Алгоритм оптимізації результатів розкладу LZ77 за рахунок мінімізації зміщень

Олександр Шпортько

Рівненський державний гуманітарний університет, Рівне, Україна
chportko@ukr.net, chportko@yandex.ru

Анотація

У цій статті пропонується алгоритм оптимізації результатів розкладу потоку словниковим алгоритмом LZ77 за рахунок мінімізації зміщень для покращення стиснення даних. Як показують експерименти, реалізація цього евристичного алгоритму дозволяє, наприклад, наблизити на 15% коефіцієнти стиснення швидких розкладів файлів зображень набору ACT у форматі PNG до майже оптимального розкладу, що вказує на його ефективність.

1. Вступ

На сьогодні, у зв'язку зі стрімким розвитком комунікаційних технологій та підвищенням інформаційних потреб суспільства, проблема стиснення даних не втрачає своєї актуальності. Адже компресія даних дозволяє пропорційно підвищити швидкість обміну інформацією по мережі та зменшити обсяги використання дискового простору. На цей час існує декілька основних методів та безліч алгоритмів стиснення даних [1]. Переважна більшість архіваторів та споріднених програм використовують алгоритми, що комбінують ідеї основних методів. Найчастіше серед них зустрічаються словникові алгоритми стиснення даних. Тому підвищення ефективності словникових алгоритмів дозволяє покращити характеристики більшості програм стиснення даних.

2. Принципи функціонування та основні стратегії розкладу LZ77

Існує два різновиди алгоритмів словникового методу. Перший з них бере початок від алгоритму LZ77 [2] і базується на заміні послідовності чергових елементів потоку посиланням на аналогічну закодовану послідовність елементів у вигляді пари чисел *<довжина; зміщення від закінчення закодованої частини потоку>*, якщо така послідовність зустрічалася раніше. Другий бере початок від алгоритму LZ78 [3] і базується на заміні послідовності чергових елементів потоку індексом аналогічної послідовності у словнику, що формується під час виконання алгоритму. Оскільки словникові алгоритми використовують опрацьовані раніше дані, то вони належать до класу контекстно-залежних.

Описуючи словникові алгоритми, фіксовану кількість попередніх закодованих елементів вхідного потоку називають словником, а наступних незакодованих – буфером. Розглянемо детальніше особливості стиснення словникового алгоритму LZ77 на основі найуживанішого *жадібного* розкладу вхідної послідовності [1]. Нехай послідовність елементів потоку $s_1 \dots s_{j-1}$ вже закодована і занесена в словник. Тоді на черговому кроці алгоритму для послідовності незакодованих елементів буфера $s_j s_{j+1} \dots$ шукається співпадаюча послідовність $s_i s_{i+1} \dots$ *максимальної* довжини *len(j)* (саме тому такий розклад називається *жадібним*), що бере початок у закодованій частині (словнику) послідовності *(i<j)*. При виявленні співпадаючої послідовності, $s_j \dots s_{j+len(j)-1}$ кодується парою чисел *<len(j); j-i>*, послідовність $s_j \dots s_{j+len(j)-1}$ заноситься в словник та виконується перехід до кодування потоку, починаючи з елемента $s_{j+len(j)}$. Очевидно, що для досягнення стиснення *len(j)* має перевищувати 2. Якщо ж співпадаюча послідовність в словнику відсутня, то елемент s_j заноситься в закодовані дані та словник без змін і виконується перехід до кодування потоку, починаючи з наступного елемента s_{j+1}. Кодування припиняється після опрацювання останнього елемента потоку. Співпадаюча послідовність може виходити за межі словника в область буфера, але має розпочинатися в словнику. Максимальні розміри словника та буфера встановлюються конкретними реалізаціями алгоритму. Сукупність словника з закодованими символами та буфера з незакодованими ще називають *ковзаючим вікном*, оскільки вони весь час синхронно переміщуються по елементах потоку. Наприклад, потік елементів *молмумолнмолмолн* в закодованому вигляді згідно *жадібного* розкладу LZ77 запишеться як *м о л м у <3; 5> н <4; 9> <3; 7>*. Покрокові результати дії алгоритму перед корегуванням словника та буфера для цього потоку відображено в табл. 1.

