Andrzej Pająk
A.Pajak@ii.pw.edu.pl
Andrzej.Pajak@pw.edu.pl

Struktury danych (cz.1)

Instytut Informatyki
Studia Podyplomowe
Java EE — produkcja oprogramowania

Grupy **JA20Z - (JA2-A, B)**
Warszawa 2021

Plan

- Cele
- Po co są potrzebne struktury danych?
- Struktury danych i orientacja obiektowa
- Literatura
- Pojęcia podstawowe i definicje
- Klasyfikacja struktur danych
- O algorytmach
- Problem – model – algorytm – program
- Przykład: problem maksymalnej podtablicy
- Rozwiązania $O(n^2)$ i $O(n \log n)$
- Rozwiązanie optymalne: $O(n)$
- Pożytki z analizy – przykład „kolejowy”

Cele

- **Struktury danych (SD)** i algorytmy to podstawowy dział informatyki – warto poznać, jak z perspektywy języka Java widać te fundamentalne kwestie
- SD rozpatruje się zawsze w kontekście **algorytmów** korzystających z ich wsparcia; miarą tego wsparcia jest wpływ użytych SD na efektywność algorytmu
- **Ocena efektywności** (czasowej, pamięciowej) algorytmów często sprowadza się do oszacowania kosztu operacji podstawowych SD i kosztu reprezentacji
- **Zasadnicze cele cząstkowe**: zaprezentować główne kategorie struktur danych, pokazać typowe ich użycie w algorytmach i wsparcie środowiska Java dla bezpośredniego korzystania ze struktur danych (JCF)

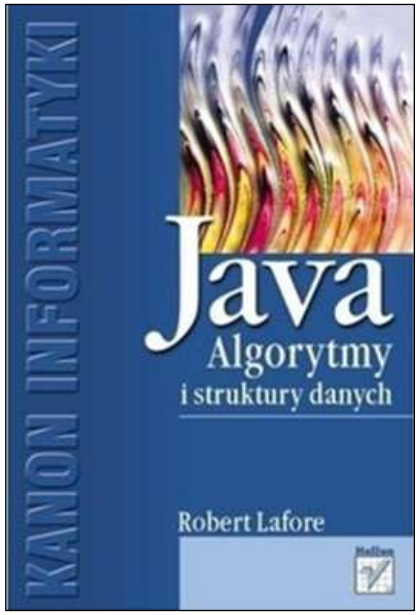
Po co są potrzebne struktury danych?

- Reprezentowanie **pojedynczych wielkości** (elementarnych danych) - wystarczają zmienne proste odpowiedniego typu
- Posługiwanie się zmiennymi prostymi gdy trzeba uwzględnić przetwarzanie **wielu wielkości** jest skrajnie niepraktyczne (na przykład: jeżeli mamy kod porządkujący 4 wartości w zmiennych a, b, c, d, to nie stosuje się on do 10 wartości)
- Jednym z wymagań algorytmizacji jest **skalowalność**: ten sam skończony algorytm powinien umożliwiać przetwarzanie instancji problemów o dowolnym (sensownym) rozmiarze
- **Struktury danych** powstawały (i powstają nadal) równolegle z rozwojem nowych zastosowań i algorytmów (np. Big Data)
- Jak prawie wszystko w informatyce SD podlegają **ocenie ilościowej**: jaki jest koszt organizacji i dostępu do danych

Struktury danych i orientacja obiektowa

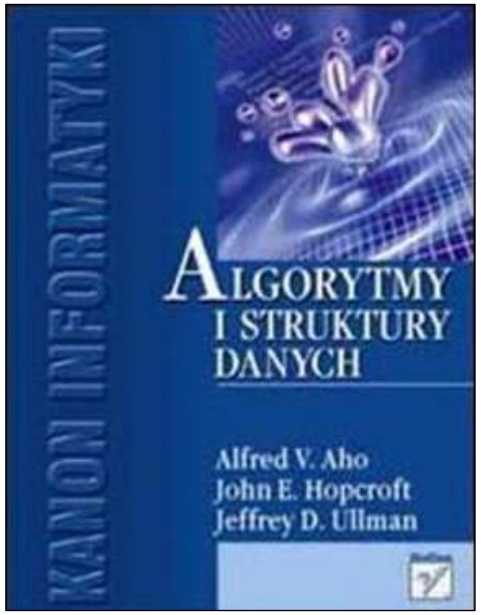
- Paradygmat programowania obiektowego i struktury danych są sprzężone specyficzną **synergią**:
  - mechanizmy obiektowe pozwalają traktować struktury danych jak zwykłe obiekty i komponować bardziej złożone struktury z istniejących
  - struktury danych wzmacniają ekspresyjność obiektową i pozwalają lepiej organizować algorytmy
- Java wykorzystuje **mechanizm interfejsów** do zorganizowania hierarchii abstrakcji struktur danych (**JCF**)
- **Klasy abstrakcyjne** pozwalają predefiniować szkieletowe realizacje najważniejszych kategorii struktur danych (do uszczegółowienia wg specjalizowanych potrzeb)
- ***Uwaga:*** wszystkie struktury danych platformy Java należą do **systemu typów**, który ma cechę **ortogonalności**.

