РОССИЙСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ НЕФТИ И

ГАЗА (НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ) ИМЕНИ И.М. ГУБКИНА

Кафедра информатики

Расчетно-графическая работа по дисциплине:

«Основы алгоритмизации и программирования»

по теме: «Вычислительная математика»

Выполнил:

Вершинин Сергей Алексеевич,

студент группы АТ-24-01

Проверила:

ФИО,

Должность

Москва 2025

**Оглавление**

[**Введение** 3](#_Toc196602080)

[**Основная часть** 4](#_Toc196602081)

[**Постановка задачи** 4](#_Toc196602082)

[**Логика работы приложения** 4](#_Toc196602083)

[**Реализация кода на Python** 5](#_Toc196602084)

[**Заключение** 10](#_Toc196602085)

[**Источники** 11](#_Toc196602086)

[**Приложение 1** 12](#_Toc196602087)

# **Введение**

Комбинаторные задачи играют важную роль в информатике, особенно при разработке алгоритмов для поиска оптимальных решений. Одной из таких задач является расстановка букв в квадратной матрице с определенными ограничениями. Такие задачи находят применение в теории кодирования, проектировании тестовых случаев и других областях.

Цель данной работы - разработка алгоритма для нахождения всех возможных расстановок 16 букв (четыре 'a', четыре 'b', четыре 'c', четыре 'd') в матрице 4×4, где в каждой строке и каждом столбце любая буква встречается только один раз.

# **Основная часть**

## **Постановка задачи**

Даны 16 букв: четыре каждого типа ('a', 'b', 'c', 'd'). Требуется расставить их в квадратной матрице 4×4 так, чтобы:

1. В каждой строке все буквы были уникальны
2. В каждом столбце все буквы были уникальны
3. Каждая буква встречалась ровно 4 раза во всей матрице

## **Бизнес-логика**

Программа последовательно выполняет генерацию и проверку всех возможных комбинаций расстановки букв в матрице 4×4. Сначала создается список всех допустимых строк, где каждая строка представляет собой уникальную перестановку четырех букв без повторений. Затем осуществляется систематический перебор возможных комбинаций из четырех таких строк с поэтапной проверкой условий: сначала анализируется уникальность букв в столбцах между парами строк, а затем проверяется общее количество каждой буквы во всей матрице.

После завершения перебора программа выводит общее количество найденных решений (576 вариантов) и демонстрирует один случайно выбранный пример корректной матрицы. Интерфейс позволяет сбрасывать результаты и повторять расчет, а также обрабатывает возможные ошибки выполнения. Алгоритм оптимизирован за счет раннего прекращения проверки заведомо неверных комбинаций, что значительно сокращает время работы. Полученные результаты наглядно отображаются в графическом интерфейсе с возможностью быстрого обновления данных.

## **Реализация кода на Python**

from tkinter import \*

from random import randint

class Rgr01:

def \_\_init\_\_(self) -> None:

self.win = Tk()

self.tk\_widgets = []

def \_\_gen\_rows(self) -> list:

'''Генерирует все возможные строки без повторений'''

letters = ['a', 'b', 'c', 'd']

rows = []

for i in letters:

for j in letters:

if j == i:

continue

for k in letters:

if k == i or k == j:

continue

for m in letters:

if m == i or m == j or m == k:

continue

rows.append([i, j, k, m])

return rows

def \_\_valid\_checker(self, matrix) -> bool:

"""Проверяет уникальность букв и их количество по типам"""

# Проверка столбцов

for col in range(4):

column = [matrix[row][col] for row in range(4)]

if len(column) != len(set(column)):

return False

# Проверка количества букв

letter\_counts = {'a': 0, 'b': 0, 'c': 0, 'd': 0}

for row in matrix:

for letter in row:

letter\_counts[letter] += 1

return all(count == 4 for count in letter\_counts.values())

def find\_matrix(self) -> list:

"""Находит все подходящие матрицы 4x4"""

rows = self.\_\_gen\_rows()

res\_matrix = []

# Перебираем все возможные комбинации из 4 строк

for row1 in rows:

for row2 in rows:

if any(row1[col] == row2[col] for col in range(4)):

continue

for row3 in rows:

if any(row1[col] == row3[col] or row2[col] == row3[col] for col in range(4)):

continue

for row4 in rows:

if any(row1[col] == row4[col] or row2[col] == row4[col] or row3[col] == row4[col] for col in range(4)):

continue

matrix = [row1, row2, row3, row4]

if self.\_\_valid\_checker(matrix):

res\_matrix.append([row.copy() for row in matrix])

return res\_matrix

def error(self, er='Ошибка') -> None:

er\_win = Toplevel(self.win)

er\_win.title("Ошибка")

er\_win.geometry("350x150")

