Висновки

Таким чином, використання запропонованої автоматизованої системи контролю знань на основі методів експертних систем дозволяє з більшою достовірністю визначати якісний рівень знань студентів і генерувати керуючі впливи для подальшого вдосконалення структури та представлення знань в предметній області.

Подальші дослідження даного напряму будуть зосереджені на формалізації структури представлення знань інформаційної інтелектуальної системи для предметної області «інформаційні та комп'ютерні технології».

Література

1. Богданевич М.А. Організація системи перевірки знань для поточного та підсумкового контролю. Друга Міжнародна науково-методична конференція «ІНТЕРНЕТ – ОСВІТА – НАУКА – 2000», м. Вінниця.

2. Зайцева Л.В.,Прокофьева Н.О., Куплис У.Г. Компьютерные системы в дистанционном обучении // ТЕЛЕМАТИКА'2001 – Санкт-Петербург, 2001. – С. 109-111.

3. Паволоцкий А.В. Методика проведения автоматизированного контроля знаний // Доповідь – МПДУ, 2007

4. Пустынникова И.Н. Методология конструирования диагностирующей экспертной системы (на базе оболочки BESS) // Вісник Донецького університету. – Серія А. Природничі науки. – 1998. – № 1. – С. 182-187.

5. Андреев А.Б., Усачев Ю.Е. Экспертная система анализа знаний как инструмент контроля усвоения зачетных единиц // Проблемы введения системы зачетных единиц в высшем профессиональном образовании: Материалы к Всероссийскому совещанию 23 апреля 2003 года, г. Москва / Под ред. В.Н. Чистохвалова. – М.: Изд-во РУДН, 2003. – 100 с.

Надійшла 5.11.2009 р.

УДК 621.37:681.33

І.І. МІТАСОВ, О.В.РУБАН, В.М. РУДНИЦЬКИЙ
Хмельницький національний університет

ОСОБЛИВОСТІ ЗАСТОСУВАННЯ МЕТОДІВ СТИСНЕННЯ ДАНИХ

В статті охарактеризовані відомі методи стиснення даних, розглянуті особливості стиснення статистичними і словниковими методами, з втратами і без втрат, порівняно статичні і динамічні моделі. Описані можливості архіваторів.

In article were discussed methods of different data compression algorithms. Different archivers and their capabilities were discassed.

Ключові слова: стиснення даних, алгоритми стиснення даних.

Вступ

При передачі та зберіганні інформації завжди виникає проблема розміру повідомлень. Розвиток засобів комп'ютерної техніки веде до збільшення швидкостей передачі та об'ємів накопичувачів. Може скластись враження про недоцільність використання методів стиснення інформації через це зростання. Але це погляд лише з одного боку. З іншого боку відбувається зростання обсягів самих інформаційних повідомлень. При чому не рідко зростання обсягів передачі відбувається швидше, ніж зростають швидкості передачі. Окрім цього за передачу інформації потрібно платити. Зменшення плати за передачу інформації ще одна з позитивних сторін використання методів стиснення інформації. Тому проблема стиснення інформації завжди залишиться актуальною.

Мета стиснення – зменшення кількості біт, необхідних для зберігання або передачі заданої інформації. Це дає можливість передавати повідомлення швидше і зберігати їх більш економно і оперативно. Останнє означає, що операція отримання даної інформації з пристрою її зберігання проходить швидше, а це можливо, якщо швидкість розпаковування даних вище за швидкість прочитування даних з носія інформації. Стиснення дозволяє, наприклад, записати більше інформації на дискету, "збільшити" розмір жорсткого диска, прискорити роботу з модемом і т.п. При роботі з комп'ютерами широко використовуються програми-архіватори даних формату *ZIP, GZ, ARJ* та інших. Методи стиснення інформації були розроблені як математична теорія, яка довгий час (до першої половини 80-х років), мало використовувалася в комп'ютерах на практиці.

