INFORMATYKA

Superkomputery i złożoność obliczeniowa algorytmów

dr Anna Beata Kwiatkowska

Współczesne zagadnienia informatyczne

cyberbezpieczeństwo

transformacja technologii

sztuczna inteligencja

internet rzeczy

szyfrowanie informacji

projektowanie 3D

robotyka

inteligentny dom

big data

technologia 5G

podpis elektroniczny

prawo autorskie

Informatyka – podejście ogólne

Bardzo rozległa dziedzina nauki i techniki:

- administracja sieciowa, administracja systemem, algorytmika, architektura procesów, bezpieczeństwo komputerowe, bezpieczeństwo danych, grafika komputerowa, inżynieria oprogramowania, języki programowania, programowanie komputerów, sprzęt komputerowy, symulacje komputerowe, systemy informatyczne, sztuczna inteligencja, teoria informacji, strony www,…

Podejście naukowe - dyscypliny informatyczne

Rozporządzenie Ministra Nauki i Szkolnictwa Wyższego z dnia 20 września 2018 r. w sprawie dziedzin nauki i dyscyplin naukowych oraz dyscyplin artystycznych:

- dziedzina nauk ścisłych i przyrodniczych – dyscyplina naukowa informatyka,
- dziedzina nauk inżynieryjno-technicznych – dyscyplina naukowa informatyka techniczna i telekomunikacja.

Obszary informatyki rozważane w edukacji

Rozdział 10

Dlaczego algorytmika i programowanie są takie ważne w informatyce?

Jednostki mocy obliczeniowej (szybkości) komputerów

FLOPS (ang. ***FL****oating point* ***O****perations* ***P****er* ***S****econd*) – liczba operacji zmiennoprzecinkowych na sekundę.

nazwa	oznaczenie	Liczba operacji
megaflops	MFLOPS	10^6
gigaflops	GFLOPS	10^9
teraflops	TFLOPS	10^{12}
petaflops	PFLOPS	10^{15}
eksaflops	EFLOPS	10^{18}
zettaflops	ZFLOPS	10^{21}
jottaflops	YFLOPS	10^{24}

Superkomputery

- Superkomputery uzyskują swoją wydajność dzięki połączeniu wielu tysięcy procesorów i zrównoleglaniu wykonywanych obliczeń.

2009	Jaguar	1,759 PFLOPS	Cray, Oak Ridge National Laboratory, Tennessee, USA
2010	Tianhe-1A	2,507 PFLOPS	National Supercomputing Center, Tiencin, Chińska Republika Ludowa
2011	K computer	8,162 PFLOPS	Fujitsu, Riken, Kobe, Japonia
		10,51 PFLOPS	
2012	Sequoia	16,32 PFLOPS	IBM, Lawrence Livermore National Laboratory, Kalifornia, USA
	Titan	17,59 PFLOPS	Cray, Oak Ridge National Laboratory, Tennessee, USA
2013	Tianhe-2	33,86 PFLOPS	NUDT, Chińska Republika Ludowa
2016	Sunway TaihuLight	93,01 PFLOPS	Wuxi, Chińska Republika Ludowa
2018	Summit	122,30 PFLOPS	IBM, Oak Ridge National Laboratory, Tennessee, USA

Źródło: Wikipedia

Zastosowania superkomputera Summit - przykłady

- astrofizyka – modelowanie procesów zachodzących podczas wybuchu gwiazd supernowych (w tym powstawanie nowych pierwiastków),
- badania nad nowotworami – prace nad narzędziami zdolnymi do automatycznej klasyfikacji i analizy czynników chorobowych, takimi jak geny, markery biologiczne czy środowisko zewnętrzne,
- biologia systemowa – zastosowanie technik uczenia maszynowego i sztucznej inteligencji do analizy zbiorów danych genetycznych i biomedycznych w celu lepszego zrozumienia zjawisk,
- inżynieria materiałowa – prace nad materiałami nowej generacji, w tym związków do magazynowania, przekształcania i produkcji energii.

Do czego służą superkomutery?

- Skrócenie czasu oczekiwania na wyniki
  Dużo zadań można podzielić na mniejsze części i liczyć jednocześnie na wielu procesorach. Współczesne maszyny budowane są z kilkunastu tysięcy procesorów.
- Możliwość analizy wielu wariantów jednocześnie
  Każdy z procesorów superkomputera może liczyć jednocześnie różne warianty tego samego zadania.
- Możliwość analizowania problemów o dużo większej złożoności
  Jeśli chcemy zwiększyć dokładność modelu - poprzez zastosowanie drobniejszego podziału analizowanego obszaru lub poprzez lepsze modele zjawisk.
- Superkomputery są używane w armii, badaniach naukowych i biznesie, itp.

Duże moce obliczeniowe - Kryptografia

Kryptografia - dziedzina wiedzy o utajnianiu (szyfrowaniu) informacji, uznawana za gałąź zarówno matematyki i informatyki;

RSA (1977, Ron Rivest, Adi Shamir, Leonard Adleman) jeden z pierwszych i obecnie najpopularniejszych asymetrycznych algorytmów kryptograficznych z kluczem publicznym, stosowany zarówno do szyfrowania jak i do podpisów cyfrowych.

Szyfr RSA bazuje na podnoszeniu do dużej potęgi dużych liczb, np.

$$1234567890987654321234567890987654321234567890987654321$$
$$11234567890^{1234567890987654321234567890987654321234567890987654321}$$

Jak można szybko obliczać takie potęgi?

Algorytm: $x^m = x*x*x* \ldots * x$ $m - 1$ mnożeń

Obliczenie nawet małej potęgi (dla mocy 1 PFLOPS):

$$x^{12345678912345678912345678912345}$$

trwałoby: $4*10^8$ lat

Szybkie komputery i jeszcze szybszy algorytm

- Ile mnożeń jest potrzebnych, by obliczyć x^{100}?
  - $x^{100} = x*x*x*x* \ldots *x$

    Czy rzeczywiście potrzeba aż 99 mnożeń?
- Szybki algorytm:
  - $x^{100} = (x^{50})^2 =$
  - $= ((x^{25})^2)^2 =$
  - $= (((x^{12})^2 *x)^2)^2 =$
  - $= ((((x^6)^2)^2 *x)^2)^2 =$
  - $= (((((x^3)^2)^2)^2 *x)^2)^2 =$
  - $= (((((x^2 *x)^2)^2)^2 *x)^2)^2$

    Wystarczy 8 mnożeń!

  - Jaki związek ma liczba mnożeń z zapisem binarnym wykładnika?

    $(100)_{10}=(1100100)_2$

Szybkie potęgowanie – szybki algorytm!

- Jak obliczyć liczbę mnożeń dla x^m ?
  - Przedstawiamy wykładnik w postaci binarnej
  - Liczba podnoszeń do kwadratu to liczba cyfr binarnych -1, ok. $\log_2 m$
  - Liczba mnożeń dodatkowo przez x, to liczba jedynek -1, nie więcej niż $\log_2 m$
  - cały algorytm wykonuje około $2*\log_2 m$ mnożeń

- Obliczenie algorytmem szybkiego potęgowania wartości

$x^{12345678901234567890123456789012345}$

to nie więcej niż....

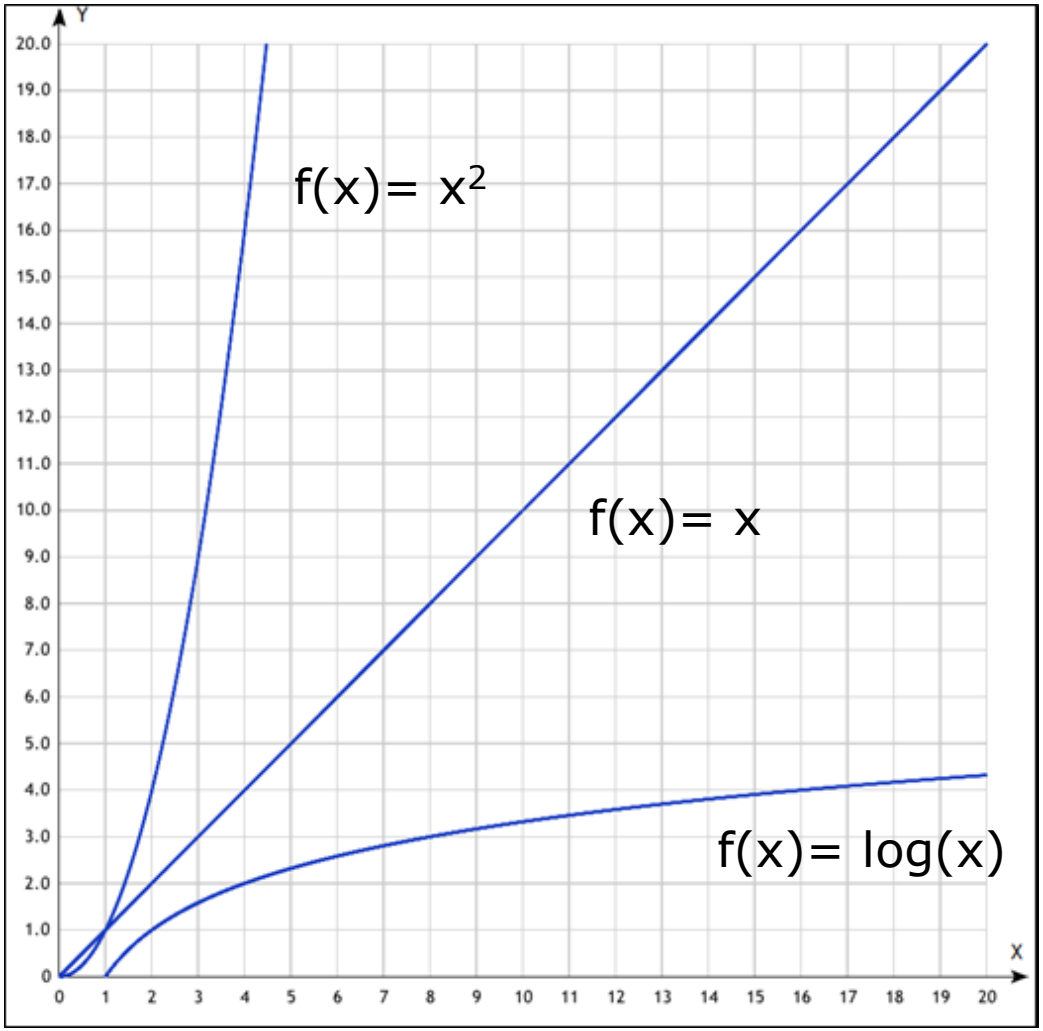
207 mnożeń

Superkomputer obliczy tę wartość
w niezauważalnym dla nas czasie!

m	$\log_2 m$
10^4	10
10^6	20
10^9	30
10^{12}	40
10^{20}	66,5
10^{50}	166
10^{100}	332,2

Liczba operacji a wielkość danych

liczba operacji



wielkość danych

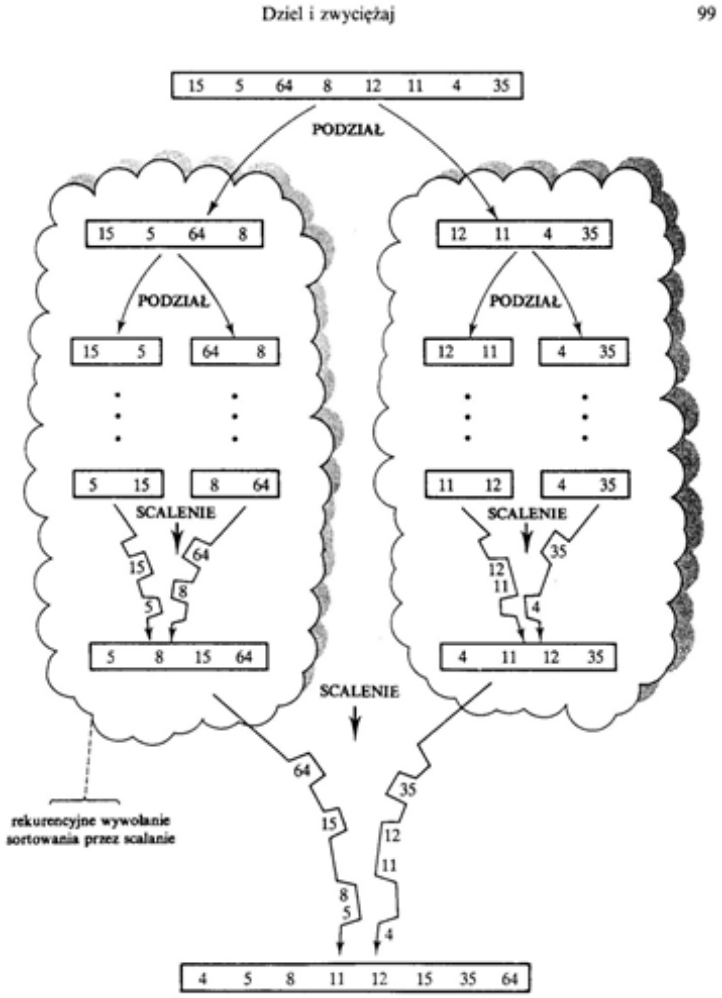
Algorytmika – ważna dziedzina informatyki

David Harel (ur. 1950 w Londynie) wykładowca informatyki w Instytucie Weizmanna w Izraelu.

