Algorytmy

na maturę

Opracowała Małgorzata Piekarska

VILO Bydgoszcz

Rozkład liczby na cyfry

<u>Dane</u>:

k – liczba naturalna

<u>Wynik</u>:

A[0..n-1] - tablica kolejnych cyfr liczby k, n – ilość cyfr

gdzie A[0] to cyfra jedności

<u>Rozwiązanie:</u>

```
n←0
dopóki k>0
    A[n] ← k mod 10
    k ← k div 10
    n ← n + 1
```

Rozwinięcie binarne liczby naturalnej

<u>Dane</u>:

k – liczba naturalna zapisana w systemie dziesiętnym

<u>Wynik</u>:

n – długość rozwinięcia binarnego liczby k
tablica B[0..n-1] zawierająca kolejne bity rozwinięcia binarnego liczby k,
przy czym B[0] – bit najmniej znaczący B[n-1] – bit najbardziej znaczący

```
n←0
dopóki k≥0 wykonuj
    B[n] ← k mod 2
    k ← k div 2
    n ← n + 1
```

Zamiana liczby binarnej na dziesiętną

Dane:

n – długość rozwinięcia binarnego liczby k

tablica B[0..n-1] zawierająca kolejne bity rozwinięcia binarnego liczby k,
przy czym B[0] – bit najmniej znaczący B[n-1] – bit najbardziej znaczący

Wynik:

k – wartość w systemie dziesiętnym liczby, której bity zapisano w tablicy B

```
p←1, k←0
dla i=0,1,..n-1 wykonuj
    k ← k + p*B[i]
    p ← p * 2
```

Czy liczba jest pierwsza?

Dane:

n – liczba naturalna

Wynik:

PRAWDA – jeśli n jest liczbą pierwszą, FAŁSZ – jeśli n nie jest liczbą pierwszą

```
funkcja pierwsza(n)
    d←2
    dopóki d*d ≤ n wykonuj
        jeśli n mod d = 0
            zwróć FAŁSZ i zakończ
        d ← d + 1
    jeśli n < 2
        zwróć FAŁSZ i zakończ
    w przeciwnym wypadku
        zwróć PRAWDA i zakończ
```

Czy liczba jest doskonała?

Dane:

n – liczba naturalna

Wynik:

PRAWDA – jeśli n jest liczbą doskonałą, FAŁSZ – jeśli n nie jest liczbą doskonałą

```
funkcja doskonala(n)
    d←2, s←1
    dopóki d*d ≤ n wykonuj
        jeśli n mod d = 0
            s ← s + d
            jeśli d <> (n div d)
                s ← s + (n div d)
        d ← d + 1
    jeśli n=s
        zwróć PRAWDA i zakończ
    w przeciwnym wypadku
        zwróć FAŁSZ i zakończ
```

Rozkład na czynniki pierwsze

<u>Dane</u>:

n – liczba naturalna

<u>Wynik</u>:

Wszystkie dzielniki liczby, które są liczbami pierwszymi, od najmniejszego do największego

```
d←2
dopóki n > 1 wykonuj
    jeśli n mod d = 0
        wypisz d
        n = n div d
    w przeciwnym wypadku
        d ← d + 1
```

Algorytm Euklidesa - iteracyjnie

Dane:

a,b – para liczb naturalnych

Wynik:

Największy wspólny dzielnik liczb a,b

```
Funkcja nwd(a,b)
    dopóki a > 0 oraz b > 0 wykonuj
        jeśli a>b
            a ← a mod b
        w przeciwnym wypadku
            b ← b mod a
    jeśli a>0
        zwróć b i zakończ
    w przeciwnym wypadku
        zwróć a i zakończ
```

Algorytm Euklidesa - rekurencyjnie

<u>Dane</u>:

a,b – para liczb naturalnych

<u>Wynik</u>:

Największy wspólny dzielnik liczb a,b

```
Funkcja nwd(a,b)
    jeśli b=0
        zwróć a i zakończ
    w przeciwnym wypadku
        zwróć nwd(b,a mod b)
        i zakończ
```

