Министерство науки и высшего образования российской федерации

Пензенский государственный университет  
Кафедра «Вычислительной техники»

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

к курсовому проектированию

по курсу: «Логика и основы алгоритмизации в инженерных задачах»

на тему: «Реализация алгоритма поиска путей в лабиринте»

Выполнил:

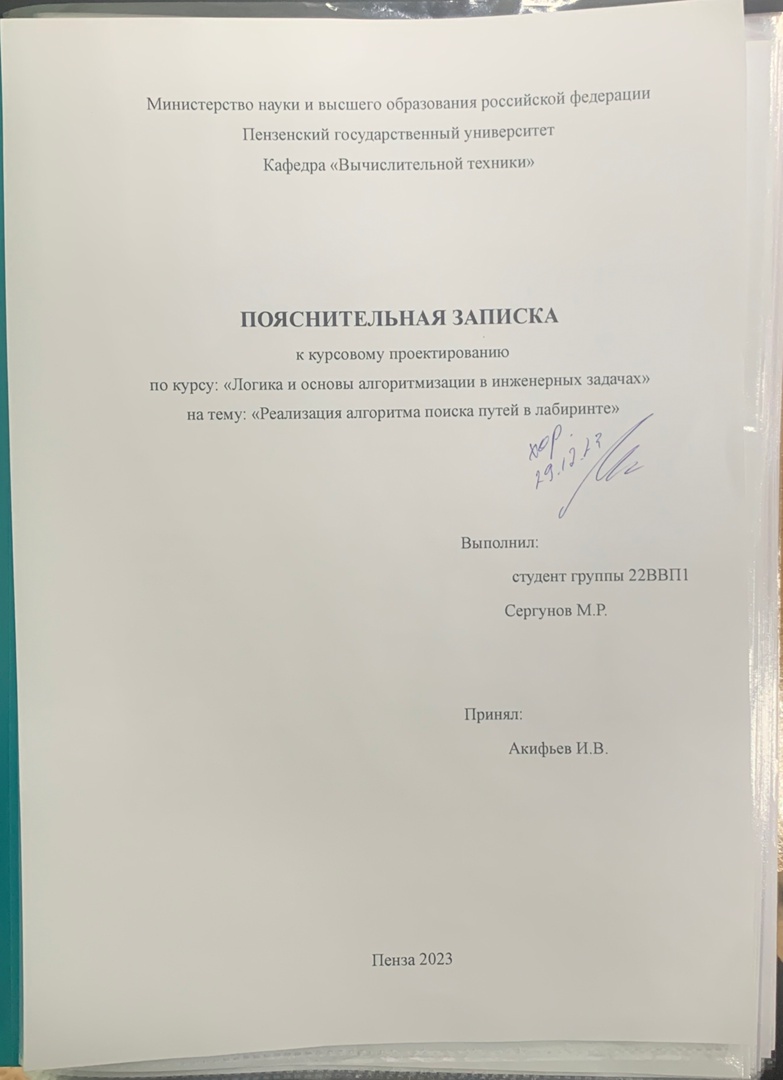
студент группы 22ВВП1

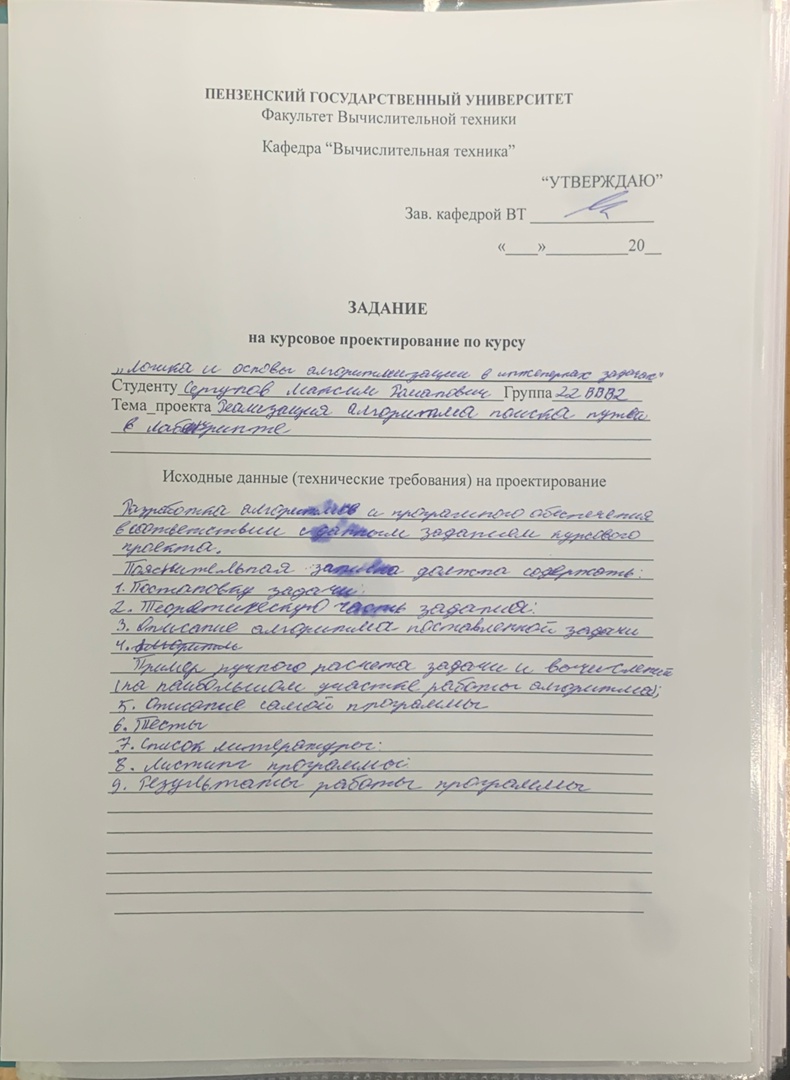
Сергунов М.Р.

