



МИНОБРНАУКИ РОССИИ

*Федеральное государственное бюджетное образовательное учреждение
высшего образования*

«МИРЭА – Российский технологический университет»

РТУ МИРЭА

Институт Информационных технологий (ИТ)

Кафедра Математического обеспечения и стандартизации
информационных технологий (МОСИТ)

ОТЧЕТ ПО ПРАКТИЧЕСКОЙ РАБОТЕ № 5

по дисциплине

«Тестирование и верификация программного обеспечения»

Выполнили студенты группы ИКБО-04-22

Егоров Л.А.

Принял ассистент

Петрова А.А.

Практическая работа
выполнена

«__» _____ 202__ г.

(подпись
студента)

«Зачтено»

«__» _____ 202__ г.

(подпись
руководителя)

Москва 2024

СОДЕРЖАНИЕ

Содержание.....	2
1 Статический анализ.....	3
1.1 Анализ кода на языке Python.....	3
1.1.1 Добавление ошибок в код.....	4
1.2 Анализ кода на языке C.....	6
1.2.1 Внесённые ошибки.....	7
1.3 Вывод.....	10
2 Динамический анализ.....	12
2.1 Анализ кода на языке Python.....	12
2.1.1 Внесённые ошибки.....	13
2.2 Анализ кода на языке C.....	14
2.2.1 Внесённые ошибки.....	16
2.3 Вывод.....	20
Вывод.....	22
Приложение А.....	23

1 СТАТИЧЕСКИЙ АНАЛИЗ

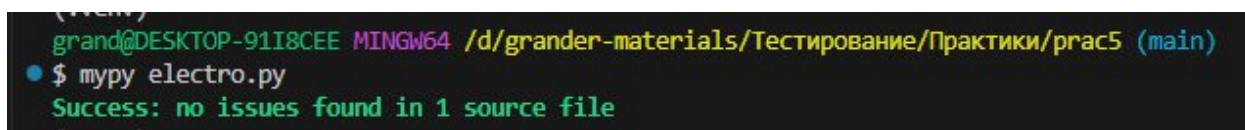
Статический анализ кода — это процесс анализа исходного кода программы без его исполнения, направленный на поиск ошибок, уязвимостей, нарушений стандартов и других потенциальных проблем. Он проводится с использованием специальных инструментов (статических анализаторов), которые могут анализировать код на разных уровнях: от синтаксического до логического.

1.1 Анализ кода на языке Python

Для анализа был использован код электромагнитного алгоритма, который представлен в Листинге А.1.

Для анализа кода на языке Python использовалось три анализатора — MyPy, Pylint, flake8.

MyPy — это статический анализатор кода для Python, который проверяет типы данных в программах, написанных на Python. Он позволяет разработчикам использовать аннотации типов, чтобы явно указывать ожидаемые типы переменных, аргументов функций и возвращаемых значений. MyPy анализирует код и выявляет потенциальные ошибки, связанные с несоответствием типов, что помогает улучшить качество кода и предотвратить ошибки на этапе разработки. Результат анализа этой программой представлен на Рисунке 1.1.1.



```
grand@DESKTOP-91I8CEE MINGW64 /d/grander-materials/Тестирование/Практики/prac5 (main)
$ mypy electro.py
Success: no issues found in 1 source file
```

Рисунок 1.1.1 – Анализ программой MyPy

Pylint — это статический анализатор кода для Python, который проверяет качество и стиль кода на соответствие стандартам PEP 8 и другим рекомендациям по написанию чистого и поддерживаемого кода. Pylint анализирует исходный код и генерирует отчеты о найденных ошибках,

потенциальных проблемах, нарушениях стиля и других аспектах качества кода. Результат анализа этой программой представлен на Рисунке 1.1.2.

```
grand@DESKTOP-91I8CEE MINGW64 /d/grander-materials/Тестирование/Практики/prac5 (main)
$ pylint electro.py

-----
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

Рисунок 1.1.2 – Анализ программой pylint

Flake8 — это инструмент статического анализа кода для Python, который объединяет в себе несколько других инструментов для проверки качества и стиля кода. Flake8 включает в себя проверки на соответствие стандартам PEP 8, поиск синтаксических ошибок и потенциальных проблем в коде. Он объединяет функциональность трех основных компонентов: PyFlakes, pycodestyle (ранее известный как pep8) и mccabe. Результат анализа этой программой представлен на Рисунке 1.1.3.

