**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра САПР.**

**отчет**

**по курсовой работе**

**по дисциплине «Алгоритмы и структуры данных»**

**Тема: «Реализация алгоритма решения судоку через алгоритм раскраски графа».**

|  |  |  |
| --- | --- | --- |
| Студентка гр. 1301 |  | Ищенко Д.О. |
| Преподаватель |  | Пестерев Д.О. |

Санкт-Петербург

2023

**ЗАДАНИЕ**

**на курсовую работу**

|  |  |  |
| --- | --- | --- |
| Студентка Ищенко Д.О. | | |
| Группа 1301 | | |
| Тема работы: Реализация алгоритма решения судоку через алгоритм раскраски графа. | | |
| Исходные данные:  Частично заполненная таблица судоку. | | |
| Содержание пояснительной записки:  ..  Предполагаемый объем пояснительной записки:  Не менее 5 страниц. | | |
| Дата выдачи задания: 15.05.2023 | | |
| Дата сдачи реферата: 24.05.2023 | | |
| Дата защиты реферата: 24.05.2023 | | |
| Студентка |  | Ищенко Д.О. |
| Преподаватель |  | Пестерев Д.О. |

Оглавление

[Формулировка задания. 5](#_Toc28096)

[Теоретические сведения. 6](#_Toc23664)

[Файловая организация проекта. 7](#_Toc19901)

[Описание алгоритмов и структур данных программы. 7](#_Toc6063)

[Пример работы программы. 9](#_Toc7803)

[Текст программы. 9](#_Toc24991)

# Формулировка задания.

Реализация алгоритма решения судоку через алгоритм раскраски графа. Создание графического интерфейса программы.

**Контрольный пример.**

Входные данные: частично заполненная таблица судоку (рис 1).

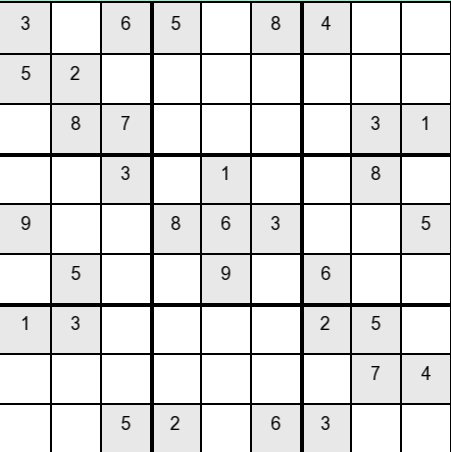


рис. 1

Выходные данные: Полностью заполненная таблица, с сохранением исходных данных (рис. 2).

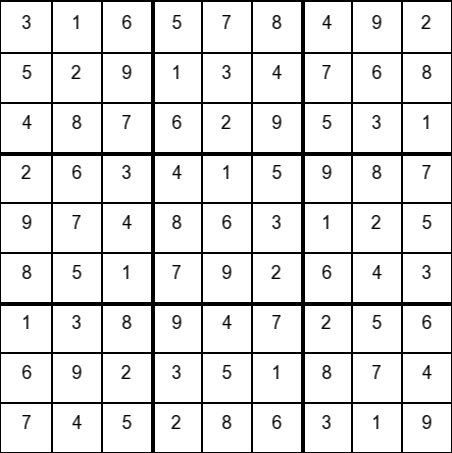


рис. 2

# Теоретические сведения.

**Судоку** - это головоломка, которая состоит из квадрата 9x9, разделённого на малые квадраты 3x3. Задача заключается в том, чтобы заполнить все свободные клетки цифрами от 1 до 9 таким образом, чтобы в каждой строке, в каждом столбце и в каждом малом квадрате 3x3 каждая цифра встречалась только один раз. Судоку является популярной логической головоломкой, которая развивает умственные способности, такие как логика и концентрация внимания. Существуют различные методы решения судоку, от простых до сложных, включая методы перебора и логические стратегии.

**Задача раскраски** графа заключается в том, чтобы раскрасить вершины графа таким образом, чтобы никакие две смежные вершины не имели одинаковый цвет.

**Применение задачи раскраски к решению судоку.**

Решение судоку можно рассматривать как завершение раскраски 9 цветами заданного графа из 81 вершины. Каждый из девяти цветов должен появиться в каждой строке, столбце и блоке (рис. 3). Для решения задачи раскраски графа существует множество алгоритмов, таких как жадная раскраска графа и рекурсивная раскраска графа.

