I) Podstawowe definicje

I - 1) **Informatyka** – nauka o przetwarzaniu informacji.

I - 2) **Zadanie algorytmiczne** – polega na określeniu:

- wszystkich poprawnych danych wejściowych
- oczekiwanych wyników, jako funkcji danych wejściowych

I - 3) **Algorytm** – specyfikacja ciągu elementarnych operacji, które przekształcają dane wejściowe na wyniki.

I - 3.a) Może występować w postaci:

- werbalnej
- symbolicznej (schemat blokowy)
- programu

I - 3.b) **Algorytm Euklidesa**, obliczający największy wspólny dzielnik. !

-
```
cin >> a, b;
while (a != b)
{
    if (a>b)    a = a-b;
    else        b = b-a;
}
cout << a;
```

I - 3.c) Algorytm „dobry" **rozkładu liczby na czynniki pierwsze !**

-
```
int b=3, n;

cin >> n;
while ( n%2 == 0 )
{
    wypisz(2);
    n = n/2;
}

while ( n > 1 && b < sqrt(n) )
{
    if ( n%b == 0 )
    {
        wypisz(b);
        n = n/b;
    }
    else    b = b+2;
}

if ( n > 1 )    wypisz(n);
```

I - 4) Aspekty języka programowania

I - 4.a) Syntaktyka (składnia)

- Zbiór reguł dot. konstrukcji językowych

I - 4.b) Semantyka

- Opisuje znaczenie konstrukcji językowych

I - 4.c) Pragmatyka

- Opisuje wszystkie inne aspekty języka

I - 5) Instrukcje proste i strukturalne !!!

I - 5.a) Instrukcje proste

- przypisania
- wywołania funkcji
- operacje wejścia/wyjścia

I - 5.b) Instrukcje strukturalne

- warunkowa
- wyboru
- iteracyjne

I - 6) Typy danych

- Skalarne
  - uporządkowane i skończone zbiory wartości
    (i) np. int, bool, float, char…
- Strukturalne
  (i) Wskaźnikowe

I - 7) Złożoność obliczeniowa

I - 7.a) Definiuje się ją, jako ilość zasobów komputera potrzebnych do jego wykonania.

(i) Wchodzi w to wielkość pamięci, czas procesora…

I - 7.b) Wyróżnia się:

- Złożoność pesymistyczna
  - Ilość zasobów potrzebna przy „najgorszych” danych wejścia rozmiaru n.
- Złożoność oczekiwana
  - Ilość zasobów potrzebna przy „typowych” danych wejściowych rozmiaru n.

I - 7.c) Funkcja złożoności obliczeniowej

- Przyporządkowuje każdej wartości „rozmiaru problemu” maksymalną liczbę kroków elementarnych potrzebnych do rozwiązania konkretnego problemu o tym rozmiarze.

I - 7.d) Rząd złożoności obliczeniowej

- Przykłady:
  - $O(logN)$ - logarytmiczna
  - $O(N)$ - liniowa
  - $O(N * logN)$ - liniowo-logarytmiczna
  - $O(N^2)$ - kwadratowa / wielomianowa
  - $O(2^N)$ - wykładnicza
- Problem stałej:
  - Algorytm wykonujący $10000 * N$ operacji ma złożoność liniową $O(N)$, algorytm wykonujący N^2 operacji ma złożoność kwadratową $O(N^2)$, ale dla N mniejszych niż 100 algorytm kwadratowy będzie bardziej optymalny.

I - 7.e) Stos

(i) Wrzucasz kartki na stos(1-2-3-4-5-6), a zdejmujesz tylko z góry (6-5-4-3-2-1)

- Struktura LIFO (Last In First Out).

I - 7.f) Kolejka

(i) Ustawiasz ludzi jeden za drugim(1-2-3-4), a zdejmujesz pierwszego(1-2-3-4)

- Struktura FIFO (First In First Out).

I - 8) Wyrażenia (postać wyrażenia)

I - 8.a) Postać wyrostkowa, infiksowa

- $(x + y) - (z * 3)$

I - 8.b) Postać przedrostkowa, prefiksowa (// notacja Łukasiewicza)

- $-(+(x, y),\ *(z, 3))$
  (i) Działanie przed liczbami, które on dotyczy.

