**Реферат**

**Отчет 33 стр, 12 рисунков.**

**ГРАФ, ЛАБИРИНТ, ПОИСК В ШИРИНУ.**

**Цель исследования –** разработка программы, способная реализоавать поиск путей в лабиринтах, использую алгоритм поиска в

ширину.

В работе рассмотрены правила поиска в ширину, на основе которых

находится компонента сильной связности орграфа. Установлено, что с

помощью данного алгоритма можно выделить компоненту сильной связности

подграфа орграф, которого несвязен, слабо связан или сильно связан.

**Содержание**

[Реферат 3](#_Toc153478642)

[Введение 4](#_Toc153478643)

[1. Постановка задачи 5](#_Toc153478644)

[2. Теоретическая часть задания 6](#_Toc153478645)

[3. Описание алгоритма программы 7](#_Toc153478646)

[4. Описание программы 10](#_Toc153478647)0

[5. Отладка и тестирование 14](#_Toc153478648)2

[6. Ручной расчет 20](#_Toc153478649)5

[Заключение 23](#_Toc153478652)6

[Список литературы 24](#_Toc153478653)7

[Приложение А. Листинги программы 25](#_Toc153478654)18

# Введение

BFS — это алгоритм обхода графа (в нашем случае, лабиринта), который исследует граф по уровням. Это значит, что сначала он посещает все соседние узлы от начальной точки, затем всех соседей этих соседей и так далее. Ключевая особенность BFS в том, что он гарантирует нахождение кратчайшего пути (если такой путь существует) в графе, где все ребра имеют одинаковый вес (например, в лабиринте, где каждое перемещение на одну клетку имеет одинаковую стоимость).

Для реализации данного алгоритма была выбрана среда разработки PyCharm, а язык программирования python использован в качестве основного инструмента. Этот выбор обусловлен широким распространением языка python и его применимостью в различных областях программирования.

# Постановка задачи

Необходимо разработать программу для алгоритма поиска путей в лабиринте.

Программа должна отвечать следующим требованиям:

1. Текстовое или графическое меню для удобства использования программы.

2. Возможность задания пользователем размера лабиринта.

3.Возможность выбора автоматического (случайного) или загрузка из файла задания лабиринта.

4. Возможность сохранения результатов работы программы.

5. Устройство ввода информации: клавиатура/мышь.

# Теоретическая часть задания

Граф — это абстрактный тип данных, предназначенный для реализации концепций неориентированного графа и ориентированного графа из области теории графов в математике.

Структура данных графа состоит из конечного (и, возможно, изменяемого) набора вершин (также называемых узлами или точками) вместе с набором неупорядоченных пар этих вершин для неориентированного графа или набором упорядоченных пар для ориентированного графа. Эти пары известны как ребра (также называемые связями или линиями), а для ориентированного графа также известны как ребра, но также иногда как стрелки или дуги. Вершины могут быть частью структуры графа или могут быть внешними объектами, представленными целочисленными индексами или ссылками.

Алгоритм обхода в ширину — это метод нахождения кратчайшего путей от одной вершины к другой.

# Описание алгоритма программы

Перед началом разработки программы был проведен анализ требований и определены основные функциональности, которые она должна включать. Основные компоненты программы включают в себя генерацию и отображение лабиринта, обработка его путей, сохранение лабиринта в файл.

На вход подаётся кол-во вершин для генерации лабиринта. После окончания генерации или копирования из файла, можно выполнить алгоритм обхода в ширину или сохранить его в файл.

Была реализована функция read, которая отвечает за чтение из файла и обход в глубину графа.

**Функция read:**

async def read(message: Message, state:FSMContext):  
 try:  
 file\_id = message.document.file\_id  
 file\_info = await bot.get\_file(file\_id)  
 file\_path = file\_info.file\_path  
 file\_name = message.document.file\_name  
  
 download\_folder = "downloaded\_files"  
 os.makedirs(download\_folder, exist\_ok=True)  
 local\_file\_path = os.path.join(download\_folder, file\_name)  
  
 await bot.download\_file(file\_path, local\_file\_path)  
  
 with open(local\_file\_path, 'r', encoding='utf-8') as f:  
 for line in f:  
 a = len(line)  
 a //= 2  
  
 maze = [['' for \_ in range(a)] for \_ in range(a)]  
  
 with open(local\_file\_path, 'r', encoding='utf-8') as file:  
 row\_index = 0  
 for line in file:  
 line = line.strip()  
 if line and row\_index < len(maze):  
 # Разделяем строку на графемовые кластеры  
 symbols = regex.findall(r'\X', line)  
 for col\_index, char in enumerate(symbols):  
 if col\_index < len(maze[row\_index]):  
 maze[row\_index][col\_index] = char  
 row\_index += 1  
  
 start = (1, 1) # Начальная точка (y, x)  
 width, height = len(maze[0]), len(maze)  
 goal = (a - 2, a - 2) # Конечная точка  
 queue = deque([start]) # Очередь для BFS  
 visited = {start} # Множество посещённых узлов  
 parent = {start: None} # Словарь для отслеживания пути  
  
 directions = [(0, 1), (1, 0), (0, -1), (-1, 0)] # Направления (вперед, вправо, назад, влево)  
  
 while queue:  
 current = queue.popleft()  
  
 if current == goal:  
 break # Найдено решение  
  
 for d in directions:  
 neighbor = (current[0] + d[0], current[1] + d[1])  
 if (0 <= neighbor[0] < height and  
 0 <= neighbor[1] < width and  
 (maze[neighbor[0]][neighbor[1]] == '⬜️' or maze[neighbor[0]][neighbor[1]] == '🚀' or  
 maze[neighbor[0]][neighbor[1]] == '🏁') and  
 neighbor not in visited):  
 visited.add(neighbor)  
 parent[neighbor] = current  
 queue.append(neighbor)  
  
