Algorytmy i struktury danych

Złożoność obliczeniowa algorytmów
Techniki projektowania algorytmów

Witold Marańda
maranda@dmcs.p.lodz.pl

1

Złożoność obliczeniowa – miara efektywności algorytmu

Złożoność obliczeniowa wyraża zależność czasu „t" wykonywania algorytmu (tj. liczby operacji potrzebnych do wykonania algorytmu) od liczby elementów „n" zbioru (n), na którym działa dany algorytm.

$$t = f(n)$$

Ponieważ dokładna postać tej zależności może być skomplikowana, zwykle rozważa się jedynie jej przybliżenie, tzw. jest postać asymptotyczną

$$f(n) = n^2 + 100n + \log_{10}n + 1000$$

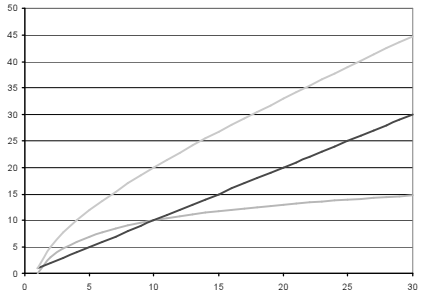
$$n \to \infty$$

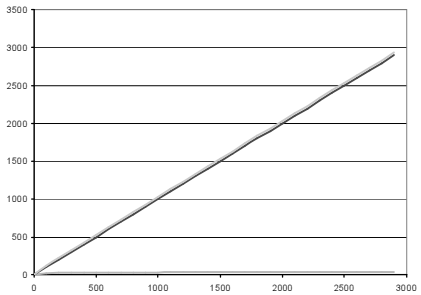
$$f(n) = n^2$$

2

Postać asymptotyczna

**Dla n dążącego do nieskończoności, funkcja f(n)
dąży asymptotycznie do wartości dominującego składnika**



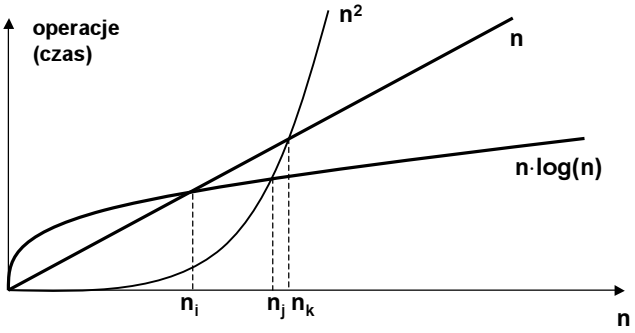


$f(n) = n + 10 \cdot \log_{10}(n)$ $\quad n \to \infty \quad$ $f(n) = n$

Dominujący składnik

Wpływ dominacji postaci asymptotycznej uwidacznia się dla dopiero „odpowiednio dużych" wartości n

Dla „mniejszych" wartości n, algorytm może wykazywać inne własności – dobór algorytmów powinien dotyczyć przewidywanych zakresów zastosowań.



$f(n) = n^2 + n + n \cdot \log(n)$

Postać asymptotyczna - przykład

$f(n) = n^2 + 100n + \log_{10}n + 1000$

n	f(n)	n^2	100n	$\log_{10}n$	1000
1	1101	0.1%	9%	$\rightarrow$ 0%	91%
10	2101	4.8%	48%	0.05%	48%
100	21002	48%	48%	0.001%	4.8%
10^3	1101003	91%	9%	0.0003%	0.09%
10^4		99%	1%	$\rightarrow$ 0%	0.001%
10^5		99.9%	0.1%	$\rightarrow$ 0%	$\rightarrow$ 0%

$f(n>10^5) \approx n^2$ (błąd < 0.1%)

5

Notacja Θ

Definicja:

Funkcja f(n) jest rzędu $\Theta(g(n))$ jeśli istnieją liczby dodatnie c i N takie, że $f(n) < c \cdot g(n)$ dla wszystkich $n \geq N$

Funkcja g(n) jest złożonością obliczeniową w sensie $\Theta()$.

Notacja $\Theta()$ jest tzw. pesymistyczną oceną złożoności obliczeniowej – opisuje złożoność najgorszego przypadku (tj. jest to ograniczenie górne funkcji f(n) dla wszystkich możliwych przypadków danych dla algorytmu.)

Np. algorytm przeszukiwania liniowego: $\Theta(n)$, ale możemy znaleźć element już w pierwszym kroku algorytmu.