I - 8.c) Postać przyrostkowa, postfiksowa (Reverse Polish Notation RPN)

- $x\ y + z\ 3 * -$ (- to samo, co $x + y - z * 3$)
  (i) Działanie po elementach, których dotyczy. Obliczenia wykonuj w sposób:
  1. Wrzucaj poszczególne elementy na stos.
  2. Jeśli pojawi się znak działania, zdejmij dwie liczby ze stosu i wynik wrzuć na stos.

I - 8.d) **UWAGA** - Zwracaj uwagę na kolejność wykonywania działań!!!

I - 9) Języki programowania ogólnego zastosowania

- języki imperatywne (instrukcyjne)
- języki aplikatywne (funkcyjne)
- języki deklaratywne (logiczne)

I - 10) Architektury:

I - 10.a) von Neumanna

- Cechy:
  - rozkazy i dane są nierozróżnialne i w tej samej pamięci

I - 10.b) Harwardzka

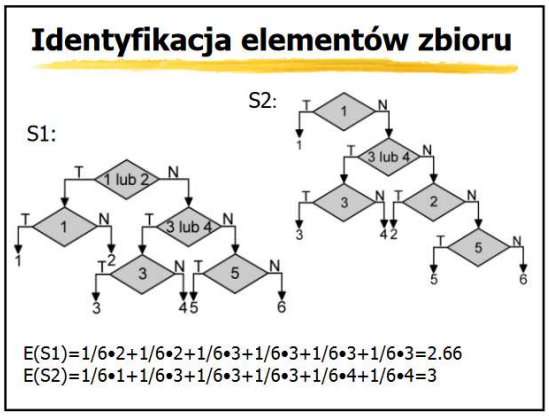
- Cechy:
  - rozkazy i dane są w oddzielnych pamięciach i mogą mieć różne długości

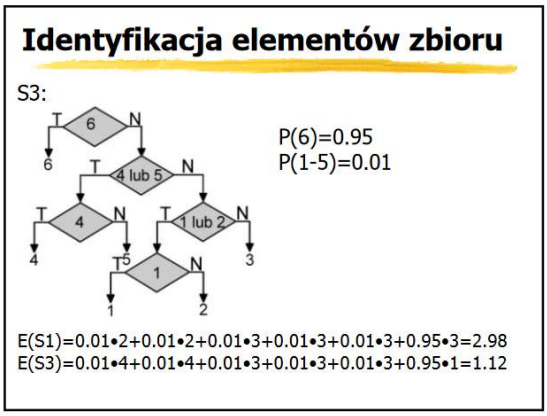
II) Podstawowe informacje

II - 1) Teoria informacji

II - 1.a) Ilość informacji zawarta w danym zbiorze jest miarą trudności rozpoznania elementów tego zbioru.

- $H(X) = \sum_{i=1}^{N} |p_i * log_2(p_i)|$ - jest **ilością informacji na znak**! (w zadaniach pomnóż przez ilość znaków w ciągu**!!!**)
  (i) gdzie p_i jest prawdopodobieństwem wystąpienia elementu x_i.





-

II - 2) Kompresja danych

II - 2.a) Polega na zapisaniu jak najwięcej danych, używając jak najmniej zasobów.

- Kodowanie proste
  - Zapis każdej „litery” jako permutacja długości n zer i jedynek.

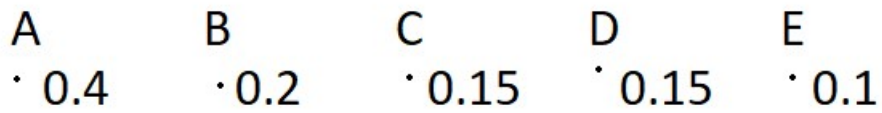
- **<u>Kodowanie Huffmana</u>**
  - Zapis „liter” częściej powtarzających się, jako krótszej długości ciągi 0 i 1.

2) Kodowanie Huffmana

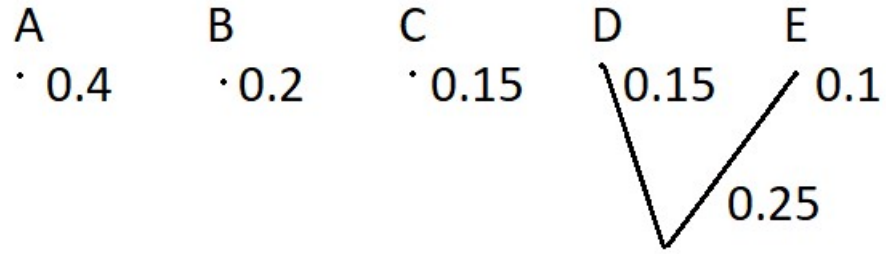
A - 0	0.6
B - 10	0.2
C - 110	0.1
D - 111	0.1

