Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение

высшего образования

«Ульяновский государственный технический университет»

Кафедра «Вычислительная техника»

Дисциплина «Информационная безопасность»

**Лабораторная работа №1**

**Тестирование псевдослучайных последовательностей**

Выполнил

Студент группы ИВТАСбд-41

Ведин В.А.

Проверил:

Старший преподаватель кафедры «ВТ»

Мартынов А.И.

Ульяновск

2025

# Задание

Реализовать приложение, позволяющее выполнять следующие действия:

1. Задавать длину генерируемой последовательности в битах
2. Генерировать псевдослучайную последовательность 0 и 1 с помощью стандартного алгоритма генерации случайных чисел
3. Загружать последовательность из текстового файла
4. Сохранять полученную последовательность в файл и выводить ее на экран приложения
5. Проверять полученную последовательность с помощью реализованных тестов. Результат проверки должен отображаться в приложении

# .

# Ход работы

# 1. Частотный тест

# Данный тест оценивает пропорцию нулей и единиц в проверяемой последовательности. Тест определяет, является ли количество нулей и единиц в последовательности приблизительно таким же, как должно быть в истинно случайно последовательности.

# 1. Входная последовательность, состоящая из 0 и 1 (будем обозначать ее ε), преобразовывается в последовательность -1 и 1 (будем обозначать ее X) соответственно:

# 2. Вычисляется сумма , где n – количество элементов проверяемой последовательности.

# 3. Вычисляется статистика

# 4. Если , то тест считается успешно пройденным, иначе делается вывод о том, что последовательность является неслучайной.

# Код, реализующий частотный тест представлен в листинге 1.

# Листинг 1. Частотный тест

|  |
| --- |
| def generate\_bit\_seq(seq\_len: Optional[int] = 10000,input\_file\_path: Optional[str] = "",output\_file\_path: Optional[str] = "",seed: int = 42) -> list[int]:"""Генерирует или загружает последовательность битов (0 и 1).Args:seq\_len: Длина генерируемой последовательности (по умолчанию 10000 бит)input\_file\_path: Путь к файлу для загрузки последовательностиoutput\_file\_path: Путь для сохранения последовательностиseed: Seed для генератора случайных чиселReturns:list[int]: последовательность битов (0 и 1).Note:Если указан input\_file\_path, последовательность загружается из файла.Иначе генерируется случайная последовательность заданной длины."""bit\_seq = Nonerandom.seed(seed)if input\_file\_path != "":with open(input\_file\_path, "r", encoding="utf-8") as f:bit\_seq = list(f.read())bit\_seq = [int(bit) for bit in bit\_seq]bit\_seq = [0 if random.random() < 0.5 else 1 for \_ in range(seq\_len)] if bit\_seq is None else bit\_seqif output\_file\_path != "":with open(output\_file\_path, "w", encoding="utf-8") as f:f.write("".join(map(str, bit\_seq)))return bit\_seq |

2. Тест на последовательность одинаковых бит.

Этот тест анализирует количество цепочек в проверяемой последовательности, где цепочка – это непрерывная последовательность одинаковых бит. Под цепочкой длиной k понимается цепочка, состоящая из ровно k бит и ограниченная до и после битами с противоположным значением. Тест определяет, является ли количество цепочек из нулей и единиц различной длины в последовательности приблизительно таким же, как должно быть в истинно случайной последовательности.

Шаги алгоритма:

1. Вычисляется частота, с которой в проверяемой последовательности встречаются единицы:

2. Вычисляется значение , где , если и иначе

3. Вычисляется статистика

4. Если , то тест считается успешно пройденным, иначе делается вывод о том, что последовательность является не случайной

Код реализующий данный тест представлен в листинге 2.

