**Министерство науки и высшего образования Российской Федерации**

**Федеральное государственное бюджетное образовательное учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**

**Факультет «Информатика и системы управления»**

# **Курс «Базовые компоненты интернет-технологий»**

## Отчёт по лабораторной работе №6

## «Разработка бота на основе конечного автомата для Telegram с использованием языка Python.»

Выполнил:

студент группы ИУ5-33Б Кузнецов В. А.

подпись: , дата:

Проверил:

лектор Гапанюк Ю. Е.

подпись: , дата:

2022 г.

**Задание:**

1. Разработайте бота для Telegram. Бот должен реализовывать конечный автомат из трех состояний.

**Текст программы:**

dbworker.py

**from** enum **import** Enum  
**from** vedis **import** Vedis  
  
  
*# Файл базы данных Vedis*db\_file = **"db.vdb"***# Ключ записи в БД для текущего состояния*CURRENT\_STATE = **"CURRENT\_STATE"  
  
  
class** States(Enum):  
 *"""Состояния автомата"""* STATE\_COEFFICIENT\_A = 0  
 STATE\_COEFFICIENT\_B = 1  
 STATE\_COEFFICIENT\_C = 2  
  
 **def** \_\_next\_\_(self):  
 **try**:  
 **return** States(self.value + 1)  
 **except** ValueError:  
 **raise** StopIteration  
  
  
**def** get(key):  
 *"""Чтение значения"""* **with** Vedis(db\_file) **as** db:  
 **try**:  
 **return** db[key].decode()  
 **except** KeyError:  
 *# в случае ошибки значение по умолчанию - начало диалога* **return** States(0)  
  
  
**def** set(key, value):  
 *"""Запись значения"""* **with** Vedis(db\_file) **as** db:  
 **try**:  
 db[key] = value  
 **return True  
 except**:  
 **return False  
  
  
def** make\_key(chatid, keyid):  
 *"""Создание ключа для записи и чтения"""* res = str(chatid) + **'\_\_'** + str(keyid)  
 **return** res

equation\_solving.py

**import** math  
  
  
**def** get\_roots(a, b, c):  
 result = []  
  
 *# Частный случай квадратного уравнения* **if** a == 0.0:  
 **if** b != 0.0:  
 root = - c / b  
 **if** root > 0:  
 result.extend((math.sqrt(root), -math.sqrt(root)))  
 **if** c == 0:  
 result.append(0)  
 **return** result  
  
 *# Общий случай решения биквадратного уравнения* D = b \* b - 4 \* a \* c  
  
 **if** c == 0:  
 result.append(0)  
  
 **if** D == 0.0:  
 root = -b / (2.0 \* a)  
 **if** root > 0:  
 result.extend((math.sqrt(root), -math.sqrt(root)))  
 **elif** D > 0.0:  
 sqD = math.sqrt(D)  
 root1 = (-b + sqD) / (2.0 \* a)  
 root2 = (-b - sqD) / (2.0 \* a)  
 **if** root1 > 0:  
 result.extend((math.sqrt(root1), -math.sqrt(root1)))  
 **if** root2 > 0:  
 result.extend((math.sqrt(root2), -math.sqrt(root2)))  
 **return** result  
  
  
**def** calc\_result(a, b, c):  
 *# Составить биквадратное уравнение* result = **'Биквадратное уравнение: '  
 if** a:  
 result += str(a).rstrip(**'0'**).rstrip(**'.'**) + **'x⁴'  
 if** b:  
 **if** b > 0 **and** a:  
 result += **'+'** result += str(b).rstrip(**'0'**).rstrip(**'.'**) + **'x²'  
 if** c:  
 **if** c > 0 **and** (a **or** b):  
 result += **'+'** result += str(c).rstrip(**'0'**).rstrip(**'.'**)  
 **elif not** a **and not** b:  
 result += **'0'** result += **'=0\n'** *# Вычисление корней* roots = get\_roots(a, b, c)

*# Дописать решение* **match** len(roots):  
 **case** 0:  
 result += **'Не имеет корней'  
 case** 1:  
 result += **'Имеет один корень:\n{}'**.format(\*roots)  
 **case** 2:  
 result += **'Имеет два корня:\n{}\n{}'**.format(\*roots)  
 **case** 3:  
 result += **'Имеет три корня:\n{}\n{}\n{}'**.format(\*roots)  
 **case** 4:  
 result += **'Имеет четыре корня:\n{}\n{}\n{}\n{}'**.format(\*roots)  
  
 **return** result

equation\_test.py

**import** unittest  
**from** equation\_solving **import** get\_roots  
  
  
**class** BiquadrateTest(unittest.TestCase):  
 **def** test\_no\_roots(self):  
 expected\_answer = set()  
 self.assertEqual(expected\_answer, set(get\_roots(0, 0, 0)))  
 self.assertEqual(expected\_answer, set(get\_roots(0, 1, 4)))  
 self.assertEqual(expected\_answer, set(get\_roots(0, -1, -4)))  
 self.assertEqual(expected\_answer, set(get\_roots(1, 0, 4)))  
 self.assertEqual(expected\_answer, set(get\_roots(-1, 0, -4)))  
  