Obszary badań: logika modalna, teoria obliczalności i inżynierii oprogramowania, języki wizualne, sposoby reprezentacji grafów, biologia systemów i komunikacja zapachowa, komputerowy model nicieni.

Rzecz o istocie informatyki Algorytmika (1987 rok) – nauka o algorytmach

Metody algorytmiczne



Motto

Najlepszym sposobem przyspieszania komputerów

jest obarczanie ich mniejszą liczbą działań (szybszymi algorytmami).

[Ralf Gomory, IBM]

- Przede wszystkim trzeba szukać algorytmów wykonujących mniej działań.
- Przetwarzane informacje powinny zajmować jak najmniej pamięci.

Rozdział 11

Jak przechowywać cyfrowo duże zbiory informacji?

Kilka słów o kompresji danych

Kody ASCII

- ASCII (ang. *American Standard Code for Information Interchange)*
- *k*ody 7-bitowe (zakres 0-127) przyporządkowane literom alfabetu angielskiego, cyfrom, znakom przestankowym, innym symbolom i znakom sterującym.
- Znaki ASCII dzielą się na:
  - drukowane: 95 znaków o kodach 32–126
  - sterujące: 33 znaki o kodach 0–31 i 127
- Rozszerzenie kodowania o dodatkowy ósmy bit (zakres 128-256) wykorzystuje się do kodowania m. in. charakterystycznego dla danego języka.
- Fragment tabeli kodów ASCII:

Bin	Dec	Hex	Znak	Skrót
0000 0000	0	00	Null	NUL
0000 0001	1	01	Start of Heading	SOH
0000 0010	2	02	Start of Text	STX
0000 0011	3	03	End of Text	ETX
0000 0100	4	04	End of Transmission	EOT
0000 0101	5	05	Enquiry	ENQ
0000 0110	6	06	Acknowledge	ACK
0000 0111	7	07	Bell	BEL
0000 1000	8	08	Backspace	BS
0000 1001	9	09	Horizontal Tab	HT
0000 1010	10	0A	Line Feed	LF
0000 1011	11	0B	Vertical Tab	VT
0000 1100	12	0C	Form Feed	FF
0000 1101	13	0D	Carriage Return	CR
0000 1110	14	0E	Shift Out	SO
0000 1111	15	0F	Shift In	SI
0001 0000	16	10	Data Link Escape	DLE
0001 0001	17	11	Device Control 1 (XON)	DC1
0001 0010	18	12	Device Control 2	DC2
0001 0011	19	13	Device Control 3 (XOFF)	DC3
0001 0100	20	14	Device Control 4	DC4
0001 0101	21	15	Negative Acknowledge	NAK
0001 0110	22	16	Synchronous Idle	SYN
0001 0111	23	17	End of Transmission Block	ETB
0001 1000	24	18	Cancel	CAN
0001 1001	25	19	End of Medium	EM
0001 1010	26	1A	Substitute	SUB
0001 1011	27	1B	Escape	ESC
0001 1100	28	1C	File Separator	FS
0001 1101	29	1D	Group Separator	GS
0001 1110	30	1E	Record Separator	RS
0001 1111	31	1F	Unit Separator	US

Bin	Dec	Hex	Znak
0010 0000	32	20	Spacja
0010 0001	33	21	!
0010 0010	34	22	"
0010 0011	35	23	#
0010 0100	36	24	$
0010 0101	37	25	%
0010 0110	38	26	&
0010 0111	39	27	'
0010 1000	40	28	(
0010 1001	41	29	)
0010 1010	42	2A	*
0010 1011	43	2B	+
0010 1100	44	2C	,
0010 1101	45	2D	-
0010 1110	46	2E	.
0010 1111	47	2F	/
0011 0000	48	30	0
0011 0001	49	31	1
0011 0010	50	32	2
0011 0011	51	33	3
0011 0100	52	34	4
0011 0101	53	35	5
0011 0110	54	36	6
0011 0111	55	37	7
0011 1000	56	38	8
0011 1001	57	39	9
0011 1010	58	3A	:
0011 1011	59	3B	;
0011 1100	60	3C	<
0011 1101	61	3D	=
0011 1110	62	3E	>
0011 1111	63	3F	?

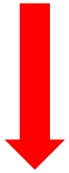
Bin	Dec	Hex	Znak
0100 0000	64	40	@
0100 0001	65	41	A
0100 0010	66	42	B
0100 0011	67	43	C
0100 0100	68	44	D
0100 0101	69	45	E
0100 0110	70	46	F
0100 0111	71	47	G
0100 1000	72	48	H
0100 1001	73	49	I
0100 1010	74	4A	J
0100 1011	75	4B	K
0100 1100	76	4C	L
0100 1101	77	4D	M
0100 1110	78	4E	N
0100 1111	79	4F	O
0101 0000	80	50	P
0101 0001	81	51	Q
0101 0010	82	52	R
0101 0011	83	53	S
0101 0100	84	54	T
0101 0101	85	55	U
0101 0110	86	56	V
0101 0111	87	57	W
0101 1000	88	58	X
0101 1001	89	59	Y
0101 1010	90	5A	Z
0101 1011	91	5B	[
0101 1100	92	5C	\
0101 1101	93	5D	]
0101 1110	94	5E	^
0101 1111	95	5F	_

Bin	Dec	Hex	Znak	Skrót
0110 0000	96	60	`	
0110 0001	97	61	a	
0110 0010	98	62	b	
0110 0011	99	63	c	
0110 0100	100	64	d	
0110 0101	101	65	e	
0110 0110	102	66	f	
0110 0111	103	67	g	
0110 1000	104	68	h	
0110 1001	105	69	i	
0110 1010	106	6A	j	
0110 1011	107	6B	k	
0110 1100	108	6C	l	
0110 1101	109	6D	m	
0110 1110	110	6E	n	
0110 1111	111	6F	o	
0111 0000	112	70	p	
0111 0001	113	71	q	
0111 0010	114	72	r	
0111 0011	115	73	s	
0111 0100	116	74	t	
0111 0101	117	75	u	
0111 0110	118	76	v	
0111 0111	119	77	w	
0111 1000	120	78	x	
0111 1001	121	79	y	
0111 1010	122	7A	z	
0111 1011	123	7B	{	
0111 1100	124	7C	|	
0111 1101	125	7D	}	
0111 1110	126	7E	~	
0111 1111	127	7F	Delete	DEL

Jak informacje są pamiętane w komputerze?

Alfabet ={a, b, d, k, r}

abrakadabra

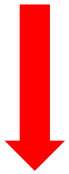


kody ASCII

o stałej długości 8 bitów

01100001011000100111001001100001011010110110000101100100011000010110001001110010011000011

potrzebna pamięć



11 znaków * 8 bitów = 88 bitów pamięci (11 Bajtów)

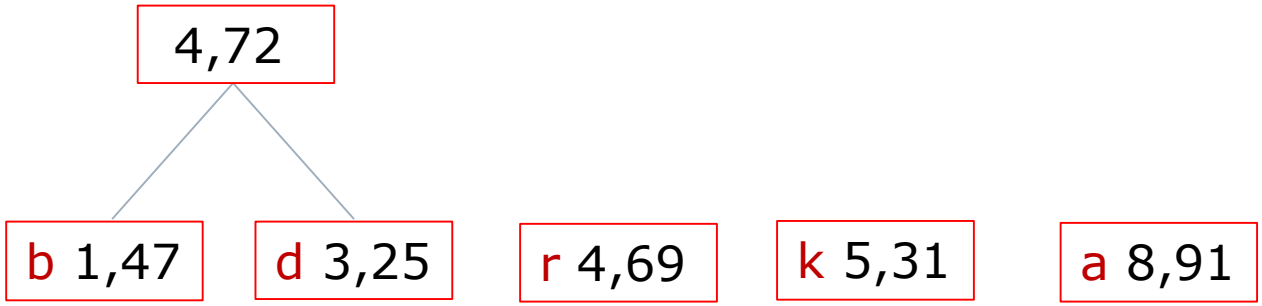
Jak zmniejszyć objętość informacji?

- Wykorzystujemy informację o częstości występowania poszczególnych znaków w tekście (%).
- Ustawiamy znaki w kolejności niemalejących częstości.

b 1,47	d 3,25	r 4,69	k 5,31	a 8,91

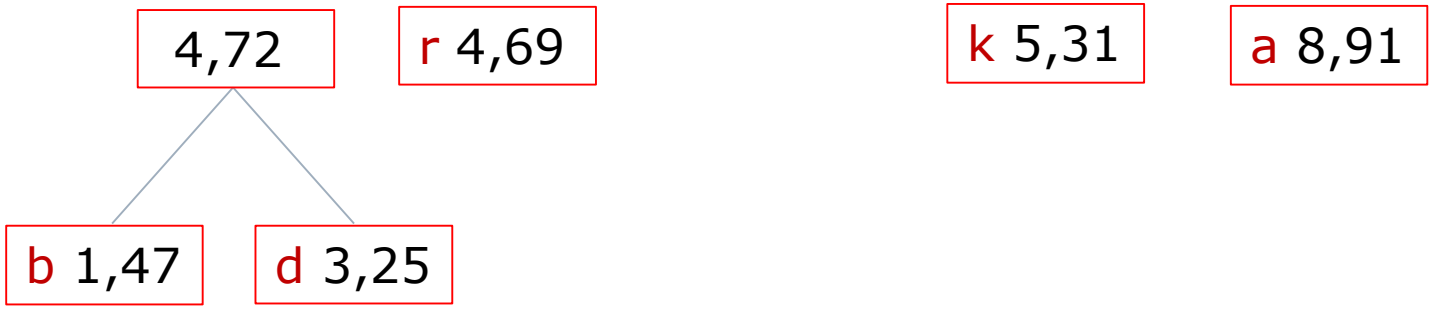
Jak zmniejszyć objętość informacji?

- Wykorzystujemy informację o częstości występowania poszczególnych znaków w tekście (%).
- Ustawiamy znaki w kolejności niemalejących częstości.
- Łączymy dwa znaki o najmniejszych częstościach i obliczamy sumę ich częstości.



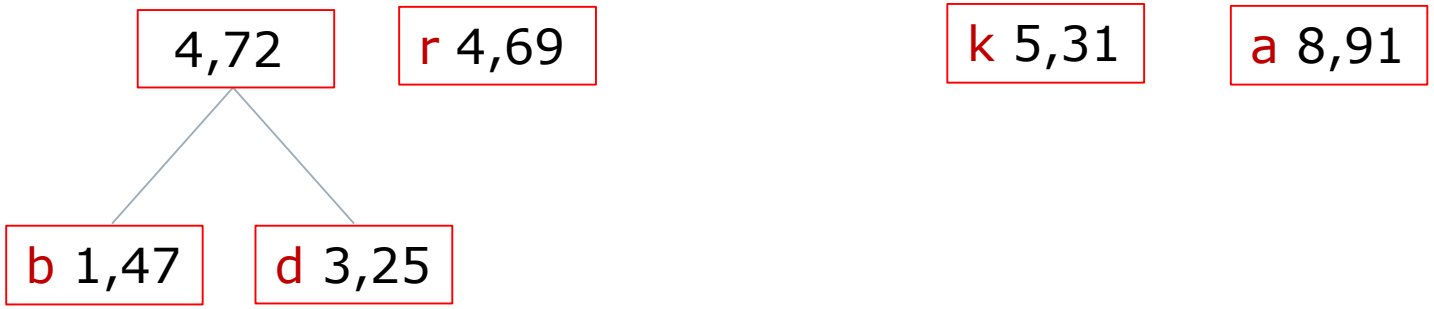
Jak zmniejszyć objętość informacji?

- Wykorzystujemy informację o częstości występowania poszczególnych znaków w tekście (%).
- Ustawiamy znaki w kolejności niemalejących częstości.
- Łączymy dwa znaki o najmniejszych częstościach i obliczamy sumę ich częstości.

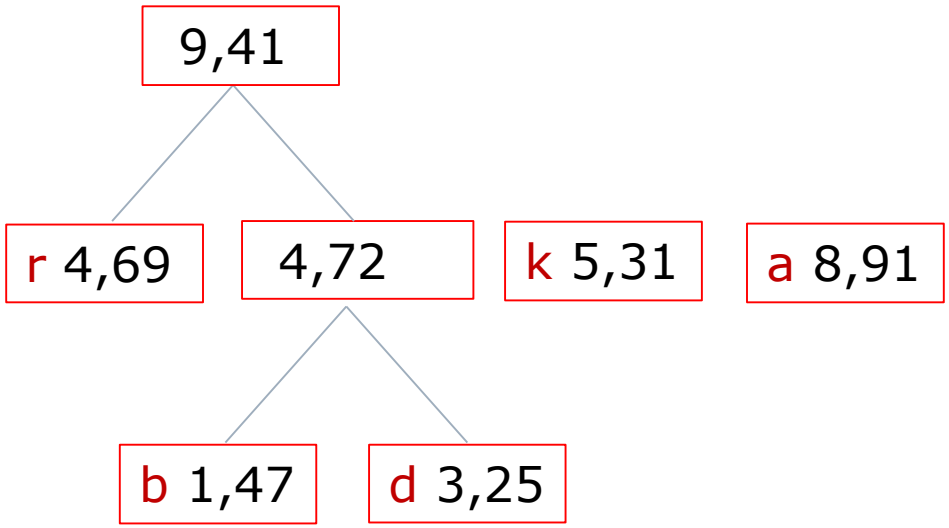


Jak zmniejszyć objętość informacji?

