

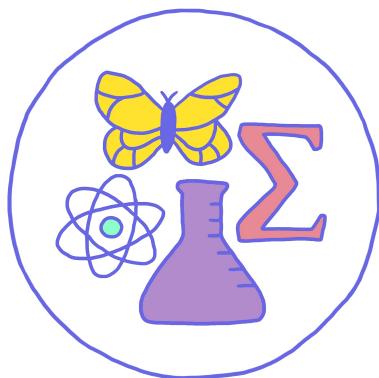
Примеры кода в main.ipynb. Текст:

## Вступление

---

Талантливый школьник во время сезона 2024 г. нарисовал новую эмблему Красноярской летней школы. Он так всем понравился, что дирекция решила отсканировать изображение и сохранить его на компьютере. Получился файл формата BMP с расширением ".bmp".

Эту картинку можно найти по ссылке 1:



Известно, что байтовое представление файла ".bmp" имеет следующую структуру.

1. BITMAPFILEHEADER — первые 14 байт.
2. BITMAPINFO — в наших задачах это всегда будут следующие 40 байт.
3. Пиксельные данные — всё остальное.

Подробную информацию про этот формат можно прочитать на [Википедии](#), но из всех этих данных нас будут интересовать в основном следующие.

1. С 10 по 14 байт записан номер байта, с которого начинается описание пиксельных данных картинки. В нашем случае это всегда 36 (54 в десятичной).
2. С 18 по 22 байт записана ширина изображения в пикселях.
3. С 22 по 26 байт записана высота изображения в пикселях. Если это число положительное, то строки пикселей перечислены снизу вверх. Если это число отрицательное, то запись строк идёт сверху вниз, причём количество строк равно модулю этого числа.
4. С 28 по 30 байт записано количество бит, которым шифруется цвет каждого пикселя. В нашем случае это всегда 3 байта (24 бита) : по одному байту на каждую цветовую компоненту.

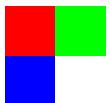
Пиксельные данные — это последовательность цветов пикселей, записанная по следующим правилам.

- Всё изображение разбито на строки высотой в один пиксель.
- Данные о пикселях строк перечислены по порядку (сначала 1-я строка, потом 2-я и т.д.).
- Порядок строк зависит от значения высоты, записанной в BITMAPINFO: сверху вниз при отрицательном значении, снизу вверх при положительном.
- Пиксели каждой строки перечислены слева направо.

- Каждый пиксель зашифрован тремя числами от 0 до 255 в палитре rgb, то есть значениями **синей, зелёной и красной** компонент цвета пикселя, **именно в таком порядке**.
- Количество байт, соответствующих каждой строке, должно делиться на 4. Если количество байт не делится на 4, то в конец каждой строки дописывается необходимое количество нулей.

## Пример

Чтобы лучше разобраться, как это работает, давайте рассмотрим следующий пример изображения из 4 пикселей (увеличено для наглядности). Изображение доступно по ссылке 2.



Это изображение занимает ровно 70 байт.

1. Первые 14 байт секции BITMAPFILEHEADER:

42	4D	46	00	00	00	00	00	00	36	00	00	00	00
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Последние 4 байта хранят позицию, с которой начнутся пиксельные данные. Чтобы правильно считать число состоящее из 4 байт, нам необходимо читать байты в "порядке от младшего к старшему" ("little-endian" или просто "little"). В нашем случае пиксельные данные начнутся с 54 ( $3*16 + 6 = 54$ ) байта.

2. Следующие 40 байт секции BITMAPINFO:

28	00	00	00	00	02	00	00	00	02	00	00	00	01	00	18	00	00	00	00	00	00	00	00
14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33				

10	00	00	00	C4	0E	00	00	C4	0E	00	00	00	00	00	00	00	00	00	00	00	00	00	00
34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53				

В этих данных мы можем найти о ширине, высоте и кодировке изображения.

- Ширина изображения равна **02** пикселя.
- Высота изображения равна **02** пикселя. 2 — положительное число, а значит строки будут записаны снизу вверх.
- Каждый пиксель кодируется 24 битами или тремя байтами

3. Оставшиеся 16 байт пиксельных данных записаны так (для удобства мы выписали строки друг под другом, в файле переноса строки нет):

Синий			Белый				
FF	00	00	FF	FF	FF	00	00
54	55	56	57	58	59	60	61
Красный			Зелёный				
00	00	FF	00	FF	00	00	00
62	63	64	65	66	67	68	69

Нижняя строка

Верхняя строка

- Как мы уже знаем, строки записаны снизу вверх, причём каждый пиксель записан 3 байтами в порядке: синий, зелёный, красный.
- Сначала идёт левый нижний пиксель — **ff 00 00** (синий).

