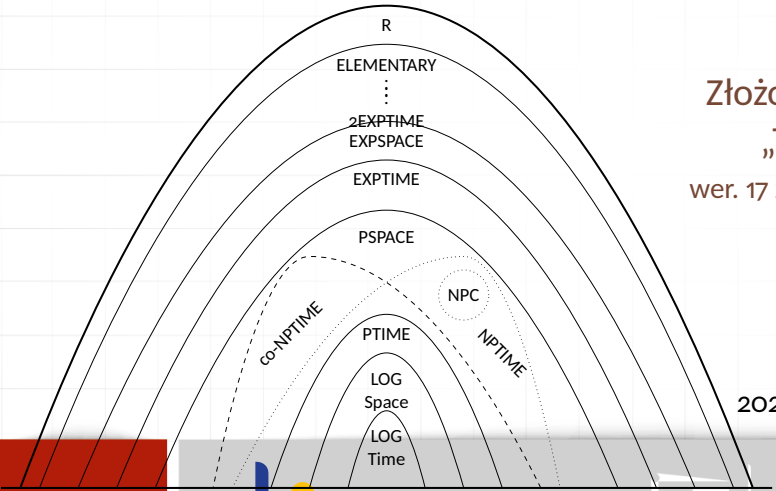


Złożoność obliczeniowa. „Trudne” zadania

wer. 17 z drobnymi modyfikacjami!

Wojciech Myszka

2023-01-12 19:13:38 +0100

Część I

Złożoność obliczeniowa

Pomyślmy o budowie mostu I

Zleciliśmy budowę mostu przez rzekę; Bardzo łatwo zaprojektować konstrukcję nie do przyjęcia:

1. Most może nie mieć wystarczającej szerokości, żeby przenieść oczekiwany ruch,
2. może nie mieć wystarczającej nośności, żeby unieśść wszystkie pojazdy w godzinacz szczytu, lub
3. Może być za krótki, nie sięgając drugiego brzegu!

Nawet jeżeli onnstrukcja będzie poprawna i spełni wszystkie oczekiwania…

- może okazać się, że wymaga zbyt wielu zasbów, żeby ją wybudować:
  - ludzi, lub
  - materiałów lub
  - komponentów;
- Może się okazać, że budow będzie trwałą zbyt długo…

Innymi słowy: **będzie za drogi**.

Tworzenie oprogramowania

1. Dokładnie ten sam problem.
2. Nie chcemy *złych* algorytmów.
3. Nawet poprawny algorytm może być *nieużyteczny* wymagając zbyt wiele zasobów do realizacji

Przykład

Ciąg Fibonacciego

$f(0) = 1; \qquad f(1) = 1; \qquad f(n) = f(n-2) + f(n-1)$

Rekurencyjnie

n	time
10	0,003s
20	0,003s
30	0,016s
40	1,265s
45	14,016s

Nie-rekurencyjnie

n	time
10	0,003s
20	0,003s
30	0,003s
40	0,003s
45	0,003s

Podstawowe kryteria efektywności

1. Potrzebna **pamięć**
   i/lub
2. **czas.**

Pamięć

Czyli zapotrzebowanie na zmienne i struktury danych potrzebne do realizacji algorytmu.

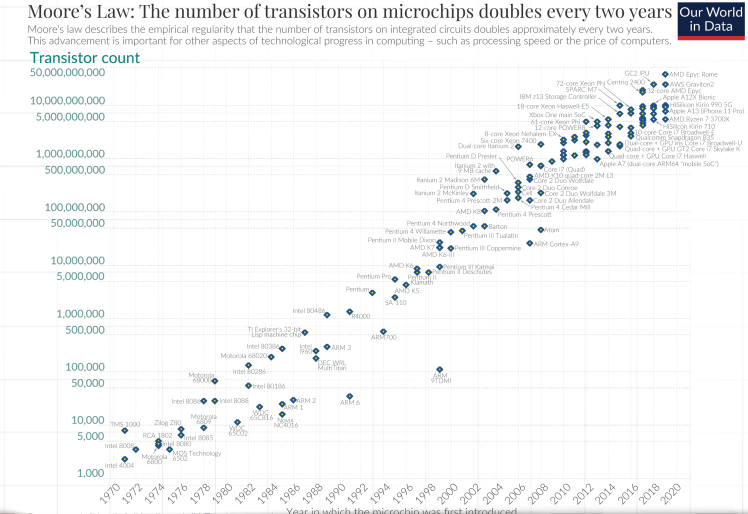
Czas

Czyli liczba elementarnych operacji niezbędnych do realizacji algorytmu.

„Prawo” Moore’a I

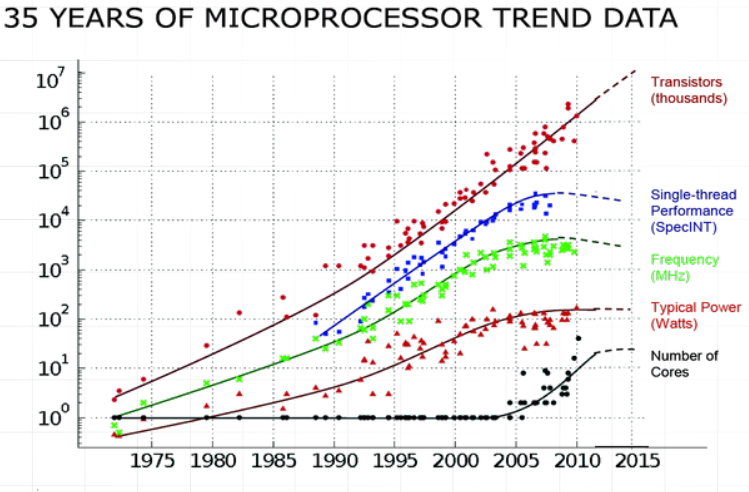
Prwo Moore’a to obserwacja dokonana w 1965 roku przez Gordona Moore’a (założyciela firmy Intel), że liczba tranzystorów w umieszczana układach scalonych podwaja się co dwa lata (w pierwotnym sformułowaniu było to 18 miesięcy).
Nie jest to jakaś zależność fizyczna, ale raczej wnioski z obserwacji na linii produkcyjnej.

