Курсовой проект: Архиватор (Arithmetic + LZ77)

Выполнил студент группы 08-208Б МАИ *Лисовский Олег Романович.*

Условие

Необходимо реализовать два известных метода сжатия данных для сжатия одного файла.

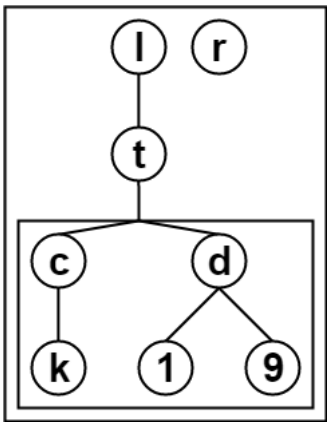
Формат запуска должен быть аналогичен формату запуска программы gzip, должны быть поддержаны следующие ключи: -c, -d, -k, -l, -r, -t, -1, -9. Должно поддерживаться указание символа дефиса в качестве стандартного ввода.

Метод решения

Как и требуется в условии запуск программы аналогичен запуску утилиты gzip: ./main <ключи> <файлы> <ключи> <файлы> ...

Обработка входных данных

Первоначально программа обрабатывает то, что получает на входе, отделяя имена файлов и директорий от введённых ключей. Ключи от файлов программа отделяет по первому символу слова: если этот символ является «-», то полученное слово является набором ключей. В процессе изучения поведения утилиты gzip я вывел следующее дерево приоритетов ключей:



Т. е. если уже был введён ключ -t, то далее ключи -c, -k, -d, -1, -9 программа учитывать не будет, но если вы ввели ключ -l, то он делает ключи -c, -k, -d, -1, -9, -t недействительными как сейчас так и при их будущих вводах. В то же время сочетания -cd, -c1, -c9, -kd, -k1, -k9, а так же сочетания ключа -r со всеми остальными ключами не являются взаимоисключающими. Если же во время обработки ключей встречается неизвестный ключ, то программа прекращает свою работу с соответствующей ошибкой, как и утилита gzip.

Если же полученное слово не начинается с символа «-», то программа идентифицирует его как имя файла/директории и заносит в красно-чёрное дерево для последующей обработки.

Общий препроцессинг

После обработки всех полученных слов программа проверяет красно-чёрное дерево на пустоту: если оно пустое, то программа завершается с соответствующим сообщением. Если же оно не пустое, то программа начинает обрабатывать все строки в данном дереве по следующему алгоритму: проверяется принадлежит ли имя директории - если нет, то это имя файла, и в дальнейшем оно обрабатывается как файл; если же это имя принадлежит директории, то проверяется активность ключа -r: если он активен, то с данной директорией идёт работа, иначе она игнорируется, и выводится соответствующее сообщение.

Работа с директорией сводится к получению всех имен файлов и директорий из неё, кроме имён «.» «..» - они пропускаются. Работа с директориями повторяет вышеописанную.

Работа с файлами во многом зависит от введённых ключей.

Препроцессинг компрессии

Если введённые ключи говорят о необходимости компрессии, то при отсутствии ключа -c производится проверка на наличие у файла суффикса «.gz». Если он присутствует, то файл не обрабатывается, и выводится соответствующее сообщение. Если же у этого файла нет расширения «.gz», то при отсутствии ключа -c проверяется наличие файла, имя которого отличается от полученного только стоящим в конце суффиксом. Если такой файл не найден, то работа продолжается. Если же он найден, то пользователю предлагается сделать выбор - перезаписывать данный файл или нет. В случае отрицательного ответа работа с данным файлом прекращается. Далее в зависимости от наличия ключей -1 и -9 определяется степень сжатия. Далее поочерёдно происходит компрессия файла с помощью обоих алгоритмов компрессии (они будут описаны ниже вместе с алгоритмами декомпрессии). В случае провала любого алгоритма компрессии работа с файлом прекращается, и выводится соответствующая ошибка. Если же оба алгоритма сработали нормально, то при отсутствии ключа -c выбирается временный файл, созданный алгоритмами, с меньшим размером, и именно он получает расширение «.gz», а файл с большим размером удаляется. Так же проверяется наличие ключа -k. При его отсутствии изначальный файл удаляется.