Literatura

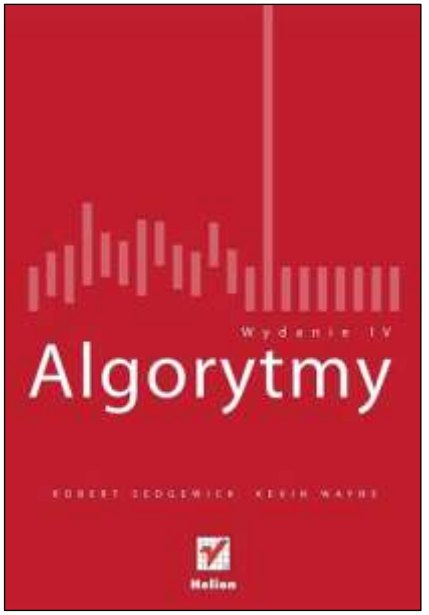


R. Lafore: *Java. Algorytmy i struktury danych.* Helion, 2004, 704s.

E. Koffman, P. Wolfgang: *Struktury danych i techniki obiektowe na przykładzie Javy 5.0*, Helion 2006, 934s.



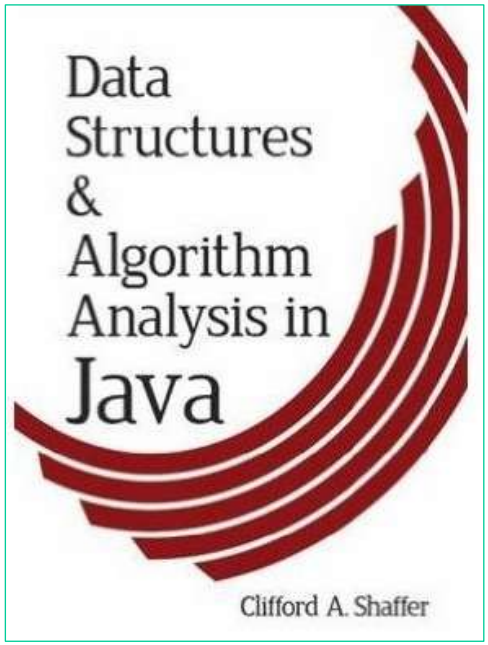
A. Aho, J. Hopcroft, J. Ullman: *Algorytmy i struktury danych*. Helion 2003, 442s.



R. Sedgewick, K. Wayne: *Algorytmy*, Helion, 2012. 952s..

Literatura (cd)

Data Structures and Algorithm Analysis, Edition 3.2 (Java Version)
Clifford A. Shaffer, Dept. of Computer Science, Virginia Tech, Update 3.2.0.10
For a list of changes, see
http://people.cs.vt.edu/~shaffer/Book/



Strona do odwiedzenia: https://opendsa-server.cs.vt.edu/home/books

Pojęcia podstawowe i definicje

- **Typ danych:** zbiór dopuszczalnych wartości i zestaw postulowanych operacji na tych wartościach. Przykłady:
  - **int** – podzbiór liczb całkowitych; operacje arytmetyczne, wyznaczanie nwp(a, b), badanie pierwszości, faktoryzacja, konwersje, ...
  - **DateTime** – reprezentacja daty i czasu kalendarza gregoriańskiego; konwersje do innych kalendarzy, operacja różnicy dat (suma nie ma sensu), prezentacja znakowa wg konwencji regionalnych, ...
- **Abstrakcyjny typ danych (ADT):** sformalizowana specyfikacja typu danych uwzględniająca postulowane własności (niezmienniki) i operacje (**w Javie interfejs**); ADT może być zrealizowany na wiele sposobów
- **Struktura danych (SD):** reprezentacja danych przydatna do realizacji pewnych ADT, efektywnie wspierająca postulowane operacje; jedna SD może być użyta w realizacji wielu ADT.

Pojęcia podstawowe i definicje (cd1)

- W językach obiektowych pojęcia ***typ, ADT, SD*** należą do **otwartego** systemu typów języka. Wyróżnienie struktur danych ma znaczenie tylko narracyjne dla podkreślenia roli przyjętej organizacji danych w ramach pewnego typu.
- **Mechanizmy tworzenia nowych typów** (w Javie i nie tylko**)**

1. **Agregacje jednorodne (tablice `Typ[]`)** – szybki dostęp przez **indeksowanie**; rozmiar agregacji ustalony w momencie tworzenia

```
// Tablica zajmuje spójny obszar pamięci
int[] primes = {2,3,5,7,11,13,17,19}; // indeks 0..7

for (int p : primes)
  System.out.print(p + " ");  // 2 3 5 7 11 13 17 19
System.out.println();

primes[primes.length]=23;
// ArrayIndexOutOfBoundsException
int[][] T3x3 = {{1,2,3},{4,5,6},{7,8,9}}; // Tablica 3x3
```

Uwaga:
Tablice są kowariantne

Pojęcia podstawowe i definicje (cd2)

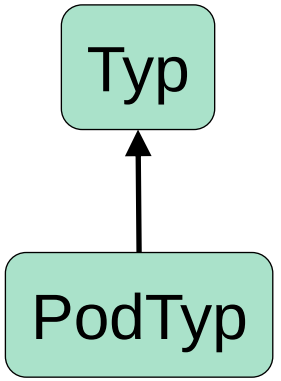
- Przypomnienie: **kowariantność** typów tablicowych oznacza zachowanie relacji:
  PodTyp<=Typ ⇒ PodTyp[]<=Typ[]

```
String[] s = {"Zero", null};
Object[] o = s; // String[]<=Object[]
o[1] = "Jeden"; // OK, operacja
                // dotyczy String[]
// s zawiera {"Zero", "Jeden"};

o[1]=1;// Kompilacja OK;
       //wyk.: ArrayStoreException

Para<String, Integer> psi =
  new Para<>("Dwa", 2);

Para<String, Integer>[] psiTab =
  new Para<>[2];
// Cannot create a generic array
```




```
class Para<P, Q> {
  private P p;
  private Q q;
  public Para(P p, Q q){
    this.p=p; this.q=q;
  }
  // ...
}
```

Pojęcia podstawowe i definicje (cd3)

2. **Enumeracje** – wygodny i intuicyjny mechanizm tworzenia wykazu ustalonych wartości symbolicznych (z możliwością dodawania dodatkowych atrybutów):

```
enum Agent {                  // "Wyliczanka" 3 agentów
  KLOSS("J23"),               // Wywołuje Agent("J23");
  BOND("007"),
  SKARBEK("Granville"); // Polska agentka w GB

  private final String pseudonim;// atrybut
  private Agent(String pseudo) { // Konstruktor
    pseudonim = pseudo;
  }
  public String getPseudo() {
    return pseudonim;
  }
}
// ...
for(Agent a : Agent.values())
  System.out.println(a + " " + a.getPseudo());
```

```
KLOSS J23
BOND 007
SKARBEK Granville
```

