Kompresja danych DKDA (3)

Marcin Gogolewski
marcing@wmi.amu.edu.pl

Uniwersytet im. Adama Mickiewicza w Poznaniu

Poznań, 17 października 2025

Kodowanie arytmetyczne

- Tekst wejściowy zostaje odwzorowany na liczbę z przedziału $[0, 1)$.
- Kod tekstu to liczba n – długość tekstu i liczba z (znacznik) reprezentowany z odpowiednio dobraną dokładnością.
- Elementom alfabetu $a_1, \ldots, a_N$ o prawdopodobieństwach $p_1, \ldots, p_N$ przyporządkowujemy przedział $[F(i), F(i+1))$, gdzie $F(i) = \sum_{j=1}^{i-1} p_j$.

Kodowanie

- Mamy ciąg liter $x_1 x_2 \ldots x_m$ z alfabetu $a_1, \ldots, a_N$.
- Na początku przedział $[l, p) = [0, 1)$.
- Dla $i = 1, 2, \ldots, m$:
  - Niech $x_i = a_j$.
  - Wtedy $d \leftarrow p - l$, $p \leftarrow l + F(j+1)d$ i $l \leftarrow l + F(j)d$.
- Znacznik to dowolna liczba z przedziału $[l, p)$, np. $z = (l + p)/2$.

Przykład

- Weźmy alfabet a, b, c z prawdopodobieństwami $0.7, 0.1, 0.2$.
- Zakodujmy tekst abc.
- Na początku mamy przedział: $[0, 1)$.
- $F(1) = 0$, $F(2) = 0.7$, $F(3) = 0.8$, $F(4) = 1$.
- Kodujemy a i otrzymujemy przedział $[0, 0.7)$.
- Kodujemy b i otrzymujemy przedział $[0.49, 0.56)$.
- Kodujemy c i otrzymujemy przedział $[0.546, 0.56)$.
- Za znacznik możemy przyjąć 0.553.

Własności

- Dla ustalonej długości tekstu n, każdy ciąg jest odwzorowany na przedział rozłączny z przedziałami odpowiadającymi innym ciągom (tej samej długości!). Gwarantuje to jednoznaczność kodowania.
- Wygenerowanie znacznika dla konkretnego ciągu nie wymaga wyznaczania bądź pamiętania znaczników innych ciągów.

Dekodowanie

- Dostajemy n – długość tekstu i z – znacznik tekstu.
- $l \leftarrow 0$ i $p \leftarrow 1$.
- Dla $i = 1, 2, \ldots, n$:
  - wybieramy j takie, że $l + F(j)(p - l) \leqslant z < l + F(j + 1)(p - l)$;
  - przyjmujemy, że $x_i = a_j$;
  - $d \leftarrow p - l$, $p \leftarrow l + F(j + 1)d$ i $l \leftarrow l + F(j)d$.
- Ciąg oryginalny to $x_1, \ldots, x_n$.

Przykład

- $P(a) = 0.7$, $P(b) = 0.1$, $P(c) = 0.2$, $z = 0.55$ i $n = 3$.
- $F(1) = 0$, $F(2) = 0.7$, $F(3) = 0.8$ i $F(4) = 1$.
- $l = 0$ i $p = 1$.
- Dla a mamy $0 \leqslant 0.55 < 0.7$, stąd $x_1 = a$, $l = 0$ i $p = 0.7$.
- Dla b mamy $0.49 \leqslant 0.55 < 0.56$, stąd $x_2 = b$, $l = 0.49$ i $p = 0.56$.
- Dla c mamy $0.546 \leqslant 0.55 < 0.56$, stąd $x_3 = c$, $l = 0.546$ i $p = 0.56$.
- Odkodowany ciąg to *abc*.

Własności

- Jak reprezentować znacznik (liczba rzeczywista) aby był jak najkrótszy.
- Niech $x = x_1, \ldots, x_n$ będzie ciągiem danych o prawdopodobieństwie wystąpienia $P(x) = \Pi_{i=1}^{n} P(x_i)$. Zaokrąglenie z' znacznika z do

$$m(x) = \left\lceil \log \frac{1}{P(X)} \right\rceil + 1$$

bitów, gwarantuje jednoznaczność kodowania.
- Kod arytmetyczny dla ustalonej długości tekstu jest kodem prefiksowym.