N-ty wyraz ciągu Fibonacciego iteracyjnie

<u>Dane</u>:

n –liczba naturalna

<u>Wynik</u>:

n-ty wyraz ciągu Fibonacciego

```
Funkcja fib(n)
    jeśli n=0 lub n=1
        zwróć 1 i zakończ
    w przeciwnym wypadku
        a←1, b←1
        dla i=2,3,..,n wykonuj
            c ← a + b
            a ← b
            b ← c
        zwróć c i zakończ
```

N-ty wyraz ciągu Fibonacciego rekurencyjnie

<u>Dane</u>: *n –liczba naturalna*

<u>Wynik</u>: *n-ty wyraz ciągu Fibonacciego*

Rozwiązanie rekurencyjne

```
Funkcja fib(n)
    jeśli n=0 lub n=1
        zwróć 1 i zakończ
    w przeciwnym wypadku
        zwróć fib(n-1)+fib(n-2)
        i zakończ
```

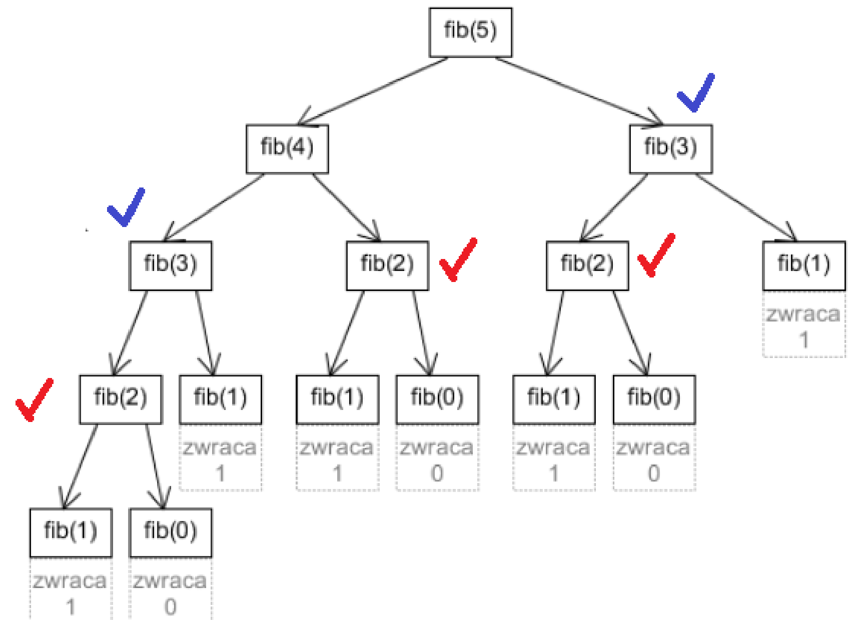
Sposób II: Programowanie dynamiczne

Nie musisz wyliczać wielokrotnie tych samych wartości ciągu. Zapamiętuj je w tablicy i wykorzystuj ponownie.

```
f[1]=1
f[2]=1
Dla każdego i=3,4,..,n wykonuj
        f[i] = f[i-1] + f[i-2]
```

Niestety

wywołania rekurencyjne powodują, iż komputer wielokrotnie oblicza te same liczby Fibonacciego



Strategia zachłanna wydawania reszty

Dane:

W – kwota do wydania,
n – liczba dostępnych nominałów,
tablica T[1..n] przechowująca posortowane wartości nominałów
T[1] – nominał najwyższy

Wynik:

K[1..n] – tablica ilości kolejnych nominałów, jakie należy użyć aby wydać kwotę W ***używając najmniejszej możliwej ilości monet***

Rozwiązanie:

```
i←1
dopóki W>0 wykonuj
    K[i] ← W div T[i]
    W ← W mod T[i]
    i ← i + 1
```

Strategia zachłanna - w każdym kroku wybieramy rozwiązanie najlepsze na ten moment, nie analizując stanów poprzednich.
Zaletą podejścia zachłannego jest szybkość i prostotc implementacji

Algorytm zachłanny jest poprawny dla tego problemu w większości stosowanych obecnie systemów monetarnych jednak nie działa np. w bankomatach, które nie używają banknotów 10zł

