

Федеральное агентство связи
Федеральное государственное бюджетное образовательное
учреждение высшего образования
«Сибирский государственный университет телекоммуникаций и
информатики»

Лабораторная работа №11

Выполнил:
студенты 4 курса
группы ИП-216
Андрущенко Ф.А.
Литвинов А. Е.
Русецкий А. С.

Проверил:
преподаватель кафедры ПМиК
Агалаков Антон Александрович

Новосибирск, 2025 г.

Задание

1. Разработать программу для вероятностного моделирования процесса написания программы программистом с длиной словаря программы = 16, 32, 64, 128
2. С помощью разработанной программы получить статистические оценки:
 - длины программы L ,
 - дисперсии длины $D(L)$,
 - среднеквадратического отклонения ($\sigma(L)$),
 - относительной ожидаемой погрешности.
3. С помощью приведенных формул получить теоретические значения и сравнить их с результатами моделирования
4. По тексту разработанной программы посчитать длину ее словаря и длину программы. Рассчитать длину программы по размеру ее словаря с помощью приведенных формул. Сравнить посчитанное по тексту значение длины текста программы, с длиной текста программы, полученной по формуле
5. По первому и второму пунктам задания определить * 2 – число единых по смыслу входных и выходных параметров представленных в сжатой без избыточной форме. Сравнить прогнозируемую длину программы с длиной программы, рассчитанной по тексту программы

Исходный код программы

```
import random
import math
import statistics
import tokenize
import io
import keyword
import builtins

class ProgramLengthSimulator:
    def __init__(self):
        self.results = {}
        self.python_keywords = set(keyword.kwlist)
        self.builtin_names = set(dir(builtins))

    def simulate_program_writing(self, eta, num_trials=10000):
        """
        Моделирование процесса написания программы
        eta - размер словаря программы
        num_trials - количество испытаний
        """
        lengths = []

        for _ in range(num_trials):
            unique_items = set()
```

```

        program_length = 0

        while len(unique_items) < eta:
            item = random.randint(1, eta)
            program_length += 1
            unique_items.add(item)

        lengths.append(program_length)

    mean_length = statistics.mean(lengths)
    variance = statistics.variance(lengths)
    std_dev = statistics.stdev(lengths)
    relative_error = std_dev / mean_length if mean_length > 0 else 0

    self.results[eta] = {
        "experimental": {
            "mean_length": mean_length,
            "variance": variance,
            "std_dev": std_dev,
            "relative_error": relative_error,
        },
        "theoretical": self.calculate_theoretical_values(eta),
    }

    return lengths

def calculate_theoretical_values(self, eta):
    """Вычисление теоретических значений по формулам"""
    theoretical_length = 0.9 * eta * math.log2(eta)
    theoretical_variance = (math.pi**2 * eta**2) / 6
    theoretical_std_dev = math.sqrt(theoretical_variance)
    theoretical_relative_error = 1 / (2 * math.log2(eta)) if eta > 1 else 0
    alternative_length = eta * math.log2(eta)

    return {
        "mean_length": theoretical_length,
        "variance": theoretical_variance,
        "std_dev": theoretical_std_dev,
        "relative_error": theoretical_relative_error,
        "alternative_length": alternative_length,
    }

def analyze_program_text(self, program_text):
    """
    Корректный лексический анализ текста программы
    Возвращает словарь и длину в терминах операторов и операндов
    """
    operators = set()
    operands = set()
    total_tokens = 0

    try:

```

```

f = io.BytesIO(program_text.encode("utf-8"))
for tok in tokenize.tokenize(f.readline):
    if tok.type in (
        tokenize.COMMENT,
        tokenize.NL,
        tokenize.NEWLINE,
        tokenize.ENCODING,
        tokenize.ENDMARKER,
    ):
        continue

    total_tokens += 1
    token_str = tok.string

    if tok.type == tokenize.OP:
        operators.add(token_str)
    elif tok.type == tokenize.NAME:
        if token_str in self.python_keywords:
            operators.add(token_str)
        elif token_str in self.builtin_names:
            operands.add(token_str)
        else:
            operands.add(token_str)
    elif tok.type in (tokenize.NUMBER, tokenize.STRING):
        operands.add(token_str)

except tokenize.TokenError:
    pass

eta = len(operators) + len(operands)
actual_length = total_tokens

if eta > 1:
    predicted_length_1 = 0.9 * eta * math.log2(eta)
    predicted_length_2 = eta * math.log2(eta)
else:
    predicted_length_1 = predicted_length_2 = 0

error_1 = (
    abs(actual_length - predicted_length_1) / actual_length * 100
    if actual_length > 0
    else 0
)
error_2 = (
    abs(actual_length - predicted_length_2) / actual_length * 100
    if actual_length > 0
    else 0
)

return {
    "eta": eta,
    "actual_length": actual_length,

