**Алгоритм и анализ сложности**

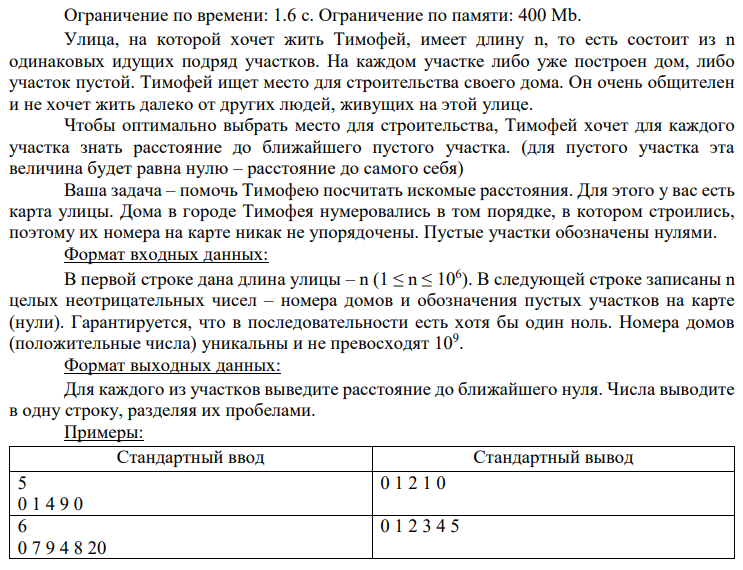
Лабораторная работа 1

Крюков Никита А. (АТ-01)