Znajdź najmniejszy

<u>Dane</u>:

tablica wartości T[1..n]
n – ilość wartości

<u>Wynik</u>:

min – wartość najmniejsza w tablicy T

```
min ← T[1]
dla i=2,3,..,n wykonuj
    jeśli T[i] < min
        min ←  T[i]
```

Znajdź najmniejszy i największy

<u>Dane</u>:

tablica wartości T[1..n]
n – ilość wartości

<u>Wynik</u>:

min – wartość najmniejsza w tablicy T, max – wartość największa

```
min ← T[1], max ← T[1]
dla i=2,3,..,n wykonuj
    jeśli T[i] < min
        min ←  T[i]
    jeśli T[i] > max
        max ←  T[i]
```

Znajdź najmniejszy i największy – algorytm optymalny

Dane:

tablica wartości T[1..n] złożona z co najmniej dwóch wartości
n – ilość wartości (n jest parzyste, jeśli wcześniej nie było to ostatni element został powielony)

Wynik:

min – wartość najmniejsza w tablicy T, max – wartość największa

```
jeśli T[1]<T[2]: min ← T[1], max ← T[2]
w przeciwnym razie: min ← T[2], max ← T[1]
    dla i=3,5,..,n-1 wykonuj
        jeśli T[i] < T[i+1]
            jeśli T[i]<min: min ←  T[i]
            jeśli T[i+1]>max: max ←  T[i+1]
        w przeciwnym razie
            jeśli T[i]>max: max ←  T[i]
            jeśli T[i+1]<min: min ←  T[i+1]
```

Wyszukiwanie binarne

<u>Dane</u>:

tablica wartości A[1..n] – uporządkowana rosnąco
n – długość tablicy, y – poszukiwana wartość

<u>Wynik</u>:

indeks – numer elementu w tablicy który przechowuje wartość y
lub informacja BRAK

```
lewo ← 1, prawo ← n
dopóki lewo<prawo wykonuj
    srodek ← (lewo+prawo) div 2
    jeśli A[srodek]<y
        lewo ← srodek+1
    w przeciwnym wypadku
        prawo ← srodek
jeśli A[lewo]=y
    indeks← lewo
w przeciwnym wypadku
    indeks← BRAK
```

Sortowanie bąbelkowe

Dane:

tablica wartości A[0..n-1], n – długość tablicy

Wynik:

tablica wartości A[0..n-1] – uporządkowana rosnąco

```
przedzial ← n-1
dopóki przedzial≥2
    dla i=1,2,..,przedzial-1
        jeśli A[i] > A[i+1]
            A[i] ↔ A[i+1]
    przedzial ← przedzial-1
```

Sortowanie przez wybieranie

Dane:

tablica wartości A[0..n-1], n – długość tablicy

Wynik:

tablica wartości A[0..n-1] – uporządkowana rosnąco

```
dla k=0,1,..n-2
    min ← k
    dla i=k+1,k+2,..,n-1
        jeśli A[i] < A[min]
            min ← i
    A[k] ↔ A[min]
```

Sortowanie przez wstawianie

<u>Dane</u>:

tablica wartości A[0..n-1], n – długość tablicy

<u>Wynik</u>:

