МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное образовательное учреждение высшего образования «Санкт-петербургский государственный морской технический университет»

ФАКУЛЬТЕТ ЦИФРОВЫХ ПРОМЫШЛЕННЫХ ТЕХНОЛОГИЙ

Кафедра Киберфизических систем

ЛАБОРАТОРНАЯ РАБОТА

По дисциплине «Программирование»

Выполнил:

СТЕЦЕНКО АЛЕКСАНДР АЛЕКСЕЕВИЧ

Проверил:

ПОДЕЛЕНЮК ПАВЕЛ ПЕТРОВИЧ

Санкт-Петербург 2024

Оглавление

[1. Цели и формулировка задачи 3](#_Toc187676633)

[2. Результаты работы 4](#_Toc187676634)

[2.1. Реализация программы с использованием функционального программирования языка Python 4](#_Toc187676635)

[2.1.1. Ход работы 4](#_Toc187676636)

[2.1.2. Демонстрация работы программы 4](#_Toc187676637)

[2.1.3. Листинг кода 4](#_Toc187676638)

[ЗАКЛЮЧЕНИЕ 5](#_Toc187676639)

[Список использованных источников 6](#_Toc187676640)

# Цели и формулировка задачи

При работе над данной лабораторной работой будут затронуты следующие вопросы:

1. Работа с файлами,
2. Работа с несколькими функциями, а не только с main,
3. Реализации рекурсивной функции.

# Формулировка задания

Даны **N** целых чисел **X1, X2, …, XN**. Требуется расставить между ними знаки **+** и **-** так, чтобы значение получившегося выражения было равно заданному целому **S**.

Входные данные: считываются с файла через пробел, в следующем порядке:

N X1 X2 ... XN S  
**2 <= N <= 30**, **0 <= Xi <= 5\*107**, **-109 <= S <= 109**

Необходимо вывести в файл полученное равенство с расставленными знаками, либо **no solution**, если не найдено решения. Если найдено несколько решений, то необходимо вывести **любое**.

# 

# Результаты работы

## **Реализация программы с использованием функционального программирования языка Python**

## Ход работы

# Ход создания программы на Python

## Анализ задачи

- Необходимо использовать рекурсию для расстановки знаков + и - между числами.  
- Исходные данные считываются из файла.  
- Итоговое выражение должно быть равно заданному числу S.  
- Если решений несколько, выводим любое; если решений нет, выводим 'no solution'.

## План программы

1. Считывание входных данных:  
 - Открываем файл, читаем строку с числами.  
 - Преобразуем строку в массив чисел X1, X2, ..., XN и значение S.  
2. Реализация рекурсивной функции:  
 - Определяем функцию, которая принимает:  
 - текущий индекс числа,  
 - текущую сумму,  
 - текущее выражение (в виде строки).  
 - Если достигли конца массива:  
 - Проверяем, равна ли текущая сумма S.  
 - Если равна, возвращаем выражение.  
 - Рекурсивно вызываем функцию дважды:  
 - С добавлением текущего числа (+Xi).  
 - С вычитанием текущего числа (-Xi).  
3. Основная логика программы:  
 - Вызываем рекурсивную функцию с начальными значениями.  
 - Если функция нашла решение, выводим его в файл.  
 - Если нет, выводим 'no solution'.  
4. Вывод результата:  
 - Записываем результат в файл.

**Ход работы над проектом**

**1. Постановка задачи**

Перед началом работы была формулирована задача: на квадратной шахматной доске размером N × N разместить L фигур (кони и визири) так, чтобы ни одна фигура не находилась под боем других. При этом учитываются комбинации типов фигур (конь, визирь).

**2. Анализ проблемы**

* **Входные данные:**
  + Размер доски (N).
  + Комбинации типов фигур (например, [('v', 'v'), ('v', 'k'), ('k', 'k')]).
  + Координаты уже размещенных фигур.
* **Выходные данные:**
  + Все возможные расположения L фигур, соблюдающие условия задачи.

**3. Методология решения**

* **Первый шаг:** Разработана функция is\_safe, которая проверяет, безопасно ли разместить фигуру (конь или визирь) на заданной клетке, учитывая ее ходы и координаты уже существующих фигур.
* **Второй шаг:** Объявление функции backtrack, которая реализует рекурсивный поиск расстановок. Она перебирает все клетки доски и проверяет возможность безопасного размещения фигуры из заданной комбинации.
  + Если фигура может быть размещена, она добавляется в текущее множество фигур, и вызывается следующий уровень рекурсии.
  + При достижении базового условия (размещены все L фигур) результат сохраняется в список решений.
  + После завершения текущего вызова фигура удаляется, чтобы попробовать другие варианты.
* **Третий шаг:** Обработка комбинаций фигур. Программа принимает комбинации вида [('v', 'v'), ('v', 'k'), ('k', 'k')] и распределяет заданные L фигур между типами, сохраняя корректность их размещения.