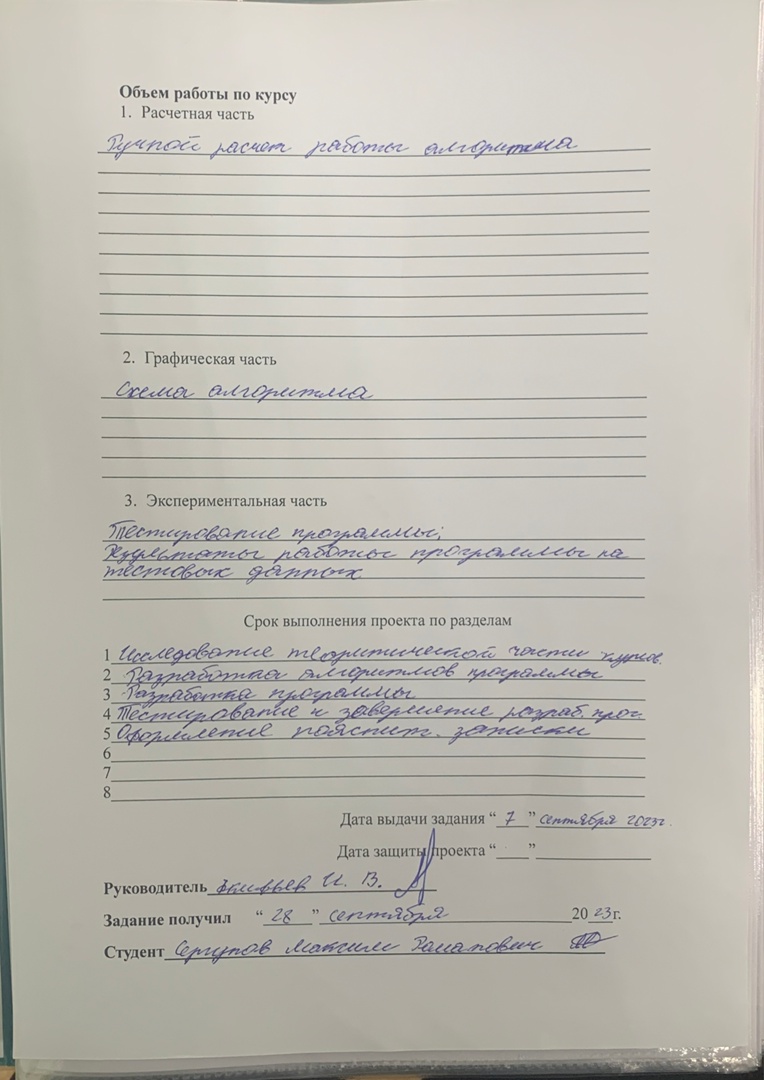
Принял:

Акифьев И.В.

Пенза 2023







**Содержание**

[**Реферат.** 3](#_Toc154668937)

[**Введение** 4](#_Toc154668938)

[**1. Постановка задачи** 6](#_Toc154668939)

[**2. Теоретическая часть программы** 7](#_Toc154668940)

[**3. Описание алгоритма программы** 8](#_Toc154668941)

[**4. Описание программы** 9](#_Toc154668942)

[**5. Тестирование** 14](#_Toc154668943)

[**6. Ручной расчет задачи** 17](#_Toc154668944)

[**Список литературы** 19](#_Toc154668945)

[**Приложение А** 20](#_Toc154668946)

[**Код программы** 20](#_Toc154668947)

[**Приложение B** 26](#_Toc154668948)

[**Результаты работы программы** 26](#_Toc154668949)

# **Реферат**

Отчет 27 стр., 20 рисунков.

ГРАФ, ТЕОРИЯ ГРАФОВ, ЛАБИРИНТ, АЛГОРИТМ ПОИСКА В ШИРИНУ.

Цель исследования – разработка программы для поиска путей в

лабиринте.

В работе рассмотрена реализация алгоритма поиска в ширину для поиска путей в лабиринте. Установлено, что с помощью данного алгоритма можно найти кротчайший путь в лабиринте.

# **Введение**

Тео́рия гра́фов — раздел дискретной математики, изучающий графы, одна из ветвей топологии. В самом общем смысле граф — это множество точек (вершин, узлов), которые соединяются множеством линий (рёбер, дуг).

Теория графов (то есть систем линий, соединяющих заданные точки) включена в учебные программы для начинающих математиков, поскольку:

1 как и геометрия, обладает наглядностью;

2 как и теория чисел, проста в объяснении и имеет сложные нерешённые задачи;

3 не имеет громоздкого математического аппарата («комбинаторные методы нахождения нужного упорядочения объектов существенно отличаются от классических методов анализа поведения систем с помощью уравнений»;

4 имеет выраженный прикладной характер.

На протяжении более сотни лет развитие теории графов определялось в основном проблемой четырёх красок. Решение этой задачи в 1976 году оказалось поворотным моментом истории теории графов, после которого произошло её развитие как основы современной прикладной математики.

Универсальность графов незаменима при проектировании и анализе коммуникационных сетей.