 # Восстановление пути  
 path = []  
 step = goal  
 print(maze)  
 while step is not None:  
 path.append(step)  
 step = parent[step]  
  
 path.reverse() # Путь от старта к финишу  
 for y, x in path:  
 if (not (y == 1 and x == 1) and not (y == a - 2 and x == a - 2)):  
 maze[y][x] = '👣'  
  
 text = ''  
  
 for row in maze:  
 text += ''.join(row)  
 text += '\n'  
  
 if (a <= 15):  
 with open('resh.txt', 'w', encoding='utf-8') as f:  
 f.write(text)  
 doc = FSInputFile('resh.txt')  
 # Отправляем файл пользователю  
 with (open('resh.txt', 'rb') as f):  
 await message.answer\_document(doc, caption=f'<code>{text}</code>', reply\_markup=kb.back)  
  
 # Удаляем файл после отправки (по желанию)  
 os.remove('resh.txt')  
 else:  
 with open('resh.txt', 'w', encoding='utf-8') as f:  
 f.write(text)  
 doc = FSInputFile('resh.txt')  
 # Отправляем файл пользователю  
 with (open('resh.txt', 'rb') as f):  
 await message.answer\_document(doc, reply\_markup=kb.back)  
  
 # Удаляем файл после отправки (по желанию)  
 os.remove('resh.txt')  
  
 except Exception as e:  
 await message.answer('ошибка попробуйте ещё раз')

Ниже представлен псевдокод функции read:

1. async def read(message: Message, state:FSMContext):  
:

a) прочитать файл содержащий лабиринт

a = длине первой строки файла

б)создать матрицу  
maze = a\*a

в) ещё раз считываем файл и заполняем созданную матрицу учитывая что в файле не простые символы а эмодзи.

г) задаём стартовую и конечную точку для лабиринта

start = 1;1

goal = размер лабиринта -2: размер лабиринта -2

д) выполняем обход в ширину учитывая стены

если элемент матрицы не равен стене то мы можем посетить его

е) совершаем обход матрицы и заполняем её найденным путём

ё) из полученной матрицы формируем строку и сохраняем её в файл

ж) отправляем результат пользователю

конец функции read

# Описание программы

Программа состоит из нескольких модулей (многомодульная): run.py, где производиться запуск, keyborads.py – файл где прописываются все кнопки, handlers.py – основной файл со всеми функциями, models.py – файл в котором задаётся база данных, requests.py – файл где прописаны функции для обращения к базе данных, config.py – файл где находится токен бота .

Разработанная программа состоит из нескольких функций модуля, которые реализовывались в следующем порядке:

1. сmd\_start()– начальное меню.
2. get\_maze\_size – генерация лабиринта.
3. play()– функция для прохождения лабиринта вручную.
4. read() – функция для считывания лабиринта из файла и его решения.

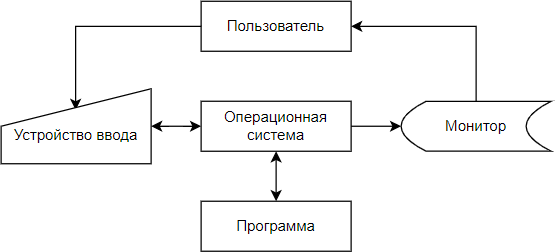
По порядку реализации модулей можно понять, что использовался нисходящей подход реализации программы. Этот метод основан на том, что 5 функций сначала проектируются основные компоненты программы, а затем уже дорабатываются ее мелкие детали. Это позволяет упростить ее тестирование, так как каждая функция (модуль) тестируется сразу же после его реализации, что помогает в создании более структурированных программ.

### При запуске программы происходит запуск бота после чего пользователь может взаимодействовать с ним используя callback кнопки.

#### Таблица 1 – Главная функция

|  |  |  |
| --- | --- | --- |
| Клавиши,  вызывающие событие | Действие  пользователя | Действие программы |
| /start | Выбран пункт  «Запуск бота» | Запускается бот и выводится стартовое меню |
| Создать лабиринт | Выбран пункт  «Создать лабиринт» | Рандомно генерируется лабиринт и отправляется |
| Считать лабиринт | Выбран пункт  «Считать лабиринт из файла» | Запрашивается файл и считывается |
| Создать решение | Выбран пункт  «Создать решение» | Создаётся решение и отправляется решённый лабиринт |
| сыграть | Выбран пункт  «сыграть» | Выводится лабиринт и кнопки для его прохождения в ручную |

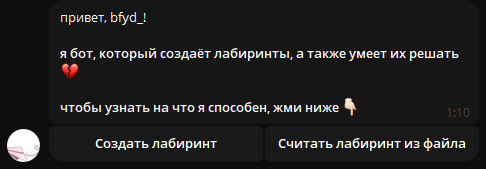
### Далее на рисунке 1 представлена схема данных.