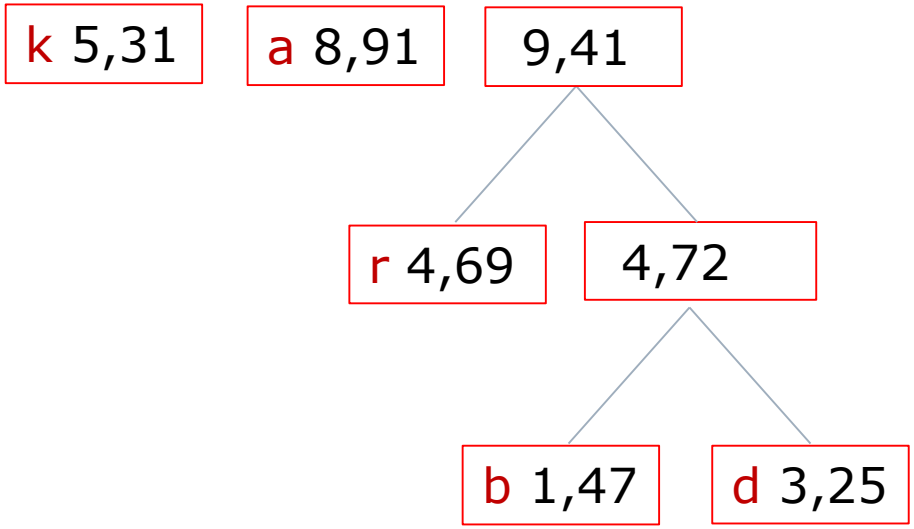
- Wykorzystujemy informację o częstości występowania poszczególnych znaków w tekście (%).
- Ustawiamy znaki w kolejności niemalejących częstości.
- Łączymy dwa znaki o najmniejszych częstościach i obliczamy sumę ich częstości.
- Sumę wstawiamy w odpowiednie miejsce uporządkowanego ciągu.
- Tak długo, jak to możliwe, powtarzamy czynności, budując drzewo.



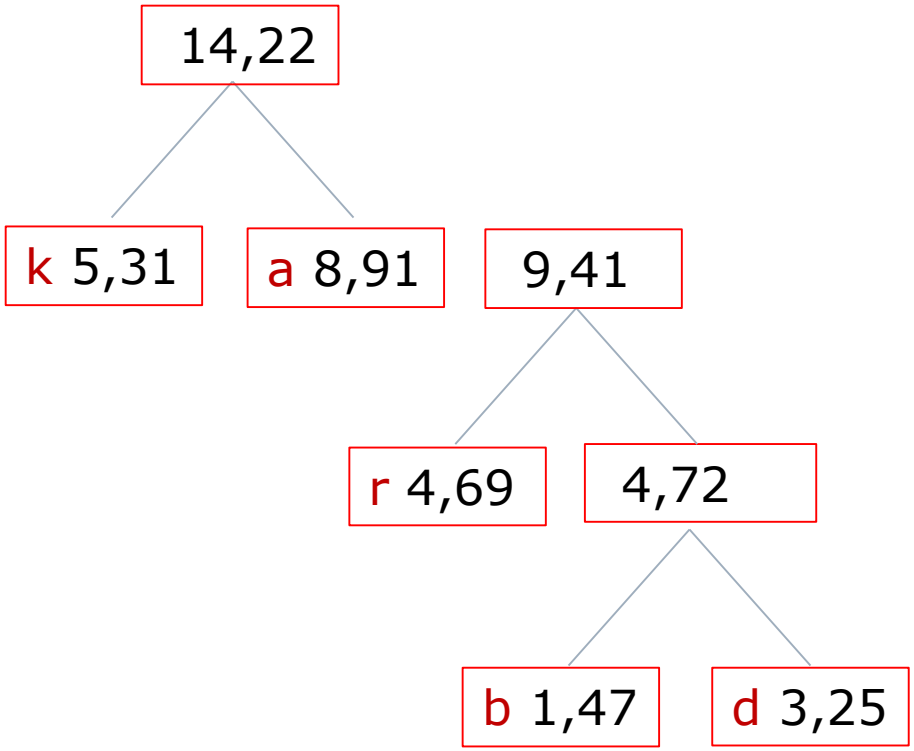
Jak zmniejszyć objętość informacji?



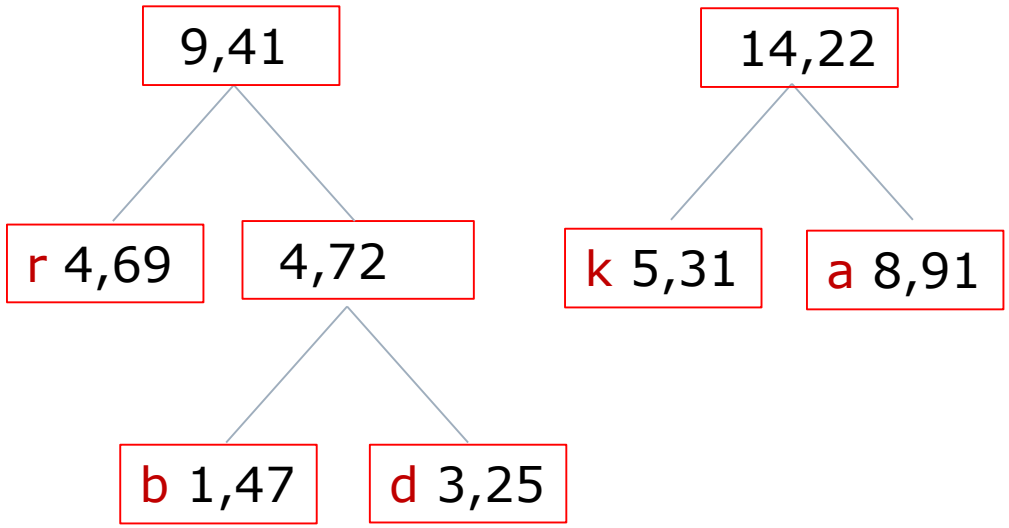
Jak zmniejszyć objętość informacji?



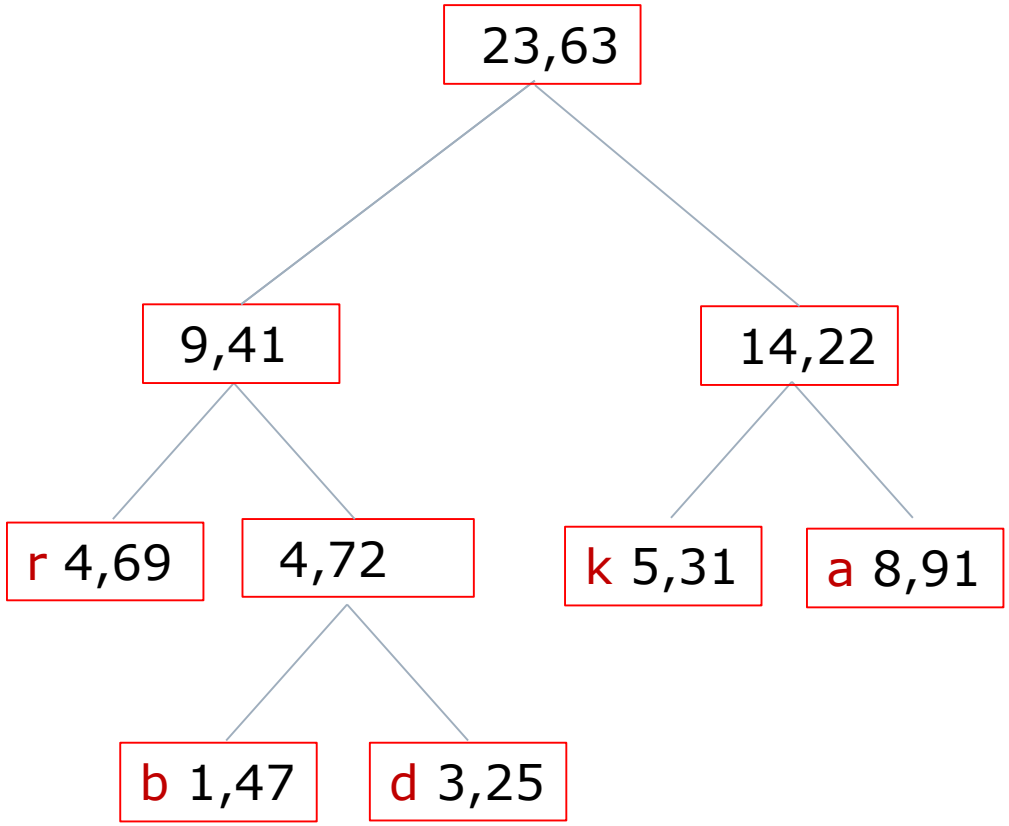
Jak zmniejszyć objętość informacji?



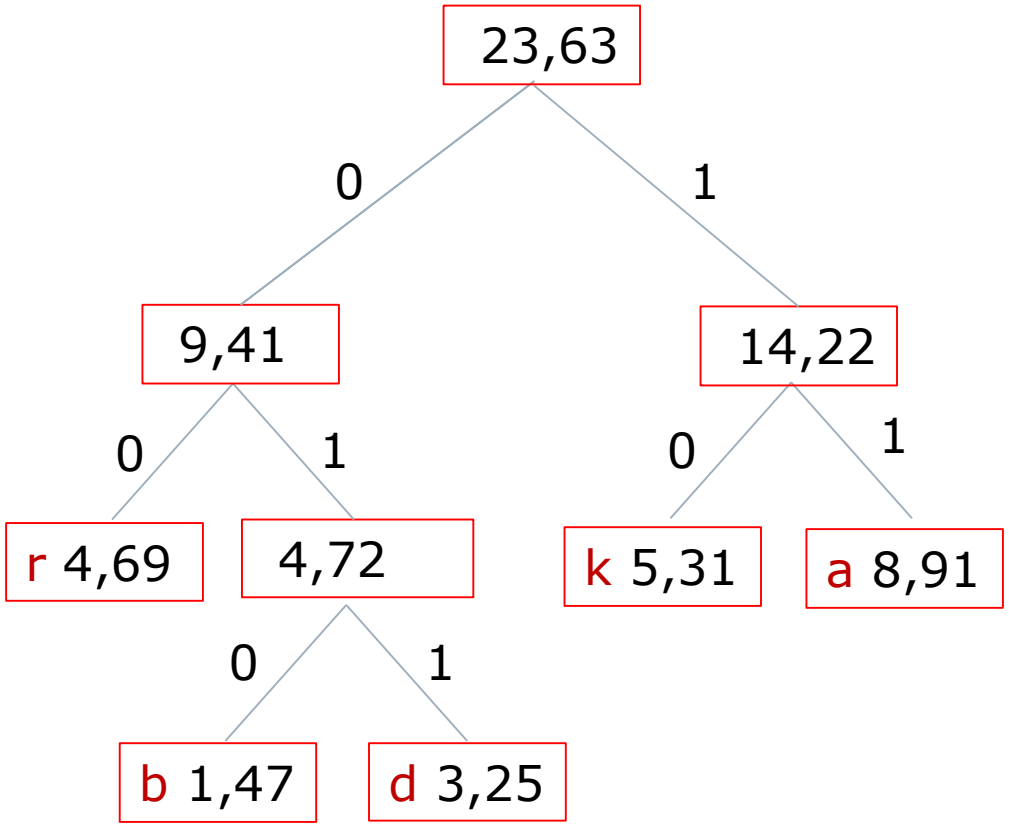
Jak zmniejszyć objętość informacji?



Jak zmniejszyć objętość informacji?



