УДК 621.391 (075.8)

О.О. Кузьменко, Ю.М. Рибка

АЛГОРИТМ ШВИДКОЇ ПОБУДОВИ КОДІВ ХАФФМЕНА

Постановка проблеми. Розглядаються основні недоліки класичних способів побудови коду Хаффмена. Описано алгоритм швидкої побудови цих кодів, який дозволяє отримати код за час $O(m_1)$ (де m_1 – кількість символів у алфавіті).

Аналіз останніх досліджень, невирішені проблеми. Інтерес до оптимального кодування росте пропорційно збільшенню обсягів інформації, що спостерігається у всіх галузях науки й техніки. Зараз майже немає програм, призначених безпосередньо для побудови кодів Хаффмена. Однак є ряд програм стиснення даних при використанні цих кодів (наприклад, [1,2]). Всі вони мають серйозний недолік: використовуються алфавіти, що містять не більше 256 символів. Оскільки, відповідно до основної теореми кодування Шенона [3], ефективність коду збільшується при кодуванні блоків символів (що рівносильно збільшенню алфавіту), можливість використання великих алфавітів у програмі може значно підвищити ефективність отримуваних кодів. Зазначимо також, що часова складність класичних алгоритмів побудови коду Хаффмена дорівнює $O(m_1^2)$ [3].

Постановка задачі. Необхідно розробити алгоритм швидкої побудови кодів Хаффмена, який дозволяє отримати код за час $O(m_1)$. Методика повинна працювати з алфавітами довільної розмірності.

Основні результати. Алгоритм побудови коду Хаффмена складається з двох частин [3]:

знаходження на кожному кроці двох символів алфавіту з найменшими ймовірностями та об'єднання цих символів;

визначення кодової комбінації для кожного символу алфавіту.

При реалізації класичного алгоритму побудови цього коду для знаходження символів із найменшими ймовірностями та їх об’єднання виконується повне перебирання усіх символів, тому часова ефективність становить $O(m_1^2)$ [3].

Можна використати для цієї мети структуру “піраміда”. Ця структура даних, по суті, є двійковим деревом, тому обчислювальна складність виконання злиттів символів складає $O(m_1 \log_2 m_1)$ [3].

Запропонуємо алгоритм, що дозволяє вибирати два символи з найменшими ймовірностями за час, що не залежить від розміру алфавіту. Виконання всіх необхідних об'єднань символів при цьому займає час $O(m_1)$.

Розглянемо процес виконання злиттів на прикладі алфавіту, символи якого мають ймовірності: 0,25, 0,25, 0,125, 0,1, 0,1, 0,06, 0,04, 0,025, 0,025, 0,025 (рис.1). На цьому рисунку використовуються такі умовні позначення: стрілки показують, які символи поєднуються на поточному кроці стиснення алфавіту, товста горизонтальна лінія розділяє символи на дві групи: символи, які ще не брали участь у злиттях розташовуються над нею, символи, отримані шляхом об'єднання символів - під нею. Будемо символ, отриманий при злитті, поміщати під цією лінією (якщо під нею були деякі символи, вони зрушуються вниз).