***Рисунок 1 - Схема данных***

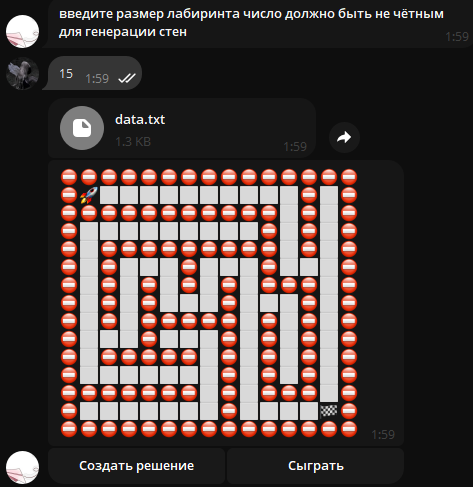


При запуске программы открывается консольное окно с меню:



**Рисунок 2 – Главное меню**

После выбора способа создания лабиринта, выведется запрос на размер лабиринта и после ввода кол-ва выведется лабиринт (в случае копирования) сразу выведется лабиринт:



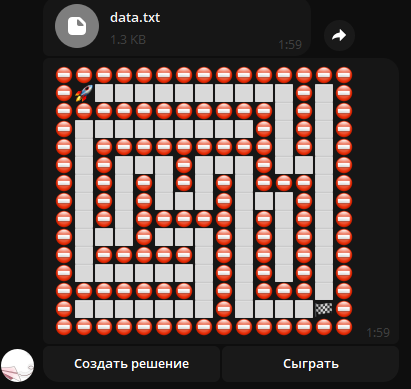
**Рисунок 3 – Создание графа**

# Отладка и тестирование

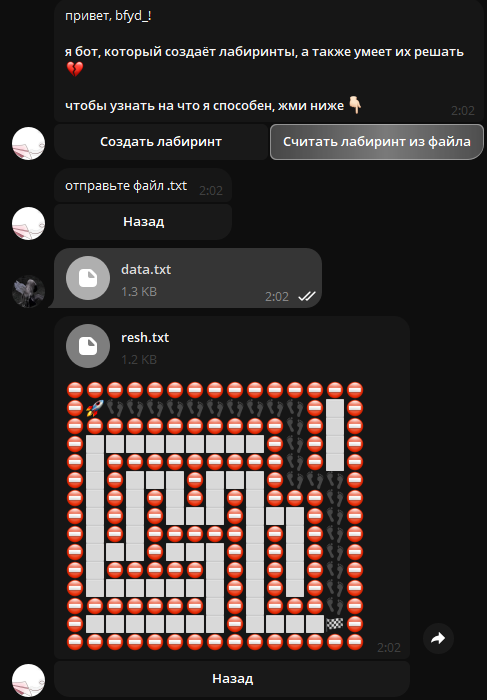
В качестве среды разработки была выбрана программа Pycharm. Программа обладает всеми средствами необходимыми при разработке и отладке программы. Для отладки использовались несколько возможностей PyCharm: точка останова, трассировка, анализ содержимого переменных.

Тестирование проводилось во время разработки и также после завершения разработки. В ходе нее было выявлено огромное количество проблем, связанных с работой с файлами, работой с памятью, работой с базой данныз, с размерами самих массивов и т. д.

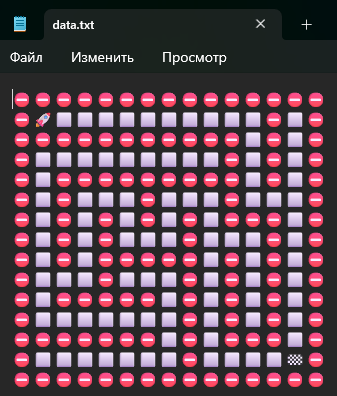
Тестирование программы:



**Рисунок 4 – создание лабиринта случайным образом**



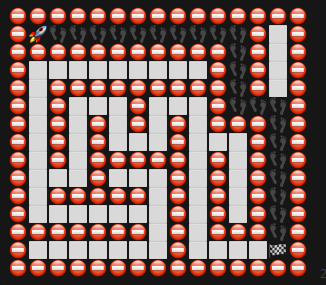
**Рисунок 5 – создание лабиринта из файла**



**Рисунок 6 – копирование графа из файла**



**Рисунок 7 – сохранение графа в файл**



**Рисунок 8 – алгоритм поиска путей в лабиринте**

***Таблица 1 – Описание поведения программы при тестировании***

|  |  |  |
| --- | --- | --- |
| Описание теста | Ожидаемый результат | Полученный результат |
| Ввод кол-ва вершин | Правильное считывание размера, генерация графа нужного размера и вывод графа | Верно |
| Алгоритм поиска путей в лабиринте | Правильное нахождение кратчайших расстояний. | Верно |
| Сохранение в файл/чтение из файла | Граф верно считывается из файла и верно записывается результат. | Верно |

# Ручной расчет

**Шаг 1:**

\* Извлекаем узел A из очереди. queue = []

\* Соседи узла A - B и C (по матрице).

\* B и C не посещены. Добавляем их в очередь и в visited:

\* queue = [B, C]

\* visited = {A, B, C}

**3. Шаг 2:**

\* Извлекаем узел B из очереди. queue = [C]

\* Соседи узла B - A и D.

\* A уже посещен. D не посещен. Добавляем D в очередь и visited:

\* queue = [C, D]

\* visited = {A, B, C, D}

**4. Шаг 3:**

**\*** Извлекаем узел C из очереди. queue = [D]

\* Соседи узла C - A, D, и E.

\* A и D уже посещены. E не посещен. Добавляем E в очередь и visited:

\* queue = [D, E]

\* visited = {A, B, C, D, E}

**5. Шаг 4:**

**\*** Извлекаем узел D из очереди. queue = [E]

\* Соседи узла D - B и C.

\* B и C уже посещены.

\* queue = [E]

\* visited = {A, B, C, D, E}

**6. Шаг 5:**

**\*** Извлекаем узел E из очереди. queue = []

\* Соседи узла E - C.

\* C уже посещен.

\* queue = []

\* visited = {A, B, C, D, E}

1. Цикл завершается, т.к очередь пуста. Обход закончен.

