PROJEKTOWANIE I ANALIZA ALGORYTMÓW

Wykład 1

dr inż. Łukasz Jeleń
Na podstawie wykładów dr. T. Fevensa

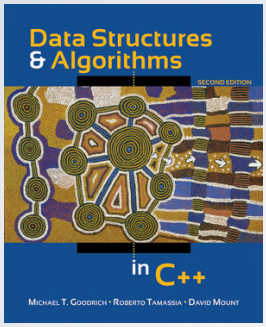
INFORMACJE PODSTAWOWE

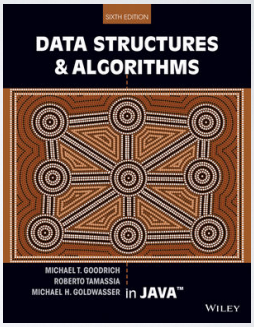
- Kontakt
  - e-mail: lukasz.jelen@pwr
  - E-portal
  - Pok. 230, c-3 w godzinach konsultacji

ZALICZENIE

- ***Ocena z przedmiotu:***
  - ***Egzamin (50%) + laboratorium (50%) - I termin***
    - ***Warunkiem koniecznym podejścia do egzaminu jest pozytywna ocena z laboratorium***
    - ***Egzamin -*** *test na e-portalu w sesji egzaminacyjnej - sala i czas: TBA*
    - ***Egzamin II termin -*** *w sesji poprawkowej*
      - *Wszystkie daty będą zgłoszone do dydaktyki oraz podane na e-portalu oraz stronie WIT*
  - ***Projekty -*** *szczegóły u prowadzących*

LITERATURA





LITERATURA

- LITERATURA PODSTAWOWA:

  [1] T.H. Cormen, C.E. Leiserson, R.L. Rivest, Wprowadzenie do algorytmów, WNT, W-wa.

  [2] N. Wirth. Algorytmy+struktury danych = Programy, WNT, W-wa

- LITERATURA UZUPEŁNIAJĄCA:

  [1] M.T. Goodrich, R. Tamassia, D. Mount, Data Structures and Algorithms, John Wiley and Sons, Inc., Hoboken, NJ, USA

  [2] Aho A.V., Hopcroft J.E., Ullman J.D.: Projektowanie i analiza algorytmów komputerowych. PWN, W-wa

PROBLEM VS. ALGORYTM

Co to jest?

- Ciasto:
  - 1 szklanka wody
  - 150g margaryny
  - 1 szklanka mąki pszennej
  - 5 jajek
  - szczypta soli
  - szczypta proszku do pieczenia

Mając dane wejściowe (składniki) mamy rozwiązać problem pieczenia ciasta

Jak?

PRZEPIS NA CIASTO

Przepis

Przygotuj	Piekarnik : 180º	Woda: 250ml	Olej: 83ml
Instrukcje	Podgrzej piekarnik do 180º. Posmaruj formę margaryną		
Ciasto	W wysokim naczyniu wymieszać zawartość torebki z wodą i olejem. Ubijać przez 2 minuty na wysokich obrotach		
Pieczenie	32x22 cm: 30 – 40 min.	ø 20 cm: 30 – 40 min.	Forma do babeczek: 15 – 20 min.

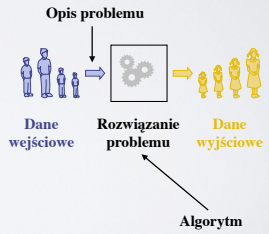
1. Przygotować ciasto. Wodę zagotować z margaryną. (…) po jednym jajku, szczyptę soli i proszek do pieczenia.
2. Ciasto podzielić na 2 części.
3. Formę prostokątną o wymiarach ok. 35x 24cm wysmarować margaryną i posypać mąką. Połowę ciasta rozprowadzić łyżką w formie.
4. Piec w nagrzanym piekarniku, na złoty kolor, ok. 30min. w temperaturze 180°C. W ten sam sposób upiec drugą połowę ciasta.
5. Przygotować masę budyniową. 2 szklanki mleka i cukier zagotować. (…) dodawać stopniowo zimny budyń.
6. Masę rozsmarować na jednym blacie ciasta. (…) Ciasto wstawić do lodówki, na co najmniej 2 godz.
7. Gotowe posypać cukrem pudrem.

