Kompresja danych

Implementacja BWT oraz algorytmów dodatkowych

Jakub Machoń, Łukasz Banaśkiewicz, Kacper Szkudlarek

7 stycznia 2011

Streszczenie

Transformata Burrowsa-Wheelera to algorytm użyteczny przy bezstratnej kompresji danych. Dane po przetworzeniu tą transformacją dają się znacznie lepiej skompresować za pomocą klasycznych algorytmów kompresji. W ramach projektu powstała implementacja transformaty oraz kilku algorytmów drugiego kroku wykorzystywanych razem z BWT.

1 Wstęp

2 Implementacja

2.1 Transformata Burrowsa-Wheelera

Transformata Burrowsa-Wheelera nie jest metodą kompresji, a jedynie sposobem modyfikacji danych (położenia poszczególnych bajtów). Po modyfikacji dane wyjściowe zawierają zazwyczaj ciągi równych sobie bajtów umieszczonych po sobie. Tak ułożone dane lepiej poddają się kompresji. Algorytm transformaty Burrowsa-Wheelera zaimplementowano zgodnie z opisem w [1] (książce PANA PRZELAWSKOWSKIEGO TODO PRZYPIS).

Strumień danych wejściowych dzielony jest na bloki o rozmiarze będącym parametrem algorytmu. Dane w każdym bloku sortowane są metodą sortowania przedrostków (ang. *Suffix Sorting*) - opis algorytmu sortowania znajduje się w kolejny akapicie. Wynikiem sortowania jest macierz indeksów sortowanych bajtów. Dane wyjściowe tworzone są poprzez przeglądanie posortowanej macierzy indeksów - jako wynik zastosowania transformaty należy zwrócić dane skonstruowane poprzez kolejne wypisywanie bajtów z pozycji *i-1*, gdzie *i* jest wartością zapisaną w macierzy indeksów. Dla *i=0* należy wypisać ostatni bajt z bloku. Do tak wypisanych danych należy dołączyć liczbę wskazującą pozycję na której znalazł się bajt o indeksie 1 z bloku wejściowego.

W implementacji Transformaty Burrowsa-Wheelera najbardziej problematycznym krokiem algorytmu jest sortowanie bajtów metodą sortowania przedrostków. Krok ten jest problematyczny ze względu na swoją złożoność czasową. Algorytm sortowania spowodował również najwięcej problemów podczas implementacji transformaty. W projekcie

Rysunek 2: Przykład - sposób wstawiania bajtów typu X do posortowanej tablicy. W pogrubionej ramce znajduje się element powodujący wstawienie pogrubionego elementu typu X.

po dokonaniu przeglądów algorytmów sortowania przedrostkowego zdecydowano się zaimplementować algorytm Itoh'a (wg opisu w [2] (thesis.ps)).
Algorytm sortowania składa się z 3 kroków:

1. W pierwszym kroku należy podzielić dane na dwa typy: X i Y. Wykonujemy to przeglądając bajty w bloku i jeśli obecnie analizowany bajt jest leksykograficznie po następnym bajcie, wtedy zaznaczamy go jako należący do X. W przeciwnym przypadku jest on typu Y. Ostatni bajt z bloku należy porównywać z pierwszym bajtem tego samego bloku. (Rys. 1)

2. W drugim kroku sortujemy dane przez zliczanie, umieszczając w tablicy najpierw dane typu X, a później typu Y. Dane typu Y sortujemy zmodyfikowanym algorytmem Radix Sort, jeśli dla danego symbolu jest więcej elementów typu Y, niż 1. Modyfikacja algorytmu Radix Sort polega na tym, iż rozpoczyna on analizowanie danych od najbardziej znaczących bajtów, a nie tak jak w swej standardowej wersji - on najmniej znaczących. Dzięki temu poza przypadkami długich ciągów takich samych bajtów nie ma potrzeby analizowania całego bloku dla każdego sortowanego bajtu. Algorytm zaimplementowano iteracyjnie - z użyciem stosu umieszczonego na stercie.

3. W trzecim kroku następuje łączenie koszyków w następujący sposób - przeglądamy częściowo posortowany blok wejściowy (posortowane są tylko dane typu Y). Jeśli dla i będącego pozycją elementu z posortowanego bloku w pierwotnym bloku (wejściowym), bajt na pozycji *i-1* jest typu X, to należy go umieścić na pozycji, na której powinien się znajdować ze względu na swoją wartość i przesunąć wskaźnik dla tej wartości.

Niewątpliwą zaletą zastosowanego algorytmu jest fakt, iż występujące w kroku 3 łączenie koszyków ma złożoność liniową, co oznacza, że bajty z koszyka X zostały posortowane w czasie liniowym. Pewną wadą przyjętego rozwiązania jest to, iż w zależności od charakteru danych, spora ich część trafia do koszyka Y, który jest już sortowany algorytmem Radix Sort - podatnym na głęboką rekurencję.

Podczas dekodowania danych (transformata odwrotna - przywracająca pierwotne ułożenie bajtów w bloku) stosowane jest sortowanie przez zliczanie - ponieważ podczas dekodowania wystarczy posortować bajty leksykograficznie (ściślej: tworzona jest tablica indeksów wskazujących na posortowane dane). W związku z tym dekodowanie danych jest dużo szybsze, niż kodowanie - algorytm sortowania przez zliczanie ma złożoność liniową w stosunku do długości bloku. Pozostałe działania wykonywane podczas dekodowania również posiadają liniową złożoność.

Dane:	S	V	E	E	A	E	E	V
Typ:	Y	X	Y	X	Y	Y	Y	X
Indeks:	0	1	2	3	4	5	6	7

Rysunek 1: Przykład - podział danych wejściowych na typy

W transformacie bardzo ważny jest algorytm sortowania, ponieważ błędne posortowanie danych powoduje przekłamania w odkodowanych danych.

2.2 RLE0 i RLE2

2.3 Increment Frequency Count

2.4 Distance Codding

2.5 Move To Front

2.6 Invesrion Frequencies

3 Testy

3.1 Aplikacja testująca

3.2 Testy wydajności

3.2.1 Dane testowe

3.2.2 Wyniki