Збільшення обсягів передачі, статичних картинок, звуку та відеозображень призвело до розроблення багатьох методів стиснення з втратами або незворотного кодування. Для передачі саме цих типів повідомлень допускається використання таких методів. Але досить часто необхідно передавати документи, фрагменти баз даних, виконуванні файли. Неможливість повного відновлення таких файлів призведе фактично до втрати інформації.

Тому і досі залишаються актуальними методи оборотного стиснення інформації, при яких гарантується повне відновлення початкового повідомлення. До таких методів відносяться найпростіші алгоритми стиснення, розроблені класиками теорії кодування К. Шенноном, Р. Фано, Д. Хаффменом [1,2,3].

Стиснення даних

Стиснення даних – процедура перекодування даних, що проводиться з метою зменшення їх обсягу. Стиснення засновано на усуненні надлишковості інформації, що міститься у вхідних даних. Стиснення даних, що не мають властивості надлишковості (наприклад, випадковий сигнал або шум), неможливо. Найпростіші алгоритми стиснення, що також називаються алгоритмами оптимального кодування, є статистичними і засновані на врахуванні розподілу ймовірностей елементів вхідного повідомлення (тексту, зображення, файлу). На практиці в якості наближення до ймовірностей використовують частоти появи елементів у вхідному повідомленні. Ймовірність – абстрактне математичне поняття, пов'язане з нескінченними експериментальними вибірками даних, а частота появи – величина, яку можна обчислити для кінцевих множин даних. При досить великій кількості елементів у множині експериментальних даних можна говорити, що частота появи елемента близька (з деякою точністю) до його ймовірності.

Якщо згадані ймовірності неоднакові, то є можливість найбільш ймовірним елементам (тим, що часто зустрічаються) зіставити більш короткі кодові слова і, навпаки, малоймовірним елементам зіставити більш довгі кодові слова. Таким способом можна зменшити середню довжину кодового слова. Оптимальний алгоритм кодування робить це так, щоб середня довжина кодового слова була мінімальною, тобто при меншій довжині кодування стане незворотним. Такі алгоритми існують, вони відомі під назвами Шеннона-Фано і Хаффмена [1,2]. Недоліком обох методів є те, що вони не здатні закодувати повідомлення більш ощадливо, ніж один біт на елемент повідомлення (літеру).

Отже, була поставлена мета винайти таку схему кодування, яка дозволяла б кодувати деякі елементи менш ніж одним бітом. В 1977 р. Дж. Райссанен запропонував для цього один з кращих методів під назвою «арифметичне кодування» і запатентував цю розробку для IBM.

Відомим в цій групі методів є також алгоритм групового кодування – *RLE* (*Run - Length - Encoding*).

Припустимо, що розмір алфавіту *N*=256, і ми стискаємо звичайний текстовий файл (*ASCII*). Швидше за все, ми не зустрінемо всі *N* символів нашого алфавіту в такому файлі. Покладемо тоді довжину коду відсутніх символів рівною нулю. У цьому випадку список довжин кодів буде містити досить велику кількість нулів (довжин кодів відсутніх символів) згрупованих разом. Кожну таку групу можна стиснути за допомогою групового кодування – *RLE* . Цей алгоритм надзвичайно простий. Замість послідовності з *M* однакових елементів, що йдуть підряд, будемо зберігати перший елемент цієї послідовності та число його повторень, тобто (*M*-1), наприклад: *RLE* ("*AAAABBBCDDDDDDD*") = *A*3 *B*2 *C*0 *D*6.

Дотепер ми розглядали *статистичний* підхід до стиснення файлів невідомого формату.

Є ще другий фундаментальний підхід, *словниковий*, де на кожному кроці алгоритму стиснення наступний символ поміщається таким, як він є (зі спеціальним прапором відсутності стиснення), або вказуються границі слова з попереднього тексту, що збігається з наступними символами файлу. Розархівування файлів, стиснутих у такий спосіб, виконуються дуже швидко, тому ці алгоритми використовуються для створення програм, що саморозпаковуються. Словникові алгоритми носять менш математично обгрунтований, але більш практичний характер.

