

# KODY SYMBOLI

---

Materiały KODA, A.Przelaskowski

- Koncepcja drzewa binarnego
    - Metoda S-F
    - Kod Huffmana
    - Adaptacyjne drzewo Huffmana
    - Problemy implementacji
  - Koncepcja przedziałów nieskończonego alfabetu
    - Proste kody przedrostkowe (skrajnie równoważone)
    - Kod Golomba w różnych wersjach
    - Inne kody przedziałowe
  - Podsumowanie
-

# Kod Shannona-Fano

---

- Na wejściu rozkład:  $P_S = \{p_1, \dots, p_n\}$ , czyli **kod entropijny**
- Naturalny pomysł budowy drzewa binarnego od góry
- Dzielimy na podgrupy o zbliżonym prawdopodobieństwie



# Algorytm S-F

---

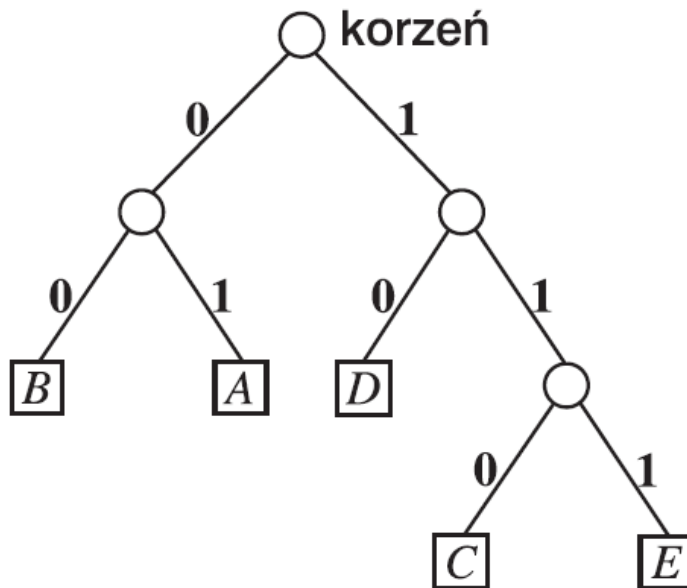
1. Określ wagi symboli
  2. U szereguj symbole w nierosnącym porządku wag (grupa początkowa)
  3. Podziel grupę symboli na dwie części o możliwie równej sumie wag symboli, z zachowaniem porządku listy (tj. jedna podgrupa obejmuje częściej występujące symbole, druga – mniej prawdopodobne).
  4. Przyporządkuj symbolom z jednej podgrupy binarne 0, a symbolom z drugiej – 1.
  5. Rekursywnie powtórz kroki 3 i 4 dla utworzonych przez ostatni podział podgrup o przynajmniej dwóch symbolach
  6. Przypisz kolejnym symbolom z listy słowa kodowe, składające się z bitów kolejno przyporządkowanych grupom, do których trafiał dany symbol w kolejnych podziałach
-

# Przykład S-F

Procedura podziałów:

Symbol	Waga		Podziały	
„B”	12	0	0	
„A”	6	0	1	
„D”	5	1	0	
„C”	4	1	1	0
„E”	4	1	1	1

Efekt:



„A” → 01  
„B” → 00  
„C” → 110  
„D” → 10  
„E” → 111

# Efektywność S-F w przykładzie

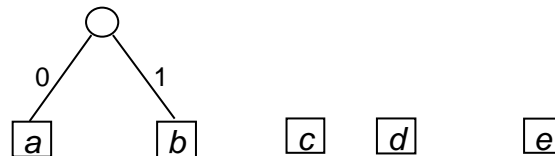
Symbol $a_i$	$P(a_i)$	$I(a_i) = -\log_2 P(a_i)$ [bity/symbol]	$N(a_i) \cdot I(a_i)$ [bity]	$ s_i $ [bity]	$N(a_i) \cdot  s_i $ [bity]
„A”	6/31	2,369	14,215	2	12
„B”	12/31	1,369	16,431	2	24
„C”	4/31	2,954	11,817	3	12
„D”	5/31	2,632	13,161	2	10
„E”	4/31	2,954	11,817	3	12
Suma	–	–	67,441	–	70

$$H(S_{\text{DMS}}) = 2,176 \quad \text{wobec} \quad L_{\text{S-F}} = \sum_i P(a_i) \cdot |s_i| = 2,258$$

S-F stosowana w *WinZip* i produkcie firmy Microsoft: *Cabarc*

# Optymalny Kod Symboli

- Metoda: budowa drzewa binarnego od dołu



- **Optymalne drzewo binarne kodu symboli**

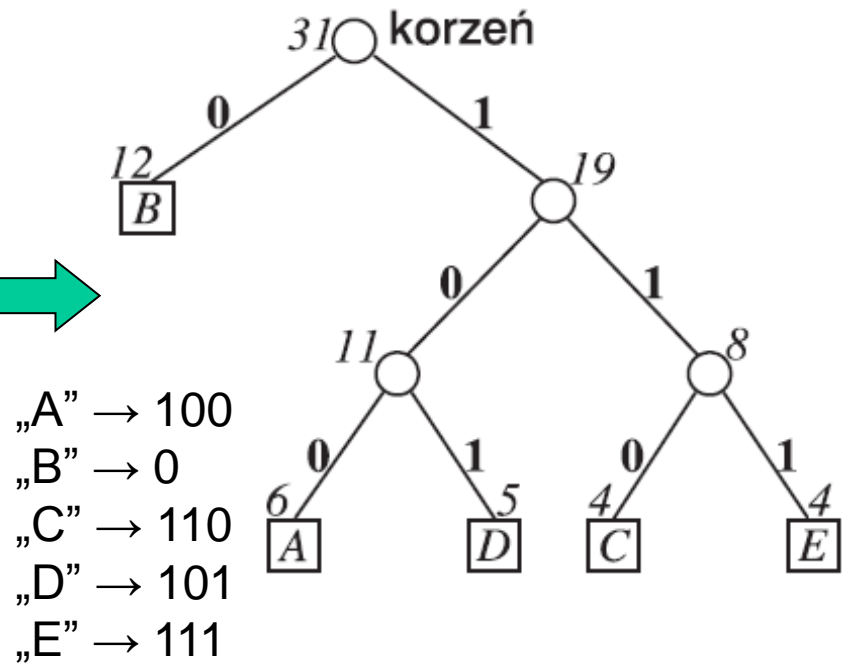
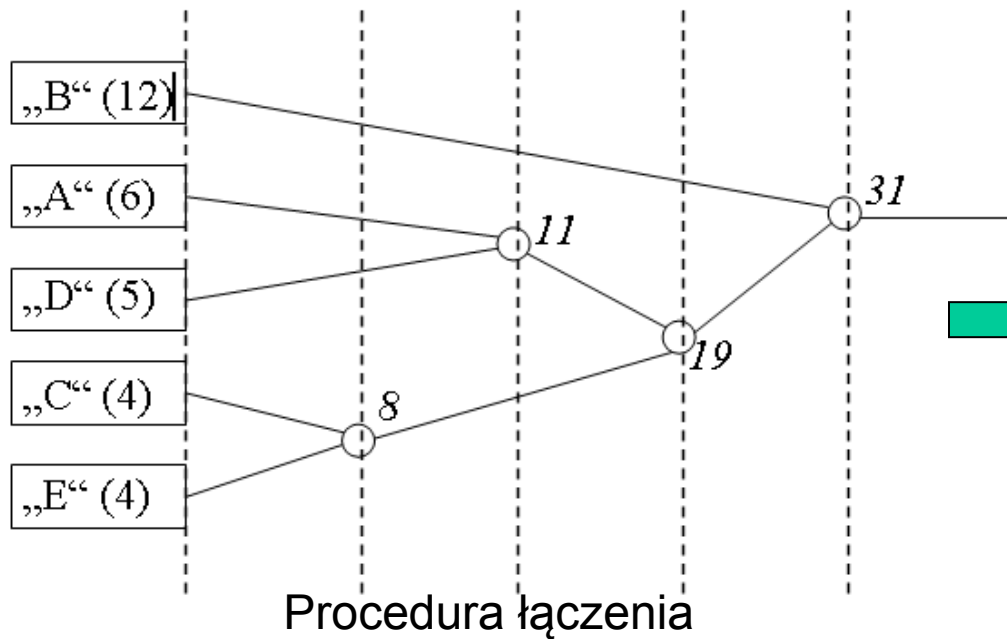
- liść symbolu o najmniejszej wadze ma najdłuższe słowo, czyli leży najgłębiej w drzewie
  - liść symbolu o drugiej w kolejności najmniejszej wadze ma także najdłuższe słowo (lokalnie pełne drzewo binarne)
-

