Piotr Chrząstowski-Wachtel
Uniwersytet Warszawski

Podstawowe pojęcia algorytmiki

Algorytmika

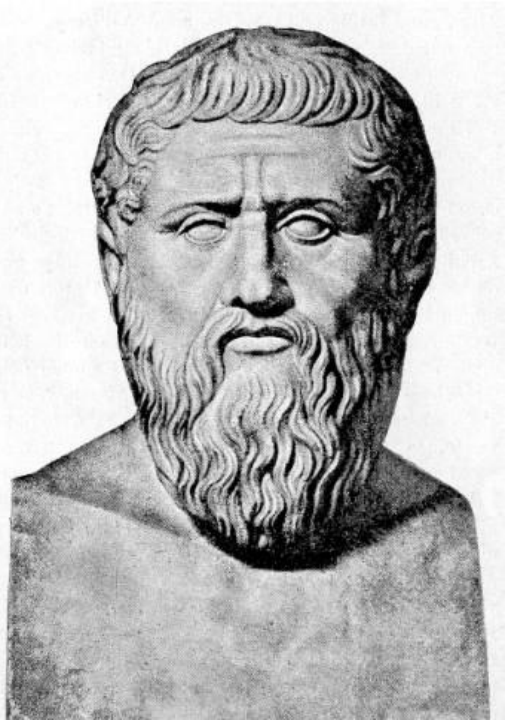
- Najważniejsza część informatyki
- Opisuje jak rozwiązywać problemy algorytmiczne, jakie struktury danych dobierać, jak analizować zachowanie się programów.
- Pozwala na osiągnięcie znacznie bardziej spektakularnych wyników, niż samo przyspieszanie działania sprzętu

Czego dotyczy algorytmika?

- Wszelkiego planowania działań – w szczególności przy pisaniu programów komputerowych
- Musimy pamiętać, że komputerom trzeba niezwykle wyraźnie wyspecyfikować polecenia – będąc dość głupimi urządzeniami nie domyślą się, o co nam mogło chodzić, jeśli nieprecyzyjnie przedstawimy o co nam chodzi.

Kiedy zaczęła się algorytmika?

- Pierwszymi wielkimi algoryt-mikami byli Starożytni Grecy
- Pierwszymi wielkimi naukowymi problemami algorytmicznymi były konstrukcje goemetryczne, zwane platońskimi



Platon

Konstrukcje platońskie

- Nieformalnie chodzi o to, żeby wyznaczać pewne obiekty na płaszczyźnie (punkty, okręgi, proste) spełniające dane założenia.
- Przykładowe zadanie:
  - Mając dany okrąg $o(O,r)$ oraz punkt A leżący poza okręgiem, poprowadzić prostą styczną do danego okręgu, przechodzącą przez punkt A

Czy poprawne jest takie rozwiązanie:

- Wbijamy nóżkę cyrkla w punkt A i opierając na ostrzu linijkę obracamy ją, aż się ukaże punkt okręgu.
- Rysujemy linię łączącą te dwa punkty.

Platon zabraniał takich operacji

- ... jak i wielu innych rzeczy takich jak kreślenie paraboli, spirali, wychodzenie w trzeci wymiar itd.
- Co zatem wolno było robić i na jakich obiektach?

Dziedzina operacji platońskich

- Koncentrujemy się na 3 rodzajach obiektów: punktach, prostych i okręgach
- Wolno na tych obiektach przeprowadzać jedną z pięciu operacji.

Operacje platońskie

- Dla danych dwóch punktów narysować prostą przez nie przechodzącą,
- Dla danych dwóch punktów wykreślić okrąg o środku w jednym z nich i promieniu równym odległości między nimi,
- Dla dwóch prostych wyznaczyć punkt ich przecięcia (o ile istnieje),
- Dla prostej i okręgu wyznaczyć ich punkty przecięcia,
- Dla dwóch okręgów wyznaczyć punkty ich przecięcia.
- ... i nic ponadto!

Dozwolone operacje

- Nazwijmy nasze operacje odpowiednio
  - l := line(X,Y) – prosta przechodząca przez X i Y
  - o:= circle(O,Y) – okrąg o środku O i promieniu OY
  - X := l $\times$ k – punkt przecięcia prostych l i k
  - (X,Y) := l $\varnothing$ o – punkty przecięcia prostej l i okręgu o
  - (X,Y) := o1 ∞ o2 – punkty przecięcia okręgów o1 i o2
- Wszystkie te operacje są *częściowe*: są określone nie dla wszystkich argumentów

Rozwiązanie zadania

- Możemy przedstawić rozwiązanie w postaci sekwencji czynności dla okręgu o(O,Y) oraz punktu A leżącego poza nim:
  - l := line(O,A) – kreślimy prostą l łączącą środek okręgu z punktem A
  - o1:= circle(O,A) – kreślimy okrąg o środku O i promieniu OA
  - o2:= circle(A,O) – kreślimy okrąg o środku A i promieniu O
  - (P,Q) := o1 ∞ o2 – wyznaczamy punkty przecięcia okręgów o1 i o2
  - k := line(P,Q) – prowadzimy symetralną odcinka OA
  - X := l × k – znajdujemy środek odcinka OA
  - o3 := circle (X,O) – kreślimy okrąg o środku X i promieniu XO
  - (R,S) := o ∞ o3 – wyznaczamy punkty przecięcia okręgów o i o3
  - s := line(R,A) – prosta s jest jedną z dwóch poszukiwanych stycznych