Drzewo Huffmana – kompresja bezstratna



Kody prefiksowe

a 11
b 010
d 011
k 10
r 00

Odczytaj słowo: 1101000111011011110100011 (25 bitów)

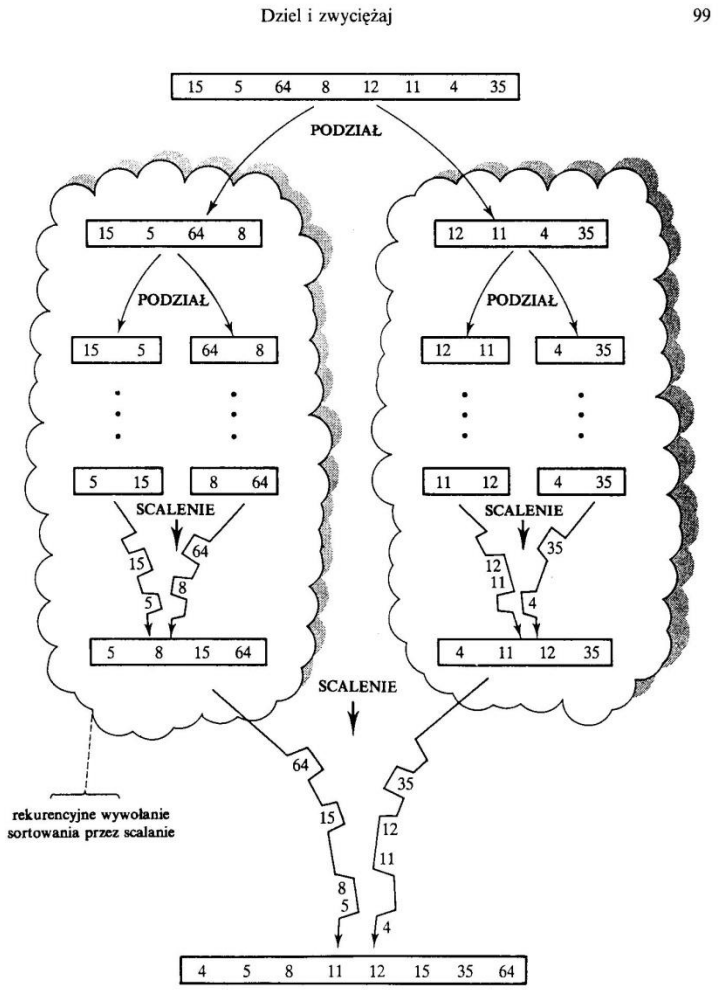
Algorytmika

Rzecz o istocie informatyki Algorytmika (1987 rok) – nauka o algorytmach

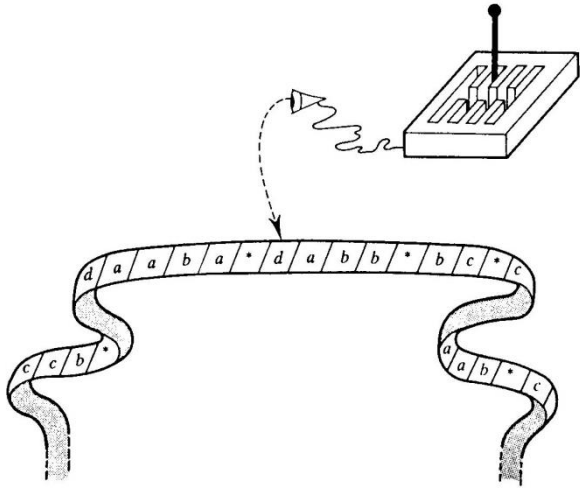
David Harel (ur. 1950 w Londynie) wykładowca informatyki w Instytucie Weizmanna w Izraelu.

Obszary badań: logika modalna, teoria obliczalności i inżynierii oprogramowania, języki wizualne, sposoby reprezentacji grafów, biologia systemów i komunikacja zapachowa, komputerowy model nicieni.

Dziel i zwyciężaj!



Uniwersalny model komputera - maszyna Turinga

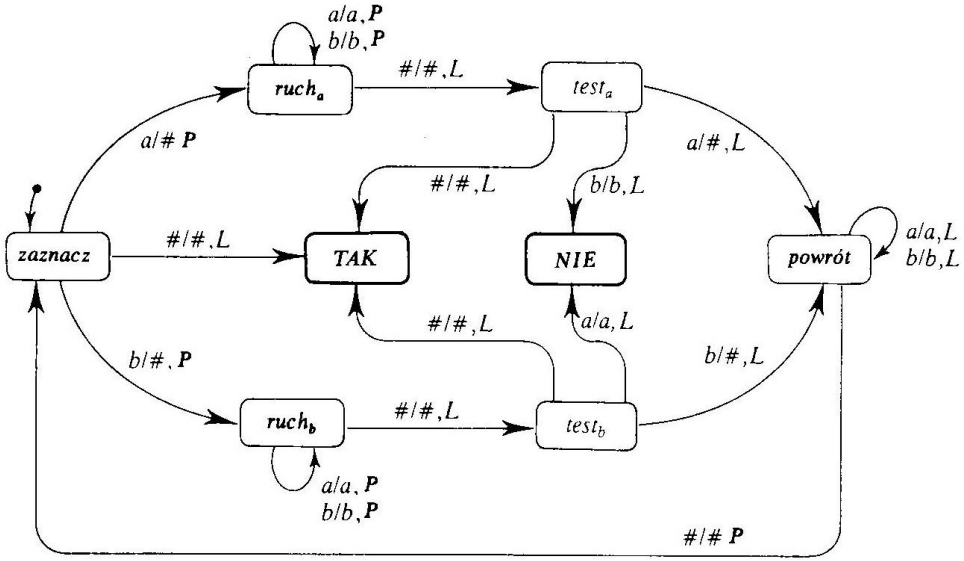


Maszyna Turinga (Alan M. Turing, 1936)

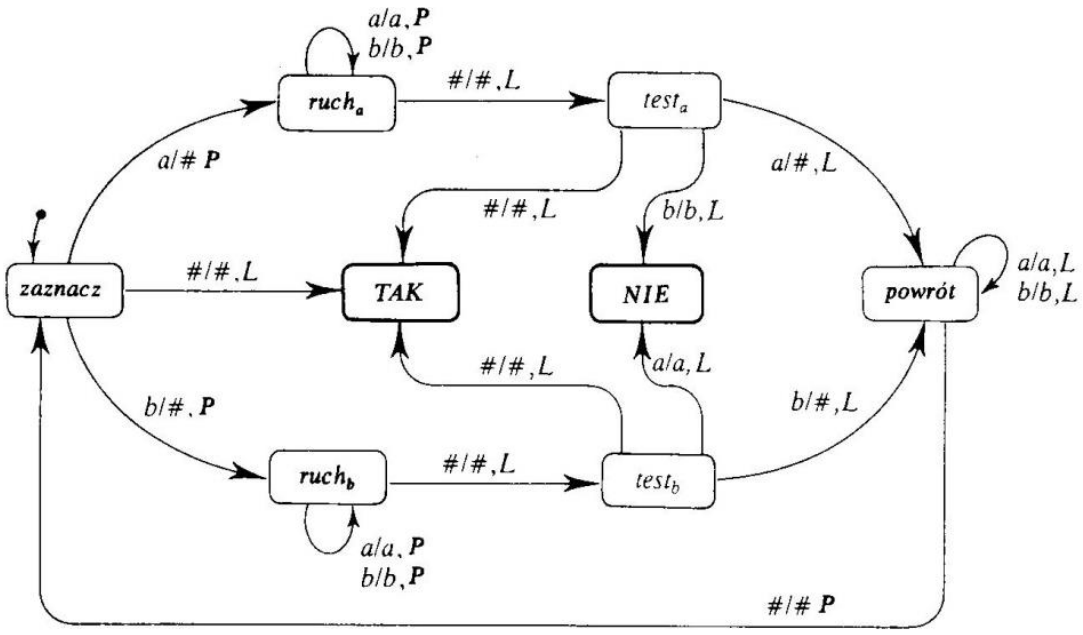
- taśma – dane, #
- skrzynia biegów – stan bieżący, stan następny
- oko o ograniczonym polu widzenia - jedno pole

Diagram przejść między stanami:

- wierzchołki oznaczają stany
- etykiety krawędzi: stan bieżący/stan następny, kierunek
- graf skierowany!

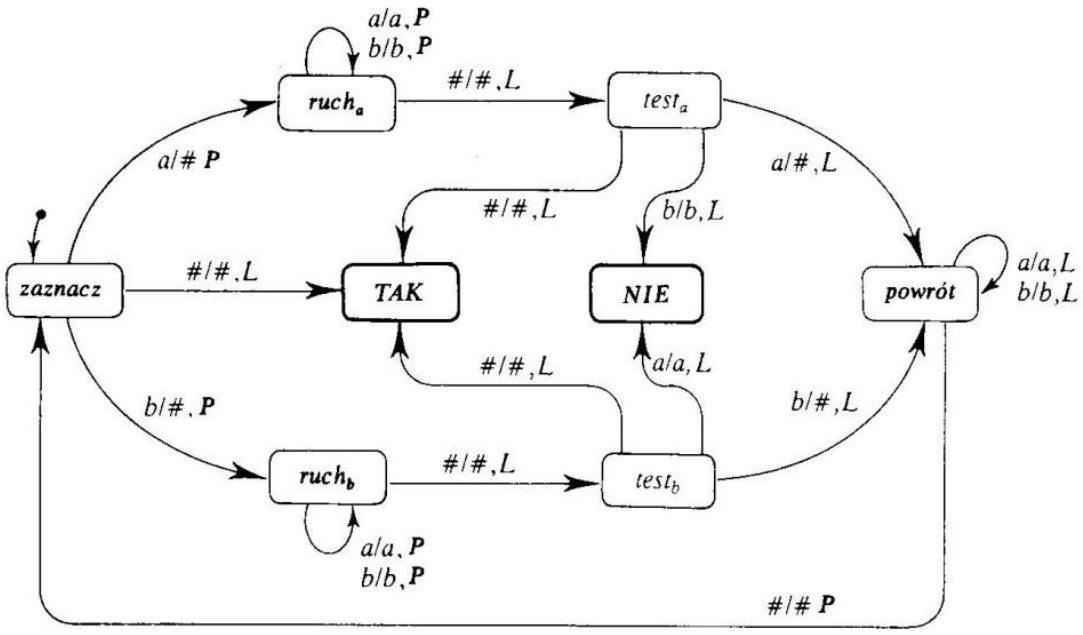


Problem: Czy słowo jest palindromem?



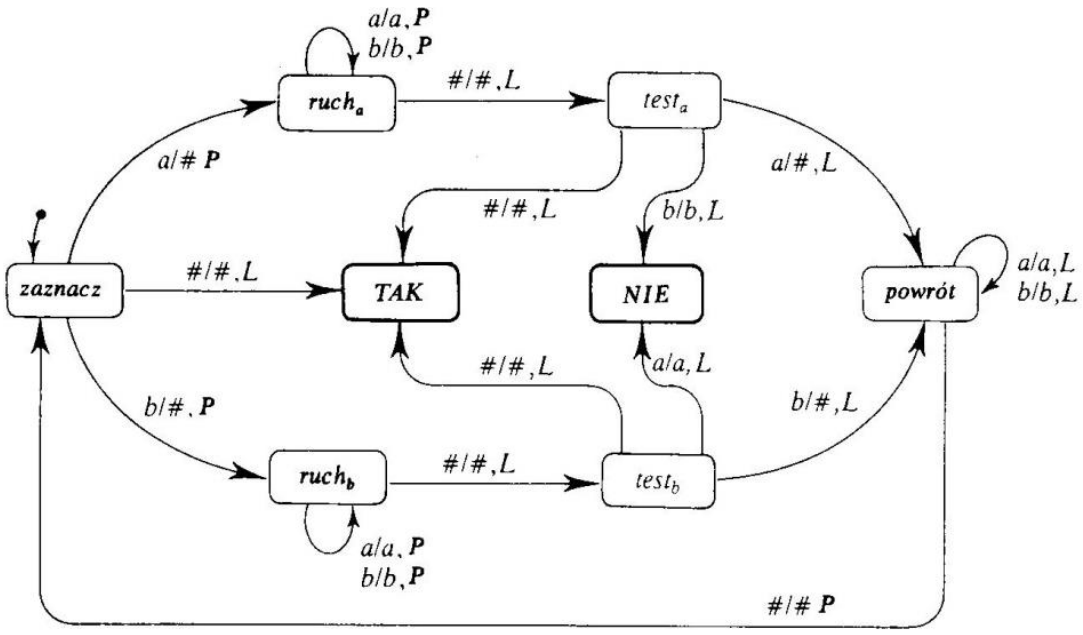
...	...	#	a	b	b	a	#	...	...

Czy słowo jest palindromem?



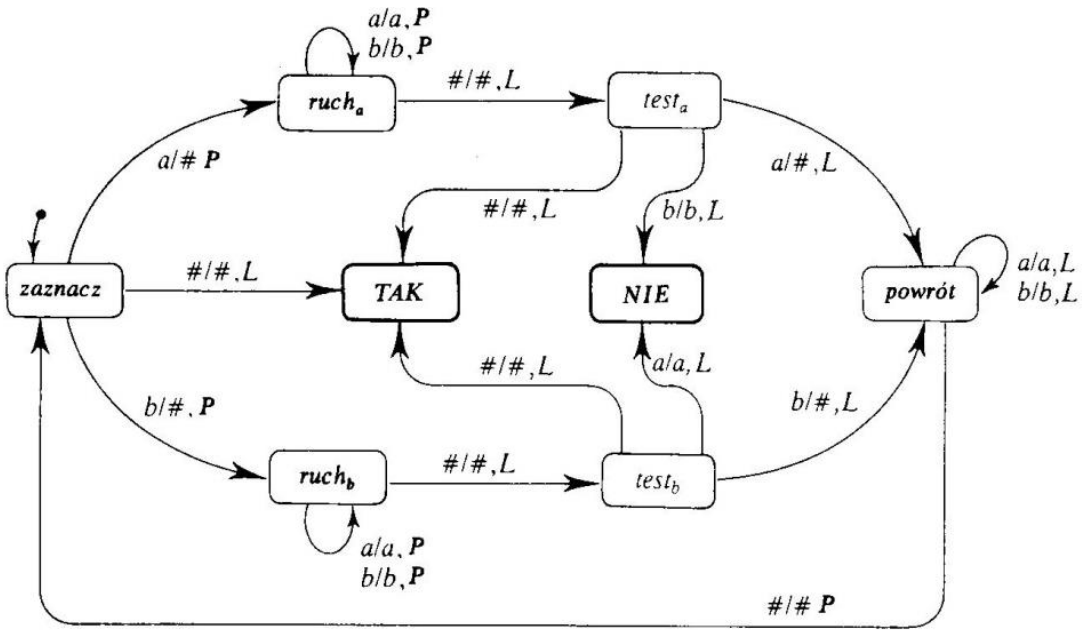
...	...	#	#	b	b	a	#	...	...