```

```

        "predicted_length_1": predicted_length_1,
        "predicted_length_2": predicted_length_2,
        "error_1": error_1,
        "error_2": error_2,
        "operators_count": len(operators),
        "operands_count": len(operands),
    }

```

```

def analyze_eta2_star(self, program_text):
    """

```

```

    Определение  $\eta_2^*$  -- уникальные операнды
    (переменные, параметры, константы),
    исключая ключевые слова и встроенные функции.
    """

```

```

    operands = set()

```

```

    try:

```

```

        f = io.BytesIO(program_text.encode("utf-8"))

```

```

        for tok in tokenize.tokenize(f.readline):

```

```

            if tok.type in (
                tokenize.COMMENT,
                tokenize.NL,
                tokenize.NEWLINE,
                tokenize.ENCODING,
                tokenize.ENDMARKER,

```

```

            ):

```

```

                continue

```

```

            if tok.type == tokenize.NAME:

```

```

                if (
                    tok.string not in self.python_keywords
                    and tok.string not in self.builtin_names
                ):

```

```

                    operands.add(tok.string)

```

```

            elif tok.type in (tokenize.NUMBER, tokenize.STRING):

```

```

                operands.add(tok.string)

```

```

        except tokenize.TokenError:

```

```

            pass

```

```

    return operands

```

```

def main():

```

```

    simulator = ProgramLengthSimulator()

```

```

    eta_values = [16, 32, 64, 128]

```

```

    print("=" * 80)

```

```

    print("ЛАБОРАТОРНАЯ РАБОТА №1")

```

```

    print("Вероятностное моделирование метрических характеристик программ")

```

```

    print("=" * 80)

```

```

for eta in eta_values:
    print(f"\n--- МОДЕЛИРОВАНИЕ ДЛЯ  $\eta$  = {eta} ---")
    lengths = simulator.simulate_program_writing(eta, 10000)

    exp = simulator.results[eta]["experimental"]
    theory = simulator.results[eta]["theoretical"]

    print("ЭКСПЕРИМЕНТАЛЬНЫЕ РЕЗУЛЬТАТЫ:")
    print(f"Средняя длина программы L: {exp['mean_length']:.2f}")
    print(f"Дисперсия D(L $\eta$ ): {exp['variance']:.2f}")
    print(f"СКО  $\sqrt{D(L\eta)}$ : {exp['std_dev']:.2f}")
    print(
        f"Относительная погрешность  $\delta$ : {exp['relative_error']:.4f} "
        f"({exp['relative_error'] * 100:.2f}%)")
    )

    print("\nТЕОРЕТИЧЕСКИЕ ЗНАЧЕНИЯ:")
    print(f"Средняя длина программы L: {theory['mean_length']:.2f}")
    print(f"Дисперсия D(L $\eta$ ): {theory['variance']:.2f}")
    print(f"СКО  $\sqrt{D(L\eta)}$ : {theory['std_dev']:.2f}")
    print(
        f"Относительная погрешность  $\delta$ : {theory['relative_error']:.4f} "
        f"({theory['relative_error'] * 100:.2f}%)")
    )
    print(f"Альтернативная формула L: {theory['alternative_length']:.2f}")

    length_error = (
        abs(exp["mean_length"] - theory["mean_length"])
        / theory["mean_length"]
        * 100
    )
    print(f"\nОшибка предсказания длины: {length_error:.2f}%)

print("\n" + "=" * 80)
print("АНАЛИЗ ТЕКСТА ПРОГРАММЫ (ЭТОГО СКРИПТА)")
print("=" * 80)

with open(__file__, "r", encoding="utf-8") as f:
    program_text = f.read()

analysis = simulator.analyze_program_text(program_text)

print(f"Размер словаря программы  $\eta$ : {analysis['eta']}")
print(f" из них операторов: {analysis['operators_count']}")
print(f" из них операндов: {analysis['operands_count']}")
print(f"Фактическая длина программы (токенов): {analysis['actual_length']}")
print(f"Прогнозируемая длина (формула 1): {analysis['predicted_length_1']:.2f}")
print(f"Прогнозируемая длина (формула 2): {analysis['predicted_length_2']:.2f}")
print(f"Ошибка прогноза (формула 1): {analysis['error_1']:.2f}%)")
print(f"Ошибка прогноза (формула 2): {analysis['error_2']:.2f}%)")

