PROJEKTOWANIE I ANALIZA ALGORYTMÓW

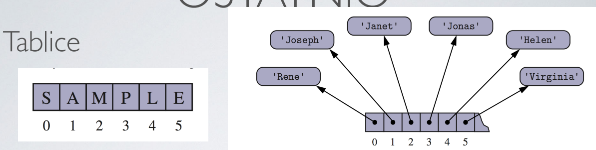
KOLEJKA, KOLEJKA PRIORYTETOWA CZ.I, OPIS ALGORYTMÓW, ANALIZA ALGORYTMÓW CZ.I

Wykład 3

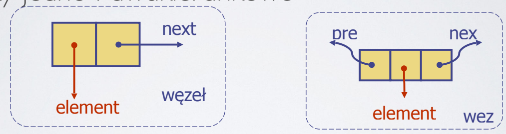
dr inż. Łukasz Jeleń
Na podstawie wykładów dr. T. Fevensa

OSTATNIO

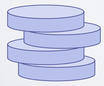
- Tablice



- Listy jedno i dwukierunkowe



- Stos

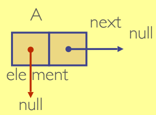


Implementacja:
- na tablicy
- na liście jednokierunkowej

REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

Node<String> A = new Node<String>();

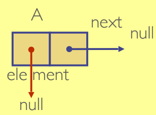


REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

Node<String> A = new Node<String>();

A->setElement(new String("fred"));

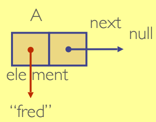


REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

Node<String> A = new Node<String>();

A->setElement(new String("fred"));

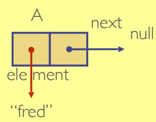


REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

Node<String> A = new Node<String>();

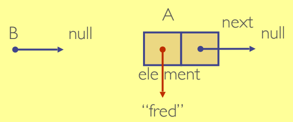
A->setElement(new String("fred"));
Node<String> B = null;



REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

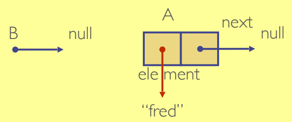
```
Node<String> A = new Node<String>();
A->setElement(new String("fred"));
Node<String> B = null;
```



REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

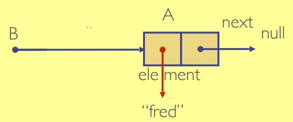
```
Node<String> A = new Node<String>();
A->setElement(new String("fred"));
Node<String> B = null;
B = A;  // B jest referencją do obiektu A
```



REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

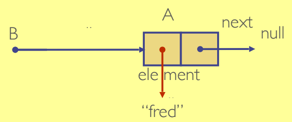
```
Node<String> A = new Node<String>();
A->setElement(new String("fred"));
Node<String> B = null;
B = A;  // B jest referencją do obiektu A
```



REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

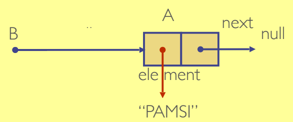
```
Node<String> A = new Node<String>();
A->setElement(new String("fred"));
Node<String> B = null;
B = A;  // B jest referencją do obiektu A
A->setElement(new String("PAMSI"));
```



REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

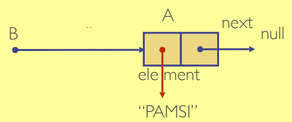
```
Node<String> A = new Node<String>();
A->setElement(new String("fred"));
Node<String> B = null;
B = A;  // B jest referencją do obiektu A
A->setElement(new String("PAMSI"));
```



REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

```
Node<String> A = new Node<String>();
A->setElement(new String("fred"));
Node<String> B = null;
B = A;  // B jest referencją do obiektu A
A->setElement(new String("PAMSI"));
B->setElement(new String("Montréal"));
```

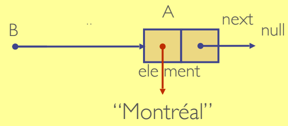


REFERENCJE

Krótkie przypomnienie referencji (t.j., śledzenie powiązań lub wskaźników).