Листинг 2. Тест на последовательность одинаковых бит

|  |
| --- |
| def r(bit\_seq: list[int], k: int):  """  Вспомогательная функция для теста на последовательность одинаковых бит.  Определяет, является ли k-тый бит началом новой цепочки.  Args:  bit\_seq: Последовательность битов  k: Индекс текущего бита    Returns:  int: 1 если бит k и k+1 разные (начало новой цепочки) 0 если одинаковые  """  if bit\_seq[k] == bit\_seq[k + 1]:  return 0  return 1  def identical\_bit\_seq\_test(bit\_seq: list[int]) -> bool:  """  Тест на последовательность одинаковых бит (Rusn Test).  Анализирует количество цепочек (непрерывных последовательностей одинаковых бит)  в проверяемой последовательности  Args:  bit\_seq: Последовательность битов (0 и 1)    Returns:  bool: True если тест пройден, False в противном случае    Algorithm:  1. Вычисление частоты единиц (pi)  2. Вычисление количества переходов V\_n  3. Вычисление статистики S  4. Сравнение с критическим значением CONST  """  pi = 1 / (len(bit\_seq)) \* sum(bit\_seq)  V\_n = sum([r(bit\_seq, k) for k in range(len(bit\_seq) - 1)]) + 1  S = abs(V\_n - 2 \* len(bit\_seq) \* pi \* (1 - pi))  S /= 2 \* math.sqrt(2 \* len(bit\_seq) \* pi \* (1 - pi))  if S <= CONST:  return True  return False |

3. Расширенный тест на произвольные отклонения

Этот тест оценивает общее число посещений определенного состояния при произвольном обходе кумулятивной суммы. Цель этого теста – определить отклонения от ожидаемого числа посещений различных состояний при произвольном обходе. Фактически данный тест состоит из 18 тестов, по одному для каждого состояния: −9, −8, … − 1, 1, 2, … , 9.

Шаги алгоритма:

1. Входная последовательность, состоящая из 0 и 1 (будем обозначать ее ), преобразовывается в последовательность -1 и 1 (будем обозначать ее X) соответственно:

2. Вычисляются суммы последовательно удлиняющихся подпоследовательностей, начинающихся с

...

3. Формируется новая последовательность

4. Вычисляется где k – количество нулей в полученной последовательности

5. Для каждого из 18 состояний вычисляется которое показывает, сколько раз состояние j встречалось в последовательности . Здесь j = -9, -8, ..., -1, 1, 2, ..., 9.

6. Вычисляются 18 статистик для каждого состояния j = -9, -8, ..., -1, 1, 2, ..., 9.

7. Если все статистики то тест считается успешно пройденным, если же хотя бы для одной статистики это условие не выполнилось, то делается вывод о том, что последовательность является неслучайной.

Код данного теста представлен в листинге 3.

Листинг 3. Расширенный тест на произвольные отклонения

|  |
| --- |
| def extended\_random\_deviation\_test(bit\_seq: list[int]) -> bool:  """  Расширенный тест на произвольные отклонения.  Оценивает общее число посещений определенного состояния при произвольном обходе  кумулятивный суммы. Состоит из 18 подтестов для состояний из -9 до 9 (кроме 0).  Args:  bit\_seq: Последовательность битов (0 и 1)    Returns:  bool: True если все 18 тестов пройдены, False если хотя бы один не пройден    Algorithm:  1. Преобразование битов в -1 и 1  2. Вычисление кумулятивных сумм  3. Фомирование расширенной последовательности S'  4. Подсчёт количества нулей L  5. Вычисление количества посещений каждого состояния  6. Вычисление статистик Y\_j для каждого состояния  7. Проверка всех статистик на превышение CONST  """  states = [str(i) for i in range(-9, 10) if i != 0]  fixed\_bit\_seq = [2 \* bit - 1 for bit in bit\_seq]  S = [0, fixed\_bit\_seq[0]]  fixed\_bit\_seq = fixed\_bit\_seq[1:]  S.extend([S.append(S[-1] + bit) for bit in fixed\_bit\_seq])  S.append(0)  L = S.count(0)  theta = {state: S.count(int(state)) for state in states}  Y = {  j: (abs(theta[j]) - L) / math.sqrt(2 \* L \* (4 \* abs(int(j)) - 2))  for j in states  }  if all([y < CONST for y in list(Y.values())]):  return True  return False |

# Вывод

В ходе выполнения лабораторной работы было разработано приложение, позволяющее генерировать, загружать, сохранять и анализировать псевдослучайные битовые последовательности с использованием трёх статистических тестов: частотного теста, теста на последовательность одинаковых бит и расширенного теста на произвольные отклонения. Каждый из тестов был реализован в соответствии с теоретическими основами, проверяя различные аспекты случайности последовательностей, такие как сбалансированность нулей и единиц, распределение цепочек одинаковых бит и поведение кумулятивных сумм.

