Московский государственный университет имени Ломоносова

Механико-математический факультет

**Курсовая работа**

**на тему «Алгоритмы сжатия данных»**

Студент 3го курса И. Межиров

Научный руководитель А. Шень

Москва 2004

**Оглавление**

Введение 1

Основной алгоритм сравнения пары букв 3

Быстрый отказ для пары разных букв 5

Получение монохромных изображений 7

Ссылки 9

**Введение**

В этой работе описывается способ улучшения сжатия файлов формата DjVu.

Файл формата DjVu хранит в сжатом виде одно или несколько растровых изображений, при этом, чем больше эти изображения похожи на печатный текст, тем лучше сжатие. В отличие от сжатия алгоритмом JPEG, при сжатии в DjVu края букв не размываются.

Этот формат был придуман и развивался в фирме AT&T между 1995 и 1999 годами. В 2000 году все права были проданы фирме LizardTech. В том же году LizardTech опубликовала часть кода под лицензией GPL. На основе этого кода группа разработчиков, работавшая над этим проектом в AT&T, создала свой независимый вариант (тоже под лицензией GPL), который называется DjVuLibre [1]. В бесплатно доступных программах есть возможности разархивирования и просмотра DjVu-файлов, но только демонстрационное сжатие. К сожалению, эти программы не работают под Windows (разве что через cygwin — перенос среды Unix в Windows). Программы с полными возможностями продаются фирмой LizardTech; с ее сайта [4] можно также бесплатно скачать программу для просмотра DjVu под Windows.

В DjVu-файле хранится отдельно фон, передний план и монохромная маска, определяющая, какие пиксели изображения принадлежат фону, а какие — переднему плану. Фон и передний план кодируются алгоритмом IW44, который, как и JPEG, несколько размывает изображение. Авторы утверждают, что IW44 достигает уменьшения размеров файла в два раза по сравнению с JPEG при том же уровне искажений.

Черно-белая маска кодируется алгоритмом JB2. Этот алгоритм достигает сжатия в 4-6 раз лучшего, чем TIFF-G4 (CCITT/MMR Group 4). Как и в случае со многими другими алгоритмами (МРЗ, например), алгоритм декодировщика определен жестко, однако в алгоритме кодировщика возможны изменения, улучшающие сжатие.

Часть DjVu-файла, сжатая по алгоритму JB2, представляет собой заархивированную последовательность команд. Эти команды могут иметь вид "поставить такое-то изображение в такую-то точку", а могут иметь вид "поставить одно из ранее встречавшихся изображений в такую-то точку". Так как в первом случае команда содержит целое изображение, пусть и заархивированное, а во втором случае — всего лишь номер, то ясно, что чем больше элементарных изображений одинаковы, тем лучше будет сжатие.

Изображение, поступающее на вход кодировщика, разделяется на буквы выделением черных компонент связности. Алгоритм рассчитан на то, что элементарными изображениями будут как раз буквы, и действительно, после такого разбиения обычно остаются отдельные буквы, хотя и куски в 2-3 буквы тоже часто встречаются.

В отсканированном документе, скорее всего, не будет ни одной пары одинаковых изображений букв, хотя будет очень много букв, неотличимых для человека. Если научиться разбивать буквы на классы неотличимости, то можно заменить все буквы в классе на одну. Человек не заметит изменений в изображении, а размер файла сильно уменьшится.

При помощи алгоритмов, разработанных автором, размер файла уменьшается на 55 ± 5% (для страниц, содержащих только текст; рисунки и диаграммы уменьшают этот показатель, так как занимают много места и не сжимаются). Единственная некоммерческая программа для ЛВ2-сжатия, cjb2 из DjVuLibre, достигает уменьшения размера только на 30 ± 5%.

Теперь опишем общую схему алгоритма классификации букв. Предположим, что имеется алгоритм для сравнения пары букв (сам алгоритм будет описан дальше). По паре букв он выдает один из трех ответов — «да», «нет» или «может быть». Ответ «да» означает, что буквы, по мнению алгоритма, эквивалентны; ответ «нет» означает, что алгоритм считает буквы разными, ответ «может быть» означает, что алгоритм не уверен, но лучше считать их разными во избежание ошибок.