Таблиця 1: Покрокові результати стиснення послідовності *молмумолнмолмолн* згідно *жадібного* розкладу алгоритму LZ77

№ кроку	Ковзаюче вікно (словник | буфер)	Співпадаюча послідовність	Закодовані дані	
			<довжина; зміщення>	елемент
1.	| молмумолнмолмолн	-	-	м
2.	м | олмумолнмолмолн	-	-	о
3.	мо | лмумолнмолмолн	-	-	л
4.	мол | мумолнмолмолн	-	-	м
5.	молм | умолнмолмолн	-	-	у
6.	молму | молнмолмолн	мол	<3; 5>	-
7.	молмумол | нмолмолн	-	-	н
8.	молмумолн | молмолн	молм	<4; 9>	-
9.	молмумолнмолм | олн	олн	<3; 7>	-

На практиці, для отримання кращих показників стиснення даних результати дії словникового алгоритму додатково стискають одним з контекстно-незалежних алгоритмів (на основі кодів Хафмана, арифметичного стиснення чи інших). Саме такий підхід реалізовано, наприклад, у форматі стиснення даних DEFLATE [1], що

використовується в графічному растровому форматі PNG [4], де результати дії словникового алгоритму LZ77 стискаються контекстно-незалежними кодами Хафмана (ідея використання цих кодів полягає у заміні елементів з більшою частотою послідовностями меншої кількості біт, ніж для елементів з меншою частотою). Оскільки більші зміщення кодуються, як правило, більшою кількістю біт, то співпадаючі послідовності максимальної довжини шукають у словнику з кінця справа наліво і запам'ятовують найменші зміщення до них. Тому, якщо позначити через *offset(j; k)* найменше зміщення від позиції *j* до співпадаючої послідовності у словнику довжини *k* то, згідно попередніх викладок $i = j - offset(j; len(j))$. Коротші співпадаючі послідовності від послідовності максимальної довжини можуть зустрічатися у словнику і правіше, тому для $3 \le k < len(j)$

$$offset(j;k) \le offset(j;len(j)). \quad (1)$$

Під час декодування кодів алгоритму LZ77 окремі елементи копіюються у вихідний потік без змін. Пари ж <*довжина; зміщення*> декодуються шляхом послідовного копіювання з кінця вихідного потоку за вказаним зміщенням в кінець вихідного потоку необхідної кількості елементів. Природно, що алгоритм декодування має розрізняти окремі елементи та групи <*довжина; зміщення*>. Наприклад, у форматі DEFLATE з цією метою довжини замін та окремі елементи кодуються разом (ця модифікація алгоритму LZ77 має назву LZH). Після базових значень довжин міститься визначена форматом кількість біт, що разом з базовим значенням однозначно визначає довжину заміни. Зміщення зберігається після відповідної довжини аналогічно – у вигляді базового значення та додаткових біт. У форматі DEFLATE максимальне значення довжини закодованої послідовності може сягати 285, а зміщення – 32768. Коди Хафмана для елементів/довжин та зміщень визначаються для кожного блоку стиснутих даних окремо, що сприяє покращенню стиснення.

Розглянутий *жадібний* розклад алгоритму LZ77 забезпечує максимальну швидкість виконання, але не максимальне стиснення. Оскільки співпадаючих послідовностей у словнику може бути багато, то й розкладів LZ77 теж може бути чимало. Природно, що хотілося б мати оптимальну стратегію розкладу, яка б *завжди* забезпечувала мінімальну довжину закодованої послідовності. Але результати кодування LZ77 не використовуються самостійно а, як правило, кодуються контекстно-незалежним алгоритмом, тому вартість кодування окремих елементів та замін змінюється для різних вхідних послідовностей. Враховуючи цей факт, в [5] дану задачу було визнано *NP*-повною, тобто такою, що вимагає перебору всіх можливих варіантів для знаходження оптимального розв'язку. Тому для зменшення довжини закодованої послідовності був розроблений алгоритм майже оптимального розкладу [1]. Ідея цього алгоритму полягає в обчисленні мінімальної вартості (кількості біт) V_j кодування потоку від початку до кожного чергового елемента *j* включно. Згідно алгоритму,