Теория графов, как математическое орудие, приложима как к наукам о поведении (теории информации, кибернетике, теории игр, теории систем, транспортным сетям), так и к чисто абстрактным дисциплинам (теории множеств, теории матриц, теории групп и так далее).

Несмотря на разнообразные, усложнённые, малопонятные и специализированные термины многие модельные (схемные, структурные) и конфигурационные проблемы переформулируются на языке теории графов, что позволяет значительно проще выявить их концептуальные трудности.

В разных областях знаний понятие «граф» может встречаться под следующими названиями:

1 структура (гражданское строительство);

2 сеть (электротехника);

3 социограмма (социология и экономика);

4 молекулярная структура (химия);

5 навигационная карта (картография);

6 распределительная сеть (энергетика)

и так далее.

Часто требуется производить различные операции над графами, например нахождение кратчайшего расстояния между вершинами. Для этого придуманы различные алгоритмы, один из которых Алгоритм Дейкстры.

Поиск в ширину (англ. breadth-first search) — один из основных алгоритмов на графах, позволяющий находить все кратчайшие пути от заданной вершины и решать многие другие задачи.

Поиск в ширину также называют обходом — так же, как поиск в глубину и все другие обходы, он посещает все вершины графа по одному разу, только в другом порядке: по увеличению расстояния до начальной вершины.

В качестве среды разработки мною была выбрана среда PyCharm Community Edition 2023.1, язык программирования – Python.

Целью данной курсовой работы является разработка программы на языке Python, который является широко используемым. Именно с его помощью в данном курсовом проекте реализуется алгоритм поиска кратчайшего пути в лабиринте

# **1. Постановка задачи**

Требуется разработать программу, которая находит путь в лабиринте.

В программе должно быть текстовое меню.

Также должна быть реализована возможность выбора случайной или ручного задания матрицы .

Должна быть предусмотрена возможность сохранить результат работы программы в файл.

# **2. Теоретическая часть программы**

Данный алгоритм способен найти кратчайшие расстояние в лабиринте

Алгоритм двигаемся от начальной вершины (точки, места) в определенном направлении (по определенному пути) до тех пор, пока не достигнем конца пути или пункта назначения (искомой вершины). Если достигли конца пути, но он не является пунктом назначения, то возвращаемся назад (к точке разветвления или расхождения путей) и идем по другому маршруту.

Теоретическое описание работы алгоритма поиска в ширину по шагам:

Шаг 1. Выберем начальную вершину и пометим её как посещённую. Добавим начальную вершину в очередь.

Шаг 2. Пока очередь не пуста

Шаг 3. Извлекаем вершину из очереди.

Шаг 4. Посещаем все её непосещённые соседи.

Шаг 5. Помечаем каждого соседа и добавляем в конец очереди.

Шаг 6. Расстояние до каждой вершины минимально.

# **3. Описание алгоритма программы**

С помощью функции mazemake() мы генерируем лабиринт. Вначале лабиринт массив заполняется стенками. Затем мы выбираем в лабиринте ячейку (0; 0) и начиная от нее начинаем создавать проходы. Далее начиная с ячейки (1; 1) мы начинаем поиск путей в лабиринте к ячейке (N-1; N1), где N – размер лабиринта Создание множества visited для отслеживания посещенных точек. Начало цикла: пока очередь не пуста, продолжаем выполнение алгоритма. Извлечение текущей точки из очереди. Если текущая точка - конечная, возвращается длина пути и сам путь. Итерация по соседям текущей точки. Если соседняя точка не посещена и она является проходимой (содержит 0), добавляем ее в очередь с обновленным расстоянием и путем. Пометка текущей точки как посещенной. Если очередь опустела, и конечная точка не достигнута, возвращается -1 и пустой путь.

Ниже представлен псевдокод функции **shortest\_path ()**:

Функция shortest\_path(matrix):

rows, cols = размеры(matrix)

направления = [(0, 1), (1, 0), (0, -1), (-1, 0)]

q = queue.Queue()

положить в очередь((0, 0, 0, [(0, 0)]))# (строка, столбец, расстояние, путь)

посещенные = set()

Пока очередь не пуста:

row, col, dist, path = q.get()

Если (row, col) == (rows - 1, cols - 1):

return dist+1, path

Для каждого направления (dr, dc) в направлениях:

nr, nc = row + dr, col + dc

Если 0 <= nr < rows and 0 <= nc < cols and matrix[nr][nc] == 0 and (nr, nc) not in visited

положить\_в\_очередь(nr, nc, dist + 1, path + [(nr, nc)])

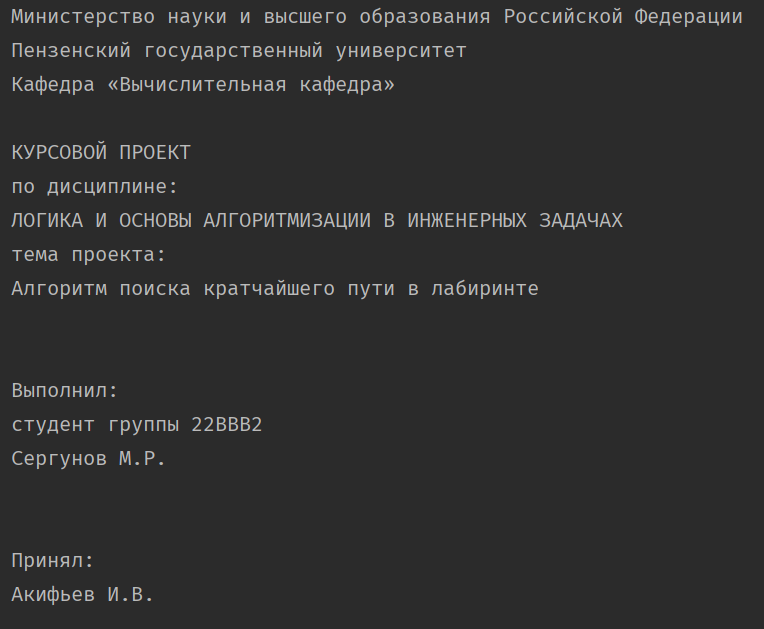
добавить\_в\_посещенные(nr, nc)

# **4. Описание программы**

Для написания программы использован язык программирования Python.