Возможность «нет» существует только для ускорения. Если алгоритм вместо «нет» будет всегда говорить «может быть», то результат не изменится, но скорость упадет (в программе скорость падает в 4,5 раза).

Будем считать, что алгоритм не ошибается, когда говорит «да», потому что в противном случае никакой алгоритм классификации не застрахован от ошибок.

Рассмотрим все буквы в тексте и применим к каждой паре сравнение. Теперь объявим классами связные компоненты относительно ответа «да». Это и будет тот ответ, который выдает программа. Однако программа работает быстрее за счет того, что классов обычно намного (в среднем в 5-10 раз) меньше, чем букв. Кроме того, она учитывает ответы «нет».

Программа работает следующим образом. Будем классифицировать буквы по одной, сравнивая каждую букву с ранее просмотренными. При этом ранее просмотренные буквы уже разбиты на классы.

Итак, возьмем новую букву и по очереди сравним ее со всеми имеющимися классами. Сравнение буквы с классом устроено так: буква сравнивается со всеми элементами класса до получения первого ответа «да» или «нет», который и объявляется результатом сравнения буквы с классом. Если все сравнения дали «может быть», то результатом сравнения буквы с классом будет «нет».

Теперь объединим букву и все классы, давшие ответ «да», в один класс, после чего перейдем к следующей букве.

Алгоритм сравнения пары букв должен не ошибаться, если он говорит «да» или «нет»; при этом алгоритм тем лучше, чем меньшее число раз он говорит «может быть».

**Основной алгоритм сравнения пары букв**

В DjVuLibre 3.5 используется простой алгоритм сравнения пары букв. Изображения накладываются одно на другое и считается количество отличающихся пикселей. Если это количество не превышает 6% от их общего числа (в одной из картинок), то буквы считаются одинаковыми. Порог в 6% подбирался экспериментально и, кстати, иногда дает ошибки, отождествляя буквы «и» и «н».

У этого алгоритма есть недостаток: он никак не учитывает специфику искажений, вносимых сканером. Обычно сканер не меняет произвольные пиксели на противоположные, а искажает контуры изображения. Поэтому, алгоритм можно улучшить, если оценить для каждого пикселя вероятность изменения. Эта вероятность больше у границы буквы и меньше в ее середине.

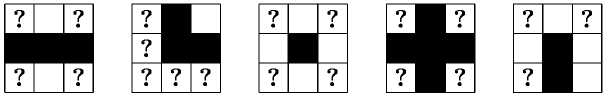
Предлагается способ, присваивающий каждому черному пикселю изображения уровень важности; чем важнее пиксель, тем меньше вероятность, что сканер заменит его на белый. Если в паре картинок из двух соответствующих друг другу пикселей один черный, а другой белый, то паре картинок начисляются штрафные очки, пропорциональные уровню важности черного пикселя. Таким образом, влияние пограничных пикселей на сумму штрафных очков уменьшается.

Способ состоит в следующем: определяется некоторое преобразование черно-белого изображения — «очистка». В ходе очистки белые пиксели остаются белыми. После нескольких последовательных очисток изображение перестает меняться. Черные пиксели, выжившие все очистки, объявляются наиболее важными; важность остальных черных пикселей уменьшается в геометрической прогрессии по мере того, на сколько очисток меньше они выдержали.

Каждая очистка выполняется в два прохода. При каждом проходе последовательно (строчки идут сверху вниз, в строчке слева направо) просматриваются все пиксели изображения.

Во время первого прохода некоторые черные пиксели могут быть объявлены кандидатами на удаление. Во время второго прохода некоторые кандидаты могут быть удалены, то есть перекрашены в белый.

Во время первого прохода пиксель объявляется кандидатом на удаление, если он и его восемь соседей не раскрашены ни одним из следующих пяти способов с точностью до поворотов и отражений:



Знаки вопроса означают, что пиксель может быть любого цвета. Таким образом, картинка с четырьмя знаками вопроса эквивалентна шестнадцати отдельным раскраскам.

В случае если пиксель лежит на границе изображения, его соседи, лежащие за пределами изображения, считаются белыми.

Первые два правила означают, что связность буквы относительно переходов от пикселя к одному из его четырех соседей не должна нарушаться.