tablica wartości A[0..n-1] – uporządkowana rosnąco

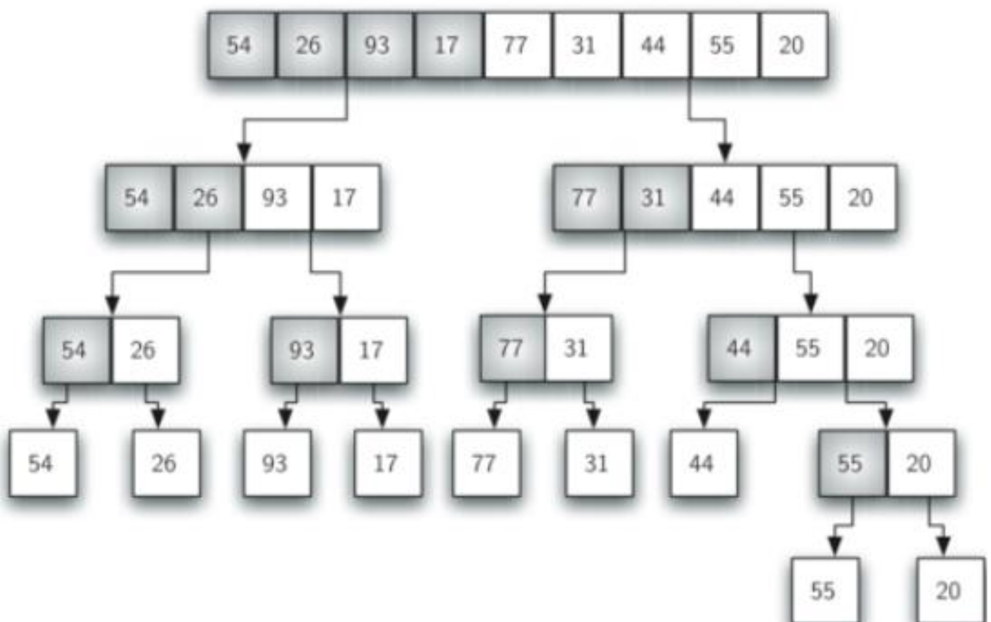
```
dla k=1,2,..,n-1
    dla i=k-1,k-2,..,0
        jeśli A[i+1] < A[i]
            A[i+1] ↔ A[i]
        w przeciwnym wypadku
            break
```

Sortowanie „szybkie”

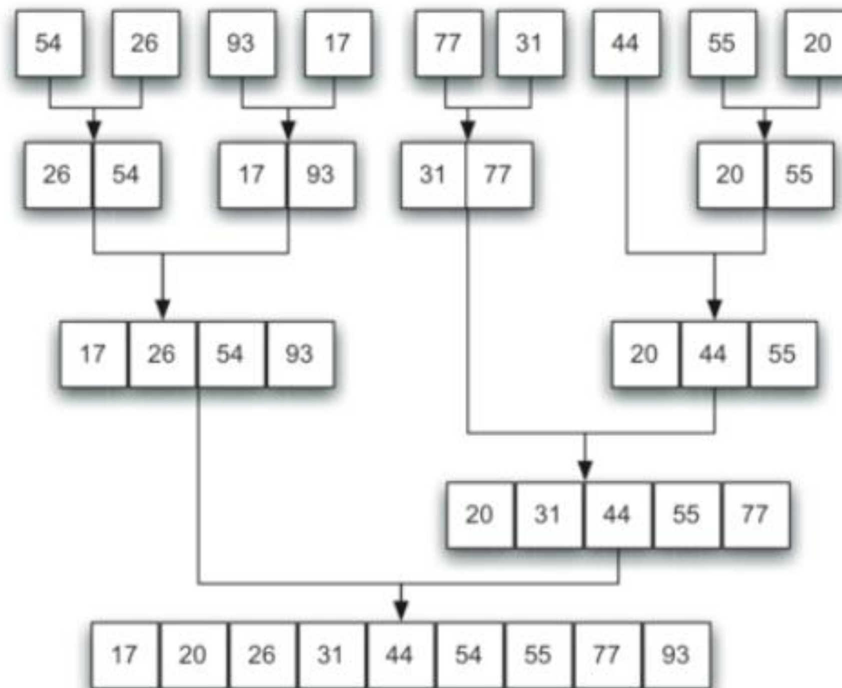
```
void quicksort(int tab[], int left,
int right){
	int i=left;
	int j=right;
	int x=tab[(left+right)>>1];
	do{
		while(tab[i]<x) i++;
		while(tab[j]>x) j--;
		if(i<=j){
			int temp=tab[i];
			tab[i]=tab[j];
			tab[j]=temp;
			i++;
			j--;
		}
	}while(i<=j);
	if(left<j)
quicksort(tab,left,j);
	if(right>i)
quicksort(tab,i,right);
}
```

Sortowanie przez scalanie

ETAP I - dzielimy



ETAP II - scalamy



posortowane

Obliczanie pierwiastka kwadratowego metoda kolejnych przybliżeń

<u>Dane</u>:

P - Liczba rzeczywista, e – dokładność obliczeń

<u>Wynik</u>:

a – wartość pierwiastka kwadratowego z P, wyznaczona bez używania pierwiastkowania

```
a ←1, b ← P
dopóki abs(a-b)>=e
    a ←(a + b)/2
    b ← P/a
```