Серед словникових першим був розроблений алгоритм *LZ*77 [2,3] ізраїльськими математиками Якобом Зівом (*Ziv*) і Авраамом Лемпелом (*Lempel*) і опублікований в 1977 р. Багато програм стиснення інформації використовують ту або іншу модифікацію *LZ*77. Основна ідея *LZ*77 полягає в тому, що друге і подальші входження деякого рядка символів в повідомленні замінюються посиланнями на його перше входження.

*LZ*77 використовує вже проглянуту частину повідомлення як словник. Аби добитися стиснення, він намагається замінити черговий фрагмент повідомлення на покажчик у вміст словника. Для цього використовується зміщуване по повідомленню вікно, розділене на дві нерівні частини. Перша, велика за розміром, включає вже проглянуту частину повідомлення. Друга, набагато менша, є буфером, що містить ще незакодовані символи вхідного потоку. Зазвичай розмір вікна складає декілька кілобайт, а розмір буфера - не більше ста байт. Алгоритм намагається знайти в словнику (більшій частині вікна) фрагмент, співпадаючий з вмістом буфера.

Спрощений приклад: розмір вікна — 20 символов, словника — 12 символов, а буфера — 8. Кодується повідомлення "®_ПРОГРАМНІ_ПРОДУКТИ_ФІРМИ_*MICROSOFT*". Нехай словник вже заповнений. Тоді він містить рядок "®_ПРОГРАМНІ_", а буфер — рядок "ПРОДУКТИ". Переглядаючи словник, алгоритм виявить, що співпадаючим підрядком буде "ПРО", в словнику він розташований із зсувом 2 (рахувати починаємо з нуля) і має довжину 3 символи, а наступним символом в буфері є "Д". Таким чином, вихідним кодом буде трійця (2,3,'Д'). Після цього алгоритм зсуває вліво весь вміст вікна на довжину співпадаючого підрядка +1 і одночасно прочитує стільки ж символів з вхідного потоку в буфер. Отримуємо в словнику рядок "ОГРАМНІ_ПРОД", в буфері — "УКТИ_ФІРМИ". У даній ситуації співпадаючого підрядка виявити не вдається і алгоритм видасть код (0,0,'У'), після чого зсуне вікно на один символ. Потім словник міститиме "ГРАМНІ_ПРОДУ", а буфер — "КТИ_ФІРМ". І так далі.

Словникові алгоритми інтенсивно вдосконалювались. У 1978 р. ті ж автори вдосконалили свій алгоритм під назвою *LZ*78, у 1982 р. Сторером (*Storer*) і Шиманським (*Szimanski*) на базі *LZ*77 був

розроблений алгоритм *LZSS*, який відрізняється від *LZ*77 вироблюваними кодами, а у 1984 р. Велчем (*Welch*) був створений алгоритм *LZW*, шляхом модифікації *LZ*78.

Стиснення без втрат

Стиснення буває без втрат (коли можливе відновлення вхідних даних без спотворень) або з втратами (відновлення можливе з незначними спотвореннями). Стиснення без втрат використовується при обробці комп'ютерних програм і даних. Стиснення з втратами зазвичай застосовується для скорочення обсягу звукової, фото- і відеоінформації, воно значно ефективніше ніж стиснення без втрат.

Стиснення без втрат *LDC* (англ. *Lossless data compression*) – метод стиснення інформації, при використанні якого закодована інформація може бути відновлена з точністю до біта. При цьому оригінальні дані повністю відновлюються зі стиснутого стану. Цей тип стиснення кардинально відрізняється від стиснення даних із втратами. Для кожного типу цифрової інформації, як правило, існують свої алгоритми стиснення без втрат.

Стиснення даних без втрат використовується в багатьох додатках. Наприклад, воно використовується в популярному файловому форматі *ZIP* та *Unix*-утиліті *Gzip*. Воно також використовується як компонент у стисненні з втратами.