Node<String> A = new Node<String>();

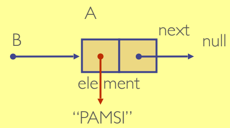
A->setElement(new String("fred"));
Node<String> B = null;
B = A; // B jest referencją do obiektu A
A->setElement(new String("PAMSI"));
B->setElement(new String("Montréal"));



REFERENCJE II

Możemy także śledzić powiązania…

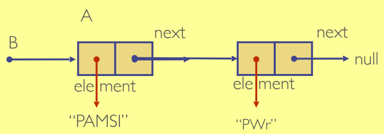
A->setNext(new Node<String>("PWr",null)); // konstruktor



REFERENCJE II

Możemy także śledzić powiązania…

A->setNext(new Node<String>("PWr",null)); // konstruktor

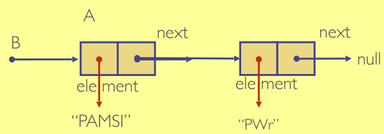


REFERENCJE II

Możemy także śledzić powiązania…

A->setNext(new Node<String>("PWr",null)); // konstruktor

B.getNext().setElement(new String("Wykład"));

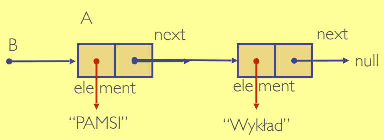


REFERENCJE II

Możemy także śledzić powiązania…

A->setNext(new Node<String>("PWr",null)); // konstruktor

B.getNext().setElement(new String("Wykład"));

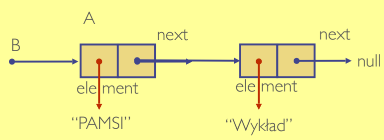


REFERENCJE II

Możemy także śledzić powiązania…

A->setNext(new Node<String>("PWr",null)); // konstruktor

B.getNext().setElement(new String("Wykład"));
B = B.getNext();

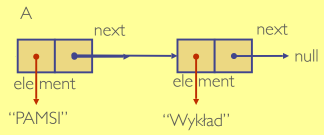


REFERENCJE II

Możemy także śledzić powiązania…

A->setNext(new Node<String>(“PWr”,null)); // konstruktor

B.getNext().setElement(new String(“Wykład”));
B = B.getNext();

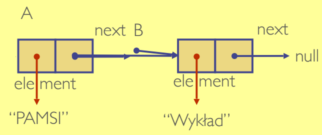


REFERENCJE II

Możemy także śledzić powiązania…

A->setNext(new Node<String>(“PWr”,null)); // konstruktor

B.getNext().setElement(new String(“Wykład”));
B = B.getNext();

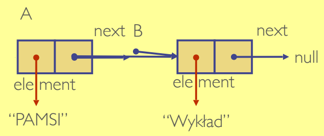


REFERENCJE II

Możemy także śledzić powiązania…

A->setNext(new Node<String>(“PWr”,null)); // konstruktor

B.getNext().setElement(new String(“Wykład”));
B = B.getNext();
B.setNext(new Node<String>(“Laboratorium”,null)); ;

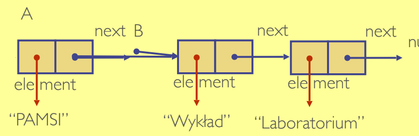


REFERENCJE II

Możemy także śledzić powiązania…

A->setNext(new Node<String>(“PWr”,null)); // konstruktor

B.getNext().setElement(new String(“Wykład”));
B = B.getNext();
B.setNext(new Node<String>(“Laboratorium”,null)); ;



KOLEJKI

KOLEJKA ADT

- Kolejka przechowuje dowolne obiekty
- Dodawanie i usuwanie jest wykonywane według zasady FIFO - first-in first-out
- Elementy dodawane są na końcu kolejki, a usuwane z przodu kolejki
- Główne operacje na kolejce:
  - enqueue(element): dodaje element na końcu kolejki
  - element dequeue(): usuwa i zwraca element z początku kolejki

- Dodatkowe operacje:
  - element front(): zwraca element na przodzie listy bez usuwania go
  - integer size(): zwraca ilość przechowywanych elementów
  - boolean isEmpty(): informuje czy w kolejce są przechowywane jakieś elementy
- Wyjątki
  - Próba wywołania dequeue lub front na pustej kolejce wyrzuca EmptyQueueException

ZASTOSOWANIA KOLEJEK

Zastosowanie bezpośrednie
- Listy oczekujących,
- Dostęp do zasobów współdzielonych (np.:, drukarka),
- Multiprogramming