Przykład

- $P(a) = 0.7$, $P(b) = 0.1$, $P(c) = 0.2$.
- Kod dla tekstu abc to $0.553 = (0.100011011)_2$.
- $P(abc) = 0.014$.
- $\left\lceil \log \frac{1}{0.014} \right\rceil + 1 = 8$.
- Czyli do zakodowania tekstu wystarczy wysłać 10001101.

Usprawnienia

- Dla długich ciągów potrzebujemy długich liczb i przetwarzanie wymaga przeczytania całego ciągu.
- Można zmodyfikować algorytm do pracy przyrostowej – znacznik powstaje etapami i można wysyłać go fragmentami.

Kodowanie ze skalowaniem

Po zakodowaniu kolejnej litery:

- Jeśli $[l, p) \subseteq [0, 0.5)$:
  - zamień $[l, p)$ na $[2l, 2p)$;
  - dołącz do kodu słowo $01^{licznik}$;
  - $licznik \leftarrow 0$.
- Jeśli $[l, p) \subseteq [0.5, 1)$:
  - zamień $[l, p)$ na $[2l - 1, 2p - 1)$;
  - dołącz do kodu słowo $10^{licznik}$;
  - $licznik \leftarrow 0$.
- Jeśli $l < 0.5 < p$ i $[l, p) \subseteq [0.25, 0.75)$:
  - zamień $[l, p)$ na $[2l - 0.5, 2p - 0.5)$;
  - $licznik \leftarrow licznik + 1$.

Analogicznie można zmodyfikować procedurę dekodowania, aby dekodowanie odbywało się na podstawie otrzymywanych fragmentów.

Dalsze możliwe usprawnienia

- Przejście z arytmetyki zmiennoprzecinkowej na arytmetykę całkowitoliczbową: zastąpienie przedziału $[0, 1)$ przez przedział liczb całkowitych $[0, 2^m - 1]$.
- Problem: Jak dobrać m aby uniknąć błędów zaokrągleń?
- Jak zakodować długość ciągu (kiedy zakończyć dekodowanie)?

Kodowanie słownikowe – motywacja

- Kompresowane dane nie tworzą zwykle ciągu wartości niezależnych (kolejne symbole są zależne od poprzednich).
- Pewne ciągi (słowa) powtarzają się bardzo często.

Słowniki statyczne

- Mamy ustalony stały słownik.
- Tekst kodujemy jako ciąg pozycji ze słownika.
- Co z elementami których nie ma w słowniku?
  Można np. umieścić pojedyncze litery w słowniku.
- **Problem:** słowniki statyczne są wrażliwe na zmianę charakteru danych.

Przykład: kodowanie tekstów programów – statyczny słownik słów kluczowych.

Kodowanie digramowe

- Ustalamy wielkość słownika np. 256 (2^8).
- Słownik składa się ze wszystkich liter i tylu par liter (digramów), ile się jeszcze zmieści (umieszczamy tam najbardziej prawdopodobne pary).
- Na przykład w kodzie ASCII używamy przeważnie tylko 95 znaków (znaki drukowalne). Pozostałe (161) możemy zastąpić najczęściej występującymi parami.
- **Problem:** Słowniki nie są uniwersalne, a poziom kompresji niezbyt wysoki.

Słowniki dynamiczne

- Słownik tworzymy w trakcie kodowania.
- Słownik dostosowuje się do charakteru danych.
- Dekoder może go odtworzyć w oparciu o odkodowaną część danych (nie trzeba go dołączać do danych).

Ziv, Lempel, LZ77

- **Idea:** słownikiem jest zakodowana/odkodowana część tekstu.
- Dla zakodowanej części długości n i niezakodowanej długości m, czyli dla ciągu $x_1 \dots x_n x_{n+1} \dots x_{n+m}$ szukamy najdłuższego podsłowa w ciągu $x_1 \dots x_n$ będącego prefiksem ciągu $x_{n+1} \dots x_{n+m}$ (dopasowanie).
- Jako kod podajemy trzy liczby:
  - wielkość przesunięcia w lewo,
  - liczbę kopiowanych znaków,
  - kod pierwszej niepasującej litery.
- Jeśli pojawia się nowa litera, której nie można znaleźć w zakodowanej części, to wysyłamy trójkę $(0, 0, \mathrm{kod})$.