Стиснення без втрат використовується, коли важлива ідентичність стиснутих даних оригіналу. Звичайний приклад – виконувані файли і вихідний код. Деякі графічні файлові формати, такі як *PNG* або *GIF*, використовують тільки стиснення без втрат; тоді як інші (*TIFF*, *MNG*) можуть використовувати стиснення як із втратами, так і без.

Методи стиснення без втрат можуть бути розподілені по типу даних, для яких вони були створені. Три основних типи даних для алгоритму стиснення - це текст, зображення і звук. У принципі, будь-який багатоцільовий алгоритм стиснення даних без втрат (багатоцільовий означає, що він може обробляти будь-який тип бінарних даних) може використовуватися для будь-якого типу даних, але більшість із них неефективні для кожного основного типу. Звукові дані, наприклад, не можуть бути добре стиснуті алгоритмом стиснення тексту.

Порівняльний аналіз методів стиснення

Порівння алгоритмів Хаффмена та Шеннона-Фано

Кодування Шеннона-Фано є досить старим методом стиснення, і на сьогоднішній день воно не представляє особливого практичного інтересу. У більшості випадків, довжина стиснутої по даному методу послідовності дорівнює довжині стиснутої послідовності з використанням кодування Хаффмена. Але існують послідовності, на яких методика Шеннона-Фано не дає однозначного кодування, тому стиснення методом Хаффмена прийнято вважати більш ефективним. Для прикладу (табл. 1), розглянемо послідовність що містить наступні символи: ***a*** - 14, ***b*** - 7, ***c*** - 5, ***d*** - 5, ***e*** – 4 (цифри означають кількість повторень). Метод Хаффмена стискає її до 77 біт, а Шеннона-Фано - до 79 біт.

Таблиця 1

Приклад стиснення послідовності

символ	код Хаффмена	код Шеннона-Фано
a	0	00
b	111	01
c	101	10
d	110	110
e	100	111

Такі відмінності в ступені стиснення виникають через нестроге визначення способу розподілу символів на групи в методиці Шеннона-Фано.

Як необхідно ділили на групи ?

1. Ймовірність першої групи (p_1) і другої (p_2) дорівнює нулю;
2. $p_1 \le p_2$?
   - так: додати в першу групу символ з початку таблиці;
   - ні: додати в другу групу символ з кінця таблиці;
3. Якщо всі символи розділені на групи, то завершити алгоритм, інакше перейти до кроку 2.

При побудові коду Шеннона-Фано розбиття множини елементів може бути здійснено, загалом кажучи, декількома способами. Вибір розбиття на рівні n може погіршити варіанти розбивки на наступному рівні (n+1) і призвести до неоптимальності коду в цілому. Інакше кажучи, оптимальне поводження на кожному кроці шляху ще не гарантує оптимальності всієї сукупності дій. Тому код Шеннона-Фано не є оптимальним у загальному значенні, хоча і дає оптимальні результати при деяких розподілах ймовірностей. Для того самого розподілу ймовірностей можна побудувати, загалом кажучи, кілька кодів Шеннона-Фано, і всі вони можуть дати різні результати. Якщо побудувати всі можливі коди Шеннона-Фано для даного розподілу ймовірностей, то серед них будуть знаходитися і всі коди Хаффмена, тобто оптимальні коди.

Відмінності статичної і динамічної моделей

Практично всі існуючі алгоритми стиснення даних можна реалізувати у двох варіантах – статичному і динамічному. Кожний з них займає певну нішу, використовуючись у тих випадках, коли він найбільш зручний. Алгоритм Хаффмена не є винятком – застосовуючись у багатьох архіваторах і форматах, він використовується іноді в статичному (*JPEG, ARJ*) або в динамічному (утиліта *compact* для *UNIX, ICE*) варіантах.