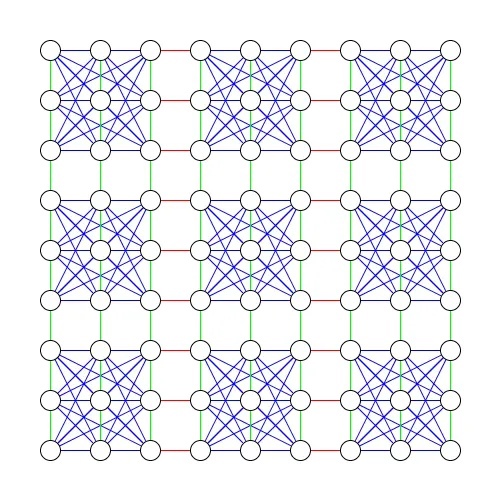


рис. 3

# Описание алгоритмов.

Для решения задачи раскраски графа существует множество различных алгоритмов, таких как:

1. **Жадный алгоритм**: этот алгоритм раскрашивает вершины графа одну за другой в некотором порядке и назначает наименьший доступный цвет каждой вершине, которой еще не назначен цвет. Этот алгоритм прост и эффективен, но не всегда дает оптимальное решение.
2. **Алгоритм DSatur**: этот алгоритм похож на жадный алгоритм, но он выбирает следующую вершину для окрашивания на основе степени насыщенности, которая представляет собой количество различных цветов, присвоенных ее соседям. Этот алгоритм имеет тенденцию давать лучшие результаты, чем жадный алгоритм.
3. **Алгоритм рекурсивного наибольшего первого** (RLF): этот алгоритм рекурсивно окрашивает вершины графа, начиная с вершины с наибольшей степенью и присваивая ей наименьший доступный цвет. Затем он рекурсивно окрашивает оставшиеся вершины. Этот алгоритм похож на алгоритм DSatur, но работает медленнее.
4. Алгоритм **Кемпе**: этот алгоритм используется для уменьшения количества цветов, необходимых для раскрашивания графика. Он работает, меняя местами цвета двух соседних вершин, а затем перекрашивая граф с помощью жадного алгоритма. Если жадный алгоритм дает правильную раскраску, то количество требуемых цветов уменьшено на один.
5. Алгоритм **поиска с возвратом**. Этот алгоритм является точным алгоритмом, который рекурсивно перебирает все возможные раскраски графа, пока не найдет правильную раскраску с наименьшим количеством цветов. Этот алгоритм очень медленный и непрактичный для больших графов, но он всегда дает **оптимальное решение**.

Первой попыткой решения поставленной задачи была реализация жадного алгоритма.

***Листинг 1.***

func **greedyColoring**():

*// A temporary array to store the available colors. True*

*// value of IsLinkedCr[cr] would mean that the color cr is*

*// assigned to one of its adjacent vertices*

IsLinkedCr[] = false

**for** u = 0 to N-1

**if** result[u] *already colored*

**continue**

// Process all adjacent vertices and flag their colors

// as unavailable

**for** each v **in** adj[u]

**if** result[v] *not colored*

IsLinkedCr[result[v]] = true

**for** cr = 0 **to** N-1

**if** !IsLinkedCr[cr] //cr not linked

**break**

result[u] = cr

*Reset isLinkedCr to false*

Асимптотическая сложность алгоритма greedyColoring() зависит от размера графа и числа его вершин, обозначим их как N.

Внешний цикл проходит по всем вершинам графа, итераций будет N.

Внутренний цикл перебирает смежные вершины каждой вершины графа, количество смежных вершин в худшем случае может быть равно N.

В худшем случае, когда все вершины имеют смежные вершины, внутренний цикл будет выполняться N раз в каждой итерации внешнего цикла.

Таким образом, общая асимптотическая сложность алгоритма greedyColoring() будет O(N^2), так как имеется два вложенных цикла, которые могут выполниться до N раз каждый.

Однако он не даёт оптимального решения для раскраски графа, то есть количество цветов превосходит 9. Что в корне не подходит для изначальной задачи решения судоку.

Не смотря на то, что алгоритмы 2-4, работаю лучше, поскольку остаётся вероятность получить раскраску с неоптимальным количество цветов, решено было реализовывать алгоритм поиска с возвратом.

Алгоритм работает следующим образом:

***Листинг 2.***

func **solveSudoku**()

row, col = *find empty place*

**If** *table filled*:

**return** true;

**for** num = 1to 9

**if** *is valid place* for (row, col, num)

grid[row][col] = num;

**if solveSudoku**()

return true;

grid[row][col] = 0;

//turn to unassigned space when conditions are not satisfied

**return** false;

Если алгоритм возвращает false, значит решение не было найдено. Поскольку алгоритм рассматривает все корректные на каждом этапе решения, то не нахождение им решения, означает, что задачу решить невозможно в целом.

**Асимптотическая сложность** этого алгоритма зависит от размера судоку и количества вариантов для заполнения каждой пустой ячейки.

