HW - Radon utility

Задача

> Найти репозиторий на github и на безвозмездной основе попробовать улучшить цикломатическую сложность кода

Шаги

Поиск репозитория для работы

Первым делом, нужно обращать внимание на популярность репозитория и активность в нем - если он был заброшен пару лет назад, контрибьютить (от англ. *contribute* - жертвовать) в него сравнимо с выкрикиванием ответа уже после семинара в пустой аудитории

Во-вторых, нужно разбираться в том, что собираешься исправлять - для меня было бы затруднительно разбираться в тонкостях работы с видео и аудио файлами на python, так как подобного опыта у меня не было

Выбор пал на ~~возможно~~ самый популярный репозиторий с алгоритмами и структурами данных на python - TheAlgorithms/Python

В этом репозитории есть огромное количество алгоритмов и структур данных, поэтому, думаю, можно будет найти, что улучшить

Ищем сложный код

Для начала, нужно определиться, что мы хотим улучшать. Для меня проще всего было бы разобраться в математических алгоритмах из теории чисел и всего подобного

Изучив репозиторий, стало понятно, что все подобные алгоритмы находятся в `Python/maths/`

Теперь ищем, код с большой цикломатической сложностью. Для этого используем утилиту Radon:

```
$ radon cc Python/maths -a -nc
Python/maths/jaccard_similarity.py
    F 17:0 jaccard_similariy - C
Python/maths/segmented_sieve.py
    F 6:0 sieve - C
```

Отлично, у нас есть претенденты на улучшения, посмотрим на них подробнее:

```
$ radon cc Python/maths/jaccard_similarity.py -a
Python/maths/jaccard_similarity.py
    F 17:0 jaccard_similariy - C

1 blocks (classes, functions, methods) analyzed.
Average complexity: C (11.0)
```

```
$ radon cc Python/maths/segmented_sieve.py -a
Python/maths/segmented_sieve.py
    F 6:0 sieve - C

1 blocks (classes, functions, methods) analyzed.
Average complexity: C (13.0)
```

Сложность у второго претендента (`segmented_sieve`) выше, им и займемся

Анализ кода

Подробнее посмотрим на код файла segmented_sieve.py

Нам нужно смотреть на цикломатическую сложность, поэтому обращаем внимание на циклы и условные ветки (aka ифы)

1. Зачем нужна вот эта проверка?

   ```
   if temp[i] is True:
     temp[i] = False
   ```

   Правильно, нам неважно, какое значение находится в `temp[i]`, после этого блока кода он должен стать `false`

   Так и запишем:

   ```
   temp[i] = False
   ```

2. Проверка там, где можно обойтись без нее

   ```
   low = end + 1
   high = low + end - 1
   if high > n:
     high = n
   ```

Запишем все это короче и без условного оператора (но придется немного подумать)

```
low = end + 1
# low + end - 1 = end + 1 + end - 1 = 2 * end
high = min(2 * end, n)
```

3. Видим такой же *(почти)* код, но еще и в цикле 😎

```
while low <= n:
  ...
  low = high + 1
  high = low + end - 1
  if high > n:
    high = n
```

Меняем:

```
low = high + 1
# low + end - 1 = high + 1 + end - 1 = high + end
high = min(high + end, n)
```

Проверяем, что сделали

Запускаем утилиту Radon:

```
$ radon cc Python/maths/segmented_sieve.py -a
Python/maths/segmented_sieve.py
    F 6:0 sieve - B

1 blocks (classes, functions, methods) analyzed.
Average complexity: B (10.0)
```

🎉 Ура!
Получилось уменьшить среднюю сложность **13.0** -> **10.0**
Более того, мы повысили ранг C -> B

Кричим на весь мир

Теперь остается отправить изменения в свой fork-репозиторий, создать PR в основном репозитории, предварительно ознакомившись с правилами контрибьюции в него, оставив комментарий по проделанной работе

Результаты

PR тут:

> https://github.com/TheAlgorithms/Python/pull/6372

По всем вопросам: @heartsker