Реализованные тесты позволяют оценить, насколько близка исследуемая последовательность к истинно случайной, используя пороговое значение константы. Приложение предоставляет удобный интерфейс для взаимодействия: пользователь может задавать длину последовательности, генерировать её, загружать из файла или сохранять результаты для дальнейшего анализа.

Практическая значимость работы заключается в возможности проверки качества генераторов псевдослучайных последовательностей, что особенно важно в контексте задач информационной безопасности, где надёжность таких генераторов напрямую влияет на стойкость криптографических систем. Успешное прохождение тестов свидетельствует о хороших статистических свойствах последовательности, что является необходимым условием для её использования в криптографии и других приложениях, требующих высокой степени случайности.

Таким образом, в рамках лабораторной работы было создано программное средство для тестирования псевдослучайных последовательностей, которое может быть использовано для учебных и практических целей в области анализа и обеспечения информационной безопасности.

**Приложение А. bit\_tests.py**

|  |
| --- |
| from typing import Optional  import random  import math  import json  CONST = 1.82138636 # Критическое значение статистики для всех тестов  def generate\_bit\_seq(  seq\_len: Optional[int] = 10000,  input\_file\_path: Optional[str] = "",  output\_file\_path: Optional[str] = "",  seed: int = 42  ) -> list[int]:  """  Генерирует или загружает последовательность битов (0 и 1).  Args:  seq\_len: Длина генерируемой последовательности (по умолчанию 10000 бит)  input\_file\_path: Путь к файлу для загрузки последовательности  output\_file\_path: Путь для сохранения последовательности  seed: Seed для генератора случайных чисел  Returns:  list[int]: последовательность битов (0 и 1).    Note:  Если указан input\_file\_path, последовательность загружается из файла.  Иначе генерируется случайная последовательность заданной длины.  """  bit\_seq = None  random.seed(seed)  if input\_file\_path != "":  with open(input\_file\_path, "r", encoding="utf-8") as f:  bit\_seq = list(f.read())  bit\_seq = [int(bit) for bit in bit\_seq]  bit\_seq = [0 if random.random() < 0.5 else 1 for \_ in range(seq\_len)] if bit\_seq is None else bit\_seq  if output\_file\_path != "":  with open(output\_file\_path, "w", encoding="utf-8") as f:  f.write("".join(map(str, bit\_seq)))    return bit\_seq  def frequency\_test(bit\_seq: list[int]) -> bool:  """  Частотный тест (Frequency Test).  Проверяет пропорцию нулей и единиц в последовательности.  Определяет, является ли кол-во нулей и единиц приблизительно таким же,  как в истинно случайно последовательности.  Args:  bit\_seq: Последовательность битов (0 и 1)  Returns:  bool: True если тест пройден (последовательность случайна), False в противном  случае  Algorithm:  1. Преобразование битов в значения -1 и 1  2. Вычисление суммы S\_n  3. Вычисление статистики S = |S\_n| / sqrt(n)  4. Сравнение с критическим значением CONST  """  fixed\_bit\_seq = [2 \* bit - 1 for bit in bit\_seq]  S\_n = sum(fixed\_bit\_seq)  if (abs(S\_n) / (math.sqrt(len(bit\_seq)))) <= CONST:  return True  return False  def r(bit\_seq: list[int], k: int):  """  Вспомогательная функция для теста на последовательность одинаковых бит.  Определяет, является ли k-тый бит началом новой цепочки.  