(i) Tworzenie kodu:

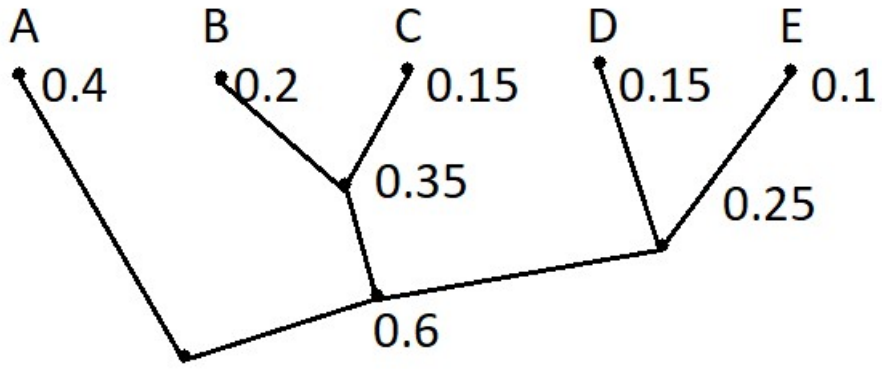
1. Oblicz prawdopodobieństwo wystąpienia danego znaku w ciągu (ilość/wszystkie). Ustaw poszczególne znaki malejąco/rosnąco.



2. Twórz drzewo binarne, grupując zawsze dwa wierzchołki o najmniejszych prawdopodobieństwach (wierzchołek „połączony” ma prawdopodobieństwo równe sumie elementów).



3. Powtarzaj, aż wszystkie elementy będą połączone w drzewo.



4. A teraz zapisujemy każdy element, jako ciąg 0/1, gdzie 0 i 1 reprezentują poszczególne „gałęzie” drzewa (numeruj gałęzie).

```
A - 0
B - 100
C - 101
D - 110
E - 111
```

II - 3) <u>Opis składni (syntaktyki) języka</u>

- Notacja **EBNF**
- Diagramy syntaktyczne (składniowe)

II - 3.b) Notacja EBNF

- Elementy notacji EBNF:
  - symbole pomocnicze
  - symbole końcowe
  - produkcje

(i) **metasymbole**:

- < > symbol pomocniczy
- ::= symbol produkcji
- | symbol alternatywy
- [ ] wystąpienie 1/0 razy
- { } wystąpienie 0/więcej razy

Przykład

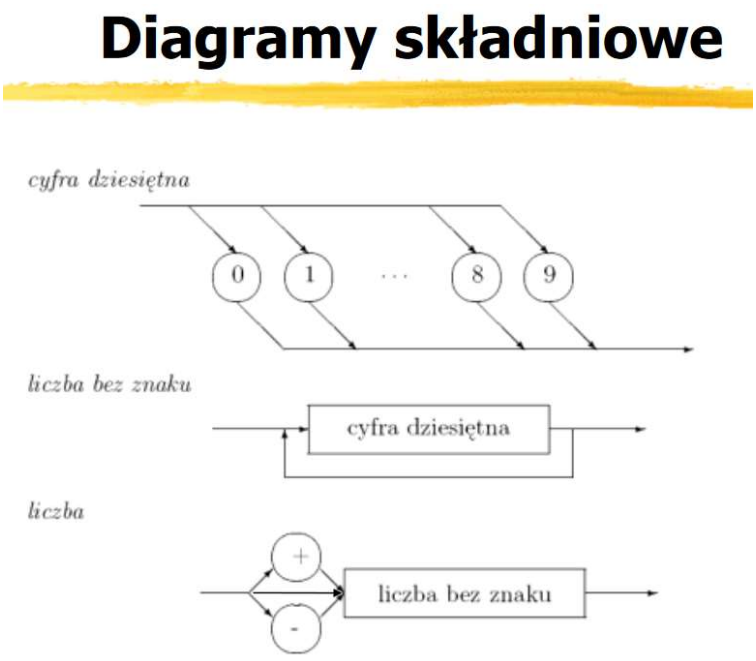
(i)

<cyfra dziesiętna> ::= 0|1|2|3|4|5|6|7|8|9

<liczba bez znaku> ::= <cyfra dziesiętna> {<cyfra dziesiętna>}

<liczba> ::= [+|-] <liczba bez znaku>

II - 3.c) Diagramy syntaktyczne (składniowe)



II - 4) Kod ASCII

- 8-bitowy system kodowania znaków.
  - Większość współczesnych systemów kodowania jest jego rozszerzeniem.