Schemat Hornera

<u>Dane</u>:

x – punkt, n – stopień wielomianu, W[0..n] – tablica współczynników
W[0] – wyraz wolny, W[n] – współczynnik przy x w najwyższej potędze

<u>Wynik</u>:

y– wartość wielomianu w punkcie X

```
y ←W[n]
dla i=n-1,n-2,..0
    y ← y * x + W[i]
zwróć y
```

$$
\begin{aligned}
w_x(x) &= a_0x^n + a_1x^{n-1} + \cdots + a_{n-1}x + a_n = \\
&= (a_0x^{n-1} + a_1x^{n-2} + \cdots + a_{n-1})x + a_n = \\
&= \left((a_0x^{n-2} + a_1x^{n-3} + \cdots + a_{n-2})x + a_{n-1}\right)x + a_n = \\
&= \cdots = \\
&= \left(\left(\ldots\left((a_0x + a_1)x + a_2\right)x + \cdots + a_{n-2}\right)x + a_{n-1}\right)x + a_n
\end{aligned}
$$

Zamiana binarnej na dziesiętną – schemat Hornera

<u>Dane</u>:

n – długość rozwinięcia binarnego liczby k
tablica B[0..n] zawierająca kolejne bity rozwinięcia binarnego liczby k, przy czym B[0] – bit najmniej znaczący B[n-1] – bit najbardziej znaczący

<u>Wynik</u>:

k – wartość w systemie dziesiętnym liczby, której bity zapisano w tablicy B

```
k←B[n-1]
dla i=n-2,n-3,..0 wykonuj
    k ← k*2 + B[i]
```

Szybkie potęgowanie - LDP schemat Hornera

<u>Dane</u>:

p, B[1..n] – podstawa i rozwinięcie binarne wykładnika (wykładnik jest naturalny)
b[1] – najmniej znaczący bit

<u>Wynik</u>:

y – wartość p^w

```
y ← p
dla i=n-1,n-2,..,1
    jeśli B[i]=0
        y ← y*y
    w przeciwnym wypadku
        y ← y*y*p
```

Szybkie potęgowanie - PDL

Dane:

p, w – podstawa i wykładnik (wykładnik jest naturalny)

Wynik:

y – wartość p^w

```
y ← 1
dopóki w>0
    jeśli w mod 2=1
        y ← y * p
    w ← w div 2
p ← p * p
```

Znajdowanie zera funkcji metoda połowienia przedziału

Dane:

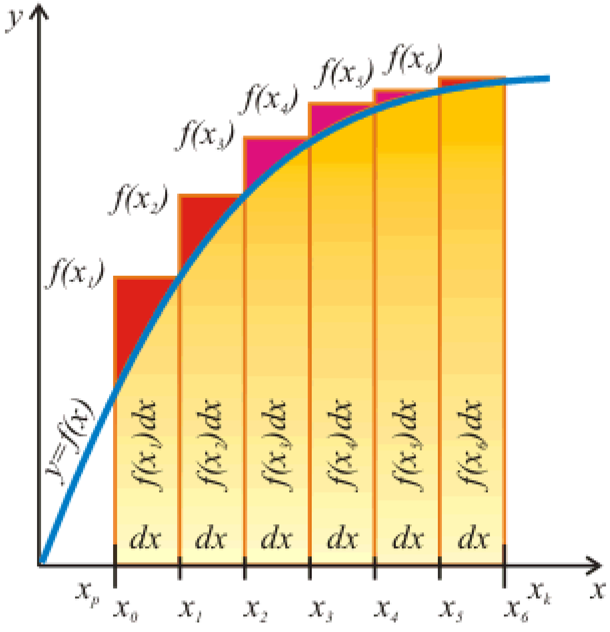
F(x) – funkcja ciągła w przedziale <a,b>
*a,b – krańce przedziału (a≤b), takie że f(a)*f(b)<0*
e – dokładność obliczeń

