Run-Length Huffman - alternatywny algorytm kompresji map bitowych

Michał Stabno[1], **Robert Wrembel**[2]

Streszczenie: Artykuł prezentuje nowy algorytm kompresji indeksów bitmapowych dla zastosowań w hurtowniach danych. Algorytm ten, o nazwie *Run-Length Huffman* (RLH), bazuje na dwóch technikach kompresji, mianowicie na kodowaniu run-length i kodowaniu Huffmana. W artykule omówiono koncepcję algorytmu RLH i jego ocenę eksperymentalną.

Słowa kluczowe: kompresja indeksów bitmapowych, BBC, WAH, RLH

1. Wprowadzenie

Indeks bitmapowy (ang. bitmap index) (O'Neil, Quass, 1997; Davis, Gupta, 2007) jest jedną z podstawowych struktur wykorzystywanych w optymalizacji zapytań analitycznych. Rozmiar indeksu bitmapowego silnie zależy od dziedziny indeksowanego atrybutu. Dla dziedzin szerokich i ciągłych indeksy bitmapowe osiągają ogromne rozmiary. Konsekwencją ogromnego rozmiaru indeksu bitmapowego jest gwałtowny spadek efektywności wyszukiwania danych z wykorzystaniem takiego indeksu. Z tego względu, w praktyce stosuje się kompresje indeksów bitmapowych. W literaturze naukowej zaproponowano dwie ogólnie uznane metody kompresji indeksów bitmapowych, tj. BBC (ang. Byte-aligned Bitmap Compression) (Antoshenkov, Ziauddin, 1996) i WAH (ang. Word-Aligned Hybrid) (Wu, Otoo, Shoshani, 2004A).

Kompresja BBC opiera się o tzw. kodowanie *run-length*, w którym zliczaniu podlegają jednorodne ciągi zer i jedynek. Mapa bitowa jest dzielona na 8-bitowe słowa, które są grupowane albo w tzw. wypełnienia (bajty złożone z samych zer lub jedynek) albo w dopełnienia (bajty złożone z zer i jedynek). Kompresji podelgają wypełnienia. Rozwinięciem BBC jest kompresja WAH, która również opiera się o kodowanie run-length, ale sekwencja bitów jest dzielona na słowa 31-bitowe. BBC i WAH zapewniają dobry współczynnik kompresji dla map bitowych opisujących rekordy uporządkowane zgodnie z wartością indeksowanego atrybutu. Jeśli warunek ten nie jest spełniony, wówczas współczynnik kompresji jest gorszy. Przy dość wysokiej gęstości występowania jedynek w mapach bitowych i w miarę równomiernym losowym ich rozmieszczeniu, prawdopodobieństwo wystąpienia jednorodnych ciągów bitów dłuższych niż $n * 8$ bitów dla BBC i $n * 31$ bitów dla WAH jest małe.

[1] QXL Poland Sp. z o. o. ul. Marcelińska 90 60-324 Poznań;
Politechnika Poznańska, Instytut Informatyki Piotrowo 2, 60-965 Poznań
e-mail: Michal.Stabno@allegro.pl

[2] Politechnika Poznańska, Instytut Informatyki Piotrowo 2, 60-965 Poznań
e-mail: Robert.Wrembel@cs.put.poznan.pl

Oznacza to, że niewiele słów można zapisać jako wypełnienia. W konsekwencji, współczynnik kompresji jest mały. Jest to istotną wadą kompresji BBC i WAH.

W celu wyeliminowania tej wady, w niniejszym artykule proponujemy nową technkę kompresji o nazwie *Run-Length Huffman* (RLH). Podobnie jak BBC i WAH, proponowana przez nas technika wykorzystuje kodowanie run-length w powiązaniu z algorytmem kodowania Huffmana (Huffman). W algorytmie RLH zliczaniu podlegają odległości pomiędzy kolejnymi bitami o wartości jeden, a nie długości jednorodnych ciągów bitów o tej samej wartości. Algorytm RLH został zaimplementowany i porównany eksperymentalnie z algorytmem WAH. Jako odniesienie wybrano WAH jako bardziej efektywny niż BBC z punktu widzenia dostępu do danych za pomocą skompresowanych map bitowych (Wu, Otoo, Shoshani, 2002).