Третье правило запрещает удаление одиночных пикселей.

Четвертое правило означает, что только пиксели с границы черного и белого могут быть удалены.

Пятое правило помогает сохранить форму буквы при очистке, запрещая удалять «выступающие» пиксели.

Во время второго прохода кандидаты на удаление перекрашиваются в белый цвет, если эти пиксели и их восемь соседей не раскрашены одним из способов, показанных выше на первых четырех рисунках. Правило сохранения формы теряет смысл, если часть изображения уже очищена, а часть — нет.

Пиксели, ставшие белыми, участвуют в последующих тестах уже как белые. Это гарантирует сохранение связности в изображении.

Время, требуемое для одной очистки, пропорционально числу пикселей в изображении. Время, требуемое для работы всего алгоритма, больше во столько раз, сколько требуется очисток. Их число заранее ограничено числом черных пикселей в изображении плюс один (после очистки либо хотя бы один черный пиксель станет белым, либо все кончится). На практике, однако, их число приблизительно равно половине характерной толщины линии. Если в изображении есть большой черный участок, то алгоритм будет работать долго, поэтому в программе алго­ритм применяется к каждой букве по отдельности.

В программе используются два теста, основанные на этом алгоритме. Численные значения порогов подобраны экспериментально.

В первом тесте знаменатель геометрической прогрессии, по которой убывает уровень важности черных пикселей, равен 0, то есть учитываются только самые важные пиксели. Пусть максимальный уровень важности равен 1, тогда если число штрафных очков меньше 2,1% от площади изображения (считая и белые и черные пиксели), то они признаются одинаковыми; если больше 5%, то заведомо разными.

Во втором тесте знаменатель геометрической прогрессии равен 0,85. Если штраф меньше 3,1% площади, изображения считаются одинаковыми, если больше 7,8% — заведомо разными.

**Быстрый отказ для пары разных букв**

Для того, чтобы ускорить программу сравнения пары букв, весьма полезно уметь быстро отбрасывать пары разных букв.

Предлагается алгоритм, который генерирует по каждому изображению короткое слово — «подпись». Подпись может иметь любой наперед заданный размер; сейчас в программе ее длина равна 31 байту.

Подписи можно интерпретировать как точки в многомерном евклидовом пространстве и находить расстояние между ними. Абсолютно одинаковые изображения, естественно, попадут в одну точку, а похожие буквы будут близки. Если расстояние слишком большое, то буквы можно считать различными.

Алгоритм работает следующим образом. Прямоугольник буквы разрезается на две части по горизонтали, но не обязательно пополам, а так, чтобы число черных пикселей по обе стороны разреза было примерно равным. Если ни одного черного пикселя не было, то разрез проводится посередине. Затем каждый из двух получившихся прямоугольников разрезается надвое вертикальным разрезом по тому же правилу: число пикселей по разные стороны разреза должно быть примерно равным. Затем каждый из четырех прямоугольников разрезается по горизонтали и так далее, горизонтальные и вертикальные разрезы чередуются.

Положению каждого разреза соответствует число от 0 до 1: 0 соответствует крайнему левому или крайнему верхнему положению, 1 — крайнему правому или крайнему нижнему. Каждый разрез порождает два прямоугольника, которые тоже могут быть разрезаны. Таким образом, получается двоичное дерево, в каждой вершине которого стоит число от 0 до 1. Для хранения в байте числа можно перенормировать в интервал от 0 до 255 и округлить.

Дерево разрезов укладывается в одну последовательность по следующему правилу: весь прямоугольник изображения соответствует первому числу в последовательности; если какой-то прямоугольник соответствует элементу номер п, то прямоугольники, которые из него получились разрезом, соответствуют элементам номер 2п и 2п + 1.

Чтобы уменьшить вес приграничных пикселей и улучшить точность алгоритма, его можно комбинировать с алгоритмом определения важности пикселей, описанным в предыдущем разделе. В этом случае алгоритм проводит разрез так, чтобы сравнять сумму уровней важности черных пикселей по обе стороны разреза.