6

Niektóre własności notacji Θ

$f(n)=c \cdot g(n)$ jest $\Theta(g(n))$

np. $1000n^4$ jest $\Theta(n^4)$

$f(n)=n^k$ jest $\Theta(n^{k+j})$

np. $7n^4 - 3n^3 + 4n^2 - 12n + 102$ jest $\Theta(n^4)$

$f(n)=\log_a n$ jest $\Theta(\log_2 n)$

np. $4\log_{10} n$ jest $\Theta(\log_2 n)$

$f(n)$ jest $\Theta(g(n))$ i $g(n)$ jest $\Theta(h(n)) \Rightarrow f(n)$ jest $\Theta(h(n))$

np. $\log_{10}(n^3+n^2+n)$ jest $\Theta(\log_{10} n^3)$ i $\log_{10} n^3$ jest $\Theta(\log_2 n^3)$
$\Rightarrow \log_{10}(n^3+n^2+n)$ jest $\Theta(\log_2 n^3)$

$f(n)$ jest $\Theta(h(n))$ i $g(n)$ jest $\Theta(h(n)) \Rightarrow f(n)+g(n)$ jest $\Theta(h(n))$

np. $(5n^2 - 7n - 2) + (0.5n^2 - \log_{10} n)$ jest $\Theta(n^2)$

7

Wybrane algorytmy

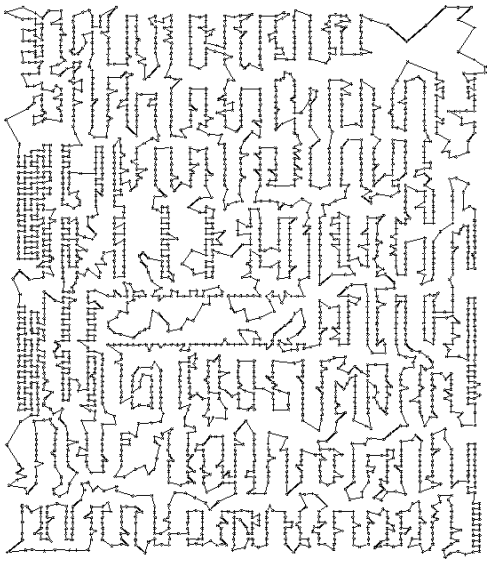
Przeszukiwanie binarne:	$\Theta(\log(n))$
Przeszukiwanie liniowe:	$\Theta(n)$
Sortowanie stogowe:	$\Theta(n \cdot \log(n))$
Sortowanie proste:	$\Theta(n^2)$
…	
„Wieża w Hanoi”:	$\Theta(2^n)$
„Problem komowojażera” (TSP):	$\Theta(n!)$

8

Problem komiwojażera (TSP)

Zastosowanie: optymalizacja trasy wiercenia otworów w płytka drukowanych (PCB)

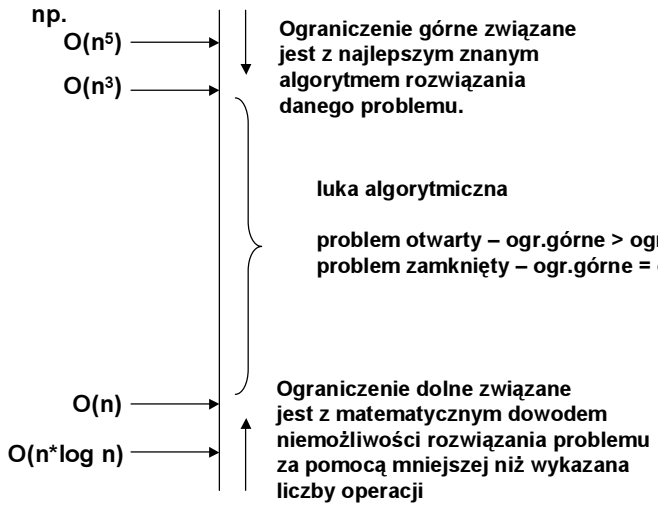
Przykład z roku 1998:
Optymalna trasa przez 3038 punktów.
Wykorzystano iteracyjny algorytm poszukiwań, który daje rozwiązanie z założoną dokładnością (np. nie gorszą niż 5% od optimum).
Do rozwiązania wykorzystano jeden 400 MHz Digital AlphaServer 4100, czas obliczeń 1.5 dnia.



http://www.cs.rutgers.edu/~chvatal/pcb3038.html

9

Ograniczenia złożoności obliczeniowej



10

Złożoność rozsądna i nierozsądna

1	stała
log n	logarytmiczna
n	liniowa
n*log n	liniowo-logarytmiczna
n^2	kwadratowa
n^k	wielomianowa
2^n	wykładnicza
n!	silnia
n^n	...