Препроцессинг декомпрессии

Если же ключи указывают на необходимость декомпрессии, то в случае отсутствия ключа -c и наличии ключа -d проверяется наличие у файла суффикса «.gz». При выполнении всех этих условий работа с файлом прекращается, и выводится соответствующее

сообщение. Далее при отсутствии ключей -t и -c проверяется наличие файла с таким же именем, но без расширения «.gz». Если такой файл есть, то пользователю предлагается сделать выбор о его перезаписи. В случае отказа, работа с данным файлом прекращается, и выводится соответствующее сообщение. Если ответ положительный или такого файла нет, то работа продолжается. Из поступившего архива считывается первый байт, который содержит указание на метод архивации. Если этот байт интерпретируется как символ «7» или «A», то декомпрессия производится по алгоритму LZ77 или Арифметика соответственно. Если первый байт интерпретируется иначе, то работа с файлом прекращается, и выводится соответствующее сообщение. В случае неудачного завершения алгоритма, выводится соответствующее сообщение и при отсутствии ключей -t и -c удаляется файл, в который записывались данные после декомпрессии, и работа с файлом прекращается. Далее при отсутствии ключей -t, -c и -k, удаляется изначальный архив, а при отсутствии ключей -t и -c временный файл для декомпрессии переименовывается и получает имя изначального архива без расширения «.gz».

Получение информации об архиве

В случае указания ключа -l производятся следующие действия. Читается первый байт, и в случае если он не совпадает с буквами, указывающими на метод архивации, то работа прекращается, и выводится соответствующее сообщение. Далее читается 8 байт, в которые помещается размер файла до компрессии. После программа считывает размер архива, и вычисляется процент сжатия. Далее выводятся размер сжатого файла, размер до компрессии, процент сжатия в полуинтервале $[-100\%; 100\%)$ и имя файла до архивации (если файл имеет расширение «.gz», то имя выводится без этого расширения, в противном случае выводится имя архива).

Далее незакодированный файл будет упоминаться как файл, а закодированный файл как архив.

LZW компрессия

Компрессия методом LZW происходит по следующему принципу: определяется верхняя граница буфера, в котором будут храниться слова, и из размера файла определяется, каким количеством байт будут кодироваться слова, после чего строится префиксное дерево из всех односимвольных слов символов ASCII. Дальнейший процесс компрессии будет описан для случая записи в архив, но процесс вывода результатов в стандартный вывод аналогичен (как и в случае декомпрессии). В архив записывается 9 байт информации, первый из которых - это указание метода архивации, а остальные 8 - размер изначального файла. Если поступивший файл имеет размер 0, то компрессия прекращается. Далее читается первый символ файла, и в архив записывается код соответствующего слова. Полученный символ указывает на узел, в который будет добавлен следующий символ. Затем символы считываются до создания новой вершины в префиксном дереве, а в архив записывается код вершины, предшествующей новой. Последняя буква, полученная до добавления вершины заносится в буфер. После чего

из корня ищется вершина, к которой ведёт эта буква, и процесс повторяется вплоть до окончания символов в файле или создания максимального количества вершин, которое сможет прочитать декомпрессор. Если произошло второе, то в архив записывается 0, (никак иначе на этом этапе он быть записан не может), ранее установленным количеством байт, из префиксного дерева удаляются все вершины кроме корневой и потомков первого рода, после чего процесс компрессии начинается заново, но позиции в исходном файле и архиве не получают откат. Если на каком либо этапе компрессии возникает ошибка, его работа прекращается, и выводится соответствующая ошибка.

LZW декомпрессия

Декомпрессия начинается с прочтения размера изначального файла из архива, который необходим для определения нужно количества байт для прочтения слова и проверки на безошибочность декомпрессии. Далее, из архива считываются только коды слов определённого ранее размера. После считывания первого слова создаётся красно-чёрное дерево, и в него записываются все односимвольные слова из ASCII символов. Далее, по полученному коду в дереве находится необходимая строка, и она записывается в файл. После чего этот символ записывается во временное слово. Далее алгоритм считывает коды из архива. При обработке кодов возможны 4 ситуации:

1. Код входит в список полученных слов. В этом случае в красно-чёрном дереве ищется слово с необходимым кодом, и это слово записывается в файл, после чего в красно-чёрное дерево записывается новое слово, которое является предыдущим словом, к которому добавили первую букву только что полученного. Декомпрессия продолжается.