Przykład – kodowanie

- Ustalamy $n = 7$ i $m = 8$ ($2^k - 1$ i 2^k).
- wabba-wabba-wabba-woo-woo-woo (0,0,k(w))
- wabba-wabba-wabba-woo-woo-woo (0,0,k(a))
- wabba-wabba-wabba-woo-woo-woo (0,0,k(b))
- wabba-wabba-wabba-woo-woo-woo (1,1,k(a))
- wabba-wabba-wabba-woo-woo-woo (0,0,k(-))
- wabba-wabba-wabba-woo-woo-woo (6,7,k(a))
- wabba-wabba-wabba-woo-woo-woo (6,5,k(o))
- wabba-wabba-wabba-woo-woo-woo (1,1,k(-))
- wabba-wabba-wabba-woo-woo-woo (4,6,k(o))

Przykład – dekodowanie

- Ustalamy $n = 7$ i $m = 8$ $(2^k - 1$ i $2^k)$.
- (0,0,k(w)) w
- (0,0,k(a)) wa
- (0,0,k(b)) wab
- (1,1,k(a)) wabba
- (0,0,k(-)) wabba-
- (6,7,k(a)) wabba-wabba-wa
- (6,5,k(o)) wabba-wabba-wabba-wo
- (1,1,k(-)) wabba-wabba-wabba-woo-
- (4,6,k(o)) wabba-wabba-wabba-woo-woo-woo

LZ77 - podsumowanie

- Mamy bufor słownika i bufor kodowania – razem nazywamy to oknem.
- **Co można usprawnić:** Zamiast 3 liczb wysyłać tylko dwie: kopiowany ciąg (bez kodu ostatniej litery) lub 0 i kod nowej litery.
- Krotki kodowane przy pomocy algorytmu Huffmana (adaptacyjnego).
- Zastosowanie: zip, gzip, png, arj, rar, ...

Ziv, Lempel, LZ78

- Powtórzenia nie muszą być na małej odległości – wada LZ77.
- Słownik to zbiór numerowanych słów.
- Zawartość słownika jest tworzona w oparciu o zakodowaną część tekstu.
- Kodowanie to ciąg indeksów ze słownika tworzący kodowany tekst.

LZ78 - Algorytm

- Na początku słownik jest pusty (ewentualnie ma jeden wpis $0 - \varepsilon$).
- Szukamy w słowniku najdłuższego prefiksu tekstu i wysyłamy numer tego słowa oraz kod następnej litery. Do słownika dodajemy kolejne nowe słowo (to które wysłaliśmy). Jeśli w słowniku nie ma takiego prefiksu to wysyłamy 0 i kod pierwszej litery.
- Czasami na samym końcu wysyłamy krótszy prefiks i ostatnią literę.

wabba-wabba-wabba-woo-woo-woo

- (0,k(w)) 1 = w abba-wabba-wabba-woo-woo-woo
- (0,k(a)) 2 = a bba-wabba-wabba-woo-woo-woo
- (0,k(b)) 3 = b ba-wabba-wabba-woo-woo-woo
- (3,k(a)) 4 = ba -wabba-wabba-woo-woo-woo
- (0,k(-)) 5 = - wabba-wabba-woo-woo-woo
- (1,k(a)) 6 = wa bba-wabba-woo-woo-woo
- (3,k(b)) 7 = bb a-wabba-woo-woo-woo
- (2,k(-)) 8 = a- wabba-woo-woo-woo
- (6,k(b)) 9 = wab ba-woo-woo-woo
- (4,k(-)) 10 = ba- woo-woo-woo
- (1,k(o)) 11 = wo o-woo-woo
- (0,k(o)) 12 = o -woo-woo
- (5,k(w)) 13 = -w oo-woo
- (12,k(o)) 14 = oo -woo
- (13,k(o)) 15 = -wo o
- (0,k(o))

Odkodowywanie

- (0,k(w)) 1 = w w
- (0,k(a)) 2 = a wa
- (0,k(b)) 3 = b wab
- (3,k(a)) 4 = ba wabba
- (0,k(-)) 5 = - wabba-
- (1,k(a)) 6 = wa wabba-wa
- (3,k(b)) 7 = bb wabba-wabb
- (2,k(-)) 8 = a- wabba-wabba-
- (6,k(b)) 9 = wab wabba-wabba-wab
- (4,k(-)) 10 = ba- wabba-wabba-wabba-
- (1,k(o)) 11 = wo wabba-wabba-wabba-wo
- (0,k(o)) 12 = o wabba-wabba-wabba-woo
- (5,k(w)) 13 = -w wabba-wabba-wabba-woo-w
- (12,k(o)) 14 = oo wabba-wabba-wabba-woo-woo
- (13,k(o)) 15 = -wo wabba-wabba-wabba-woo-woo-wo
- (0,k(o)) wabba-wabba-wabba-woo-woo-woo

Ziv, Lempel, Welch – LZW

- Poprawki do LZ78 – rezygnacja z drugiego elementu pary.
- Potrzebny słownik początkowy zawierający wszystkie używane litery.
- Słownik konstruujemy tak samo jak w LZ78.