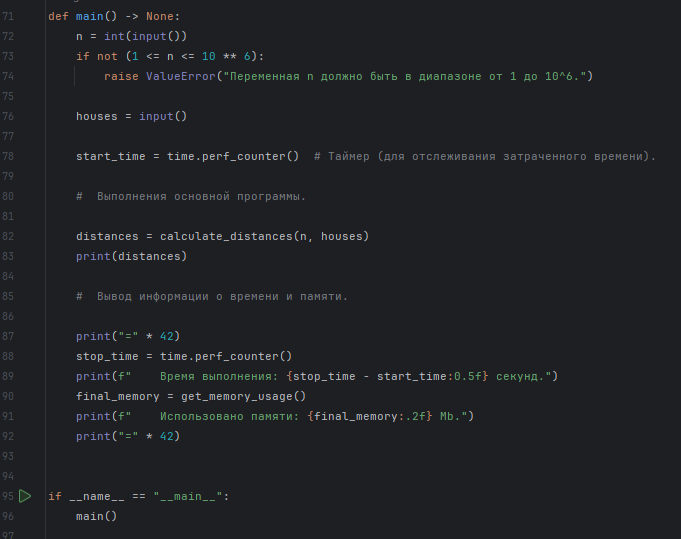
РИ-230916

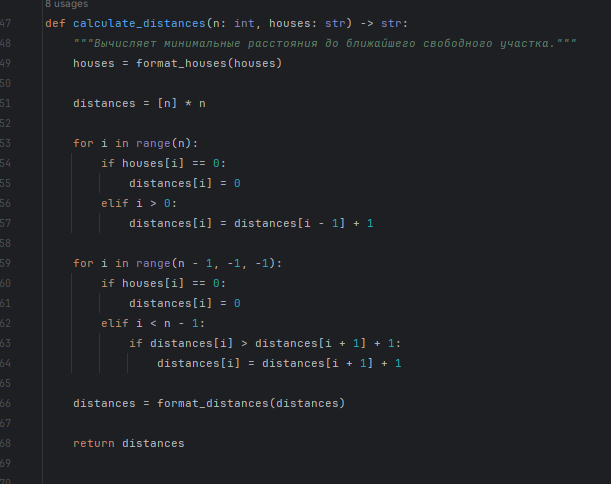
Вариант: 14

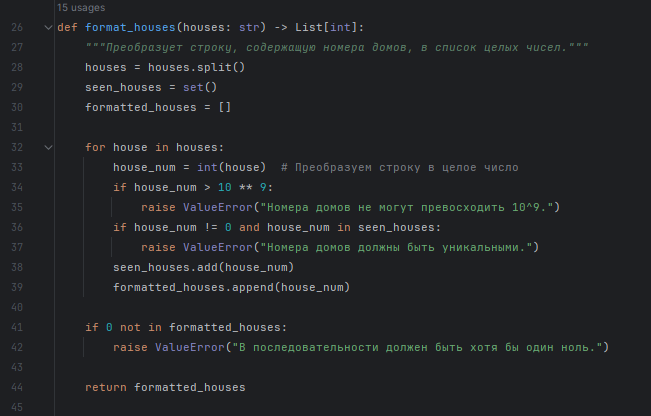
**Задание 1. Ближайший ноль.**

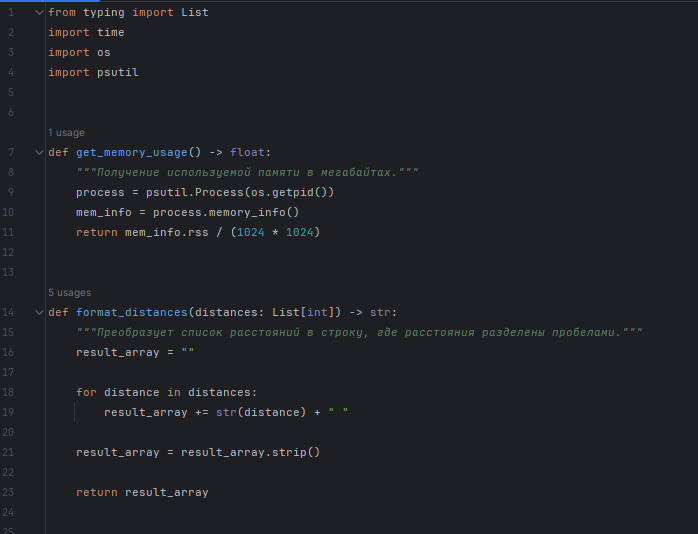


Код:

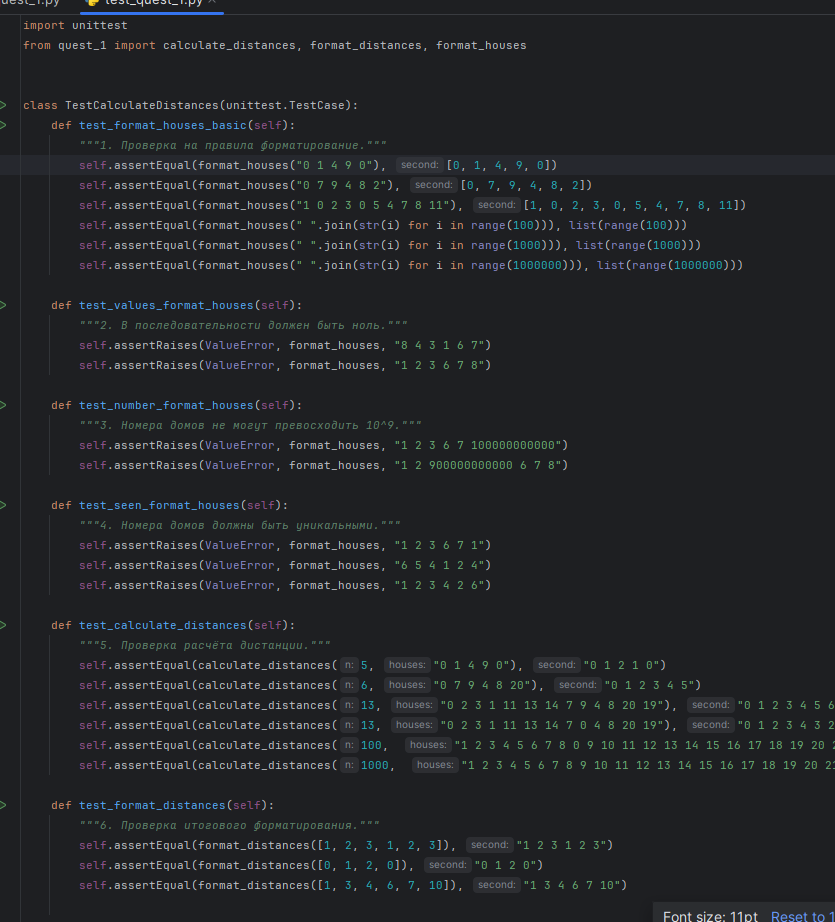


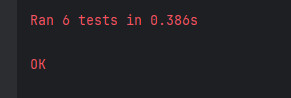






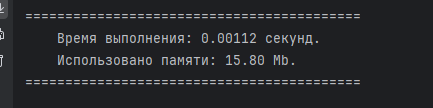
Тесты (Проходят успешно):



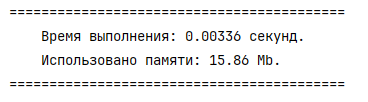


В данном задании на мой взгляд наилучшим является вариант с итеративным подходом, а не с рекурсивным.

Тесты на 1000 значений показали такие значения:  
  
Итеративный способ:

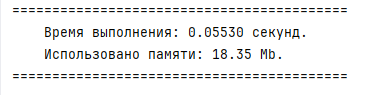


Рекурсивный:



Тесты на 100000 значений показали такие результаты:

Итеративный способ:



Рекурсивный способ:  


Ход решения задачи:

Задача вычисления минимальных расстояний до ближайшего свободного участка для каждого дома в заданной последовательности домов.

Функция main – это точка входа. Считывает кол-во домов и строку с номером домов, вызывает функцию calculate\_distances. Выводит результат и время выполнения программы.

Функция calculate\_distances вычисляет минимальные расстояния до ближайшего свободного участка для каждого дома.