**Результат:**

**•** Порядок посещения узлов: A, B, C, D, E.

• Множество посещенных узлов: visited = {A, B, C, D, E} (все узлы посещены, если граф связный).

# 

**Рисунок 11 – ручной расчет**

# 

**Рисунок 12 – расчет программы**

# Заключение

При выполнении данной курсовой работы были получены навыки разработки многомодульных программ. Была освоена работа с файлами, изучены функции работы c асинхронными функциями, базами данных и оперативной памятью. Также были получены основные навыки отладки и тестирования программ и программирования в среде PyCharm на языке python.

В рамках курсовой работы была написана программа для реализации алгоритма поиска путей в лабиринте. Она имеет все необходимые функции для работы с лабиринтом.

Важной особенностью программы является реализация текстового пользовательского интерфейса, что обеспечивает удобство использования и повышает интуитивность взаимодействия пользователя с программой.

Код программы структурирован, что улучшает его читаемость и возможность доработки.

# В дальнейшем программу можно улучшить, путём добавления улучшенного графического интерфейса, что позволит улучшить навигацию по меню программы, путём добавления web приложения для бота.

# Список литературы

1. Лекция, обход графов в ширину, Митрохин М.А.
2. Документация Python
3. Документация Aiogram

# Приложение А. Листинги программы

**handlers.py**

from aiogram import F, Router  
from aiogram.filters import CommandStart  
from aiogram.types import Message,CallbackQuery  
from aiogram.fsm.state import StatesGroup, State  
from aiogram.fsm.context import FSMContext  
from aiogram import Bot  
from pyexpat.errors import messages  
  
from config import TOKEN  
from aiogram.client.bot import DefaultBotProperties  
from aiogram.enums import ParseMode  
  
from aiogram.types.input\_file import FSInputFile  
  
from collections import deque  
#.venv\Scripts\activate  
import asyncio  
import app.keyboards as kb  
import config  
import random  
import os  
import regex  
import database.requests as rq  
  
  
  
  
  
  
router = Router()  
bot= Bot(token= TOKEN,default=DefaultBotProperties(parse\_mode=ParseMode.HTML))  
  
  
global link  
class adm(StatesGroup):  
 admin\_id = State()  
 del\_adm = State()  
 gen\_maze = State()  
 get\_file = State()  
  
@router.message(CommandStart())  
async def cmd\_start(message: Message):  
 if(len(message.text)>=7):  
 await rq.new\_user(message.text[7:])  
 await rq.set\_user(message.from\_user.id, message.text[7:], message.from\_user.is\_premium)  
 else:  
 await rq.set\_user(message.from\_user.id,None,message.from\_user.is\_premium)  
 await message.answer(f'привет, '+message.from\_user.first\_name+'!\n\n<b>я бот, который создаёт лабиринты, а также умеет их решать 💔\n\nчтобы узнать на что я способен, жми ниже👇🏻</b>',  
 reply\_markup=kb.main)  
  
@router.callback\_query(F.data == 'main')  
async def main(callback:CallbackQuery,state: FSMContext):  
 await callback.message.answer(  
 f'привет, ' + callback.from\_user.first\_name + '!\n\n<b>я бот, который создаёт лабиринты, а также умеет их решать 💔\n\nчтобы узнать на что я способен, жми ниже👇🏻</b>',  
 reply\_markup=kb.main)  
  
@router.callback\_query(F.data == 'create\_maze')  
async def create\_maze(callback:CallbackQuery,state:FSMContext):  
 await callback.message.answer('<b>введите размер лабиринта число должно быть не чётным для генерации стен</b>')  
 await state.set\_state(adm.gen\_maze)  
  
@router.message(adm.gen\_maze)  
async def get\_maze\_size(message: Message, state:FSMContext):  
 text = ''  
 try:  
 a = 1  
 a += int(message.text)  
 a-=1  
 x = 1/(a%2)  
 maze = [['⛔️' for \_ in range(a)] for \_ in range(a)]  
 # Начальная точка  
 start\_x, start\_y = 1, 1  
 maze[start\_y][start\_x] = '⬜️'  
  
 # Стек для хранения пути  
 stack = [(start\_x, start\_y)]  
  
 while stack:  
 x, y = stack[-1]  
  
 # Собираем доступные направления  
 directions = []  
 if x > 1 and maze[y][x - 2] == '⛔️':  
 directions.append((-2, 0)) # Влево  
 if x < a - 2 and maze[y][x + 2] == '⛔️':  
 directions.append((2, 0)) # Вправо  
 if y > 1 and maze[y - 2][x] == '⛔️':  
 directions.append((0, -2)) # Вверх  
 if y < a - 2 and maze[y + 2][x] == '⛔️':  
 directions.append((0, 2)) # Вниз  
  
 if directions:  
 dx, dy = random.choice(directions)  
 maze[y + dy // 2][x + dx // 2] = '⬜️'  
 maze[y + dy][x + dx] = '⬜️'  
 stack.append((x + dx, y + dy))  
 else:  
 stack.pop()  
 maze[1][1] = '🚀'  
 maze[a-2][a-2] = '🏁'  
 for row in maze:  
 text+=''.join(row)  
 text+='\n'  
 if (a <= 15):  
 with open('data.txt', 'w', encoding='utf-8') as f:  
 f.write(text)  
 doc = FSInputFile('data.txt')  
 # Отправляем файл пользователю  
 with (open('data.txt', 'rb') as f):  
 await message.answer\_document(doc)  
 await message.answer(f'<code>{text}</code>',reply\_markup = await kb.second\_part(message.from\_user.id))  
 await rq.set\_maze(message.from\_user.id,maze)  
 else:  
 with open('data.txt', 'w', encoding='utf-8') as f:  
 f.write(text)  
 doc = FSInputFile('data.txt')  
 # Отправляем файл пользователю  
 with (open('data.txt', 'rb') as f):  
 await message.answer\_document(doc,reply\_markup = await kb.second\_part(message.from\_user.id))  
  
 except Exception as e:  
 print(e)  
 await message.answer('ошибка,введите число заново:')  
 await state.set\_state(adm.gen\_maze)  
  