Args:  bit\_seq: Последовательность битов  k: Индекс текущего бита    Returns:  int: 1 если бит k и k+1 разные (начало новой цепочки) 0 если одинаковые  """  if bit\_seq[k] == bit\_seq[k + 1]:  return 0  return 1  def identical\_bit\_seq\_test(bit\_seq: list[int]) -> bool:  """  Тест на последовательность одинаковых бит (Rusn Test).  Анализирует количество цепочек (непрерывных последовательностей одинаковых бит)  в проверяемой последовательности  Args:  bit\_seq: Последовательность битов (0 и 1)    Returns:  bool: True если тест пройден, False в противном случае    Algorithm:  1. Вычисление частоты единиц (pi)  2. Вычисление количества переходов V\_n  3. Вычисление статистики S  4. Сравнение с критическим значением CONST  """  pi = 1 / (len(bit\_seq)) \* sum(bit\_seq)  V\_n = sum([r(bit\_seq, k) for k in range(len(bit\_seq) - 1)]) + 1  S = abs(V\_n - 2 \* len(bit\_seq) \* pi \* (1 - pi))  S /= 2 \* math.sqrt(2 \* len(bit\_seq) \* pi \* (1 - pi))  if S <= CONST:  return True  return False  def extended\_random\_deviation\_test(bit\_seq: list[int]) -> bool:  """  Расширенный тест на произвольные отклонения.  Оценивает общее число посещений определенного состояния при произвольном обходе  кумулятивный суммы. Состоит из 18 подтестов для состояний из -9 до 9 (кроме 0).  Args:  bit\_seq: Последовательность битов (0 и 1)    Returns:  bool: True если все 18 тестов пройдены, False если хотя бы один не пройден    Algorithm:  1. Преобразование битов в -1 и 1  2. Вычисление кумулятивных сумм  3. Фомирование расширенной последовательности S'  4. Подсчёт количества нулей L  5. Вычисление количества посещений каждого состояния  6. Вычисление статистик Y\_j для каждого состояния  7. Проверка всех статистик на превышение CONST  """  states = [str(i) for i in range(-9, 10) if i != 0]  fixed\_bit\_seq = [2 \* bit - 1 for bit in bit\_seq]  S = [0, fixed\_bit\_seq[0]]  fixed\_bit\_seq = fixed\_bit\_seq[1:]  S.extend([S.append(S[-1] + bit) for bit in fixed\_bit\_seq])  S.append(0)  L = S.count(0)  theta = {state: S.count(int(state)) for state in states}  Y = {  j: (abs(theta[j]) - L) / math.sqrt(2 \* L \* (4 \* abs(int(j)) - 2))  for j in states  }  if all([y < CONST for y in list(Y.values())]):  return True  return False    def run\_tests(seq\_len: Optional[int]) -> tuple[list[int], dict]:  """  Основная функция для запуска всех тестов псевдослучайных последовательностей.  Args:  seq\_len: Длина последовательности для генерации    Returns:  tuple: (строковое представление последовательности, словарь с результатами тестов)    Note:  Загружает конфигурацию из config.json, генерирует/загружает последовательность  выполняет все три теста и возвращает результаты  """  config\_path = "/home/v\_vedin/university/labs/FouthCourse/FirstTerm/Information-Security/lab1/config.json"  with open(config\_path, "r", encoding="utf-8") as f:  config = json.load(f)    bit\_seq = generate\_bit\_seq(  seq\_len, config["input\_file\_path"],  config["output\_file\_path"]  )    str\_bit\_seq = "".join(str(bit) for bit in bit\_seq)  test\_results = {  "Частотный тест": frequency\_test(bit\_seq),  "Тест на последовательность одинаковых бит": identical\_bit\_seq\_test(bit\_seq),  "Расширенный тест на произвольные отклонения": extended\_random\_deviation\_test(bit\_seq)  }  return str\_bit\_seq, test\_results |