3. Потом правый нижний пиксель — **ff ff ff** (белый).
4. Потом идут 2 байта **00 00**, которые нужны, чтобы количество байт в строке делилось на 4 ( $2 * 3 + 2 = 8$ ).
5. Потом левый верхний пиксель — **00 00 ff** (красный).
6. Потом правый верхний пиксель — **00 ff 00** (зелёный).
7. И в самом конце стоят 2 байта **00 00**, которые нужны, чтобы количество байт делилось на 4.

## Задание 1

---

**Из-за ошибки при сканировании изображение получилось перевёрнутым. Тебя попросили вернуть изображению правильную ориентацию.**

Напиши функцию (метод) `vertical_reverse_image(input_image, result_image)`, которая изменит порядок строк изображения на противоположный. У этой функции должно быть два аргумента:

- `input_image : string` — относительный или абсолютный путь к файлу BMP изображения,
- `result_image : string` — относительный или абсолютный путь к файлу, в который будет записан результат выполнения функции.

Твоя функция (метод) должна использовать только функции стандартных библиотек используемого языка программирования. При выполнении этого задания считай, что входное изображения обязательно имеет описанный выше формат.

## Задание 2

---

**Дирекция очень довольна твоей работой и захотела отправить это изображение по интернету из "Орбиты" в Красноярск, но интернет в "Орбите" не всегда стабильный, а наше изображение весит целых 7,5 мегабайт! Тебя попросили сжать это изображение так, чтобы при разжатии восстанавливалось в точности то же самое изображение.**

Давай посмотрим на верхнюю пиксельную строку нашего изображения. Она состоит из белого пикселя "ff ff ff", повторенного много раз (столько раз, сколько пикселей поместилось в одну строку изображения). Но!

Мы же можем сильно сократить запись этого изображения, если вместо имеющейся записи укажем что-то вроде "**пиксель ff ff ff стоит 1629 раз**", или просто "**1629 ff ff ff**".

Будем обозначать число повторов одним байтом. Тогда 1629 белых пикселей можно записать следующим образом: "**255 ff ff ff, 255 ff ff ff, 99 ff ff ff**".

Такая запись уже сильно сократит размер нашего файла, ведь теперь первая строка будет занимать **28 байт вместо 4887**.

1. Напиши две функции (метода): `compress(input_image, result_image)` и `decompress(input_image, result_image)`, которые будут сжимать и разжимать обратно файл изображения по определённому ниже алгоритму. Аргументы этих функций имеют такой же смысл, что в задании 1.

Результатом применения функции `compress()` должен стать файл с расширением ".cpr", который получается из оригинального BMP файла по следующему алгоритму.

1. Первые 54 байта файла ".cpr" такие же как и в файле ".bmp".
2. Далее идёт пиксельная информация. Пиксели теперь кодируются следующим образом: сначала идёт один байт, обозначающий то количество раз, которое некоторый цвет встречается в строке подряд, а потом идут 3 байта, которые кодируют этот цвет в палитре rgb.

Результатом применения функции `decompress()` должен стать файл ".bmp", разжатый из файла ".cpr".

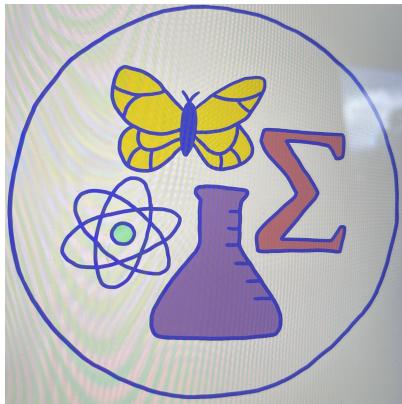
Обе написанные тобой функции должны использовать только функции стандартных библиотек используемого языка программирования.

2. Какие изображения алгоритм будет сжимать в наибольшее количество раз? Во сколько раз будут сжаты пиксельные данные таких изображений?
3. Если считать, что целью сжатия изображения является как можно большее уменьшение его размера, то при применении к каким изображением данный алгоритм сжатия покажет самый плохой результат? Каким будет этот результат?

## Задание 3

---

Дирекция потеряла цифровой исходник новой эмблемы. Осталась только её фотография, сделанная на телефон. Эта фотография доступна по ссылке 3. Дирекция заметила, что результат применения реализованного алгоритма сжатия к этой фотографии очень плохой.



1. Объясни, почему результат оказался таким плохим.

Этот алгоритм может давать более приемлемый результат, если при сжатии "потерять" часть информации, имеющейся в изображении. Например, нескольким пикселям, которые изначально имели разные цвета, при сжатии можно присвоить один и тот же цвет.

2. Напиши функцию `super_compress(input_image, result_image)`, которая улучшит степень сжатия фотографии новой эмблемы алгоритмом из задания 2 за счёт потери части информации. Добейся того, чтобы при сжатии размер фотографии уменьшился хотя бы на 20%. Также напиши функцию `super_decompress(input_image, result_image)`, которая создаст изображение BMP из файла, полученного применением функции `super_compress()`.

Постарайся сделать так, чтобы исходная фотография и её "разжатая" версия не сильно отличались друг от друга.