PRZEPIS NA CIASTO A PROGRAM KOMPUTEROWY

- Przepis
  - daje nam specyfikację potrzebnych przedmiotów i przedstawia szczegółowy opis instrukcji ich użycia
    - opis składników i czynności potrzebnych do upieczenia ciasta.
    - rozwiązanie problemu pieczenia ciasta
- Dokładne wykonanie instrukcji skutkuje poprawnie upieczonym ciastem.
- Producent rozwiązał za nas problem pieczenia ciasta.
- Program jest zestawem instrukcji potrzebnych do poprawnego rozwiązania problemu.
- Pisząc program analizujemy problem i rozwiązujemy go tworząc przepis (program).

DEFINICJA PROBLEMU

- Zbiór danych wraz z poleceniem wykonania
  - dane wejściowe
  - definicja danych
  - opis problemu
    - pytanie
    - polecenie



PROGRAM KOMPUTEROWY (KOD) = STRUKTURA(Y) DANYCH + ALGORYTM(Y)

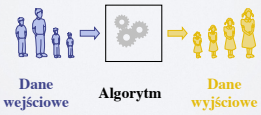
- Musimy znać podstawowe algorytmy i struktury danych aby móc tworzyć kod dobrej jakości
  - Przez „dobrą jakość" rozumieć będziemy program, który:
    - jest wydajny
    - rozwiązuje problem zgodnie z nałożonymi ograniczeniami sprzętowymi
- Nie można zostać dobrym programistą bez dobrej znajomości algorytmów i struktur danych

Struktury danych organizują informację

Algorytm przetwarza informację

ALGORYTM

- Jest zbiorem dobrze zdefiniowanych zasad niezbędnych do rozwiązania problemu.
- Przetwarza dane wejściowe problemu na dane wyjściowe, które ten problem rozwiązują.
- Problem może zawierać wiele algorytmów



ALGORYTM

- Właściwości:
  - Musi być poprawny
  - Musi się składać ze ściśle zdefiniowanych kroków
  - Kolejność kroków musi być ściśle określona
  - Musi się składać ze skończonej ilości kroków
  - Musi się zakończyć dla wszystkich danych wejściowych

Program komputerowy jest instancją lub konkretną reprezentacją algorytmu w dowolnym języku programowania

STRUKTURY DANYCH - DEFINICJE

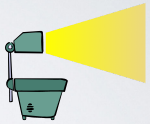
- **Typ** jest to zbiór wartości
  - np.: Integer, Boolean
- **Typ danych** jest typem i zbiorem operacji, które przetwarzają ten typ.
  - np.: Suma, Iloczyn
- **Dana** jest elementem typu danych - informacją
- Np.: Typy danych w C++:
  prosty: int, float, bool
  zagregowany: array, struct, vector, string
- Zagregowany typ danych jest przykładem struktury danych składający się z:
  - prostych pól
  - powiązań między polami
  - operacji na strukturze danych, które pozwalają na manipulację tych pól

STRUKTURY DANYCH

- W językach programowania, niektóre struktury danych są wbudowane (np.: tablice, łańcuchy znakowe)
- Wiele innych struktur danych jest często niezbędnych
  - możemy je zaimplementować z wykorzystaniem wbudowanych struktur danych
- Nazywamy je strukturami zdefiniowanymi przez użytkownika (user-defined)
  - STL dla C++
- Do zdefiniowania struktur danych w C++ wykorzystuje się klasy
- Na wykładzie poznamy struktury danych i algorytmy, które są najczęściej wykorzystywane w wielu aplikacjach

ABSTRACT DATA TYPE (ADT)

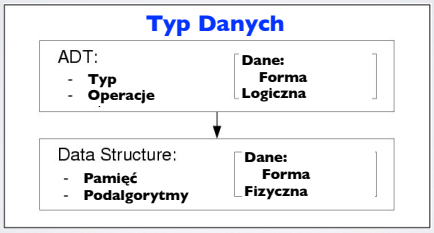
- Definicja typu danych tylko za pomocą wartości i operacji na tym typie danych
- Jest specyfikacją typu danych, która:
  - ukazuje istotne cechy
  - ukrywa detale implementacyjne



- Definiuje typ danych przez zależności **wejścia - wyjścia**
  - np.: każda operacja ADT jest zdefiniowana przez jej wejścia i wyjścia
- ADT radzą sobie ze złożonością poprzez abstrakcję: metaforę