$$V_j = \min_{l<j}\left(V_l + price_{l+1,j}\right),$$

де $price_{l+1,j}$ – мінімальна вартість зберігання пари <*довжина; зміщення*> для кодування елементів з *l+1* по *j*-й, якщо *l<j-1* або вартість кодування елемента s_j, якщо *l=j-1*. Після визначення мінімальної вартості кодування останнього елемента потоку відбираються позиції та зміщення, що дозволяють досягнути цієї мінімальної вартості, і саме з них формується закодований потік. Складається враження, що цей алгоритм описується за допомогою чотирьох вкладених циклів по елементах потоку: перший – для елементів *j*, стосовно яких визначається мінімальна вартість кодування, другий – для елементів *l*, згідно яких визначається мінімальна вартість кожного елемента, третій – по співпадаючих послідовностях, четвертий – по елементах цих послідовностей. Але якщо врахувати, що значення $price_{l+1,j}$ існують лише для тих елементів, між якими *можливе* кодування елементом s_j або парою <*довжина; зміщення*>, то зовнішній цикл можна відкинути, аналізуючи лише *можливі* вартості кодувань від кожного *l*-го символа до наступних.

Основним недоліком майже оптимального розкладу є низька швидкість кодування його реалізацій, що в декілька разів поступається швидкостям кодування реалізацій жадібного розкладу, адже майже оптимальний розклад виконує пошук співпадаючих послідовностей для *кожної*, а не для окремих позицій потоку. Ось чому на практиці використовуються стратегії, що забезпечують краще стиснення і незначно поступаються по часу кодування *жадібному* розкладу, тобто виконують пошук співпадаючої послідовності не для всіх позицій потоку. Найчастіше відійти від жадібного розкладу потоку намагаються за рахунок використання елементів замість замін там, де це доцільно (див. рис. 1), адже окремі елементи кодуються меншою кількістю біт, ніж пари <*довжина; зміщення*>.

Рисунок 1: Приклад оптимізації *жадібного* розкладу.

Для такої оптимізації, як правило, використовують розклад послідовності LZ77 з *лінивими* порівняннями [1]. Згідно алгоритму цього розкладу, у випадку виявлення співпадаючої послідовності в позиції *j* виконують пошук співпадаючої послідовності і з позиції *j+1*. Якщо *len(j+1)>len(j)* (див. рис. 1), то в позиції *j* кодують елемент і переходять до кодування з позиції *j+1*, інакше кодують віднайдену з цієї позиції заміну і переходять до позиції *j+len(j)*. Основними недоліками такого розкладу є можливість генерування послідовності декількох елементів замість однієї заміни та ймовірне виконання пошуку з позиції *j+2*, яке рідко покращує результати розкладу.

Зазначені недоліки можна усунути, використовуючи цікавий та маловідомий розклад Кадача [6]. Згідно алгоритму цього розкладу, у випадку виявлення

співпадаючої послідовності з позиції j виконують *синхронний* жадібний розклад з позицій j та $j+1$. Позначимо через a_l позиції жадібного розкладу, який починається з j, а через b_l – позиції жадібного розкладу, що починається з $j+1$ (див. рис. 2).

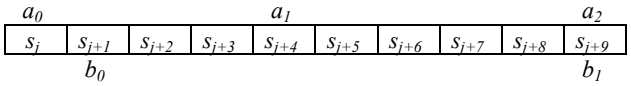

Рисунок 2: Позиції розкладу Кадача.