Rozwiązanie zadania – wersja kompaktowa

- Można krócej:
  - s := line((o ∞_1 (circle((line(O,Y) $\times$ line(circle(A,O)) ∞ circle(O,A)),O))),A)
    - *tutaj przez ∞_1 rozumiemy pierwszy z dwóch punktów przecięcia*
- ... i tak mniej więcej wygląda *programowanie funkcyjne*

Problemy nierozwiązywalne

- Starożytni Grecy nie umieli sobie poradzić z trzema konstrukcjami:
  - wyznaczeniem boku kwadratu o polu równym polu koła o promieniu 1 (kwadratura koła)
  - podziałem dowolnego kąta na 3 równe części (trysekcja kąta)
  - wyznaczeniem boku sześcianu o dwukrotnie większej objętości niż sześcian jednostkowy (podwojenie sześcianu)

Nierozwiązywalność niektórych zadań konstrukcyjnych

- Dopiero w XIX wieku pokazano, że żadnej z tych trzech konstrukcji nie da się wykonać.
- Być może powodem jest zbyt wąski repertuar środków?
- Ale czy gdy dorzucimy parę innych operacji, to czy nie znajdą się nowe niewykonywalne konstrukcje?

Problemy nierozwiązywalne

- Dużo później, w XX wieku, Alan Turing pokazał, że istnieją problemy algorytmiczne, których nie da się rozwiązać w żadnej dziedzinie algorytmicznej. To był jeden z najciekawszych wyników w historii informatyki i to uzyskany jeszcze przed powstaniem komputerów (lata 30-te XX wieku).

Problem odpowiedniości Posta


Emil Post

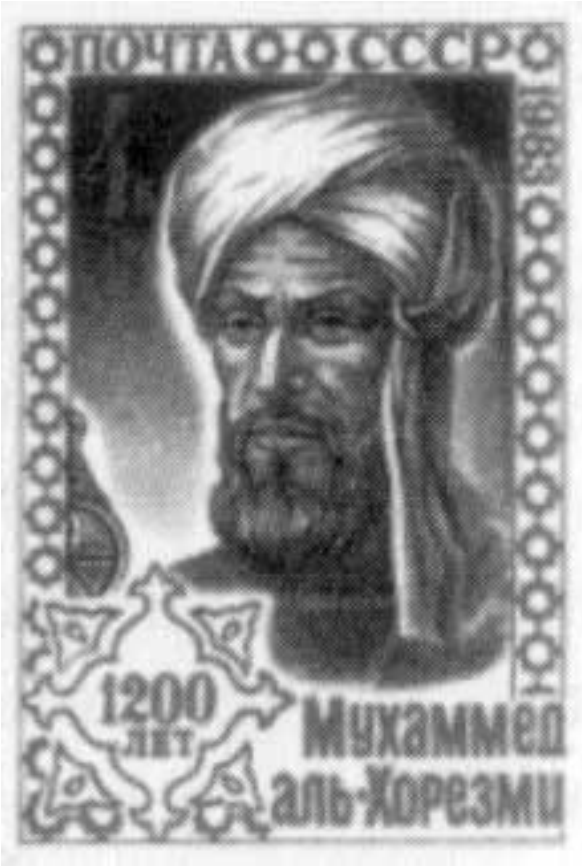
- Przykład:
  - x_1=abb y_1=a
  - x_2=b y_2=abb
  - x_3=a y_3=bb
- Czy istnieje taki ciąg indeksów i1,i2,…,in, że $x_{i1}…x_{in}=y_{i1}…y_{in}$?
- Problem odpowiedniości Posta jest w ogólnym przypadku nierozstrzygalny! Choć dla niektórych przypadków (np. dla powyższego) można podać odpowiedź, nie ma jednak ogólnego algorytmu, który dla dowolnych danych $x_1,...,x_n$ i $y_1,...,y_n$ stwierdziłby, czy można wyrównać odpowiednie słowa x-owe i y-owe za pomocą tego samego ciągu indeksów.

Skąd się wzięło słowo „algorytm”?