# Algorytm Huffmana (optymalny)

---

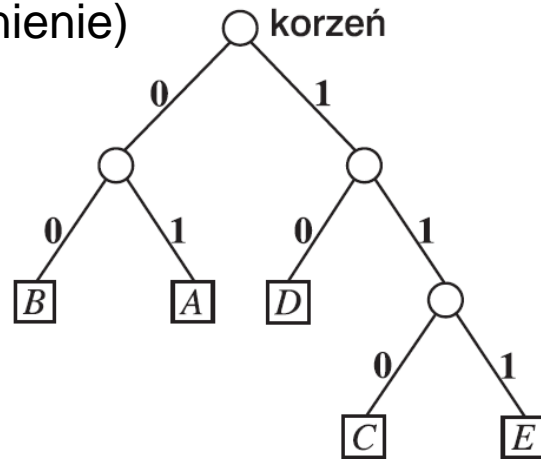
1. Określ wagi symboli; symbole wraz z wagami przypisz liściom – wolnym węzłom
  2. Sortuj listę wierzchołków wolnych w nierosnącym porządku wag
  3. Połącz dwa wolne wierzchołki z najmniejszymi wagami w nowe, większe poddrzewo
    - Przypisz gałęziom prowadzącym od rodzica do węzłów-dzieci różne etykiety: 0 i 1 (np. lewa gałąź – 0, prawa – 1)
  4. Usuń z listy wierzchołków wolnych dwa połączone węzły i wpisz na tę listę nowy wierzchołek rodzica (o wadze równej sumie wag dzieci)
  5. Powtarzaj kroki 3-4 aż do momentu, gdy na liście wierzchołków wolnych pozostanie tylko korzeń drzewa
  6. Odczytaj ze struktury drzewa słowa kodowe kolejnych liści (symboli) - od korzenia do danego liścia
-

# Przykład Huffmana



drzewo S-F (przypomnienie)

„A” → 01  
„B” → 00  
„C” → 110  
„D” → 10  
„E” → 111





# Efektywność kodu Huffmana

Symbol $a_i$	Oszacowane prawdopodobieństwo $P(a_i)$	Długość słowa kodowego $ s_i $ [bity]	Długość sekwencji kodowej [bity]
„A”	6/31	3	18
„B”	12/31	1	12
„C”	4/31	3	12
„D”	5/31	3	15
„E”	4/31	3	12
Suma	–	–	69

efektywność S-F (przypomnienie)

Symbol $a_i$	$P(a_i)$	$I(a_i) = -\log_2 P(a_i)$ [bity/symbol]	$N(a_i) \cdot I(a_i)$ [bity]	$ s_i $ [bity]	$N(a_i) \cdot  s_i $ [bity]
„A”	6/31	2,369	14,215	2	12
„B”	12/31	1,369	16,431	2	24
„C”	4/31	2,954	11,817	3	12
„D”	5/31	2,632	13,161	2	10
„E”	4/31	2,954	11,817	3	12
Suma	–	–	67,441	–	70

$H(S_{DMS}) = 2,176$  wobec  $L_{S-F} = \sum_i P(a_i) \cdot |s_i| = 2,258$  i  $L_{huff} = 2,226$  (-1,5%)

# Wersje kodu Huffmana

---

- Zastosowań jest wiele (JPEG, Deflate, uproszczenia niemal wszędzie)
- Ograniczenia (1 bit/symbol - wpływ długości ciągu kodowanego słowem)

## Usprawnienia:

- Kod co celów transmisji
  - Wykorzystanie heurystyk (charakterystyk a priori)
  - Optymalizacja alfabetu (ograniczone modelowanie):  
łączenie z RLE w JPEG
  - Kod adaptacyjny
-

# Statyczne kodowanie Huffmana (JPEG)

klasyfikacja współczynników DC

Kategoria $k$	Zakres wartości różnicowej dla współczynnika DC	Długość słowa kodowego	Słowo kodowe
0	0	2	00
1	-1,1	3	010
2	-3,-2,2,3	3	011
3	-7,...,-4,4,...,7	3	100
4	-15,...,-8,8,...,15	3	101
5	-31,...,-16,16,...,31	3	110
6	-63,...,-32,32,...,63	4	1110
7	-127,...,-64,64,...,127	5	11110
8	-255,...,-128,128,...,255	6	111110
9	-511,...,-256,256,...,511	7	1111110
10	-1023,...,-512,512,...,1023	8	11111110
11	-2047,...,-1024,1024,...,2047	9	111111110

Przedrostek: słowo kategorii, przyrostek: kod dwójkowy  $k$  najmłodszych bitów wartości  $C$  (lub  $C-1$  dla ujemnych)

**1 → 010 1; -1 → 010 0; 40 → 1110 101000; -40 → 1110 010111**

# RLE plus Huffman

Kategoria(liczba bitów uzupełniającej części słowa kodowego)	Zakres wartości współczynnika AC
1	-1,1
2	-3,-2,2,3
3	-7,...,-4,4,...,7
4	-15,...,-8,8,...,15
5	-31,...,-16,16,...,31
6	-63,...,-32,32,...,63
7	-127,...,-64,64,...,127
8	-255,...,-128,128,...,255
9	-511,...,-256,256,...,511
10	-1023,...,-512,512,...,1023

klasyfikacja współczynników AC

..., niezerowy, 0, -3, 0, 0, 0, 1, -7, ...



1/2, 3/1, 0/3



111001 00, 111010 1, 100 000

Słowo RS (liczba poprzedzających zer/kategoria)	Długość słowa kodowego	Słowo kodowe
0/0 (EOB)	4	1010
0/1	2	00
0/2	2	01
0/3	3	100
...	...	...
0/8	10	1111110110
0/9	16	1111111110000010
0/A	16	1111111110000011
1/1	4	1100
1/2	6	111001
...	...	...
2/A	16	1111111110001110
3/1	6	111010
...	...	...
C/9	16	1111111111100000
C/A	16	1111111111100001
D/1	11	11111111000
...	...	...
F/0 (same zerowe współczynniki)	11	11111111001
F/1	16	1111111111110101
...	...	...
F/9	16	1111111111111101
F/A	16	1111111111111110

# Adaptacyjny kod Huffmana

---

- Zmienna statystyka (w modelu przyczynowym na podstawie już zakodowanych danych)
- Aktualizacja statystyki po każdym zakodowanym symbolu
- Modyfikacja (korekcja) drzewa zamiast jego budowy – redukcja złożoności obliczeniowej