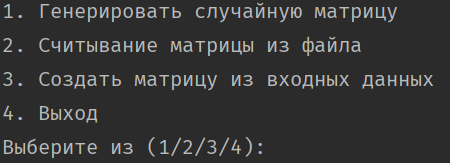
Данная программа является многомодульной, поскольку состоит из нескольких функций:

Работа программы начинается с вывода титульного листа на экран:



1. Титульный лист работы

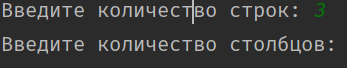
Далее на экран выводится текстовое меню:



2. Меню программы

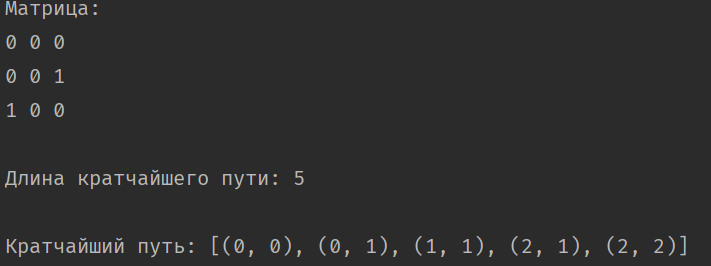
Пользователь может сгенерировать случайную матрицу или ввести матрицу вручную.

При создании случайной матрицы у пользователя запрашивается его размер (количество строк и столбцов). Размер можно ввести используя клавиатуру:



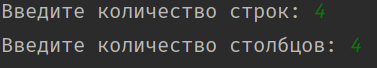
3. Ввод размера матрицы

Далее выводятся кратчайшие расстояния:



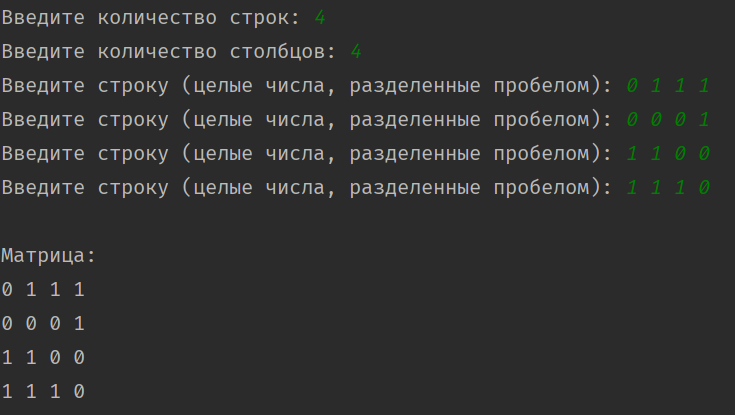
4. Результат случайно сгенерированной матрицы

При задании матрицы вручную сначала нужно ввести размеры матрицы:



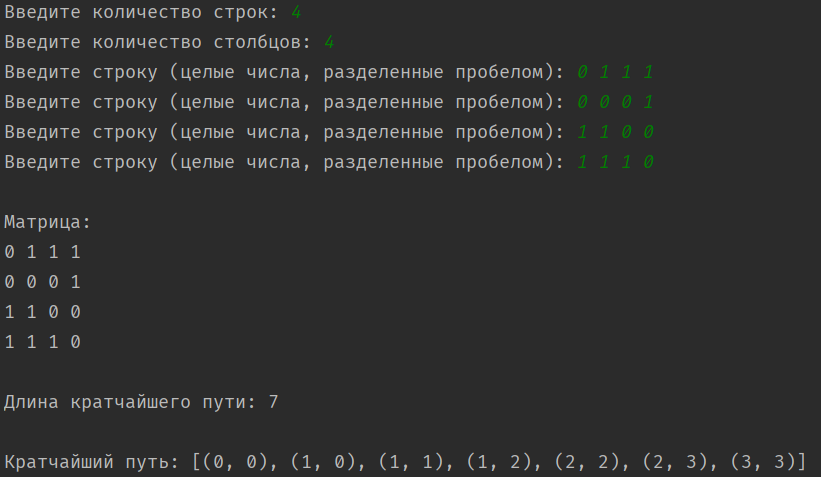
5. Ввод размера матрицы

После запрашивается каждая строка:



6. Ввод строк матрицы и вывод этой матрицы

После вывода матрицы происходит вывод кратчайшего пути на экран:



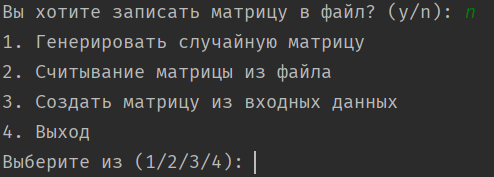
7. Результат лабиринта, введенного вручную

После успешного завершения программы пользователя спрашивают хочет ли он сохранить результат.