Алгоритм Хаффмена стискає дані за рахунок ймовірностей появи символів у джерелі, отже для того щоб успішно щось стиснути і розтиснути, потрібно знати ці самі ймовірності для кожного символу. Статичні алгоритми справляються із цією справою просто – перед тим, як почати стиснення файлу, програма пробігається по самому файлу і підраховує, який символ скільки разів зустрічається. Потім, відповідно до ймовірностей появ символів, будується двійкове дерево Хаффмена, звідки витягаються відповідні кожному символу коди різної довжини. І на третьому етапі знову здійснюється прохід по вихідному файлі, коли кожен символ заміняється на свій код з дерева. У такий спосіб статичному алгоритму потрібно два проходи по файлу-джерелу, щоб закодувати дані.

Динамічний алгоритм дозволяє реалізувати однопрохідну модель стиснення. Не знаючи реальних ймовірностей появ символів у вихідному файлі, програма поступово змінює двійкове дерево з кожним символом, збільшуючи частоту його появи в дереві та перебудовуючи у зв'язку із цим саме дерево. Однак стає очевидним, що вигравши в кількості проходів по вихідному файлу, губиться ступінь стиснення, тому що в статичному алгоритмі частоти появи символів були відомі із самого початку і довжини кодів цих символів були більш близькі до оптимальних, у той час як динамічна модель, вивчаючи джерело, поступово доходить до його реальних частотних характеристик і формує їх, лише повністю пройшовши вихідний файл. Цей недолік можна компенсувати іншою перевагою динамічного алгоритму. Оскільки динамічне двійкове дерево постійно модифікується новими символами, немає необхідності запам'ятовувати їхні частоти заздалегідь. При розархівуванні, програма, одержавши з архіву код символу, точно так само відновить дерево, як вона це робила при стисненні та збільшить на одиничку частоту його появи. Більше того, не потрібно запам'ятовувати які символи у двійковому дереві не зустрічаються. Всі символи, котрі будуть додаватися в дерево, і є тими, що нам будуть потрібні для відновлення первісних даних. У статичному алгоритмі із цією справою небагато складніше – на самому початку стиснутого файлу потрібно зберігати інформацію про символи, що зустрічаються у файлі джерела, і їхні частоти. Це обумовлено тим, що ще до початку розархівування необхідно знати, які символи будуть зустрічатися і який буде їхній код.

Архіватори

Якщо коди алгоритмів типа *LZ* передати для кодування алгоритму Хаффмена або арифметичному, то отриманий двокроковий (конвеєрний, а не двопрохідний) алгоритм дасть результати стиснення, подібні до широко відомих програм: *GZIP, ARJ, PKZIP* ...

Найкращий показник стиснення дають двопрохідні алгоритми, які стискають початкові дані послідовно двічі, але вони працюють до двох разів повільніше однопрохідних при незначному збільшенні степені стиснення.

Більшість програм-архіваторів стискають кожен файл окремо, але деякі стискають файли в загальному потоці, що дає збільшення степені стиснення, але одночасно ускладнює способи роботи з отриманим архівом. Наприклад, заміна в такому архіві файлу на його новішу версію може вимагати перекодування всього архіву. Прикладом програми, що має можливість стискати файли в загальному потоці, є *RAR*. Архіватори ОС *Unix (gzip, bzip2 ...)* стискають файли в загальному потоці практично завжди.

Одними з представників цієї категорії є *Winzip* та *WinRar*.

WinZip – потужна програма для роботи як з *Zip*-файлами, так і з архівами інших форматів. У *WinZip* вбудована підтримка *CAB*-файлів, а для таких популярних форматів як *TAR*, *gzip*, *BinHex*, *ARJ*, *LZH*, *ARC* та ін. здійснюється підтримка через зовнішні програми. У ньому вдосконалено технологію стиснення, що створює менші по розміру архівні файли. Покращено керування файлами – з'явилася можливість використання "*drill down*" для перегляду тільки тих документів у папках, які необхідні для роботи, а також додавати файли безпосередньо у вже існуючі або нові папки. *WinZip Job Master* - новий інструмент в *WinZip* 11.0 *Pro*, що дозволяє попередньо виконати резервне копіювання визначених даних, дає можливість настроювання та планування завдань стиснення і функціональні можливості *FTP*-завантаження. Так само немає необхідності створювати архівні файли спочатку на жорсткому диску - можна відразу створити їх на *CD* або *DVD*, великі *ZIP*-файли можуть бути автоматично розміщені на декількох *CD* або *DVD*.