## Problemy

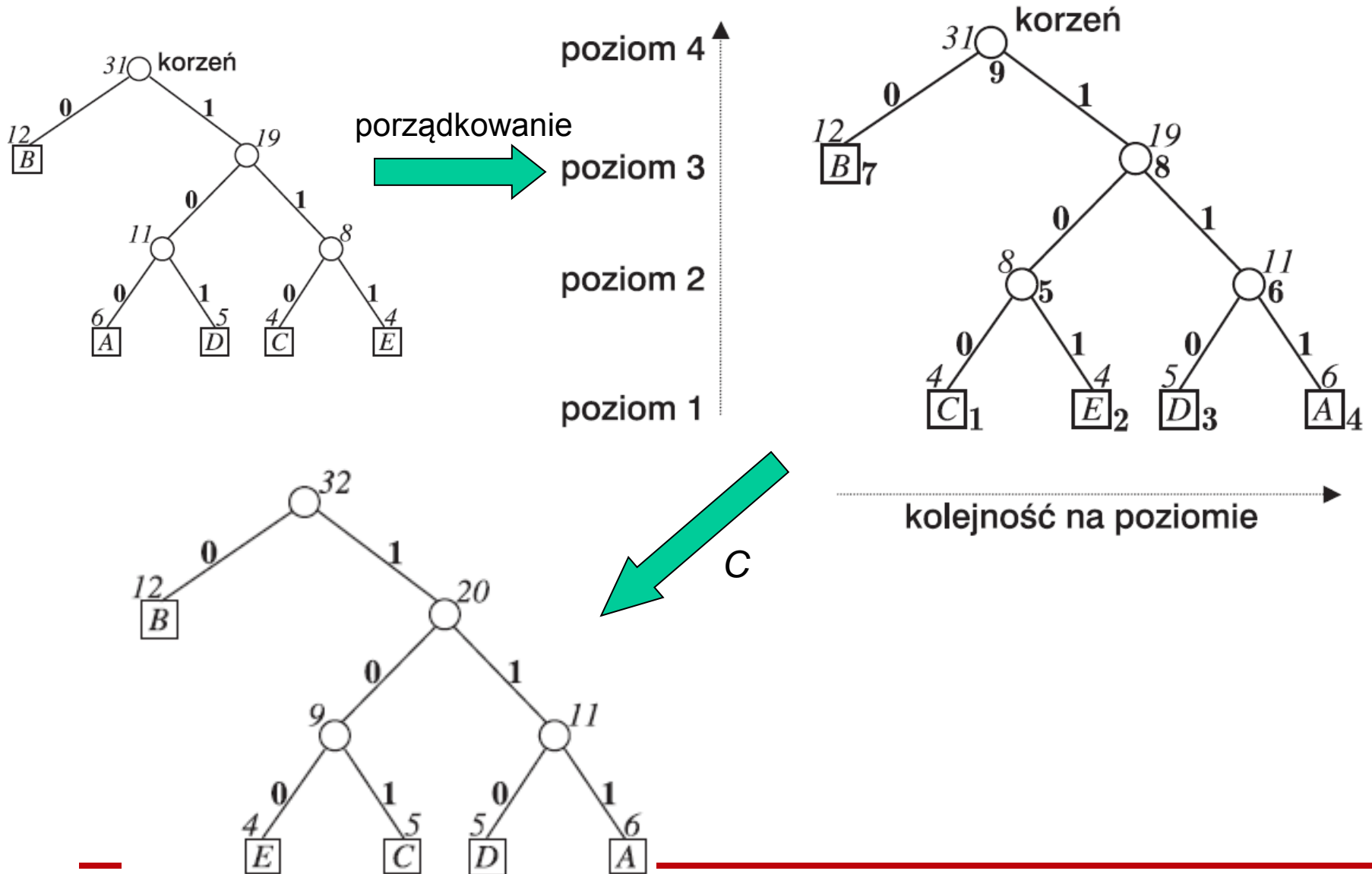
- Przepętnienie drzewa
  - Wprowadzenie nowych symbolu alfabetu (rozbudowa alfabetu)
  - EOF
  - Zagospodarowanie istniejących stopni swobody
-

# Adaptacyjny algorytm Huffmana

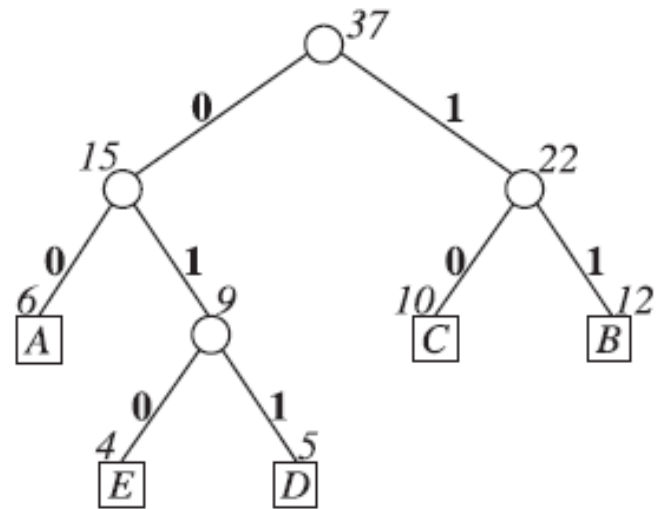
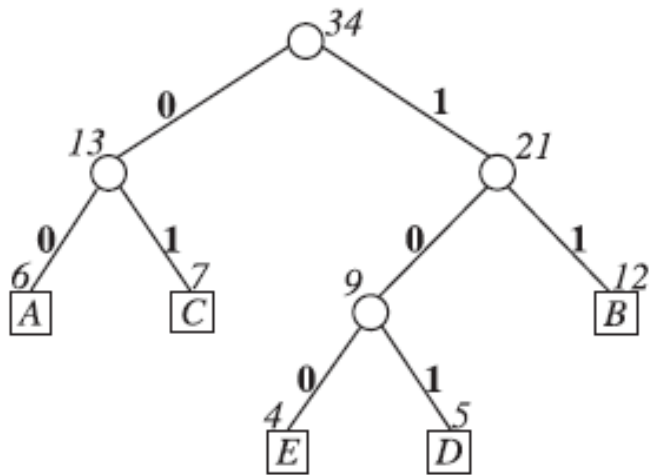
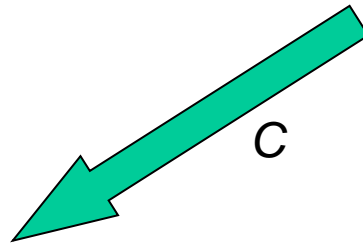
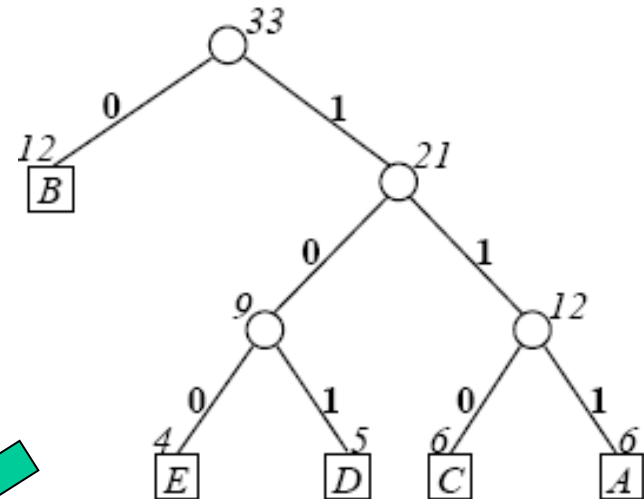
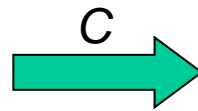
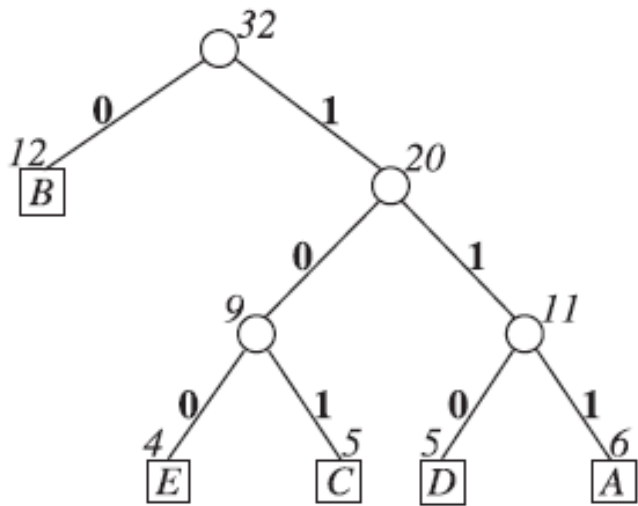
---