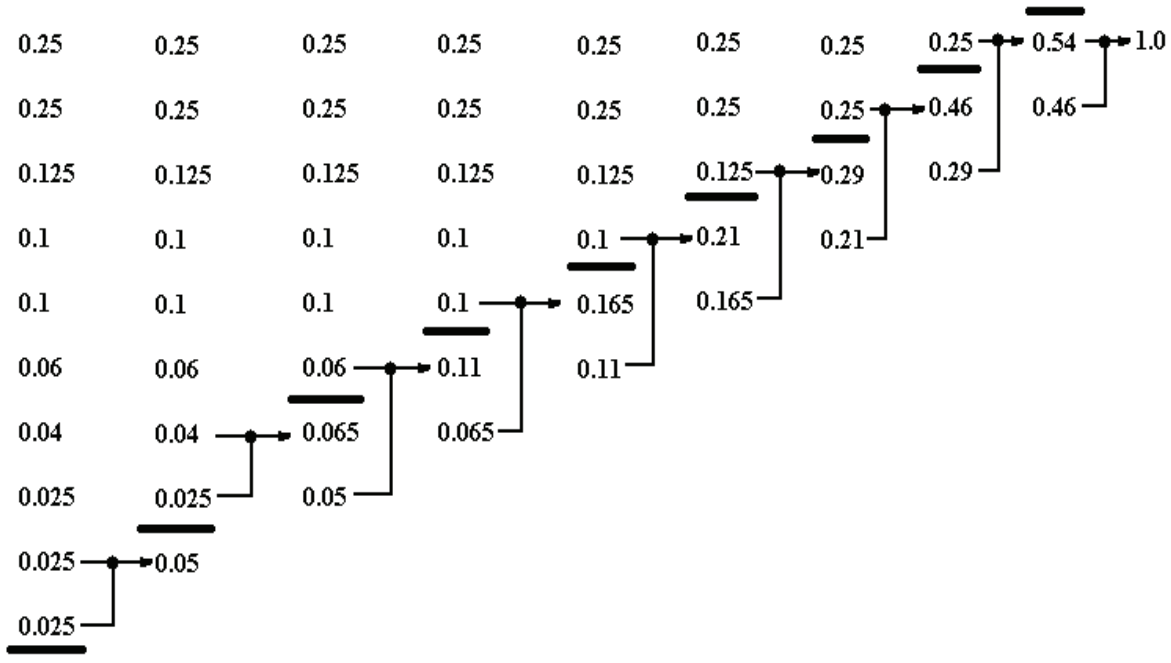

Рисунок 1 - Об’єднання символів алфавіту

Даний алгоритм можна описати наступним чином:

Упорядкувати символи вхідного алфавіту в порядку не зростання їх ймовірностей. Оскільки ще не було зроблено жодного об'єднання символів, всі вони потраплять у групу 1 - групу символів, що не

 ISSN 1562-9945

брали участь у злиттях. Група 2 (символи, отримані шляхом об'єднання символів) на даному кроці є пустою.

Знайти два символи із найменшими ймовірностями. Пошук цих символів ведеться як у групі 1, так і у групі 2.

Знайдені символи виключити із груп і поєднати в один символ. Новий символ помістити в групу 2 на перше місце.

Якщо в обох групах разом більше одного символу, перейти до кроку 2.

Крок 2 алгоритму зводиться до того, що спочатку в групі 1 визначаються два символи з найменшими ймовірностями, потім ця ж операція повторюється для групи 2. Для визначення двох символів, що підлягають злиттю, потрібно вибрати із чотирьох раніше отриманих символів (два символи із групи 1 і два із групи 2) два символи із найменшими ймовірностями. Обрані символи виключаються із груп, а символ, утворений шляхом їх злиття поміщається на першу позицію в групу 2.

Оскільки символи групи 1 початково були впорядковані за зменшенням їх ймовірностей і надалі в цю групу не додаються нові символи, то впорядкованість символів у ній буде зберігатися. Таким чином, для знаходження символів з найменшими ймовірностями в групі №1 досить взяти із неї два останні символи.

З рис. 1 видно, що символ котрий додається на кожному кроці в початок групи 2 має ймовірність, більшу за ймовірність символу, що помістився на перше місце на попередньому кроці. Тобто на протязі даного алгоритму символи групи 2 також упорядковані за зменшенням їх ймовірностей. Тому два символи з найменшими ймовірностями - два останні символи цієї групи.

Обчислення довжин кодових значень. Розглянемо алгоритм обчислення довжин кодових комбінацій з ефективністю $O(m_1)$.