Тоді синхронний жадібний розклад продовжують до виконання однієї з двох умов ($k>0$):

$$a_k \geq b_k - 1 \tag{2}$$

або

$$b_{k-1} \geq a_k - 1 . \tag{3}$$

Якщо виконується умова (2), то в розклад Кадача включають всі заміни з позиції a_0 до позиції a_k. У випадку ж виконання умови (3) в розклад Кадача включають елемент s_j та заміни з позиції b_0 до позиції b_{k-1}. Оскільки розклад Кадача є ефективнішим від *лінивого* розклад як за критерієм стиснення так і за часом кодування, то саме його слід розглядати як основну альтернативу *жадібному* розкладу. Розклад Кадача генерує коротшу закодовану послідовність від *жадібного* розкладу, хоча й вимагає вдвічі більше пошуків співпадаючих послідовностей, що призводить до збільшення часу виконання.

3. Алгоритм мінімізації зміщень результатів розкладу LZ77.

Ми пропонуємо алгоритм оптимізації **результатів** розкладів LZ77, що використовують пошук співпадаючих послідовностей не зі всіх позицій потоку. Цей алгоритм не вимагає додаткових пошуків співпадаючих послідовностей, тому і не призводить до кардинального збільшення часу кодування відповідними програмами.

Розглянемо пару замін, що розташовуються підряд. Нехай довжина чергової заміни рівна m, а попередньої – n (див. верхню частину рис. 3).

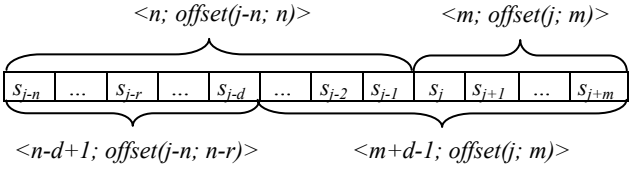

Рисунок 3: Приклад мінімізації зміщень результатів розкладу LZ77.

Визначимо максимальну кількість елементів r, на яку можна розширити чергову заміну за рахунок попередньої за тим самим зміщенням *offset(j; m)*, тобто

$$r = \max_{k=0,n-3,n-1} \left\{ k \mid \forall\, i = \overline{0,k}\; s_{j-i} = s_{j-offset(\,j;m\,)-i} \right\}. \tag{4}$$

Фактично для знаходження r порівняння елементів послідовності необхідно здійснювати не вперед, а назад. Подальший аналіз пари замін виконуємо за умови $r>0$. Якщо $r=n-1$, то замість попередньої заміни запишемо елемент s_{j-n} а чергову заміну розширимо до $m+r$ елементів, зменшивши тим самим кількість біт для кодування потоку. Інакше порівнюємо між собою зміщення попередньої заміни до та після можливого зменшення. Враховуючи (1), у випадку

$$offset(\,j-n;\,n-r\,) < offset(\,j-n;\,n\,) \tag{5}$$

(тобто **коротшу** співпадаючу послідовність вдається віднайти за **меншим** зміщенням) замість попередньої заміни запишемо *<n-d+1; offset(j-n; n-r)>*, де

$$d = \min_{k=0,r+1} \left\{ k \,\middle|\, offset(\,j-n;\,n-r\,) = offset(\,j-n;\,n-k+1\,) \right\}, \tag{6}$$

тобто $n-d+1$ – це максимальна довжина співпадаючої послідовності з початку попередньої заміни за мінімально допустимим зміщенням *offset(j-n; n-r)*. У цьому випадку активна заміна перезапишеться як *<m+d-1; offset(j; m)>* (див. нижню частину рис. 3). Виходячи з логіки описаного алгоритму, пари суміжних зміщень слід аналізувати у розкладі LZ77 з кінця до початку.