1. Inicjalizacja: początkowa postać drzewa Huffmana:
    - a) wierzchołki są **uszeregowane w porządku niemalejących** wag na kolejnych poziomach (od dołu do korzenia, od lewej do prawej);
    - b) każdemu wierzchołkowi jest przypisany **numer porządkowy**
  2. Modyfikacja drzewa po kodowaniu symbolu  $s$  w strumieniu wejściowym:
    - a) ustal numer  $v$  liścia, któremu w drzewie  $T$  przypisano symbol  $s$ :  $v = T(s)$  oraz jego wagę  $w_v$ ; jeśli  $T(s) = \emptyset$  **dodaj nowy liść** z symbolem  $s$  do drzewa:
      - na miejscu liścia o najmniejszej wadze (z numerem 1 na uporządkowanej liście) ustal węzeł wewnętrzny i dołącz do niego dwa węzły potomne: nowy liść z  $s$  i wagą 0 oraz liść o najmniejszej dotąd wadze;
      - zmodyfikuj listę wierzchołków ustalając numer nowego liścia jako  $v = 1$ , numer jego brata jako 2, a numer rodzica jako 3 zwiększając o 2 numery pozostałych węzłów;
    - b) inkrementuj wagę wierzchołka  $v$ :  $w_v \leftarrow w_v + 1$ ;
    - c) zamiana wierzchołków: jeśli na liście wierzchołków za węzłem  $v$  występują węzły z wagą  $w_v - 1$ , to zamień miejscami  $v$  z ostatnim węzłem o tej wadze (wraz z poddrzewami);
    - d) kontynuacja modyfikacji w górę drzewa aż do poziomu korzenia:  
 $v \leftarrow \text{rodzic}(v)$  i skok do p. 2b)
-

# Przykład adaptacyjnej modyfikacji drzewa Huffmana

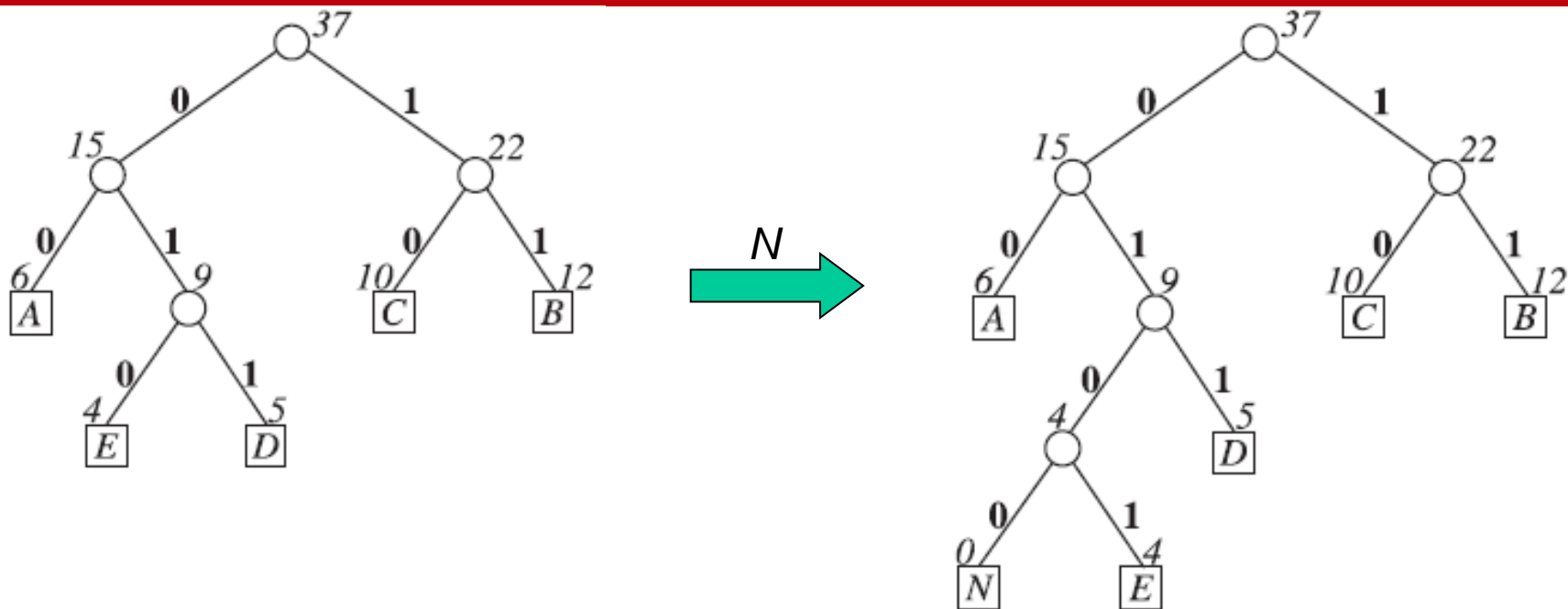


# Adaptacyjna modyfikacja(2)

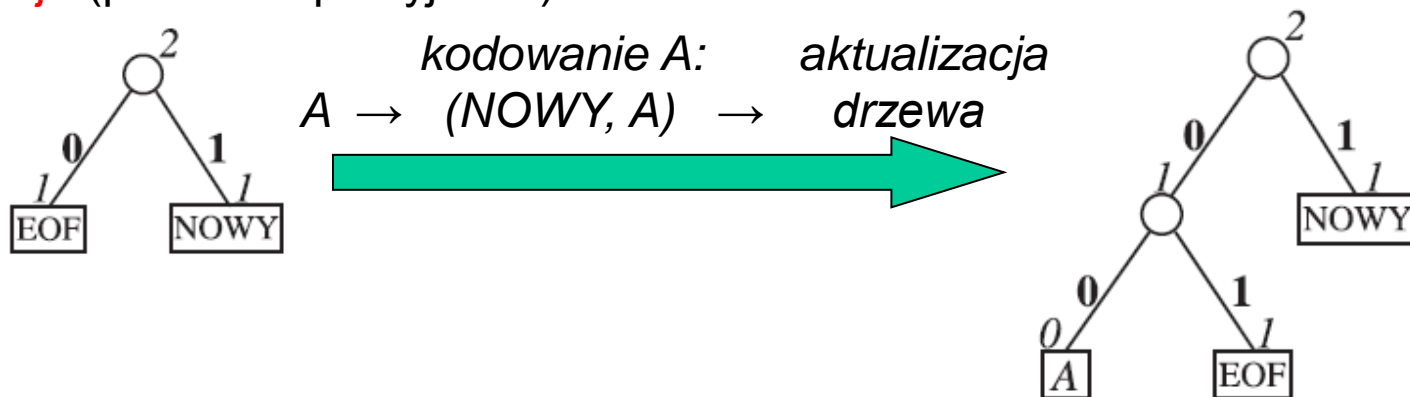




# Dodanie nowego symbolu (inicjalizacja)

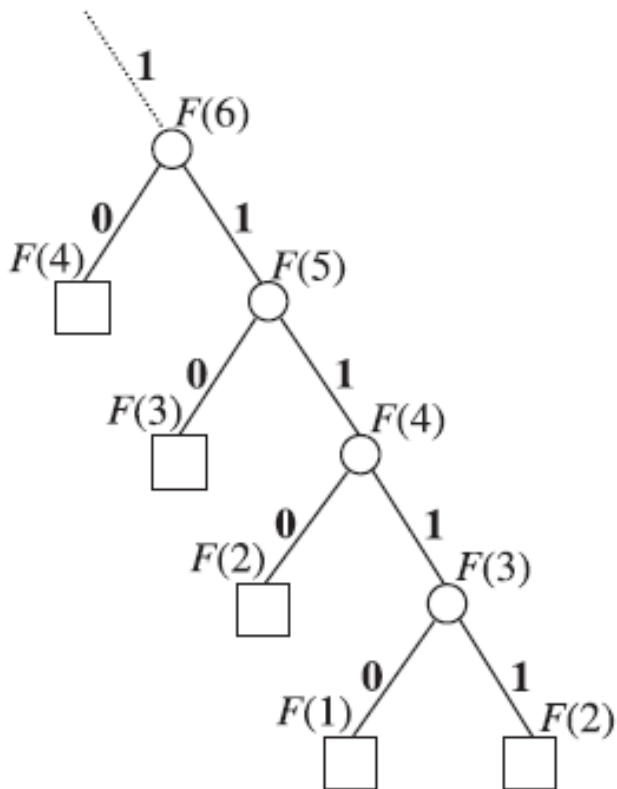


Inicjalizacja (pełna adaptacyjność)