Pojęcia podstawowe i definicje (cd4)

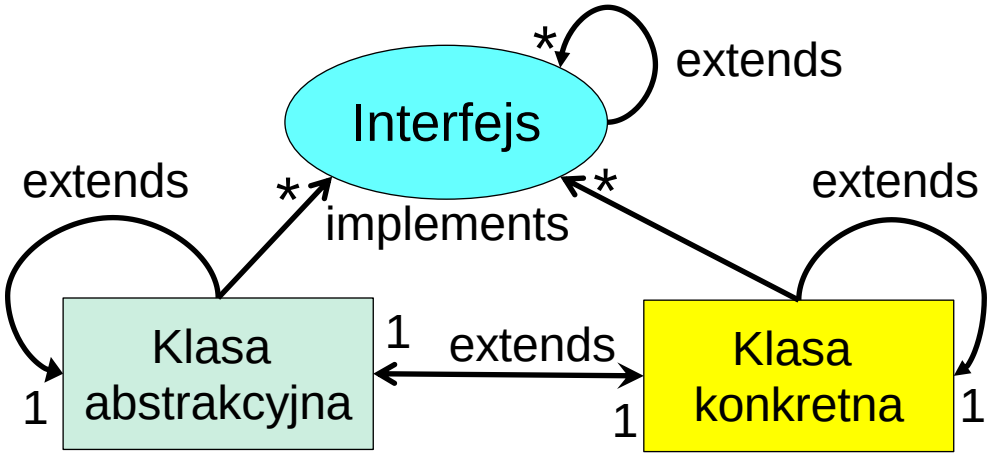
3. **Agregacje niejednorodne (klasy)** – ogólny i bardzo silny mechanizm tworzenia nowych typów

- Zagregowane składowe (dowolnych typów) są łącznie traktowane jako logiczna całość – **reprezentacja stanu wewnętrznego** obiektu utworzonego wg klasy. **[składowe instancyjne i statyczne]**
- Operacje, definiowane jako metody, także instancyjne lub statyczne
- Dostęp do składowych poprzez nazwy (nazwy metod są przeciążalne)
- Mechanizm gwarantuje enkapsulację i kontrolę dostępu do składowych
- Agregacje niejednorodne pozwalają tworzyć **typy dynamiczne** (wykorzystujące odpowiednio zarządzane agregacje jednorodne lub struktury powiązane wskaźnikami/referencjami)

4. **Dziedziczenie, polimorfizm i zarządzanie interfejsami** – elastyczny system z wykorzystaniem specjalnych typów (**interfejsów i klas abstrakcyjnych**) pozwala tworzyć nowe typy wg istniejących

Pojęcia podstawowe i definicje (cd5)

Dziedziczenie i definiowanie zachowania dla klas odbywa się w trójkącie pojęć:
klasa konkretna, klasa abstrakcyjna, interfejs



Efekt definicji	Perspektywa systemu typów	Perspektywa implementacji
Klasa konkretna	Definiuje typ i jego operacje	Zapewnia implementację stanu i wszystkich operacji
Klasa abstrakcyjna	Definiuje typ i jego operacje	Może implementować wybrane operacje i określać częściowo stan
Interfejs	Definiuje typ i jego operacje	Może implementować wybrane operacje (**default**)

Klasyfikacja struktur danych

Powszechnie przyjmowany podział SD:

- **Struktury liniowe**
  - tablica (wektor, array) – agregacja indeksowana
  - kopiec (heap) binarny, kopiec k-arny – drzewo wirtualne
  - lista (jednokierunkowa, dwukierunkowa, cykliczna, z wartownikiem) – realizowana przy pomocy odnośników
  - **kontenery sekwencyjne JCF (ArrayList, LinkedList, ...)**
- **Struktury drzewiaste (hierarchiczne) wiązane odnośnikami**
  - drzewo binarne, drzewo binarne szukania, drzewa zrównoważone (AVL, czerwono-czarne), B-drzewa
  - drzewa czwórkowe, drzewa ósemkowe, kd-drzewa
  - **kontenery asocjacyjne JCF (TreeSet, TreeMap, ...)**

Klasyfikacja struktur danych (cd)

- **Struktury grafowe (uogólnienie struktur drzewiastych)**
  - grafy skierowane, grafy nieskierowane
  - grafy ważone
  - grafy acykliczne
  - sieci przepływowe
  - ...
- **Struktury z haszowaniem**
  - tablica z haszowaniem
  - słowniki, tabele symboli, tablice asocjacyjne
  - **struktury biblioteki JCF (HashSet, HashMap, ...)**

O algorytmach

Algorytm jest **ścisłym** opisem **skończonego** ciągu **czynności**, gwarantującego osiągnięcie pewnego celu (rozwiązania) przez określonego **wykonawcę**.

Ścisłość opisu oznacza brak niejednoznaczności w specyfikacji; opis w istocie może być przeznaczony dla wykonawcy pozbawionego świadomości, intelektu i innych cech ludzkich

Skończoność ciągu czynności odnosi się do algorytmów typu wejście – wyjście; istnieje klasa algorytmów cyklicznych nieskończonych ("protokołów"), w których skończoność dotyczy treści powtarzalnej sekwencji czynności

Czynność jest pojęciem wymagającym zdefiniowania; zbiór czynności dopuszczalnych określa **model wykonawcy**.

