

# TEORETYCZNE PODSTAWY INFORMATYKI

10/10/2016

WFAiS UJ, Informatyka Stosowana  
I rok studiów, I stopień

# Wykład 2

2

Struktury  
danych i  
algorytmy

- **Typy danych i struktury danych**
- **Analiza algorytmów**
- **Sposoby zapisu algorytmów**
- **Rodzaje algorytmów**
- **Schematy blokowe i algografy**
- **Wybór algorytmu**

\* Niektóre przykłady z wykładu prof. T. Roughgarden, Stanford, USA

# Struktury danych i algorytmy

3

- **Struktury danych** to **narzędzia** do reprezentowania informacji która ma być przetworzona przez program komputerowy,
- **Algorytmy** to **przepisy** wykonania czynności niezbędnych do jej przetworzenia.
- **Wybór algorytmu** do rozwiązania konkretnego problemu programistycznego pomaga w ustaleniu, jaką strukturę danych należałoby użyć, ale i odwrotnie – **wybrana struktura danych** ma ogromny wpływ na szczegóły realizacji i efektywności algorytmu.

# Typy danych i struktury danych

4

- Dane są to „**obiekty**” którymi manipuluje algorytm.
- Te obiekty to nie tylko dane wejściowe lub wyjściowe (wyniki działania algorytmu), to również obiekty pośrednie tworzone i używane w trakcie działania algorytmu.
- Dane mogą być różnych **typów**, do najpospolitszych należą liczby (całkowite, dziesiętne, ułamkowe) i słowa zapisane w rozmaitych alfabetach.

# Typy danych i struktury danych

5

- **Interesują nas sposoby w jaki algorytmy mogą organizować, zapamiętywać i zmieniać zbiory danych oraz „sięgać” do nich.**
  - ▣ **Zmienne czyli „pudelka” w których chwilowo przechowujemy jakąś wartość,**
  - ▣ **Wektory,**
  - ▣ **Listy,**
  - ▣ **Tablice czyli tabele (macierze), w których to możemy odwoływać się do indeksów,**
  - ▣ **Kolejki i stosy,**
  - ▣ **Drzewa, czyli hierarchiczne ułożenie danych,**
  - ▣ **Zbiory.... Grafy.... Relacje....**

# Typy danych i struktury danych

6

- W wielu zastosowaniach same struktury danych nie wystarczają.
- Czasami potrzeba bardzo **obszernych zasobów danych**, stanowiących dla wielu algorytmów potencjalne dane wejściowe, a więc mające ustaloną strukturę i nadające się do odszukiwania i manipulowania nimi. Nazywa się je **bazami danych** (relacyjne i hierarchiczne).
- Kolejny krok to **bazy wiedzy**, których elementami są bazy danych, a które zawierają również informacje o związkach pomiędzy danymi.

# Algorytmy

7

- **Algorytm** to „**przepis postępowania**” prowadzący do rozwiązania konkretnego zadania; zbiór poleceń dotyczących pewnych obiektów (danych) ze wskazaniem kolejności w jakiej mają być wykonane”. Jest jednoznaczna i precyzyjną definicją (specyfikacją) kroków które mogą być wykonywane „mechanicznie”.
- **Algorytm** odpowiada na pytanie „**jak to zrobić**” postawione przy formułowaniu zadania. Istota algorytmu polega na rozpisaniu całej procedury na kolejne, możliwie elementarne kroki.
- **Algorytmiczne myślenie** można kształtować niezależnie od programowania komputerów, chociaż każdy program komputerowy jest zapisem jakiegoś algorytmu.

# Analiza algorytmów

8

- **Analiza algorytmów i powiązanych z nimi struktur danych.**
  - ▣ **Znalezienie najlepszych sposobów wykonywania najczęściej spotykanych poleceń,**
    - **musimy nauczyć się podstawowych technik projektowania dobrych algorytmów.**
  - ▣ **Zrozumienie w jaki sposób wykorzystywać struktury danych i algorytmy tak, by tworzyć efektywne (szybkie) programy.**