„Prawo” Moore’a II



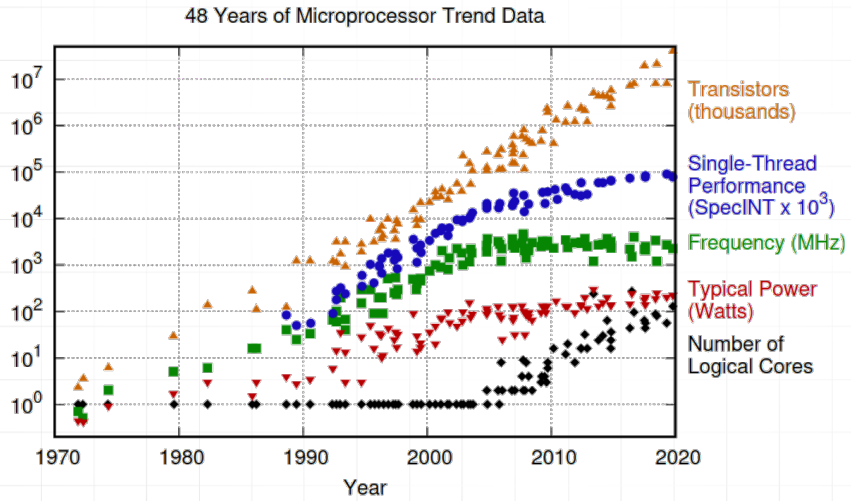
Data source: Wikipedia (wikipedia.org/wiki/Transistor_count)
OurWorldinData.org – Research and data to make progress against the world’s largest problems.

Trend z 35 lat



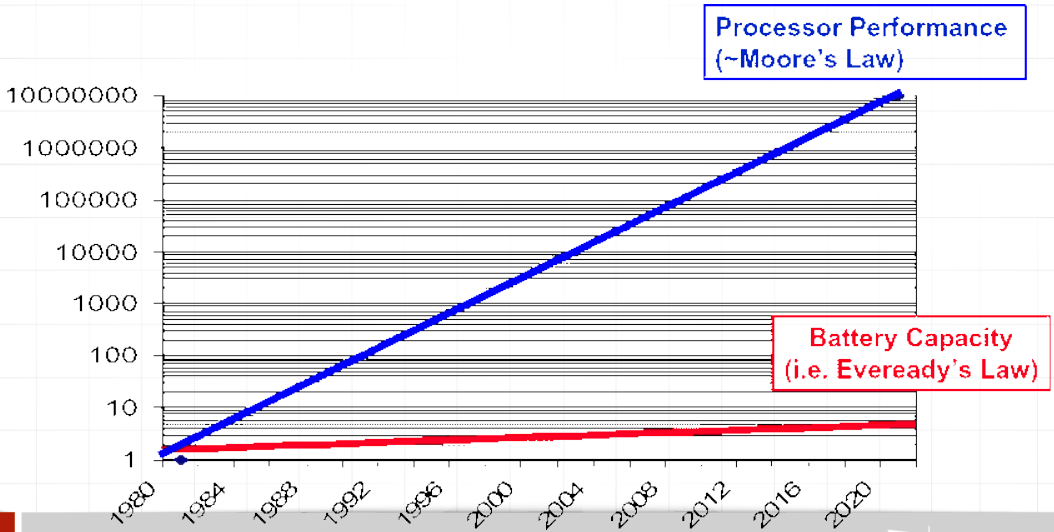
Original data collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond and C. Batten
Dotted line extrapolations by C. Moore

Trend z 48 lat

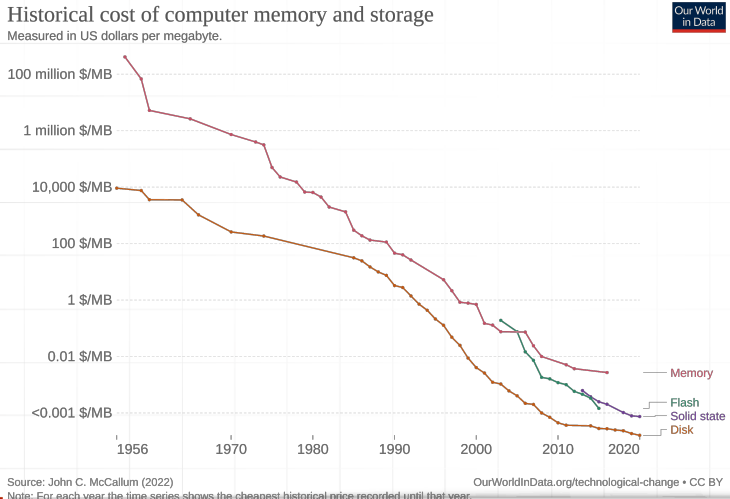


Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Pojemność baterii, a wydajność procesorów



Historyczne koszty pamięci komputerów



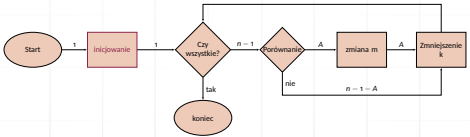
Problemy

1. Wyobraźmy sobie, że chcemy znaleźć **najktrótszą** trasę wędrówki przez 200 miast. Ciągle nie istnieje komputer, który jest w stanie rozwiązać ten problem w czasie krótszym niż miliony lat.
2. Nie istnieje komputer mogący sfaktoryzować (rozłożyć na czynniki pierwsze) liczbę mającą, powiedzmy, 300 cyfr (dziesiętnych) w czasie krótszym niż miliony lat.