Нехай маємо алфавіт із ймовірностями символів 0,25, 0,25, 0,15, 0,15, 0,05, 0,05, 0,05, 0,05. Виконаємо злиття символів. При цьому на кожному кроці записуємо номери символів що об’єднуються до робочого масиву (рис. 2). Кожний елемент цього масиву являє собою пари номерів об'єднаних символів. На підставі цих даних легко можна побудувати дерево Хаффмена (рис 3).

a								
1	0.25	0.25	0.25	0.25	0.25	0.25	0.55	1.0
2	0.25	0.25	0.25	0.25	0.25	0.45	0.45	0
3	0.15	0.15	0.15	0.15	0.3	0.3	0	0
4	0.15	0.15	0.15	0.15	0	0	0	0
5	0.05	0.05	0.1	0.2	0.2	0	0	0
6	0.05	0.05	0	0	0	0	0	0
7	0.05	0.1	0.1	0	0	0	0	0
8	0.05	0	0	0	0	0	0	0

Вміст робочого масиву: (8, 7); (6, 5); (7, 5); (4, 3); (5, 2); (3, 1); (2, 1)

Рисунок 2 - Приклад заповнення робочого масиву

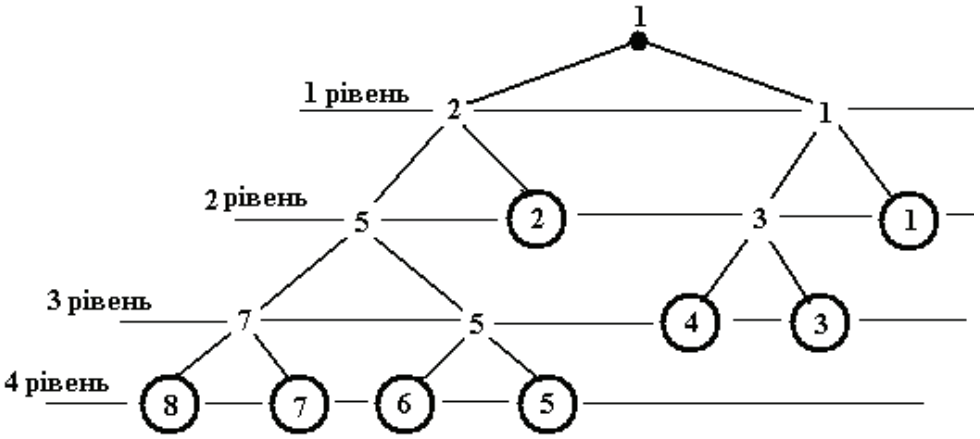

Рисунок 3 - Дерево Хаффмена, побудоване на основі робочого масиву

Довжина коду символу дорівнює номеру того рівня в дереві, на якому міститься цей символ. Випишемо всі вузли дерева, починаючи із “найглибших”: 4 рівень - 8,7,6,5; 3 рівень - 7,5,4,3; 2 рівень - 5,2,3,1; 1 рівень - 2,1. Таким чином, отримуємо наступну послідовність вузлів: 8, 7, 6, 5, 7, 5, 4, 3, 5, 2, 3, 1, 2, 1. Видно, що дана послідовність збігається із вмістом робочого масиву. А оскільки її елементи розташовані в порядку зменшення їх рівня в дереві, то й елементи робочого масиву (номера символів) будуть розташовані в порядку зменшення довжин їх кодів. Ця властивість обумовлює однократний обхід масиву з $m_1 - 1$ елементів. При цьому на кожному кроці виконується проста операція - інкремент умісту деякого лічильника. Очевидно, що складність алгоритму складе $O(m_1)$.

 ISSN 1562-9945

Обчислення кодів. Класичний алгоритм визначення кодових комбінацій [3] не дуже зручний при реалізації, тому що для побудови кодів необхідно додатково визначити максимальну довжину коду, кількість кодів кожної довжини та початкове значення коду для кожної з довжин.

Наведемо алгоритм, що дозволяє безпосередньо, тільки на підставі довжин кодових комбінацій, визначати коди символів.

Перейти до символу, що має найбільшу довжину коду. Змінній code_len присвоїти значення, рівне цій довжині.

Нехай поточний символ - a_i. l_i - довжина коду для a_i. Якщо l_i менше значення code_len, то зсунути значення code_val вправо на code_len - l_i розрядів. Змінній code_len присвоїти значення l_i.