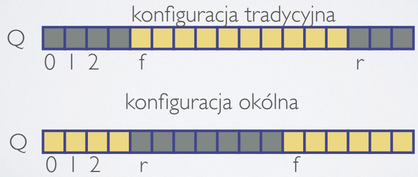
Zastosowania pośrednie
- Pomocnicza struktura danych dla algorytmów
- Składowa innych struktur danych

KOLEJKA BAZUJĄCA NA TABLICY

Zastosowanie tablicy o rozmiarze N w sposób okrężny/kolisty

Dwie zmienne kontrolują przód i tył kolejki
- f indeks pierwszego elementu
- r indeks następny do ostatniego elementu

Pozycja r w tablicy jest pusta

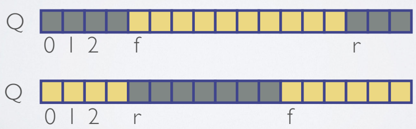


OPERACJE NA KOLEJCE

Wykorzystujemy operator modulo

Algorytm size()
  return (N − f + r) mod N

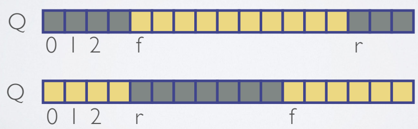
Algorytm isEmpty()
  return (f = r)



OPERACJE NA KOLEJCE

Operacja enqueue wyrzuca wyjątek jeśli tablica jest pełna
Wyjątek jest zależny od implementacji

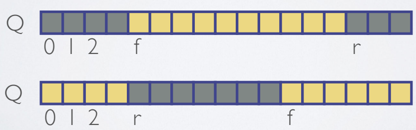
Algorytm enqueue(e)
  if size() = N − 1 **then**
    throw FullQueueException
  else
    Q[r] ← e
    r ← (r + 1) mod N



OPERACJE NA KOLEJCE

Operacja dequeue wyrzuca wyjątek jeśli kolejka jest pusta
Ten wyjątek jest określony w ADT dla kolejki

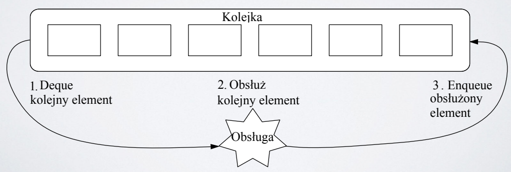
Algorytm dequeue()
  if isEmpty() **then**
    throw EmptyQueueException
  else
    temp ← Q[f]
    f ← (f + 1) mod N
    return temp



ZASTOSOWANIE : SYMULATORY SYSTEMU KOŁOWEGO

Możemy zaimplementować symulator systemu kołowego (tzw. systemu każdy z każdym) z zastosowaniem kolejki, Q, poprzez wielokrotne wywoływanie następujących kroków:

1. e ← Q.dequeue()
2. Obsługa elementu e
3. Q.enqueue(e)

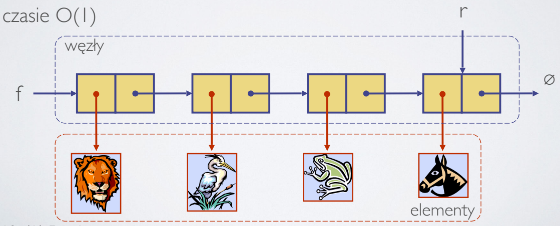


KOLEJKA BAZUJĄCA NA POWIĘKSZANEJ TABLICY

- Wykonując operację enqueue, kiedy tablica jest pełna, zamiast wyrzucania wyjątku, możemy możemy zastąpić ją większą tablicą
- Analogicznie do procedury, którą omawialiśmy w przypadku stosu
- Operacja enqueue ma średni czas działania:
  - *O(n)* w przypadku strategii inkrementalnej
  - *O(1)* w przypadku strategii podwajającej

KOLEJKA NA LIŚCIE JEDNOKIERUNKOWEJ

- Możemy zaimplementować kolejkę za pomocą listy jednokierunkowej
  - Element początkowy jest przechowywany w pierwszym węźle
  - Ostatni element jest przechowywany w ostatnim węźle
- Wykorzystane miejsce to O(n), a każda operacja kolejki jest wykonywana w czasie O(1)