Пусть размер судоку равен N (обычно N = 9 для стандартного судоку). В лучшем случае, когда судоку уже полностью заполнено, алгоритм закончит работу сразу же и вернёт true. Это займёт константное время O(1).

В худшем случае, когда алгоритм должен пробовать все возможные варианты для каждой пустой ячейки, он будет рекурсивно вызываться для каждой ячейки. В каждом вызове алгоритма рассматриваются N возможных значений для заполнения ячейки. Таким образом, общее количество вызовов рекурсии будет равно N^M, где M - количество пустых ячеек.

Следовательно, в худшем случае асимптотическая сложность алгоритма составляет O(N^M).

Однако при большом количистве пустых клеток алгоритм наоборот начинает работать быстрее, поскольку количество верных раскрасок увеличивается.

На рис. 4 представлены практические данные, полученные для среднего времени работы алгоритма (ОY). В качетсве входных данных использовались сгенерированные задачи судоку с разным количеством пустых клеток (ОХ) по 100 задач на каждую заполненность.

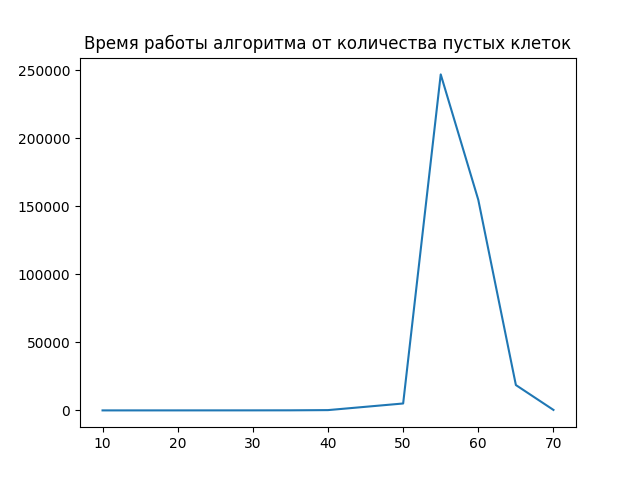


рис. 4.

# Внутреннее устройство проекта.

В качестве инструмента для написания графического интерфейса была выбрана библиотека SFML.

**SFML** (Simple and Fast Multimedia Library) - это свободная кроссплатформенная мультимедийная библиотека для разработки программного обеспечения, которая обеспечивает простой интерфейс для представления мультимедийных данных.

Таким образом классы Field, UserInterface и Button используют инструменты из библиотеки sfml для отрисовывания графики и получения от пользователя команд. Классы модуля SudokuSolver работают уже с данными полученными от пользователя и решают задачу курсовой.

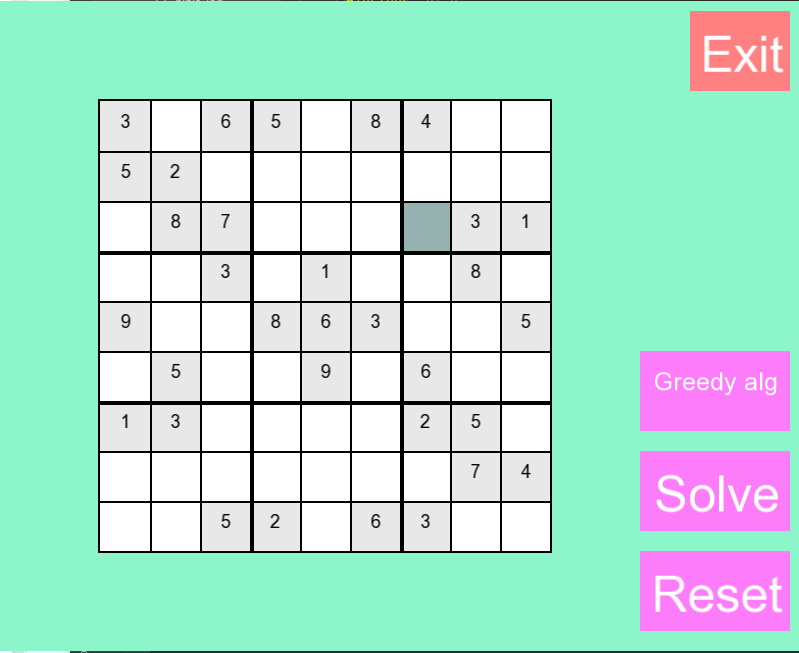


рис 5. Интерфейс программы.

Интерфейс программы позволяет вводить числа в поле судоку. Нажатие на кнопку Solve решает судоку с помощью рекурсивного алгоритма. Reset полностью очищает поле судоку. Greedy alg применяет жадный алгоритм к заданному полю.

# Текст программы.

[Ссылка на github](https://github.com/Nekttuman/Algosee)