# Mnożenie dwóch liczb całkowitych (szkolny algorytm)

9

- Wejście: dwie **n-cyfrowe** liczby  $x, y$
- Wyjście:  $z = x * y$
- Operacje: mnożenie i dodawanie dwóch 1-dno cyfrowych liczb

$$\begin{array}{r} 5678 \\ \times 1234 \\ \hline 22712 \\ 17034 \\ 11356 \\ 5678 \\ \hline 7006652 \end{array}$$

Ilość operacji:  
 $\sim n$  dla każdego wiersza

Ilość wszystkich operacji:  
 $\sim n * n = n^2$   
z dokładnością do stałej

# Mnożenie dwóch liczb całkowitych (algorytm Karatsuba)

10

- **Krok 1: policz**  $a \cdot c = 672$
- **Krok 2: policz**  $b \cdot d = 2652$
- **Krok 3: policz**

$x = 5678$   
 $y = 1234$

$$(a+b)(c+d) = 134 \cdot 46 = 6164$$

- **Krok 4: policz**  $(3) - (2) - (1) = 2840$
- **Krok 5:**

$$\begin{array}{r} 6720000 \\ 2652 \\ \hline 284000 \\ \hline 7006652 = (1234)(5678) \end{array}$$

# Mnożenie dwóch liczb całkowitych (algorytm Karatsuby)

11

- **Rozpisujemy liczby  $x, y$ :**

$$x = 10^{n/2}a + b \quad y = 10^{n/2}c + d$$

**$a, b, c, d$  są liczbami  $n/2$  cyfrowymi**

- **Reprezentujemy iloczyn jako**

$$\boxed{x \cdot y} = (10^{n/2}a + b)(10^{n/2}c + d) \\ = \boxed{10^n ac + 10^{n/2}(ad + bc) + bd}$$

- **Należy obliczyć:**

- (1)  $ac$  (2)  $bd$  (3)  $(a+b)(c+d) = ac + ad + bc + bd$
- Gauss trick:  $(3) - (1) - (2) = ad + bc$

**Kilka mnożeń i dodawań liczb  $n/2$  cyfrowych, czy to jest lepszy algorytm?**

# Sposoby zapisu algorytmu

12

- **Najprostszy sposób zapisu to zapis słowny**
  - ▣ **Pozwala określić kierunek działań i odpowiedzieć na pytanie, czy zagadnienie jest możliwe do rozwiązania.**
- **Bardziej konkretny zapis to lista kroków**
  - ▣ **Staramy się zapisać kolejne operacje w postaci kolejnych kroków które należy wykonać.**
- **Bardzo wygodny zapis to zapis graficzny**
  - ▣ **schematy blokowe i grafy.**

# Sposoby zapisu algorytmu

13

- **Zapis algorytmu przy pomocy listy kroków:**
  - ▣ **sformułowanie zagadnienia (zadanie algorytmu),**
  - ▣ **określenie zbioru danych potrzebnych do rozwiązania zagadnienia (określenie czy zbiór danych jest właściwy),**
  - ▣ **określenie przewidywanego wyniku (wyników): co chcemy otrzymać i jakie mogą być warianty rozwiązania,**
  - ▣ **zapis kolejnych ponumerowanych kroków, które należy wykonać, aby przejść od punktu początkowego do końcowego.**

# Rodzaje algorytmów

14

## Algorytm liniowy:

- Ma postać ciągu kroków których jest **liniowa ilość** (np. stała albo proporcjonalna do liczby danych) które muszą zostać bezwarunkowo wykonane jeden po drugim.
- Algorytm taki **nie zawiera żadnych warunków ani rozgałęzień**: zaczyna się od podania zestawu danych, następnie wykonywane są kolejne kroki wykonawcze, aż dochodzimy do wyniku.

# Algorytm liniowy - przykład

15

- **Przykład: dodanie lub mnożenie dwóch liczb**
- **Sformułowanie zadania:**  
**oblicz sumę dwóch liczb naturalnych:  $a, b$ . Wynik oznacz przez  $S$ .**
- **Dane wejściowe:** dwie liczby  $a$  i  $b$
- **Cel obliczeń:** obliczenie sumy  $S = a + b$
- **Dodatkowe ograniczenia:** sprawdzenie warunku dla danych wejściowych np. czy  $a, b$  są naturalne.
  - ▣ **Ale sprawdzenie pewnych warunków sprawia że algorytm przestaje być liniowy**