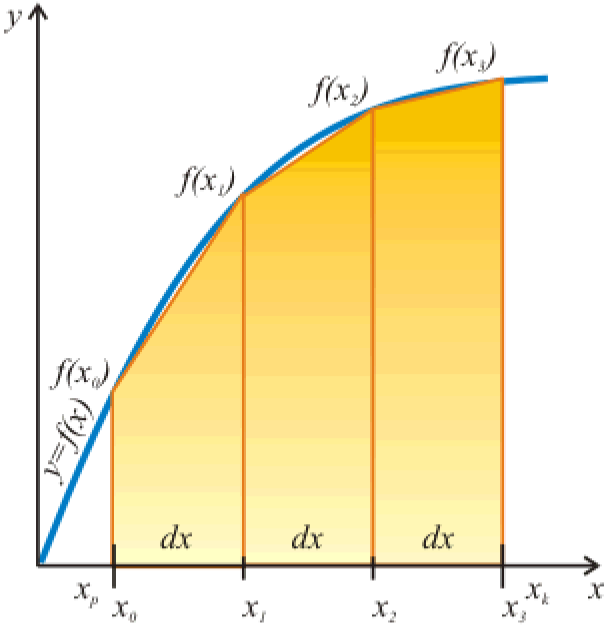
Wynik:

X – punkt na osi OX, w którym funkcja przyjmuje wartość 0

```
dopóki (abs(a-b))>e
    x ← (a + b) / 2
    jeśli abs(f(x))≤e
        break
    jeśli f(x)*f(a)≤0
        b ← x
    w przeciwnym wypadku jeśli f(x)*f(a)≤0
        a ← x
zwróć (a+b)/2
```

Obliczanie pola obszarów zamkniętych wykresem





Czy tekst jest palindromem?

Dane:

S[0..n-1] –tekst o długości n, zapisany w tablicy znaków

Wynik:

PRAWDA – jeśli tekst jest palindromem, FAŁSZ – jeśli nie jest

```
Funkcja palindrom(S[])
    l←0, p←n-1
    dopóki l<p
        jeśli S[l] <> S[p]
            zwróć FAŁSZ i zakończ
        l ← l + 1
        p ← p - 1
    zwróć PRAWDA i zakończ
```

Czy teksty są anagramami?

<u>Dane</u>:

S1[1..n], S2[1..m] –teksty zapisane w tablicach znaków, n i m długości tekstów

<u>Wynik</u>:

PRAWDA – jeśli teksty są anagramami, FAŁSZ – jeśli nie są

```
Funkcja anagram(S1,n,S2,m)
    jeśli n<>m
        zwróć FAŁSZ i zakończ
    posortuj bąbelkowo S1
    posortuj bąbelkowo S2
    jeśli S1==S2
        zwróć PRAWDA i zakończ
    w przeciwnym wypadku
        zwróć FAŁSZ i zakończ
```

Wyszukiwanie wzorca w tekście

Dane:

T[1..n], W[1..m] –tekst i wzorzec zapisane w tablicach znaków, n i m długości tekstów

Wynik:

numer pozycji na której jest pierwsze z lewej wystąpienie tekstu W w tekście T,
BRAK – jeśli W nie występuje w T

```
Funkcja szukaj(T,n,W,m)
    dla i=1,2,..n-m
        jeśli T[i]=W[1]
            licz←1
            dla j=2,3,..,m
                jeśli T[i+j-1]=W[j]
                    licz←licz+1
                w przeciwnym wypadku
                    break
            jeśli licz=m
                zwróć (i) i zakończ
    zwróć BRAK i zakończ
```

Obliczanie wartości wyrażenia zapisanego w Odwrotnej Notacji Polskiej

Notacja tradycyjna	ONP
2+3	2 3 +
$4 \cdot (2+3)$	$4\ 2\ 3 + \cdot$
$(2+3) \cdot 4$	$2\ 3 + 4 \cdot$
$2 \cdot 3 - 4 \cdot 5$	$2\ 3 \cdot 4\ 5 \cdot -$