STRUKTURA DANYCH VS. ADT

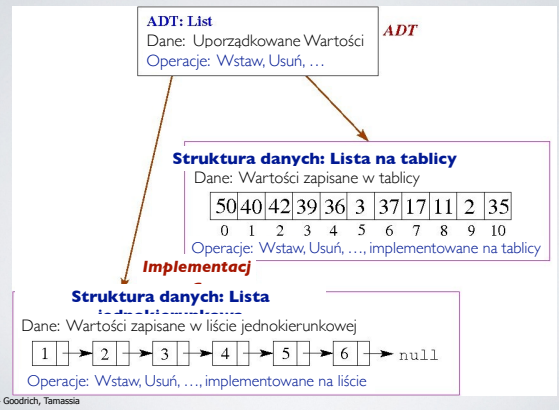
- Struktura danych jest konkretną implementacją ADT



WYBÓR IMPLEMENTACJI ADT

- Wszystkie struktury danych posiadają swoje wady i zalety.
- Bardzo rzadko jedna struktura danych jest lepsza od innych we wszystkich sytuacjach.
- Struktura danych wymaga:
  - przestrzeni dla każdej przechowywanej danej
  - czasu do przeprowadzenia każdej podstawowej operacji
  - wysiłku programistycznego.

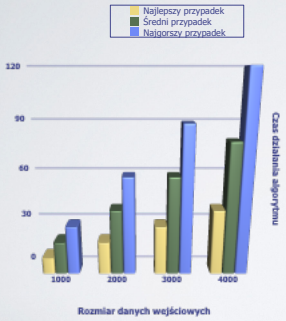
PRZYKŁAD: SD VS. ADT



WYBÓR IMPLEMENTACJI ADT

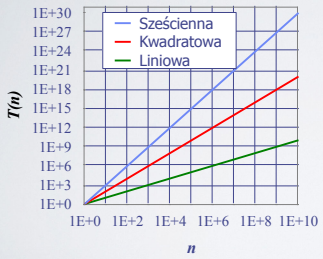
- Tylko po dokładnej analizie charakterystyki problemu możemy wybrać najlepszą strukturę danych dla danego zadania
- Przykład z banku:
  - Otwarcie konta: kilka minut
  - Transakcje: kilka sekund
  - Zamknięcie konta: doba

ZŁOŻONOŚĆ OBLICZENIOWA



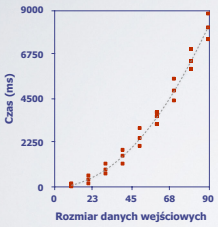
- Większość algorytmów przekształca obiekty wejściowe w obiekty wyjściowe
- Czas działania (złożoność obliczeniowa) algorytmu zazwyczaj wzrasta wraz z rozmiarem danych wejściowych
- Średni czas działania jest najczęściej trudny do określenia
- koncentrujemy się na przypadku najgorszym
  - łatwiejszy do analizy
  - Istotny w aplikacjach takich jak gry, finanse i robotyka

SIEDEM WAŻNYCH FUNKCJI



- Siedem funkcji często wykorzystywanych w analizie algorytmów:
  - Stała ≈ 1
  - Logarytmiczna ≈ log n
  - Liniowa ≈ n
  - N-Log-N ≈ n log n
  - Kwadratowa ≈ n^2
  - Sześcienna ≈ n^3
  - Wykładnicza ≈ 2^n
- Na wykresie log-log, nachylenie linii świadczy o wzroście funkcji

DOŚWIADCZENIA



- Napisz program implementujący algorytm
- Przetestuj napisany program na danych o różnych rozmiarach
- Wykorzystaj metodę typu System.currentTimeMillis() do dokładnego oszacowania czasu działania algorytmu
- Zrób wykres dla otrzymanych wyników.

OGRANICZENIE EKSPERYMENTÓW

- Niezbędne jest zaimplementowanie algorytmu, który może być trudny
- Wyniki złożoności obliczeniowej mogą nie być znaczące dla danych wejściowych, które nie były wykorzystywane w eksperymentach
- W celu porównania dwóch algorytmów należy korzystać z tego samego sprzętu i oprogramowania