Kodowanie

- Słownik początkowy: 0 = a, 1 = b, 2 = o, 3 = w, 4 = -.
- wabba-wabba-wabba-woo-woo-woo
- 3 5 = wa abba-wabba-wabba-woo-woo-woo
- 0 6 = ab bba-wabba-wabba-woo-woo-woo
- 1 7 = bb ba-wabba-wabba-woo-woo-woo
- 1 8 = ba a-wabba-wabba-woo-woo-woo
- 0 9 = a- -wabba-wabba-woo-woo-woo
- 4 10 = -w wabba-wabba-woo-woo-woo
- 5 11 = wab bba-wabba-woo-woo-woo
- 7 12 = bba a-wabba-woo-woo-woo
- 9 13 = a-w wabba-woo-woo-woo
- 11 14 = wabb ba-woo-woo-woo
- 8 15 = ba- -woo-woo-woo
- 10 16 = -wo oo-woo-woo

Kodowanie

- 2 17 = oo o-woo-woo
- 2 18 = o- -woo-woo
- 16 19 = -woo o-woo
- 18 20 = o-w woo
- 3 21 = wo oo
- 17

Dekodowanie

- Słownik początkowy: 0 = a, 1 = b, 2 = o, 3 = w, 4 = -.
- 3 w 5 = w?
- 0 wa 5 = wa, 6 = a?
- 1 wab 6 = ab, 7 = b?
- 1 wabb 7 = bb, 8 = b?
- 0 wabba 8 = ba, 9 = a?
- 4 wabba- 9 = a-, 10 = -?
- 5 wabba-wa 10 = -w, 11 = wa?
- 7 wabba-wabb 11 = wab, 12 = bb?
- 9 wabba-wabba- 12 = bba, 13 = a-?
- 11 wabba-wabba-wab 13 = a-w, 14 = wab?
- 8 wabba-wabba-wabba 14 = wabb, 15 = ba?
- 10 wabba-wabba-wabba-w 15 = ba-, 16 = -w?
- ...

Trudny przypadek

- Słowo abababababab... i słownik $1 = a$ i $2 = b$.
- $1 \quad 3 = ab$, $2 \quad 4 = ba$, $3 \quad 5 = aba$, $5 \quad 6 = abab$, ...
- Dekodowanie:
  - $1 \quad a \quad 3 = a$?
  - $2 \quad ab \quad 3 = ab, 4 = b$?
  - $3 \quad abab \quad 4 = ba, 5 = ab$?
  - $5 \quad ababab$?
- Ten problem można rozwiązać przez porównanie z algorytmem kodowania (wywnioskować, że $5 = aba$).

bzip2

- Algorytm kompresji popularny w systemach linuksowych.
- Standardowa implementacja kompresuje bloki danych o rozmiarach od 100 do 900 kb.
- Blok jest przekształcany transformatą Burrowsa-Wheelera, następnie kodowany algorytmem Move-To-Front i kompresowany algorytmem Huffmana.
- Algorytm osiąga o 10%-20% lepsze wyniki niż najczęściej stosowane algorytmy strumieniowe, ale nie może być używany do kodowania strumieni
  (cały ciąg musi być znany przed rozpoczęciem kodowania!).

Transformata Burrowsa-Wheelera

- Na początku mamy blok danych o rozmiarze N, np.

a	l	a	-	m	a	-	k	o	t	a

- Generujemy wszystkie N rotacji kompresowanego bloku

0	a	l	a	-	m	a	-	k	o	t	a
1	l	a	-	m	a	-	k	o	t	a	a
2	a	-	m	a	-	k	o	t	a	a	l
3	-	m	a	-	k	o	t	a	a	l	a
4	m	a	-	k	o	t	a	a	l	a	-
5	a	-	k	o	t	a	a	l	a	-	m
6	-	k	o	t	a	a	l	a	-	m	a
7	k	o	t	a	a	l	a	-	m	a	-
8	o	t	a	a	l	a	-	m	a	-	k
9	t	a	a	l	a	-	m	a	-	k	o
10	a	a	l	a	-	m	a	-	k	o	t