KOLEJKI PRIORYTETOWE

KOLEJKA PRIORYTETOWE - ADT

- Kolejka priorytetowa przechowuje kolekcję wpisów
- Każdy element jest parą (klucz, wartość)
- Główne metody kolejki priorytetowej
  - insert(k, x)
    dodaje element o kluczu k i wartości x
  - removeMin()
    usuwa i zwraca element o najmniejszym kluczu
- Dodatkowe metody:
  - min()
    zwraca, ale nie usuwa, element o najmniejszym kluczu
  - size(), isEmpty()
- Zastosowania:
  - Aukcje
  - Giełda papierów wartościowych

RELACJE UPORZĄDKOWANIA

- Klucze w kolejce priorytetowej mogą być dowolnymi obiektami, na których podstawie da się zdefiniować uporządkowanie
- Dwa różne wpisy w kolejce priorytetowej mogą posiadać ten sam klucz.
- Matematyczna koncepcja całkowitego uporządkowania $\leq$
  - Właściwość refleksyjna:
    $x \leq x$
  - Właściwość antysymetryczna:
    $x \leq y \land y \leq x \Rightarrow x = y$
  - Właściwość tranzytywna:
    $x \leq y \land y \leq z \Rightarrow x \leq z$

KOMPARATOR

- Komparator porównuje dwa obiekty zgodnie z koncepcją całkowitego uporządkowania
- Uogólniona postać kolejki priorytetowej wykorzystuje komparator
  - definicja sposobu porównywania obiektów
- Komparator jest niezależny od przechowywanych kluczy
  - takie same obiekty mogą zostać posortowane w różny sposób
    - zależny od komparatora

ZASTOSOWANIE KOMPARATORA W C++

- Klasa komparatora przeciąża operator "()" funkcją porównującą
- Przykład: Porównaj leksykograficznie dwa punkty na płaszczyźnie

```
class LexCompare{
public:
    int operator()(Point a, Point b){
        if      (a.x < b.x) return -1
        else if (a.x > b.x) return 1
        else if (a.y < b.y) return -1
        else if (a.y > b.y) return 1
        else return 0;
    }
};
```

- W celu wykorzystania komparatora należy zdefiniować obiekt tego typu i wywołać jego operator "()"
- Przykład:

```
Point p(2.3, 4.5);
Point q(1.7, 7.3);
LexCompare lexCompare;
if (lexCompare(p,q) < 0)
    cout<< "p jest mniejsze od q";
else if (lexCompare(p,q) == 0)
    cout<< "p jest równe q";
else if (lexCompare(p,q) > 0)
    cout<< "p jest większe od q";
```


SORTOWANIE Z ZASTOSOWANIEM KOLEJEK PRIORYTETOWYCH

- Możemy wykorzystać kolejkę priorytetową do posortowania zbioru porównywalnych elementów
  1. Pojedynczo umieść elementy w kolejce
  2. Usuń elementy z wykorzystaniem serii operacji removeMin
- Złożoność obliczeniowa takiego sortowania jest zależna od implementacji kolejki priorytetowej

Algorytm *PriorityQueueSort*(*S, P*)
  Wejście: sekwencja ***S***, kolejka priorytetowa ***P*** wykorzystująca metodę całkowitego uporządkowania kluczy
  Output: posortowana sekwencja ***S*** z zastosowaniem metody całkowitego uporządkowania
  while !***S.isEmpty***() **do**
    e ← ***S.removeFirst*** ()
    P.insert (***e***, ***null***)
  while !***P.isEmpty***() **do**
    e ← ***P.removeMin***().***getKey***()
    S.addLast(***e***)

KOLEJKA BAZUJĄCA NA LIŚCIE

- Implementacja z wykorzystaniem nieposortowanej listy

4 — 5 — 2 — 3 — 1

- Wydajność:
- umieszczanie elementów zajmuje O(1)
  - możemy umieszczać elementy na początku i na końcu
- removeMin i min zajmują O(n)
  - musimy przeskanować całą listę w celu odnalezienia najmniejszego klucza

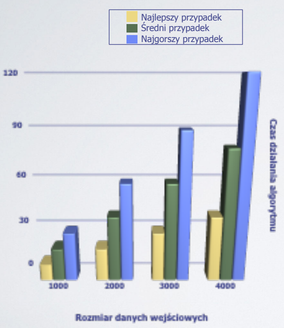
- Implementacja z wykorzystaniem posortowanej listy