```

```

print("\n" + "=" * 80)
print("ОПРЕДЕЛЕНИЕ  $\eta_2^*$  - ЧИСЛА УНИКАЛЬНЫХ ОПЕРАНДОВ")
print("=" * 80)

operands = simulator.analyze_eta2_star(program_text)
eta2_star = len(operands)

print(f"Число уникальных операндов  $\eta_2^*$ : {eta2_star}")
examples = list(operands)[:10]
print(f"Примеры операндов: {examples}{'...' if len(operands) > 10 else ''}")

if eta2_star > 1:
    predicted_by_eta2 = 0.9 * eta2_star * math.log2(eta2_star)
    print(f"\nПрогноз длины по  $\eta_2^*$  (условно): {predicted_by_eta2:.2f}")
    print(f"Фактическая длина: {analysis['actual_length']}")
else:
    print("\nНедостаточно операндов для прогноза по  $\eta_2^*$ ")

if __name__ == "__main__":
    main()

```

Модульные тесты для тестирования класса

```

import unittest
from ProgramLengthSimulator import ProgramLengthSimulator
import math

class TestProgramLengthSimulator(unittest.TestCase):
    """Тесты для класса ProgramLengthSimulator"""

    def setUp(self):
        """Инициализация перед каждым тестом"""
        self.simulator = ProgramLengthSimulator()

    def test_initialization(self):
        """Тест инициализации симулятора"""
        self.assertIsNotNone(self.simulator.results)
        self.assertIsNotNone(self.simulator.python_keywords)
        self.assertIsNotNone(self.simulator.builtin_names)
        self.assertIsInstance(self.simulator.results, dict)

    def test_simulate_program_writing(self):
        """Тест симуляции написания программы"""
        eta = 16
        lengths = self.simulator.simulate_program_writing(eta, num_trials=100)

        self.assertEqual(len(lengths), 100)
        self.assertIn(eta, self.simulator.results)

```

```
self.assertIn("experimental", self.simulator.results[eta])
self.assertIn("theoretical", self.simulator.results[eta])
```

```
def test_simulate_program_writing_values(self):
    """Тест корректности значений симуляции"""
    eta = 32
    self.simulator.simulate_program_writing(eta, num_trials=1000)

    exp = self.simulator.results[eta]["experimental"]

    self.assertGreater(exp["mean_length"], eta)
    self.assertGreater(exp["variance"], 0)
    self.assertGreater(exp["std_dev"], 0)
    self.assertGreaterEqual(exp["relative_error"], 0)
```

```
def test_calculate_theoretical_values(self):
    """Тест вычисления теоретических значений"""
    eta = 64
    theory = self.simulator.calculate_theoretical_values(eta)

    self.assertIn("mean_length", theory)
    self.assertIn("variance", theory)
    self.assertIn("std_dev", theory)
    self.assertIn("relative_error", theory)
    self.assertIn("alternative_length", theory)

    self.assertGreater(theory["mean_length"], 0)
    self.assertGreater(theory["variance"], 0)
    self.assertAlmostEqual(
        theory["std_dev"], math.sqrt(theory["variance"]), places=5
    )
```

```
def test_theoretical_formulas(self):
    """Тест корректности теоретических формул"""
    eta = 128
    theory = self.simulator.calculate_theoretical_values(eta)

    expected_mean = 0.9 * eta * math.log2(eta)
    expected_variance = (math.pi**2 * eta**2) / 6
    expected_alt_length = eta * math.log2(eta)

    self.assertAlmostEqual(theory["mean_length"], expected_mean, places=5)
    self.assertAlmostEqual(theory["variance"], expected_variance, places=5)
    self.assertAlmostEqual(
        theory["alternative_length"], expected_alt_length, places=5
    )
```

```
def test_analyze_simple_program(self):
    """Тест анализа простой программы"""
    program = ""
```

```
x = 5
y = 10
```

```

z = x + y
print(z)
"""
    analysis = self.simulator.analyze_program_text(program)

    self.assertIn("eta", analysis)
    self.assertIn("actual_length", analysis)
    self.assertIn("predicted_length_1", analysis)
    self.assertIn("operators_count", analysis)
    self.assertIn("operands_count", analysis)

    self.assertGreater(analysis["eta"], 0)
    self.assertGreater(analysis["actual_length"], 0)

def test_analyze_program_with_keywords(self):
    """Тест анализа программы с ключевыми словами"""
    program = """
if x > 0:
    for i in range(10):
        print(i)
"""
    analysis = self.simulator.analyze_program_text(program)

    self.assertGreater(analysis["operators_count"], 0)
    self.assertGreater(analysis["operands_count"], 0)

def test_analyze_eta2_star(self):
    """Тест определения уникальных операндов"""
    program = """
x = 5
y = 10
z = x + y
result = x * y
"""
    operands = self.simulator.analyze_eta2_star(program)

    self.assertIsInstance(operands, set)
    self.assertGreater(len(operands), 0)
    self.assertIn("x", operands)
    self.assertIn("y", operands)
    self.assertIn("z", operands)

def test_analyze_eta2_star_no_keywords(self):
    """Тест что ключевые слова не включаются в операнды"""
    program = """
if True:
    for i in range(10):
        pass
"""
    operands = self.simulator.analyze_eta2_star(program)

    self.assertNotIn("if", operands)