 **def** test\_one\_root(self):  
 expected\_answer = {0}  
 self.assertEqual(expected\_answer, set(get\_roots(0, 1, 0)))  
 self.assertEqual(expected\_answer, set(get\_roots(1, 0, 0)))  
 self.assertEqual(expected\_answer, set(get\_roots(0, -1, 0)))  
 self.assertEqual(expected\_answer, set(get\_roots(-1, 0, 0)))  
  
 **def** test\_two\_roots(self):  
 expected\_answer = {-2, 2}  
 self.assertEqual(expected\_answer, set(get\_roots(0, 1, -4)))  
 self.assertEqual(expected\_answer, set(get\_roots(0, -1, 4)))  
 self.assertEqual(expected\_answer, set(get\_roots(1, 0, -16)))  
 self.assertEqual(expected\_answer, set(get\_roots(-1, 0, 16)))  
  
 **def** test\_three\_roots(self):  
 expected\_answer = {-2, 0, 2}  
 self.assertEqual(expected\_answer, set(get\_roots(1, -4, 0)))  
 self.assertEqual(expected\_answer, set(get\_roots(-1, 4, 0)))  
  
 **def** test\_four\_roots(self):  
 expected\_answer = {-3, -1, 1, 3}  
 self.assertEqual(expected\_answer, set(get\_roots(1, -10, 9)))  
 self.assertEqual(expected\_answer, set(get\_roots(-1, 10, -9)))  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 unittest.main()

main.py

**import** os  
**import** telebot  
**from** dotenv **import** load\_dotenv  
**import** dbworker  
**from** equation\_solving **import** calc\_result  
  
  
load\_dotenv()  
TOKEN = os.environ.get(**'TOKEN'**)  
  
  
*# Создание бота*bot = telebot.TeleBot(TOKEN)  
  
  
*# Начало диалога*@bot.message\_handler(commands=[**'start'**])  
**def** cmd\_start(message):  
 bot.send\_message(message.chat.id, **'Я решаю биквадратные уравнения!\nОни выглядят вот так: Ax⁴+Bx²+C=0'**)  
 current\_state = dbworker.States.STATE\_COEFFICIENT\_A  
 dbworker.set(dbworker.make\_key(message.chat.id, dbworker.CURRENT\_STATE), current\_state.value)  
 bot.send\_message(message.chat.id, **f'Введите коэффициент {**chr(current\_state.value + 65)**}'**)  
  
  
*# По команде /reset будем сбрасывать состояния, возвращаясь к началу диалога*@bot.message\_handler(commands=[**'reset'**])  
**def** cmd\_reset(message):  
 bot.send\_message(message.chat.id, **'Сбрасываем результаты предыдущего ввода.'**)  
 current\_state = dbworker.States.STATE\_COEFFICIENT\_A  
 dbworker.set(dbworker.make\_key(message.chat.id, dbworker.CURRENT\_STATE), current\_state.value)  
 bot.send\_message(message.chat.id, **f'Введите коэффициент {**chr(current\_state.value + 65)**}'**)  
  
  
*# Считывание коэффициента*@bot.message\_handler()  
**def** coefficient(message):  
 *# Получаем текущее состояние* current\_state = dbworker.States(int(dbworker.get(dbworker.make\_key(message.chat.id, dbworker.CURRENT\_STATE))))  
  
 text = message.text  
 **if not** text.lstrip(**"-"**).isdigit():  
 *# Состояние не изменяется, выводится сообщение об ошибке* bot.send\_message(message.chat.id, **'Пожалуйста введите число!'**)  
 **return** *# Сохраняем полученное значение  
 # bot.send\_message(message.chat.id, f'Вы ввели коэффициент {chr(current\_state.value + 65)}')* dbworker.set(dbworker.make\_key(message.chat.id, current\_state.value), text)  
  
 *# Если есть все три коэффициента, то решаем биквадратное уравнение* **try**:  
 current\_state = next(current\_state)  
 **except** StopIteration:  
 a = float(dbworker.get(dbworker.make\_key(message.chat.id, dbworker.States.STATE\_COEFFICIENT\_A.value)))  
 b = float(dbworker.get(dbworker.make\_key(message.chat.id, dbworker.States.STATE\_COEFFICIENT\_B.value)))  
 c = float(text)  
 bot.send\_message(message.chat.id, calc\_result(a, b, c))  
 current\_state = dbworker.States.STATE\_COEFFICIENT\_A  
  
 *# Переходим к следующему состоянию* dbworker.set(dbworker.make\_key(message.chat.id, dbworker.CURRENT\_STATE), current\_state.value)  
 bot.send\_message(message.chat.id, **f'Введите коэффициент {**chr(current\_state.value + 65)**}'**)  
  
  
**if** \_\_name\_\_ == **'\_\_main\_\_'**:  
 bot.infinity\_polling()

**Пример выполнения**