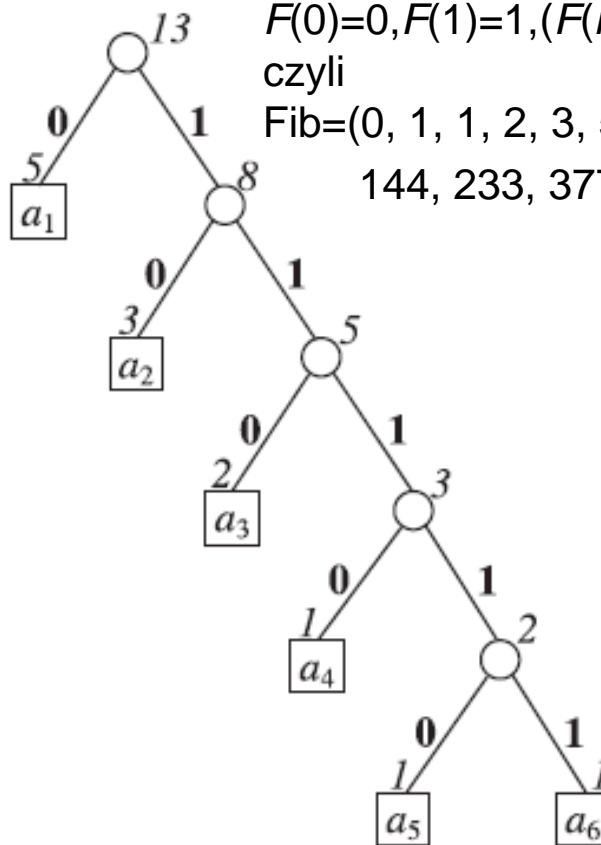


# Przepełnienie drzewa - reakcje

a)



b)



Ciąg Fibonacciego:

$F(0)=0, F(1)=1, (F(i)=F(i-1)+F(i-2)) \mid \text{dla } i=2,3,4,\dots$

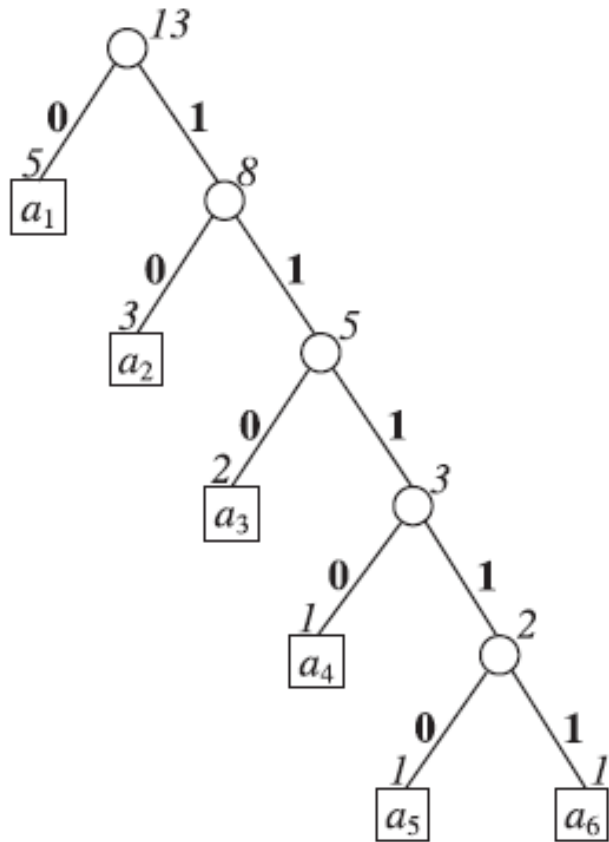
czyli

Fib=(0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89,  
144, 233, 377, 610, 987, 1597, ....)

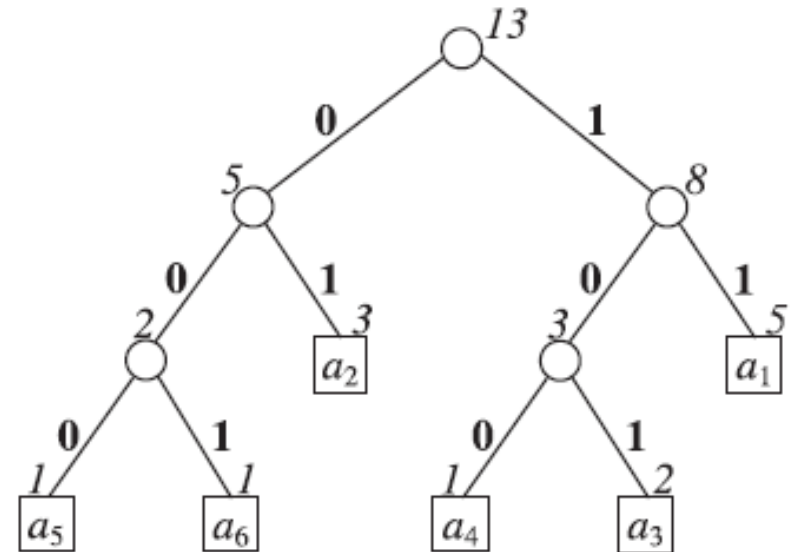
- zamrożenie
- przeskalowanie
- ograniczenie historii

Waga korzenia  $F(i+2)$  może dać maksymalnie słowo  $i$  bitowe;  
przy wadze  $F(19) = 4181$  może wystąpić 17 bitowe słowo

# Dobór postaci słów kodowych w drzewach Huffmanna



*równowartość w efektywności*



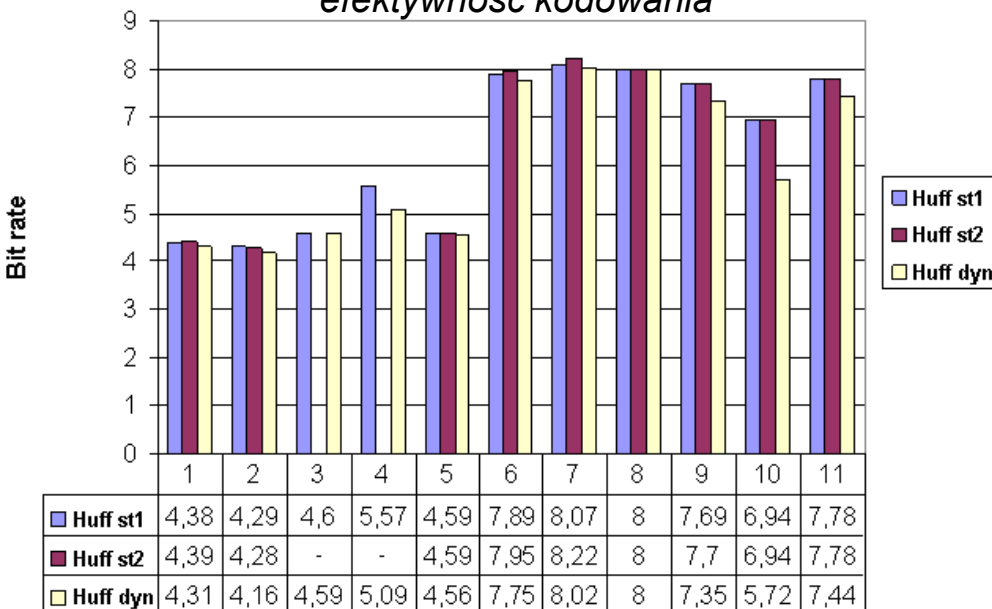
**Zasada: łączymy węzeł z płytszym poddrzewem**

Ta sama reguła może obowiązywać w drzewie adaptacyjnym (algorytm Vittera)