# Rodzaje algorytmów

16

## Algorytm z rozgałęzieniem:

- Większość algorytmów zawiera rozgałęzienia będące efektem sprawdzania warunków. Wyrażenia warunkowe umożliwiają wykonanie zadania dla wielu wariantów danych i rozważanie różnych przypadków.
- Powtarzanie różnych działań ma dwojaką postać:
  - ▣ liczba powtórzeń jest z góry określona (przed rozpoczęciem cyklu), alg. najczęściej związany z działaniami na tablicach,
  - ▣ liczba powtórzeń jest nieznana (zależy od spełnienia pewnego warunku), alg. najczęściej związany z obliczeniami typu iteracyjnego.



# Algorytm z rozgałęzieniem - przykład

17

## □ Sformułowanie zadania

Znajdź rozwiązanie równania liniowego postaci  $a \cdot x + b = 0$ .

Wynikiem jest wartość liczbowa lub stwierdzenie dlaczego nie ma jednoznacznego rozwiązania.

## □ Dane wejściowe

Dwie liczby rzeczywiste  $a$  i  $b$

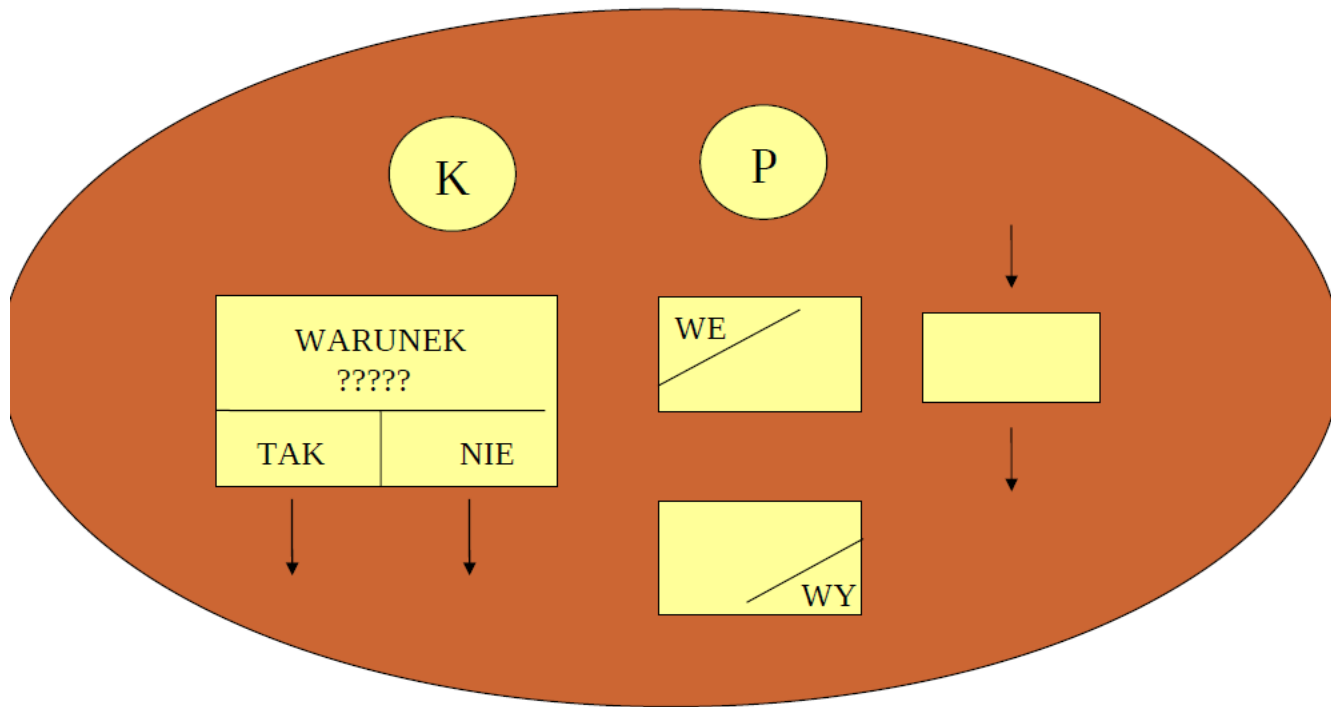
## □ Cel obliczeń (co ma być wynikiem)

Obliczenie wartości  $x$  lub stwierdzenie, że równanie nie ma jednoznacznego rozwiązania.