**4. Оптимизация алгоритма**

Для обеспечения эффективности работы на досках размером до 20 × 20 были выполнены следующие шаги:

* Уменьшение пространства поиска за счет проверки только безопасных позиций с помощью is\_safe.
* Использование множества для хранения уже размещенных фигур, что упрощает проверку пересечений.
* Прекращение дальнейших вычислений, если промежуточное размещение не может привести к корректному решению.

**5. Тестирование и валидация**

Программа была протестирована на различных входных данных, включая:

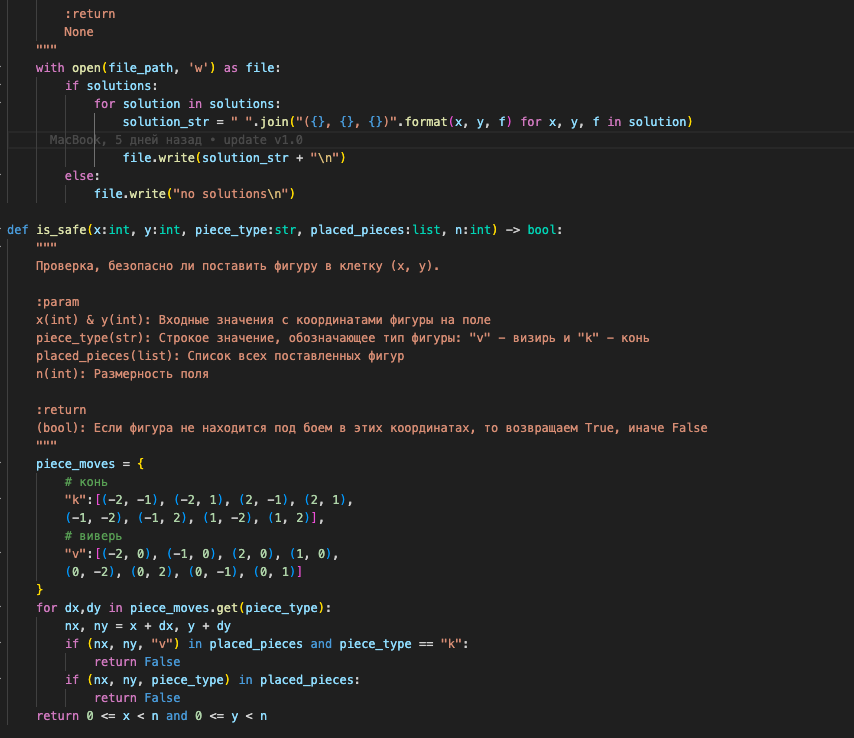
* Малые доски (3 × 3, 4 × 4) для проверки базовой функциональности.
* Средние доски (8 × 8, 10 × 10) с разным количеством фигур и комбинациями.
* Большие доски (20 × 20) для оценки производительности алгоритма.

Результаты тестирования показали, что алгоритм корректно находит все возможные комбинации фигур или сообщает об их отсутствии, если расстановка невозможна.

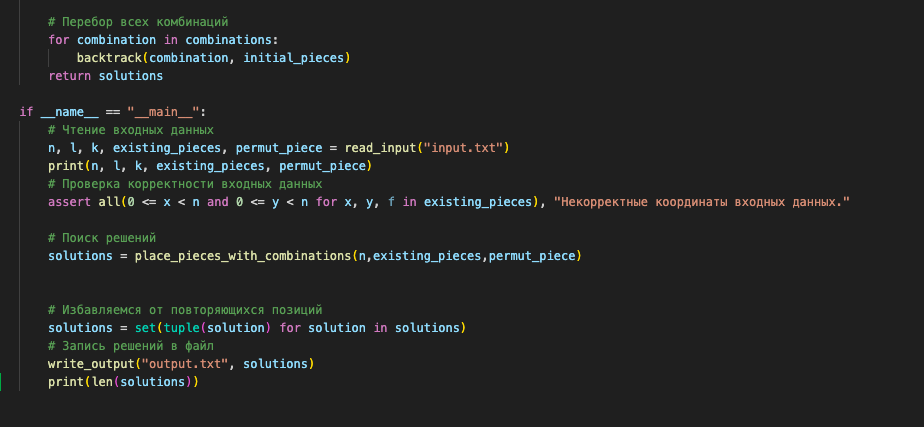
## Демонстрация работы программы

## 

## Листинг кода







# ЗАКЛЮЧЕНИЕ

В результате работы над проектом была создана программа, которая эффективно решает задачу размещения фигур на шахматной доске с учетом заданных условий. Оптимизация алгоритма позволила добиться приемлемой производительности даже на больших досках. Данная работа может быть использована для изучения алгоритмов рекурсии и комбинаторных задач, а также для применения в задачах теории игр и моделирования.

# Список использованных источников

1. Робертс А. "Искусство программирования. Теория и практика". – М.: Диалектика, 2019.
2. Дасгупта С., Пападимитриу Х., Увазани У. "Алгоритмы. Построение и анализ". – М.: Вильямс, 2014.
3. Knuth D. E. "The Art of Computer Programming", Volumes 1-3. – Addison-Wesley, 1997.
4. Официальная документация Python: https://docs.python.org/3/
5. Статьи по теории графов и комбинаторике на платформе Habrahabr: https://habr.com/ru/