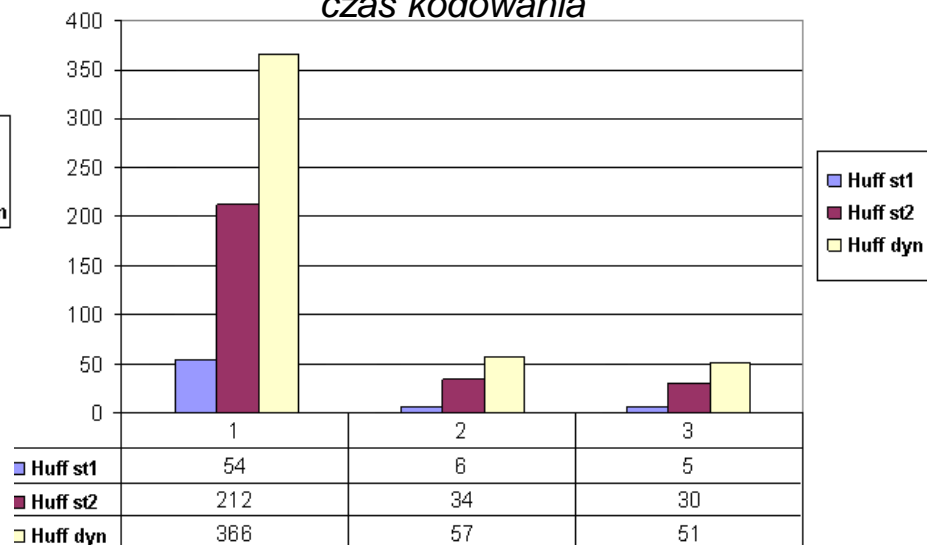
# Efektywność kodu Huffmana

Koder	Zbiór			Średnio
	Z1(10kB)	Z2(100kB)	Z3(4MB)	
Huffman statyczny	5,85	8,06	2,43	5,45
Huffman adaptacyjny	5,33	7,78	2,10	5,07
	+9%	+3,5%	+13,5%	+7%

efektywność kodowania



czas kodowania



- Wyraźna poprawa efektywności metody adaptacyjnej
- Złożoność obliczeniowa jest większa o około 50-100%

---

# **KODY PRZEDZIAŁOWE**

---

# Kody przedziałowe – elastyczne kody symboli z rozszerzeniem pojęcia alfabetu

---

- Kod dwójkowy prawie stałej długości
  - Kod unarny
  - Kod Golomba (*przedziałowy ze stałą długością przedziału*)
  - Efektywność kodu Golomba
  - Kod Rice'a
  - Przykład kodowania z JPEG-LS
  - Elementarny kod Golomba
  - Wykładniczy kod przedziałowy (exp-Golomb)
  - Testy efektywności
-

# Kod dwójkowy prawie stałej długości

- $\hat{B}_n$  (dla  $n$ -elementowego alfabetu  $A_S = \{a_0, \dots, a_{n-1}\}$ ):
  - dla  $a_i, 0 \leq i \leq r-1$  mamy

$$\hat{B}_n(a_i) = B_{\lfloor \log_2 n \rfloor}(i)$$

- dla  $a_i, r \leq i \leq n-1$  mamy

$$\hat{B}_n(a_i) = B_{\lfloor \log_2 n \rfloor}(a_{i+r}) = (i+r)_{2, \lfloor \log_2 n \rfloor}$$

- gdzie

$$r = 2^{\lceil \log_2 n \rceil} - n$$

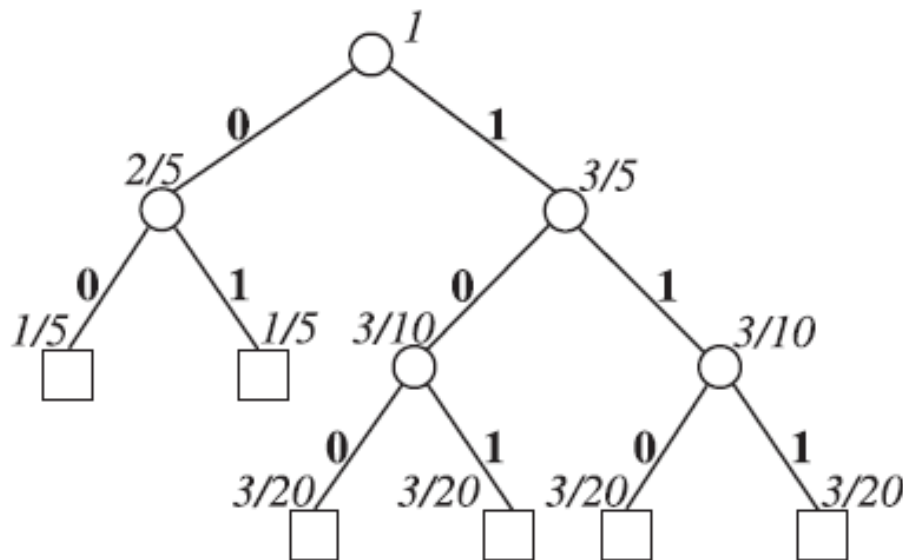
Przykład dla  $A_S = \{0, \dots, n-1\}$  :

$$\hat{B}_6(a_3) = B_3(a_5) = 101, \quad r = 2^{\lceil \log_2 6 \rceil} - 6 = 2$$

$$\hat{B}_{12}(a_3) = B_3(a_3) = 011, \quad r = 2^{\lceil \log_2 12 \rceil} - 12 = 4$$

# Efektywność kodu dwójkowego

Kod dwójkowy prawie stałej długości jest **optymalnym kodem symboli** dla zrównoważonego rozkładu prawdopodobieństw symboli danego źródła (tj. *suma dwóch najmniejszych prawdopodobieństw jest większa od prawdopodobieństwa największego oraz alfabet jest uporządkowany zgodnie z nierosnącym prawdopodobieństwem kolejnych symboli*)



drzewo optymalne



# Kod unarny

- Prosta postać kodu o nieograniczonym alfabcie:

$$U(a_i) = \underbrace{0\dots0}_i 1, \quad i = 0, 1, \dots \quad \text{lub} \quad \bar{U}(a_i) = \underbrace{1\dots1}_i 0$$

↑  
indeks symbolu w alfabecie  $A_S = \{a_0, a_1, \dots, a_i, \dots\}$

- Długość słowa:

$$L_i = i + 1, \text{ czyli jeśli mamy } p_i = 1/2^{i+1} \text{ to } l(a_i) = \log_2 1/p_i = i + 1$$

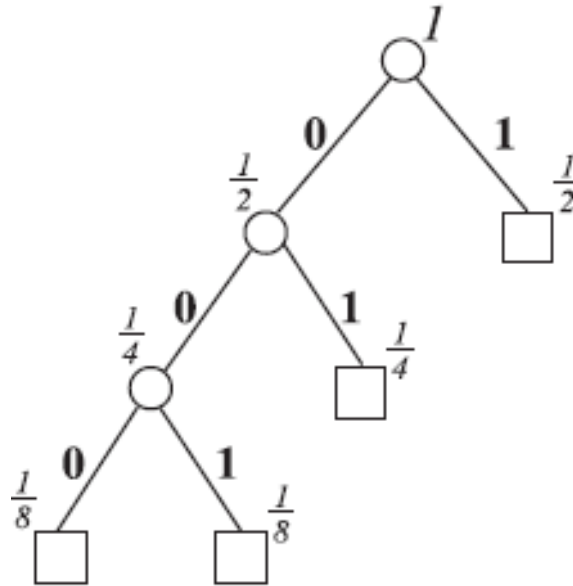
- Wobec tego: kod **jest optymalny** dla rozkładu prawdopodobieństw

$$p_i = 1/2 \cdot 1/2^i$$

- Ogólniej rozkład geometryczny  $p_i = (1-\rho) \cdot \rho^i$ , gdzie  $\rho = 1/2$
-