@router.callback\_query(lambda callback: callback.data.startswith('play:'))  
async def play(callback: CallbackQuery,state: FSMContext):  
 msg\_id = callback.data.split(':')[1]  
 loaded\_data = await rq.get\_maze(callback.from\_user.id)  
 loaded\_data = loaded\_data.split('\n')  
 a = len(loaded\_data)  
 maze = [['' for \_ in range(a)] for \_ in range(a)]  
 a = len(maze[0])  
 i=0  
 j = 0  
 for i in range(len(loaded\_data)):  
 line = loaded\_data[i].split()  
  
 for j in range(len(line)):  
 maze[i][j] = line[j]  
  
  
  
 print(maze)  
 if(a<=15):  
 text = ''  
 for row in maze:  
 text += ''.join(row)  
 text += '\n'  
 await callback.message.answer(f'<code>{text}</code>',reply\_markup = kb.game)  
 else:  
 await callback.message.answer('слишком большой лабиринт для игры',reply\_markup = kb.back)  
  
@router.callback\_query(F.data == 'read\_maze')  
async def read(callback: CallbackQuery,state: FSMContext):  
 await callback.message.answer('отправьте файл .txt',reply\_markup = kb.back)  
 await state.set\_state(adm.get\_file)  
  
@router.message(adm.get\_file,F.document)  
async def read(message: Message, state:FSMContext):  
 try:  
 file\_id = message.document.file\_id  
 file\_info = await bot.get\_file(file\_id)  
 file\_path = file\_info.file\_path  
 file\_name = message.document.file\_name  
  
 download\_folder = "downloaded\_files"  
 os.makedirs(download\_folder, exist\_ok=True)  
 local\_file\_path = os.path.join(download\_folder, file\_name)  
  
 await bot.download\_file(file\_path, local\_file\_path)  
  
 with open(local\_file\_path, 'r', encoding='utf-8') as f:  
 for line in f:  
 a = len(line)  
 a //= 2  
  
 maze = [['' for \_ in range(a)] for \_ in range(a)]  
  
 with open(local\_file\_path, 'r', encoding='utf-8') as file:  
 row\_index = 0  
 for line in file:  
 line = line.strip()  
 if line and row\_index < len(maze):  
 # Разделяем строку на графемовые кластеры  
 symbols = regex.findall(r'\X', line)  
 for col\_index, char in enumerate(symbols):  
 if col\_index < len(maze[row\_index]):  
 maze[row\_index][col\_index] = char  
 row\_index += 1  
  
 start = (1, 1) # Начальная точка (y, x)  
 width, height = len(maze[0]), len(maze)  
 goal = (a - 2, a - 2) # Конечная точка  
 queue = deque([start]) # Очередь для BFS  
 visited = {start} # Множество посещённых узлов  
 parent = {start: None} # Словарь для отслеживания пути  
  
 directions = [(0, 1), (1, 0), (0, -1), (-1, 0)] # Направления (вперед, вправо, назад, влево)  
  
 while queue:  
 current = queue.popleft()  
  
 if current == goal:  
 break # Найдено решение  
  
 for d in directions:  
 neighbor = (current[0] + d[0], current[1] + d[1])  
 if (0 <= neighbor[0] < height and  
 0 <= neighbor[1] < width and  
 (maze[neighbor[0]][neighbor[1]] == '⬜️' or maze[neighbor[0]][neighbor[1]] == '🚀' or  
 maze[neighbor[0]][neighbor[1]] == '🏁') and  
 neighbor not in visited):  
 visited.add(neighbor)  
 parent[neighbor] = current  
 queue.append(neighbor)  
  
 # Восстановление пути  
 path = []  
 step = goal  
 print(maze)  
 while step is not None:  
 path.append(step)  
 step = parent[step]  
  
 path.reverse() # Путь от старта к финишу  
 for y, x in path:  
 if (not (y == 1 and x == 1) and not (y == a - 2 and x == a - 2)):  
 maze[y][x] = '👣'  
  
 text = ''  
  
 for row in maze:  
 text += ''.join(row)  
 text += '\n'  
  
 if (a <= 15):  
 with open('resh.txt', 'w', encoding='utf-8') as f:  
 f.write(text)  
 doc = FSInputFile('resh.txt')  
 # Отправляем файл пользователю  
 with (open('resh.txt', 'rb') as f):  
 await message.answer\_document(doc, caption=f'<code>{text}</code>', reply\_markup=kb.back)  
  
 # Удаляем файл после отправки (по желанию)  
 os.remove('resh.txt')  
 else:  
 with open('resh.txt', 'w', encoding='utf-8') as f:  
 f.write(text)  
 doc = FSInputFile('resh.txt')  
 # Отправляем файл пользователю  
 with (open('resh.txt', 'rb') as f):  
 await message.answer\_document(doc, reply\_markup=kb.back)  
  
 # Удаляем файл после отправки (по желанию)  
 os.remove('resh.txt')  
  
 except Exception as e:  
 await message.answer('ошибка попробуйте ещё раз')  
  
  
  