2. Definicje podstawowe

2.1. Indeks bitmapowy

Indeks bitmapowy bazuje na tzw. mapach bitowych umożliwiających znalezienie rekordów z zadaną wartością indeksowanego atrybutu. Mapa bitowa jest wektorem bitów. Każda wartość z dziedziny indeksowanego atrybutu posiada odrębną mapę. Liczba bitów każdej mapy jest identyczna i odpowiada liczbie rekordów indeksowanej tabeli. Mapa bitowa utworzona dla wartości w indeksowanego atrybutu A opisuje te rekordy, których wartością atrybutu A jest w. Bit o numerze n przyjmuje wartość 1 w mapie, jeśli atrybut A rekordu o numerze n przyjmuje wartość w. W przeciwnym przypadku bit n przyjmuje wartość 0. Poniższy przykład ilustruje koncepcję indeksu bitmapowego.

Przykład 1. *Rozważmy tabelę Klienci, której fragment przedstawiono poniżej. Na atrybucie płeć zdefiniowano indeks bitmapowy, przedstawiony w tabeli 1. Ponieważ dziedziną indeksowanego atrybutu płeć są dwie wartości, tj. 'kobieta' i 'mężczyzna', więc indeks bitmapowy składa się z dwóch map bitowych. Bit pierwszy mapy opisującej wartości 'kobieta' ma wartość 0 ponieważ wartością atrybutu płeć dla pierwszego rekordu nie jest 'kobieta', itp.*

□

Jak wspomniano wcześniej, rozmiar indeksu bitmapowego wzrasta wraz ze wzrostem dziedziny indeksowanego atrybutu. Przy szerokich dziedzinach indeksowanego atrybutu, poszczególne mapy bitowe są rzadkie, tj. liczba jedynek w mapach bitowych w stosunku do liczby zer jest mała. Tego typu mapy bitowe bardzo dobrze się kompresują. W literaturze naukowej zaproponowano dwie ogólnie uznane i stosowane w praktyce metody kompresji indeksów bitmapowych, tj. BBC i WAH.

2.2. Kompresja BBC i WAH

Jak wspomniano, kompresje BBC i WAH są oparte o tzw. kodowanie run-length. W kodowaniu tym, kompresowana mapa bitowa jest dzielona na słowa. Słowo złożone z samych zer lub z samych jedynek jest tzw. wypełnieniem. Słowo złożone z zer i jedynek jest tzw. dopełnieniem. Wypełnienia są kompresowane. Format

Tabela 1. Przykładowa tabela *Klienci* i indeks bitmapowy na atrybucie *płeć*

Klienci

ID	płeć
1	mężczyzna
2	kobieta
3	kobieta
4	kobieta
5	mężczyzna
6	mężczyzna
7	mężczyzna
8	kobieta
9	kobieta
10	mężczyzna
11	mężczyzna
12	mężczyzna
13	kobieta
14	kobieta
15	kobieta
16	mężczyzna
17	kobieta
18	kobieta
19	kobieta

indeks bitmapowy

kobieta	mężczyzna
0	1
1	0
1	0
1	0
0	1
0	1
0	1
1	0
1	0
0	1
0	1
0	1
1	0
1	0
1	0
0	1
1	0
1	0
1	0

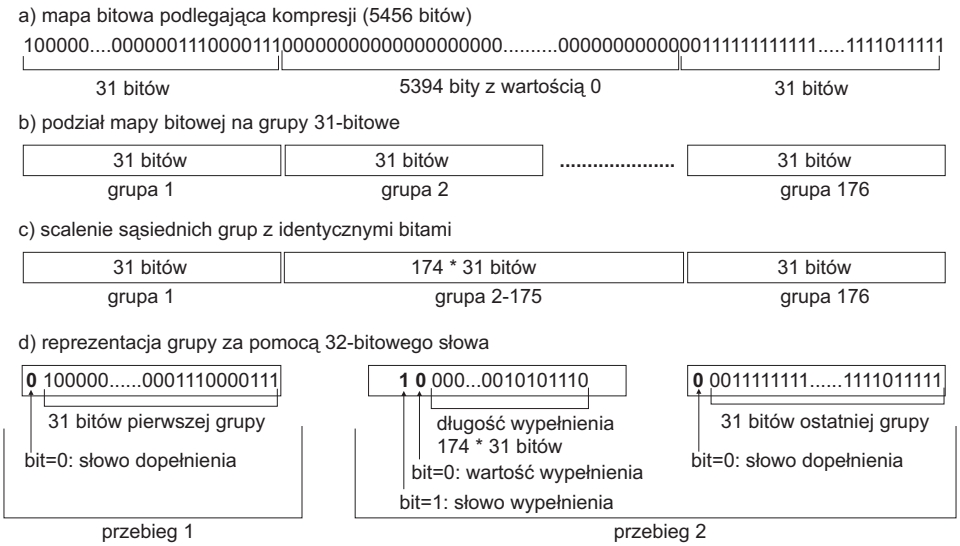
skompresowanego wypełnienia jest następujący: bit pierwszy z wartością 1 oznacza wypełnienie; bit drugi oznacza wartość wypełnienia, tj. wartość jaką miały bity oryginalnego wypełnienia; kolejne bity służą do zapisania długości wypełnienia, tj. liczby jednorodnych zer lub jedynek. Dopełnienia, ze względu na swoją niejednorodną zawartość, nie są kompresowane. W ogólności, podział mapy bitowej na słowa pogarsza jakość jej kompresji ponieważ jednorodne ciągi bitów nie zawsze pasują do długości słowa i w takim przypadku nie mogą być kompresowane. Ogólną ideę kompresji WAH przedstawiono na poniższym przykładzie, zaczerpniętym z (Stockinger, Wu, 2007).