Raz jeszcze min/max z liczb…

Danych jest n elementów $X[1], X[2], \ldots, X[n]$; szukamy takich m i j, że $m = X[j] = \max_{1 \le i \le n} X[i]$ i j jest największym indeksem spełniającym ten warunek.
Algorytm wyglądać będzie tak:

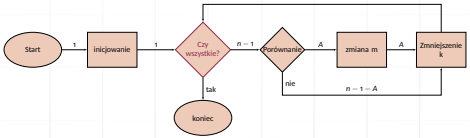
1. **Inicjowanie** Niech $j \leftarrow n$, $k \leftarrow n - 1$, $m \leftarrow X[n]$
2. **Czy wszystkie?** Jeśli $k = 0$ to **koniec**
3. **Porównanie** Jeśli $X[k] \le m$ to przejdź do 5
4. **Zmiana** m $j \leftarrow k$, $m \leftarrow X[k]$
5. **Zmniejsz** k $k \leftarrow k - 1$; przejdź do 2



Raz jeszcze min/max z liczb…

Danych jest n elementów $X[1], X[2], \ldots, X[n]$; szukamy takich m i j, że $m = X[j] = \max_{1 \le i \le n} X[i]$ i j jest największym indeksem spełniającym ten warunek.
Algorytm wyglądać będzie tak:

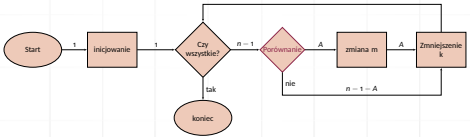
1. **Inicjowanie** Niech $j \leftarrow n$, $k \leftarrow n - 1$, $m \leftarrow X[n]$
2. **Czy wszystkie?** Jeśli $k = 0$ to **koniec**
3. **Porównanie** Jeśli $X[k] \le m$ to przejdź do 5
4. **Zmiana** m $j \leftarrow k$, $m \leftarrow X[k]$
5. **Zmniejsz** k $k \leftarrow k - 1$; przejdź do 2



Raz jeszcze min/max z liczb…

Danych jest n elementów $X[1], X[2], \ldots, X[n]$; szukamy takich m i j, że $m = X[j] = \max_{1 \le i \le n} X[i]$ i j jest największym indeksem spełniającym ten warunek.
Algorytm wyglądać będzie tak:

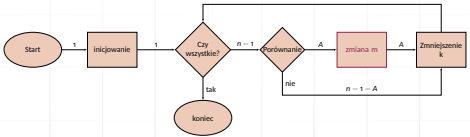
1. **Inicjowanie** Niech $j \leftarrow n$, $k \leftarrow n - 1$, $m \leftarrow X[n]$
2. **Czy wszystkie?** Jeśli $k = 0$ to **koniec**
3. **Porównanie** Jeśli $X[k] \le m$ to przejdź do 5
4. **Zmiana** m $j \leftarrow k$, $m \leftarrow X[k]$
5. **Zmniejsz** k $k \leftarrow k - 1$; przejdź do 2



Raz jeszcze min/max z liczb…

Danych jest n elementów $X[1], X[2], \ldots, X[n]$; szukamy takich m i j, że $m = X[j] = \max_{1 \le i \le n} X[i]$ i j jest największym indeksem spełniającym ten warunek.
Algorytm wyglądać będzie tak:

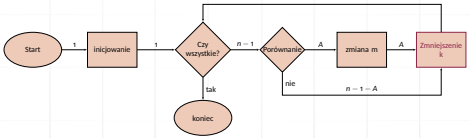
1. **Inicjowanie** Niech $j \leftarrow n$, $k \leftarrow n - 1$, $m \leftarrow X[n]$
2. **Czy wszystkie?** Jeśli $k = 0$ to **koniec**
3. **Porównanie** Jeśli $X[k] \le m$ to przejdź do 5
4. **Zmiana** m $j \leftarrow k$, $m \leftarrow X[k]$
5. **Zmniejsz** k $k \leftarrow k - 1$; przejdź do 2



Raz jeszcze min/max z liczb…

Danych jest n elementów $X[1], X[2], \ldots, X[n]$; szukamy takich m i j, że $m = X[j] = \max_{1 \leq i \leq n} X[i]$ i j jest największym indeksem spełniającym ten warunek.
Algorytm wyglądać będzie tak:

1. **Inicjowanie** Niech $j \leftarrow n$, $k \leftarrow n - 1$, $m \leftarrow X[n]$
2. **Czy wszystkie?** Jeśli $k = 0$ to **koniec**
3. **Porównanie** Jeśli $X[k] \leq m$ to przejdź do 5
4. **Zmiana** m $j \leftarrow k$, $m \leftarrow X[k]$
5. **Zmniejsz** k $k \leftarrow k - 1$; przejdź do 2



Analiza algorytmu

- Zajętość pamięci. W tym przypadku nic ciekawego — zapotrzebowanie na pamięć jest proporcjonalne do liczby danych.

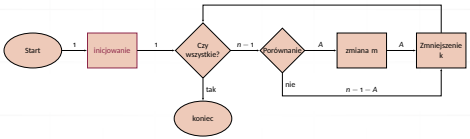
Analiza algorytmu

- Zajętość pamięci. W tym przypadku nic ciekawego — zapotrzebowanie na pamięć jest proporcjonalne do liczby danych.
- Dokładność. Praktycznie nie są wykonywane żadne operacje matematyczne zatem **nie dotyczy**.