**Приложение Б. Interface.py**

|  |
| --- |
| from rich.console import Console  from rich.panel import Panel  from rich.prompt import Prompt  from rich import print  import json  import os  import time  from bits\_tests import run\_tests  class Interface:  """  Графический интерфейс для тестирования псведослучайных последовательностей.  Предоставляет интерактивное меню для работы с программой тестирования битовых последовательностей.  Включает главное меню, меню запуска тестов, справку и настройки.  Attributes:  console (Console): Объект для работы с консольным выводом (библиотека rich)  config\_path: Путь к файлу конфигурации  config: Загруженная конфигурация программы  """  def \_\_init\_\_(self) -> None:  """  Инициализирет интерфейс, загружает конфигурацию из JSON-файла  """  self.console = Console()  self.config\_path = "/home/v\_vedin/university/labs/FouthCourse/FirstTerm/Information-Security/lab1/config.json"  with open(self.config\_path, "r", encoding="utf-8") as f:  self.config = json.load(f)    def print\_title(self) -> None:  """  Отображает заголовок программы с информацеий об университете и студенте.  Очищает экран и выводит:  - Шапку с названием университета  - Название лабораторной работы и дисциплины  - Информацию о студенте  """  self.console.clear()  self.console.rule('Федеральное государственное бюджетное образовательное ' \  'учреждение высшего образования "Ульяновский ' \  'государственный технический университет"', style="blue")  self.console.print(  "\nЛабораторная Работа №1\n" \  "Дисциплина: Информационная Безопасность\n" \  "Работа выполнена студентом группы ИВТАСбд-41 Ведином Владимиром Александровичом\n",  justify="center"  )  self.console.rule(style="blue")    def print\_menu(self) -> None:  """  Отображает главное меню программы и обрабатывает выбор пользователя.  Пункты меню:  1. Старт программы - запуск тестирования последовательности  2. Инструкция - отображание справки по программе  3. Настройки - изменение путей к файлам  4. Выход - завершение программы  Вызывает соответствующие методы в зависимости от выбора пользователя  """  print(Panel.fit(  "[bold cyan] Главное Меню [/bold cyan]",  style="cyan",  ))  print("1. Старт программы")  print("2. Инструкция")  print("3. Настройки")  print("4. Выход")  self.console.rule(style="blue")  choice = Prompt.ask(  "[bold yellow]Выберите пункт[/bold yellow]",  choices = ["1", "2", "3", "4"],  default="1"  )  if choice == "1":  self.run\_menu()  elif choice == "2":  self.help\_menu()  elif choice == "3":  self.settings\_menu()  elif choice == "4":  print("Завершение программыы...")  time.sleep(0.5)  self.console.clear()  exit(0)    def run\_menu(self) -> None:  """  Запускает процесс тестирования последовательности битов.  Если не указан входной файл, запрашивает длину генерируемой последовательности.  Выполняет тесты и отображает результаты:  - Выводит саму последовательность  - Показывает результаты каждого теста (пройден/не пройден)  - Прекращает тестирование при первом неудачном тесте    После завершения тестов возвращает управление в главное меню  """  seq\_len = 10000  if self.config["input\_file\_path"] == "":  self.console.clear()  self.console.rule(style="blue")  print(Panel.fit(  "[bold green] Запуск программы [/bold green]",  style="green"  ))  self.console.rule(style="blue")  seq\_len = int(Prompt.ask(  "[bold yellow]Укажите длину генерируемой последовательности[/bold yellow]",  default="10000"  ))  self.console.clear()  self.console.rule("Результаты тестов", style="blue")  bit\_seq, test\_data = run\_tests(seq\_len)  print("Вывод битов...")  if self.config["input\_file\_path"] != "":  print(  "Последовательность прочитанная из файла:\n"  f"{bit\_seq}"  )  else:  print(  "Сгенерированная последовательность\n"  f"{bit\_seq}"  )      for name, res in test\_data.items():  print(f"Результаты {name}: ", end="")  if res == True:  text = "пройден"  style = "green"  else:  text = "непройден"  style = "red"  self.console.print(text, style=style)  if res == False:  print("Последовательность не случайна")  Prompt.ask(  "[bold yellow]Нажмите любую кнопку, чтобы вернуться в главное меню[/bold yellow]"  )  break  print("Все тесты пройдены, последовательность случайна")  Prompt.ask(  "[bold yellow]Нажмите любую кнопку, чтобы вернуться в главное меню[/bold yellow]"  )      def help\_menu(self) -> None:  """  Отображает инструкцию по работе с программой.  