Кодова комбінація символу a_i дорівнює code_val.

Значення змінної code_val збільшити на 1.

Якщо ще не обчислені кодові представлення для всіх символів, то повторити кроки 2-4 для символу a_{i-1}.

Покажемо, що цей алгоритм дійсно обчислює канонічні коди Хаффмена. Символи впорядковані за зменшенням довжини відповідних їм кодів. Припустимо, довжина коду першого символу дорівнює L, тоді цей символ одержить кодову комбінацію $S_L = 0...0$, що складається з L нулів. Другий символ з довжиною коду L одержить представлення $S_L + 1$, третій символ - $S_L + 2$, і т.д. Останній символ з довжиною коду L одержить представлення рівне $S_L + N_L - 1 = 0 + N_L - 1$, де N_L - кількість символів, що мають коди довжини L. Припустимо, далі йде символ з довжиною коду рівної $L-1$, тоді йому буде поставлений у відповідність код $S_{L-1} = (N_L - 1 + 1) >> L - (L-1)$, тобто $S_{L-1} = (0 + N_L) >> 1$. Символи з довжиною коду $L-1$ отримають коди S_{L-1}, $S_{L-1} + 1$, $S_{L-1} + 2, ..., S_{L-1} + N_{L-1} - 1$... Символи з довжиною коду $L-2$ одержать коди $S_{L-2} = (S_{L-1} + N_{L-1}) >> 1$, $S_{L-2} + 1$, $S_{L-2} + 2, .., S_{L-2} + N_{L-2} - 1$.

При цьому немає необхідності заздалегідь обчислювати початкові значення для кодів кожної довжини, оскільки вони утворюються автоматично в процессі роботи алгоритму.

Порівняння з іншими алгоритмами. Нижче наведені результати порівняння часу роботи програм побудови кодів Хаффмена tplzh (автор Joe Jared [1]) та PQHuff (автор Mark Nelson [2]) з розробленою авторами даної роботи програми Huff.

У програмі tplzh для знаходження символів із найменшими ймовірностями використовується повне перебирання всіх символів, що зумовлює її низьку ефективність - $O(m_1^{\,2})$. Програма PQHuff використовує для цієї мети структуру “піраміда” і має ефективність порядку $O(m_1 \log_2 m_1)$.

Програма Huff реалізує описаний у даній роботі алгоритм швидкої побудови кодів Хаффмена. Її ефективність $O(m_1)$.

Тестування проводилося на комп’ютері, з процесором Intel Celeron 700 МГц, оперативна пам’ять 128 Мб. Для кожного алфавіту виконувалося 5 запусків кожної з програм і обчислювався середній час побудови кодів. Результати дослідження представлені у табл. 1.

Таблиця 1

Порівняння програм для побудови кодів Хаффмена

Розмір алфавіту, тис. символів	Час побудови кодів, мс		
	tplzh v0.31	PQHuff	Huff v1.0.7
50	2845	2254	120
100	3100	2603	225
250	4790	3585	510
500	8910	5980	970
1000	---	9744	1909
2000	---	---	3926

Висновки. В роботі було проведено аналіз існуючих програм для побудови кодів Хаффмена та розроблено алгоритм швидкої побудови коду Хаффмена (на основі нього було написано программу Huff).

Проведено також порівняння розробленого програмного продукту із двома аналогами, в результаті якого програма Huff показала вищу ефективність.

Алгоритм реалізовано у вигляді параметризованих класів мови C++. Це значно спростить його повторне використання у інших програмах.

 ISSN 1562-9945

ЛІТЕРАТУРА

1. Програма для стиснення з використання кодів Хаффмена. FTP://webworldinc.com/joejared/tplzh031.zip.
2. Програма PQHuff для стиснення даних з використання кодів Хаффмена. http://www.dogma.net/markn/articles/pqstl/source.zip.
3. Жураковський Ю.П., Полупорак В.П. Теорія інформації та кодування. –К.: Вища школа. 2001.

Получено 22.11.2007 г.