Czy słowo jest palindromem?



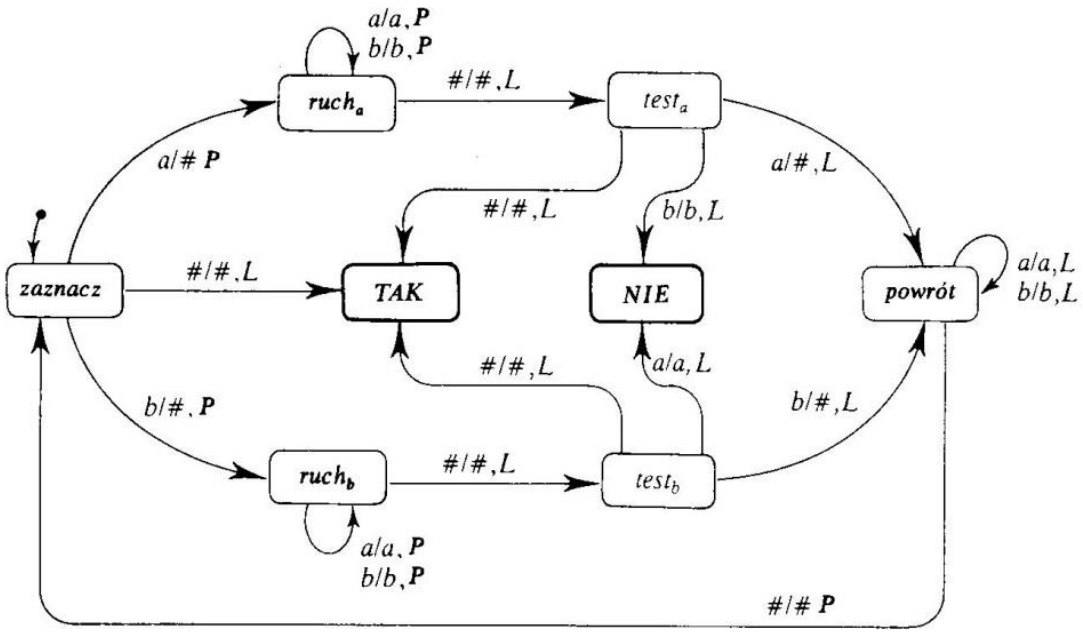
...	...	#	#	b	b	a	#	...	...

Czy słowo jest palindromem?



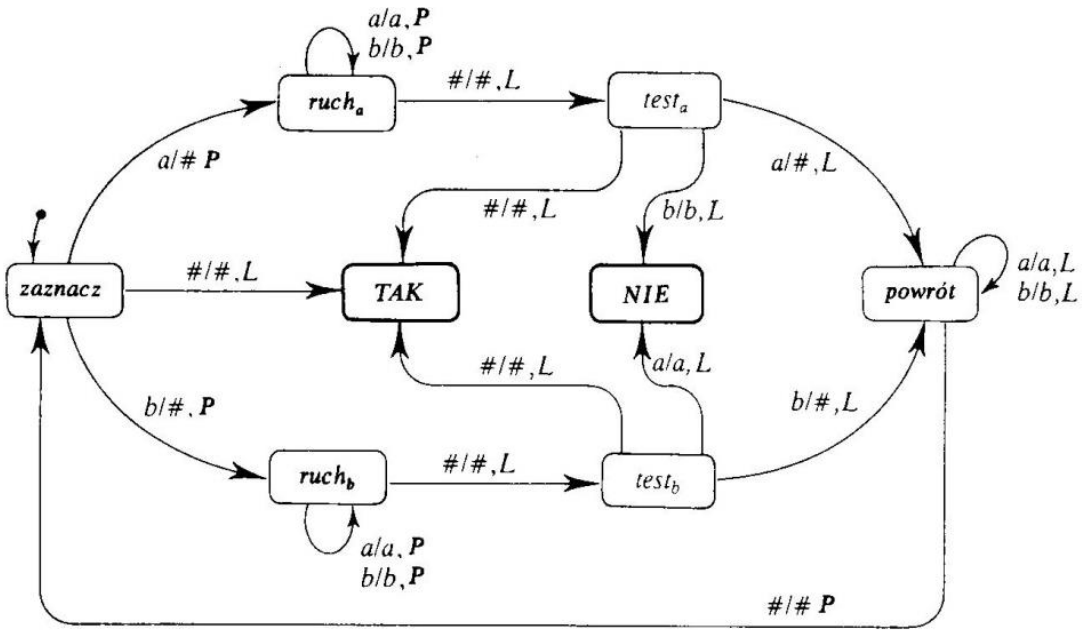
...	...	#	#	b	b	a	#	...	...

Czy słowo jest palindromem?



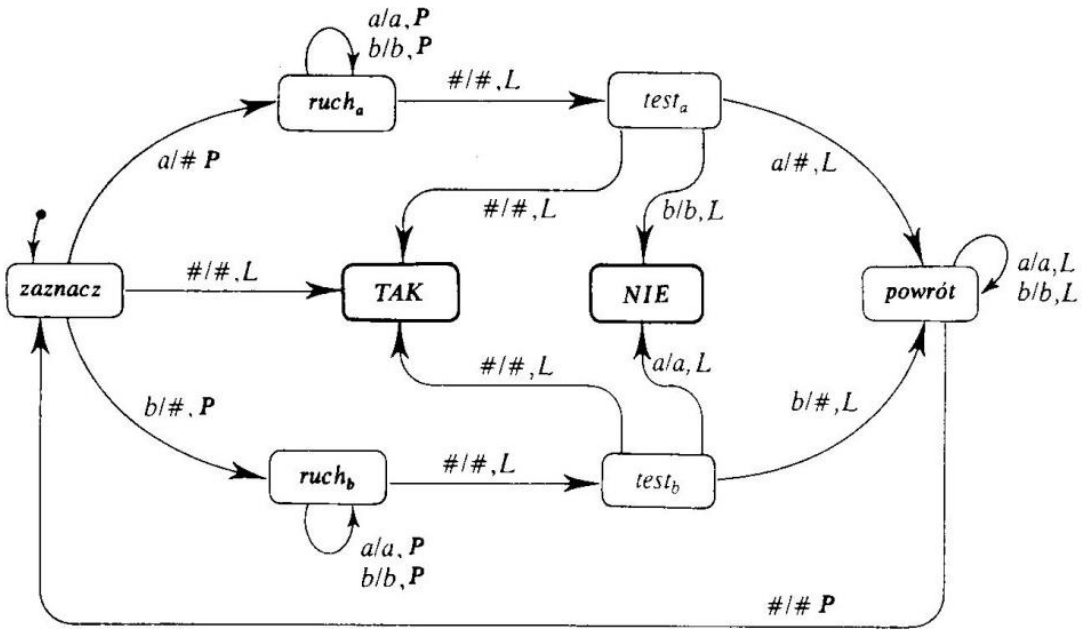
...	...	#	#	b	b	a	#	...	...

Czy słowo jest palindromem?



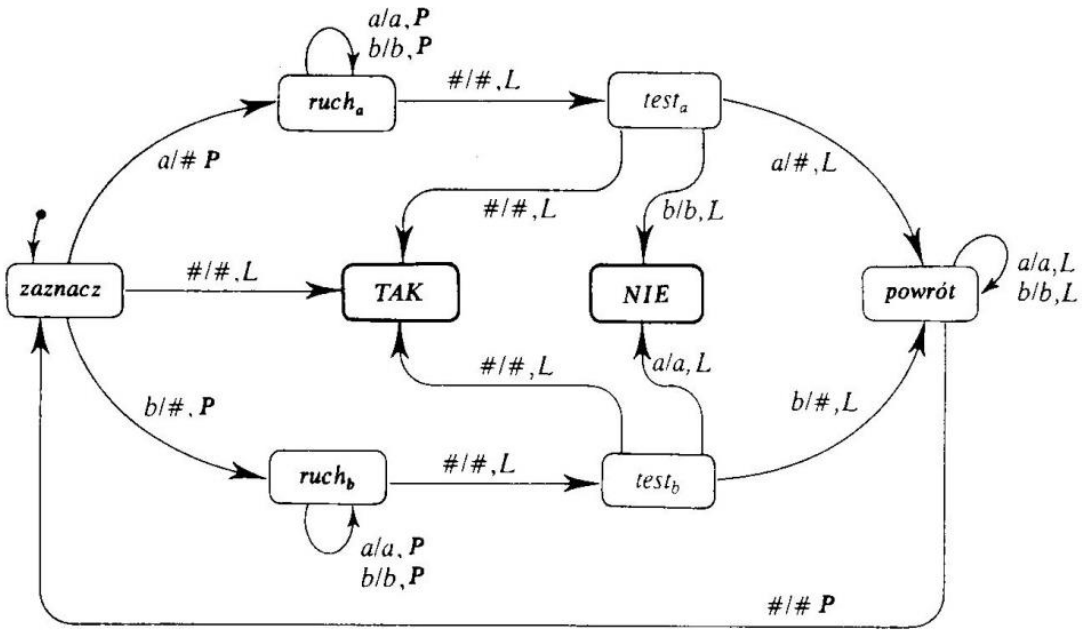
...	...	#	#	b	b	a	#	...	...

Czy słowo jest palindromem?



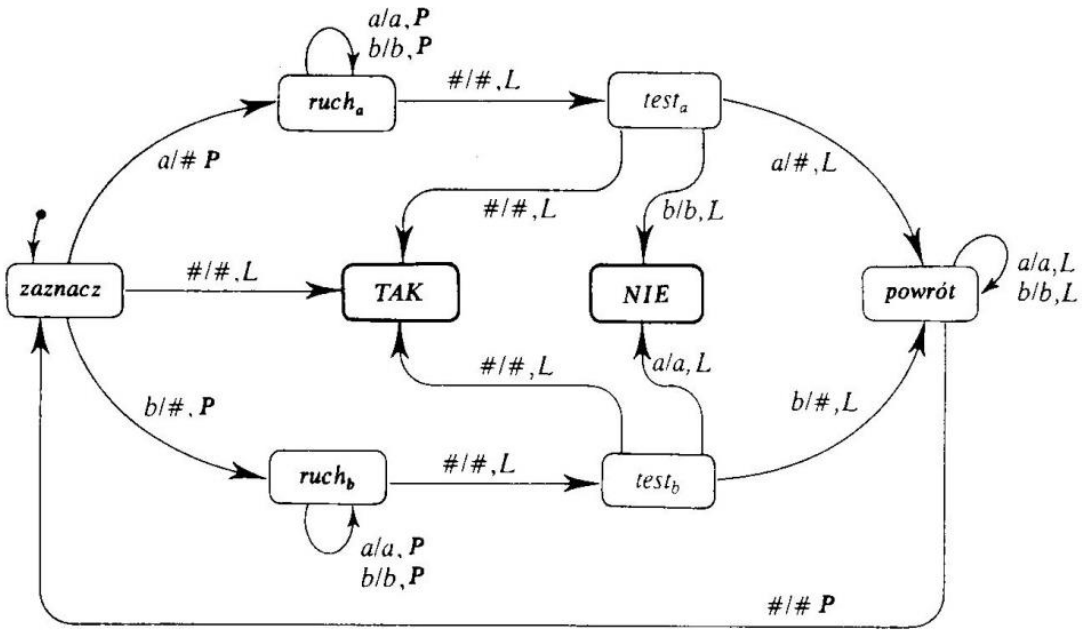
...	...	#	#	b	b	#	#	...	...

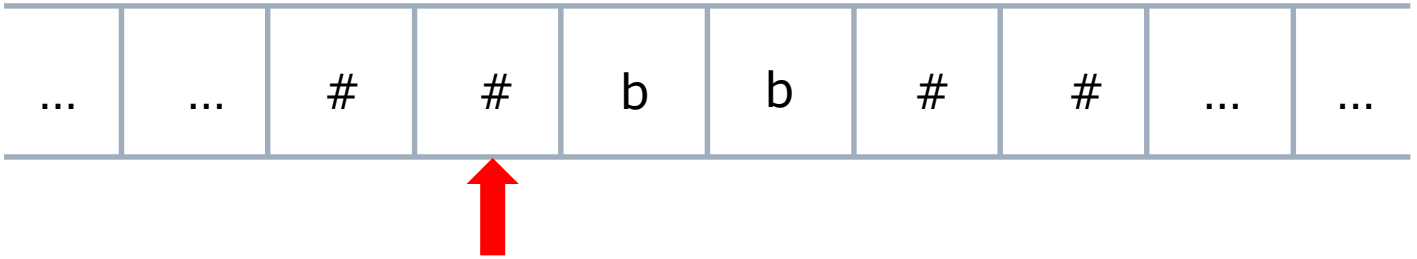
Czy słowo jest palindromem?