- gdy  $a = 0$  to sprawdź czy  $b = 0$ , jeśli tak to równanie sprzeczne lub tożsamościowe
- gdy  $a \neq 0$  to oblicz  $x = -b/a$

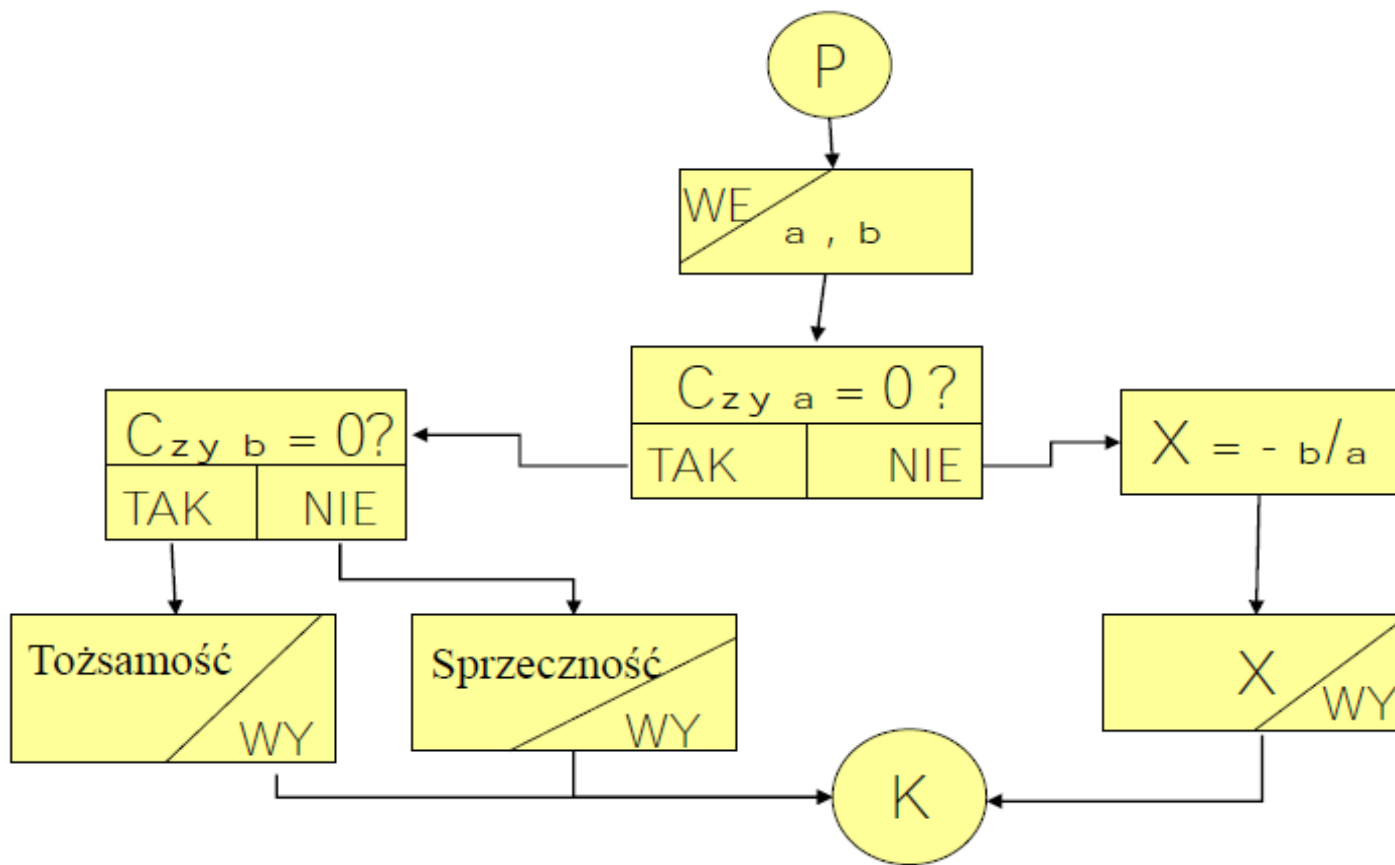
# Schematy blokowe i algografy

18



# Schemat blokowy rozwiązania równania liniowego

19



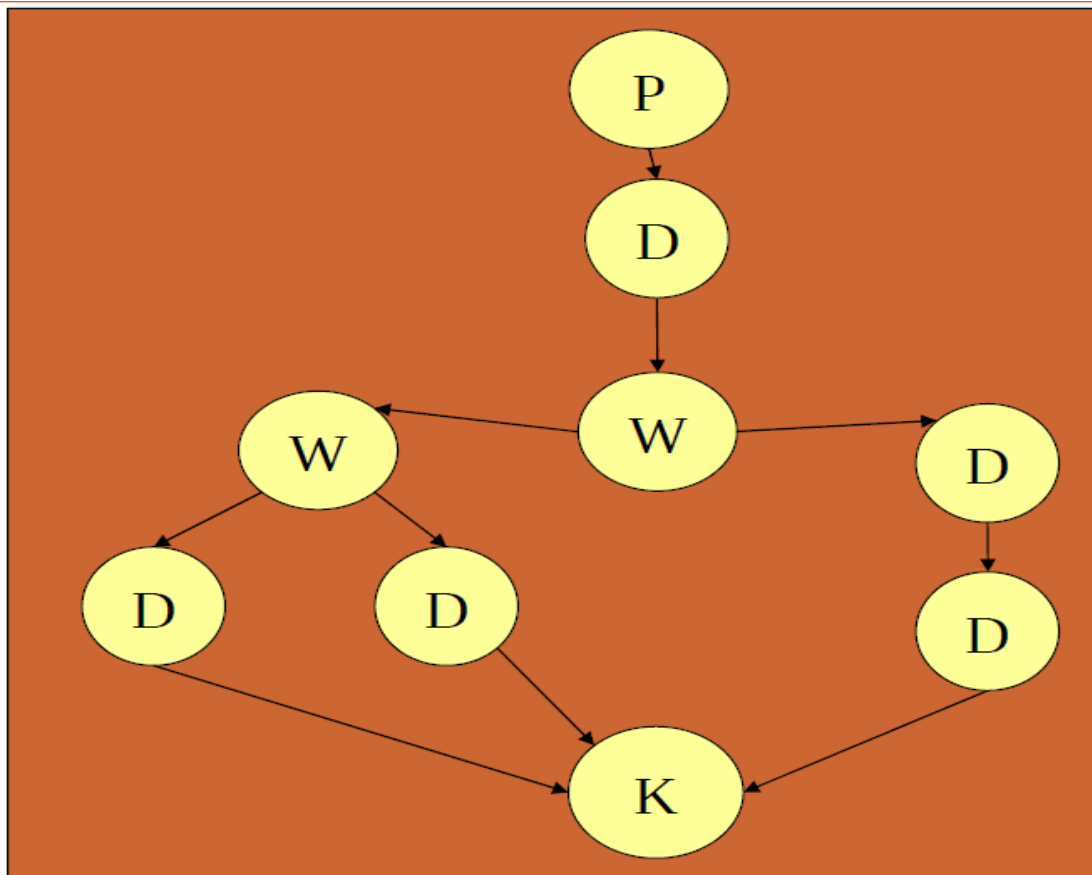
# Grafy

20

- **Graf** symbolizuje przepływ informacji.
- Graf składa się z **węzłów i ścieżek**.
- W przypadku algorytmów graf można wykorzystać aby w uproszczonej formie zilustrować ilość różnych dróg prowadzących do określonego w zadaniu celu.
- Graf pozwala wykryć drogi, które nie prowadzą do punktu końcowego, których to poprawny algorytm nie powinien posiadać.