@router.callback\_query(F.data == 'up')  
async def up(callback: CallbackQuery,state: FSMContext):  
 await callback.message.delete()  
 loaded\_data = await rq.get\_maze(callback.from\_user.id)  
 loaded\_data = loaded\_data.split('\n')  
 a = len(loaded\_data)  
 maze = [['' for \_ in range(a)] for \_ in range(a)]  
 a = len(maze[0])  
 i = 0  
 j = 0  
 for i in range(len(loaded\_data)):  
 line = loaded\_data[i].split()  
  
 for j in range(len(line)):  
 maze[i][j] = line[j]  
 check = True  
 a = len(maze[0])  
 for i in range(a):  
 for j in range(a):  
 if(maze[i][j] == '🚀' and check):  
 if(maze[i-1][j] == '⛔️' and check):  
 check = False  
 await callback.message.answer('💥<b>YOU DIED</b>💥',reply\_markup = kb.back)  
 if (maze[i - 1][j] == '🏁' and check):  
 check = False  
 await callback.message.answer('💥<b>Ура вы финишировали</b>💥', reply\_markup=kb.back)  
 if (maze[i - 1][j] == '⬜️' and check):  
 check = False  
 msg = await rq.get\_game(callback.from\_user.id)  
 maze[i][j] = '👣'  
 maze[i-1][j]='🚀'  
 await rq.set\_maze(callback.from\_user.id, maze)  
 text = ''  
 for row in maze:  
 text += ''.join(row)  
 text += '\n'  
 await callback.message.answer(text = text,reply\_markup = kb.game)  
 break  
  
@router.callback\_query(F.data == 'down')  
async def up(callback: CallbackQuery,state: FSMContext):  
 await callback.message.delete()  
 loaded\_data = await rq.get\_maze(callback.from\_user.id)  
 loaded\_data = loaded\_data.split('\n')  
 a = len(loaded\_data)  
 maze = [['' for \_ in range(a)] for \_ in range(a)]  
 a = len(maze[0])  
 i = 0  
 j = 0  
 for i in range(len(loaded\_data)):  
 line = loaded\_data[i].split()  
  
 for j in range(len(line)):  
 maze[i][j] = line[j]  
  
 a = len(maze[0])  
 check = True  
 for i in range(a):  
 for j in range(a):  
 if(maze[i][j] == '🚀' and check):  
 if(maze[i+1][j] == '⛔️' and check):  
 check = False  
 print(maze[i-1][j])  
 await callback.message.answer('💥<b>YOU DIED</b>💥',reply\_markup = kb.back)  
 if (maze[i + 1][j] == '🏁' and check):  
 check = False  
 await callback.message.answer('💥<b>Ура вы финишировали</b>💥', reply\_markup=kb.back)  
 if (maze[i + 1][j] == '⬜️' and check):  
 check = False  
 msg = await rq.get\_game(callback.from\_user.id)  
 maze[i][j] = '👣'  
 maze[i+1][j]='🚀'  
 await rq.set\_maze(callback.from\_user.id, maze)  
 text = ''  
 for row in maze:  
 text += ''.join(row)  
 text += '\n'  
 await callback.message.answer(text = text,reply\_markup = kb.game)  
  
@router.callback\_query(F.data == 'right')  
async def up(callback: CallbackQuery,state: FSMContext):  
 await callback.message.delete()  
 loaded\_data = await rq.get\_maze(callback.from\_user.id)  
 loaded\_data = loaded\_data.split('\n')  
 a = len(loaded\_data)  
 maze = [['' for \_ in range(a)] for \_ in range(a)]  
 a = len(maze[0])  
 i = 0  
 j = 0  
 for i in range(len(loaded\_data)):  
 line = loaded\_data[i].split()  
  
 for j in range(len(line)):  
 maze[i][j] = line[j]  
  
 a = len(maze[0])  
 check = True  
 for i in range(a):  
 for j in range(a):  
 if(maze[i][j] == '🚀' and check):  
 if(maze[i][j+1] == '⛔️' and check):  
 check = False  
 await callback.message.answer('💥<b>YOU DIED</b>💥',reply\_markup = kb.back)  
 if (maze[i][j + 1] == '🏁' and check):  
 check = False  
 await callback.message.answer('💥<b>Ура вы финишировали</b>💥', reply\_markup=kb.back)  
 if (maze[i][j + 1] == '⬜️' and check):  
 check = False  
 msg = await rq.get\_game(callback.from\_user.id)  
 maze[i][j] = '👣'  
 maze[i][j+1]='🚀'  
 await rq.set\_maze(callback.from\_user.id, maze)  
 text = ''  
 for row in maze:  
 text += ''.join(row)  
 text += '\n'  
 await callback.message.answer(text = text,reply\_markup = kb.game)  
 break  
  
@router.callback\_query(F.data == 'left')  
async def up(callback: CallbackQuery,state: FSMContext):  
 await callback.message.delete()  
 loaded\_data = await rq.get\_maze(callback.from\_user.id)  
 loaded\_data = loaded\_data.split('\n')  
 a = len(loaded\_data)  
 maze = [['' for \_ in range(a)] for \_ in range(a)]  
 a = len(maze[0])  
 i = 0  
 j = 0  
 for i in range(len(loaded\_data)):  
 line = loaded\_data[i].split()  
  
 for j in range(len(line)):  
 maze[i][j] = line[j]  
  
 a = len(maze[0])  
 check = True  
 for i in range(a):  
 for j in range(a):  
 if(maze[i][j] == '🚀' and check):  
 if(maze[i][j-1] == '⛔️' and check):  
 check = False  
 await callback.message.answer('💥<b>YOU DIED</b>💥',reply\_markup = kb.back)  
 if (maze[i][j - 1] == '🏁' and check):  
 check = False  
 await callback.message.answer('💥<b>Ура вы финишировали</b>💥', reply\_markup=kb.back)  
 if (maze[i][j - 1] == '⬜️' and check):  
 check = False  
 msg = await rq.get\_game(callback.from\_user.id)  
 maze[i][j] = '👣'  
 maze[i][j-1]='🚀'  
 await rq.set\_maze(callback.from\_user.id, maze)  
 text = ''  
 for row in maze:  
 text += ''.join(row)  
 text += '\n'  
 await callback.message.answer(text = text,reply\_markup = kb.game)  
 break  
  