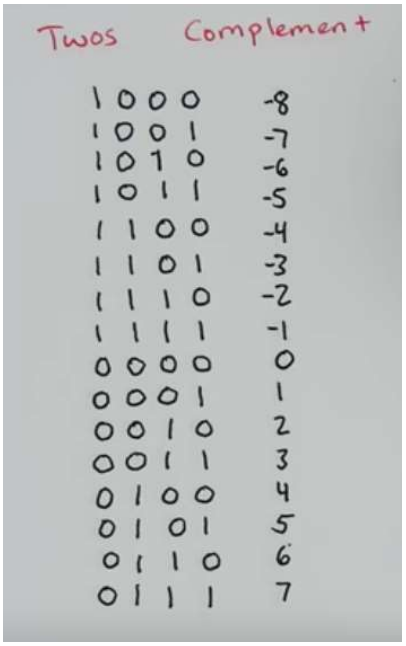
Kod ASCII

	0	1	2	3	4	5	6	7
0	NUL	DLE	space	0	@	P	`	p
1	SOH	DC1 XON	!	1	A	Q	a	q
2	STX	DC2	"	2	B	R	b	r
3	ETX	DC3 XOFF	#	3	C	S	c	s
4	EOT	DC4	$	4	D	T	d	t
5	ENQ	NAK	%	5	E	U	e	u
6	ACK	SYN	&	6	F	V	f	v
7	BEL	ETB	'	7	G	W	g	w
8	BS	CAN	(	8	H	X	h	x
9	HT	EM	)	9	I	Y	i	y
A	LF	SUB	*	:	J	Z	j	z
B	VT	ESC	+	;	K	[	k	{
C	FF	FS	,	<	L	\	l	|
D	CR	GS	-	=	M	]	m	}
E	SO	RS	.	>	N	^	n	~
F	SI	US	/	?	O	_	o	del

II - 5) Kod Unicode

- Znaki ASCII zapisywane za pomocą jednego bajta, a inne znaki na 2/3/4 kolejnych bajtach.
- Wspólny dla całego świata.

II - 6) Kod uzupełnień do dwóch **U2**

- 

II - 6.b) Bardzo dobry sposób na **zrozumienie systemu U2** ma się następująco:

- Pierwszy element jest „odwrócony” (tj. potęga dwójki o najwyższych współczynniku pomnożona jest przez -1).

Liczby 6-bitowe w U2

-32	16	8	4	2	1	
1	1	1	1	1	1	= -1
1	0	0	0	0	0	= -32
1	0	1	0	1	0	= -22

(i) **Nadmiar** (analogicznie niedomiar) można wykryć, gdy liczba przekroczy maksymalną wartość i stanie się ujemna (również dla liczb zmiennoprzecinkowych!!!).

II - 6.c) Dodawanie i odejmowanie działają!!!

- Zamiana z dodatniej na ujemną to:
  1. Odwróć znaki.
  2. Dodaj 1.

II - 7) Liczby zmiennopozycyjne

II - 7.a) $liczba = mantysa * 2^{cecha}$

- Przykład: System binarny
  - znak [+/-] 1 bit
  - mantysa zakres [0.5, 1) 23 bity
  - cecha w kodzie U2, zakres [-128, 127] 8 bitów

III) Sortowanie

III - 1) Quicksort

III - 1.a) Sposób postępowania:

- Wybierasz element dzielący (może być w połowie).
- Względem niego dzielisz tablicę na elementy mniejsze i większe od niego.
- Zaczynasz sortowanie idąc od pierwszego i ostatniego elementu. Gdy znajdziesz z lewej element większy od dzielącego, a z prawej mniejszy to zamieniasz te dwa elementy miejscami. Wykonujesz do momentu, gdy znacznik lewy znajdzie się na prawym.
- Wykonujesz „quicksort” na poszczególnych częściach tablicy (przedział [0, znacznik] i [znacznik, N-1]).

III - 2) Insertion Sort

III - 2.a) Sposób postępowania:

- Jak znajdziesz nieposortowany element idąc od lewej to:
- „Wyciągnij go” i przesuń odpowiednią ilość liczb z lewej o 1 miejsce w prawo
  (i) Aby zrobić miejsce dla tego „nieposortowanego” elementu.
- Przesuń znacznik o jedno miejsce w prawo.