2. Код не входит в красно-чёрное дерево, но он является следующим по счёту, следовательно это слово можно интерпретировать как предыдущее, к которому дописали в конце букву, с которой оно начинается. Полученное слово записывается в файл и в красно-чёрное дерево, после чего декомпрессия продолжается.

3. Код не входит в красно-чёрное дерево и не является следующим на подходе, следовательно архив повреждён. Декомпрессия прекращается, и выводится соответствующее сообщение.

4. Код равен 0. Красно-чёрное дерево очищается, и процесс декомпрессии начинается сначала, но позиции в файле и архиве не получают откат.

По окончании чтения архива, количество байт, которое было в изначальном файле, сверяется с тем, сколько было записано в его новую версию. При несовпадении выводится соответствующее сообщение, и декомпрессия завершается неудачно.

Описание файлов программы

Код программы разбит на файлов:

1. Arithmetic.h - Содержит перечисление методов и описание класса TArithmetic, необходимого для работы арифметической компрессии и декомпрессии.
2. Arithmetic.cpp - Содержит реализацию всех методов класса TArithmetic.
3. BFile.h - Содержит перечисление методов и описание класса TOutBinary и класса TInBinary, необходимых для записи в файл и чтения из файла соответственно.
4. BFile.cpp - Содержит реализацию всех методов классов TOutBinary и TInBinary.
5. Globals.h - Содержит в себе все необходимые глобальные переменные и библиотеки используемые несколькими файлами.
6. LZ77.h - Содержит перечисление методов и описание класса TLZ77, необходимого для работы алгоритма LZ77.
7. LZ77.cpp - Содержит реализацию всех методов класса TLZ77.
8. LZW.h - Содержит перечисление методов и описание класса TLZW, необходимого для работы алгоритма LZW.
9. LZW.cpp - Содержит реализацию всех методов класса TLZW.
10. main_help.h - Содержит в себе перечисление и описание всех функций необходимых для препроцессинга перед началом работы алгоритмов компрессии и декомпрессии.
11. main_help.cpp - Содержит реализацию всех функций, необходимых для препроцессинга, описанных в файле main_help.h.
12. TPrefix.h - Содержит перечисление методов и описание класса TPrefix, необходимого для работы LZW компрессии.
13. TPrefix.cpp - Содержит реализацию всех методов класса TPrefix.
14. main.cpp - Содержит в себе алгоритм чтения файлов и ключей.
15. Makefile - Файл для сборки программы.

Основные типы данных

1. TArithmetic - класс, описывающий работу арифметического алгоритма компрессии и декомпрессии.
2. TOutBinary - класс обеспечивающий запись необходимого количества байт в файл.
3. TInBinary - класс обеспечивающий считывание необходимого количества байт из файла.
4. TLZ77 - класс, описывающий работу алгоритма LZ77.
5. TLZW - класс, описывающий работу алгоритма LZW.
6. TPrefix - класс, обеспечивающий построение префиксного дерева для LZW сжатия.
7.

Описание методов и функций программы

Основные свойства и методы класса TArithmetic

public:

1.
2.
3.
4.
5.
6.
7.
8.

private:

1.
2.
3.

4.

5.

6.

7.

8.

Основные свойства и методы класса TOutBinary

public:

1.

2.

3.

4.

5.

6.

7.

8.

private:

1.

2.

3.

4.

5.

6.

7.

8.

Основные свойства и методы класса TInBinary

public:

1.
2.
3.
4.
5.
6.
7.
8.

private:

1.
2.
3.
4.
5.
6.
7.
8.

Основные свойства и методы класса TLZ77

public:

1.
2.
3.
4.

5.

6.

7.

8.

private:

1.

2.

3.

4.

5.

6.

7.

8.

Основные свойства и методы класса TLZW

public:

1. TLZW(int, TInBinary*, TOutBinary*) - Конструктор класса. Передаваемое число определяет степень сжатия или говорит о декомпрессии. Так же передаются файл для чтения и файл для записи.

2. bool Compress(std::string) - Производит компрессию данных. На вход получает имя файла для компрессии. В случае успешного выполнения возвращает true, иначе false.