Przykład 2. *Dla uproszczenia, przyjmijmy, że procesor używa 32-bitowe słowa, a mapa bitowa jest złożona z 5456 bitów, jak pokazano na rysunku 1a.*

***Krok 1**. Mapa bitowa jest dzielona na grupy złożone z 31 bitów, jak pokazano na rysunku 1b. W przykładzie grup takich jest 176.*

***Krok 2**. Sąsiednie grupy, zawierające identyczne bity, są scalane w jedną grupę. Pokazano to na rysunku 1c. Ponieważ grupa 1 nie jest jednorodna, składa się z bitów o wartości 0 i bitów o wartości 1, więc nie jest ona scalana z następującą po niej grupą. Grupy 2 do 175 są jednorodne - składają się z bitów o wartości 0. Grupy te są więc scalane w jedną grupę (oznaczoną jako grupa 2-175) zawierającą 174 * 31 bitów. Grupa 176, podobnie jak grupa 1, jest niejednorodna i nie może być scalona z grupą ją poprzedzającą. W wyniku scalania powstają trzy grupy, jak pokazano na rysunku 1c.*

***Krok 3**. Te trzy grupy są kodowane w 32 bitowych słowach w sposób następujący. Grupa pierwsza reprezentuje dopełnienie przebiegu 1. Najstarszy bit tej grupy (pierwszy z lewej) przyjmuje wartość 0, co oznacza, że grupa jest dopełnieniem. Kolejne 31 bitów to oryginalne bity grupy 1. Grupa 2-175 reprezentuje*



Rysunek 1. Kroki kompresji WAH

wypełnienie przebiegu 2. Najstarszy bit tej grupy przyjmuje wartość 1 co oznacza, że grupa jest wypełnieniem. Bit na pozycji 2^{30} *przyjmuje wartość 0 co oznacza, że wszystkie bity oryginalnej grupy przyjmują wartość 0. Na kolejnych 30 bitach jest zapisywana liczba jednorodnych grup z bitami o wartości 0. W przykładzie grup tych jest 174. Zapisana binarnie liczba 174 ma postać: 10101110. Grupa 176 reprezentuje dopełnienie przebiegu 2. Najstarszy bit tej grupy przyjmuje wartość 0, co oznacza, że grupa jest dopełnieniem. Kolejne 31 bitów to oryginalne bity grupy 176.*

□

2.3. Kodowanie Huffmana

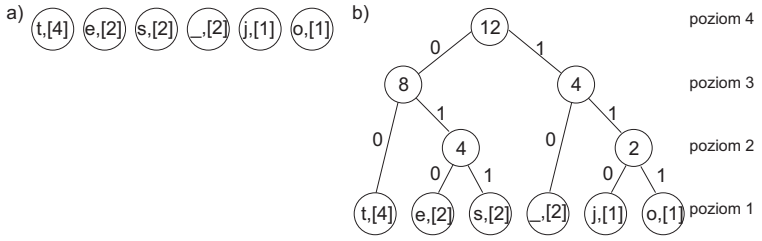
W kodowaniu Huffmana (ang. Huffman coding), oryginalne symbole są zastępowane ciągami bitów, przy czym symbole występujące częściej są zastępowane krótszymi ciągami bitów. Algorytm jest realizowany w trzech krokach, zilustrowanych poniższym przykładem.