O(•)	10	50	100	300	1000
n log n	33	282	665	2469	9966
n^2	100	2500	10,000	90,000	1,000,000
n^3	1000	125000	1,000,000	27mln	1mld (10-cyfr)
2^n	1024	16-cyfr	31-cyfr	91-cyfr	☺
n!	3,6mld	65-cyfr	161-cyfr	☺	☺
n^n	10mld	85-cyfr	201-cyfr	☺	☺

dla porównania: liczba protonów we wszechświecie – 126-cyfr
liczba mikrosekund od powstania wszechświata – 24 cyfry

11

Złożoność obliczeniowa - przykład

Sortowanie zbioru n=1,000,000
(słowniki, książki telefoniczne, bazy danych)

Proste metody sortowania $O(n^2)$ (wstawianie, wybieranie, bąbelkowe)

Sprzęt	Czas
1mln op/s	⇒ 12 dni
100,000 op/s	⇒ 4 miesiące
10,000 op/s	⇒ 3 lata

Zaawansowane metody sortowania O(n*log(n)) (stogowe, przez podział)

Sprzęt	Czas
1mln op/s	⇒ 6 s
100,000 op/s	⇒ 1 min
10,000 op/s	⇒ 10 min

(op/s – dotyczy operacji na elementach sortowanego zbioru, które mogą być bardziej złożone niż elementarne operacje procesora)

12

Złożoność obliczeniowa - przykład

Sortowanie zbioru n=1,000,000,000
(symulacje fizyczne, astronomiczne, biologiczne)

Proste metody sortowania $O(n^2)$ (wstawianie, wybieranie, bąbelkowe)

Sprzęt	Czas
1mln op/s	$\Rightarrow$ 31700 lat

Zaawansowane metody sortowania O(n*log(n)) (stogowe, przez podział)

Sprzęt	Czas
1mln op/s	$\Rightarrow$ 2.5h
100,000 op/s	$\Rightarrow$ 1 dzień
10,000 op/s	$\Rightarrow$ 10 dni

(op/s – dotyczy operacji na elementach sortowanego zbioru, które mogą być bardziej złożone niż elementarne operacje procesora)

13

Notacja Ω

Definicja:

Funkcja f(n) jest rzędu $\Omega(g(n))$ jeśli istnieją liczby dodatnie c i **N** takie, że $f(n) \geq c \cdot g(n)$ dla wszystkich $n \geq N$

Funkcja g(n) jest złożonością obliczeniową w sensie $\Omega()$.

Notacja $\Omega()$ jest tzw. optymistyczną oceną złożoności obliczeniowej – opisuje złożoność najlepszego przypadku (tj. jest to ograniczenie dolne funkcji f(n) dla wszystkich możliwych przypadków danych dla algorytmu.)

Np. algorytm przeszukiwania liniowego: $\Theta(n)$, ale $\Omega(1)$,
bo możemy znaleźć element już w pierwszym kroku algorytmu.

14

Uwagi n/t Θ() i Ω()

Oprócz czasowej istnieje tez pamięciowa złożoność obliczeniowa, tj wymaganie na zajętość pamięci komputera. Istotne znaczenie ma zwykle kompromis *czas*pamięć* złożoności obliczeniowej.

W konkretnych zastosowaniach nie wolno lekceważyć czynników stałych wydajności algorytmów.
Np. $10^8*n > 10*n^2$ dla szerokiego zakresu n, mimo iż $\Theta(n) << \Theta(n^2)$

Notacje Θ(n) i Ω() opisują górne i dolne ograniczenie, ale wszystkich możliwych sytuacji. W praktyce ważne jest zwykle prawdopodobieństwo układu danych, co powoduje, że efektywna złożoność jest gdzieś pomiędzy Θ(n) i Ω(). Np. Quicksort jest $\Theta(n^2)$ ale w praktyce Θ(n*log n)

W algorytmach numerycznych względy dokładności i stabilności mogą być dużo ważniejsze o szybkości obliczeń.

15

Ocena złożoności obliczeniowej

Pojedyncza instrukcja: Θ(1)

czas wykonania pojedynczych instrukcji jest zwykle porównywalny, sekwencja i selekcja tez jest Θ(1).

Uwaga na instrukcje, które są istotnie złożone, np. wywołania procedur, funkcji, przeciążone operatory (np. dodawanie tablic).

Cykl: Θ(n)

czas wykonania pojedynczego cyklu jest Θ(n).

```
FOR i:=1 TO n DO
BEGIN
      ...
END
```

16

Ocena złożoności obliczeniowej, c.d.