Изложенный алгоритм основан на следующей идее. Предположим, что имеется два одномерных массива (например, звуки). Их можно представить как две непрерывные функции, / и д. Необходимо определить расстояние между этими функциями, и желательно, чтобы чем вероятнее был переход одного образца в другой под действием каких-то естественных искажений, тем меньше было определяемое расстояние. Естественно очертить круг допустимых элементарных искажений, присвоить каждому из них некую стоимость, и определить расстояние между / и д как минимальную стоимость набора элементарных искажений, необ­ходимую, чтобы превратить / в д.

Если считать элементарными искажениями сдвиги точек графика по вертикали, то расстоянием станет что-то вроде интеграла от |/ — д\\ точный вид зависит от стоимости, приписываемой элементарным искажениям. Однако вертикальное смещение всего графика на 1 обычно считается меньшим искажением, чем наложение случайного шума, по модулю не превосходящего 0,5. Значит, одинаковое смещение большого числа точек стоит дешевле, чем смещение каждой точки по отдельности.

Алгоритмы, использующие вейвлеты, учитывают это обстоятельство. В простейшем варианте разложение по вейвлетам строится так. Сначала в первый вейв-лет-коэффициэнт записывается среднее значение по всему отрезку. Затем отрезок разрезается на две равные части, и разница в среднем значении в левой и правой части записывается во второй вейвлет-коэффициэнт. Этот процесс разрезаний пополам и выписывания разности в средних продолжается дальше с каждым из полученных отрезков.

Изменение одного вейвлет-коэффициэнта приводит к одинаковому смещению целой группы точек. Таким образом, разница в вейвлет-коэффициэнтах представляет лучшее приближение к стоимости естественных искажений, чем разница в отдельных точках.

Однако сканер обычно искажает образцы по-другому: он искажает еще и аргументы функций, то есть смещает точки на графике горизонтально. Такие искажения соответствуют представлению f = g о h, причем мера этих искажений тем больше, чем дальше h от тождественной функции. Возможно, интеграл от (h! — I)2 хорошо приближает квадрат искомого расстояния; но горизонтальные искажения обычно смешаны с вертикальными, и поэтому нужен способ, позволяющий находить приближение устойчиво к вертикальным искажениям.

Предлагаемый способ — не что иное, как модификация алгоритма с вейвлетами под измерение горизонтальных искажений. Отрезок делится на две части, не обязательно ровно пополам, но так, чтобы интегралы по двум половинам были равны; затем делится еще раз, и так далее. Выше описан тот же самый алгоритм в двумерном варианте.

В программе при подсчете разности элементы подписи дополнительно умножа­лись на коэффициэнт, в геометрической прогрессии зависящий от уровня разреза. Подобранные экспериментально пороги такие: отказ от пары букв, если расстояние больше 170 (знаменатель 0,9, с уровнями важности), либо если расстояние больше 250 (знаменатель 1,15, с уровнями важности), либо если расстояние больше 215 (без прогрессии и без уровней важности).

**Получение монохромных изображений**

Алгоритм JB2 умеет сжимать только монохромные (черно-белые, без промежуточных серых цветов) изображения. Файлы формата DjVu могут содержать и изображения в тонах серого, и цветные рисунки, но все равно в таком файле отдельно хранится монохромный рисунок, а отдельно добавочная информация о цвете. Поэтому чтобы сжать в таком виде любое изображение, содержащее текст, необходимо, сначала выделить из него монохромное.

Самый простой способ для этого — объявить все пиксели темнее половины белого черными, а остальные — белыми. Под Linux это выполняется командой pgmtopbm -threshold, а под Windows эту операцию выполняет стандартный редактор Paint, когда сохраняет цветные картинки как монохромные.

Этот способ хорошо работает только тогда, когда изображение уже похоже на монохромное. Однако, если в нем есть слишком бледные буквы или затемненные поля, то в результате одни контуры могут пропасть, а другие — появиться. Чтобы справляться с такими случаями, нужен алгоритм, который обращает больше внимания на резкость границ и правильность линий, чем на абсолютную величину цвета.

Предлагаемый алгоритм — шаг в этом направлении: он стремится выделить наиболее резкие контуры; при этом в качестве побочного эффекта удаляются черные поля вокруг отсканированного листа и между страницами книги.