Analiza algorytmu

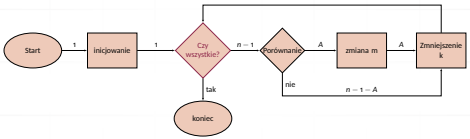
- Zajętość pamięci. W tym przypadku nic ciekawego — zapotrzebowanie na pamięć jest proporcjonalne do liczby danych.
- Dokładność. Praktycznie nie są wykonywane żadne operacje matematyczne zatem **nie dotyczy**.
- Czas obliczeń. To może być ciekawe.

Czas obliczeń



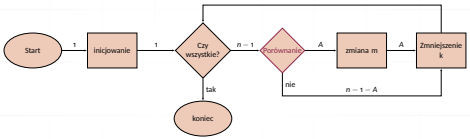
- Krok 1 wykonany będzie 1 raz.

Czas obliczeń



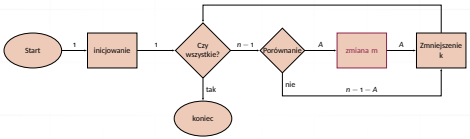
- Krok 1 wykonany będzie 1 raz.
- Krok drugi *n* razy.

Czas obliczeń



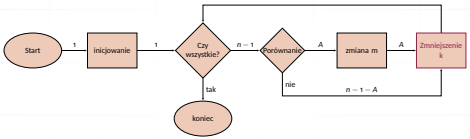
- Krok 1 wykonany będzie 1 raz.
- Krok drugi n razy.
- Krok 3 $n - 1$ razy.

Czas obliczeń



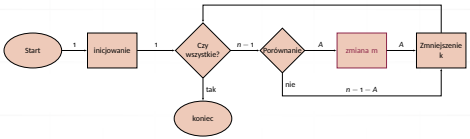
- Krok 1 wykonany będzie 1 raz.
- Krok drugi n razy.
- Krok 3 $n - 1$ razy.
- Krok 4 A razy (jego wykonanie zależy od tego jakie są dane).

Czas obliczeń



- Krok 1 wykonany będzie 1 raz.
- Krok drugi n razy.
- Krok 3 $n - 1$ razy.
- Krok 4 A razy (jego wykonanie zależy od tego jakie są dane).
- Krok 5 $n - 1$ razy.

Czas obliczeń



- Krok 1 wykonany będzie 1 raz.
- Krok drugi n razy.
- Krok 3 $n - 1$ razy.
- Krok 4 A razy (jego wykonanie zależy od tego jakie są dane).
- Krok 5 $n - 1$ razy.

Zatem zajmiemy się liczbą wykonań kroku 4.

Krok 4

Średnia liczba wykonań

- Gdy dane wejściowe uporządkowane są w kolejności malejącej $A = n - 1$.

Krok 4

Średnia liczba wykonań

- Gdy dane wejściowe uporządkowane są w kolejności malejącej $A = n - 1$.
- Gdy dane wejściowe uporządkowane są w kolejności roznącej $A = 0$.

Krok 4

Średnia liczba wykonań

- Gdy dane wejściowe uporządkowane są w kolejności malejącej $A = n - 1$.
- Gdy dane wejściowe uporządkowane są w kolejności roznącej $A = 0$.
- A jaka będzie średnio?

Przykład

Niech $n = 3$; załóżmy również, że wszystkie liczby są różne. Wypiszmy wszystkie możliwe układy liczb $X[1], X[2], X[3]$:

Układ	Wartość A

Przykład

Niech $n = 3$; załóżmy również, że wszystkie liczby są różne. Wypiszmy wszystkie możliwe układy liczb $X[1], X[2], X[3]$:

Układ	Wartość A
$X[1] < X[2] < X[3]$	o

Przykład

Niech $n = 3$; załóżmy również, że wszystkie liczby są różne. Wypiszmy wszystkie możliwe układy liczb $X[1], X[2], X[3]$:

Układ	Wartość A
$X[1] < X[2] < X[3]$	0
$X[1] < X[3] < X[2]$	1

Przykład

Niech $n = 3$; załóżmy również, że wszystkie liczby są różne. Wypiszmy wszystkie możliwe układy liczb $X[1], X[2], X[3]$:

Układ	Wartość A
$X[1] < X[2] < X[3]$	0
$X[1] < X[3] < X[2]$	1
$X[2] < X[1] < X[3]$	0

Przykład

Niech $n = 3$; załóżmy również, że wszystkie liczby są różne. Wypiszmy wszystkie możliwe układy liczb $X[1], X[2], X[3]$:

Układ	Wartość A
$X[1] < X[2] < X[3]$	0
$X[1] < X[3] < X[2]$	1
$X[2] < X[1] < X[3]$	0
$X[2] < X[3] < X[1]$	1

Przykład

Niech $n = 3$; załóżmy również, że wszystkie liczby są różne. Wypiszmy wszystkie możliwe układy liczb $X[1], X[2], X[3]$:

Układ	Wartość A
$X[1] < X[2] < X[3]$	0
$X[1] < X[3] < X[2]$	1
$X[2] < X[1] < X[3]$	0
$X[2] < X[3] < X[1]$	1
$X[3] < X[1] < X[2]$	1

Przykład

Niech $n = 3$; załóżmy również, że wszystkie liczby są różne. Wypiszmy wszystkie możliwe układy liczb $X[1], X[2], X[3]$:

Układ	Wartość A
$X[1] < X[2] < X[3]$	0
$X[1] < X[3] < X[2]$	1
$X[2] < X[1] < X[3]$	0
$X[2] < X[3] < X[1]$	1
$X[3] < X[1] < X[2]$	1
$X[3] < X[2] < X[1]$	2

Przykład

Niech $n = 3$; załóżmy również, że wszystkie liczby są różne. Wypiszmy wszystkie możliwe układy liczb $X[1], X[2], X[3]$:

Układ	Wartość A
$X[1] < X[2] < X[3]$	0
$X[1] < X[3] < X[2]$	1
$X[2] < X[1] < X[3]$	0
$X[2] < X[3] < X[1]$	1
$X[3] < X[1] < X[2]$	1
$X[3] < X[2] < X[1]$	2

Jeżeli teraz założymy, że każdy z układów liczb $X[1], X[2], X[3]$ jest jednakowo prawdopodobne, to średnia wartość A wyniesie (dla $n = 3$) $(0 + 1 + 0 + 1 + 1 + 2)/6 = 5/6$.

Analiza...

Bez starty ogólności, możemy przyjąć, że liczby $X[1], X[2], \ldots, X[n]$ to liczby $1, 2, \ldots, n$ w pewnej kolejności. Dodatkowo założymy, że każda z $n!$ permutacji jest jednakowo prawdopodobna.

Tu wycinamy bardzo wiele rachunków...

Z analizy rachunków wynika, że w przypadku gdy $n = 12$ średnia dla zmiennej losowej A wynosi około 2,1, a wariancja około 1,54.

Analiza algorytmu min/max…

- Algorytm jest bardzo prosty.
- Jedyne „operacje” wykonywane to *porównanie* (krok 2) i *zamiana* (krok 4).
- W zależności od liczby danych (n) liczba porównań jest „stała” — $n - 1$, a liczba zamian jest jakimś ułamkiem n.

W takim przypadku piszemy że liczba zamian jest **proporcjonalna** do n, co zapisujemy czasami jako $\sim n$.
W teorii algorytmów częściej mówi się że „złożoność czasowa” algorytmu jest $O(n)$ (co można czytać „**duże-O od n**”) albo „rzędu n”.
Nie jest ważny współczynnik proporcjonalności, ważne jest jedynie to, że w raz ze wzrostem n rośnie **tak samo szybko** czas obliczeń.

$O(n)$

Mamy dwa (hipotetyczne) problemy:

1. Pierwszy wykonuje się w czasie proporcjonalnym do n
2. Drugi w czasie proporcjonalnym do $\frac{1}{2}n$

Zatem drugie zadanie wykonuje się dwa razy szybciej!
Oba zadania wykonują się w czasie $O(n)$! Jest to zatem dosyć „grube” spojrzenie na czas obliczeń.

Liniowe wyszukiwanie

Mamy „książkę telefoniczną” — czyli listę
<nazwisko, numer_telefonu>.
Zadanie polega na znalezieniu numeru telefonicznego osoby o zadanym nazwisku.
Najprostszy algorytm jest taki

1. $i \leftarrow 1$
2. Czy $poszukiwane\_nazwisko = nazwisko(i)$
3. Jeżeli tak — **Koniec**
4. W przeciwnym razie $i \leftarrow i + 1$
5. Jeżeli $i > n$ **Koniec:** nie znaleziono!
6. W przeciwnym razie przejdź do kroku 2

Im dłuższa lista nazwisk tym dłużej pracuje algorytm.

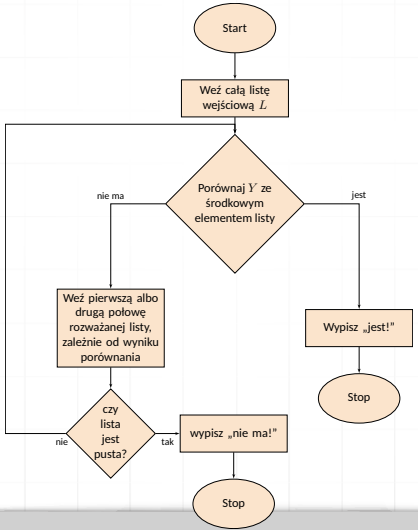
Liniowe wyszukiwanie c. d.

- W najlepszym razie pętla wykonana będzie tylko jeden raz.
- W najgorszym razie — *n* razy.
- Średnio? To zależy jaki jest rozkład prawdopodobieństwa zapytań…

Liniowe wyszukiwanie — lepszy algorytm

- Jeżeli uwzględnić fakt, że dane w książce telefonicznej są posortowane — algorytm można istotnie ulepszyć.
- Pierwsze porównanie należy wykonać nie z pierwszym (czy ostatnim) nazwiskiem, tylko ze środkowym (znajdującym się dokładnie na środku listy). Jeżeli szukane nazwisko jest większe — nie należy zajmować się pierwszą połową listy…
- Takie podejście w każdym kroku redukuje rozmiar zadania o połowę! Każdy kolejny krok jest podobny — zatem wielkość zadania zmniejsza się za każdym razem.
- W przypadku książki telefonicznej o N nazwiskach potrzeba (w najgorszym razie) $\log_2(N)$ kroków.
- Złożoność algorytmu wynosi $O(\log(N))$

Liniowe wyszukiwanie — schemat blokowy



Jak bardzo $O(\log(N))$ jest lepsze od $O(N)$?

N	$\log_2(N)$
10	4
100	7
1000	10
milion	20
miliard	30
miliard miliardów	60

Pętle zagnieżdżone

1. wykonaj co następuje $N-1$ razy

   ...

   1.1 wykonaj co następuje $N-1$ razy

   ...