3. bool Decompress(std::string) - Производит декомпрессию данных. На вход получает имя файла для декомпрессии. В случае успешного выполнения возвращает true, иначе false.

4. ∼TLZW() - Стандартный деконструктор.

private:

1. TInBinary* ForRead - Файл для чтения.

2. TOutBinary* ForWrite - Файл для записи.
3. TPrefix* CompressionTree - Префиксное дерево для хранения слов при компрессии.
4. std::map<unsigned long long int, std::string> DecompressionTree - Красно-чёрное дерево для хранения слов при декомпрессии.

Основные свойства и методы класса TPrefix

public:

1. TPrefix(unsigned long long int, TInBinary*, TOutBinary*) - Конструктор для корневой вершины. Передаваемое число отвечает за значение верхней границы (Border). Так же передаются файл для чтения и файл для записи.
2. TPrefix() - Конструктор для всех прочих вершин.
3. int Update(char) - Добавление вершины из других вершин. Возвращает коды ошибок или успехов.
4. int UpdateForRoot() - Добавление вершины из корня. Возвращает коды ошибок или успехов.
5. void Clear(bool) - Очистка дерева после переполнения.
6. ∼TPrefix() - Стандартный деконструктор.

private:

1. std::vector<std::pair<char, TPrefix*>> Next - Вектор потомков вершины и путей в них.
2. unsigned long long int NumberOfWord - Номер слова в данном узле.
3. static char LastLetter - Последняя прочитанная буква. Необходима для построения нового слова.
4. static unsigned long long int NeedToRead - Вспомогательная переменная для чтения нужного кол-ва символов.
5. static unsigned long long int LastNumber - Номер следующего добавленного слова.
6. static unsigned long long int Border - Максимальная граница количества слов перед очисткой дерева.
7. static TInBinary* ForRead - Файл для чтения.
8. static TOutBinary* ForWrite - Файл для записи
9. static unsigned short int Bites - Количество байт, необходимое для кодирования слова.

Прочие функции

1. unsigned long long int CalculateSize(unsigned long long int, int) - Вычисляет и возвращает размер буфера для кодирования.
2. bool KeyManager(std::string) - Обрабатывает полученные ключи. В случае получения неизвестного ключа возвращает false, иначе true.
3. bool DifferensOfSizes(TInBinary*, std::string) - вывод для каждого файла размера сжатого, оригинального, коэффициента сжатия(%) и имя оригинального файла(ключ l). В случае повреждения. архива возвращает false, иначе true.
4. void WorkWithDirectory(std::string) - работает с директорией (ключ r).
5. void WorkWithFile(std::string) - работает с файлом (определяет наличие файла, принимает решение о компрессии или декомпрессии, выполняет прочие ключи).
6. bool IsDirectory(std::string, bool) - Проверяет, является ли файл директорией. Если файл является директорией, возвращает true, иначе false.
7. void PrintDirectoryErrors(std::string) - Уведомляет об ошибках.
8. bool IsArchive(std::string) - Проверяет, является ли файл архивом. Если файл является архивом, возвращает true, иначе false.
9. void Rename(std::string, std::string) - Изменяет название файла после успешной компрессии или декомпрессии.
10. void Delete(std::string) - Удаляет временный файл.
11. void MainDecompress(TInBinary*, std::string) - Отвечает за подготовку декомпрессинга.
12. void MainCompress(TInBinary*, std::string) - Отвечает за подготовку компрессинга.
13. unsigned long long int LZWCompress(TInBinary*, std::string, TOutBinary*, int) - Подготавливает LZW компрессию. Возвращает размер нового файла.
14. unsigned long long int LZ77Compress(TInBinary*, std::string, TOutBinary*) - Подготавливает LZ77 компрессию. Возвращает размер нового файла.
15. unsigned long long int ArithmeticCompress(TInBinary*, std::string, TOutBinary*) - Подготавливает арифметический компрессию. Возвращает размер нового файла.
16. void KeepSmall(unsigned long long int, unsigned long long int, unsigned long long int, std::string) - Сохраняет архив самого малого размера.
17. int main(int, char*) - Осуществляет чтение входных данных.

Исходный код

Дневник отладки

Тест производительности

Выводы