МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

МАТЕРИАЛЫ

VII Международной молодежной научной конференции

«МАТЕМАТИЧЕСКОЕ И ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ИНФОРМАЦИОННЫХ, ТЕХНИЧЕСКИХ И ЭКОНОМИЧЕСКИХ СИСТЕМ»

Томск, 23–25 мая 2019 г.

Под общей редакцией
кандидата технических наук И.С. Шмырина

Томск
Издательский Дом Томского государственного университета
2019

11. *Ильин В.Д., Соколов И.А.* Символьная модель системы знаний информатики в человеко-автоматной среде // Информатика и её применения. 2007. – Т. 1. – № 1. – С. 66–78.

12. *Евтушенко Н.В., Паршина Н.А., Янковская А.Е.* Автоматная модель контроля знаний в интеллектуальных обучающих системах // Математическое моделирование. Кибернетика. Информатика. – Томск: Издательство Томского университета, 1999. – С. 60–66.

13. *Yankovskaya A., Yevtushenko N.* Finite State Machine (FSM) - based Knowledge Representation in a Computer Tutoring System // New Media and Telematic Technologies for Education in East European Countries, 1996. – P. 63–69.

СИСТЕМА СЖАТИЯ ИНФОРМАЦИИ С ВЫБОРОМ ОПТИМАЛЬНОГО АЛГОРИТМА

А.А. Кожемякина, Н.Б. Буторина

Томский государственный университет

Введение

Количество информации постоянно растёт, в связи с чем нам приходится иметь дело с огромным объёмом данных, которые необходимо хранить и обрабатывать. Возможность эффективно использовать дисковое пространство рассматривалось ещё при рождении электронно-вычислительных систем. На протяжении всего времени эта проблема решалась различными алгоритмами сжатия данных.

Главный принцип сжатия основывается на том, что в любом файле, который содержит случайные данные, информация частично повторяется, а используя статистико-математические модели, можно выявить вероятность повторного вхождения определённой комбинаций символов.

В настоящее время существует множество алгоритмов сжатия информации, а также множество программ, которые используют данные алгоритмы. Но среднестатистический пользователь не владеет информацией о том, какой алгоритм более других подходит для того или иного файла. Целью нашей работы является создание приложения, которое автоматически подбирает алгоритм для выбранного пользователем файла.

Очевидно, что эффективность алгоритма должна зависеть от типа и объёма файла. Основываясь на результатах тестов, мы разработали приложение, которое, в зависимости от этих двух параметров, рекомендует пользователю оптимальный алгоритм, также предоставляя выбор приоритета по скорости или наилучшей степени сжатия.

В данной работе рассматривается исключительно сжатие без потерь. При таком сжатии закодированные данные восстанавливаются однозначно с точностью до бита. К таким алгоритмам относятся: LZMA2, BZip2, Defalte, PPMd. Именно их мы взяли за основу программы, ввиду их распространённости в наши дни и превосходством над более ранними и менее распространёнными алгоритмами современности.

Создание визуальной и функциональной оболочки программы происходило на языке программирования Python с использованием набора привязок графического фреймворка PyQt в среде программирования PyCharm от компании по производству программного обеспечения JetBrains.

Выбор этого программного обеспечения обусловлен следующими факторами:

1. Python ориентирован на повышение производительности пользователей и читаемость кода: синтаксис ядра минималистичен в то время, как стандартная библиотека содержит огромное количество полезных методов.

2. Среда разработки PyCharm обеспечивает простую организацию файлов и быструю скорость работы по сравнению с другими средами для программирования на языке Python.

3. PyQt представляет собой смесь языка программирования Python и библиотеки Qt. В то время как Qt – одна из самых мощных библиотек, которая содержит более 300 классов и почти 6000 функций и методов. Также эта библиотека является кроссплатформенной.

1. Обзор алгоритмов

LZMA2

LZMA2 является новой версией LZMA. Отличие между данными двумя алгоритмами в том, что в первом реализована многопоточность и хранение одновременно сжатый и несжатых данных, что, в свою очередь, экономит биты.