Transformata Burrowsa-Wheelera

- Sortujemy powstałe łańcuchy leksykograficznie

0	-	k	o	t	a	a	l	a	-	m	a
1	-	m	a	-	k	o	t	a	a	l	a
2	a	-	k	o	t	a	a	l	a	-	m
3	a	-	m	a	-	k	o	t	a	a	l
4	a	a	l	a	-	m	a	-	k	o	t
5	a	l	a	-	m	a	-	k	o	t	a
6	k	o	t	a	a	l	a	-	m	a	-
7	l	a	-	m	a	-	k	o	t	a	a
8	m	a	-	k	o	t	a	a	l	a	-
9	o	t	a	a	l	a	-	m	a	-	k
10	t	a	a	l	a	-	m	a	-	k	o

- Wynik transformacji: numer wiersza z pierwotnym tekstem i ostatnia kolumna tabeli.

5

a	a	m	l	t	a	-	a	-	k	o

Odwracanie transformacji

- Mamy numer wiersza 5 i tekst

a	a	m	l	t	a	-	a	-	k	o

- Dopisujemy do tekstu indeksy

0	1	2	3	4	5	6	7	8	9	10
a	a	m	l	t	a	-	a	-	k	o
0	1	2	3	4	5	6	7	8	9	10

- I sortujemy tekst (stabilnie)

0	1	2	3	4	5	6	7	8	9	10
-	-	a	a	a	a	k	l	m	o	t
6	8	0	1	5	7	9	3	2	10	4

- Teraz zaczynając od pozycji 5 wypisujemy kolejno znaki zgodnie z indeksami $5 \to 7 \to 3 \to 1 \to 8 \to 2 \to 0 \to 6 \to 9 \to 10 \to 4$ i otrzymujemy

a	l	a	-	m	a	-	k	o	t	a

Move-To-Front

- Transformacja strumienia danych mogąca spowodować zmniejszenie entropii.
- Zyski gdy symbole wykazują tendencję do lokalnego grupowania się.
- Dane o takiej charakterystyce są często wynikiem transformaty Burrowsa-Wheelera.

Transformacja

- Na początku mamy tablicę N posortowanych symboli. Kodem symbolu jest numer pozycji w tablicy (numery od 0 do $N-1$).
- W każdym kroku bierzemy pojedynczy symbol, wypisujemy jego numer z tablicy i modyfikujemy tablicę przesuwając rozpatrywany symbol na początek tablicy.
- Jeśli symbole są lokalnie pogrupowane to są przesuwane na początek tablicy i w kodzie wynikowym pojawiają się częściej małe liczby.
- Odwrócenie transformacji jest praktycznie identyczne jak algorytm transformacji.

Przykład transformacji

- a a m l t a - a - k o
- 1 a m l t a - a - k o
- 1 0 m l t a - a - k o
- 1 0 4 l t a - a - k o
- 1 0 4 4 t a - a - k o
- 1 0 4 4 6 a - a - k o
- 1 0 4 4 6 3 - a - k o
- 1 0 4 4 6 3 4 a - k o
- 1 0 4 4 6 3 4 1 - k o
- 1 0 4 4 6 3 4 1 1 k o
- 1 0 4 4 6 3 4 1 1 5 o
- 1 0 4 4 6 3 4 1 1 5 6

0	1	2	3	4	5	6
-	a	k	l	m	o	t

0	1	2	3	4	5	6
a	-	k	l	m	o	t

0	1	2	3	4	5	6
a	-	k	l	m	o	t

0	1	2	3	4	5	6
m	a	-	k	l	o	t

0	1	2	3	4	5	6
l	m	a	-	k	o	t

0	1	2	3	4	5	6
t	l	m	a	-	k	o

0	1	2	3	4	5	6
a	t	l	m	-	k	o

0	1	2	3	4	5	6
-	a	t	l	m	k	o

0	1	2	3	4	5	6
a	-	t	l	m	k	o

0	1	2	3	4	5	6
-	a	t	l	m	k	o

0	1	2	3	4	5	6
k	-	a	t	l	m	o

0	1	2	3	4	5	6
o	k	-	a	t	l	m

Przykład transformacji

- Dla ciągu a a m l t a - a - k o entropia wynosi $2,550$.
- Dla ciągu 1 0 4 4 6 3 4 1 1 5 6 entropia wynosi $2,413$.
- Dla długich ciągów zyski mogą być dużo większe i kodowanie Huffmana bardziej efektywne.