```

```
self.assertNotIn("for", operands)
self.assertNotIn("in", operands)
self.assertNotIn("True", operands)
```

```
def test_error_calculations(self):
    """Тест вычисления ошибок прогноза"""
    program = "x = 1 + 2"
    analysis = self.simulator.analyze_program_text(program)

    self.assertIn("error_1", analysis)
    self.assertIn("error_2", analysis)
    self.assertGreaterEqual(analysis["error_1"], 0)
    self.assertGreaterEqual(analysis["error_2"], 0)
```

```
def test_empty_program(self):
    """Тест анализа пустой программы"""
    program = ""
    analysis = self.simulator.analyze_program_text(program)

    self.assertEqual(analysis["eta"], 0)
    self.assertEqual(analysis["actual_length"], 0)
```

```
def test_multiple_eta_values(self):
    """Тест симуляции для нескольких значений eta"""
    eta_values = [16, 32, 64]

    for eta in eta_values:
        self.simulator.simulate_program_writing(eta, num_trials=100)
        self.assertIn(eta, self.simulator.results)
```

```
def test_convergence_with_trials(self):
    """Тест сходимости результатов с увеличением числа испытаний"""
    eta = 32

    self.simulator.simulate_program_writing(eta, num_trials=100)
    result_100 = self.simulator.results[eta]["experimental"]["mean_length"]

    self.simulator.simulate_program_writing(eta, num_trials=10000)
    result_10000 = self.simulator.results[eta]["experimental"]["mean_length"]

    theory = self.simulator.results[eta]["theoretical"]["mean_length"]

    error_100 = abs(result_100 - theory) / theory
    error_10000 = abs(result_10000 - theory) / theory

    self.assertLessEqual(error_10000, error_100 * 2)
```

```
if __name__ == "__main__":
    from rich import print
    from rich.panel import Panel
    from rich.console import Console
```

```

from unittest import TextTestRunner, TestResult, TestSuite

console = Console()

class RichTestResult(TestResult):
    def __init__(self, stream, descriptions, verbosity):
        super().__init__(stream, descriptions, verbosity)

    def startTest(self, test):
        super().startTest(test)
        console.print(f"[cyan]Running:[/cyan] {test._testMethodName}")

    def addSuccess(self, test):
        super().addSuccess(test)
        console.print(f"[green]✓ PASS:[/green] {test._testMethodName}")

    def addFailure(self, test, err):
        super().addFailure(test, err)
        console.print(f"[red]✗ FAIL:[/red] {test._testMethodName}")

    def addError(self, test, err):
        super().addError(test, err)
        console.print(f"[magenta]💣 ERROR:[/magenta] {test._testMethodName}")

loader = unittest.defaultTestLoader
suite = TestSuite()
suite.addTests(loader.loadTestsFromTestCase(TestProgramLengthSimulator))

runner = TextTestRunner(resultclass=RichTestResult, verbosity=0)
result = runner.run(suite)

console.print(
    Panel.fit(
        f"[green]Passed: {result.testsRun - len(result.failures) - "
len(result.errors)}[/green]\n"
        f"[red]Failed: {len(result.failures)}[/red]\n"
        f"[magenta]Errors: {len(result.errors)}[/magenta]\n"
        f"[yellow]Total: {result.testsRun}[/yellow]",
        title="Test Results",
    )
)