SPECYFIKACJA

- Dane mamy wyrażenie

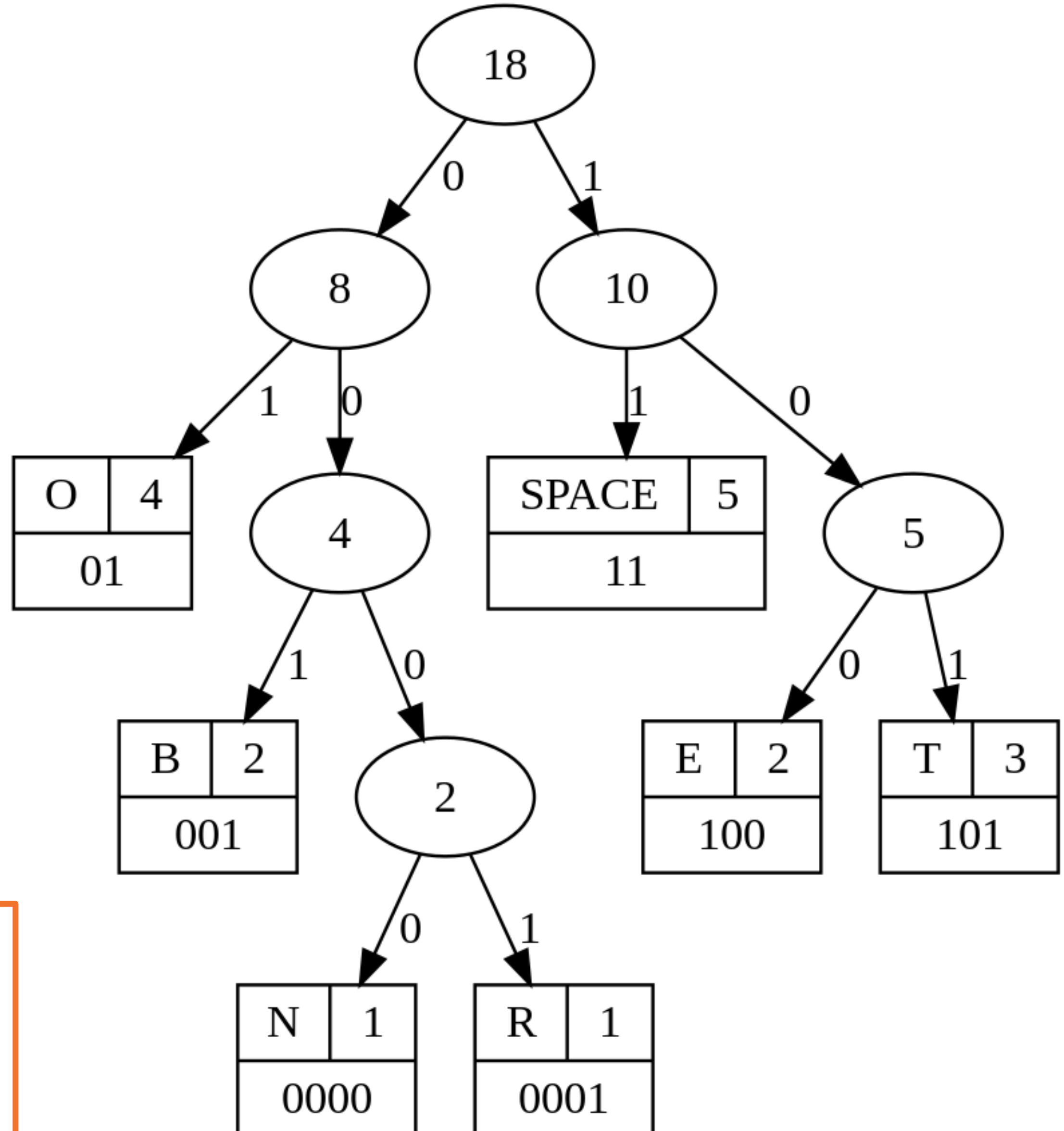
  zapisane w ONP, w postaci ciągu liczb i operatorów działań (+,-,*,/,=}
- W wyniku oczekujemy wartości wyrażenia odczytanego z wejścia

ROZWIĄZANIE

- Z wejścia odczytujemy kolejne elementy wyrażenia.
- Jeśli element jest liczbą, zapisujemy go na stosie.
- Jeśli element jest operatorem, ze stosu zdejmujemy dwie liczby, wykonujemy nad nimi operację określoną przez odczytany operator, wynik operacji umieszczamy z powrotem na stosie.
- Jeśli element jest końcowym znakiem '=', to na wyjście przesyłamy liczbę ze szczytu stosu i kończymy. Inaczej kontynuujemy odczyt i przetwarzanie kolejnych elementów

Kody Huffmanna

Drzewo Huffmana wygenerowane z frazy „TO BE OR NOT TO BE”



Odkoduj:

0011000001100101

Szyfr Cezara

znaki zamieniamy na inne, będące ich przesunięciem w alfabecie o k.