Cykle zagnieżdżone: $\Theta(n^k)$

czas wykonania jest iloczynem k-czasów pojedynczego cyklu (n*n*..*n) i jest $\Theta(n^k)$,
jeśli zakresy cykli są niezależne.

```
FOR i:=1 TO n DO
      FOR j:=1 TO n DO
            FOR k:=1 TO n DO
            BEGIN
                  ...
            END
```

$\Theta(n^3)$

Oczywiście, w algorytmach nie zawsze jest konieczne wykonanie wszystkich cykli do końca, jednak zwykle zwiększa to tylko szybkość o czynnik stały lub poprawia złożoność średnią.

17

Ocena złożoności obliczeniowej, c.d.

Rekurencja:
należy brać pod uwagę zarówno złożoność czasową jak i pamięciową

Rekurencja prosta: (pojedynczy odwołanie rekurencyjne)
decyduje tempo dochodzenia do końca,
czyli redukcji skali problemu

Silnia:

```
n! = n*(n-1)!
```

Liniowa redukcja skali problemu, złożoność (czasowa i pamięciowa) $\Theta(n)$

Bisekcja:

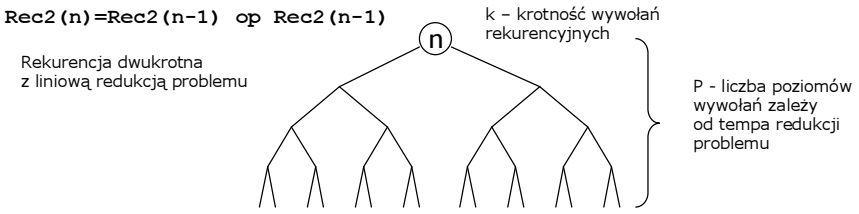
```
Procedure bisekcja(a,b)
...
c:=(a+b)/2
IF x>c THEN bisekcja(c,b)
       ELSE bisekcja(a,c)
```

Logarytmiczna redukcja skali problemu, złożoność (czasowa i pamięciowa) $\Theta(\log n)$

18

Ocena złożoności obliczeniowej, c.d.

Rekurencja wielokrotna: $\Theta(k^P)$ – złożoność wykładnicza
(wielokrotne odwołanie rekurencyjne)
decyduje krotność wywołań i redukcja skali problemu



Ciąg Fibbonacciego (wzór rekurencyjny): `F(n) = F(n-1) + F(n-2)`

Rekurencja dwukrotna +liniowa redukcja skali problemu:
złożoność (czasowa i pamięciowa) $\Theta(2^n)$
F(100) wg tego wzoru jest poza zasięgiem możliwości współczesnych komputerów

19

Klasy P, NP, NPC

P – Polinomial-time Problems (wielomianowe)
klasa problemów, dla których istnieją algorytmy o złożoności czasowej wielomianowej $\Theta(n^k)$, ogólnie przyjmowane jako łatwo rozwiązywalne (np. sortowania, wyszukiwania, operacje macierzowe, etc..)

NP – Nondeterministic Polinomial-time Problems (wielomianowe niedeterministyczne)
klasa problemów, które mają niedeterministyczne algorytmy o czasie wielomianowym. Algorytmy takie byłyby łatwe, gdyby istniał mechanizm „zgadywania" pewnych rozwiązań w kolejnych iteracjach algorytmu.

NPC – Nondeterministic Polinomial-time Complete Problems (wielomianowe niedeterministyczne zupełne)
klasa problemów NP, których rozwiązania można do siebie wzajemnie sprowadzić – rozwiązanie jednego problemu NPC pociąga za sobą możliwość rozwiązania ich wszystkich.

20

P = NP ?

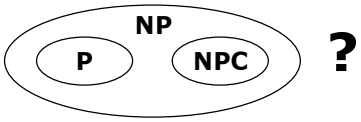
Do klasy NPC należy wiele bardzo istotnych dla nauki problemów:

faktoryzacja liczb (łamanie szyfrów),
poszukiwanie dróg i połączeń (TSP),
szeregowanie i dopasowywanie (plan zajęć),
weryfikacja systemów logicznych (spójność prawa),
pokrycia powierzchni dwuwymiarowych (układanki),
kolorowanie map i grafów (mapa o trzech barwach)...

Jak dotąd nikt nie udowodnił, że problemy NPC nie są łatwo rozwiązywalne.

Problemy NPC są trudno rozwiązywalne, ale ich rozwiązania są bardzo łatwe to weryfikacji (np. weryfikacja podzielności liczby).