Przykład 3. *Rozważmy ciąg znaków 'to_jest_test', który zostanie skompresowany z wykorzystaniem kodowania Huffmana.*

Krok 1*. Dla każdego symbolu z kompresowanego ciągu jest obliczana częstość występowania symbolu. Dla podanego tekstu, częstości poszczególnych liter zostały przedstawione w tabeli 2. Następnie, na podstawie częstości jest tworzony zbiór węzłów postaci: symbol, częstość jego występowania (podana w nawiasach graniastych), jak pokazano na rysunku 2a.*

Tabela 2. Częstości występowania symboli w kompresowanym ciągu 'to_jest_test'

symbol	t	e	s	_	j	o
częstość	4	2	2	2	1	1



Rysunek 2. Budowanie drzewa kodów Huffmana

Krok 2. *Drzewo kodów Huffmana jest budowane w sposób następujący. Dwa dowolnie wybrane węzły z poziomu 1 reprezentujące symbole o najmniejszych częstościach są łączone w jeden węzeł nadrzędny na poziomie 2. W naszym przykładzie, jako pierwsze zostały połączone węzły (j,[1]) i (o,[1]), tworząc węzeł (2), jak pokazano na rysunku 2b. Wartość nowego węzła jest sumą częstości węzłów, które on łączy. Następnie, w podobny sposób są łączone kolejne dwa węzły o najniższych częstościach. Węzły są łączone do momentu uzyskania jednego węzła na poziomie najwyższym.*

Krok 3. *Łuki łączące węzły drzewa kodów Huffmana otrzymują wartości 0 lub 1. Przyjmuje się, że łuki wychodzące na lewo z węzła otrzymują wartości 0, a łuki wychodzące na prawo otrzymują wartości 1. W ten sposób, kodowany symbol z poziomu najniższego jest reprezentowany ścieżką dostępu (tzw. kodem Huffmana) do niego z korzenia drzewa. Kody Huffmana dla symboli z kompresowanego ciągu przedstawiono w tabeli 3a. Następnie, wszystkie symbole z kompresowanego ciągu są zastępowane kodami Huffmana, jak pokazano w tabeli 3b.*

Tabela 3. Kodowane symbole i odpowiadające im kody Huffmana

a)

kodowany symbol	t	e	s	_	j	o
kod Huffmana	00	010	011	10	110	111

b)

t	o	_	j	e	s	t	_	t	e	s	t
00	111	10	110	010	011	00	10	00	010	011	00

□

3. Kompresja RLH

Zaproponowany w niniejszym artykule algorytm kompresji, zwany *Run-Length Huffman* (RLH) łączy w sobie dwie techniki: kodowanie run-length i kodowanie Huffmana.

W kodowaniu run-length, kodowaniu podlegają jednorodne ciągi zer lub jedynek. W algorytmie RLH, kodowaniu podlegają *odległości* pomiędzy kolejnymi bitami o wartości jeden. Przykładowo, ciąg `000011110100` jest reprezentowany w algorytmie RLH kodem 400012 (przy założeniu, że początek i koniec ciągu są interpretowane tak jak bit o wartości jeden). Niniejszy przykładowy kod należy interpretować następująco, analizując go od lewej strony. Pierwsza jawnie występująca jedynka w ciągu jest odległa o 4 pozycje (bity) od niejawnej jedynki rozpoczynającej ciąg; druga jawna jedynka jest odległa od pierwszej o 0 pozycji; trzecia jawna jedynka jest odległa od trzeciej o 0 pozycji, itp. Takie rozwiązanie zapewnia, że wraz ze zmniejszaniem się gęstości map bitowych zmniejsza się liczba symboli wykorzystywanych do kodowania tych map.

Mapy bitowe, zakodowane z wykorzystaniem zmodyfikowanego kodowania run-length są następnie kompresowane z wykorzystaniem algorytmu kodowania Huffmana. Danymi wejściowymi dla tego algorytmu są częstości odległości we wszystkich mapach bitowych zakodowanych z użyciem zmodyfikowanego kodowania run-length. Na podstawie tych częstości zostaje utworzone drzewo kodów Huffmana, które służy do dalszego kodowania odległości.

Mapy bitowe skompresowane za pomocą algorytmu RLH są przechowywane na dysku. Drzewo kodów Huffmana jest zapisywane na dysku w oddzielnej strukturze. Rozmiar drzewa kodów Huffmana jest niewielki, dzięki czemu może ono być przechowywane w pamięci RAM, zwiększając efektywność kompresowania i dekompresowania map bitowych.

W celu zilustrowania działania algorytmu RLH, rozważmy przykład.