O algorytmach (cd1)

Kategorie ogólne problemów i algorytmów

- problemy ***jednostkowe sporadyczne*** – mało ciekawe algorytmicznie ("algorytm" jest sztywnym przepisem postępowania w konkretnych, ustalonych warunkach)
- problemy ***jednostkowe uporczywe*** – pojawiają się w innych problemach i algorytmach jako czynności podstawowe; często same są interesujące i inspirujące.
- problemy masowe i **algorytmy skalowalne** – najbardziej interesująca i najważniejsza kategoria problemów; problem może mieć nieograniczoną liczbę konkretyzacji (instancji) różniących się **rozmiarem** i szczegółami wewnętrznymi; są to **PROBLEMY OBLICZENIOWE**.

O algorytmach (cd2)

Rodzaje algorytmów

- Pełna klasyfikacja algorytmów jest praktycznie niemożliwa; można przeprowadzać doraźne klasyfikacje ze względu na różne kryteria zależnie od potrzeb
- Ze względu na model wykonawcy:
  - **Sekwencyjne (TE NAS INTERESUJĄ)**
  - Równoległe
  - Rozproszone
- Ze względu na ogólny sposób wykonywania obliczeń
  - Dokładne
  - Przybliżone
  - Randomizowane
  - Heurystyczne / Metaheurystyczne

O algorytmach (cd3)

- Klasyfikacja algorytmów ze względu na dziedzinę
  - Numeryczne (np. wyznaczanie pierwiastków układu równań)
  - Geometryczne (np. wyznaczanie otoczki wypukłej zbioru punktów na płaszczyźnie)
  - Kombinatoryczne (np. szukanie rozwiązania Su-Do-Ku)
  - Grafowe (np. znajdowanie minimalnej ścieżki w grafie)
  - Przetwarzania tekstu (np. parser języka Java)
  - Przetwarzania obrazu (np. kompresja obrazu)
  - .....
- Ze względu na kategorię wyniku
  - Decyzyjne (wynikiem jest rozstrzygnięcie TAK / NIE)
  - Konstruktywne (wynikiem jest opis pewnego obiektu)
  - Optymalizacyjne (wynikiem jest opis obiektu najlepszego wg ustalonych kryteriów)

O algorytmach (cd4)

- Ze względu na zastosowaną metodykę
  - Algorytmy dekompozycyjne (typu dziel i zwyciężaj): rozwiąż problem dzieląc go na mniejsze podproblemy tego samego rodzaju i łącząc rozwiązania podproblemów cząstkowych w rozwiązanie ostateczne
  - Algorytmy zachłanne: podczas optymalizacji wieloetapowej wykonaj sekwencję kroków najbardziej korzystnych
  - Algorytmy programowania dynamicznego: rozwiązanie optymalne jest konstruowane wg (zapamiętanych) rozwiązań optymalnych podproblemów cząstkowych
  - Algorytmy z nawrotami (wiele odmian): zwykle oznaczają systematycznie zorganizowane przeszukiwanie zbioru rozwiązań dopuszczalnych (np. szukanie drogi w labiryncie)
  - Algorytmy z transformacją dziedziny: oryginalny problem jest odwzorowywany na problem w innej dziedzinie.

Przykłady problemów

Problemy jednostkowe sporadyczne

- Instrukcje dla znajomego z Gdańska: jak z Dw. Centralnego dostać się do mojego mieszkania na Oś. Wyżyny

  1. Zadzwoń, że już jesteś; 2. Przejdź do stacji metra "Centrum"; 3. Pojedź metrem do stacji "Natolin"; 4. Wyjdź z metra wyjściem prawym idąc w kierunku jazdy pociągu – tam będę czekał.

  1. Zadzwoń gdy będziesz na dworcu Warszawa Wschodnia; 2. Na Dw. Centralnym będę czekał na peronie.

  1. Wsiądź w taksówkę i podaj adres.

- **Instrukcja dotycząca instalacji prostego oprogramowania**

  Sekwencja czynności operatorskich może być powierzona osobie nie znającej szczegółów technicznych komputera; polecenia wydaje automat w programie instalacyjnym.

- **Przepis na babę wielkanocną**

  1. Weź pół kopy jaj, ...

Przykłady problemów (cd1)

Problemy jednostkowe uporczywe

- Klasyczne (euklidesowe) konstrukcje geometryczne z wykorzystaniem cyrkla i linijki (tylko cyrkla w konstrukcjach Platona albo tylko linijki w konstrukcjach Steinera):
  - **Zbudować trójkąt foremny**
  - **Wyznaczyć środek okręgu opisanego na trójkącie**
  - **Zbudować odcinek o długości √7**
  - **Zbudować siedmiokąt foremny (niewykonalne)**
  - **Przeprowadzić trysekcję kąta (niewykonalne)**
  - **.....**

Dopuszczalne operacje podstawowe

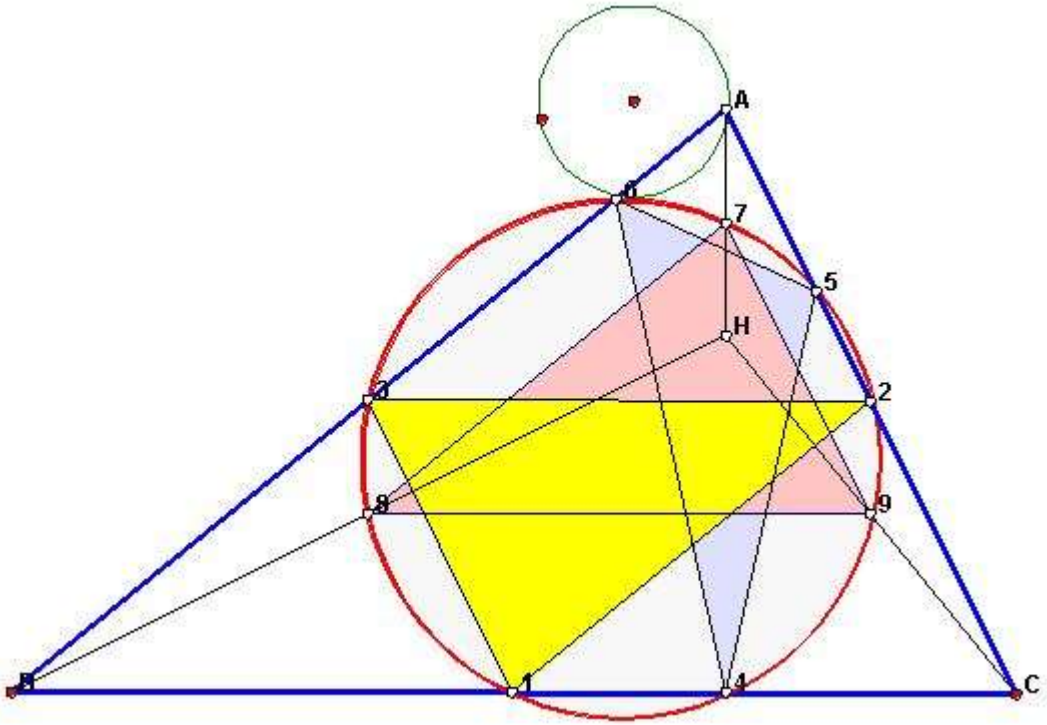
L1: poprowadź linię przez punkt (może generować punkty przecięć)