III - 3) Selection Sort

III - 3.a) Sposób postępowania:

- Posiadasz znacznik „już posortowanych” elementów tablicy (na początku 0).
- Szukasz najmniejszej z wartości w „nieposortowanej” tablicy.
- Zamieniasz go z pierwszym nieposortowanym elementem.
- Przesuwasz znacznik „posortowania” o jeden w prawo.

III - 4) Bubble Sort

III - 4.a) Sposób postępowania:

- Sortowanie poprzez porównanie sąsiadujących ze sobą elementów.
- Porównywanie wykonujemy do ostatniego elementu, jednak przy każdej następnej pętli opuszczamy 1 więcej element z końca (on już na pewno jest dobrze ustawiony).

III - 5) Combsort

III - 5.a) Ulepszony BubbleSort. Jedyną różnicą jest zmieniająca się „odległość sprawdzanych komórek”.

- Zaczynasz od odległości 10, a w każdym następnym wykonaniu dzielisz przez 1.3.

IV) Rady przed egzaminem teorytycznym!!!

- Na pytanie: „Co zwróci funkcja” staraj się zrozumieć program i odpowiedzieć „co ma na celu ten program”. Nie przedstawiaj algorytmu w formie ustnej!!!
- W obliczaniu ilości informacji w ciągu, przemnóż wzór Shannona przez ilość znaków!!! (wzór zwraca średnią na znak)
- Na zadania „co zwróci funkcja”, odpowiadaj tak, jakby było zadane „co ma na celu zwrócić funkcja” (sens logiczny, a nie algorytm słowny)!
- Przy zamianie na RPN, czy postać prefiksową uważaj na kolejność wykonywania działań!!! (mnożenie i dzielenie przed dodawaniem [Gajęcki rzadko daje w nawiasach])
- Różnice w sortowaniach podawaj np.:
  - Jeden zakończy pracę szybciej przy „dobrych” danych.
  - Ten stabilny, tamten nie.
  - Ilość odwołań do pamięci, zamiany wartości w tablicach.
- Nadmiar (analogicznie niedomiar) występuje, gdy wartość liczby przekroczy swoją maksymalną wartość.
  - A w U2 zamieni się w minimalną liczbę.
- W zadaniach typu „ile bitów potrzeba, aby zapisać liczby z zakresu” z dokładnością po przecinku, pamiętaj o tym, że jeśli nie jest podane kodowanie w U2, to sam możesz deklarować zakres liczb!!!
  - Np.

```
Ile bitów potrzeba do zapisania liczb z zakresu -20 do 40 z dokładnością do 2 miejsc po przecinku?

Przesuwam zakres / Liczby koduję od -21 (jako same zera)

Liczb całkowitych jest 60, a więc potrzeba 6 bitów na część całkowitą.

Aby uzyskać dokładność do 2 cyfr po przecinku (w systemie dziesiętnym),
muszę mieć większą lub równą dokładność w dwójkowym, tj.
1/2^6   <   1/100   <   1/2^7
A więc potrzeba 7 bitów do zapisania części ułamkowej liczby.

Łącznie potrzeba 13 bitów do zapisania tych liczb.
```

1 Teoria Informacji

1.1 Ilość informacji

Ilością informacji zawartą w zbiorze $X = \{x_1, x_2, \ldots, x_n\}$ nazywamy liczbę:

$$H(X) = -\sum_{i=1}^{N} p_i * \log_2(p_i)$$

Gdzie p_i jest prawdopodobieństwem wystąpienie elementu x_i, oraz $\forall_i : p_i > 0 \wedge \sum_i p_i = 1$

1.2 Kodowanie Huffmana

Im częściej dany symbol występuje w ciągu danych, tym mniej zajmie bitów.
Np. AABACADABA
A - 0 - 0.6
B - 10 - 0.2
C - 110 - 0.1
D - 111 - 0.1
Kody różnej długości, żaden kod nie może być prefiksem drugiego.