Содержит информацию о:  - Назначении программы  - Доступных тестах и их описании  - Возможностях генерации и загрузки последовательностей  - Логике выполнения тестов  """  self.console.clear()  self.console.rule(style="blue")  print(Panel.fit(  "[bold blue] Инструкция [/bold blue]"  ))  print(  "Это программа по тестированию последовательности бит на равномерность и случайность \n" \  "В ней представлены такие тесты как:\n" \  "\t1. Частотный тест (Оценка пропорции нулей и единиц в последовательности)\n" \  "\t2. Тест на последовательность одинаковых бит (Анализ кол-ва непрерывных последовательностей одинаковых бит)\n" \  "\t3. Расширенный тест на произвольные отклонения (Оцнека общего числа посещения состояния при произвольном обходе кумулятивной суммы)\n" \  "\t\tСостояния - последовательность чисел вида [-9, -8, ..., -1, 1, 2, ..., 9]\n\n"  )  print(  "Саму последовательность любой длинны (на выбор пользователя) можно как случайно генерировать" \  " так и считывать из файла, путь до кторого пользователь может указать\n" \  "Также, саму последовательность можно и сохранить в файл, снова указав до него путь\n" \  "Указать путь до входного и выходного файла можно в настройках\n\n"  )  print(  "Тесты проводятся по очереди (от 1 до 3, как указано выше). Если какой-то из тестов не проходит, то остальные не проводятся\n"  )  self.console.rule(style="blue")  Prompt.ask(  "[bold yellow]Нажмите любую кнопку чтобы выйти[/bold yellow]",  )  return  def settings\_menu(self) -> None:  """  Позволяет настроить пути к входному и выходному фалам.  Предоставляет возможности:  1. Указать путь до входного файла с последовательностью  2. Указать путь до выходного файла  3. Вернуться в главное меню  Проверяет существование указанных файлов и сохраняет настройки в конфигурационный файл  """  self.console.clear()  self.console.rule(style="blue")  print(Panel.fit(  "[bold magenta] Настройки [/bold magenta]"  ))  print(  "1. Указать путь до входного файла\n" \  "2. Указать путь до выходного файла\n" \  "3. Выход\n" \  )  self.console.rule(style="blue")  choice = Prompt.ask(  "[bold yellow]Выберите пункт[/bold yellow]",  choices = ["1", "2", "3"],  default="1"  )  if choice == "1" or choice == "2":  choices\_str\_dict = {  "1": "входного",  "2": "выходного"  }  choices\_keys\_dict = {  "1": "input\_file\_path",  "2": "output\_file\_path"  }  self.console.clear()  self.console.rule(style="blue")  print(  f"Текущий путь до {choices\_str\_dict[choice]} файла: " \  f"{self.config[choices\_keys\_dict[choice]] if self.config[choices\_keys\_dict[choice]] != "" else "не указан"}"  )  self.console.rule(style="blue")    while True:  file\_path = Prompt.ask(  f"[bold yellow]Введите путь до {choices\_str\_dict[choice]} файла или введите 0 для сброса[/bold yellow]",  default="Нажмите enter, чтобы вернуться назад"  )  if file\_path == "0":  self.config[choices\_keys\_dict[choice]] = ""  with open(self.config\_path, "w", encoding="utf-8") as f:  json.dump(self.config, f, indent=4)  print("Путь до файла сброшен")  break    if file\_path == "Нажмите enter, чтобы вернуться назад":  break  if os.path.exists(file\_path) and os.path.isfile(file\_path):  self.config[choices\_keys\_dict[choice]] = file\_path  with open(self.config\_path, "w", encoding="utf-8") as f:  json.dump(self.config, f, indent=4)  print("Путь до файла успешно сохранён")  break  print("Такого файла не существует или путь указан неверно попробуйте ещё раз")    def main\_loop(self) -> None:  """  Основный цикл работы интерфейса.  Бесконечно отображает заголовок и главное меню, обеспечивая нерперывную работу интерфейса  до явного выхода пользователя через пункт меню "Выход"  """  while True:  self.console.clear()  self.print\_title()  self.print\_menu() |

**Приложение В. main.py**

|  |
| --- |
| from interface import Interface  if \_\_name\_\_ == "\_\_main\_\_":  interface = Interface()  interface.main\_loop() |