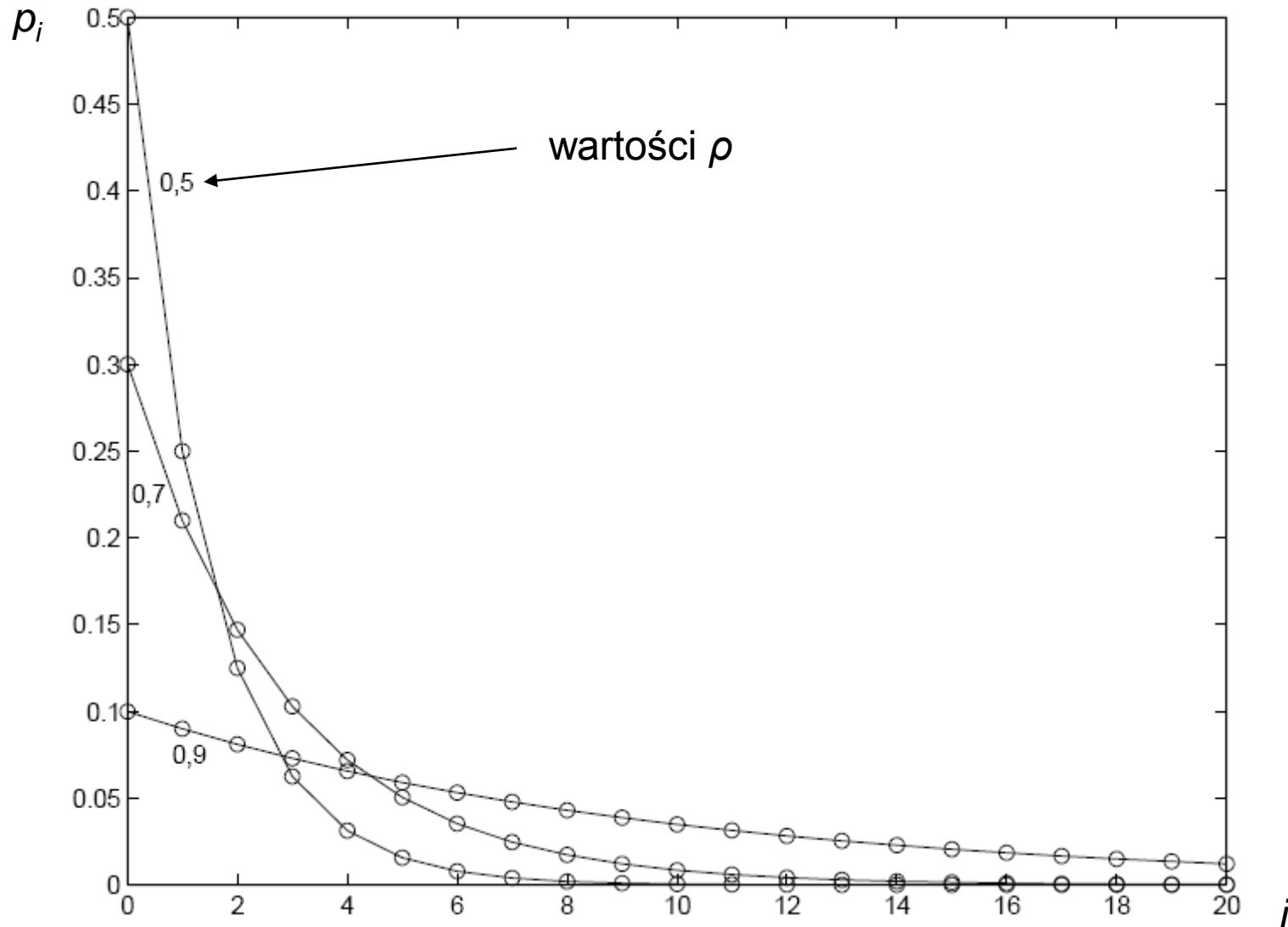
# Optymalne drzewo unarne

- Jeśli  $\rho \leq 1/2$  kod unarny jest kodem optymalnym, bo  $\log_2 1/\rho > 1$



drzewo unarne

# Jaki kod jest optymalny dla $\rho > 1/2$



# Kod Golomba

- Kod o nieograniczonym alfabetcie
- Robi się użytek z kodu dwójkowego i unarnego
- Alfabet nieskończony  $A_S = \{a_0, a_1, \dots, a_i, \dots\}$ , uporządkowany nierosnąco według prawdopodobieństw  $p_i \geq p_{i+1}$
- Podział alfabetu na podprzedziały stałej długości  $m > 0$  (rząd  $m \geq 1$  kodu)

$$A_S = \{ \underbrace{a_0, \dots, a_{m-1}}_{\text{przedział 0}}, \underbrace{a_m, \dots, a_{2m-1}}_{\text{przedział 1}}, a_{2m}, \dots \}$$

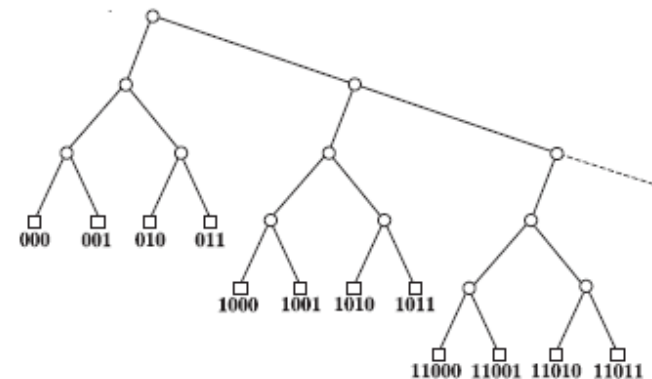
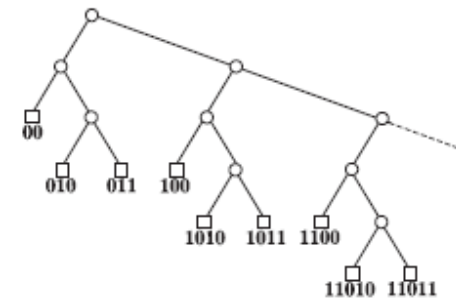
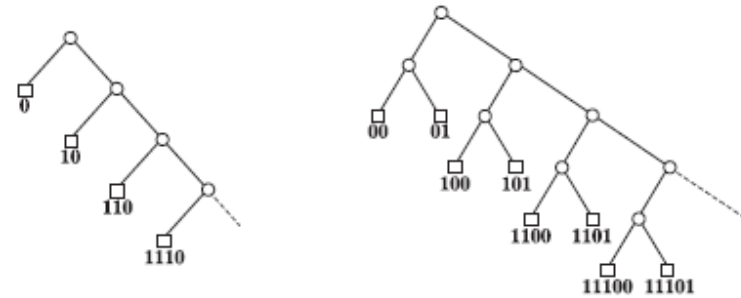
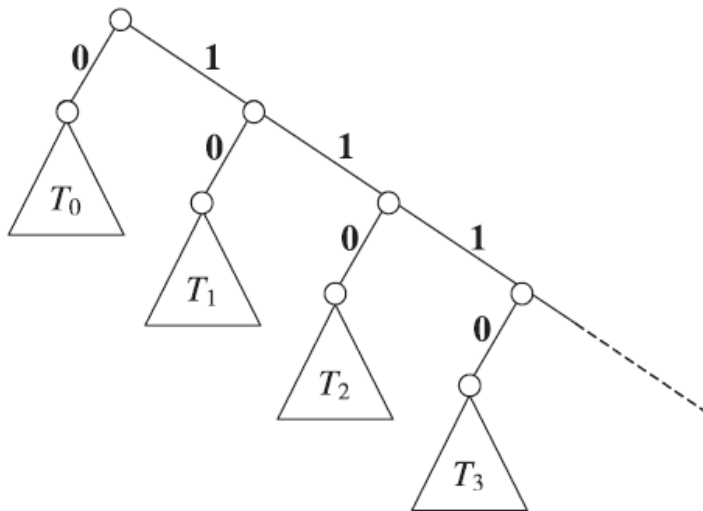
- Kod  $G_m(i) = U(\text{nr przedziału } k: a_i \in \pi_k) \hat{B}_m(\text{odległość } l = i - k \cdot m \text{ od początku przedziału})$ , czyli

$$G_m(i) = U(\lfloor i/m \rfloor) \hat{B}_m(i \bmod m)$$



# Cechy kodu Golomba

- Minimalna długość słowa:  $\lfloor \log_2 m \rfloor + 1$
- Drzewo kodu Golomba



# Efektywność kodu Golomba

---

- Interpretacja rzędu kodu:  $\rho^m \cong 1/2$  (wolniej narasta długość słów)