2 Podstawowe pojęcia

- Syntaktyka (składnia) - zbiór reguł określający formalnie poprawne konstrukcje językowe
- Semantyka - opisuje znaczenie konstrukcji językowych, które są poprawne składniowo
- Pragmatyka - opisuje wszystkie inne aspekty języka

Instrukcje:
Proste:

- przypisania
- wywołania funkcji
- operacje we/wy

Strukturalne:

- warunkowa
- wyboru
- iteracyjne

3 Notacja BNF i EBNF

3.1 BNF

Notacja BNF jest zestawem reguł produkcji o następującej postaci:
¡symbol¿ ::= ¡wyrażenie zawierające symbole¿
Znaczenie użytych tu symboli jest następujące:

- < – lewy ogranicznik symbolu
- > – prawy ogranicznik symbolu
- ::= – jest zdefiniowane jako
- | – lub

3.2 EBNF

Extended BNF
Symbole EBNF:

- < > - symbol pomocniczy
- ::= - symbol produkcji
- | - symbol alternatywy
- [ ] - wystąpienie 0 lub 1 raz (EBNF)
- { } - powtórzenie 0 lub więcej razy (EBNF)

Różnice względem BNF:

- BNF używa symboli (<, >, |, ::=) dla siebie; gdy pojawią się one w języku który ma być definiowany, notacja BNF nie może być użyta bez modyfikacji i objaśnień.
  W EBNF symbole terminalne są zamknięte pomiędzy znakami cudzysłowu ("..." lub '...'), a symbole w sekwencji rozdzielane przecinkami. Ostre nawiasy ("<...>") dla symboli nieterminalnych mogą zostać pominięte.
- Składnia BNF może reprezentować wyłącznie jedną regułę w jednej linii.
  Każdą regułę kończy znak ograniczający – według normy średnik. Ponadto istnieją mechanizmy ulepszenia definiujące ilość powtórzeń, wyłączanie (np. wszystkie znaki bez cudzysłowów), komentarze.

Pomimo wszystkich ulepszeń EBNF nie jest "silniejszy" w rozumieniu języka który może definiować. Gramatyka zdefiniowana w EBNF może być również reprezentowana w BNF.

Symbole EBNF:

- definicja =
- złączenie ,
- zakończenie ;
- alternatywa |
- zawartość opcjonalna [...]
- powtórzenie { ... }
- grupowanie (...)
- tekst dosłowny " ... "
- tekst dosłowny ' ... '
- komentarz (* ... *)
- wyrażenie specjalne ? ... ?
- wyjątek -
- wielokrotność *

Przykłady:

W notacji BNF:
liczba ze znakiem = znak , liczba | liczba ;
liczba = cyfra | liczba cyfra;

W notacji EBNF:
liczba ze znakiem = [znak ,] liczba ;
liczba = { cyfra } ;

<cyfra dziesiętna> ::= 0|1|2|3|4|5|6|7|8|9
<liczba bez znaku> ::= <cyfra dziesiętna> {<cyfra dziesiętna>}
<liczba> ::= [+|−] <liczba bez znaku>

4 Maksymy Lorda Icona

- Programy mają być czytane przez ludzi.
- Czytelność jest zwykle ważniejsza niż sprawność.
- Najpierw projekt potem kodowanie.
- Dawaj więcej komentarzy niż będzie ci potrzeba.
- Stosuj komentarze wstępne.
- Stosuj przewodniki w długich programach.
- Komentarz ma dawać coś więcej, niż tylko parafrazę tekstu programu.
- Błędne komentarze są gorsze niż zupełny brak komentarzy.
- Stosuj odstępy dla poprawienia czytelności.
- Używaj dobrych nazw mnemonicznych.
- Wystarczy jedna instrukcja w wierszu.
- Porządkuj listy według alfabetu.
- Nawiasy kosztują mniej niż błędy.
- Stosuj wcięcia dla uwidocznienia struktury programu i danych.

5 Sortowanie proste

- Wstawianie $O(n^2)$
  Przenosimy element ciągu źródłowego do ciągu wynikowego, wstawiając go w odpowiednim miejscu.
- Wybieranie $O(n^2)$
  Wybieramy element najmniejszy z ciągu źródłowego i wymieniamy go z pierwszym elementem tegoż ciągu.
- Bąbelkowe $O(n^2)$ Porównujemy sąsiednie pary elementów i zamieniamy aż do skutku.

5.1 Złożoność obliczeniowa

Mówimy, że algorytm jest klasy $O(n^2)$, gdy czas jego wykonania jest proporcjonalny do n^2.

6 Reprezentacja liczb w pamięci komputera

6.1 Reprezentacja liczb stałopozycyjnych

Przez liczbę stałopozycyjną rozumiemy taką, gdzie znak przecinka nie zmienia swojej pozycji.