```

Статистические и расчетные характеристики длин программ для заданных размеров словарей

Статистики по словарям η

Для $\eta=16$: $\bar{L} \approx 54.15$, $Var \approx 369.21$, $SD \approx 19.21$, $\delta \approx 0.3548$; теория $L \approx 57.60$, $Var \approx 421.10$, $SD \approx 20.52$, $\delta \approx 0.1250$; альтернативно $L \approx 64.00$; ошибка предсказания среднего $\approx 5.98\%$

Для $\eta=32$: $\bar{L} \approx 129.84$, $Var \approx 1492.86$, $SD \approx 38.64$, $\delta \approx 0.2976$; теория $L \approx 144.00$, $Var \approx 1684.41$, $SD \approx 41.04$, $\delta \approx 0.1000$; альтернативно $L \approx 160.00$; ошибка предсказания среднего $\approx 9.84\%$

Для $\eta=64$: $\bar{L} \approx 301.81$, $Var \approx 6218.69$, $SD \approx 78.86$, $\delta \approx 0.2613$; теория $L \approx 345.60$, $Var \approx 6737.65$, $SD \approx 82.08$, $\delta \approx 0.0833$; альтернативно $L \approx 384.00$; ошибка предсказания среднего $\approx 12.67\%$

Для $\eta=128$: $\bar{L} \approx 692.76$, $Var \approx 26106.37$, $SD \approx 161.57$, $\delta \approx 0.2332$; теория $L \approx 806.40$, $Var \approx 26950.60$, $SD \approx 164.17$, $\delta \approx 0.0714$; альтернативно $L \approx 896.00$; ошибка предсказания среднего $\approx 14.09\%$

Таблица

η	Эксп. \bar{L}	Эксп. Var	Эксп. SD	Эксп. δ	Теор. L	Теор. Var	Теор. SD	Теор. δ	Альт. L	Ошибка \bar{L}
16	54.15	369.21	19.21	0.3548	57.60	421.10	20.52	0.1250	64.00	5.98%
32	129.84	1492.86	38.64	0.2976	144.00	1684.41	41.04	0.1000	160.00	9.84%
64	301.81	6218.69	78.86	0.2613	345.60	6737.65	82.08	0.0833	384.00	12.67%
128	692.76	26106.37	161.57	0.2332	806.40	26950.60	164.17	0.0714	896.00	14.09%

Длина разработанной программы:

По анализу текста: число операторов 33, операндов 99, словарь $\eta=132$, фактическая длина (число значимых токенов) 618 .

Прогноз по формулам:

$L1=0.9 \eta \log_2 \eta \approx 836.87$ $L1=0.9 \eta \log_2 \eta \approx 836.87$, $L2=\eta \log_2 \eta \approx 929.86$ $L2=\eta \log_2 \eta \approx 929.86$;

ошибки относительно фактической длины: 35.42% 35.42% и 50.46% 50.46% соответственно

Результат тестирования методов класса

```
> python Test_ProgramLengthSimulator.py
Running: test_analyze_eta2_star
✓ PASS: test_analyze_eta2_star
Running: test_analyze_eta2_star_no_keywords
✓ PASS: test_analyze_eta2_star_no_keywords
Running: test_analyze_program_with_keywords
✓ PASS: test_analyze_program_with_keywords
Running: test_analyze_simple_program
✓ PASS: test_analyze_simple_program
Running: test_calculate_theoretical_values
✓ PASS: test_calculate_theoretical_values
Running: test_convergence_with_trials
✓ PASS: test_convergence_with_trials
Running: test_empty_program
✓ PASS: test_empty_program
Running: test_error_calculations
✓ PASS: test_error_calculations
Running: test_initialization
✓ PASS: test_initialization
Running: test_multiple_eta_values
✓ PASS: test_multiple_eta_values
Running: test_simulate_program_writing
✓ PASS: test_simulate_program_writing
Running: test_simulate_program_writing_values
✓ PASS: test_simulate_program_writing_values
Running: test_theoretical_formulas
✓ PASS: test_theoretical_formulas
Ran 13 tests in 0.377s
```

OK

Test Results

Passed: 13
Failed: 0
Errors: 0
Total: 13