Przykład 4. *Rozważmy tabelę Klienci z przykładu 1. Kompresja obu map bitowych z wykorzystaniem algorytmu RLH przebiega następująco.*

***Krok 1**. Mapy bitowe są kodowane za pomocą zmodyfikowanego kodowania run-length. Kolejne symbole oznaczają odległości pomiędzy kolejnymi jedynkami w mapach bitowych. Wynik zmodyfikowanego kodowania run-length przedstawiono w tabeli 4a.*

***Krok 2**. Odległości otrzymane w kroku 1 są kodowane za pomocą kodowania Huffmana. Najpierw są zliczane częstości występowania poszczególnych symboli (por. tabela 4b). Są one danymi wejściowymi dla algorytmu Huffmana. Następnie jest budowane drzewo kodów Huffmana i na jego podstawie są wyznaczane kody dla poszczególnych symboli. Drzewo kodów Huffmana dla symboli z tabeli 4b zostało przedstawione na rysunku 3. Symbole i odpowiadające im kody Huffmana przedstawiono w tabeli 5.*

W wyniku działania algorytmu RLH powstają skompresowane mapy bitowe, jak pokazano na rysunku 4.

□

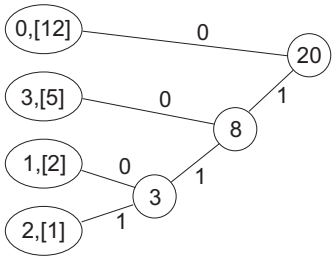
Tabela 4. Zmodyfikowane kodowanie run-length

a) zmodyfikowane kodowanie run-length

kobieta	mężczyzna
1	0
0	3
0	0
3	0
0	2
3	0
0	0
0	3
1	3
0	
0	

b) częstości symboli

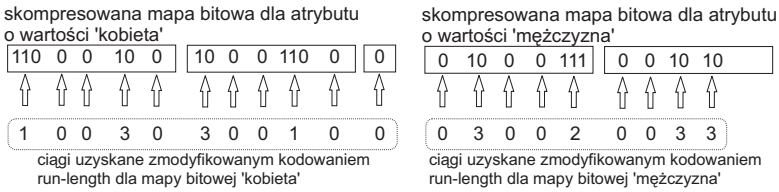
symbol	częstość
0	12
3	5
1	2
2	1



Rysunek 3. Drzewo kodów Huffmana dla symboli z tabeli 4b

Tabela 5. Symbole z tabeli 4b i odpowiadające im kody Huffmana

symbol	kod symbolu
0	0
3	10
1	110
2	111



Rysunek 4. Mapy bitowe skompresowane algorytmem RLH

Rozmiar drzewa kodów Huffmana wpływa na efektywność algorytmu dekompresji i przetwarzania map bitowych skompresowanych algorytmem RLH. Z przeprowadzonych eksperymentów wynika, że w praktyce, rozmiar ten jest mały. Przykładowo, dla testowej tabeli zawierającej 2 miliony rekordów i liczności dziedziny indeksowanego atrybutu równej 1000, rozmiar drzewa kodów Huffmana wynosił zaledwie 65 KB. Dzięki małemu rozmiarowi, drzewa kodów Huffmana mogą być przechowywane w pamięci RAM, zwiększając efektywność kodowania i dekodowania map bitowych.

3.1. Dekompresowanie map bitowych

Dekompresowanie map bitowych wykorzystuje drzewo kodów Huffmana. W tym celu, kolejne bity skompresowanej mapy bitowej są odczytywane przez algorytm dekompresji. Bity te służą do nawigowania w drzewie kodów Huffmana od korzenia do liścia. Odczytany z liścia kod Huffmana zastępuje bity, które posłużyły do nawigowania do tego liścia.

Dekompresowanie map bitowych skompresowanych algorytmem RLH wymaga większej liczby operacji niż w przypadku map bitowych skompresowanych algorytmem WAH, czy BBC. Pomimo tego, efektywność dekompresji jest duża ponieważ drzewo kodów Huffmana jest na stałe przechowywane w pamięci RAM.

3.2. Modyfikowanie map bitowych

Proces modyfikowania map bitowych skompresowanych algorytmem RLH jest bardziej złożony niż dla kompresji WAH i BBC. W kompresji WAH, zmodyfikowanie skompresowanej mapy bitowej sprowadza się do zmodyfikowania jednej lub dwóch wartości typu *integer* w poszczególnych mapach bitowych, bez konieczności dekompresowania całych map bitowych. W przypadku kompresji RLH należy: (1) zdekompresować całą mapę bitową, (2) zmodyfikować odpowiednie bity mapy, (3) ponownie skompresować mapę bitową. Operacje te są konieczne ze względu na zmiany w częstościach występowania symboli w mapach bitowych, w zmodyfikowanym kodowaniu run-length. Używając starego drzewa kodów Huffmana, powstawałyby mapy bitowe skompresowane w sposób nieoptymalny. Dodatkowo, modyfikacja danych mogłaby spowodować pojawienie się nowych symboli (odległości pomiędzy jedynkami) nieujętych w starym drzewie kodów Huffmana.