Итак, сначала в изображении выделяются контуры — связные компоненты линий уровня (яркости), у которых уровень кратен заранее выбранной величине. В программе эта величина составляет одну четверть полного расстояния между белым и черным. При этом мы будем считать, что эта яркость проходит между уровнями пикселей, и таким образом, контур проходит между пикселями, а не через них. Топологически каждый контур эквивалентен окружности, и у него есть бесконечная внешняя и конечная внутренняя часть. Контур может быть осветляющим (внутри ярче) и затемняющим (внутри темнее).

Для каждого контура подсчитывается «резкость» — величина, равная по опре­делению модулю суммы перепадов яркости по всем ребрам контура (ребро — это часть контура, проходящая между двумя пикселями). Перепад яркости у ребра равен разности яркости того пикселя, который внутри контура, и того, который снаружи. При этом для контуров, проходящих по краю всего изображения, будем считать пиксели, лежащие вне этого края абсолютно черными. При сканировании у краев рисунка обычно возникают черные поля; такое решение искусственно делает внешние контуры этих полей осветляющими и уменьшает их резкость.

Контуры могут быть вложены друг в друга (точнее, один во внутренность другого). Будем считать, что есть фиктивный контур, проходящий по краю изображе­ния, в который вложены все остальные. Тогда контуры образуют ориентированное дерево по вложенности, и фиктивный контур — его корень.

Раскрасим контуры в черный и белый цвета так, чтобы максимизировать описанную ниже функцию. При этом каждый пиксель получит тот цвет, который имеет самый внутренний из содержащих его контуров.

Построим функцию выигрыша от раскраски дерева контуров, которую надо устремить к максимуму. При этом можно запретить некоторые раскраски, присвоив им специальное значение «минус бесконечность».

Положим выигрыш равным сумме по всем контурам следующей величины. Фиктивный корневой контур должен быть белым (этим и достигается удаление полей); таким образом, черный корневой контур запрещен. Контур, имеющий тот же цвет, что и родитель, дает нулевой вклад в выигрыш. В следующих случаях контуру запрещено иметь цвет, отличный от цвета родителя: если отдельный алгоритм для фильтрования ненужных контуров признал его мусором; если контур осветляющий, а родитель белый; или если контур затемняющий, а родитель черный. Если же родитель затемняющего контура белый, а сам контур черный, или родитель осветляющего контура черный, а сам контур белый, то его вклад в выигрыш равен резкости.

Для того чтобы найти раскраску с максимальным выигрышем, используются два прохода по дереву. Во время первого прохода — от листьев к корню — для каждой вершины ищется максимально возможный выигрыш от ее поддерева (включая ее саму) при условии, что ее родитель белый; и то же самое при условии, если ее родитель черный. Чтобы найти каждую из этих величин, разбираются два случая: вершина белая или вершина черная; наибольшим выигрыш в этих двух случаях даст искомую величину. Если и цвет родителя, и цвет самой вершины известен, то максимальный выигрыш в ее поддереве равен ее вкладу в выигрыш плюс сумма по всем потомкам выигрыша на их поддеревьях при известном цвете их родителя.

Во время второго прохода — от корня к листьям — по найденным величинам для каждой вершины проставляются окончательные цвета при известном цвете родителя так, чтобы выигрыш от поддерева каждой вершины была наибольшим.

Алгоритм для устранения мусора — часть, требующая наибольшей ручной настройки. Максимальный уровень яркости будем считать равным 255. В программе принято следующее правило: контур считается подозрительным, если он затемняющий и его уровень (как линии уровня) меньше 156 или если он осветляющий и его уровень больше 100; подозрительный контур считается мусором, если его резкость меньше 10000 или отношение его резкости к его длине меньше 100.

Время работы всего алгоритма пропорционально числу пикселей; время работы второй части, раскраски дерева, пропорционально числу контуров. На практике программа работает быстро, как ни странно, даже быстрее, чем pgmtopbm -threshold.

**Ссылки**

[1] Сайт DjVuLibre: http://djvu.sourceforge.net

С него можно скачать архиватор и программу для просмотра DjVu под Unix с исходным кодом.

[2] Сайте DjVuZone: http: //www. dj vuzone. org Разные ресурсы, посвященные DjVu.

[3] Сервер Any2DjVu: http://ciny2djvu.djvuzone.org

Сжимает присланные на него изображения с качеством коммерческих программ.