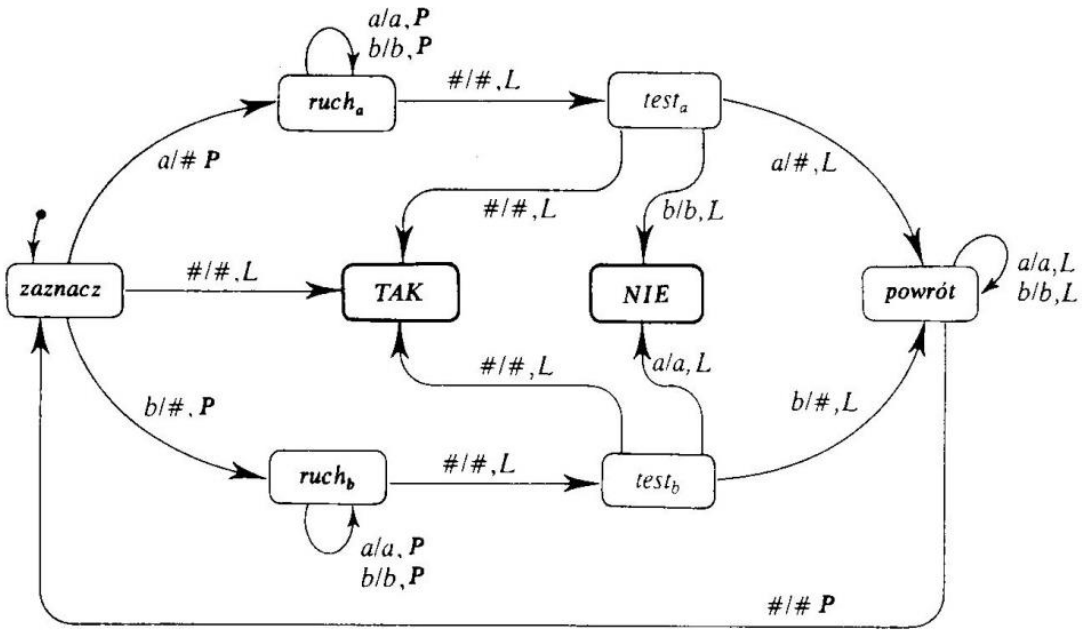
...	...	#	#	b	b	#	#	...	...

Czy słowo jest palindromem?



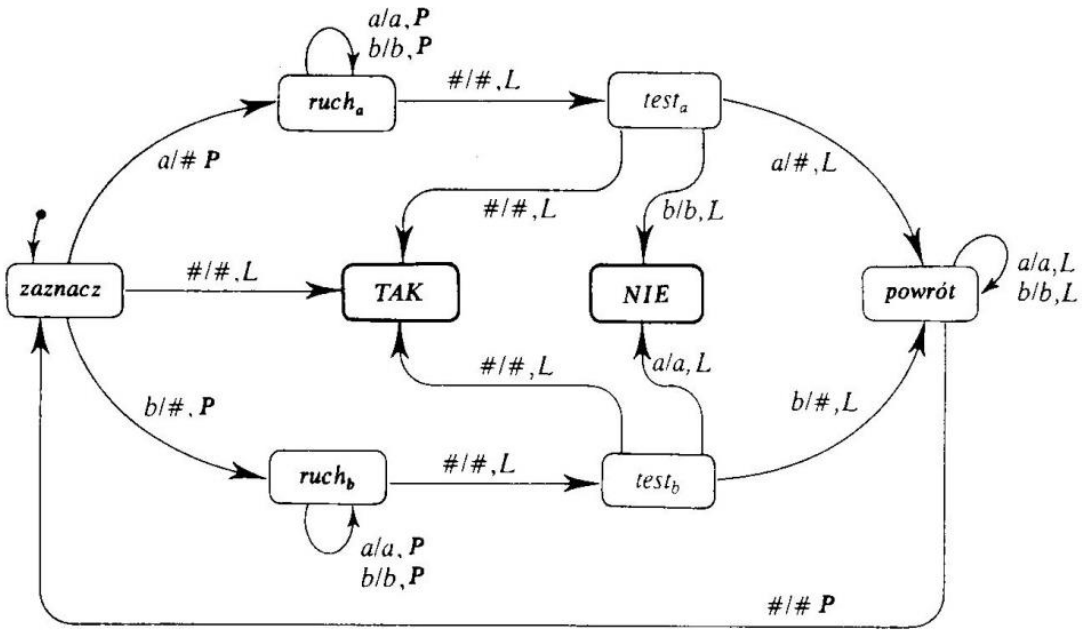


Czy słowo jest palindromem?



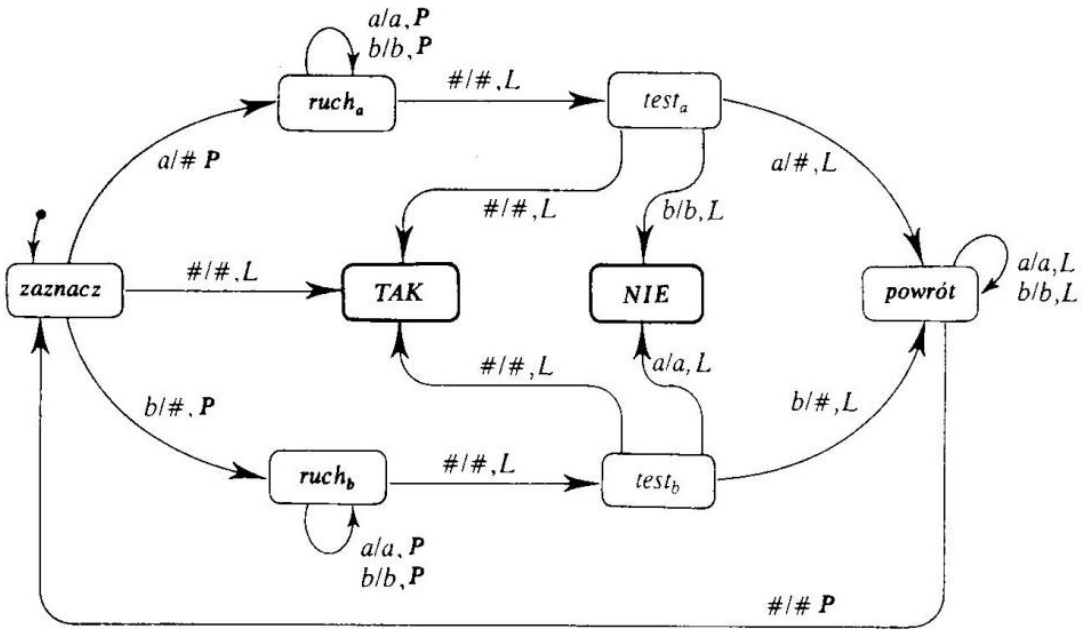
...	...	#	#	b	b	#	#	...	...

Czy słowo jest palindromem?



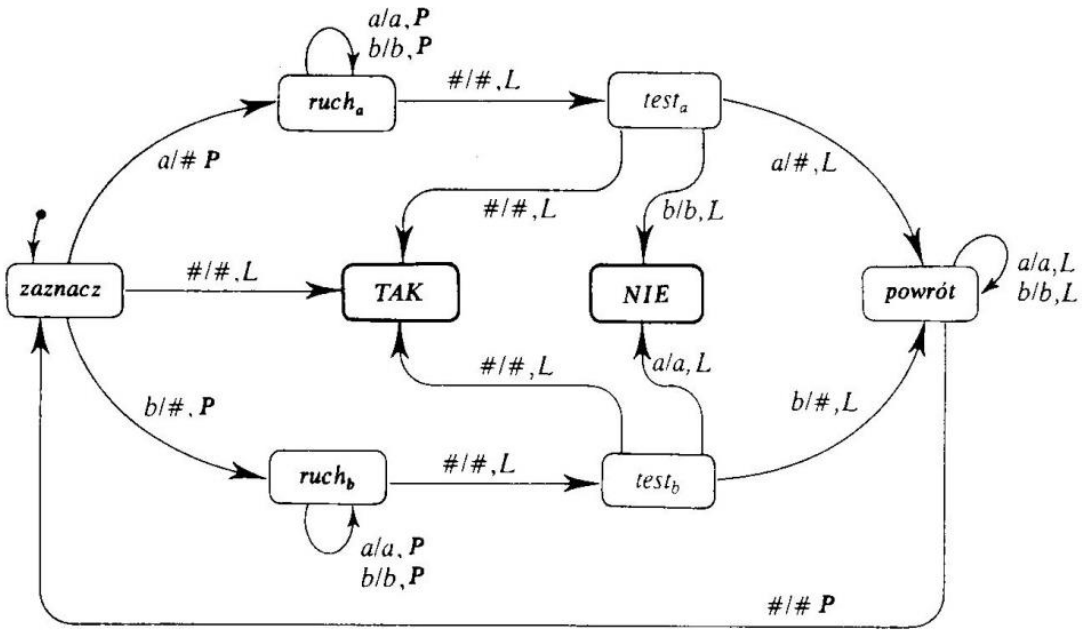
...	...	#	#	#	b	#	#	...	...

Czy słowo jest palindromem?



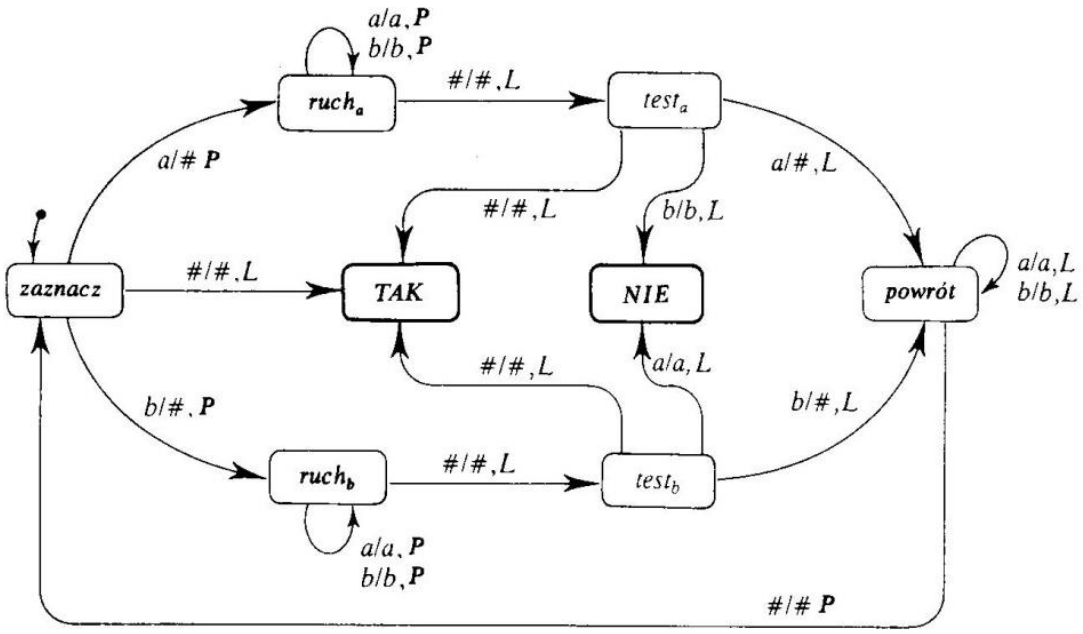
...	...	#	#	#	b	#	#	...	...

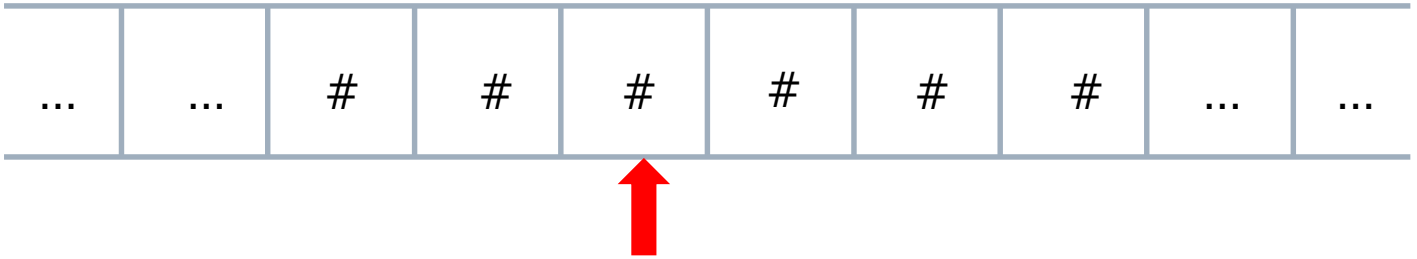
Czy słowo jest palindromem?



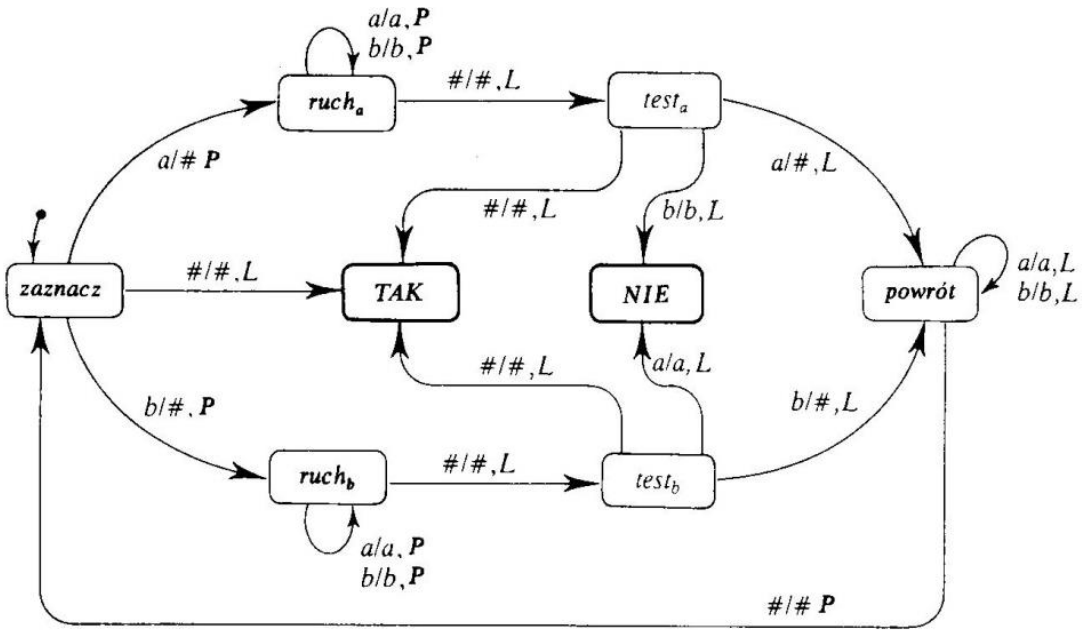
...	...	#	#	#	b	#	#	...	...