Problemy podobnego typu są znane w literaturze naukowej. Rozwiązuje się je częściowo za pomocą dynamicznych algorytmów Huffmana (Vitter, 1989). Algorytmy te są jednak zbyt złożone obliczeniowo dla zastosowań w hurtowniach danych.

W hurtowniach danych, uaktualnianie struktur indeksowych ma miejsce w końcowej fazie procesu odświeżania hurtowni. W praktyce, przed wczytaniem danych indeksy są usuwane. Po wczytaniu danych indeksy są tworzone od początku. Z tego względu, wada kompresji RLH, jaką jest złożony proces modyfikowania skompresowanych map bitowych, staje się mniej istotna w zastosowaniach hurtowni danych. W tym bowiem przypadku, skompresowany indeks bitmapowy będzie tworzony od początku po każdym odświeżeniu hurtowni.

4. Eksperymentalna ocena algorytmu RLH

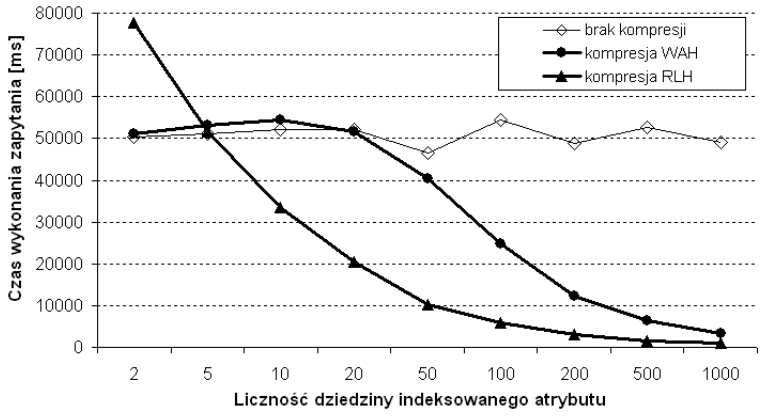
Algorytm kompresji RLH został zaimplementowany i oceniony eksperymentalnie. Charakterystykę kompresji RLH odniesiono do kompresji WAH ponieważ, jak już wcześniej wspomniano, WAH oferuje większą niż BBC efektywność przetwarzania map bitowych i dostępu do danych za ich pomocą (Wu, Otoo, Shoshani, 2002). Eksperymenty zostały przeprowadzone na komputerze PC z procesorem AMD Athlon(TM) XP 2500+, dyskiem Seagate 80 GB i pamięcią RAM 768 MB, pracującym pod kontrolą Windows XP.

Testy przeprowadzono na tabeli zawierającej 2000000 rekordów. Indeks bitmapowy zdefiniowano na atrybucie typu *integer* o parametryzowanej liczności dziedziny wynoszącej 2, 5, 10, 20, 50, 100, 200, 500 i 1000 różnych wartości. Przyjęto losowy rozkład wartości indeksowanego atrybutu. W ramach testów porównywano kompresję RLH z WAH pod kątem: (1) czas odpowiedzi na zapytanie (scenariusz 1), (2) rozmiaru skompresowanych map bitowych (scenariusz 2).

4.1. Scenariusz 1: czas odpowiedzi na zapytanie

W tych eksperymentach mierzono czas odpowiedzi na zapytanie postaci: `select ... from ... where indeksowany_atrybut in (w1, w2, ..., w100)`, gdzie $w1, w2, \ldots, w100$ reprezentują losowo wybrane wartości indeksowanego atrybutu.

Powyższe zapytanie kierowano do tabeli dla trzech następujących przypadków: (1) na indeksowanym atrybucie zdefiniowano standardowy nieskompresowany indeks bitmapowy, (2) na indeksowanym atrybucie zdefiniowano indeks bitmapowy skompresowany algorytmem WAH, (3) na indeksowanym atrybucie zdefiniowano indeks bitmapowy skompresowany algorytmem RLH. Wyniki eksperymentów przedstawiono na rysunku 5.