Tekst jawny	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
Szyfr Cezara	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	A	B	C

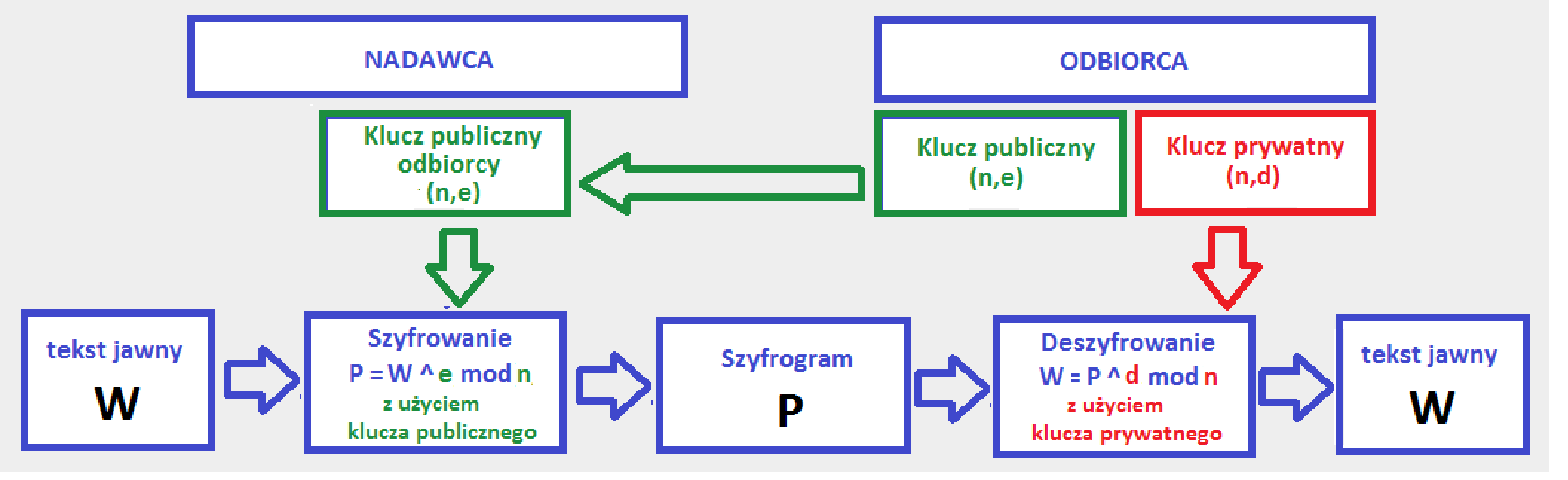
SPECYFIKACJA

- Dany mamy tekst s i klucz k (liczba oznaczająca przesunięcie)
- W wyniku oczekujemy zaszyfrowanego tekstu z wejścia

ROZWIĄZANIE

```
Dla każdego znaku z w tekście s:
    Jeśli z < "A"  lub  z > "Z" to
        następny obieg pętli
    z ← znak (kod(z) + klucz)
    Jeśli z > "Z" to
        z ← z - 26
Pisz s
```

Szyfr RSA


NADAWCA
ODBIORCA
Klucz publiczny odbiorcy (n,e)
Klucz publiczny (n,e)
Klucz prywatny (n,d)
tekst jawny
W
Szyfrowanie
P = W ^ e mod n
z użyciem klucza publicznego
Szyfrogram
P
Deszyfrowanie
W = P ^ d mod n
z użyciem klucza prywatnego
tekst jawny
W

Algorytmy badające własności geometryczne

- sprawdzanie warunku trójkąta,
- badanie położenia punktów względem prostej,
- badanie przynależności punktu do odcinka,
- przecinanie się odcinków,
- przynależność punktu do obszaru,

Konstrukcje rekurencyjne