```
grand@DESKTOP-91I8CEE MINGW64 /d/grander-materials/Тестирование/Практики/prac5 (main)
$ flake8 electro.py
```

Рисунок 1.1.3 – Анализ программой flake8

1.1.1 Добавление ошибок в код

В код исходной программы добавлено 5 ошибок:

1. Из функций были убраны строки документации (Рисунок 1.1.4).
2. Одна строка была сделана большой длины (больше 79 символов) (Рисунок 1.1.5).
3. Добавлена неиспользуемая переменная (Рисунок 1.1.6).
4. Допущена ошибка в типизации нескольких функций (Рисунок 1.1.7).
5. Добавлены лишние отступы между функциями (Рисунок 1.1.8).

```

41
42 def create_population(self) -> None:
43     self.x = np.vstack([
44         self._min + np.random.uniform(0, 1, size=self.n) *
45         (self._max - self._min) for _ in range(self.population_size)
46     ])
47     self.calculate_best()
48

```

Рисунок 1.1.4 – Отсутствие документации в функции

```

84
85     else:
86         force[i] += (((self.x[i] - self.x[j]) / np.linalg.norm(self.x[j] - self.x[i])**2) * q[i] * q[j])
87
88     return force
89

```

Рисунок 1.1.5 – Строка длиной больше 79 символов

```

74         (np.sum(values - best_value)))
75     force = np.zeros_like(self.x)
76     unused = 1
77     for i in range(self.population_size):
78         for j in range(self.population_size):
79             if i != j:

```

Рисунок 1.1.6 – Добавление неиспользуемой переменной

```

1 def rastrigin(x: NDArray[np.float32]) -> str:
2     ...
3     Вычисление функции Растригина
4     ...
5     return float(np.sum(x**2 - 10 * np.cos(2 * np.pi * x) + 10))
6

```

Рисунок 1.1.7 – Ошибка в типизации результата функции

```

31
32 def calculate_best(
33     self) -> Tuple[NDArray[np.float32], float, NDArray[np.float32]]:
34     ...
35     Расчёт лучшего решения на итерации
36     ...
37     values = np.array([rastrigin(x) for x in self.x])
38     best_value = float(np.min(values))
39     best_x = self.x[np.where(abs(values - best_value) < 1e-3)].flatten()
40     return values, best_value, best_x
41
42
43
44
45
46 def create_population(self) -> None:
47     self.x = np.vstack([
48         self._min + np.random.uniform(0, 1, size=self.n) *
49         (self._max - self._min) for _ in range(self.population_size)
50     ])
51     self.calculate_best()
52
53 def local_search(self) -> None:
54     ...
55

```

Рисунок 1.1.8 – Добавление лишних отступов

Результаты анализа представлены на Рисунках 1.1.9 - 1.1.11.

```
grand@DESKTOP-91I8CEE MINGW64 /d/grander-materials/Тестирование/Практики/prac5 (main)
$ mypy --strict electro.py
electro.py:15: error: Incompatible return value type (got "float", expected "str") [return-value]
Found 1 error in 1 file (checked 1 source file)
```

Рисунок 1.1.1.1 – Анализ программой муру

```
grand@DESKTOP-91I8CEE MINGW64 /d/grander-materials/Тестирование/Практики/prac5 (main)
$ pylint electro.py
***** Module electro
electro.py:90:0: C0301: Line too long (120/100) (line-too-long)
electro.py:46:4: C0116: Missing function or method docstring (missing-function-docstring)
electro.py:80:8: W0612: Unused variable 'unused' (unused-variable)

-----
Your code has been rated at 9.62/10 (previous run: 9.88/10, -0.27)
```

Рисунок 1.1.1.2 – Анализ программой pylint

```
grand@DESKTOP-91I8CEE MINGW64 /d/grander-materials/Тестирование/Практики/prac5 (main)
$ flake8 electro.py
electro.py:46:5: E303 too many blank lines (5)
electro.py:80:9: F841 local variable 'unused' is assigned to but never used
electro.py:90:80: E501 line too long (120 > 79 characters)
```

Рисунок 1.1.1.3 – Анализ программой flake8

В результате программа муру обнаружила ошибку, связанную с типизацией. Pylint больше указала на стилистические ошибки, а flake8 указала все ошибки, которые допущены в оформлении по PEP8.

1.2 Анализ кода на языке C

Для анализа взята программа на языке C, выполняющая операцию добавления строки в матрицу. Её код представлен в Листинге А.2.

Для анализа кода используются статические анализаторы cppcheck, clang-tidy и gcc analyzer.

Cppcheck — это легковесный статический анализатор кода, который выполняет анализ исходного кода на предмет ошибок, связанных с безопасностью, производительностью и другими аспектами качества кода. Он поддерживает языки C и C++ и предназначен для выявления потенциальных проблем, которые могут быть пропущены компилятором или другими инструментами. Результат анализа представлен на Рисунке 1.2.1.