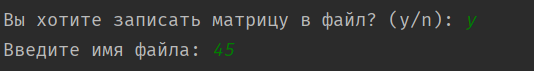
8.Запрос пользователю про сохранение кратчайшего пути в матрице

Если пользователь отправил в консоль «n», то программа не сохраняет результат, и пользователь обратно возвращается в главное меню

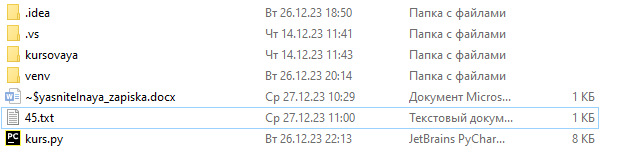


9. Пользователь отказался сохранять результат

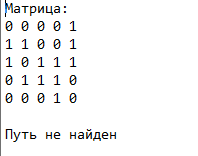
Если пользователь отправил в консоль «y», то программа запросит имя файла и создаст его.



10. Пользователь согласился сохранить файл и ввел его имя.



11. Создание файла

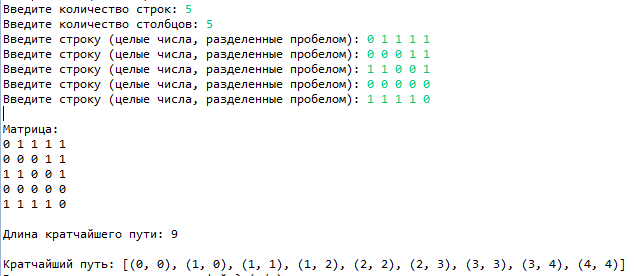


12. Сохранение матрицы в файл

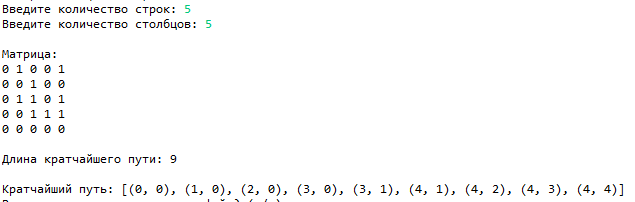
# **5. Тестирование**

Тестирование проводилось и в процессе разработки и после написания программы. В ходе тестирования было выявлено и исправлено множество проблем, связанных с вводом данных, выводом данных в файл, алгоритмом программы, взаимодействием функций.

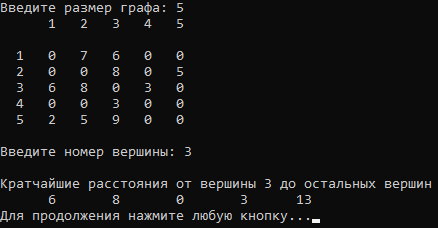
Ниже представлены результаты работы программы.



13. Результат матрицы, введенную вручную

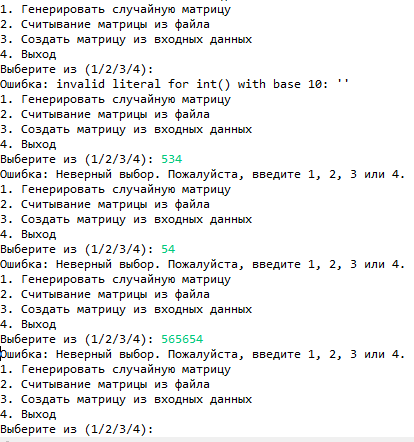


14. Результат случайно сгенерированной матрицы



15. Результат случайно сгенерированного ориентированного графа

Также в программе предусмотрена проверка на некорректный ввод данных. При попытке ввода пользователем некорректного значения (такого, с каким программа не должна работать) на экране отображается сообщение об ошибке и запрашивается повторный ввод.



16. Проверка ввода некорректных значений.