L2: poprowadź linię przez 2 punkty (może generować punkty przecięć)

C1: umieść cyrkiel w punkcie albo na prostej

C2: rysuj okrąg przechodzący przez punkt (może generować punkty przecięć)

Przykłady problemów (cd2)



Przykład konstrukcji: Okrąg 9 punktowy trójkąta ΔABC

1-3: środki boków trójkąta; 4-6: krańce wysokości trójkąta; 7-9: środki odcinków pomiędzy ortocentrum H (punktem przecięcia wysokości) i wierzchołkami A, B, C. Trójkąty Δ123 (żółty), Δ456 (niebieski), Δ789 (różowy) są odpowiednio trójkątami: środkowym, orto, i Eulera w ΔABC.

Przykłady problemów (cd3)

Problemy obliczeniowe

- Mając deskryptory katalogowe *n* książek uporządkować je wg autora i wg roku wydania.
- Znaleźć *nwp(n, m)* – największy wspólny podzielnik.
- Znaleźć najkrótszą trasę okrężną przez *n* zadanych miast w Polsce (problem komiwojażera).
- *n* punktów na płaszczyźnie połączyć najkrótszą siecią (problem Steinera).
- Znaleźć wszystkich podatników, którzy w latach 2000-2003 mieli ulgę budowlaną i dochody 100-200 tys. zł (wyszukiwanie zakresowe).
- Rozwiązać układ *n* równań liniowych z *n* niewiadomymi.
- Obliczyć oczekiwaną stopę zwrotu dla wybranych spółek giełdowych na podstawie ostatnich *n* miesięcy notowań.
- .....

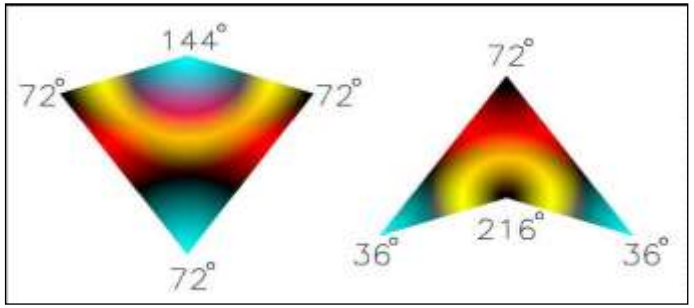
Jak rozpoznać problemy algorytmiczne ?

- Problem *jest* algorytmiczny jeżeli potrafimy zaproponować algorytm (możliwie sprawny) rozwiązania.
- Istnieją problemy *mające znamiona* algorytmicznych, dla których algorytm nie jest znany i nie wiemy czy istnieje (tak było przez ponad 2000 lat z klasycznymi problemami geometrii); problem tego rodzaju może okazać się nierozwiązywalny w przyjętym modelu obliczeń.
- Problemami *niealgorytmicznymi* są zagadnienia typu "podaj ogólny algorytm rozwiązywania problemów", czyli algorytm znajdowania algorytmów; dla pewnych szczególnych klas problemów takie metody mogą istnieć.
- Istnieje wiele zagadnień praktycznych (np. technologicznych), których rozwiązania polegają na odkrywaniu i stosowaniu zjawisk fizycznych. Np. jak łuskać nasiona słonecznika na skalę komercyjną?

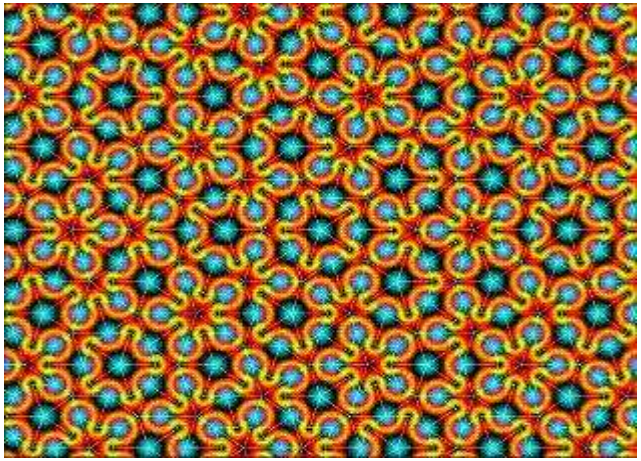
Przykład: problem niealgorytmiczny

Parkietaż aperiodyczny

Czy istnieje zestaw kształtów płytek taki, że można przy ich pomocy pokryć płaszczyznę aperiodycznie, to znaczy tak, że nie ma prostokąta (a, b), który nałożony regularnie na płaszczyznę zawierałby zawsze ten sam wzór.