Мінімізуємо, наприклад, зміщення отриманого вище *жадібного* розкладу LZ77 *м о л м у <3; 5> н <4; 9> <3; 7>*. Суміжна пара зміщень зустрічається відразу в кінці цього розкладу. Останню заміну можна розширити за рахунок передостанньої на один елемент, тобто, згідно (4) *r=1*. Мінімальне зміщення для звуженої передостанньої заміни *мол* рівне *4*, що менше *9*, тобто вимога (5) виконується. Використовуючи (6) знаходимо, що *d=2* і після перезапису двох останніх замін, оскільки інші суміжні заміни відсутні, отримаємо розклад LZ77 *м о л м у <3; 5> н <3; 4> <4; 7>*. В цьому оптимізованому розкладі зменшено зміщення передостанньої заміни, тому він і кодується меншою кількістю біт. Структура розглянутих суміжних замін до та після оптимізації наведена на рис. 4. На ньому розширення наступної заміни за рахунок попередньої виділено сірим фоном. Мінімізація ж зміщень візуально проявляється у зменшенні довжини дуг замін.

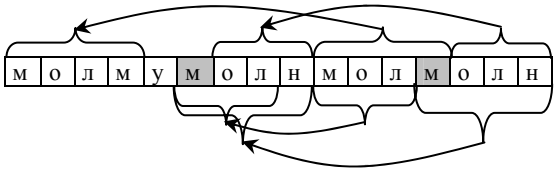

Рисунок 4: Суміжні заміни розкладу LZ77 потоку *молмумолнмолмолн* до (у верхній частині) та після (у нижній частині) застосування алгоритму мінімізації зміщень.

4. Результати експериментів

Розглянемо на завершення результати застосування описаного алгоритму мінімізації зміщень розкладу LZ77 для стиснення восьми різнотипних 24-бітних зображень стандартного набору файлів ACT у форматі PNG (завантажити їх TIFF-версії можна, наприклад, з [7]).

Тестування проводилося за допомогою програми з CD [5], у яку були внесені такі модифікації: забезпечена можливість виходу з словника в буфер під час кодування

повторів; реалізований вибір предиктора для рядка пікселів зображення на основі співставлення ефективності стиснення рядка без попередньої обробки з оцінкою нерівномірності розподілу після дії кожного предиктора; запроваджений попередній аналіз кількостей додаткових біт перед записом зміщень; розмір блоків даних збільшений до 64 Кб та відкинуті допоміжні текстові блоки.

Результати тестування наведено в табл. 2, 3, де показником компресії файлів обрано коефіцієнт стиснення, тобто процент зменшення початкового розміру файла, оскільки він носить загальний характер, як і описаний алгоритм. Розглянутий алгоритм застосовувався до результатів *жадібного* розкладу та розкладу Кадача. Дані тестування програми без застосування розглянутого алгоритму наведено в другому та четвертому, а з застосуванням – у третьому та п'ятому стовпцях. Крім цього, для порівняння ефективності стиснення у шостому стовпці таблиць вказано результати тестування програми з використанням майже оптимального розкладу, яка враховувала кількості додаткових біт для запису довжин і зміщень у форматі DEFLATE. Оскільки цей розклад враховує використання мінімально можливих зміщень, то розглянутий у цій статті алгоритм до нього не застосовувався.

Таблиця 2: Зміни коефіцієнтів стиснення (%) файлів зображень наборуACT у форматі PNG після застосування алгоритму мінімізації зміщень

Bmp-файл	*Жадібний* розклад		Розклад Кадача		Майже оптим. розклад
	без алг.	з алг.	без алг.	з алг.	
Clegg	76.30	76.49	76.82	76.87	77.39
Frymire	93.15	93.40	93.26	93.43	93.57
Lena	33.68	33.68	34.33	34.33	35.24
Monarch	45.10	45.36	46.02	46.14	46.92
Peppers	41.35	41.61	42.52	42.52	43.30
Sail	32.96	33.13	33.65	33.74	33.91
Serrano	92.83	93.03	92.83	93.03	93.10
Tulips	38.94	39.12	39.90	39.98	40.85
Середній	**56.79**	**56.98**	**57.42**	**57.51**	**58.04**
Сукупний	**67.81**	**68.02**	**68.29**	**68.40**	**68.82**