Таблица 1 - описание поведения программы при тестировании

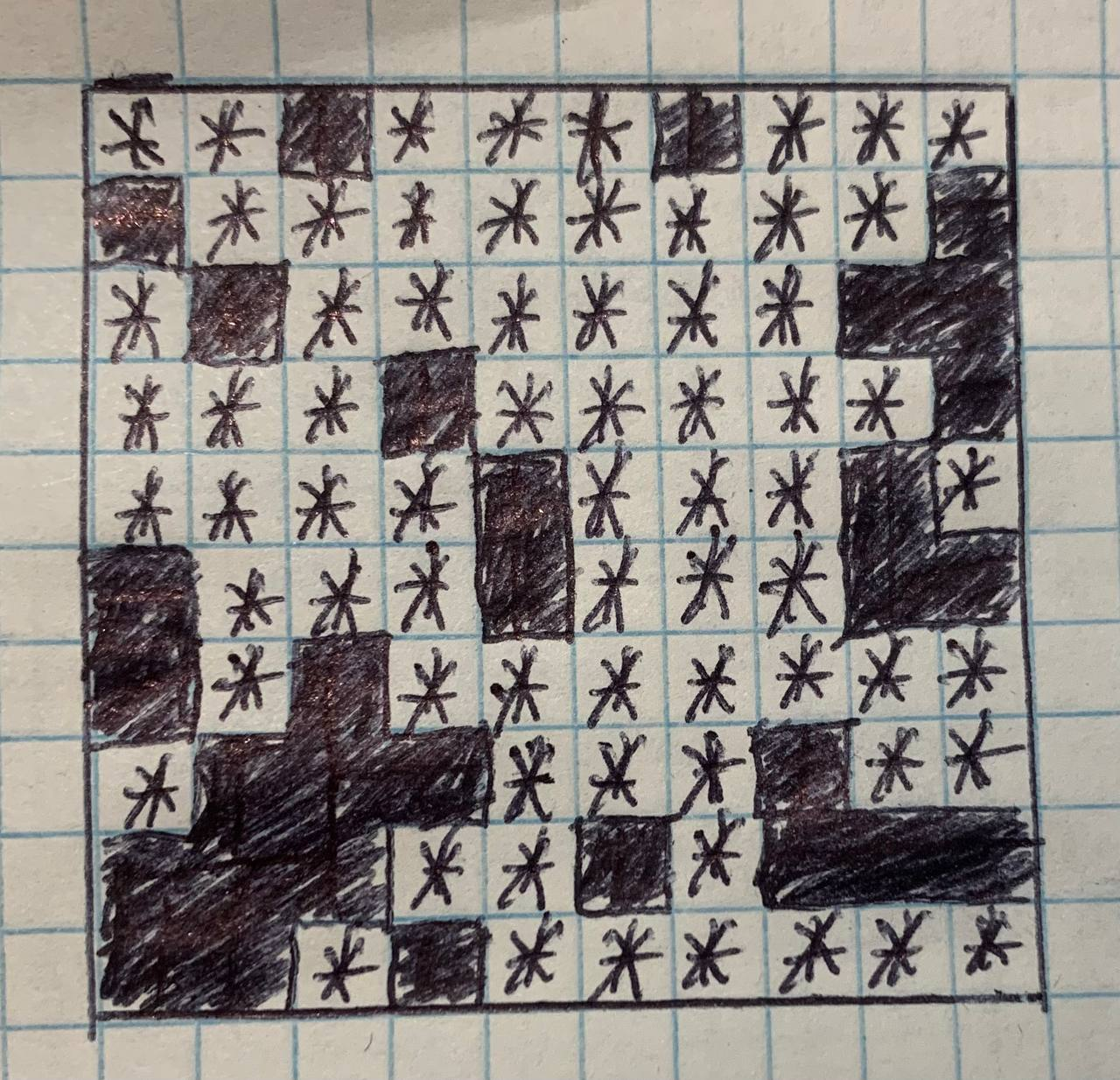
|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Запуск программы | Появление меню на экране и сообщения о выборе нужной операции | Верно |
| Выбор генерации матрицы | Запрос ввода размеры матрицы | Верно |
| Ручной ввод матрицы | Запрос ввода строк и столбцов | Верно |
| Сохранение результата | Результат записан в файл | Верно |
| Результат работы программы | Верный вывод кратчайшего расстояния | Верно |

В результате тестирования было выявлено, что программа работает успешно.

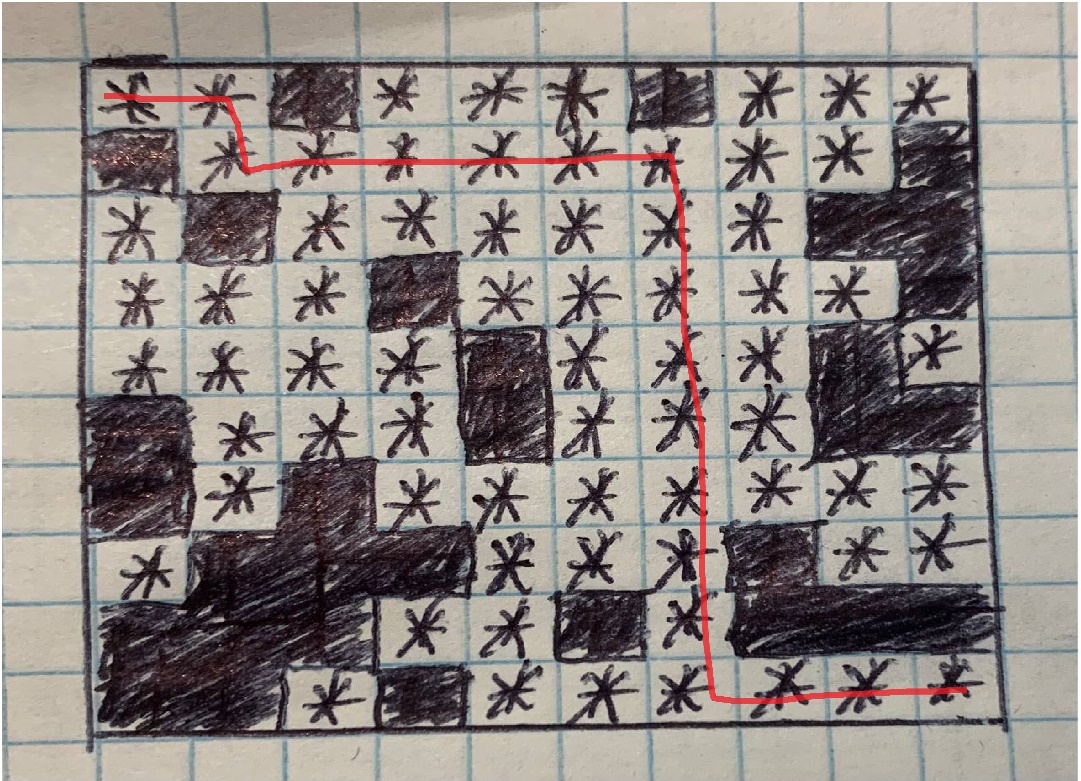
# **6. Ручной расчет задачи**

Проведем проверку программы посредством ручных вычислений на примере лабиринта с размерами 10x10.