Label(er\_win, text=er, font="30", bg="red").pack()

def tkinter\_fun(self) -> None:

self.win.title("Домашняя работа №1 Вершинин АТ-24-01")

self.win.geometry("700x400")

self.win.attributes("-alpha", 0.96)

self.win.config(bg="bisque")

Canvas(bg="peachpuff", width=395, height=140).place(x=1, y=1)

Label(text="Вершинин Сергей АТ-24-01\nВариант №3", font="30", bg="peachpuff").place(x=80, y=10)

Button(self.win, text="Решить задачу", bg="bisque",font="15", command=self.disp\_info).place(x=130, y=80)

Button(self.win, text="RESET", bg="red", command=self.rst).place(x=5, y=5)

def matrix\_for\_disp(self, mat) -> str:

res = ''

for i in range(len(mat)):

for j in range(len(mat[i])):

res += str(mat[i][j]) + ' '

res += '\n'

return res

def disp\_info(self) -> None:

self.rst()

matrixes = self.find\_matrix()

frame\_ans = Frame(master=self.win, bg="bisque")

frame\_ans.place(x=530, y=40)

frame\_examples = Frame(master=self.win, bg="bisque")

frame\_examples.place(x=220, y=180)

Label(frame\_ans, text=f"ответ\n{len(matrixes)}", font="30", bg="bisque").pack()

Label(frame\_examples, text="пример подходящей матрицы", font="30", bg="bisque").pack()

Label(frame\_examples, text=f"{self.matrix\_for\_disp(matrixes[randint(0, len(matrixes)-1)])} ", font="30", bg="bisque").pack()

self.tk\_widgets.extend([frame\_ans, frame\_examples])

def rst(self) -> None:

try:

for tk\_widget in self.tk\_widgets:

tk\_widget.destroy()

except Exception as e:

self.error(e)

def run(self) -> None:

self.tkinter\_fun()

self.win.mainloop()

app = Rgr01()

app.run()).place(x=10, y=60)

win.mainloop()

Примеры работы программы даны в Приложении 1.

# **Заключение**

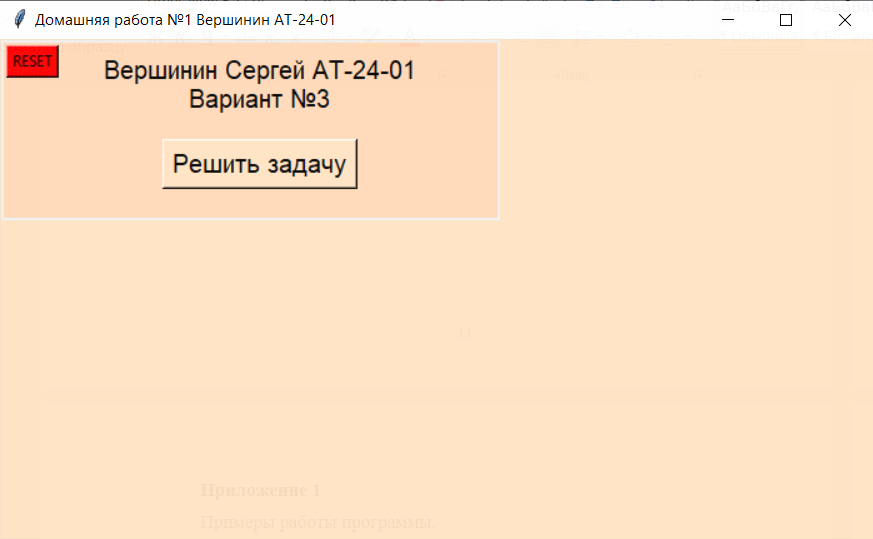
В ходе выполнения работы был разработан и реализован алгоритм для решения комбинаторной задачи расстановки 16 букв в матрице 4×4 с заданными ограничениями. Примененный метод полного перебора с оптимизациями позволил найти все 576 возможных вариантов расстановки, удовлетворяющих условиям задачи.