- Dobór  $m$

$$m = \left\lceil \frac{\log(1 + \rho)}{\log \rho^{-1}} \right\rceil$$

- Przykładowo dla  $\rho=0,9$  wartość optymalnego  $m$  wynosi 7
  - Możliwy jest adaptacyjny dobór rzędu kodu, np. w JPEG-LS
-

# Testy z kodem Golomba

Ro_083_09390_069.tst	CR	Czas (sek.)
m = 4	1,672	2,463
m = 5	1,721	2,844
m = 6	1,736	2,935
m = 7	1,725	2,543
Adaptacyjny	<b>1,880</b>	2,644

Ro_087.tst	Stopień kompresji	Czas (sek.)
m = 4	1,829	2,273
m = 5	<b>1,843</b>	2,774
m = 6	1,835	2,814
Adaptacyjny	<b>1,843</b>	3,104

Ro_09571_09266_06800.tst	CR	Czas (sek.)
m = 8	1,481	3,675
m = 9	1,495	3,034
m = 10	1,503	3,015
m = 11	1,506	3,535
m = 12	1,505	3,465
m = 13	1,498	3,525
adaptacyjny	<b>1,630</b>	3,315

Aktualizacja rzędu	CR	Czas (sek.)
10	1,880	18,857
100	1,880	3,756
1000	1,879	2,334
10000	1,866	2,223
100000	1,763	2,243



# Kod Rice'a

---

- Wygodna implementacyjnie wersja kodu Golomba z ograniczeniem na  $m=2^k$ , czyli

$$R_k = G_{m=2^k}$$

- Tworzenie słowa: odcinamy  $k$  bitów i dodajemy unarnie zapisaną liczbę utworzoną z bitów pozostałych

przykład:

$$R_3(12) = R_3(1100_2) = 01\ 100$$

$$R_3(19) = R_3(10011_2) = 001\ 011$$

---

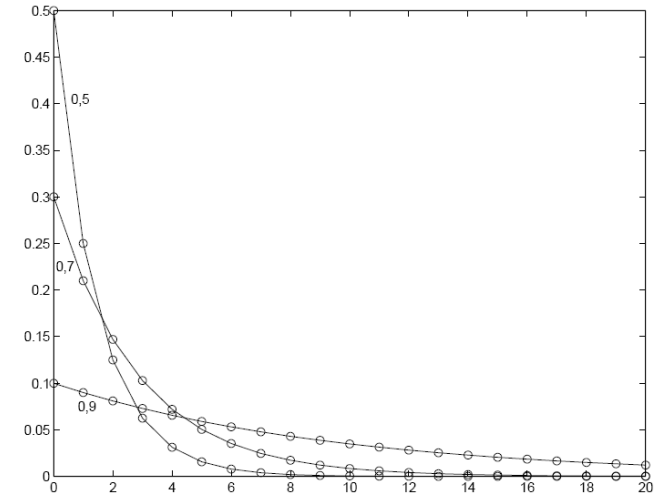
# Słowa kodu Rice'a

Słowa kodu Rice'a określonego rzędu					
$i$	$k = 1$	$k = 2$	$k = 3$	$k = 4$	$k = 5$
<b>0</b>	1 0	1 00	1 000	1 0000	1 00000
<b>1</b>	1 1	1 01	1 001	1 0001	1 00001
<b>2</b>	01 0	1 10	1 010	1 0010	1 00010
<b>3</b>	01 1	1 11	1 011	1 0011	1 00011
<b>4</b>	001 0	01 00	1 100	1 0100	1 00100
<b>5</b>	001 1	01 01	1 101	1 0101	1 00101
<b>6</b>	0001 0	01 10	1 110	1 0110	1 00110
<b>7</b>	0001 1	01 11	1 111	1 0111	1 00111
<b>8</b>	00001 0	001 00	01 000	1 1000	1 01000
<b>9</b>	00001 1	001 01	01 001	1 1001	1 01001
⋮	...	...	...	...	...

# Adaptacyjny kod Rice'a z JPEG-LS

- Dobór rzędu kodu przy określonym kontekście wystąpienia danej (symbolu)

wartość kodowanej danej to E



```
/* Zmienna A[0..364] zawiera sumę modułów błędów predykcji
kontekstu Q; w N[0..364] liczone są przypadki wystąpień
Q od inicjalizacji; A[Q] i N[Q] są aktualizowane zgodnie
z aktualnym błędem predykcji E*/
```

```
A[Q] = A[Q] + abs(E); /* aktualizacja zmiennych */
```

```
N[Q] = N[Q] + 1;
```

```
...
```

```
for(k=0; (N[Q]<<k)<A[Q]; k++); /* wyznaczenie rzędu k */
```

Prediction error	Mapped value	Code <i>LG(2,32)</i>
0	0	1 00
-1	1	1 01
1	2	1 10
-2	3	1 11
2	4	01 00
-3	5	01 01
3	6	01 10
-4	7	01 11
4	8	001 00
-5	9	001 01
5	10	001 10
-6	11	001 11
6	12	0001 00
-7	13	0001 01
7	14	0001 10
-8	15	0001 11
8	16	00001 00
-9	17	00001 01
9	18	00001 10
13s4 -10	19	00001 11
10	20	000001 00
-11	21	000001 01
11	22	000001 10
-12	23	000001 11
12	24	0000001 00
...	...	...
50	100	000000000000000000000000 00001 01100011

# Tablica kodowania z JPEG-LS

Nowy alfabet poprzez mapowanie wartości

Kod Rice'a z ograniczeniem

# Kod exp-Golomb

$i$	Słowo kodowe
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
9	0001010
....	

$$\xi_i = G_{exp}(i) = \underbrace{0..0}_m 1 (i+1-2^m)_{2,m}$$

$$\text{gdzie } m = \lfloor \log_2(i+1) \rfloor$$

Zastosowanie np. w CAVLC (h.264)

# Elementarny kod Golomba

---

- Robi się użytek z kodu dwójkowego i unarnego
- Rozszerza się alfabet źródła binarnego (modelowanie) do postaci  $A_S = \{1, 01, \dots, 0^i 1, \dots, 0^{m-1} 1, 0^m\}$ ,  
gdzie  $m$  jest parametrem kodu  $EG_m$  (rozmiar alfabetu wynosi  $m+1$ )

- Słowa kodowa dla rozszerzonego alfabetu

$$A_{EG_m} = \{1\hat{B}_m(0), 1\hat{B}_m(1), \dots, 1\hat{B}_m(i), \dots, 1\hat{B}_m(m-1), 0\}$$

- Przykład  $EG_3(00000001) = EG_3(000)EG_3(000)EG_3(01) =$   
 $= 0\ 0\ 1\hat{B}_3(1) = 00110$

- Nietypowa sytuacja: kod symboli czy strumieniowy (RLE)?
-

# Zestaw słów $EG_m$

Alfabet	Bitowe ciągi alfabetu dla $m = 8$	Słowa kodowe dla $m = 8$	Bitowe ciągi alfabetu dla $m = 9$	Słowa kodowe dla $m = 9$
$a_0$	1	1000	1	1000
$a_1$	01	1001	01	1001
$a_2$	001	1010	001	1010
$a_3$	0001	1011	0001	1011
$a_4$	00001	1100	00001	1100
$a_5$	000001	1101	000001	1101
$a_6$	0000001	1110	0000001	1110
$a_7$	00000001	1111	00000001	11110
$a_8$	00000000	0	000000001	11111
$a_9$	—	—	000000000	0

# Zróżnicowanie kodów binarnych