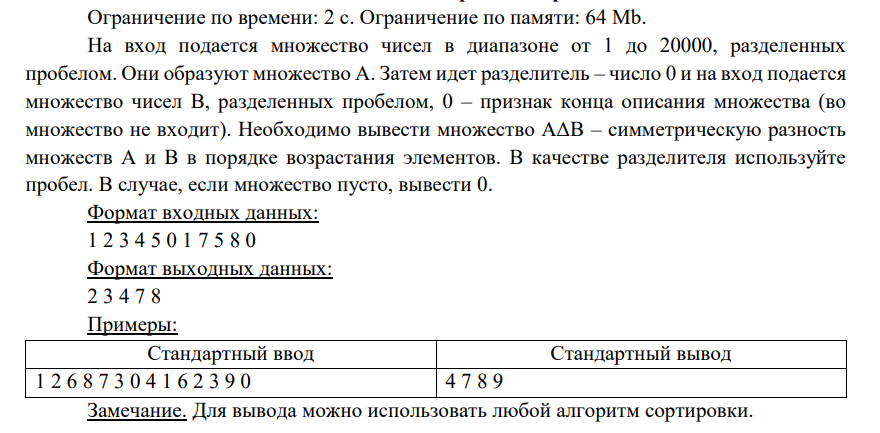
Происходит вызов функции format\_houses, которая принимает строку, содержащую номера домов, и преобразует её в список целых чисел.

Функция calculate\_distances инициализирует список расстояний distances размером n, заполненный значением n. Далее выполняет два прохода по списку домов: первый проход слева направо и второй проход справа налево, обновляя расстояния до ближайшего свободного участка. В конце преобразует список расстояний в строку с помощью format\_distances. Сложность алгоритма O(n).

Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab\_1\_complexity/tree/main/quest\_1

**Задание 3. Симметрическая разность.**

Условие задачи:

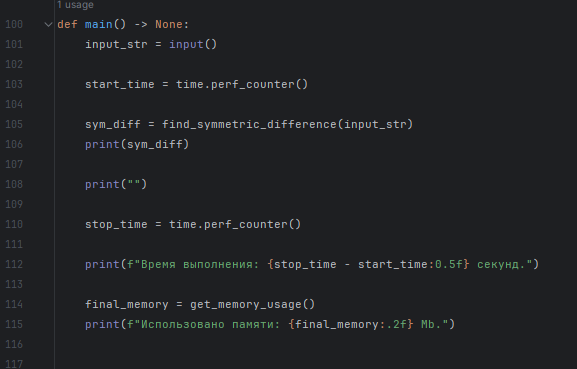


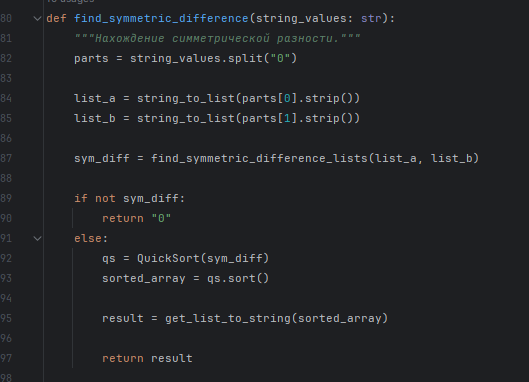
Ход решения задачи:

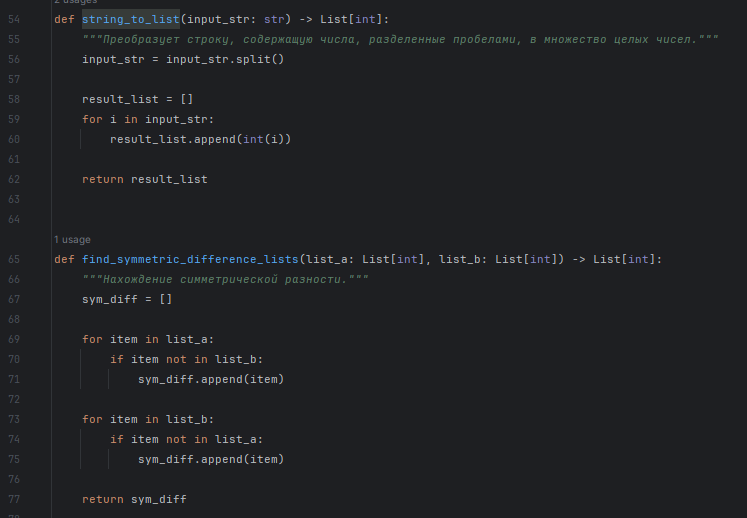
Вычисляет симметрическую разность двух множеств чисел, представленных в виде строки, разделенной нулем.

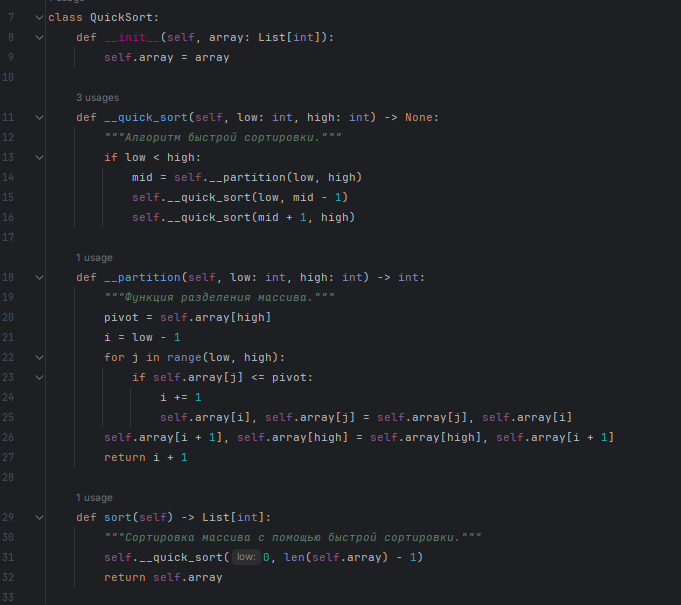
Функция main – это точка входа. Считывает строку, содержащую числа. Вызывает функцию find\_symmetric\_difference для вычисления симметрической разности. Разделяет строку на два списка, возвращает симметрическую разность. Симметрическая разность включает элементы, которые присутствуют только в одном из списков.