6.1.1 Kod uzupełnień U2

Pierwszy bit określa znak.
Aby otrzymać liczbę przeciwną negujemy wszystkie bity i dodajemy jeden. Przykład:

$$00101_2 = 5_{10} \rightarrow 11010_2 + 1_2 = 11011_2 = -5_{10}$$

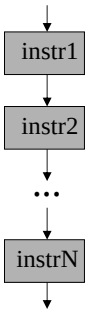
$$00000_2 = 0_{10} \rightarrow 11111_2 + 1_2 = 1|00000_2 = 0_{10}$$

Dlatego 0 ma jedną reprezentację, a zakres nie jest symetryczny.
Zakres na pięciu bitach to:

$$01111_2 = 15_{10} \ldots 10000_2 = -16_1 0$$

Ciąg instrukcji

```
{
  <instrukcja1>;
  <instrukcja2>;
  ...
  <instrukcjaN>;
}
```

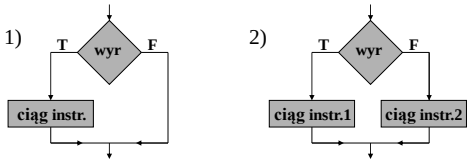


14

Instrukcja skoku warunkowego

1) **if** (<*wyrażenie*>) **{** <*ciąg instrukcji*> **}**

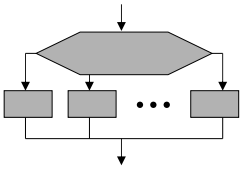
2) **if** (<*wyrażenie*>) **{** <*ciąg instrukcji 1*> **}**
else { <*ciąg instrukcji 2*> **}**



15

Instrukcja wyboru

switch (<*wyrażenie*>) **{**
case <etykieta1> : <instr1>; break;
case <etykieta2> : <instr2>; break;
...
default <*instr*>;
}



Przykład:

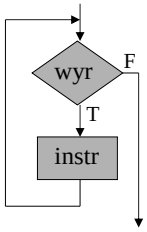
```
switch (dzien_tygodnia) {
  case 0 : printf("niedziela");  break;
  case 1 : printf("poniedziałek"); break;
  ...
  case 6 : printf("sobota"); break;
  default : printf("zla wartosc");
}
```

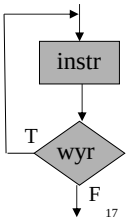
16

Instrukcje pętli

while (<*wyrażenie*>) { <*ciąg instrukcji*> }

do { <*ciąg instrukcji*> } ***while*** (<*wyrażenie*>);



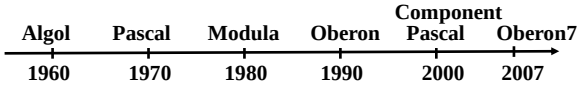


17

Konstrukcje strukturalne

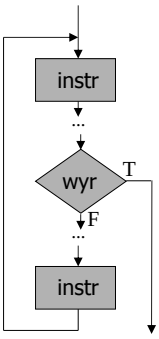
begin ... end
if ... then ...
if ... then ... else ...
case ... of ...
while ... do ...
repeat ... until ...
for ... to ... do ...

Niklaus Wirth



Instrukcja break

```
while (True) {
  ...
  ...
  if (<wyrażenie>) break;
  ...
  ...
}
```



19

Liczby całkowite

Turbo Pascal

typ	zakres	rozmiar
shortint	-128..127	1
integer	-32768..32767	2
longint	-2147483648..214748647	4
byte	0..255	1
word	0..65535	2

Java

typ	zakres	rozmiar
byte	-128..127	1
short	-32768..32767	2
int	-2147483648..214748647	4
long	-2^63..2^63-1	8

Liczby całkowite

FPC

Typ	Zakres	Rozmiar
Byte	0 .. 255	1
Shortint	-128 .. 127	1
Smallint	-32768 .. 32767	2
Word	0 .. 65535	2
Integer	either smallint or longint	size 2 or 4
Cardinal	longword	4
Longint	-2147483648 .. 2147483647	4
Longword	0 .. 4294967295	4
Int64	-9223372036854775808 .. 9223372036854775807	8
QWord	0 .. 18446744073709551615	8

Liczby całkowite

C/C++
- short int, int, long int, long long int
- signed, unsigned

Standard C99:
- int — ma minimum 16 bitów
- long int — jest co najmniej takiego rozmiaru, co int
- long long int — ma minimum 64 bity