Форматом файлов, предназначенным для размещения сжатых данных, который по умолчанию использует алгоритм LZMA2, является формат .xz. В настоящее время этот формат дает возможность выбора между LZMA и LZMA2.

Алгоритм LZMA включает в себя:

1. Дельта-кодирование.
2. Алгоритм LZ77.
3. Интервальное кодирование.

Дельта-кодирование представляет данные как разницу между последовательными символами [1].

LZ77 рассматривает последовательности символов, и если какая-то последовательность встречается более одного раза, то он заменяет её повторные вхождения ссылкой на её первый экземпляр [2]. Как и во всех алгоритмах семейства LZ, в LZ77 используется словарь, хранящий последовательности, которые встречались ранее, т.е. он использует принцип «скользящего окна».

Сам словарь хранит данные о смещении, длине серий и символ расхождения. Смещение – это расстояние фразы от начала файла. Длина серии – это количество символов, которое принадлежит фразе, если считать от смещения. Символ расхождения показывает на то, что на выход пришла фраза, которая похожа на ту, которая обозначена смещением и длиной [3].

При интервальном кодировании выделяется некоторый диапазон символов, и оценивается вероятность вхождения каждого. Таким образом, этот алгоритм преобразовывает все символы сообщения в одно число.

Название алгоритма происходит от «Lempel-Ziv Markov chain Algorithm».

BZip2

Bzip2 – это утилита командной строки для сжатия данных, разработанная Джулианом Стюардом в 1996 г. При сжатии к оригинальному расширению файла добавляется суффикс .bz2. Прежде чем приступить к кодированию, выполняются предварительные преобразования.

Алгоритм состоит из этапов:

1. Деление на блоки фиксированного размера.
2. Преобразование Барроуза-Уилера.
3. Преобразование MTF.
4. Кодирование Хаффмана.

Размер блока можно задать при помощи аргументов командной строки. Такие блоки сжимаются независимо друг от друга и записываются подряд. При этом в начале каждого блока используется последовательность 0x314159265359 (в кодировке ASCII при выравнивании на границу байта отображается как «1AY&SY»), что является записью первых цифр числа пи в формате BCD. А в конце каждого блока используется запись первых цифр корня из числа пи, т.е. последовательность 0x177245385090.

Преобразование Барроуза-Уилера (или BWT) – это алгоритм, который преобразовывает исходные данные так, что на выходе повторяющиеся подстроки образуют последовательность из повторяющихся символов [2], т.е.:

- • создаётся массив строк;
- • в этом массиве сохраняются всевозможные преобразования строки, которая подаётся на вход;

• • отсортировка массива;

• • на выходе получаем последний столбец.

Преобразование MTF(move-to-front) заменяет каждый входной символ на его номер в специальном стеке, в который заносятся символы, использованные недавно.

Алгоритм Хаффмана основан на частоте появления символа в последовательности. Символ, который встречается в строке чаще всего получает самый короткий код, а символ, который встречается реже остальных – наоборот, самый длинный код. Это нужно для того, чтобы самые частые символы занимали меньше места.

Сначала вычисляется частота каждого символа. Затем создаются узлы бинарного дерева для всех знаков и добавляются в очередь, используя частоту в качестве приоритета. После достаются два первых элемента из очереди и связываются, создавая новый узел дерева, в котором они оба будут потомками, а приоритет нового узла будет равен сумме их приоритетов. Далее получившийся новый узел отправляется обратно в очередь.

Повторяя шаги, описанные выше, последовательно, получим оптимальное кодовое дерево, на основе которого строится отображение код-символа. Чтобы получить код для любого символа, надо пройтись по дереву и для каждого перехода выбрать 1, если мы идём вправо, и 0, если мы идём влево.

Deflate

Алгоритм Deflate создан Жан-Лу Галли и Марком Адлером и впервые выпущен в 1992 году. В 1993 году была создана версия 1.0. Создаёт файлы в формате .gz и комбинирует алгоритм LZ77 и кодирование Хаффмана, которые были рассмотрены выше.

PPMd

Метод сжатия PPMd – одна из версий алгоритма PPM. PPM реализует сжатие без потерь, используя статистические методы. Он основан на контекстном моделировании и предсказании. Под контекстом подразумевается множество символов в несжатом потоке, предшествующих данному, чтобы предсказывать значение символа на основе статистических данных.