# Graf algorytmu rozwiązania równania liniowego

21



- P – początek
- K – koniec
- D – działanie
- W – warunek

# Grafy

22

- **Jeżeli w grafie znajduje się ścieżka, która nie doprowadza do węzła końcowego, to mamy do czynienia z niepoprawnym grafem.**
- **W programie przygotowanym na podstawie takiego grafu, mamy do czynienia z przerwaniem próby działania i komunikatem o zaistnieniu jakiegoś błędu w działaniu.**
- **Węzeł grafu może mieć dwa wejścia jeżeli ilustruje pętle. Wtedy liczba ścieżek początek-koniec może być nieskończona, gdyż nieznana jest liczba obiegów pętli.**

# Schemat blokowy czy graf ?

23

- **Graf to tylko schemat kontrolny służący do sprawdzenia algorytmu.**
  - ▣ **Brak informacji o wykonywanych operacjach**
- **Schemat blokowy służy jako podstawa do tworzenia programów.**

# Algorytmy: „dziel i zwyciężaj”

24

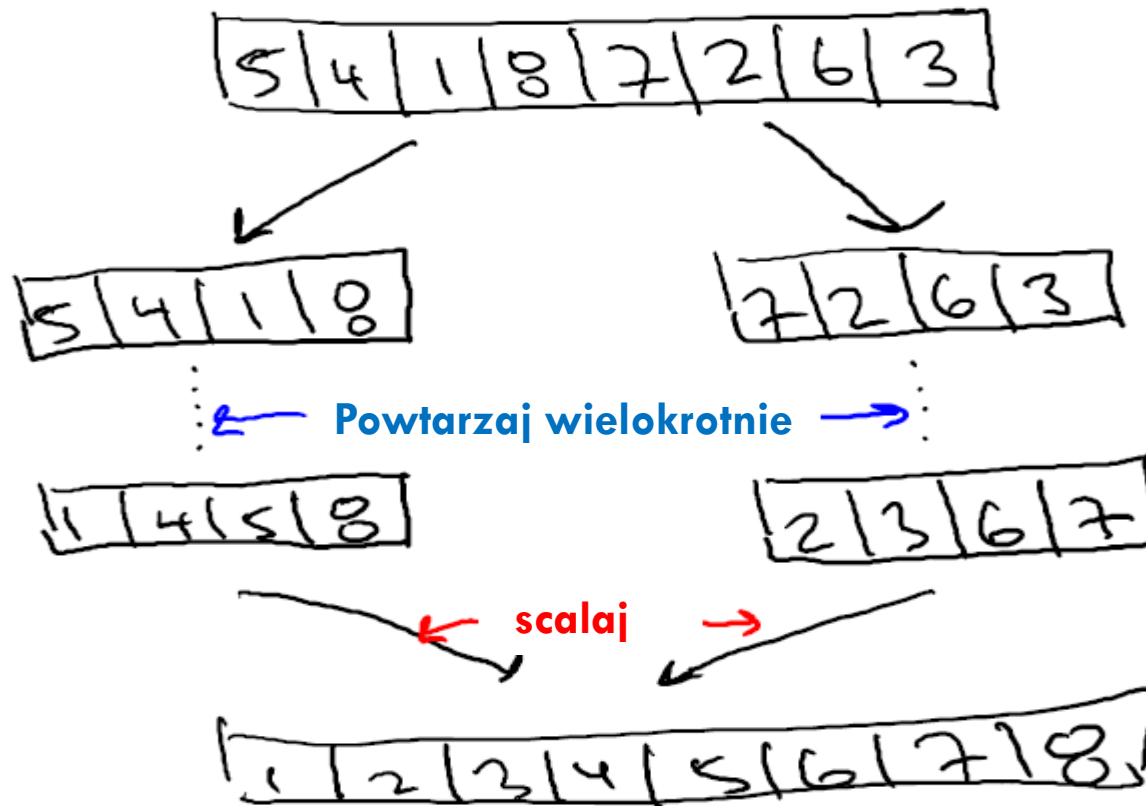
- **Metoda: „dziel i zwyciężaj” :**
  - ▣ **Dzielimy problem na mniejsze części tej samej postaci co pierwotny.**
  - ▣ **Teraz te pod-problemy dzielimy dalej na coraz mniejsze, używając tej samej metody, aż rozmiar problemu stanie się tak mały, że rozwiązanie będzie oczywiste lub będzie można użyć jakiejś innej efektywnej metody rozwiązania.**
  - ▣ **Rozwiązania wszystkich pod-problemów muszą być połączone w celu utworzenia rozwiązania całego problemu.**
- **Ten typ algorytmów zazwyczaj jest implementowany z zastosowaniem technik rekurencyjnych.**