1 — 2 — 3 — 4 — 5

- Wydajność:
- umieszczanie elementów zajmuje O(n)
  - musimy znaleźć miejsce gdzie możemy dodać nowy element
- removeMin i min zajmują O(1)
  - najmniejszy element znajduje się na początku listy

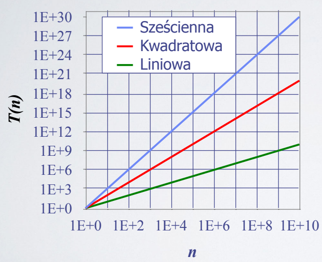
ZŁOŻONOŚĆ OBLICZENIOWA

ZŁOŻONOŚĆ OBLICZENIOWA



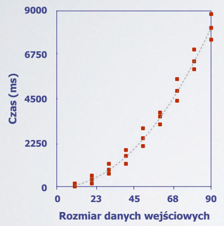
- Większość algorytmów przekształca obiekty wejściowe w obiekty wyjściowe
- Czas działania (złożoność obliczeniowa) algorytmu zazwyczaj wzrasta wraz z rozmiarem danych wejściowych
- Średni czas działania jest najczęściej trudny do określenia
- koncentrujemy się na przypadku najgorszym
  - łatwiejszy 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

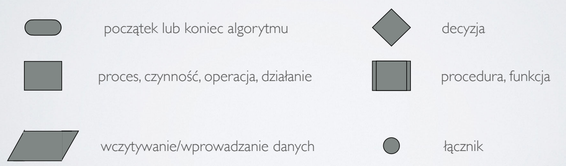
- Wykorzystuje formalną reprezentację algorytmu zamiast implementacji
- Charakteryzuje złożoność obliczeniową jako funkcję rozmiaru danych wejściowych, n
- Bierze pod uwagę wszystkie możliwe dane wejściowe
- Pozwala nam na ocenę szybkości działania algorytmu niezależnie od sprzętu/oprogramowania

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)

  n² 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

	il. operacji
Algorytm *tabMax*(*T*, *n*)	
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:

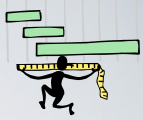
  a = Czas wykonania najszybszej operacji podstawowej

  b = Czas wykonania najwolniejszej operacji podstawowej
- Niech **T(n)** będzie najgorszym czasem tabMax. Wtedy

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

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

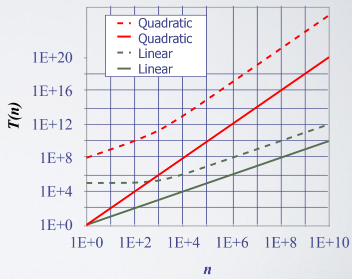
WSPÓŁCZYNNIK WZROSTU ZŁOŻONOŚCI OBLICZENIOWEJ



- 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 **n_0** takie, że

  f(n) $\leq$ **cg(n)** dla **n** $\geq$ **n_0**

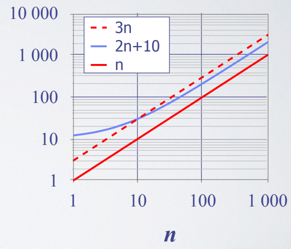
- Przykład: 2n + 10 jest w **O(n)**
  - $2n + 10 \leq cn$
  - $(c - 2)\, n \geq 10$
  - $n \geq 10/(c - 2)$
  - Weźmy $c = 3$ i $n_0 = 10$

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 **n_0** takie, że

  f(n) $\leq$ **cg(n)** dla **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 $n_0 = 10$



© 2004 Goodrich, Tamassia

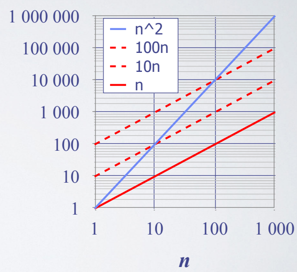
PRZYKŁAD DUŻEGO O

- 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

© 2004 Goodrich, Tamassia

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



© 2004 Goodrich, Tamassia

WIĘCEJ PRZYKŁADÓW

- 7n-2
- $3n^3 + 20n^2 + 5$
- 3 log n + 5

© 2004 Goodrich, Tamassia

WIĘCEJ PRZYKŁADÓW

- 7n-2

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

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

© 2004 Goodrich, Tamassia

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

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

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

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

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. Pominń 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)"*