Длина контекста, использующегося при предсказании, зачастую сильно ограничена. Если предсказание символа по контексту из n символов не может быть произведено, то происходит попытка предсказать его с помощью $n-1$ символов. Рекурсивный переход к моделям с меньшим порядком происходит, пока предсказание не произойдёт в одной из моделей, либо когда контекст станет нулевой длины. В случае, когда контекст становиться нулевой длины, вероятность символа определяется исключительно из частоты его появления в сжимаемом потоке данных. Для получения наилучшего значения вероятности элемента необходимо учитывать контексты разных длин.

В алгоритме сжатия PPM реализуется вариант стратегии перемешивания, т.е. оценки вероятностей, сделанные на основе анализа контекстов разных длин, формируются в одну общую вероятность. Сформировавшаяся оценка кодируется любым алгоритмом энтропийного кодирования. На этом этапе, непосредственно, и происходит сжатие.

Рассматриваемый нами алгоритм PPMd увеличивает псевдосчётчик нового символа каждый раз, когда, действительно, в потоке появляется новый символ. Другими словами, PPMd оценивает вероятность появления нового символа как отношение числа уникальных символов к общему числу используемых символов. В то время, как PPM полагают счётчик нового символа равным фиксированной величине, например, единице [4].

2. Функционал приложения

Разработанное нами приложение анализирует выбранный пользователем файл, выявляет его размер и расширение и на основе этих двух параметров делает вывод о том,

какой метод сжатия наилучшим образом подходит для данного файла по одному из приоритетов – скорости сжатия или степени сжатия.

Ниже представлен интерфейс программы, разработанный с помощью PyQt.

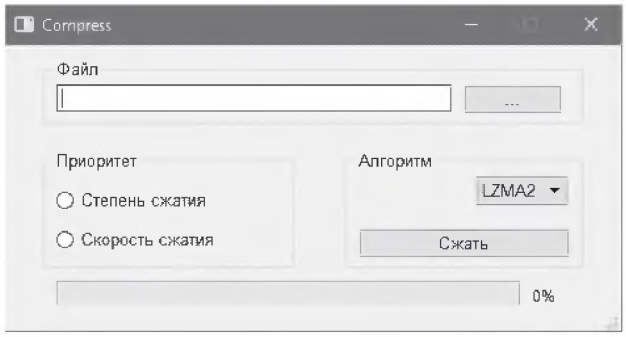

Рис. 1. Пользовательский интерфейс программы

Объекты распределены по контейнерам – элемент графического интерфейса, называемый groupBox.

Контейнер под названием «Файл» содержит кнопку, при нажатии на которую вызывается диалоговое окно выбора файла и строку, куда будет помещена директория этого файла.

Контейнер с именем «Приоритет» содержит кнопки-переключатели, которые дают пользователю выбор между приоритетами.

В контейнере «Алгоритм» расположены выпадающий список с различными методами сжатия и кнопка, которая непосредственно производит сжатие по выбранной директории и выбранному алгоритму. В выпадающем списке алгоритмы расположены в порядке убывания – от наиболее эффективному к менее эффективному.

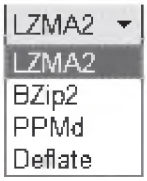

Рис. 2. Выпадающее меню

В самом низу интерфейса размещен индикатор прогресса – элемент графического интерфейса пользователя, который показывает, на сколько процентов в данный момент времени выполнена задача. В нашем случае под задачей подразумевается сжатие файла.

По завершении сжатия, программа не закрывается, предоставляя пользователю выбрать другой файл или закрыть её самостоятельно.

3. Анализ результатов тестирования по степени сжатия

Для тестирования использовались файлы разного типа: txt, doc, docx, xls, xlsx, ppt и pptx. Каждый тип условно разбили на маленький, средний и большой размер файла. Используя все алгоритмы произвели сжатие каждого файла с замером времени. Запоминался исходный размер файла и размер после компрессии, после чего на основе этих данных высчитывался коэффициент сжатия в процентах. Ключевую роль в работе играют два показателя – это скорость выполнения программы и пространство на диске, занимаемое данными. Поэтому при тестировании во внимание принимались два критерия:

1. Коэффициент сжатия.
2. Скорость сжатия.