Rysunek 5. Czas odpowiedzi na zapytanie (liczba indeksowanych rekordów: 2000000; liczność dziedziny indeksowanego atrybutu: 1-1000)

Jak widać z wykresu, kompresja RLH zapewnia większą efektywność dostępu

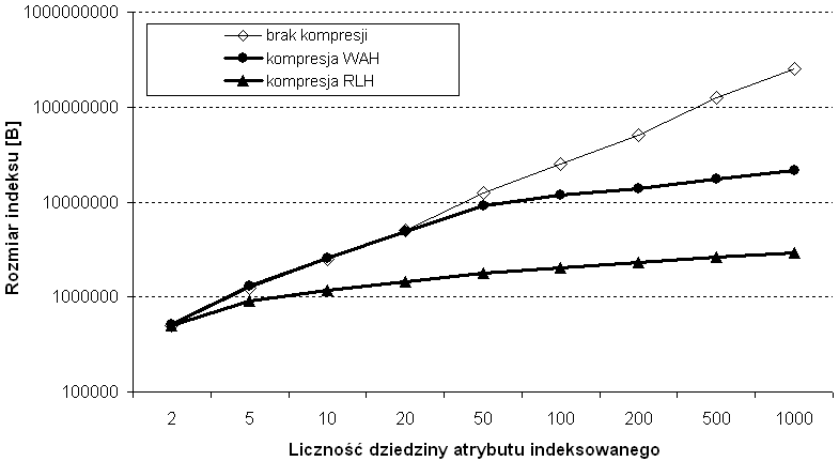
do danych dla liczności dziedziny indeksowanego atrybutu w zakresie 5-1000. Taka charakterystyka jest spowodowana mniejszą gęstością bitów o wartości jeden w mapach bitowych dla dziedzin indeksowanego atrybutu o większej liczności. W konsekwencji, do kodowania odległości pomiędzy sąsiednimi jedynkami jest wykorzystywanych mniej symboli, co z kolei powoduje, że rozmiar drzewa kodów Huffmana jest mniejszy. Mniejszy rozmiar drzewa kodów Huffmana powoduje, że kompresowanie i dekompresowanie map bitowych jest szybsze niż dla dużych drzew kodów Huffmana.

Wraz ze wzrostem liczności dziedziny indeksowanego atrybutu, począwszy od wartości 70-80, różnica w czasach dostępu do danych z wykorzystaniem indeksów skompresowanych WAH i RLH maleje. Dzieje się tak dlatego, ponieważ wraz ze wzrostem liczności dziedziny atrybutu średnie odległości pomiędzy kolejnymi jedynkami stają się dłuższe niż 31-bitów. W takim przypadku, dla WAH, mapa bitowa zawiera o wiele więcej słów (wypełnień) złożonych z bitów o wartości 0. Mapy bitowe o takiej charakterystyce, poddane kompresji WAH zapewniają większą efektywność przetwarzania i dostępu do danych. Efektywność ta wzrasta wraz z wydłużaniem się jednorodnych ciągów bitów, które wchodzą w skład wypełnień. Z tego względu, dla atrybutu o liczności dziedziny powyżej 1000 różnica w charakterystyce przetwarzania map bitowych poddanych kompresji WAH i RLH staje się niewielka.

Dla liczności dziedziny indeksowanego atrybutu mniejszej niż 5, RLH oferuje mniejszą efektywność niż WAH i nieskompresowany indeks bitmapowy. Taka charakterystyka może być wytłumaczona następująco. Dla dziedzin o niewielkiej liczności, mapy bitowe charakteryzuje duża gęstość bitów o wartości jeden. Powoduje to, że średnie odległości pomiędzy kolejnymi jedynkami są niewielkie i odległości tych jest dużo. W konsekwencji, mapy bitowe poddane kompresji RLH mają większe rozmiary, co przekłada się na dłuższe czasy odczytu skompresowanego indeksu z dysku. Ponadto, rozmiar drzewa kodów Huffmana wzrasta wraz ze zmniejszaniem się liczności dziedziny indeksowanego atrybutu. W konsekwencji, operacje logiczne dokonywane w pamięci są bardziej złożone w przypadku RLH niż operacje przeprowadzane na mapach bitowych skompresowanych z użyciem WAH i na nieskompresowanych mapach bitowych. W przeprowadzonym eksperymencie, poszczególne wartości indeksowanego atrybutu były nadawane losowo wybranym rekordom tabeli, tj. rozkład jedynek w poszczególnych mapach był nieuporządkowany. W przypadku map bitowych, w których rozkład jedynek jest przynajmniej częściowo uporządkowany, można uzyskać lepszy stopień kompresji RLH niż kompresji WAH, a co za tym idzie krótszy czas dostępu do danych nawet dla wąskich dziedzin indeksowanego atrybutu. W celu potwierdzenia tej tezy przeprowadzono dodatkowy eksperyment na tabeli zawierającej 200000 rekordów i liczności dziedziny indeksowanego atrybutu równej 2, dla zapytania z warunkiem `WHERE indeksowany_atrybut = n`. Obie wartości z dziedziny były naprzemiennie przypisywane rekordom testowej tabeli, co dawało mapy bitowe postaci `...01010101010...`. W tym przypadku, mimo małej liczności dziedziny indeksowanego atrybutu, czasy odpowiedzi na zapytanie były najkrótsze dla kompresji RLH i wynosiły: (1) 36,56 ms (RLH), (2) 51,4 ms (WAH) i (3) 48,91 ms (bez kompresji).