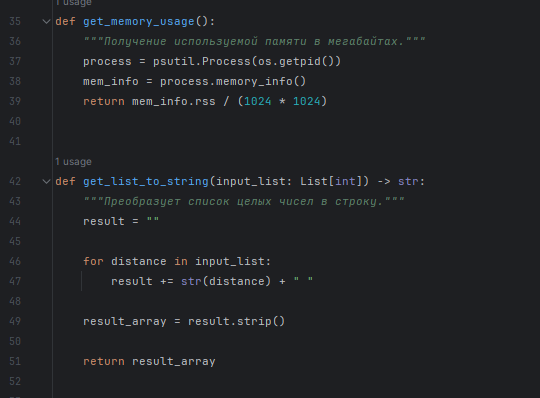
Код:



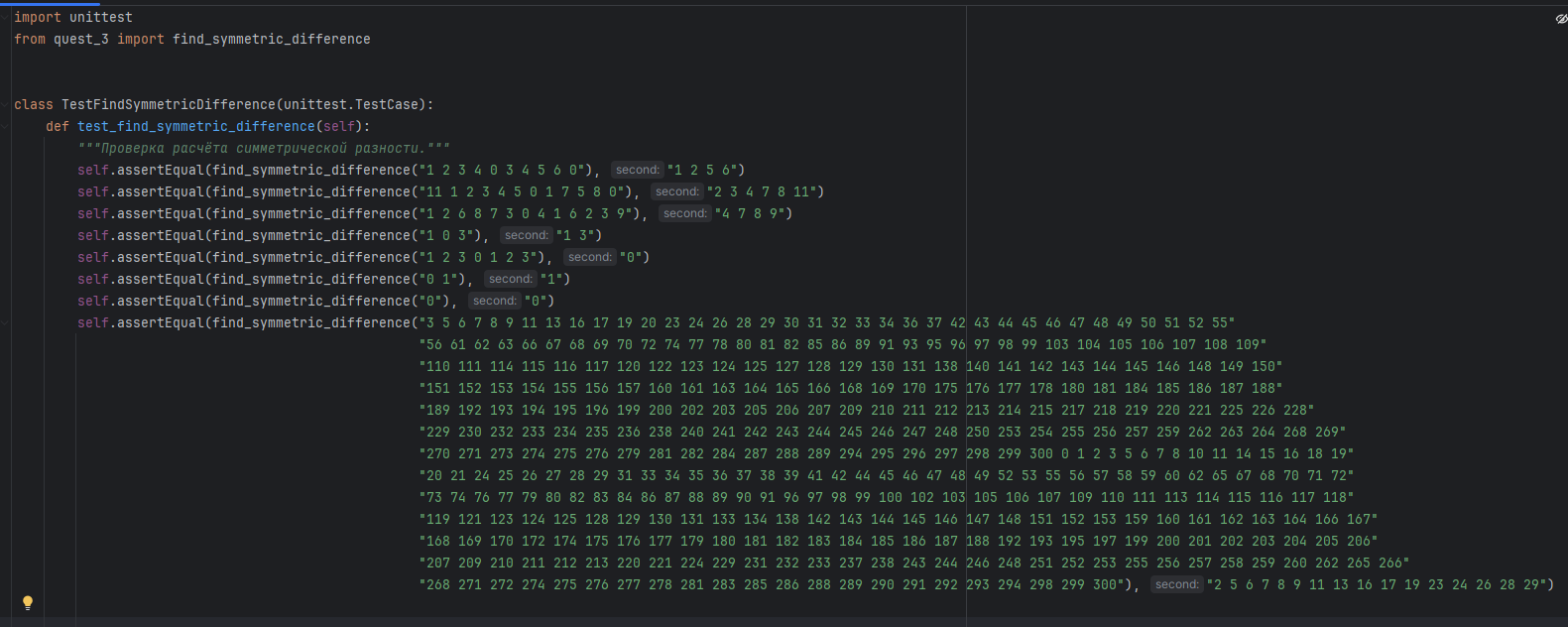


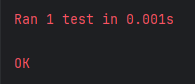




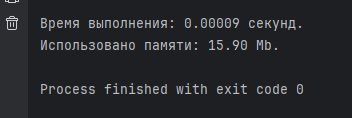


Тесты (Проходят успешно):