WinRar – популярний *RAR*-архіватор та менеджер архівів. Крім свого основного формату (*rar*), вміє працювати з *zip*, *cab*, *ace*, *arj* та іншими архівами. Ступінь стиснення файлів цією програмою значно вище, ніж у *WinZip*. За допомогою *WinRar* можна відновлювати пошкоджені файли (опція *repair*). При цьому ймовірність відновлення файлів буде більшою, якщо при створенні архіву в настройках програми була вказана опція *put recovery record*. Серед інших налаштувань *WinRar* можна відмітити можливість створення саморозпаковуючих архівів з вказанням шляху розпакування, а також встановлення ступеня стиснення – від найкращої (але найповільнішої) до найшвидшої (але менш якісною). Крім того, склад архіву можна зробити невидимим.

Внаслідок того, що з *WinRar* є можливість стискати файли на 8-15% краще і швидше, проходить велика економія дискового простору, втрат на передачу даних і, найголовніше, робочого часу. *WinRar* прекрасно підходить для створення мультимедійних файлів. Також є можливість розділити архіви на окремі

частини для запису їх на різні носії. Він ідеально підходить для передачі конфіденційних даних по Інтернету та іншим незахищеним каналам, маючи 128-бітний криптографічний захист та електронні підписи архівів.

Формат файлу, що містить дані, які перед використанням потрібно розпакувати відповідною програмою-архіватором, як правило, може бути ідентифікований розширенням імені файлу. У табл. 2 наводяться деякі типові розширення, відповідні ним програми-архіватори і методи стиснення даних.

Таблиця 2

Відповідність типових розширень, програм-архіваторів і методів стиснення даних

Розширення	Програми	Тип кодування
z	*compress*	*LZW*
arc	*arc, pkarc*	*LZW*, Хаффмена
zip	*zip, unzip, pkzip, pkunzip*	*LZ*77, *LZW*, Хаффмена, Шеннона-Фано
gz	*gzip*	*LZ*77, Хаффмена
bz2	*bzip2*	Хаффмена, Берроуза-Виллера
arj	*arj*	*LZ*77, Хаффмена
ice, lzh	*lha, lharc*	*LZSS*, Хаффмена
pak	*pak*	*LZW*

Практично всі формати файлів для зберігання графічної інформації використовують стиснення даних. Формат графічного файлу також, як правило, ідентифікується розширенням імені файлу.

У табл. 3 наводяться деякі типові розширення графічних файлів і відповідні ним методи стиснення даних.

Таблиця 3

Відповідність типових розширень графічних файлів і методів стиснення даних

Розширення	Тип кодування
gif	*LZW*
jpeg, jpg	стиснення з втратами, Хаффмена або арифметичне
bmp, pcx	*RLE*
tiff, tif	*CCITT*/3 для факсов, *LZW* або інші

Висновки.

Класичні методи стиснення даних Шеннона-Фано і Хаффмена поступаються місцем більш новим: арифметичному з групи статистичних методів і словниковим методам стиснення даних, що раціонально доповнюють статистичні. В побудові архіваторів враховано особливості стиснення і словниковими, і статистичними методами, з втратами і без втрат, статичні і динамічні моделі. Як правило, архіватори побудовані на декількох методах стиснення даних.

Література

1. Жураковський Ю.П., Полторак В.П. Теорія інформації та кодування: Підручник – К.: Вища школа, 2001, – 255 с.

2. Лидовский В.В. Теория информации: Уч. Пособие. - М.: Компания Спутник+, 2004. - 111с.

3. Шульгин В.И. Основы теории передачи информации Ч.I. Экономное кодирование. Учеб. пособие. Харьков: Нац. аэрокосм. ун-т «Харьк. авиац. ин-т», 2003. - 102с.

Надійшла 5.11.2009 р.