ANALIZA TEORETYCZNA

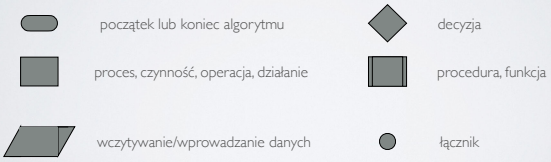
- Wykorzystuje formalną reprezentację algorytmu zamiast implementacji
- Charakteryzuje złożoność obliczeniową jako funkcję rozmiaru danych wejściowych, n
- Bierze pod uwagę wszystkie możliwe dane wejściowe
- Pozwala nam na ocenę szybkości działania algorytmu niezależnie od sprzętu/oprogramowania

DEFINICJA

- Złożoność obliczeniowa algorytmu A jest zdefiniowana przez:
  - t - czas - ilość operacji niezbędnych do rozwiązania dowolnej instancji I problemu o rozmiarze N(I) przez algorytm A => N(I) = n
  - $f_A(n) = max(t)$
- Nas interesuje jak wygląda funkcja F_A, a nie jej wartości

METODY REPREZENTACJI ALGORYTMÓW

- Pseudo kod
- Graficznie
  - Schematy blokowe



PSEUDO KOD

- Uogólniony opis algorytmu
- Bardziej strukturalny niż opis w języku polskim
- Mniej szczegółowy od programu komputerowego
- Preferowana notacja do opisu algorytmów
- Ukrywa aspekty projektowania programu

Przykład: znajdź element max tablicy

```
Algorytm tabMax(T, n)
  Wejście tab T zawierająca n
          integerów
  Wyjście element maksymalny T
  biezaceMax ← T[0]
  for i ← 1 to n – 1 do
    if T[i] > biezaceMax then
      biezaceMax ← T[i]
  return biezaceMax
```


DETALE PSEUDOCODU

- Kontrola działania
  - **if … then … [else …]**
  - **while … do …**
  - **repeat … until …**
  - **for … do …**
  - Wcięcia zastępują nawiasy
- Deklaracja metod
  - **Algorytm** *metoda (arg [, arg…])*

    Wejście …

    Wyjście
- Wywołanie metody

  zm.metoda (arg [, arg…])
- Zwracanie wartości

  return *wyrażenie*
- Wyrażenia

  ← Przypisanie (tak jak = w C++/Javie)

  = Testowanie równości (tak jak == w C++/Javie)

  n^2 Superskrypty i inne matematyczne formatowanie jest dozwolone

OPERACJE PODSTAWOWE

- Podstawowe obliczenia są wykonywane przez algorytm
- Identyfikowane w pseudokodzie
- Niezależne od języka programowania
- Dokładna definicja nie jest istotna (później zobaczymy dlaczego)
- Z założenia pobierają stałą ilość pamięci oraz wykonywane są w ściśle określonym czasie

- Przykłady:
  - Wykonywanie wyrażeń
  - Przypisanie wartości do zmiennej
  - Indeksowanie tablicy
  - Wywołanie metody
  - Powrót z metody

ZLICZANIE OPERACJI PODSTAWOWYCH

- Badając pseudokod możemy określić maksymalną ilość operacji podstawowych wykonywanych przez algorytm w funkcji n - rozmiaru danych wejściowych

Algorytm tabMax(T, n)	il. operacji
biezacyMax ← T[0]	2
for i ← 1 to n – 1 do	2n
if T[i] > biezacyMax then	2(n – 1)
biezacyMax ← T[i]	2(n – 1)
{ zwiększanie licznika i }	2(n – 1)
return biezacyMax	1
	Suma 8n – 3

OKREŚLANIE ZŁOŻONOŚCI OBLICZENIOWEJ - CZASU DZIAŁANIA ALGORYTMU

- Algorytm tabMax wykonuje $8n - 3$ operacji podstawowych w najgorszym przypadku. Zdefiniujmy:

  a = Czas wykonania najszybszej operacji podstawowej

  b = Czas wykonania najwolniejszej operacji podstawowej

- Niech **T(n)** będzie najgorszym czasem tabMax. Wtedy

  $$a(8n - 3) \leq T(n) \leq b(8n - 3)$$

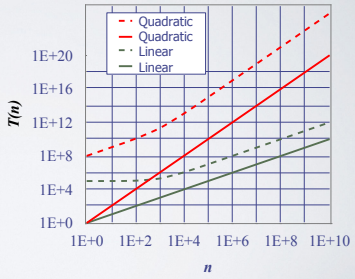
- Zatem, czas T(n) jest ograniczony przez dwie funkcje liniowe