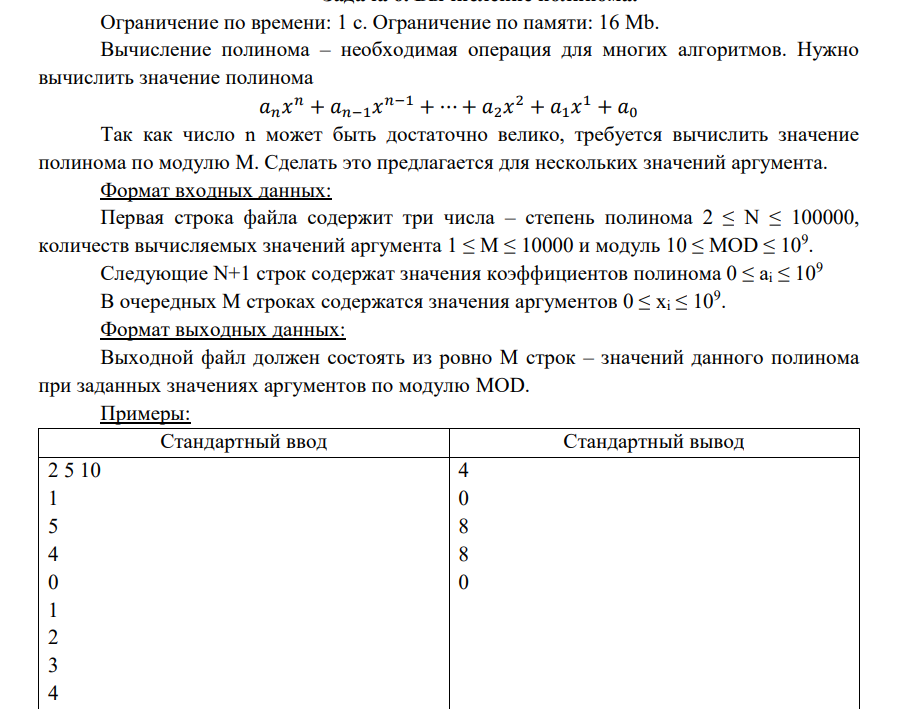
Дополнительно результат на самый тяжёлый тест  
(Более 300 значений):



Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab\_1\_complexity/tree/main/quest\_3

**Задание 6. Вычисление полинома.**

Условие задачи:

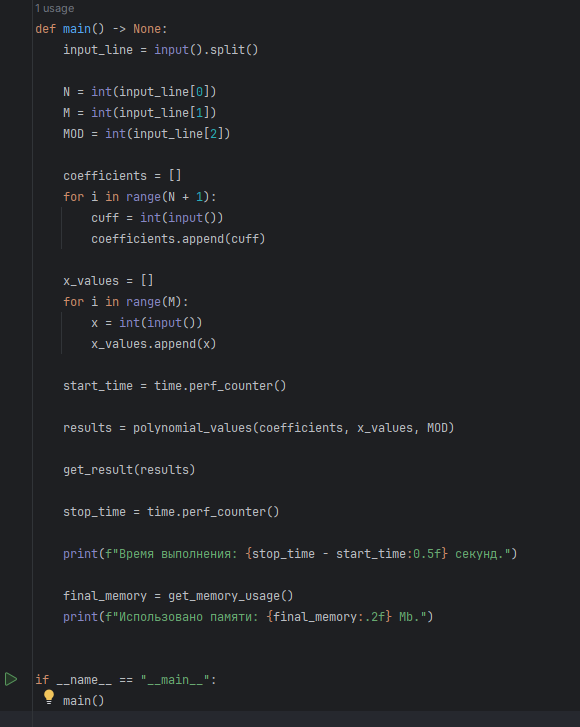
Ход решения задачи:

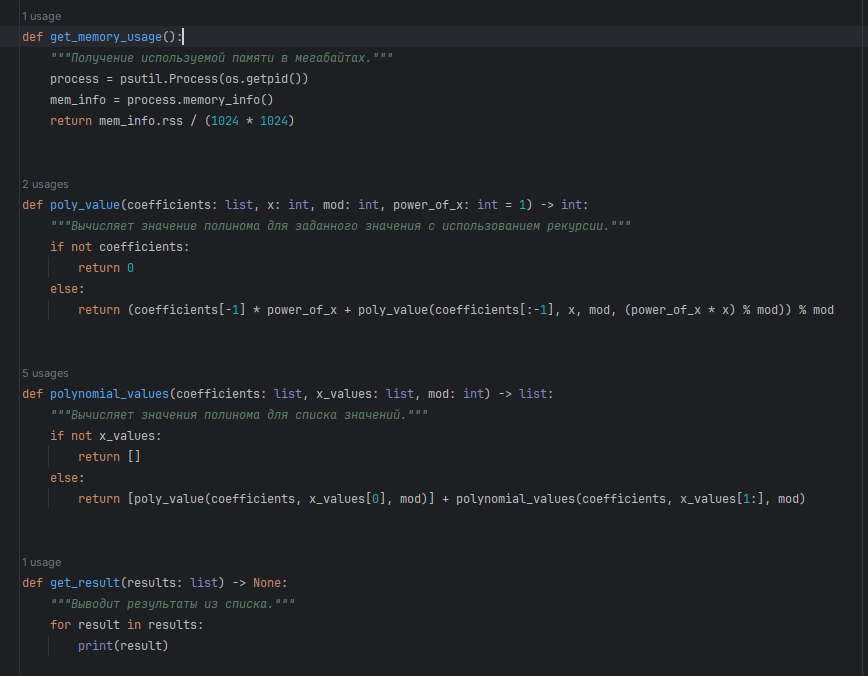
Функция main – это точка входа. Считывает и преобразует необходимые для вычислений значения (степень полинома, количество вычисляемых значений, модуль, значения коэффициентов). После чего вызывает метод polynomial\_values и передаёт туда полученные значения.