@router.callback\_query(F.data == 'create\_resh')  
async def create\_resh(callback: CallbackQuery,state: FSMContext):  
  
 try:  
  
 with open('data.txt', 'r', encoding='utf-8') as f:  
 for line in f:  
 a = len(line)  
 a//=2  
  
  
 maze = [['' for \_ in range(a)] for \_ in range(a)]  
  
  
 with open('data.txt', 'r', encoding='utf-8') as file:  
 row\_index = 0  
 for line in file:  
 line = line.strip()  
 if line and row\_index < len(maze):  
 # Разделяем строку на графемовые кластеры  
 symbols = regex.findall(r'\X', line)  
 for col\_index, char in enumerate(symbols):  
 if col\_index < len(maze[row\_index]):  
 maze[row\_index][col\_index] = char  
 row\_index += 1  
  
  
 start = (1, 1) # Начальная точка (y, x)  
 width, height = len(maze[0]), len(maze)  
 goal = (a - 2, a - 2) # Конечная точка  
 queue = deque([start]) # Очередь для BFS  
 visited = {start} # Множество посещённых узлов  
 parent = {start: None} # Словарь для отслеживания пути  
  
 directions = [(0, 1), (1, 0), (0, -1), (-1, 0)] # Направления (вперед, вправо, назад, влево)  
  
 while queue:  
 current = queue.popleft()  
  
 if current == goal:  
 break # Найдено решение  
  
 for d in directions:  
 neighbor = (current[0] + d[0], current[1] + d[1])  
 if (0 <= neighbor[0] < height and  
 0 <= neighbor[1] < width and  
 (maze[neighbor[0]][neighbor[1]] == '⬜️' or maze[neighbor[0]][neighbor[1]] == '🚀' or maze[neighbor[0]][neighbor[1]] == '🏁' )and  
 neighbor not in visited):  
 visited.add(neighbor)  
 parent[neighbor] = current  
 queue.append(neighbor)  
  
 # Восстановление пути  
 path = []  
 step = goal  
 print(maze)  
 while step is not None:  
 path.append(step)  
 step = parent[step]  
  
 path.reverse() # Путь от старта к финишу  
 for y, x in path:  
 if(not(y == 1 and x == 1) and not(y == a-2 and x == a-2)):  
 maze[y][x] = '👣'  
  
 text = ''  
  
 for row in maze:  
 text += ''.join(row)  
 text += '\n'  
  
 if (a <= 15):  
 with open('resh.txt', 'w', encoding='utf-8') as f:  
 f.write(text)  
 doc = FSInputFile('resh.txt')  
 # Отправляем файл пользователю  
 with (open('resh.txt', 'rb') as f):  
 await callback.message.answer\_document(doc,caption = f'<code>{text}</code>',reply\_markup = kb.back)  
  
 # Удаляем файл после отправки (по желанию)  
 os.remove('resh.txt')  
 else:  
 with open('resh.txt', 'w', encoding='utf-8') as f:  
 f.write(text)  
 doc = FSInputFile('resh.txt')  
 # Отправляем файл пользователю  
 with (open('resh.txt', 'rb') as f):  
 await callback.message.answer\_document(doc,reply\_markup = kb.back)  
  
 # Удаляем файл после отправки (по желанию)  
 os.remove('resh.txt')  
  
 except Exception as e:  
 pass

**keyboards.py**

from aiogram.types import (ReplyKeyboardMarkup, KeyboardButton,  
 InlineKeyboardMarkup,InlineKeyboardButton)  
from aiogram.utils.keyboard import ReplyKeyboardBuilder,InlineKeyboardBuilder  
  
  
main = InlineKeyboardMarkup(row\_width=2,inline\_keyboard=[  
 [InlineKeyboardButton(text='Cоздать лабиринт',callback\_data='create\_maze',resize\_keyboard=True),InlineKeyboardButton(text='Считать лабиринт из файла',callback\_data='read\_maze',resize\_keyboard=True)]  
 ])  
async def second\_part(msg\_id):  
 return InlineKeyboardMarkup(row\_width=2,inline\_keyboard=[  
 [InlineKeyboardButton(text='Cоздать решение',callback\_data='create\_resh',resize\_keyboard=True),InlineKeyboardButton(text='Сыграть',callback\_data=f'play:{msg\_id}',resize\_keyboard=True)]  
 ])  
  
back = InlineKeyboardMarkup(row\_width=2,inline\_keyboard=[  
 [InlineKeyboardButton(text='Назад',callback\_data='main',resize\_keyboard=True)]  
 ])  
  
  
game = InlineKeyboardMarkup(row\_width=2,inline\_keyboard=[  
 [InlineKeyboardButton(text='Вверх',callback\_data=f'up',resize\_keyboard=True),InlineKeyboardButton(text='Вниз',callback\_data=f'down',resize\_keyboard=True)],  
 [InlineKeyboardButton(text='Влево',callback\_data=f'left',resize\_keyboard=True),InlineKeyboardButton(text='Вправо',callback\_data=f'right',resize\_keyboard=True)],  
 ])