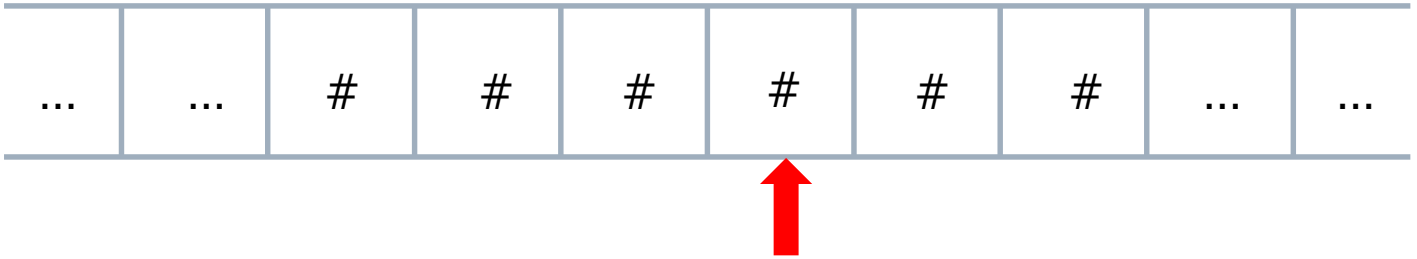
Czy słowo jest palindromem?



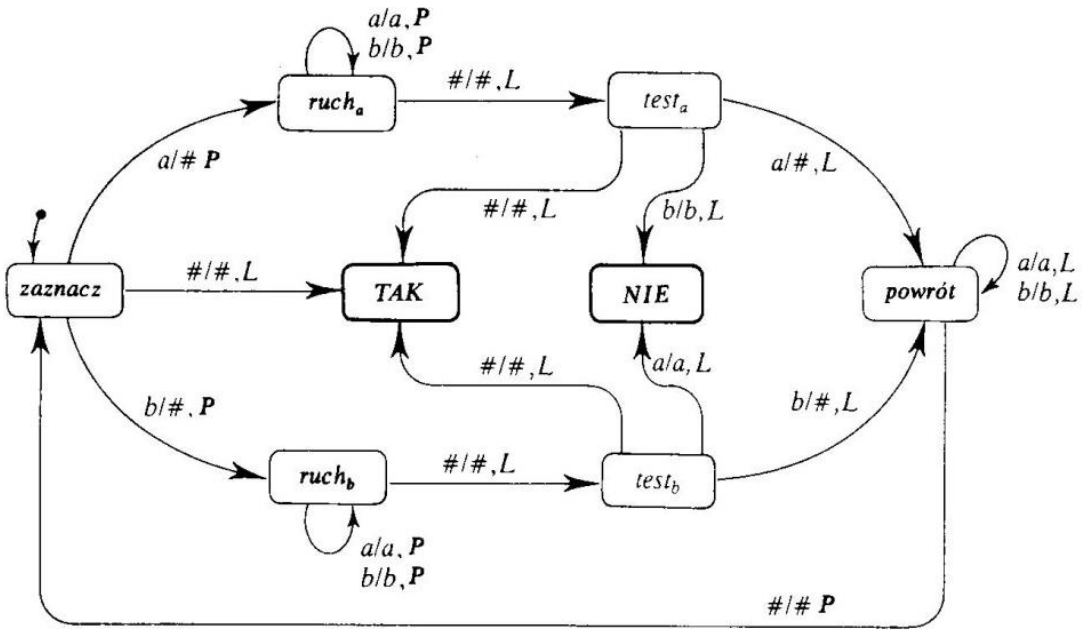


Czy słowo jest palindromem?



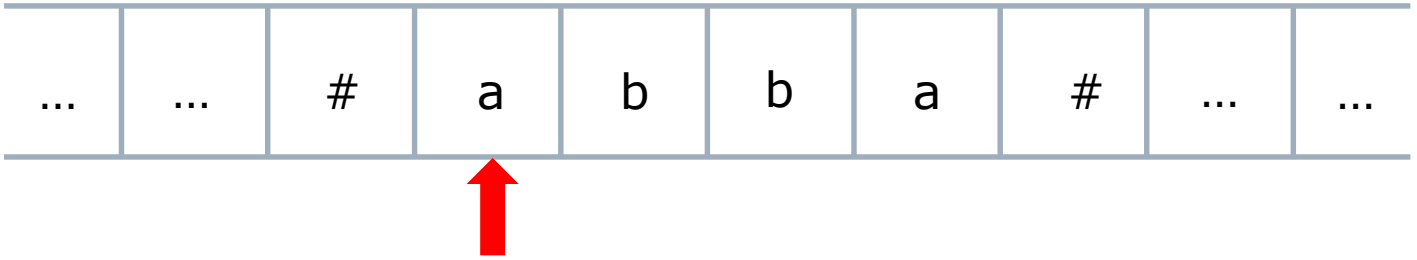


Czy słowo jest palindromem?



Maszyna Turinga a złożoność obliczeniowa algorytmów

- Maszynę Turinga możemy traktować jako komputer z jednym ustalonym programem. Oprogramowaniem jest diagram przejść, sprzęt stanowi głowica, taśma i mechanizm przejść między stanami.
- Maszyny Turinga potrafią rozwiązać każdy efektywnie rozwiązywalny problem algorytmiczny.
- **Liczba pól taśmy** wykorzystywanych do zrealizowania algorytmu odnosi się do potrzebnej do wykonania algorytmu pamięci.
- **Liczba przejść** między poszczególnymi polami taśmy odnosi się do czasu działania algorytmu.



Złożoność obliczeniowa

- Na złożoność obliczeniową algorytmu składają się:
  - złożoność pamięciowa
  - złożoność czasowa
- Złożoność pamięciowa - zależność rozmiaru pamięci potrzebnej do realizacji algorytmu od wielkości danych wejściowych, jest to ilość pamięci potrzebnej na przechowanie wszystkich zmiennych w algorytmie.
- Złożoność czasowa - zależność czasu potrzebnego do wykonania algorytmu od rozmiaru danych wejściowych. Za jednostkę czasu przy obliczaniu złożoności czasowej przyjmuje się wykonanie operacji dominującej, to jest operacji najbardziej wpływającej na czas działania danego algorytmu, np.:
  - w algorytmach numerycznych to liczba operacji arytmetycznych,
  - w algorytmach porządkowania, wyszukiwania, oprócz sortowania przez zliczanie, to liczba porównań elementów.

Złożoność asymptotyczna

- Asymptotyczne tempo wzrostu – miara określająca zachowanie wartości funkcji wraz ze wzrostem jej argumentów.

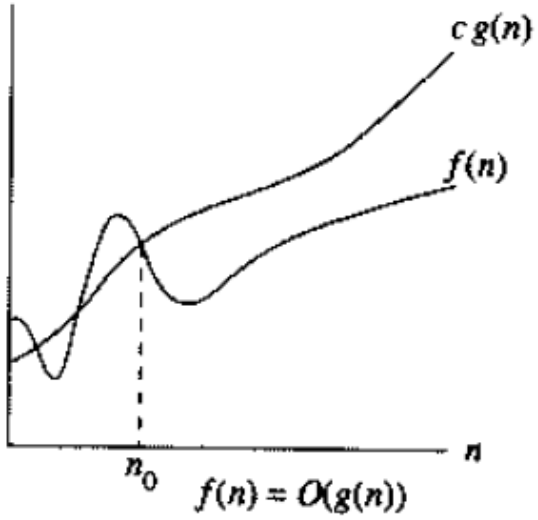
- Przy obliczaniu złożoności obliczeniowej interesuje nas, jak szybko wzrasta czas działania algorytmu (złożoność czasowa) lub wielkość pamięci (złożoność pamięciowa), gdy liczba danych rośnie do nieskończoności.

- W analizie złożoności obliczeniowej algorytmów korzystamy z notacji asymptotycznej „O wielkie”, która określa górne ograniczenie funkcji z dokładnością do stałego współczynnika.

Notacja asymptotyczna „O wielkie”

- Niech będą dane funkcje f i g, których dziedziną jest zbiór liczb naturalnych, a przeciwdziedziną zbiór liczb rzeczywistych, dodatnich.
- Def: Mówimy, że funkcja f(n) jest co najwyżej rzędu g(n), gdy:

$\exists\ n_0>0,\ c>0 \quad \forall\ n\geq n_0 \quad f(n) \leq c*g(n), \quad$ ozn. $f(n)=O(g(n))$



Własności notacji „O wielkie”

Notacja „O wielkie” ma kilka ważnych własności, które możemy wykorzystać przy szacowaniu efektywności programów, m.in.

1. Jeśli $f(n) = O(g(n))$ i $g(n) = O(h(n))$ to $f(n) = O(h(n))$
2. Jeśli $f(n) = O(h(n))$ i $g(n) = O(h(n))$ to $f(n) + g(n) = O(h(n))$

Dowód 1.:

Niech n_f i c_f będą takie, że: $\forall n \geq n_f \quad f(n) \leq c_f * g(n)$,

oraz niech n_g i c_g będą takie, że $\forall n \geq n_g \quad g(n) \leq c_g * h(n)$.
Wystarczy przyjąć $n_{f(g)} = \max(n_f, n_g)$, $c_{f(g)} = c_f * c_g$

Dowód 2.:

Niech n_f i c_f będą takie, że: $\forall n \geq n_f \quad f(n) \leq c_f * h(n)$,

oraz niech n_g i c_g będą takie, że $\forall n \geq n_g \quad g(n) \leq c_g * h(n)$.
Wystarczy przyjąć $n_{f+g} = \max(n_f, n_g)$, $c_{f+g} = c_f + c_g$

Przykłady

Przykład:

Pokaż, że $f(n)= 3*n^3+ 2*n^2+n+1$ jest rzędu $O(n^3)$

Wystarczy znaleźć takie $n_0>0$ i $c>0$, że $\forall n \geq n_0$ $3*n^3+ 2*n^2+n+1 \leq c*n^3$

Ponieważ:

$3*n^3+ 2*n^2+n+1 \leq 3*n^3+ 2*n^3+n^3+n^3=7*n^3$ dla każdego n,

więc wystarczyprzyjąć $n_0=1$, $c=7$.

Przykład:

Pokaż, że $f(n)= n^5$ jest rzędu $O(n^6)$

Wystarczy przyjąć $n_0=1$ i $c=1$

Przykład:

Pokaż, że $f(n)= \log_{10}n^6$ jest rzędu $O(\log_2 n)$

$\log_{10}n^6 = 6*(\log_2 n)/(\log_2 10) = (6*/\log_2 10)\ (\log_2 n)$

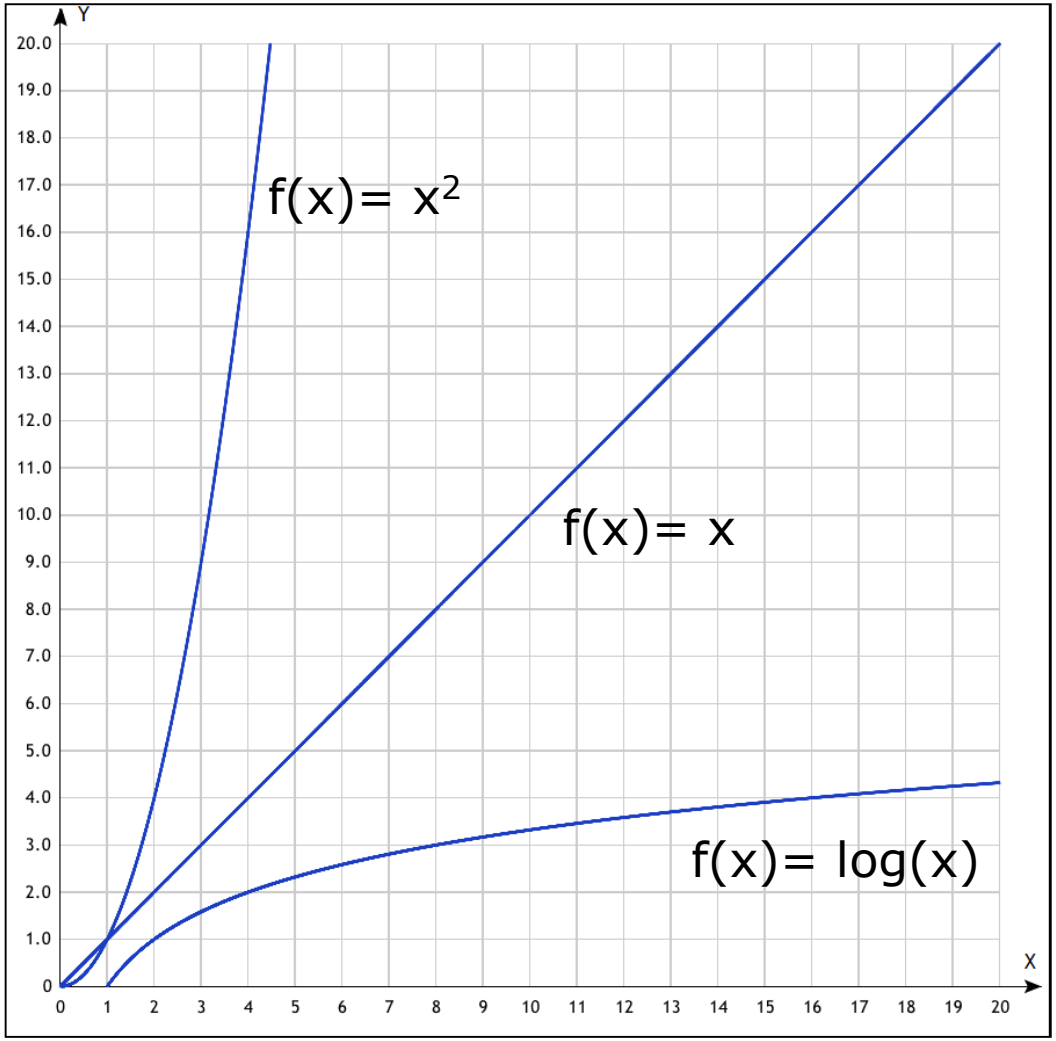
Wystarczy przyjąć $n_0=1$, $c= 6*/\log_2 10$