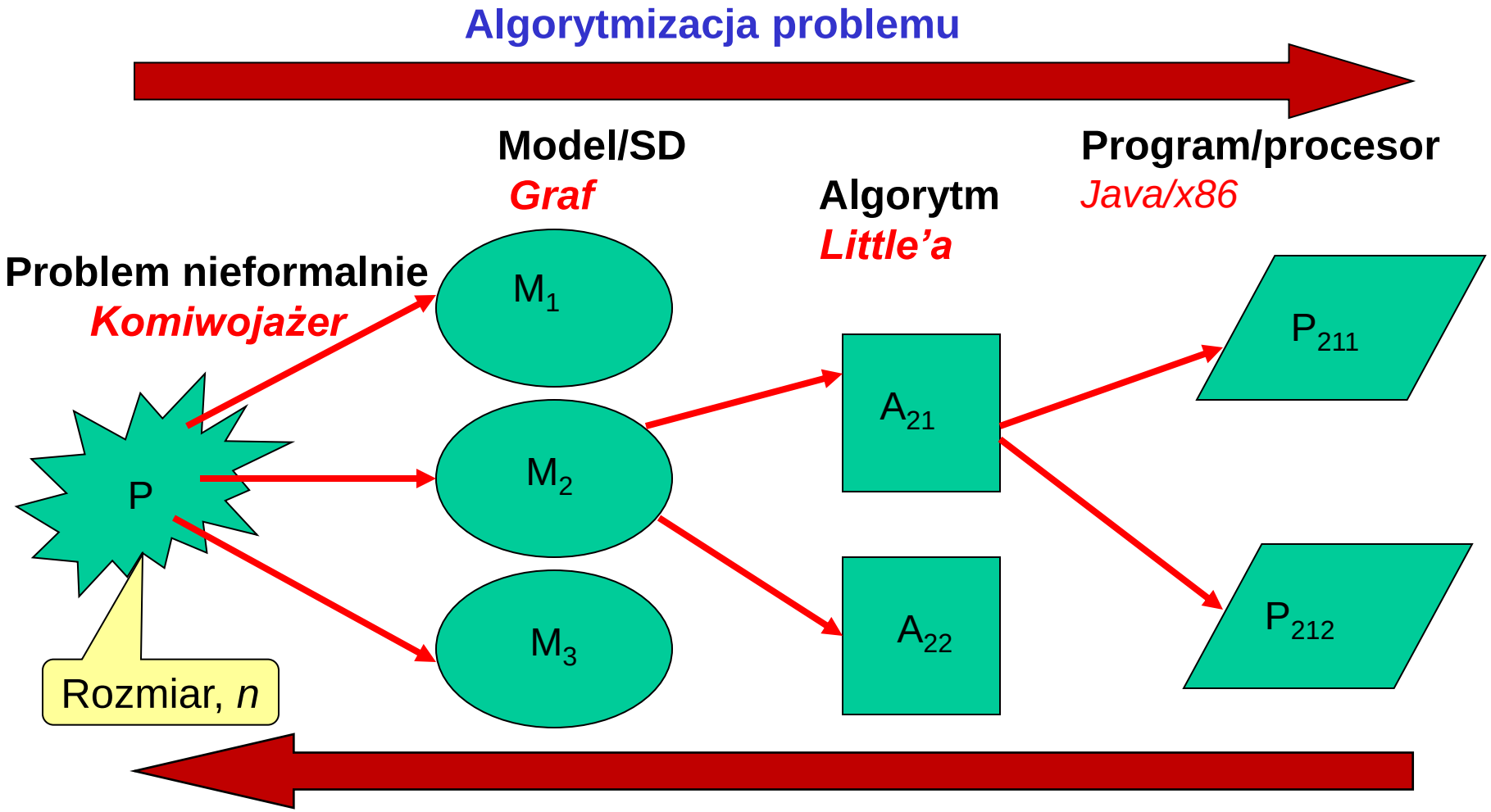


Płytki R. Penrose'a



- Berger 1966: 20426 płytek!
- Berger 1967: 104 płytki
- Robinson 1970: 6 płytek
- Penrose 1974: 2 płytki

Problem – model – algorytm – program



Wnioskowanie o złożoności problemu wg oceny czasu wykonania (najlepszy program → oszacowanie górne)

Rozmiar problemu

- **Rozumienie potoczne (najczęściej wystarcza)**

  Parametr całkowity, ***n***, charakteryzujący bezpośrednio lub pośrednio „ilość” informacji do przetworzenia i/lub wytworzenia w algorytmie; czasem stosuje się charakterystykę wielowymiarową, ***m, n, ...*** .

  W problemach decyzyjnych informacja wytwarzana = **1 bit.** Rozmiar problemu charakteryzuje zatem tylko wejście.

- **Przykłady**
  - sortuj ***n*** obiektów
  - wyznacz minimalną trasę okrężną przez ***n*** miast
  - wyznacz czynniki pierwsze liczby (nie większej od) *n*
  - rozwiąż układ ***n*** równań liniowych
  - czy formuła logiczna ***n*** zmiennych jest spełnialna

Kategorie złożoności

- **Złożoność czasowa / pamięciowa**
  - optymistyczna (małe znaczenie - przypadki „szczęśliwe”)
  - **pesymistyczna (najgorszy przypadek)**
  - oczekiwana (uśrednienie dla przypadków z pewnego rozkładu)
  - amortyzowana (inaczej, bilansowa w sekwencji operacji)
- **Przykład: operacja `add(e)` dla klasy `Vector<E>`**
  - optymistycznie koszt stały: symbolicznie *O(1)*
  - pesymistycznie koszt liniowy: *O(n)* - realokacja wektora
  - amortyzowany koszt stały: *O(1)* - przeliczony koszt jednej operacji wg bilansu kosztu sekwencji n operacji