[4] Сайт фирмы LizardTech: http://www.lizardtech.com

С него можно скачать plug-in к Internet Explorer'y для Windows. Последняя версия на момент написания этой ссылки занимает 7,5 Мб.

[5] Plug-in на МЦНМО: http://www.mccme.ru/free-books/4djvu/djvuwebbrowserplugin\_en.exe Сервер МЦНМО хранит старую версию того же plug-in'a для Windows, занимающую 1,7 Мб.

[6] Статьи, выложенные фирмой LizardTech (в том числе все нижеперечисленное) можно скачать отсюда: http://www.lizardtech.com/solutions/doc/doc\_techpapers.php

[7] Artem Mikheev, Luc Vincent, Mike Hawrylycz, Leon Bottou. Electronic Document Publishing using DjVu. Proceedings DAS'02, Fifth IAPR International Workshop on Document Analysis Systems, Princeton, NJ, August 2002.

[8] Patrick Haffher, Leon Bottou, Yann LeCun, Luc Vincent, A General Segmentation Scheme for DjVu Document Compression. Proceedings ISMM'02, International Symposium on Mathematical Morphology, Sydney, Australia, April 2002.

[9] Leon Bottou, Patrick Haffner, Yann LeCun, Efficient Compression of Digital Documents to Multilayer Raster Formats. Proceedings ICDAR'Ol, International Conference on Document Analysis and Recognition, Seattle, WA, September 2001.

[10] Yann LeCun, Leon Bottou, Patrick Haffner, Jeffery Triggs, Luc Vincent, Bill Riemers, Overview of the DjVu Document Compression Technology. Proceedings SDIUT'Ol, Symposium on Document Understanding Technologies, pp.119-122, Columbia, MD, April 2001.

[11] Yann LeCun, Leon Bottou, Andrei Erofeev, Patrick Haffner, Bill Riemers, DjVu Document Browsing with On-Demand Loading and Rendering of Image Components. Proceedings of SPIE's Internet Imaging II, San Jose, CA, February 2001.

[12] Leon Bottou, Patrick Haffner, Yann Le Cun, Paul Howard, Pascal Vincent, DjVu: Un Systeme de Compression d'Images pour la Distribution Reticulaire de Documents Numerises. Actes de la Conference Internationale Francophone sur l'Ecrit et le Document, Lyon, France, July 2000.

[13] Patrick Haffner, Yann Le Cun, Leon Bottou, Paul Howard, Pascal Vincent. Color Documents on the Web with DjVu. Proceedings of the International Conference on Image Processing, Vol. 1, pp. 239-243, Kobe, Japan, October 1999.

[14] Patrick Haffner, Leon Bottou, Paul Howard, Yann Le Cun. DjVu: Analyzing and Compressing Scanned Documents for Internet Distribution. Proceedings ICDAR'99, International Conference on Document Analysis and Recognition, pp. 625-628, 1999.

[15] Yann Le Cun, Leon Bottou, Patrick Haffner, Paul Howard, DjVu: a Compression Method for Distributing Scanned Documents in Color over the Internet, Color 6, 1ST, 1998.

[16] Leon Bottou, Patrick Haffner, Paul Howard, Patrice Simard, Yoshua Bengio, Yann Le Cun, Browsing Through High Quality Document Images with DjVu. Proceedings of IEEE Advances in Digital Libraries'98, IEEE, 1998.

[17] Leon Bottou, Patrick Haffner, Paul Howard, Patrice Simard, Yoshua Bengio, Yann Le Cun, High Quality Document Image Compression with DjVu. Journal of Electronic Imaging, Vol. 7, No. 3, pp. 410-425, SPIE, 1988.

[18] Leon Bottou, Steven Pigeon, Lossy Compression of Partially Masked Still Images. Аннотация в Proceedings of IEEE Data Compression Conference DCC'98, pp. 528-537, Snowbird, March 1998; полная версия по адресу http://www.lizardtech.com/solutions/doc/techpapers/1998\_lossy\_masked.dj vu

[19] Leon Bottou, Paul Howard, Yoshua Bengio, The Z-Coder Adaptive Binary Coder. Proceedings of IEEE Data Compression Conference DCC'98, pages 13-22, Snowbird, March 1998.