**run.py**

import asyncio  
import logging  
from aiogram.client.bot import DefaultBotProperties  
from aiogram.enums import ParseMode  
from aiogram import Bot, Dispatcher  
  
from config import TOKEN  
from app.handlers import router  
from database.models import async\_main  
  
bot= Bot(token= TOKEN,default=DefaultBotProperties(parse\_mode=ParseMode.HTML))  
dp = Dispatcher()  
  
  
async def main():  
 await async\_main()  
 dp.include\_router(router)  
 await dp.start\_polling(bot)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 logging.basicConfig(level=logging.INFO)  
 try:  
 asyncio.run(main())  
 except KeyboardInterrupt:  
 print('Exit')

models.py

from sqlalchemy import BigInteger, String, ForeignKey,Boolean, Integer,Text  
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped\_column  
from sqlalchemy.ext.asyncio import AsyncAttrs, async\_sessionmaker, create\_async\_engine  
  
engine = create\_async\_engine(url='sqlite+aiosqlite:///db.sqlite3')  
  
async\_session = async\_sessionmaker(engine)  
  
  
class Base(AsyncAttrs, DeclarativeBase):  
 pass  
  
class del\_last\_mes(Base):  
 \_\_tablename\_\_ = 'del\_last\_mess'  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 chat\_id = mapped\_column(BigInteger)  
 mes\_id = mapped\_column(BigInteger)  
class User(Base):  
 \_\_tablename\_\_ = 'users'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 tg\_id= mapped\_column(BigInteger)  
 admin = mapped\_column(Boolean)  
 link = mapped\_column(String)  
 premium = mapped\_column(Boolean)  
 mes\_id = mapped\_column(BigInteger)  
 mez = mapped\_column(Text)  
  
class Link(Base):  
 \_\_tablename\_\_ = 'links'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 name = mapped\_column(String)  
 new\_members = mapped\_column(Integer)  
  
async def async\_main():  
 async with engine.begin() as conn:  
 await conn.run\_sync(Base.metadata.create\_all)

requests.py

from database.models import async\_session  
from database.models import User,Link,del\_last\_mes  
from sqlalchemy import select,update, delete,desc,func  
  
async def set\_user(tg\_id,link,premium):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
  
 if not user:  
 session.add(User(tg\_id=tg\_id,link = link,premium = premium))  
 await session.commit()  
  
async def set\_maze(tg\_id,maze):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
  
 if user:  
 matrix\_rows = '\n'.join([' '.join(row) for row in maze])  
 user.mez = matrix\_rows  
 await session.commit()  
  
async def get\_maze(tg\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
  
 if user:  
 loaded\_data = user.mez  
 return loaded\_data  
async def add\_game(tg\_id,mes\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
 if user:  
 user.mes\_id = mes\_id  
 await session.commit()  
async def del\_game(tg\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
 if user:  
 user.mes\_id = None  
 await session.commit()  
async def get\_game(tg\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
 if user:  
 return user

**config.py**

TOKEN = '7191663816:AAGsan0vOpUzIjs7MpfKM63-R9xr9cwsPQM'

**models.py**

from sqlalchemy import BigInteger, String, ForeignKey,Boolean, Integer,Text  
from sqlalchemy.orm import DeclarativeBase, Mapped, mapped\_column  
from sqlalchemy.ext.asyncio import AsyncAttrs, async\_sessionmaker, create\_async\_engine  
  
engine = create\_async\_engine(url='sqlite+aiosqlite:///db.sqlite3')  
  
async\_session = async\_sessionmaker(engine)  
  
  
class Base(AsyncAttrs, DeclarativeBase):  
 pass  
  
class del\_last\_mes(Base):  
 \_\_tablename\_\_ = 'del\_last\_mess'  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 chat\_id = mapped\_column(BigInteger)  
 mes\_id = mapped\_column(BigInteger)  
class User(Base):  
 \_\_tablename\_\_ = 'users'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 tg\_id= mapped\_column(BigInteger)  
 admin = mapped\_column(Boolean)  
 link = mapped\_column(String)  
 premium = mapped\_column(Boolean)  
 mes\_id = mapped\_column(BigInteger)  
 mez = mapped\_column(Text)  
  
class Link(Base):  
 \_\_tablename\_\_ = 'links'  
  
 id: Mapped[int] = mapped\_column(primary\_key=True)  
 name = mapped\_column(String)  
 new\_members = mapped\_column(Integer)  
  
async def async\_main():  
 async with engine.begin() as conn:  
 await conn.run\_sync(Base.metadata.create\_all)

**requests.py**

from database.models import async\_session  
from database.models import User,Link,del\_last\_mes  
from sqlalchemy import select,update, delete,desc,func  
  
async def set\_user(tg\_id,link,premium):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
  
 if not user:  
 session.add(User(tg\_id=tg\_id,link = link,premium = premium))  
 await session.commit()  
  
async def set\_maze(tg\_id,maze):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
  
 if user:  
 matrix\_rows = '\n'.join([' '.join(row) for row in maze])  
 user.mez = matrix\_rows  
 await session.commit()  
  
async def get\_maze(tg\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
  
 if user:  
 loaded\_data = user.mez  
 return loaded\_data  
async def add\_game(tg\_id,mes\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
 if user:  
 user.mes\_id = mes\_id  
 await session.commit()  
async def del\_game(tg\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
 if user:  
 user.mes\_id = None  
 await session.commit()  
async def get\_game(tg\_id):  
 async with async\_session() as session:  
 user = await session.scalar(select(User).where(User.tg\_id == tg\_id))  
 if user:  
 return user