Operacje podstawowe

- **W różnych modelach obliczeń - różne**
  - W modelu automatowym: wykonanie przejścia do następnego stanu
  - W modelu drzewa decyzyjnego: wykonanie operacji porównania pary obiektów (np. liczb, nazw); wg tego modelu określa się złożoność algorytmów sortowania
  - W modelu RAM (Random Access Machine): dowolna operacja arytmetyczna na zawartości jednej lub pary komórek maszyny); ale nie jest operacją podstawową sprawdzenie, czy zawartość komórki jest liczbą pierwszą.
  - W modelu edukacyjnym: uzyskanie przez studenta jednego punktu ECTS

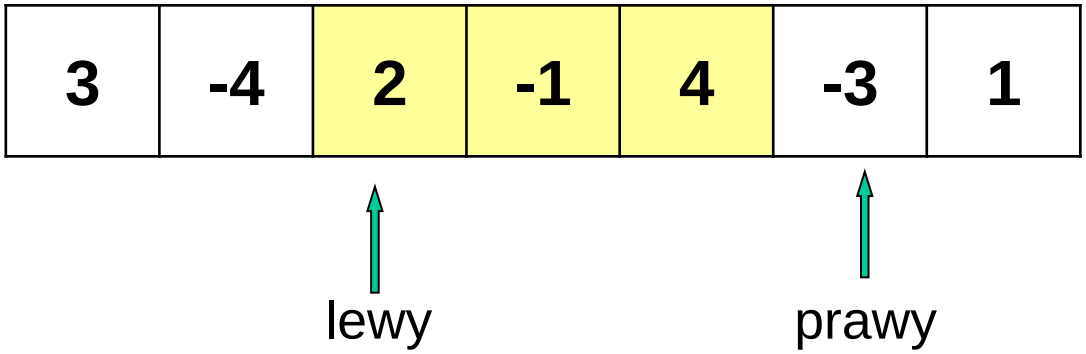
Przykład: problem maksymalnej podtablicy

- **Problem**

  Dana tablica *n* liczb (np. całkowitych): *X[n]*

  Znaleźć taki fragment tablicy X[lewy .. prawy-1], aby suma elementów była maksymalna **nieujemna**; jeśli wszystkie elementy X są niedodatnie wybór jest pusty (lewy==prawy)

  (Bentley: *Perełki oprogramowania*)



Rozwiązania

Naiwne: *koszt rzędu n^3, $O(n^3)$* *$n(n+1)(n+2)/6$ operacji*

Mniej naiwne: *koszt rzędu n^2, $O(n^2)$* *$n(n+1)/2$ operacji*

Przyzwoite: *koszt rzędu $n \lg(n)$, $O(n \lg(n))$*

Optymalne: *koszt liniowy $O(n)$*

Rozwiązanie optymalne osiąga **kres dolny złożoności** wynikający z oczywistego obowiązku wzięcia pod uwagę każdego elementu tablicy

Rozwiązanie mniej naiwne: $O(n^2)$

```
// Znajduje indeksy lewy, prawy takie, że suma elementów
// X[lewy..prawy-1] jest maksymalna, >=0.
// Zwraca indeksy i sumę w wynikowej tablicy 3 elementowej;
// lewy==prawy ==> podtablica jest pusta, smax = 0.

static int[] maxSum1(int X[]) {
  int smax = 0, n = X.length;
  int lewy = 0, prawy = 0; // Domniemanie: podtablica pusta

  for(int d=0; d<n; ++d){
    int suma = 0;
    for(int g=d; g<n; ++g){
      suma += X[g];
      if(smax < suma){
        smax = suma; lewy = d; prawy = g+1;
      }
    }
  }
  return new int[]{lewy, prawy, smax};
}
```

Rozwiązanie rekurencyjne: *O(n lg(n))*

```
static int maxSum2(int X[], int d, int g){ // koszt T(n)
  int maxL, maxP, maxA, maxB, suma;

  // Badanie przypadków bez rekursji
  if(d>=g) return 0;  // Pusta podtablica
  if(d == g-1)        // 1 element
    return Integer.max(X[d], 0);

  // Dekompozycja na dwa podproblemy połówkowe
  int m = (d+g)/2;    // indeks środka

  //Znajdź maks. na lewo od m;        koszt O(n)
  suma = maxL= 0;
  for(int i=m-1; i>=d; --i) {
    suma += X[i];
    maxL = Integer.max(suma, maxL);
  }                                          // ... cd
```

Rozwiązanie rekurencyjne *O(n lg(n)) (cd)*

```
  //Znajdź maks. na prawo od m;          koszt O(n)
  suma=maxP=0;
  for(int i=m; i<g; i++)
  { suma += X[i];
    maxP = Integer.max(suma, maxP);
  }

  maxA=maxSum2(X, d, m);// Rekursja w lewo, koszt T(n/2)
  maxB=maxSum2(X, m, g);// Rekursja w prawo,koszt T(n/2)
  return Integer.max(Integer.max(maxA, maxB),
                     maxL + maxP);
}
```

Czas wykonania opisuje równanie rekurencyjne:

$$T(n) = 2T(n/2) + O(n)$$

Rozwiązanie równania: $T(n) \in O(n \log n)$

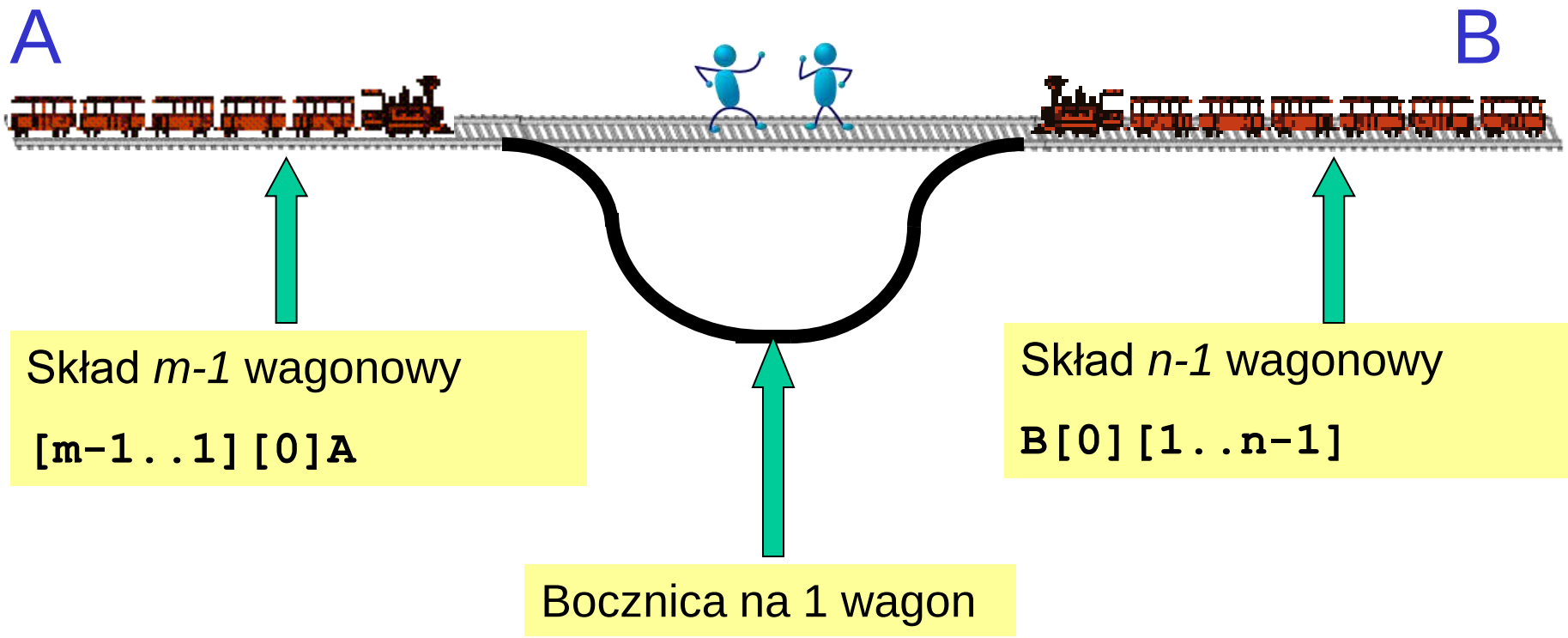
Rozwiązanie optymalne: *O(n)*

```
// Wersja optymalna (bez wyznaczania indeksów)
static int maxSumOpt(int X[])
{ int maxG = 0,  // Maksymalna suma globalna
       maxL = 0;  // Maksymalna suma lewostronnie

  for(int i=0; i<X.length; ++i)
  { // maxG, maxL: maksymalne sumy w
    // X[0..i-1], globalna, lewostronna
    maxL = Integer.max(0, maxL+X[i]);
    maxG = Integer.max(maxL, maxG);
  }
  return maxG;
}
```

Ćwiczenie: Uzupełnić funkcję o wyznaczanie położenia podtablicy (jak w maxSum1) i osadzić w klasie testującej z generacją losową.

Pożytki z analizy – przykład „kolejowy”



- **Jaką czynność obrać za operację podstawową (koszt jednostkowy)?**
- **Jaki jest koszt mijania pociągów dla wybranej operacji podstawowej?**
- **Jak rozstrzygnąć kłótnię maszynistów - kto ma wykonać pracę?**

Przykład „kolejowy” (cd1)

- **Operacja podstawowa (model „energetyczny”)**

  Przemieszczenie 1 wagonu albo lokomotywy z jednego położenia w drugie ma koszt jednostkowy. Dystans przemieszczenia jest obojętny; nie liczą się także inne czynności, jak rozpinanie / spinanie składu.

- **Oznaczenia**

`A, B`	pełne składy pociągów
`[j..i]A, B[i..j]`	fragmenty składów
`[j..i][k]A, B[k][i..j]`	zestawy nieprzyległe

- **Rozwiązanie #1**
  - Maszyniści uzgodnili, że wszelkie ruchy będzie wykonywała tylko lokomotywa składu B

```
for(i = 0; i<m; ++i)
  Ruch1(i);// Przemieść A[i] na właściwe miejsce
```

Rozwiązanie #1: Ruch1(i)

Strona A	Bocznica	Strona B	Koszt
[m-1 .. i]A		B[0..n-1] ·[i-1..0]A	--
[m-1 .. i]A · B[0]		B[1..n-1] ·[i-1..0]A	1
[m-1 .. i+1]A	[i]A·B[0]	B[1..n-1] ·[i-1..0]A	2
[n-1..i+1]A	[i]A	B[0..n-1] ·[i-1..0]A	1
[m-1..i+1]A·B[0..n-1]	[i]A	[i-1..0]A	n
[m-1 .. i+1]A		B[0..n-1] · [i..0]A	2n+1

$$T_1(m, n) = m(3n + 5) = 3mn + 5m \in O(mn)$$

manewrować powinien pociąg dłuższy

Rozwiązanie #1: (cd) - uwagi

- **Koszt $T_1(m, n) = 3mn + 5m$ jest pesymistyczny, obowiązuje dla m, n >2**
- **Dla m=n=1,** $T(1, 1) = 3$ ***wg wzoru 8***
- **Dla m=1, n>1,** $T(1, n) = n+2$ ***wg wzoru 3n+5***
- **Dla m>1, n=1,** $T(m, 1) = m+2$ ***wg wzoru 8m***
- **Usprawnienie dla pierwszego przemieszczenia:**

lokomotywa [0]A na bocznicę	**koszt 1**
skład B na lewo	**koszt n**
***lokomotywa* [0]A na prawo**	**koszt 1**
skład B powrót na prawo	**koszt n**

2n+2

- **Lepsze oszacowanie kosztu po usprawnieniu:**

$$T_1'(m, n) = 2n+2 + (m-1)(3n+5) = 3mn+5m-n-3$$

$$T_1'(m, n) \in O(mn)$$