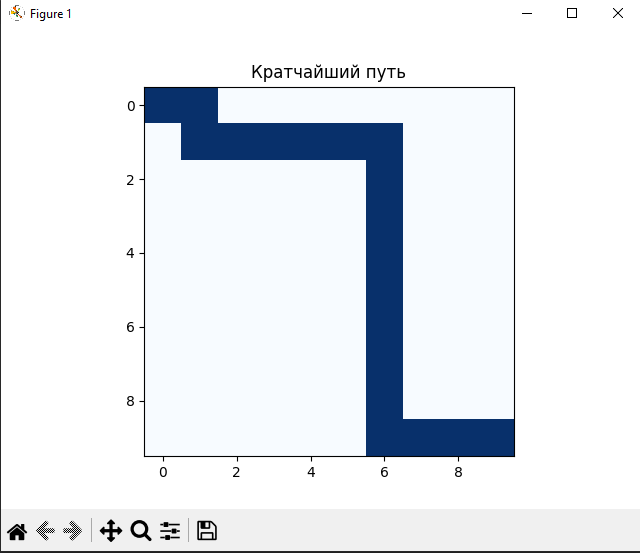
Начальная точка - верхний левый угол (0,0), а конечная - нижний правый угол (9,9). Алгоритм BFS начинает с исследования соседей начальной точки и постепенно распространяется по всему лабиринту. Начинаем с (0,0) и добавляем его в очередь. Текущий путь: (0,0), длина пути: 0. Извлекаем (0,0) из очереди. Исследуем его соседей и добавляем их в очередь. Текущий путь: (0,0), (1,0) (0,0), (1,0), длина пути: 1. Добавляем (0,1) в очередь. Текущий путь: (0,0), (1,0), (0,1) (0,0), (1,0), (0,1), длина пути: 2. И так далее. Проходим по всем возможным путям, пока не достигнем конечной точки.

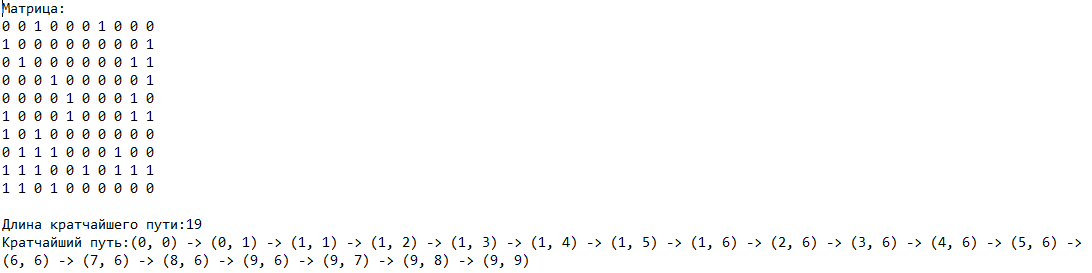


16. Ручной расчет путей в лабиринте



17. Ручной расчёт кратчайшего пути





18. - Результат работы программы

**Заключение**

Таким образом, в процессе создания данного проекта разработана программа, реализующая алгоритм поиска кротчайшего пути в лабиринте в PyCharm Community Edition 2023.1.

При выполнении данной курсовой работы были получены навыки разработки программ и освоены приемы создания алгоритма, для генерации лабиринте. Приобретены навыки по осуществлению алгоритма поиска кротчайшего пути в лабиринте. Углублены знания языка программирования Python.

Недостатком разработанной программы является примитивный пользовательский интерфейс. Потому что программа работает в консольном режиме, не добавляющем к сложности языка сложность программного оконного интерфейса.

Программа имеет небольшой, но достаточный для использования функционал возможностей.

# **Список литературы**

1. Кормен Т., Лейзерсон Ч., Ривест Р. Алгоритмы: Построение и анализ - М.: МЦНМО, 2001. - 960 с.

2. Кристофидес Н. «Теория графов. Алгоритмический подход» - Мир, 1978

3. Уилсон Р. Введение в теорию графов. Пер. с анг. 1977. 208 с.

4. Майкл Доусон «Программируем на Python»

5. З. Оре О. Графы и их применение: Пер. с англ. 1965. 176 с

# **Приложение А**

# **Код программы**

import random

import queue

import matplotlib.pyplot as plt

def titulniyList():

print("\n Министерство науки и высшего образования Российской Федерации")

print(" Пензенский государственный университет")

print(" Кафедра «Вычислительная кафедра»\n")

print(" КУРСОВОЙ ПРОЕКТ")

print(" по дисциплине:\n ЛОГИКА И ОСНОВЫ АЛГОРИТМИЗАЦИИ В ИНЖЕНЕРНЫХ ЗАДАЧАХ")

print(" тема проекта:\n Алгоритм поиска кратчайшего пути в лабиринте\n\n")

print(" Выполнил:\n студент группы 22ВВВ2\n Сергунов М.Р. \n\n")

print(" Принял:\n Акифьев И.В.\n\n Пенза 2023\n")

def generate\_random\_maze(rows, cols):

density = 0 # Пример значения плотности, можно настроить по вашему усмотрению

maze = [[random.choice([0, 1]) for \_ in range(cols)] for \_ in range(rows)]

for row in maze:

for i in range(cols):

if random.random() < density:

row[i] = 1

maze[0][0] = 0 # Start point

maze[rows - 1][cols - 1] = 0 # End point

return maze

def print\_matrix(matrix):

for row in matrix:

print(" ".join(map(str, row)))

def write\_to\_file(matrix, result, path, filename):

if not filename.endswith('.txt'):

filename += '.txt'

with open(filename, "w") as file:

file.write("Матрица:\n")

for row in matrix:

file.write(" ".join(map(str, row)) + "\n")

if result != -1:

file.write("\nДлина кратчайшего пути:" + str(result))

file.write("\nКратчайший путь:" + " -> ".join(map(str, path)))

else:

file.write("\nПуть не найден")

def read\_from\_file(filename):

with open(filename, "r") as file:

matrix = []

for line in file:

if line.strip() == "Матрица:":

break

for line in file:

if not line.strip():

break

matrix.append(list(map(int, line.strip().split())))

return matrix

def create\_matrix\_from\_input():

while True:

try:

rows = int(input("Введите количество строк: "))

cols = int(input("Введите количество столбцов: "))

if rows <= 0 or cols <= 0:

raise ValueError("Строки и столбцы должны быть целыми положительными числами.")

break

except ValueError as e:

print(f"Error: {e}")

matrix = []

for \_ in range(rows):

while True:

try:

row = list(map(int, input("Введите строку (целые числа, разделенные пробелом): ").split()))

if len(row) != cols or any(val not in {0, 1} for val in row):

raise ValueError(f"Недопустимая строка. Она должна содержать целые числа {cols} (0 или 1).")

break # Если пользователь ввел корректную строку, выходим из цикла

except ValueError as e:

print(f"Ошибка: {e}")

matrix.append(row)

return matrix

def shortest\_path(matrix):

rows, cols = len(matrix), len(matrix[0])

directions = [(0, 1), (1, 0), (0, -1), (-1, 0)]

q = queue.Queue()

q.put((0, 0, 0, [(0, 0)])) # (row, col, distance, path)

visited = set()

while not q.empty():

row, col, dist, path = q.get()

if (row, col) == (rows - 1, cols - 1):

return dist+1, path

for dr, dc in directions:

nr, nc = row + dr, col + dc

if 0 <= nr < rows and 0 <= nc < cols and matrix[nr][nc] == 0 and (nr, nc) not in visited:

q.put((nr, nc, dist + 1, path + [(nr, nc)]))

visited.add((nr, nc))

return -1, []

def plot\_path(matrix, path):

rows, cols = len(matrix), len(matrix[0])

path\_matrix = [[0] \* cols for \_ in range(rows)]

for node in path:

path\_matrix[node[0]][node[1]] = 1

plt.imshow(path\_matrix, cmap='Blues', interpolation='nearest')

plt.title('Shortest Path')

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

titulniyList()

while True:

try:

print("1. Генерировать случайную матрицу")

print("2. Считывание матрицы из файла")

print("3. Создать матрицу из входных данных")

print("4. Выход")

choice = int(input("Выберите из (1/2/3/4): "))

if choice == 4:

break

if choice not in {1, 2, 3}:

raise ValueError("Неверный выбор. Пожалуйста, введите 1, 2, 3 или 4.")

if choice == 1:

while True:

try:

rows = int(input("Введите количество строк: "))

cols = int(input("Введите количество столбцов: "))

if rows <= 0 or cols <= 0:

raise ValueError("Строки и столбцы должны быть целыми положительными числами.")

break

except ValueError as e:

print(f"Error: {e}")

maze = generate\_random\_maze(rows, cols)

elif choice == 2:

filename = input("Введите название файла: ")

maze = read\_from\_file(filename)

elif choice == 3:

maze = create\_matrix\_from\_input()

print("\nМатрица:")

print\_matrix(maze)

result, path = shortest\_path(maze)

if result != -1:

print("\nДлина кратчайшего пути:", result)

print("\nКратчайший путь:", path)

plot\_path(maze, path)

else:

print("\nПуть не найден.")

while True:

write\_choice = input("Вы хотите записать матрицу в файл? (y/n): ")

if write\_choice.lower() in {'y', 'n'}:

break

else:

print("Некорректный выбор. Пожалуйста, введите 'y' or 'n'.")

if write\_choice.lower() == 'y':

filename = input("Введите имя файла: ")

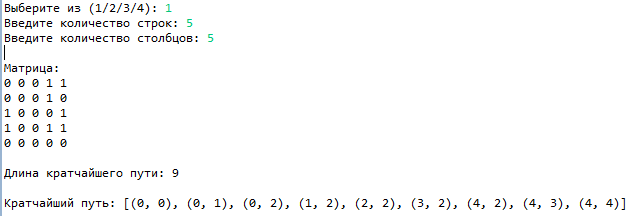
write\_to\_file(maze, result, path, filename)

except ValueError as e:

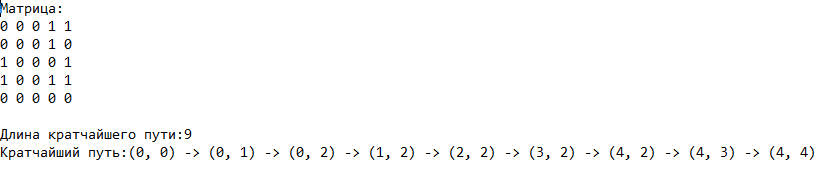
print(f"Ошибка: {e}")

# **Приложение B**

# **Результаты работы программы**



19. Результат случайно сгенерированной матрицы



20. Результат сохранения работы программы в файл