W praktyce:
- short int — ma 16 bitów
- int , long int — ma 32 bity
- long long int — ma 64 bity

Liczby zmiennopozycyjne

Cel: Oddzielenie zakresu od dokładności

Sposób zapisu:

- w matematyce **$l = m * 10^c$**
- w komputerze **$l = m * 2^c$**

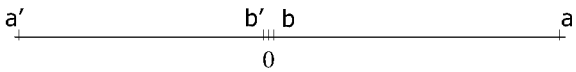
l - ***liczba***
m - ***mantysa***
c - ***cecha***

Liczby zmiennopozycyjne

Przykład 1. System dziesiętny

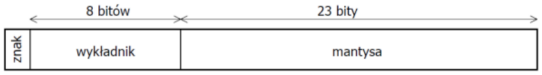
mantysa - 3 cyfry + znak (0.000 - 0.999)
cecha - 2 cyfry + znak (-99 - 99)

dodatnia wartość maksymalna (a): $0.999*10^{99}$
ujemna wartość minimalna (a'): $-0.999*10^{99}$
dodatnia wartość minimalna (b): $0.001*10^{-99}$
ujemna wartość maksymalna (b'): $-0.001*10^{-99}$



Liczby zmiennopozycyjne

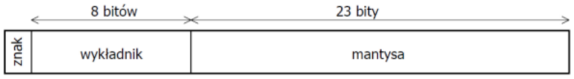
Przykład System binarny (typ 4 bajtowy)



- Znak mantysy: 0 lub 1
- Wykładnik w kodzie U2 [-128..127]
- Mantysa 0 lub [0.5..1)
- Liczba to mantysa * 2 cecha
- Liczba znormalizowana : najbardziej znacząca cyfra mantysy równa 1
- Zakres liczb: od $-(1-2^{-23})\cdot 2^{127}$ do $(1-2^{-23})\cdot 2^{127}$
- Minimalna dodatnia: $1/2\cdot 2^{-128}$

Liczby zmiennopozycyjne

Przykład: 4 bajtowe liczby typu float



Znak mantysy: 0 lub 1
Wykładnik (cecha) przesunięty o 127
Mantysa [1-2)
Liczba to mantysa * 2 cecha
Problem reprezentacji 0

Liczby zmiennopozycyjne

Przykład: 4 bajtowe liczby typu float

- liczby dodatnie: od 2^{-127} do $(2-2^{-23})\cdot 2^{128}$
- liczby ujemne: od $-(2-2^{-23})\cdot 2^{128}$ do -2^{-127}
- przepełnienie (nadmiar): Liczby$>(2-2^{-23})\cdot 2^{128}$

Liczby zmiennopozycyjne

```
union fi {
  float a;
  unsigned int b;
};

int main() {
  fi x;
  int t[100];

  while (true) {
    cin >> x.a;

    for (int i=31; i>=0; i--) { t[i]=x.b%2; x.b=x.b/2; }
    cout << t[0] << " ";
    for (int i=1; i<=8; i++) cout << t[i];
    cout << " ";
    for (int i=9; i<=31; i++) cout << t[i];
    cout << endl;
  }
}
```

Liczby zmiennopozycyjne

Turbo Pascal

typ	zakres	dokładność	rozmiar
real	2.9E-39 .. 1.7E38	11-12	6
single	1.5E-45 .. 3.4E38	7-8	4
double	5.0E-324 .. 1.7E308	15-16	8
extended	3.4E-4932 .. 1.1E4932	19-20	10

Java, C/C++

typ	zakres	dokładność	rozmiar
float	1.5E-45 .. 3.4E38	7-8	4
double	5.0E-324 .. 1.7E308	15-16	8

Standard ANSI IEEE 754

Formaty stałopozycyjne dwójkowe:
- 16-bitowy (SHORT INTEGER)
- 32-bitowy (INTEGER)
- 64-bitowy (EXTENDED INTEGER)

Formaty zmiennopozycyjne:
- pojedynczej precyzji (SINGLE)
  m=23+1, c=8
- podwójnej precyzji (DOUBLE)
  m=52+1, c=11

1/2

Dokładność obliczeń

Obliczenia iteracyjne

```
var
   p,q,r : T;
   i:integer;
begin
   r := 3.0;
   p := 0.01;
   for  i:=1 to 50 do
   begin
     q := p+r*p*(1-p);
     writeln(q);
     p := q;
   end;
end.
```