# Algorytmy: „dziel i zwyciężaj”

25

- Mamy posortować tablicę liczb, zakładamy że są różn



# Algorytmy: „dziel i zwyciężaj”

26

- **Jak znaleźć minimum ciągu liczb?**
  - ▣ **Dzielimy ciąg na dwie części, znajdujemy minimum w każdej z nich, bierzemy minimum z obu liczb jako minimum ciągu.**
  
- **Jak sortować ciąg liczb?**
  - ▣ **Dzielimy na dwie części, każdą osobno sortujemy a następnie łączymy dwa uporządkowane ciągi (scalamy).**

# Algorytmy oparte na programowaniu dynamicznym

27

- Można stosować wówczas, kiedy problem daje się podzielić na wiele pod-problemów, których rozwiązania są możliwe do zapamiętania w jedno-, dwu- lub wielowymiarowej tablicy w taki sposób że w pewnej określonej kolejności można je wszystkie (a więc i cały problem) efektywnie rozwiązać.

*Jak obliczać ciąg Fibonacciego?*

$$F(i) = \begin{cases} 1 & \text{jeśli } i = 1 \\ 1 & \text{jeśli } i = 2 \\ F(i-2) + F(i-1) & \text{jeśli } i > 2 \end{cases}$$

- Aby obliczyć  $F(n)$ , wartość  $F(k)$ , gdzie  $k < n$  musimy wyliczyć  $F(n-k)$  razy.
- Liczba obliczeń rośnie wykładniczo.
- **Korzystnie jest więc zachować (zapamiętać w tablicy) wyniki wcześniejszych obliczeń ( $F(k)$ ).**

# Jak obliczać liczbę kombinacji?

28

- Liczba kombinacji (podzbiorów)  $r$ -elementowych ze zbioru  $n$ -elementowego oznaczana  $\binom{n}{r}$ , dana jest wzorem:

$$\binom{n}{r} = n! / (r! (n-r)!)$$

Możemy użyć wzorów:

$$\binom{n}{r} = 1, \text{ jeśli } r = 0 \text{ lub } n = r$$

$$\binom{n}{r} = \binom{n-1}{r-1} + \binom{n-1}{r} \text{ dla } 0 < r < n$$

Obliczamy rząd po rzędzie w **trójkącie Pascala**

r \ n	0	1	2	3	4
0	1				
1	1	1			
2	1	2	1		
3	1	3	3	1	
4	1	4	6	4	1

# Algorytmy z powrotami

29

- Często możemy zdefiniować jakiś problem jako poszukiwanie rozwiązania wśród wielu możliwych przypadków.
- Dane:
  - ▣ Pewna przestrzeń stanów, przy czym stan jest to sytuacją stanowiącą rozwiązanie problemu albo mogąca prowadzić do rozwiązania
  - ▣ Sposób przechodzenia z jednego stanu do drugiego.
  - ▣ Mogą istnieć stany które nie prowadzą do rozwiązania.

**Przykładami tego typu algorytmów są gry.**

# Algorytmy z powrotami

30

## □ Metoda powrotów

- Wymaga zapamiętania wszystkich wykonanych ruchów czy też wszystkich odwiedzonych stanów aby możliwe było cofanie się (powroty).
- Stanów mogą być tysiące lub miliony więc bezpośrednie zastosowanie metody powrotów, mogące doprowadzić do odwiedzenia wszystkich stanów, może być zbyt kosztowne.
- Inteligentny wybór następnego posunięcia, tzw. **funkcja oceniająca**, może znacznie poprawić efektywność algorytmu.
  - Np. aby uniknąć przeglądania nieistotnych fragmentów przestrzeni stanów.