Pętla wewnętrzna wykonywana jest $N-1$ razy dla każdego z $N-1$ wykonań pętli zewnętrznej.
Koszt czasowy dwu pętli jest zatem $(N-1)(N-1)$ czyli N^2-2N+1. Dla dużych N składnik $1-2N$ jest nieistotny. Zatem koszt czasowy takiej pętli jest $O(N^2)$.

Minimum i maksimum wersja rekurencyjna

Rekurencyjny algorytm wyszukiwania minimum i maksimum w L (**znajdź-min-i-max-w** *L*):

1. jeśli L składa się z jednego elementu to nadaj *MIN* i *MAX* właśnie jego wartość; jeśli *L* składa się z dwu elementów, to nadaj *MIN* wartość mniejszego z nich, a *MAX* — większego;
2. w przeciwnym razie wykonaj co następuje:
   - 2.1 podziel *L* na połowy L_1 i L_2;
   - 2.2 wywołaj **znajdź-min-i-max-w** L_1, umieszczając otrzymane wartości w MIN_1 i MAX_1;
   - 2.3 wywołaj **znajdź-min-i-max-w** L_2, umieszczając otrzymane wartości w MIN_2 i MAX_2;
   - 2.4 nadaj *MIN* mniejszą wartość z MIN_1 i MIN_2;
   - 2.5 nadaj *MAX* większą wartość z MAX_1 i MAX_2;
3. wróć z wartościami *MIN* i *MAX*

Analiza algorytmu rekurencyjnego I

1. jeśli $N = 1$ lub $N = 2$, wykonuje się dokładnie jedno porównanie — punkt 1 procedury.
2. jeśli $N > 2$, to porównania, które się wykonuje, stanowią dokładnie dwa zbiory porównań dla list o długości $N/2$ i dwa dodatkowe porównania (wiersze 2.4 i 2.5)

Zatem ($C(N)$ oznacza liczbę porównań dla listy N elementowej)

$$C(1) = C(2) = 1 \tag{1}$$

$$C(N) = 2C(N/2) + 2 \tag{2}$$

(czemu jest to $+2$?)
Można rozwiązać to równanie.

$$C(N) = 3N/2 - 2 = \frac{3}{2}N - 2 < 2N \tag{3}$$

Minimum i maksimum — wersja klasyczna (naiwna)

1. $MIN \leftarrow L(1);\ MAX \leftarrow L(1)$
2. $i \leftarrow 1$
3. jeżeli $L(i) > MAX$ to $MAX \leftarrow L(i)$
4. jeżeli $L(i) < MIN$ to $MIN \leftarrow L(i)$
5. jeżeli $i < N$ to $i \leftarrow i + 1$, przejdź do 3
6. KONIEC

W tym algorytmie główna operacja to porównania; algorytm potrzebuje $2N$ porównań.

Wieże Hanoi

Złożoność czasowa podawanego wcześniej algorytmu wynosi $2^N - 1$ (gdzie N to liczba krążków).
Zatem algorytm jest $O(2^N)$

Wieże Hanoi

Alternatywny algorytm (nierekurencyjny)

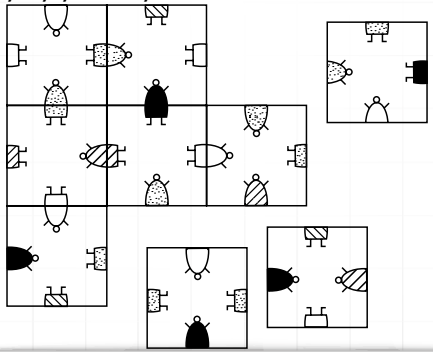
(Zakładamy, że nasze trzy kołki są ustawione w kółko i że — na początku — wszystkie N krążków spoczywa na jednym z nich.)

1. powtarzaj wykonywanie tego, co następuje, dopóty, dopóki, w chwili poprzedzającej podjęcie kroku 1.1 wszystkie krążki nie będą poprawnie ułożone na jakimś innym kołku:
   - 1.1 przenieś najmniejszy krążek z kołka, na którym właśnie spoczywa, na kołek następny w kierunku zgodnym z ruchem wskazówek zegara;
   - 1.2 wykonaj jedyne możliwe przeniesienie nie dotyczące najmniejszego krążka.

A jaka jest złożoność tego algorytmu?

„Małpia układanka”

Prosta gra, w której do dyspozycji gracza jest kilka (9?) kwadratowych kart, na bokach których wydrukowane są górna i dolne połowy kolorowych małp. Zadanie polega na ułożeniu kart w postaci kwadratu (3×3), aby na zetknięciach się krawędzi połowy małp pasowały do siebie, a kolory były identyczne…



Małpia układanka cd

- Najbardziej elementarny komputerowy algorytm „rozwiązywania” tej układanki sprowadzi się najprawdopodobniej, do przeglądu wszystkich możliwych układów kart.
- Wybieramy jedną z kart, układamy ją na pierwszym miejscu (odpowiednio wszystkie miejsca ponumerowaliśmy); na drugim miejscu mamy $N-1$ kart do wyboru, na kolejnym $N-2$, itd...
- Następnie sprawdzamy czy spełnione są warunki.
- Zatem liczba możliwych do rozpatrzenia kombinacji wynosi:

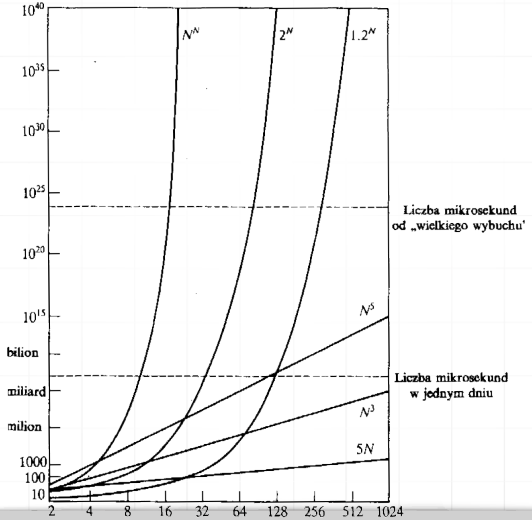
$$N \times (N-1) \times (N-2) \times \cdots \times 2 \times 1 = N! \tag{4}$$

Zatem złożoność tego algorytmu jest rzędu $O(N!)$

Rozsądny i nierozsądny czas działania

Funkcja / N	10	50	100	300	1000
$5N$	50	250	500	1500	5000
$N \log_2(N)$	33	282	665	2469	9966
N^2	100	2500	10000	90000	1 milion (7 cyfr)
N^3	1000	125000	1 milion (7 cyfr)	27 milionów (8 cyfr)	1 miliard (10 cyfr)
2^N	1024	liczba 16-cyfrowa	liczba 31-cyfrowa	liczba 91-cyfrowa	liczba 302-cyfrowa
$N!$	3,6 miliona (7 cyfr)	liczba 65-cyfrowa	liczba 161-cyfrowa	liczba 623-cyfrowa	niewyobrażalnie duża
N^N	10 miliardów (11 cyfr)	liczba 85-cyfrowa	liczba 201-cyfrowa	liczba 744-cyfrowa	niewyobrażalnie duża

Wykresy różnych funkcji



Tabelka…

N	10	20	50	100	300
N^2	1/10 000 sekundy	1/2500 sekundy	1/400 ssekundy	1/100 sekundy	9/100 sekundy
N^5	1/10 sekundy	3,2 ssekundy	5,2 minuty	2,8 godziny	28,1 dnia
2^N	1/1000 sekundy	1 sekunda	35,7 lat	400 bilionów stuleci	75-cyfrowa liczba stuleci
N^N	2,8 godziny	3,3 biliony lat	70-cyfrowa liczba stuleci	185-cyfrowa liczba stuleci	728 cyfrowa liczba stuleci

Dla porównania „wielki wybuch” był w przybliżeniu 15 miliardów lat temu
Zapotrzebowanie na czas dla hipotetycznych rozwiązań małpiej układanki
(przy założeniu, że jedna instrukcja trwa jedną milisekundę)

Zadania „łatwe” i „trudne” I

- Jak widać z wykresów i tabelek — zawsze można znaleźć zadanie którego nie da się rozwiązać w „rozsądnym czasie”.
- Przyjęto dzielić zadania na „łatwe” jeżeli funkcja opisująca czas ich wykonania (złożoność obliczeniową) jest ograniczona jakąś funkcją **wielomianową**.
- Zadania „trudne” to zadania o czasie wykonania ponadwielomianowym lub wykładniczym.

Podsumowanie

Problemy trudno rozwiązywalne wymagają niepraktycznie dużo czasu nawet dla względnie małych danych wejściowych.
Problemy łatwo rozwiązywalne mają algorytmy, które są praktyczne dla zadań o rozsądnych wymiarach.

Wątpliwości

1. Komputery stają się coraz szybsze (ostatnio, w ciągu roku szybkość komputerów wzrastało dziesięciokrotnie). Być może za czas jakiś pojawi się komputer wystarczająco szybki by rozwiązywać „trudne zadania”?

Wątpliwości

1. Komputery stają się coraz szybsze (ostatnio, w ciągu roku szybkość komputerów wzrastało dziesięciokrotnie). Być może za czas jakiś pojawi się komputer wystarczająco szybki by rozwiązywać „trudne zadania”?
2. Czy fakt, że nie znaleźliśmy lepszego algorytmu nie wskazuje na naszą niekompetencję? Może warto zwiększyć wysiłek aby polepszyć sytuację?

Wątpliwości

1. Komputery stają się coraz szybsze (ostatnio, w ciągu roku szybkość komputerów wzrastało dziesięciokrotnie). Być może za czas jakiś pojawi się komputer wystarczająco szybki by rozwiązywać „trudne zadania”?
2. Czy fakt, że nie znaleźliśmy lepszego algorytmu nie wskazuje na naszą niekompetencję? Może warto zwiększyć wysiłek aby polepszyć sytuację?
3. Być może uda się znaleźć jakiś dowód, że problem nie ma rozsądnego algorytmu?

Wątpliwości

1. Komputery stają się coraz szybsze (ostatnio, w ciągu roku szybkość komputerów wzrastało dziesięciokrotnie). Być może za czas jakiś pojawi się komputer wystarczająco szybki by rozwiązywać „trudne zadania”?
2. Czy fakt, że nie znaleźliśmy lepszego algorytmu nie wskazuje na naszą niekompetencję? Może warto zwiększyć wysiłek aby polepszyć sytuację?
3. Być może uda się znaleźć jakiś dowód, że problem nie ma rozsądnego algorytmu?
4. Czy „trudny problem” (małpiej układanki) nie jest przypadkiem jedynym takim zadaniem? Jest ono ciekawe, ale jakie ma praktyczne zastosowanie?