# **Источники**

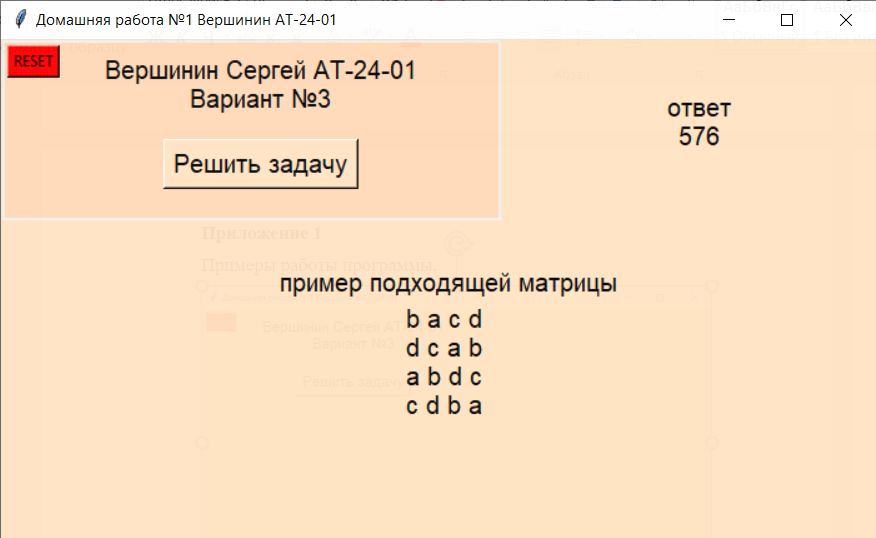
1. Руководство PEP8 Python // [Электронный ресурс] // https://peps.python.org/pep-0008/ // Дата обращения: 20.04.2025
2. Tkinter: GUI Programming in Python // [Электронный ресурс] — //  https://docs.python.org/3/library/tkinter.html// Дата обращения: 20.04.2025

# **Приложение 1**

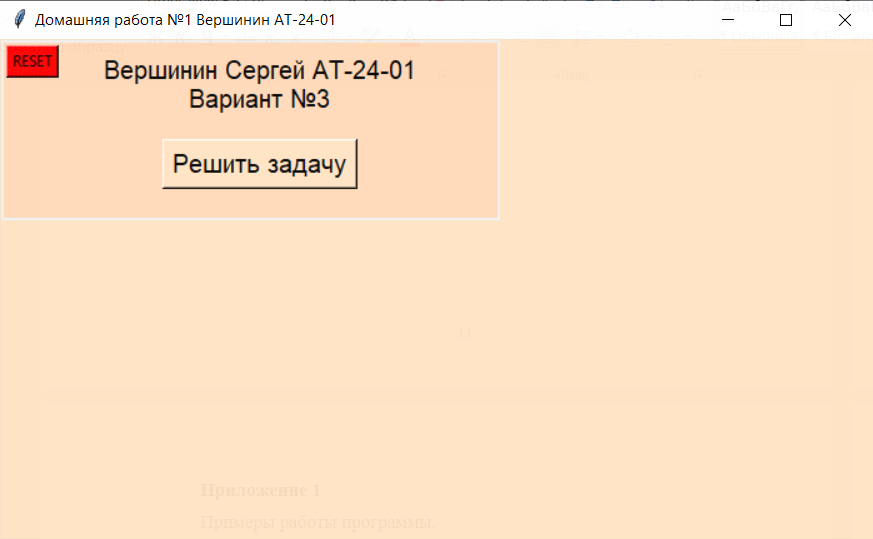
Примеры работы программы.



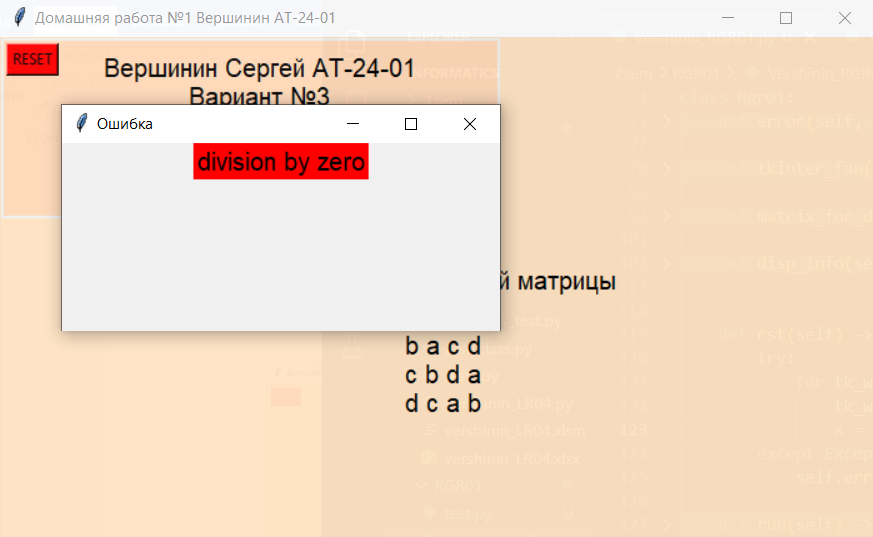
*Пример работы программы после запуска.*



*Пример работы программы после нажатия кнопки “Решить задачу”.*



*Пример работы программы после нажатия кнопки “RESET”.*



*Пример обработки ошибок (смоделированная ситуация).*