WSPÓŁCZYNNIK WZROSTU ZŁOŻONOŚCI OBLICZENIOWEJ

- Zmiana środowiska sprzętowego/oprogramowania
  - Ma stały wpływ na T(n), ale
  - nie ma wpływu na współczynnik wzrostu T(n)
- Liniowy wzrost czasu działania T(n) jest istotną właściwością algorytmu tabMax

SKŁADOWA STAŁA

- Asymptotyczny współczynnik wzrostu nie zależy od:
  - składowych stałych lub
  - wyrażeń niższego rzędu
- Przyłady
  - $10^2n + 10^5$ jest funkcją liniową
  - $10^5n^2 + 10^8n$ jest funkcją kwadratową

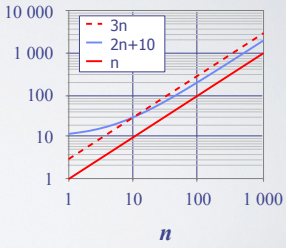


NOTACJA DUŻE O

- Mając daną funkcję **f(n)** i **g(n)** mówimy, że **f(n)** należy do **O(g(n))** jeśli istnieją stałe nieujemne **c** i $\mathbf{n_0}$ takie, że

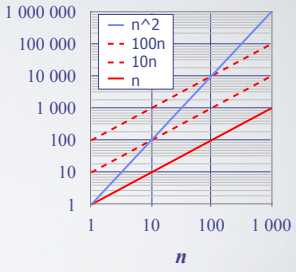
  $\mathbf{f(n) \leq cg(n)}$ dla $\mathbf{n \geq n_0}$

- Przykład: $2n + 10$ jest w **O(n)**
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Weźmy $c = 3$ i $n_0 = 10$



PRZYKŁAD DUŻEGO O

- Przykład: funkcja n^2 nie należy do O(n)
  - $n^2 \leq cn$
  - $n \leq c$
  - Powyższa nierówność nie może zostać spełniona ponieważ c musi być stałe



WIĘCEJ PRZYKŁADÓW

- $7n-2$

  7n-2 jest w O(n)
  potrzebujemy $c > 0$ i $n_0 \geq 1$ takie, że $7n-2 \leq c \bullet n$ dla $n \geq n_0$
  spełnione dla $c = 7$ i $n_0 = 1$

- $3n^3 + 20n^2 + 5$

  $3n^3 + 20n^2 + 5$ jest w $O(n^3)$
  potrzebujemy $c > 0$ i $n_0 \geq 1$ takie, że $3n^3 + 20n^2 + 5 \leq c \bullet n^3$ dla $n \geq n_0$
  spełnione dla $c = 4$ i $n_0 = 21$

- $3 \log n + 5$

  $3 \log n + 5$ jest w $O(\log n)$
  potrzebujemy $c > 0$ i $n_0 \geq 1$ takie, że $3 \log n + 5 \leq c \bullet \log n$ dla $n \geq n_0$
  spełniony dla $c = 8$ i $n_0 = 2$

DUŻE O I WSPÓŁCZYNNIK WZROSTU

- Notacja duże O daje nam górne ograniczenie współczynnika wzrostu funkcji.
- Określenie "f(n) jest w O(g(n))" oznacza, że współczynnik wzrostu funkcji f(n) jest nie większy niż współczynnik wzrostu funkcji g(n)
- Możemy wykorzystać notację duże O do porównywania (stopniowania) funkcji względem ich współczynnika wzrostu

$f(n)$ jest w $O(g(n))$	$g(n)$ jest w $O(f(n))$	
Tak	Nie	$g(n)$ rośnie szybciej
Nie	Tak	$f(n)$ rośnie szybciej
Tak	Tak	ten sam wzrost

ZASADY NOTACJI DUŻE O

- Jeśli f(n) jest wielomianem stopnia d, np.:

  $f(n) = c_d n^d + c_{d-1} n^{d-1} + \ldots + c_1 n^1 + c_0 n^0$, *to f(n) jest w* $O(n^d)$, np.:

  1. Pomiń wyrażenia niskiego stopnia
  2. Pomiń stałe
- Wykorzystaj najmniejszą możliwą klasę funkcji
  - Powiemy *"2n jest w O(n)"* zamiast *"2n jest w O(2n)"*
- Wykorzystaj najprostsze wyrażenie tej klasy
  - Powiemy *"3n + 5 jest w O(n)"* zamiast *"3n + 5 jest w O(3n)"*