В табл. 1 приведены результаты тестов, где первый столбец означает формат файла, второй – начальный размер файла и последующие столбцы – степень сжатия в процентах после применения алгоритмов, выполняющих компрессию.

Таблица 1

Сравнение степеней сжатия

		BZip2	Deflate	LZMA2	PPMd
txt	s	64,03	59,93	61,07	66,98
	m	71,96	65,58	69,39	74,97
	l	73,38	63,94	70,00	76,06
doc	s	24,49	26,53	27,55	25,51
	m	6,30	8,13	8,74	6,10
	l	3,49	5,08	5,48	3,39
docx	s	0,92	1,83	0,92	0,92
	m	-0,56	0,00	-1,11	-0,19
	l	-0,40	0,00	-1,20	0,00
xls	s	76,35	73,85	83,40	78,40
	m	81,75	75,91	91,97	82,48
	l	82,65	76,86	95,43	81,58
xlsx	s	2,97	9,14	10,13	6,94
	m	28,57	33,33	37,86	37,14
	l	50,81	48,11	57,84	57,30
ppt	s	36,21	39,09	41,15	38,68
	m	17,48	20,00	21,36	18,06
	l	10,02	11,61	12,40	9,62
pptx	s	6,71	9,15	9,15	6,10
	m	2,10	3,50	2,57	0,70
	l	0,66	1,55	0,55	0,00

Для более наглядного представления на основе таблицы 1 была построена следующая диаграмма:

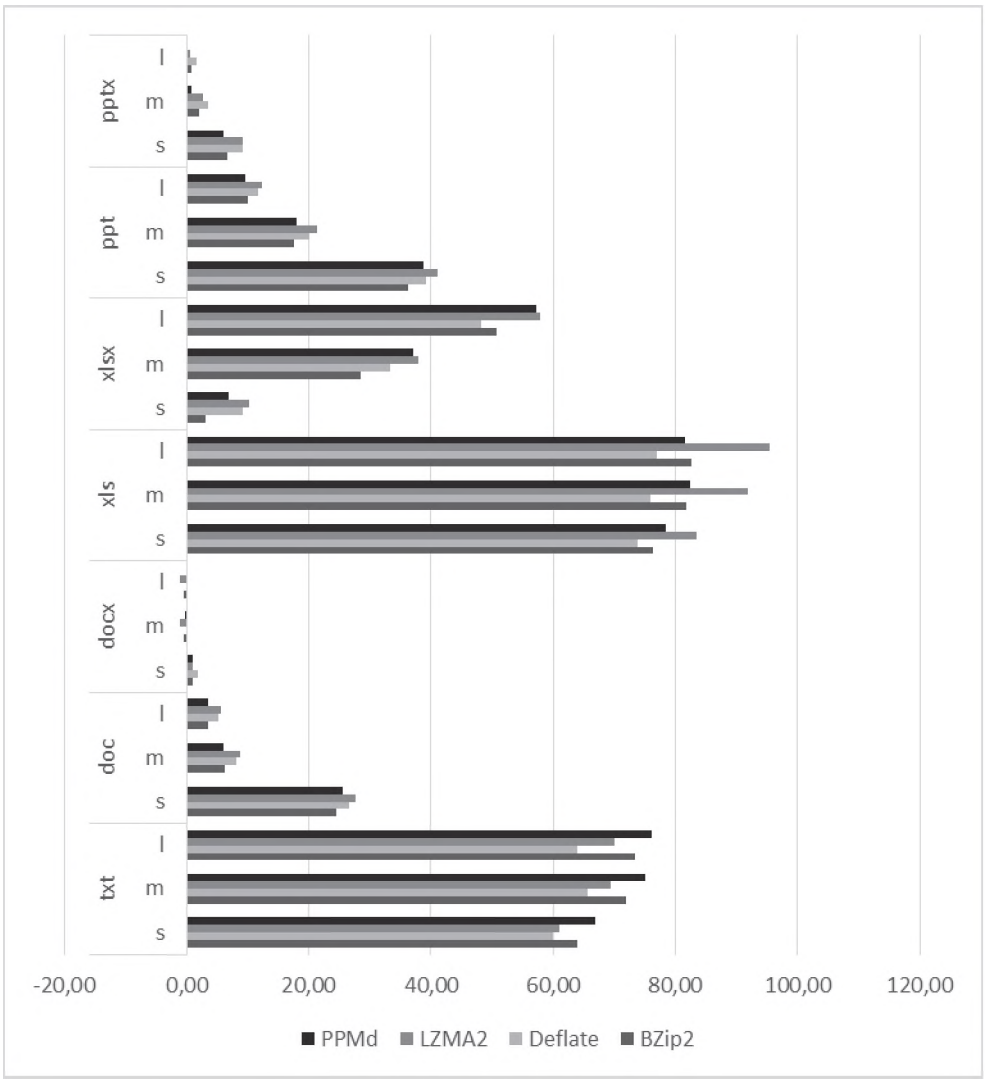

Рис. 3. Диаграмма степени сжатия

Здесь буквой «s» обозначается маленький объем файла, «m» – средний и «l» – большой. Для расширения .txt были взяты файлы размером 15, 150 и 700 килобайт. Файлы .doc и .docx тестировались на объёмах 100 килобайт, 500 килобайт и 1 мегабайт. В таблицы Microsoft Excel было занесено по 100, 1000 и 5000 строкам, что в формате .xls составило 20, 150 и 650 килобайт соответственно, а для более нового .xlsx – 10, 40 и 200 килобайт. Презентации .ppt и .pptx сжимались в размерах 250, 500 и 1000 килобайт.

Из таблицы и диаграммы мы видим, что для сжатия текстового редактора лучше всего подходит метод PPMd. Также очевидно, что LZMA2 лучше других справляется с такими файлами, как таблицы Ecxel, документы в формате .doc и презентации PowerPoint. С Microsoft Word более поздней версии, а именно в формате .docx, дела обстоят иначе – при больших размерах такие файлы могут сжать не все алгоритмы, а некоторые и вовсе увеличивают первоначальный размер. Такие алгоритмы были исключены из списка выбора метода сжатия при указании директории на подобные файлы.

4. Функция принадлежности

В процессе написания программы для определения степени принадлежности объёма файла к той или иной категории были использованы элементы нечёткой логики, а именно была использована функция принадлежности. Она получила свою популярность в связи с наличием нечётких приближённых рассуждений при описании человеком процессов, систем и объектов. Существует ряд формул для задания функции принадлежности, из которых мы выбрали треугольную как наиболее подходящую к данной задаче. Такая функция принадлежности определяется тройкой чисел (a, b, c), где $a \le b \le c$, и значение в точке x вычисляется по формуле:

$$\mu(x) = \begin{cases} 1 - \dfrac{b-x}{b-a}, & x \in [a,b], \\ 1 - \dfrac{x-b}{c-b}, & x \in [b,c], \\ 0, & x \notin [a,c]. \end{cases}$$

Здесь числа a и c означают границы интервалов объёмов, b – значение размера файла, на котором проводились тесты. Следовательно, чем ближе значение объёма выбранного файла к объёму тестированного, тем больше файл подходит под ту или иную категорию.

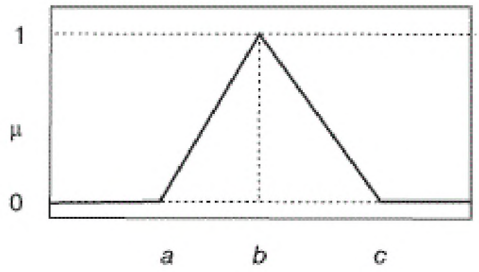

Рис. 4. Треугольная функция принадлежности

5. Анализ результатов тестирования по скорости сжатия

При рассмотрении второго приоритета – скорости сжатия, были построены следующая таблица и диаграмма, значения времени в которых указаны в секундах:

