle

Podsumowanie

1. Petle ponosza cala wine za to ze programowanie jest trudne.
2. Szczegolnie winne sa te wszystke petle ktore nie sprowadzaja sie do petli for.
3. Dwa elementy uniemozliwiajace automatyzacje programowania to:
   a. nie dajace sie poprawnie rozpoznac nieskonczone petle
   b. niewyobrazalnie duza zlozonosc obliczeniowa pewnych funkcji.

Podsumowanie

Teraz juz wiecie dlaczego programowanie jest takie trudne.

Podsumowanie

I dlatego jest ono takie interesujace.