Los problemów NPC jest ściśle ze sobą związany:
- znalezienie rozwiązania jednego z nich oznacza rozwiązanie wszystkich,
- wykluczenie łatwej rozwiązywalność jednego z nich, zamyka drogę wszystkim



21

Poza NP ...

Istnieją problemy, co do których można udowodnić „trudną" rozwiązywalność, a zatem nie należą one do klasy NP. „Trudna rozwiązywalność" oznacza zwykle wykładnicze (2^n) dolne graniczenie złożoność obliczeniowej.

(dynamiczna logika zdań)

Istnieją problemy, które mają dolne ograniczenie złożoności w postaci funkcji np. funkcji n-krotnie wykładniczej (2^{2^n}, $2^{2^{2^n}}$)

(rachunek zdań w formalizmach matematycznych)

Istnieją problemy, co do których udowodniono nierozstrzygalność, tj. niemożliwość ukończenia w dowolnie długim czasie i z dowolnie dużą pamięcią.

22

Algorytmy „dziel i zwyciężaj”

Istotą tej techniki jest podział problemu na mniejsze podproblemy, które 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.

Rozwiązanie całego problemu następuje po złożenie rozwiązań cząstkowych w całość.

Metoda zazwyczaj implementowana jest z zastosowaniem technik Rekurencyjnych.

Przykłady:
- sortowanie przez podział, sortowanie przez scalanie
- wyszukiwanie binarne, wyszukiwanie minimum (lub maksimum)
- mnożenie dużych liczb całkowitych
- ...

23

Algorytmy zachłanne

Istotą tej techniki jest podejmowanie kroków dających najlepszy efekt w danym momencie, nie troszcząc się o konsekwencje w przyszłości, czyli w każdym kroku decyzja jest lokalnie optymalna.

Zaletą jest to, że nie traci się czasu na rozważanie co może się stać później.

Okazuje się, że w wielu sytuacjach jest to postępowanie optymalne dla całego problemu.

Przykłady:
- problem kasjera: wydawanie reszty o minimalnej liczbie monet
- znajdowanie drzew rozpinających (optymalizacja sieci)
- problem plecakowy ciągły
- ...

24

Algorytmy z programowaniem dynamicznym

Technika ta wykorzystuje taką własność algorytmu, że aby go rozwiązać, trzeba też rozwiązać wszystkie podproblemy, z których się składa, zapamietać ich wyniki i wykorzystać je do rozwiązania problemu głównego.

Pewne algorytmy często muszą rozwiązywać te same podproblmy wielokrotnie, co znacznie wydłuża ich czas działania. Możliwość korzystania z gotowych wyników podproblemów redukuje tą niedogodność.

Przykłady:
- obliczanie ciągu Fibbonacciego
- problemy triangulacyjne (grafika 3D)
- problem plecakowy dyskretny
- znajdowanie minimalnej ścieżki (znużeni wędrowcy)
- ...

25

Algorytmy z powrotami

Techniki z powrotami stosuje się do problemów, które wiążą się z podejmowaniem decyzji na podstawie pewnej strategii, przypominającej gry, np. kółko-krzyżyk, warcaby, szachy,...

Rozwiązanie problemu jest zdefiniowane jako poszukiwanie jakiegoś rozwiązania wśród wielu możliwych przypadków (stanów) będących rozwiązaniem albo mogących do niego prowadzić. Czasami mogą istnieć stany które nie prowadzą do rozwiązania.

Metoda powrotów wymaga zapamiętania wszystkich wykonanych ruchów czy też wszystkich odwiedzonych stanów aby możliwe było cofanie posunięć.

Liczba stanów może być ogromna, więc odwiedzenia wszystkich stanów, może być zbyt kosztowne. Zwykle wprowadza się metody oceny wartości następnego posunięcia, odrzucając stany nie dające wiele nadziej na pomyslne rozwiązanie.

26

Heurystyki

Istnieje bardzo wiele praktycznych sytuacji, gdy wymagane jest jakiekolwiek pozytywne rozwiązanie problemu, który jest „trudno” rozwiązywalny (NP).

Algorytmy, które nie dają gwarancji znalezienia optymalnego rozwiązania, ale jednak prowadzą do uzyskiwania wartościowych rozwiazań, nazywamy heurystykami.

Heurystyka, to taka strategia postepowania dla danego problemu, która, choć nie gwarantuje uzyskania optimum, pozwala za znalezienie rozwiązania, często gwarantując określone prawdopodobieństwo jego uzyskania lub margines błędu.

27