Таблица 2

Сравнение скорости сжатия

		BZip2	Deflate	LZMA2	PPMd
txt	s	0,1	0,24	0,02	0,30
	m	0,07	0,06	0,09	0,14
	l	0,11	0,1	0,25	0,34
doc	s	0,02	0,01	0,04	0,11
	m	0,1	0,03	0,11	0,27
	l	0,22	0,07	0,23	0,30
docx	s	0,04	0,01	0,05	0,07
	m	0,13	0,02	0,12	0,19
	l	0,22	0,05	0,2	0,34
xls	s	0,01	0,02	0,02	0,06
	m	0,01	0,04	0,06	0,12
	l	0,06	0,17	0,23	0,29
xlsx	s	0	0,02	0,03	0,08
	m	0,01	0,003	0,04	0,12

	l	0,17	0,35	0,14	0,40
ppt	s	0,06	0,03	0,07	0,10
	m	0,11	0,04	0,12	0,14
	l	0,44	0,5	0,37	0,56
pptx	s	0,05	0,01	0,05	0,07
	m	0,1	0,02	0,09	0,23
	l	0,27	0,07	0,19	0,38

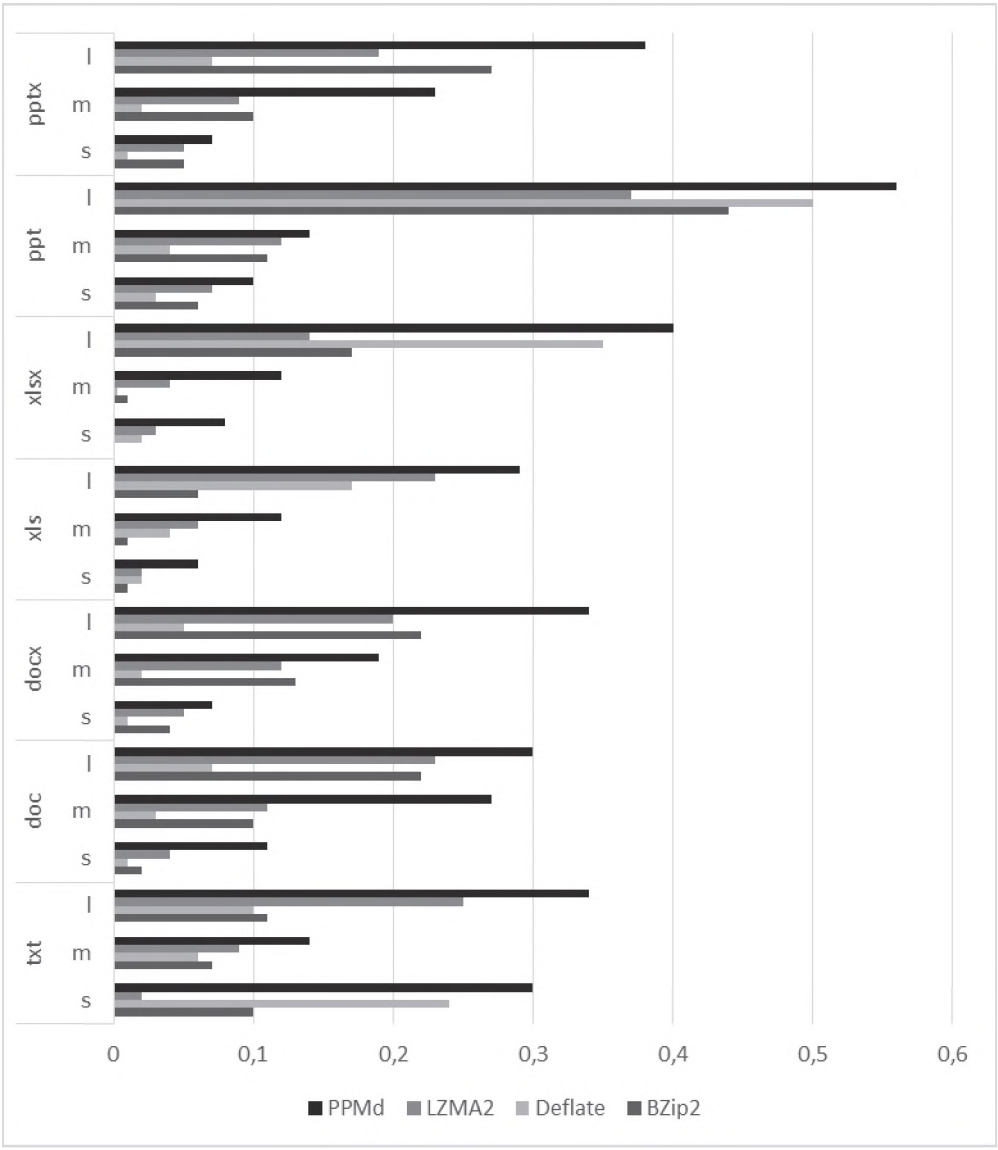

Рис. 5. Диаграмма скорости сжатия

На основе этих данных мы можем сделать вывод о том, что метод PPMd хоть и сжимает текстовые редакторы лучше остальных алгоритмов, но при этом имеет недостаток в скорости. По большей части алгоритм Deflate показывает наилучший результат по этому критерию.

Заключение

В настоящей работе были рассмотрены различные способы компрессии. Все они при тестировании дали разные показатели времени и степени сжатия. Основываясь на

результатах, представленных в таблице 1 и таблице 2, а также на рис. 3 и рис. 4, можно рекомендовать для фалов .txt алгоритм сжатия PPMd для наилучшей компрессии, но не лучшей скорости, и метод LZMA2, как лидер среди сжатия данных с таблицами, документами и презентациями. Как самый быстрый алгоритм, чаще всего показывает себя Deflate.

ЛИТЕРАТУРА

1. Академик [Электронный ресурс] : Дельта-кодирование – 2014. – URL: https://dic.academic.ru/dic.nsf/ruwiki/41976.
2. Habr [Электронный ресурс] : Алгоритмы сжатия данных без потерь, часть 2 – 2014. – URL: https://habr.com/post/235553/.
3. Habr [Электронный ресурс] : Простейшие алгоритмы сжатия: RLE и LZ77 – 2012. – URL: https://habr.com/post/141827/.
4. Wikipedia [Электронный ресурс] : Алгоритм сжатия PPM – 2018. – URL: https://ru.wikipedia.org/wiki/Алгоритм_сжатия_PPM.

ВИРТУАЛЬНАЯ МАШИНА EMERAVM

Е.П. Чалых, С.И. Самохина
Томский государственный университет

Введение