Таблиця 3: Зміни часу стиснення (с) файлів зображень набору ACT у форматі PNG після застосування алгоритму мінімізації зміщень на комп'ютері з частотою 300 МГц

Bmp-файл	*Жадібний* розклад		Розклад Кадача		Майже оптим. розклад
	без алг.	з алг.	без алг.	з алг.	
Clegg	18.18	19.00	21.15	21.97	92.11
Frymire	22.90	23.01	25.22	25.54	214.21
Lena	7.25	7.42	7.79	8.02	9.66
Monarch	12.85	13.13	15.76	16.20	27.52
Peppers	8.08	8.30	9.39	9.61	13.46
Sail	11.15	11.59	12.14	12.41	14.78
Serrano	9.06	9.23	10.10	10.27	86.29
Tulips	11.86	12.14	13.79	14.17	20.21
Разом	**101.33**	**103.82**	**115.34**	**118.19**	**478.24**

Як свідчать результати тестування, застосування описаного алгоритму підвищило коефіцієнт стиснення програми, що використовувала *жадібний* розклад, максимум на 0.26%, а у середньому – на 0.19% та скоротило відставання від результатів програми, що використовувала майже оптимальний розклад у середньому на 15%. Застосування ж описаного алгоритму до результатів розкладу Кадача підвищило коефіцієнт стиснення програми максимум на 0.2%, а у середньому – на 0.09%, скоротивши при цьому відставання від результатів програми з майже оптимальним розкладом у середньому, знову ж таки, на 15%. Нижча ефективність алгоритму мінімізації зміщень у випадку його застосування до розкладу Кадача пояснюється тим, що цей розклад відділяє окремі елементи, що ефективніше кодуються окремо, тобто виконує частину роботи описаного алгоритму. З іншого боку, застосування розглянутого алгоритму сповільнило стиснення в середньому лише на 2.46%, в той час, як програма з майже оптимальним розкладом виконується повільніше в середньому у 4.05 рази.

5. Висновки

1. Для підвищення ефективності стиснення даних у форматах, що послідовно використовують алгоритми декількох методів, слід враховувати переваги наступних алгоритмів під час опрацювання даних попередніми алгоритмами.
2. Запропонований алгоритм мінімізації зміщень доцільно насамперед використовувати для покращення стиснення результатів *жадібного* розкладу LZ77 у випадку наявності послідовностей замін.
3. Описаний алгоритм не вимагає модифікації декодера чи програм перегляду зображень і за рахунок зменшення розмірів файлів лише прискорює їх роботу. Саме тому він може бути ефективно використаний для збереження даних у стандартах, що використовують алгоритм LZ77.

6. Посилання

[1] Методы сжатия данных. Устройство архиваторов, сжатие изображений и видео / Д. Ватолин, А. Ратушняк, М. Смирнов, В. Юкин. – М.: ДИАЛОГ-МИФИ, 2003. – 384 с.

[2] Ziv J., Lempel A. A universal algorithm for sequential data compression // IEEE Transactions on Information Theory. May 1977. Vol. 23(3). P. 337-343.

[3] Ziv J. Lempel A. Compression of individual sequences via variable-rate coding // IEEE Transactions on Information Theory. Sept. 1978. Vol. 24(5). P. 530-536.

[4] Миано Дж. Форматы и алгоритмы сжатия изображений в действии: Учеб. пособ. – М.: Триумф, 2003. – С. 249-318.

[5] Storer J.A., Szymanski T.G. Data compression via textual substitution // Journal of ACM. Oct. 1982. Vol. 29(4). P. 928-951.

[6] Кадач А.В. Эффективные алгоритмы неискажающего сжатия текстовой информации. – Дис. к. ф.-м. н. – Ин-т систем информатики им. А.П. Ершова. – Новосибирск, 1997. – С. 97-108.

[7] http: //compression.ca/act/act-files.html.