```
~/grander-materials/Тестирование/Практики/prac5 main* ↑
● .venv > cppcheck --enable=all --suppress=missingIncludeSystem --check-level=exhaustive --checkers-report=main.log main.c
Checking main.c ...
nofile:0:0: information: Active checkers: 108/836 [checkersReport]
```

Рисунок 1.2.1 – Результат анализа `cppcheck`

Clang-Tidy — это инструмент статического анализа кода, встроенный в компилятор Clang, который выполняет анализ исходного кода на предмет ошибок, нарушений стиля, потенциальных уязвимостей и других аспектов качества кода. Clang-Tidy поддерживает языки C, C++ и Objective-C и предназначен для автоматизации процесса улучшения и очистки кода. Результат анализа представлен на Рисунке 1.2.2.

```
grander@grander-arch ~/grander-materials/Тестирование/Практики/prac5 main* ↓
● > clang-tidy main.c
Error while trying to load a compilation database:
Could not auto-detect compilation database for file "main.c"
No compilation database found in /home/grander/grander-materials/Тестирование/Практики/prac5 or any parent directory
json-compilation-database: Error while opening JSON database: No such file or directory
fixed-compilation-database: Error while opening fixed database: No such file or directory
Running without flags.
```

Рисунок 1.2.2 – Результат анализа `clang-tidy`

GCC -fanalyzer — это опция компилятора GCC, которая включает встроенный статический анализатор кода. Этот анализатор выполняет глубокий анализ исходного кода на предмет ошибок, потенциальных уязвимостей и других аспектов качества кода. Он работает на этапе компиляции и может выявлять проблемы, которые могут быть пропущены другими инструментами. Результат анализа представлен на Рисунке 1.2.3.

```
grander@grander-arch ~/grander-materials/Тестирование/Практики/prac5 main* ↓
● > gcc -fanalyzer main.c
```

Рисунок 1.2.3 – Результат анализа `gcc -fanalyzer`

1.2.1 Внесённые ошибки

В код исходной программы добавлено 5 ошибок:

1. Во внутренний цикл добавлена переменная, которая перекрывает переменную внешнего цикла (Рисунок 1.2.4).
2. Для считывания данных использована небезопасная функция `scanf` (Рисунок 1.2.5).
3. Добавлена бесконечный цикл `for` (Рисунок 1.2.6).

4. Добавлено обращение к неинициализированному участку памяти (Рисунок 1.2.7).
5. Убрана очистка памяти после завершения программы (Рисунок 1.2.8).

```
for (int i = 0; i < n; i++) {  
    printf("Введите строку\n");  
    for (int j = 0; j < m; j++) {  
        if (read_number(&a[i][j]) != 0) {  
            for (int j = 0; j < i; j++) {  
                free(a[j]);  
            }  
            free(a);  
            return 1;  
        }  
    }  
}
```

Рисунок 1.2.4 – Перекрывание переменных

```
while (n < 1) {  
    printf("Введите высоту матрицы: ");  
    scanf("%d", &n);  
}  
while (m < 1) {
```

Рисунок 1.2.5 – Использование небезопасной функции scanf

```
for (int j = 0; j < m; j++) {  
    if (read_number(&a[0][j]) != 0) {  
        for (int i = 0; i < n; i++) {  
            free(a[i]);  
        }  
        free(a);  
        return 1;  
    }  
}
```

Рисунок 1.2.6 – Бесконечный цикл for


```

int** a = (int**)malloc(n * sizeof(int*));
if (a == NULL) {
    printf(a[0][0]);
    return -1;
}

```

Рисунок 1.2.7 – Обращение к неинициализированному участку памяти

```

int main()
{
    int n, m;
    float result;
    n = 0;
    m = 0;
    while (n < 1) {
        printf("Введите высоту матрицы: ");
    }
}

```

Рисунок 1.2.8 – Добавление неиспользуемой переменной

На Рисунках 1.2.9 - 1.2.11 представлены результаты анализа получившегося кода тремя анализаторами.