На данный момент в мире существует большое число языков программирования, каждый из них создан с определённой целью. Изначальной целью проекта было создание мощного и быстрого языка программирования с поддержкой математических вычислений, таких как дифференцирование, интегрирование, работа с матрицами и другие. Но первая проблема нового языка возникла при выборе его типа: интерпретируемый или компилируемый. Очевидный минус первого типа заключается в том, что процесс выполнения кода будет довольно долгим, особенно на очень больших файлах исходного кода. Но главное достоинство – интерпретатор языка можно сделать кроссплатформенным. Компилируемый язык программирования будет работать гораздо быстрей, т.к. из исходного кода мы получим исполняемый файл. А минус такого языка, естественно, платформозависимость.

Очевидно, для получения кроссплатформенного и быстрого языка, нужно использовать промежуточное состояние исходного кода и исполняемого файла. Такое состояние называется байт-кодом[1]. Байт-код – это последовательность байтов, каждый из которых характеризует конкретную инструкцию выполнения. Чтобы сформировать байт-код, нужен определенный транслятор, а для выполнения полученного байт-кода – определенный интерпретатор, называемый виртуальной машиной[2].

1. Постановка задачи

Первый этап создания языка программирования – это создание работоспособной, желательно кроссплатформенной, виртуальной машины. И поскольку язык будет использоваться в математических целях, виртуальная машина должна поддерживать большие математические вычисления.

Название проекта происходит от названия языка и сокращения от virtual machine – EmeraVM, или сокращенно EVM.

Кроссплатформенность можно достичь с помощью другого кроссплатформенного языка, в данном случае используется язык Kotlin[3], работающий на JVM[4] – виртуальной машине Java. Соответственно, EVM будет работать на любой операционной системе, где установлена Java. Инструментом создания EVM является мощная интегрированная среда разработки IntelliJ IDEA[5].