Функция polynomial\_values рекурсивно вычисляет значения полинома для каждого значения **x** из списка. Полином задан в виде списка коэффициентов, где коэффициенты идут от старшей степени к младшей. Берется последний коэффициент из списка (coefficients[-1]). Умножается этот коэффициент на текущую степень x (power\_of\_x). Рекурсивно вызывается функция poly\_value для оставшихся коэффициентов (coefficients[:-1]), с увеличенной степенью x (power\_of\_x \* x) % mod). Результат суммируется и берется по модулю mod. Она использует функцию poly\_value для вычисления значения полинома для каждого элемента списка. Функция poly\_value использует рекурсию для вычисления значения полинома. Она последовательно умножает коэффициенты на соответствующие степени x и суммирует результаты.

Функция get\_result в методе main просто выводит результаты, полученные из функции polynomial\_values.

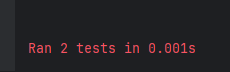
Код



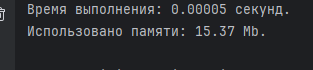


Тесты





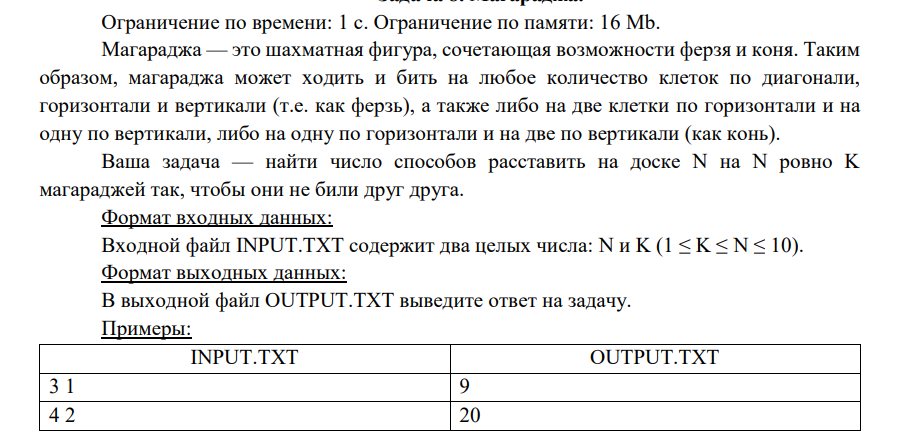
Самый сложный тест:



Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab\_1\_complexity/tree/main/quest\_6

**Задание 8. Магараджа.**

Условие задачи:

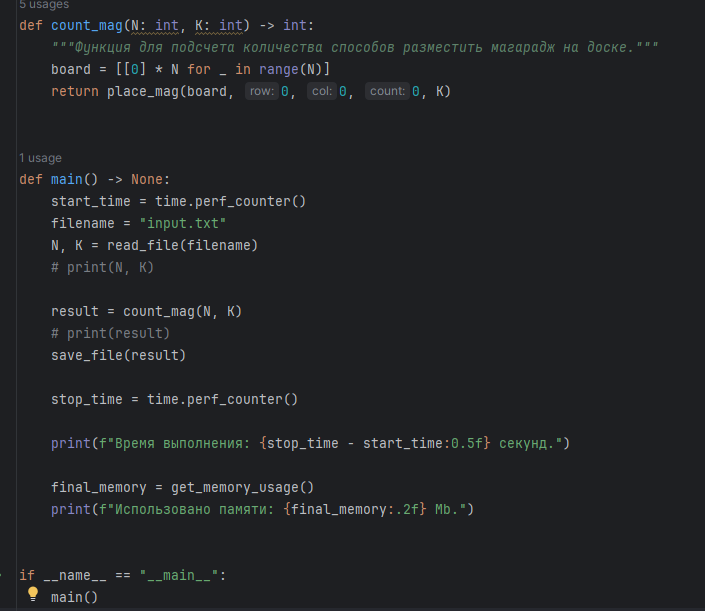


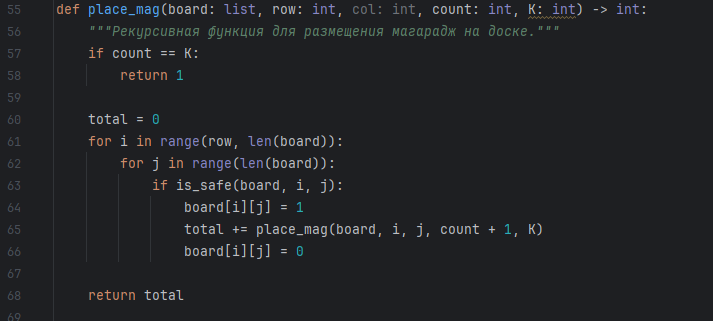
Ход решения задачи:

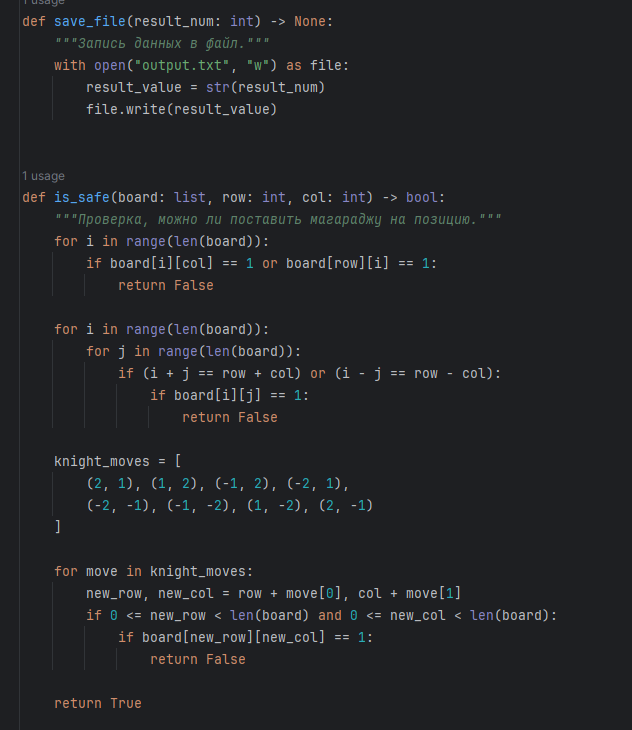
Функция main – это точка входа. Считывает входные данных из документа с помощью метода read\_file. Функция count\_mag создает пустую доску размером N x N и вызывает place\_mag для подсчета количества способов размещения магарадж. Записывает результат в файл с помощью save\_file. Измеряет и выводит время выполнения и использованную память.

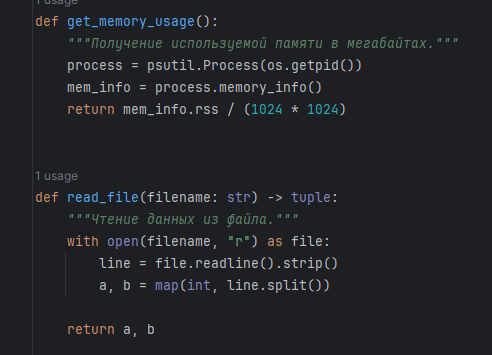
Функция place\_mag рекурсивно размещает магарадж на доске. Эта рекурсивная функция размещает магарадж на доске. Она пробует все возможные позиции и подсчитывает количество способов разместить K магарадж на доске размером Если количество размещенных магарадж достигает значения, функция возвращает 1 (успешное размещение). В противном случае, функция пробует разместить магараджу на каждой позиции, проверяя безопасность с помощью is\_safe. Если размещение безопасно, функция рекурсивно вызывает себя для следующей магараджи.

Код

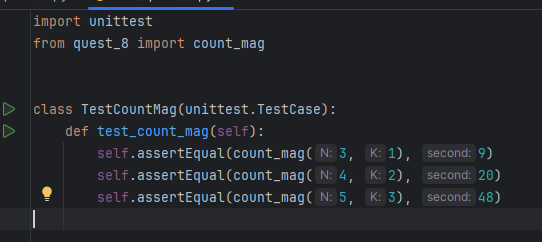


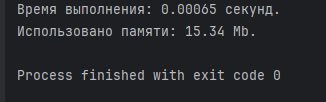






Тесты (Проходят успешно):

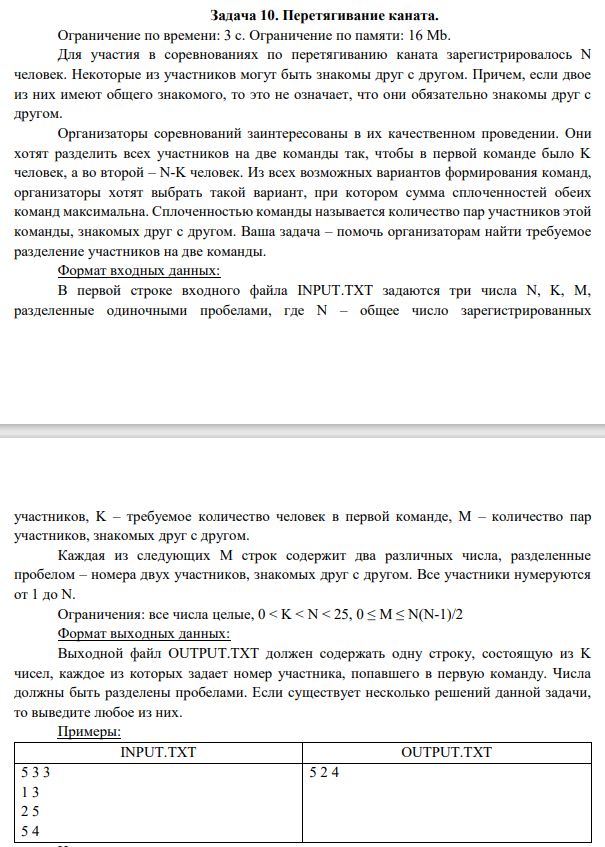




Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab\_1\_complexity/tree/main/quest\_8