# Wybór algorytmu

31

- **Regułą jest że należy implementować algorytmy najprostsze, które wykonują określone zadanie.**
- **Prosty algorytm to**
  - ▣ **łatwiejsza implementacja, czytelniejszy kod**
  - ▣ **łatwość testowania**
  - ▣ **łatwość pisania dokumentacji,....**
- **Jeśli program ma działać wielokrotnie, jego wydajność i wykorzystywany algorytm stają się bardzo ważne.**
- **Błędy zaokrągleń, powstające przy reprezentacji liczb, a także przy wykonywaniu działań na nich rozwinęły się w samodzielną dziedzinę tzw. **analiza numeryczna**.**

# Wybór algorytmu

32

- **Istnieją również inne zasoby, które należy niekiedy oszczędnie wykorzystywać w pisanych programach:**
  - ▣ **ilość przestrzeni pamięciowej wykorzystywanej przez zmienne**
  - ▣ **generowane przez program obciążenie sieci komputerowej**
  - ▣ **ilość danych odczytywanych i zapisywanych na dysku**
  - ▣ **mniej obliczeń to lepsza dokładność numeryczna (zaokrąglenia)**



# Algorytm optymalnego dodawania liczb naturalnych

33

- Posortuj dane od liczby najmniejszej do największej względem modułów.
- Dopóki są co najmniej dwa elementy w zbiorze powtarzaj następujące działanie:
  - ▣ Pobierz z posortowanego zbioru dwie najmniejsze wartości **(dlaczego najmniejsze?)**
  - ▣ Dodaj do siebie
  - ▣ Włóż do zbioru z powrotem wynik dodawania

**Problemem nietrywialnym jest wykonywanie tego szybko → użyj stosu.**

# Wybór algorytmu

34

- **Zrozumiałość i efektywność:** to są często sprzeczne cele. Typowa jest sytuacja w której programy efektywne dla dużej ilości danych są trudniejsze do napisania/zrozumienia.
  - Np. sortowanie przez wybieranie (łatwy, nieefektywny dla dużej ilości danych) i sortowanie przez „dzielenie i scalanie” (trudniejszy, dużo efektywniejszy).
- **Zrozumiałość** to pojęcie względne, natomiast **efektywność** można obiektywnie zmierzyć: **testy wzorcowe, analiza złożoności obliczeń.**

# Efektywność algorytmu

35

## □ Czas działania:

- Oznaczamy przez funkcję  $T(n)$  liczbę jednostek czasu, które zajmuje wykonanie programu lub algorytmu w przypadku problemu o rozmiarze  $n$ .
- Funkcje te nazywamy **czasem działania**. Dość często czas działania zależy od konkretnych danych wejściowych, nie tylko ich rozmiaru. W takim przypadku, funkcję  $T(n)$  definiuje się jako **najmniej korzystny przypadek** z punktu widzenia kosztów czasowych. Inną wyznaczaną wielkością jest też **czas średni**, czyli średni dla różnych danych wejściowych.

# Testy wzorcowe

36

- Podczas porównywania dwóch lub więcej programów zaprojektowanych do wykonywania tego samego zadania, opracowujemy niewielki zbiór typowych danych wejściowych które mogą posłużyć jako dane wzorcowe (ang. benchmark).
- Powinny być one reprezentatywne i zakłada się że program dobrze działający dla danych wzorcowych będzie też dobrze działał dla wszystkich innych danych.
- Np. test wzorcowy umożliwiający porównanie algorytmów sortujących może opierać się na jednym **małym** zbiorze danych, np. zbiór pierwszych 20 cyfr liczby  $\pi$ ; jednym **średnim**, np. zbiór kodów pocztowych województwa krakowskiego; oraz na **dużym** zbiorze takim jak zbiór numerów telefonów z obszaru Krakowa i okolic.
- Przydatne jest też sprawdzenie jak algorytm działa dla ciągu **już posortowanego** (często działają kiepsko).

# Uwagi końcowe

37

- **Na wybór najlepszego algorytmu dla tworzonego programu wpływa wiele czynników, najważniejsze to:**
  - **prostota,**
  - **łatwość implementacji**
  - **efektywność**

# Pytania do wykładu

38

- 1) **Co to jest algorytm i jakie znasz sposoby jego zapisu?**
- 2) **Scharakteryzuj, na czym polegają następujące typy algorytmów:**
  - liniowy
  - z rozgałęzieniem
  - z powrotami
  - „dziel i zwyciężaj”
  - zachłanny
  - oparty o programowanie dynamiczne
- 3) **Według jakich kryteriów efektywność algorytmu?**
- 4) **W jaki sposób badamy czas działania algorytmu?**