Отчет

к Лабораторной Работе №1

«Сжатие данных без потерь»

студентки
группы М4106
Оберган Татьяны Максимовны

В рамках данной ЛР были реализованы следующие алгоритмы:

- Двухпроходный Хаффман
- Стопка книг (MTF)
- Барроуза-Уиллера (BWT)
- LZW (необходима оптимизация?)
- DeepZip(Частично)

DeepZip (нейронные сети):

Язык: Python

Изначально я взялась за DeepZip. Мучилась с ним дня 4 не вставая.

Проблемы:

- **Миллион зависимостей** (numpy, keras, tensorflow, sklearn) и другие. Просто их установить уже проблема, не то, что запустить в одном .exe.
- Для обхода этого было принято решение переместить **проект в облако**: colab.research.google.com
    - была **перестроена большая часть кода** проекта в формат notebook
    - используемые библиотеки стремительно развиваются и **исходный проект был наполовину устаревшим** (неподдерживаемые функции итд), следовательно проведен рефакторинг, замена старых блоков кода.
- **Не хватало знаний** в области машинного обучения, чтобы исправить некоторые ошибки исходного проекта. (встал вопрос о том сколько еще времени понадобится чтобы все понять, к дедлайну я явно не успевала)
- В один момент все захотелось написать с нуля, используя собственную модель обучения, но ввиду **ресурсозатратности** обучения моделей (более 4 часов без гарантии успеха) было принято решение прекратить.

Наработки:

Мне в принципе интересна тема машинного обучения и было бы интересно все-таки реализовать этот алгоритм.

Вот такой блокнот получился для обучения модели:

https://colab.research.google.com/drive/1CEvqD-J5xWSom8XYCqpjb1Fe91clp_n8?usp=sharing

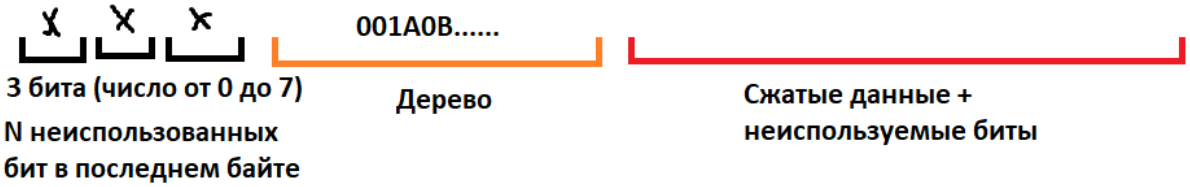
Блокнот кодирования (рефакторинг брошен на моменте, когда я осознала, что алгоритм не работает ни с маленькими, ни с большими файлами):

https://colab.research.google.com/drive/1SACZjeKUJkdWuHzekHYxVoah7_w05dkN?usp=sharing

BWT + MTF + Двухпроходный Хаффман:

Язык: C#

- Считываются все символы
- BWT
  - добавление символа конца
  - создание массива суффиксов
  - вычисление выходной последовательности
- MTF
- Двухпроходный Хаффман
  - подсчет частот
  - построение дерева
  - запись дерева
  - кодирование последовательности
  - запись информации в заголовок файла о последнем байте



LZW:

Идея: кодировать двумя способами, какой лучше сжал тот и используется, а в файл ставить флаг.

Для этого был реализован алгоритм LZW, однако на проверке показал себя хуже, чем BWT+MTF+HUFF, но лучше, чем HUFF.

Время выполнения LZW было слишком высокое (на book2 около 8 минут) и от этой идеи пришлось отказаться.

Полученные результаты:

ФАЙЛ	H(X)	H(X|X)	H(X|XX)	ИСХ. SIZE	ХАФФМАН	BWT+MTF+HUFF	LZW
BIB	5.196	3.364	2.307	111265	72863(5.239)	33187(2.386)	53724(3.863)
BOOK1	4.526	3.584	2.814	768771	438477(4.563)	267152(2.780)	241370(3.161)
BOOK2	4.791	3.745	2.735	610856	368420(5.693)	186982(2,449)	
GEO	5.626	4.264	3.458	102400	72876 (5.693)	69542(5.433)	
NEWS	5.188	4.092	2.923	377109	246516(5.229)	133500(2.832)	
OBJ1	5.853	3.464	1.400	21504	16372(6.091)	11763(4.376)	
OBJ2	6.252	3.870	2.265	246814	194416(6.302)	88712(2.875)	
PAPER1	4.969	3.646	2.332	53161	33456(5.035)	18206(2.740)	
PAPER2	4.592	3.522	3.513	82199	47729 (4.645)	28119(2.737)	
PIC	1.206	0.824	0.705	513216	106750(1.664)	101489(1.582)	
PROGC	5.184	3.603	2.314	39611	26029(5.257)	13682(2.763)	
PROGL	4.761	3.216	2.043	71646	43091(4.812)	18726(2.091)	
PROGP	4.856	3.187	1.755	49379	30325(4.913)	12809(2.075)	
TRANS	5.494	3.355	1.930	93695	65342(5.579)	22382(1.911)	
				3141626	1762662	1006251	

Вывод:

Было произведено сравнение различных алгоритмов сжатия. Наилучшим по скорости работы и степени сжатия оказался BWT+MTF+HUFF. Суммарный размер сжатых файлов в 3 раза меньше исходных. Алгоритм LZW дает результат ниже H(X), но имеет низкую скорость работы (~ в 18 раз медленнее BWT+MTF+HUFF), а степень сжатия все еще уступает BWT+MTF+HUFF.