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

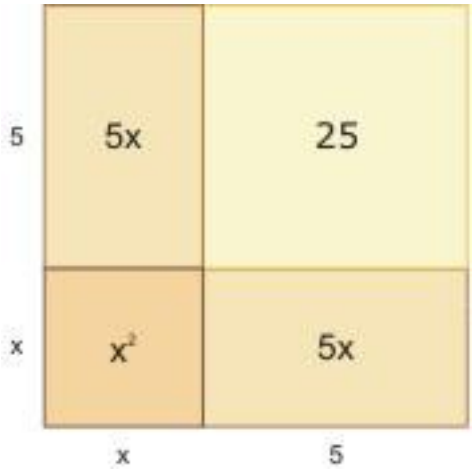
Al Chwarizmi

- **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 $0 \leq m \leq n$, $n > 0$

- ***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}/40s = ponad 790 miliardów lat ! To kilkadziesiąt razy więcej, niż trwa nasz Wszechświat!***
  - ***W szyfrowaniu RSA stosuje się liczby rzędu***

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 złośliwą daną nie będzie największa liczba z badanego zakresu oraz jedynka. Dla tej danej 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*

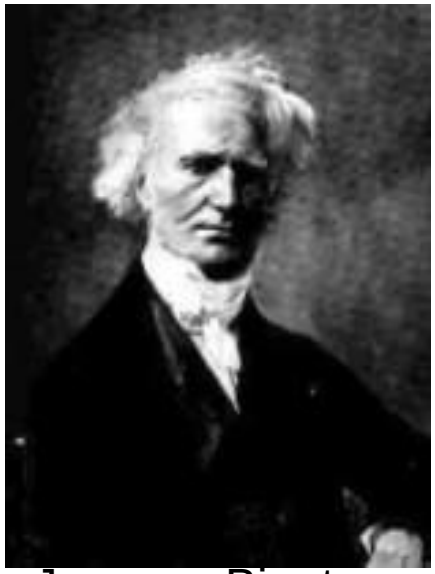
Z pary (F_{n-1},F_{n}) dla n>2 uzyskujemy n-2 obroty pętli.

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:

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

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!

Ważne sprawy, które poruszyliśmy

- Dziedzina algorytmiczna
  - Aby mówić o algorytmach, należy zawsze mieć na uwadze dokładny repertuar podstawowych środków.
  - W przypadku konstrukcji geometrycznych obiektami były punkty, proste, okręgi, wraz z pięcioma dopuszczalnymi operacjami opisanymi na poprzednich slajdach

Dziedziny algorytmiczne dla algorytmu Euklidesa

- W obu algorytmach Euklidesa używaliśmy nieco innych operacji:
  - W algorytmie Euklides 1: (N, >, –, zamień)
  - W algorytmie Euklides 2: (N, $>_0$, **mod**)

Dziedzina algorytmiczna

- Podstawowym pojęciem algorytmiki jest dziedzina algorytmiczna, czyli system relacyjny

$$\langle A, \{o_i\}_{i \in I}, \{r_j\}_{j \in J} \rangle$$

- gdzie zbiór A jest nośnikiem, zaś zbiory

$$\{o_i\}_{i \in I}, \{r_j\}_{j \in J}$$

- są odpowiednio zbiorami operacji i relacji na nośniku.

Programowanie strukturalne

- Przy programowaniu strukturalnym dziedziny tworzymy hierarchicznie, pozwalając dziedzinie wyższego poziomu korzystać z operacji poziomu niższego.
- Chodzi o to, żeby na żadnym poziomie nie bawić sie zbytnimi szczegółami
- Ten styl programowania jest charakterystyczny dla języków nowszych generacji (obiektowe, funkcyjne, w logice).

Złożoność obliczeniowa

- To było drugie, niezwykle ważne pojęcie algorytmiki.
- Chodzi o liczbę operacji, czyli koszt obliczeń.
- Można ją rozważać w stosunku do
  - algorytmu
  - problemu algorytmicznego.

Co zliczamy badając złożoność?

- Trudno byłoby skupiać się na każdej operacji wykonywanej w czasie działania algorytmu.
- Wybiera się zatem taką, która wykonuje sie najczęściej, a przy tym jest najdroższa (np. operacja mod w algorytmie Euklides 2) i liczy się liczbę jej wystąpień dla pewnych danych..

Złożoność pesymistyczna a średnia

- Najczęściej jako dane przyjmuje się jeden z 2 wariantów:
  - dane złośliwe (złożoność pesymistyczna)
  - dane typowe (złożoność średnia)

Złożoność pamięciowa

- Czasem do wykonania algorytmu potrzeba nam dodatkowej pamięci. Rozważamy wtedy złożoność pamięciową i podobnie jak poprzednio dzielimy ją, w zależności od danych, na:
  - złożoność pamięciową pesymistyczną
  - złożoność pamięciową średnią

Złożoność problemu

- W końcu mówimy też o złożoności problemu; jest to złożoność najlepszego algorytmu w danej klasie rozwiązującego ten problem. Tu też wyróżniamy:
  - złożoność problemu pesymistyczną
  - złożoność problemu średnią

Złożoność obliczeniowa

- Zatem mamy trzy kategorie złożoności, które można rozważać niezależne:

czas – pamięć

pesymistyczna – średnia

algorytm – problem

Rozmiar danych

- Rozmiar danych liczymy w odniesieniu do liczby bitów koniecznych do reprezentowania tych danych
  - dla tablic – ich długość
  - dla liczb – liczba cyfr
  - dla grafów – łączna liczba węzłów i krawędzi

Podsumowanie

- Algorytmika jest sercem informatyki
- Nie można programować byle jak. Nawet najszybsze komputery nie poradzą sobie z programami napisanymi niechlujnie.
- Zawsze warto zastanowić się nad tym, jak ustrukturalnić rozwiązanie i jaki będzie jego koszt.