Rząd złożoności obliczeniowej

W zależności od asymptotycznego tempa wzrostu, funkcje dzieli się na rzędy złożoności obliczeniowej. Najczęściej wyróżnia się złożoności:

$O(1)$	stała (stała liczba operacji niezależnie od liczby danych wejściowych)
$O(\log n)$	logarytmiczna
$O(n)$	liniowa
$O(n*\log n)$	liniowo-logarytmiczna
$O(n^2)$	kwadratowa
$O(n^c)$	wielomianowa
$O(c^n)$, $O(n!)$	wykładnicza

Porównanie funkcji rzędów złożoności



Złożoność stała O(1)

- Algorytm wykonuje stałą liczbę operacji dominujących bez względu na rozmiar danych wejściowych.
  - operacje arytmetyczne, np. +, -
  - operacje logiczne, and, or, not
  - operacje dostępu do struktur danych, np. indeksowanie tablic
  - przypisanie wartości zmiennej
  - wywołanie funkcji bibliotecznych, takich jak np. print()

- Przykłady:
  - Obliczanie liczby rozwiązań równania kwadratowego
  - Obliczanie wartości bezwzględnej liczby rzeczywistej
  - Obliczanie wartości wyrażeń stałych

Złożoność logarytmiczna *O(log n)*

- Złożoność O(log n) mają na przykład algorytmy, których problem postawiony dla danych rozmiaru n da się sprowadzić w pesymistycznym przypadku do problemu z rozmiarem danych o połowę mniejszym.

- Przykłady:
  - binarne wyszukiwanie, wstawianie
  - szukanie liczby bitów potrzebnych do reprezentowana liczby w komputerze
  - szybkie potęgowanie
  - obliczanie NWD dwóch liczb algorytmem Euklidesa
  - Liczba kroków w algorytmie „dziel i zwyciężaj”

Złożoność liniowa *O(n)*

- Taką złożoność mają na przykład algorytmy, które dla każdej danej wykonują w pesymistycznym przypadku stałą liczbę operacji podstawowych.

- Przykłady:
  - szukanie minimum, maksimum
  - jednoczesne szukanie minimum i maksimum
  - wyszukiwanie lidera, idola
  - szukanie spójnego podciągu o sumie równej ustalonej liczbie
  - porównywanie łańcuchów
  - ...

Schemat Hornera

- Dane:
  - wielomian $W_n(x) = a_0*x^n + a_1*x^{n-1} + \ldots + a_{n-1}*x + a_n$, gdzie $a_0 \neq 0$, a_i dla $0 \leq i \leq n$ są rzeczywiste,
  - x_0 rzeczywiste
- Wynik:
  - Wartość $W_n(x_o)$

Przy obliczeniach wg podanego wzoru liczba mnożeń:

$Mn(n) = n + n-1 + n-2 + \ldots + 1 = (n+1)*n/2.$

Więc $Mn(n)=O(n^2)$ – rząd kwadratowy

Przekształcając odpowiednio wielomian:

$$W_n(x) = a_0*x^n + a_1*x^{n-1} + \ldots + a_{n-1}*x + a_n\ (x) =$$
$$= (a_0*x^{n-1} + a_1*x^{n-2} + \ldots + a_{n-1})*x + a_n = \ldots =$$
$$= (a_0*x + a_1)*x + \ldots + a_{n-1})*x + a_n$$

Liczba mnożeń $Mn(n)=O(n)$ – rząd liniowy

Schemat Hornera

- Wersja iteracyjna funkcji

```
def Horner(n,lista,x):
  y=lista[0]
  for i in range(1,n+1,1):
    y=y*x+lista[i]
  return y
```

- Wersja rekurencyjna funkcji

```
def HornerRek(n,a,x):
  if n == 0:
    return a[0]
  else:
    return HornerRek(n-1,a,x)*x+a[n]
```

Problem sumy w ciągach uporządkowanych

- Dane:
  - n, x naturalne
  - dwa ciągi n elementowe A i B uporządkowane niemalejąco
- Wynik:
  - odpowiedź na pytanie, czy istnieje $a \in A$ i $b \in B$ takie, że a+b =x?

Rozwiązania o różnych złożonościach obliczeniowych:

- Algorytm naiwny: sprawdzamy sumę każdego elementu z ciągu A z każdym elementem ciągu B. Dla każdego elementu z ciągu A wykonujemy n porównań elementów. Algorytm ma złożoność $O(n^2)$. Jest to rozwiązanie, które nie wykorzystuje uporządkowania ciągów.
- Dla każdego elementu ciągu A sprawdzany sumę z elementem ciągu B wyszukanym algorytmem binarnym. Algorytm ma złożoność obliczeniową O(n*logn).
- Czy potrafisz podać algorytm o złożoności liniowej O(n)?

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za dużo!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za dużo!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za dużo!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za mało!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za dużo!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za mało!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za dużo!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za dużo!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za mało!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za mało!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za mało!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za dużo!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

Za mało!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

W sam raz!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

Algorytm liniowy dla sumy w ciągach uporządkowanych

- Przykładowe dane, x=34

0	1	2	3	4	5	6	7	8	9
4	6	11	15	16	20	25	30	33	40

W sam raz!

0	1	2	3	4	5	6	7	8	9
2	7	9	17	24	26	29	31	32	34

- W przypadku położenia szukanych elementów w pierwszym ciągu na ostatnim miejscu, a w drugim ciągu na pierwszym, lub gdy nie ma takich elementów, mamy najwięcej, bo 2*n porównań elementów ciągów.

Złożoność liniowo-logarytmiczna *O(n log n)*

- Taką złożoność mają na przykład algorytmy, w których problem postawiony dla danych rozmiaru n da się sprowadzić w liniowej liczbie operacji do rozwiązania dwóch problemów o rozmiarach *n/2*

- Przykłady:
  - sortowanie przez scalanie
  - sortowanie szybkie
  - sortowanie na kopcu

Złożoność kwadratowa $O(n^2)$

- taką złożoność mają na przykład algorytmy, w których dla każdej pary elementów danych wykonywana jest stała liczba operacji podstawowych (zwykle algorytmy z podwójnymi pętlami).

- Przykłady:
  - sortowanie przez proste wstawianie
  - sortowanie przez proste wybieranie
  - sortowanie bąbelkowe

Złożoność wykładnicza $O(2^n)$, $O(n!)$

- Są to algorytmy bardzo wolne, których realizacja w pesymistycznym przypadku jest niewykonalna nawet dla niewielkich rozmiarów danych.

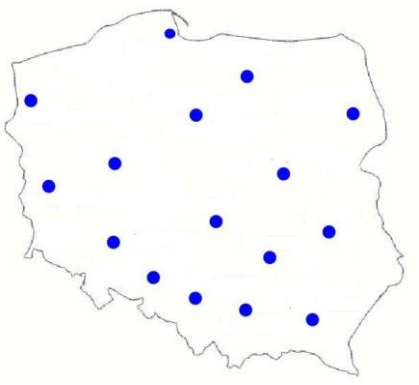
- Przykłady:
  - Wieże Hanoi
  - Obliczanie liczby permutacji

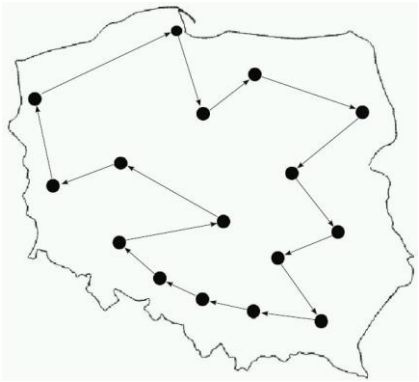
Algorytm optymalny

- Działania mające na celu opracowanie algorytmu najlepiej, w sensie złożoności obliczeniowej, rozwiązującego dany problem nazywamy **optymalizacją algorytmu**. Na etapie optymalizacji bierze się pod uwagę zarówno złożoność czasową, jak i pamięciową.
- Algorytm optymalny to algorytm o najlepszej złożoności obliczeniowej dla danego problemu.
- Przykłady:
  - algorytmy sortowania przez porównania o złożoności O(n*logn)
  - algorytm Euklidesa
  - liniowe szukanie idola, lidera
- Na złożoność obliczeniową algorytmu mają również wpływ użyte struktury danych.
  - algorytm dla idola – dla n osób musimy pamiętać n^2 operacji,
  - dobór reprezentacji grafu do algorytmu.

Problemy trudne - najkrótsza trasa premiera

Problem: Znajdź najkrótszą trasę dla Premiera przez wszystkie miasta wojewódzkie.





Rozwiązanie: Premier zaczyna w Stolicy a inne miasta może odwiedzać w dowolnej kolejności. Tych możliwości jest:

$$15*14*13*12*11*\ldots*2*1 = 15!$$ (15 silnia)

W 1990 roku było: $48*47*46*\ldots*2*1 = 48!$ (48 silnia)

Najkrótsza trasa premiera

Wartości funkcji *n*!
Rosną BARDZO SZYBKO

n	*n*!
10	3628800
15	$1.30767*10^{12}$
20	$2.4329*10^{18}$
25	$1.55112*10^{25}$
30	$2.65253*10^{32}$
40	$8.15915*10^{47}$
48	$1.24139*10^{61}$
100	$9.3326*10^{157}$

Prezydent Stanów Zjednoczonych ma problem ze znalezieniem najkrótszej trasy objazdu Stanów.

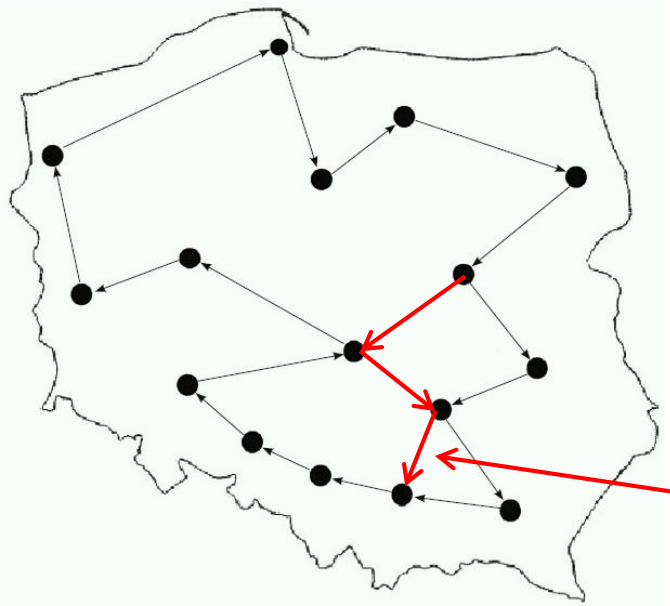
Na superkomputerze o mocy 1 PFlops – ile trwa obliczanie n!

15! = $1307674368000/10^{15}$ sek. = ok. 0.01 sek.

48! = $1,241391559253607267086228904737 3*10^{61}/10^{15}$ = $3*10^{38}$ lat

25! = $15511210043330985984000000/10^{15}$ sek. = 15511210043 sek. = 179528 dni = 491 lat

Najkrótsza trasa premiera



Trudno sprawdzić, jak dobre jest to rozwiązanie w stosunku do najlepszego, bo go nie znamy.

Zły wybór

Algorytmy przybliżone szukania rozwiązań:

1. Metoda zachłanna – najbliższy sąsiad – mogą być bardzo złe
2. Meta-heurystyki:
   - algorytmy genetyczne – krzyżowanie i mutowanie rozwiązań
   - algorytmy mrówkowe – modelowanie feromonów

Dziękuję za uwagę

Anna Beata Kwiatkowska
aba@mat.umk.pl