4.2. Scenariusz 2: rozmiary indeksów bitmapowych

W tych eksperymentach mierzono rozmiary: (1) nieskompresowanego indeksu bitmapowego, (2) indeksu skompresowanego algorytmem WAH i (3) indeksu skompresowanego algorytmem RLH. Wyniki przedstawia rysunek 6.



Rysunek 6. Rozmiary indeksów (liczba indeksowanych rekordów: 2000000; liczność dziedziny indeksowanego atrybutu: 1-1000)

Analizując powyższy wykres można zauważyć, że wraz ze wzrostem liczności dziedziny indeksowanego atrybutu, rozmiary indeksów bitmapowych kompresowanych algorytmem RLH zwiększają się wolniej niż dla algorytmu WAH (na wykresie rozmiary indeksów są przedstawione w skali logarytmicznej). Dzieje się tak dlatego, że wraz ze wzrostem liczności dziedziny indeksowanego atrybutu wzrasta rozmiar jednorodnych ciągów zer w mapach bitowych. Ciągi takie dobrze kompresują się zarówno algorytmem WAH, jak i RLH. Ten ostatni, dla dziedzin o dużej liczności wymaga jednak mniej bitów niż WAH do zakodowania skompresowanej mapy.

5. Podsumowanie

W artykule został przedstawiony oryginalny algorytm RLH kompresji indeksów bitmapowych. Algorytm ten jest oparty o kodowanie run-length i o algorytm kodowania Huffmana. Zaproponowany algorytm został porównany eksperymentalnie z algorytmem WAH. Z przeprowadzonych eksperymentów wynikają trzy ważne konkluzje. Po pierwsze, algorytm RLH oferuje większą efektywność dostępu do danych niż WAH dla indeksowanych atrybutów o liczności dziedziny w zakresie 5-1000. Po drugie, dla atrybutów o liczności dziedziny mniejszej niż 5 kompresja RLH może zapewnić większą efektywność dostępu do danych niż WAH pod warunkiem uporządkowania rekordów według wartości indeksowanego atrybutu. Po trzecie, mapy

bitowe skompresowane algorytmem RLH są znacznie mniejsze niż odpowiadające im mapy skompresowane algorytmem WAH.

Literatura

Antoshenkov, G. and Ziauddin, M. (1996) Query Processing and Optimization in Oracle RDB. *VLDB Journal* **5**, 4, 229–237.

Davis, K. C. and Gupta, A. (2007) Indexing in Data Warehouses: Bitmaps and Beyond. *In Wrembel, R. and Koncilia, C.: Data Warehouses and OLAP: Concepts, Architectures and Solutions* Idea Group Inc., ISBN 1-59904-364-5.

Huffman Huffman Compression. *Retrieved May 8, 2007, from* `http://www.geocities.com/SiliconValley/2151/huffman.html`.

O'Neil, P. and Quass, D. (1997) Improved Query Performance with Variant Indexes. *Proc. of ACM SIGMOD Int. Conference on Management of Data*, 38–49.

Stockinger, K. and Wu, K. (2007) Bitmap Indices for Data Warehouses. *In Wrembel, R. and Koncilia, C.: Data Warehouses and OLAP: Concepts, Architectures and Solutions* Idea Group Inc., ISBN 1-59904-364-5.

Vitter, J. S. (1989) Dynamic Huffman Coding. *ACM Transactions on Mathematical Software* **15**, 2, 158–167.

Wu, K., Otoo, E. J., and Shoshani, A. (2002) Compressing Bitmap Indexes for Faster Search Operations. *Proc. of International Conference on Scientific and Statistical Database Management (SSDBM)*, 99–108.

Wu, K., Otoo, E. J., and Shoshani, A. (2004a) On the performance of bitmap indices for high cardinality attributes. *Proc. of Int. Conference on Very Large Data Bases (VLDB)*, 24–35.

Run-Length Huffman - alternative bitmap compression algorithm

In this paper we present a bitmap compression algorithm for application in data warehouses. The developed algorithm, called *Run-Length Huffman* (RLH), is based on the run-length encoding and on the Huffman encoding. RLH was implemented and experimentally compared to the well known Word Aligned Hybrid bitmap compression technique. Preliminary experimental results are also discussed in the paper.