**Задание 10. Перетягивание каната.**

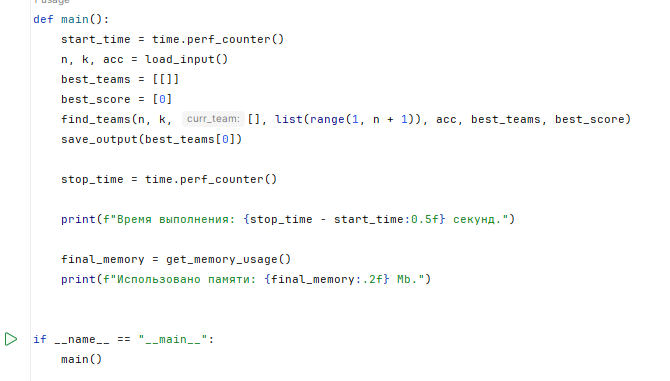
Условие задачи:

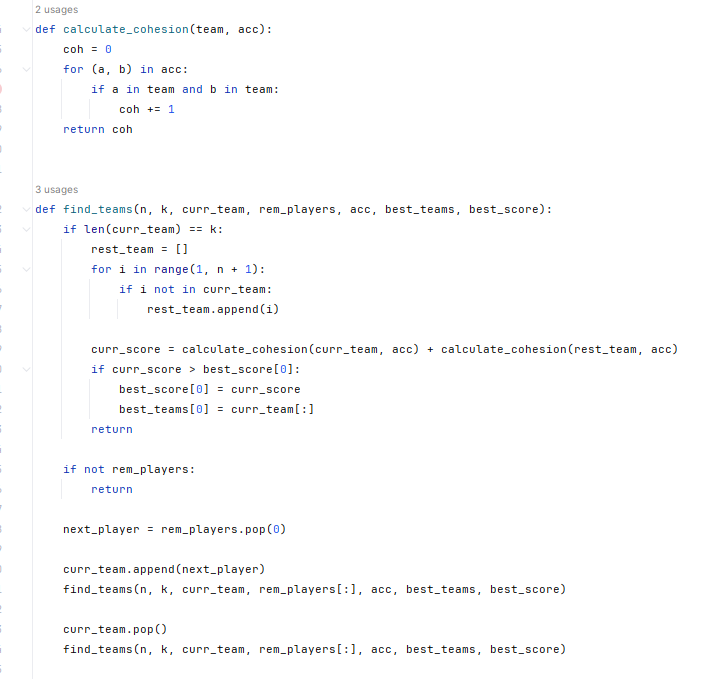


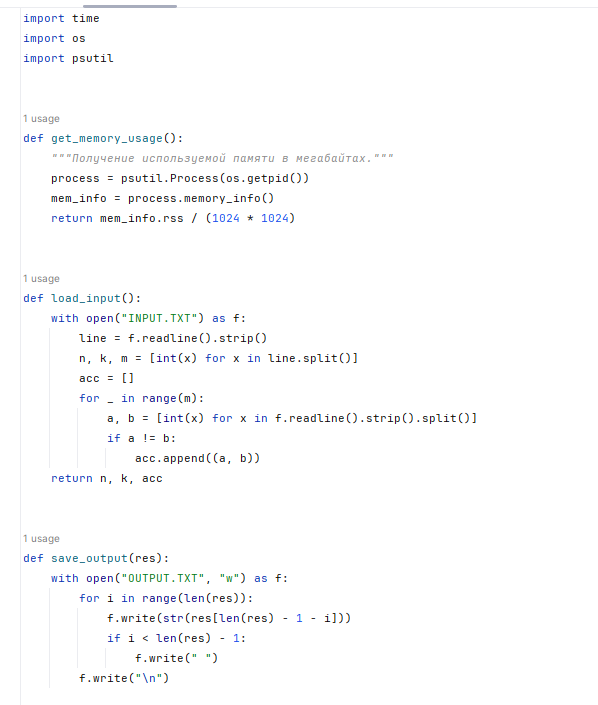
Ход решения задачи:

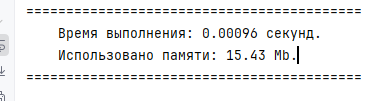
Функция find\_teams рекурсивно генерирует все возможные команды игроков, перебирая оставшихся игроков. Выбирает игрока для текущей команды, потом рекурсивно вызываю саму себя с обновленной командой и обновленным списком оставшихся игроков. Как только команда достигает нужного размера (k), я вычисляю коэффициент связности и обновляю лучшие команды и лучший результат, если это необходимо. В конце каждого вызова (при выходе из рекурсии), я очищаю команду от последнего добавленного игрока и снова вызываю себя, чтобы рассмотреть следующую возможность с оставшимися игроками. Таким образом это позволяет проходить через все возможные комбинации игроков и находить оптимальные команды.

Код









Если видно плохо или хочется протестировать, то можете посмотреть мой код по заданию тут: https://github.com/ytkinroman/lab\_1\_complexity/tree/main/quest\_10