Szybszy komputer

$O(?)$	Maksymalna liczba kart rozwiązywanych w ciągu godziny		
	na dzisiejszym komputerze	na komputerze 100 razy szybszym	na komputerze 1000 razy szybszym
N	A	$100 \times A$	$1000 \times A$
N^2	B	$10 \times B$	$31{,}6 \times B$
2^N	C	$C + 6{,}64$	$C + 9{,}97$

Wymyślony algorytm

(o małpiej układance)

- Niestety nie. Znaleziono bardzo wiele (około 1000) problemów, które sprowadzają się do „przeglądu zupełnego” wszystkich możliwych wariantów (ustawień danych).
- Można zaproponować algorytmy „nierozsądne” nie udało się pokazać że może istnieć dokładny algorytm lepszy…
- Źródłem „trudnych” zadań są takie dziedziny jak kombinatoryka, badania operacyjne, ekonomia, teoria grafów i logika.

Jak sobie radzimy z trudnymi zadaniami?

- **Zrównoleglenie** (zadanie dzielone jest na mniejsze kawałki i wysyłane na wiele komputerów).

Jak sobie radzimy z trudnymi zadaniami?

- **Zrównoleglenie** (zadanie dzielone jest na mniejsze kawałki i wysyłane na wiele komputerów).
- **Randomizacja.** Zamiast przeglądać wszystkie możliwe rozwiązania w sposób losowy wybiera się niektóre z nich. Pozostają te, które w najlepszy sposób spełniają nasze oczekiwania.

Jak sobie radzimy z trudnymi zadaniami?

- **Zrównoleglenie** (zadanie dzielone jest na mniejsze kawałki i wysyłane na wiele komputerów).
- **Randomizacja.** Zamiast przeglądać wszystkie możliwe rozwiązania w sposób losowy wybiera się niektóre z nich. Pozostają te, które w najlepszy sposób spełniają nasze oczekiwania.
- **Komputery kwantowe.** ???

Jak sobie radzimy z trudnymi zadaniami?

- **Zrównoleglenie** (zadanie dzielone jest na mniejsze kawałki i wysyłane na wiele komputerów).
- **Randomizacja.** Zamiast przeglądać wszystkie możliwe rozwiązania w sposób losowy wybiera się niektóre z nich. Pozostają te, które w najlepszy sposób spełniają nasze oczekiwania.
- **Komputery kwantowe.** ???
- **Obliczenia molekularne.** ???

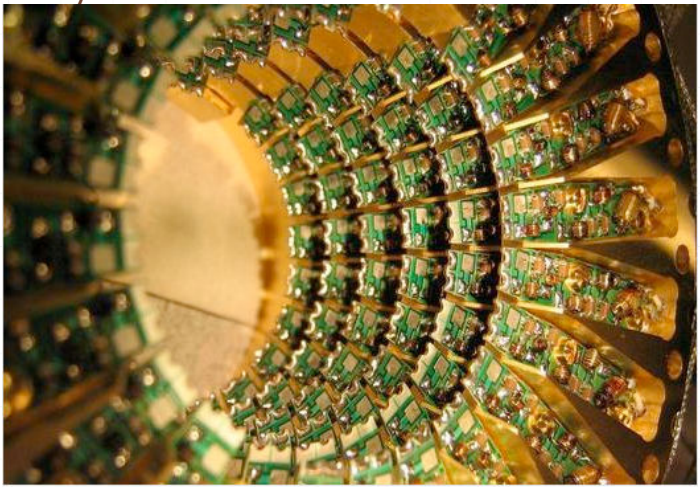
Do czego może nadać się komputer kwantowy?

Jeżeli mamy zadanie spełniające następujące warunki:

1. Jedyna znana metoda rozwiązania go to zgadnięcie i sprawdzenie.
2. Do sprawdzenia mamy *n* możliwych odpowiedzi.
3. Każde sprawdzenie zajmuje tyle samo czasu.
4. Nie ma żadnych wskazówek na temat kolejności sprawdzania (każda, nawet losowa, kolejność jest dobra).

Czas rozwiązania takich zadań przez komputer kwantowy będzie proporcjonalny do pierwiastka kwadratowego z *n*.

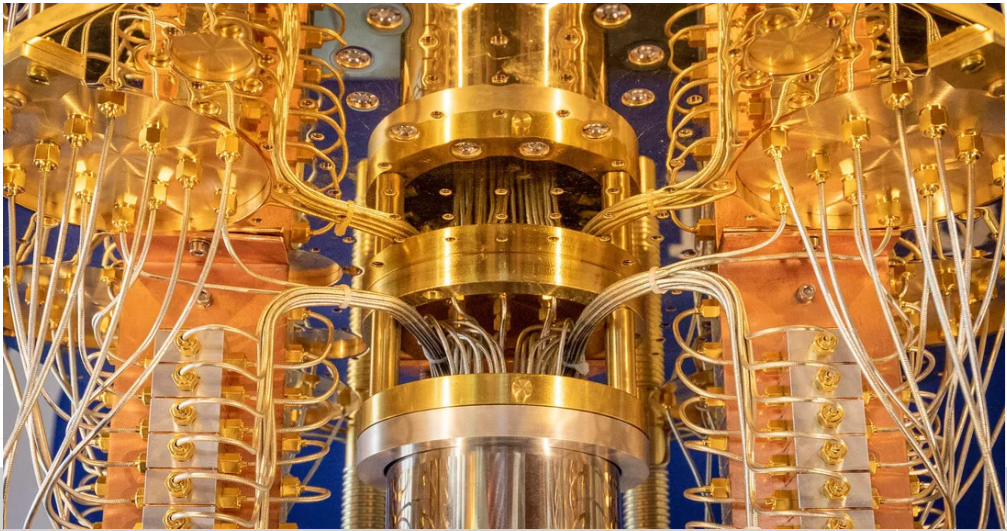
Komputer kwantowy?



Google's Foray into Faster Computing This D-Wave processor may or may not be a quantum computing device *D-Wave/M. Thom*

Komputer kwantowy?

Komputer kwantowy?



Bibliografia