```

~/grander-materials/Тестирование/Практики/prac5 main* ↑
• .venv > cppcheck --enable=all --suppress=missingIncludeSystem --check-level=exhaustive --checkers-report=main.log main.c
Checking main.c ...
main.c:49:16: warning: Either the condition 'a==NULL' is redundant or there is possible null pointer dereference: a. [nullPointerCheck]
    printf(a[0][0]);
           ^
main.c:48:11: note: Assuming that condition 'a==NULL' is not redundant
    if (a == NULL) {
        ^
main.c:49:16: note: Null pointer dereference
    printf(a[0][0]);
           ^
main.c:66:26: style: Local variable 'j' shadows outer variable [shadowVariable]
    for (int j = 0; j < i; j++) {
                   ^
main.c:64:18: note: Shadowed declaration
    for (int j = 0; j < m; j++) {
           ^
main.c:66:26: note: Shadow variable
    for (int j = 0; j < i; j++) {
                   ^
main.c:34:11: style: Unused variable: result [unusedVariable]
    float result;
        ^
nofile:0:0: information: Active checkers: 108/836 [checkersReport]

```

Рисунок 1.2.9 – Результат анализа cppcheck

```
~/grander-materials/Тестирование/Практики/prac5 main* ↑
.venv > clang-tidy main.c
Error while trying to load a compilation database:
Could not auto-detect compilation database for file "main.c"
No compilation database found in /home/grander/grander-materials/Тестирование/Практики/prac5 or any parent directory
json-compilation-database: Error while opening JSON database: No such file or directory
fixed-compilation-database: Error while opening fixed database: No such file or directory
Running without flags.
3 warnings generated.
/home/grander/grander-materials/Тестирование/Практики/prac5/main.c:39:9: warning: Call to function 'scanf' is insecure as it does not provide security checks introduced in the C11 standard. Replace with analogous functions that support length arguments or provides boundary checks such as 'scanf_s' in case of C11 [clang-analyzer-security.insecureAPI.DeprecatedOrUnsafeBufferHandling]
 39 |         scanf("%d", &n);
    |         ^~~~~~
/home/grander/grander-materials/Тестирование/Практики/prac5/main.c:39:9: note: Call to function 'scanf' is insecure as it does not provide security checks introduced in the C11 standard. Replace with analogous functions that support length arguments or provides boundary checks such as 'scanf_s' in case of C11
 39 |         scanf("%d", &n);
    |         ^~~~~~
/home/grander/grander-materials/Тестирование/Практики/prac5/main.c:49:22: warning: Array access (from variable 'a') results in a null pointer dereference [clang-analyzer-core.NullDereference]
 49 |         printf("%d", a[0][0]);
    |                        ^
/home/grander/grander-materials/Тестирование/Практики/prac5/main.c:37:5: note: Loop condition is true. Entering loop body
 37 |         while (n < 1) {
```

Рисунок 1.2.10 – Результат анализа clang-tidy

```
~/grander-materials/Тестирование/Практики/prac5 main* ↑
.venv > gcc -fanalyzer main.c
main.c: В функции «main»:
main.c:49:23: предупреждение: dereference of NULL «a» [CWE-476] [-Wanalyzer-null-dereference]
 49 |         printf("%d", a[0][0]);
    |                        ~^~~~
```

Рисунок 1.2.11 – Результат анализа gcc –fanalyzer

```
main.c:108:31: предупреждение: infinite loop [CWE-835] [-Wanalyzer-infinite-loop]
108 |         for (int i = 0; i < n; j++) {
    |                                ~^~~~
«main»: event 1
```

Рисунок 1.2.12 – Результат анализа gcc –fanalyzer

Все программы смогли обнаружить обращение к неинициализированному участку памяти; cppcheck также обнаружил перекрытие переменной во внутреннем цикле и объявление неиспользуемой переменной; clang-tidy указал на использование небезопасной функции scanf; gcc –fanalyzer указал на бесконечный цикл.

1.3 Вывод

1. Раннее обнаружение ошибок: Статический анализ позволяет обнаруживать потенциальные ошибки и несоответствия типов на ранних этапах разработки, до запуска программы. Это сокращает время на отладку и улучшает качество кода.

2. Улучшение читаемости и поддерживаемости: Аннотации типов делают код более понятным для разработчиков, особенно в больших проектах. Это облегчает навигацию по коду и его поддержку.

3. Автоматизация проверок: Инструменты статического анализа, такие как `tu`, могут быть интегрированы в системы непрерывной интеграции (CI/CD), что позволяет автоматизировать процесс проверки кода на соответствие стандартам и правилам.

4. Также статический анализ позволяет обнаружить участки кода, которые очень сложны для чтения и которые нужно разбить на функции, а также ошибки в оформлении кода.

2 ДИНАМИЧЕСКИЙ АНАЛИЗ

2.1 Анализ кода на языке Python

Для динамического анализа кода на языке Python использованы два анализатора кода – `memory_profiler` и `line_profiler`.

memory_profiler — это инструмент для профилирования использования памяти в Python-программах. Он позволяет отслеживать, сколько памяти используется на каждой строке кода, что полезно для выявления утечек памяти, оптимизации использования памяти и понимания, какие части кода потребляют больше всего ресурсов. Результат анализа этой программой представлен на Рисунке 2.1.1.

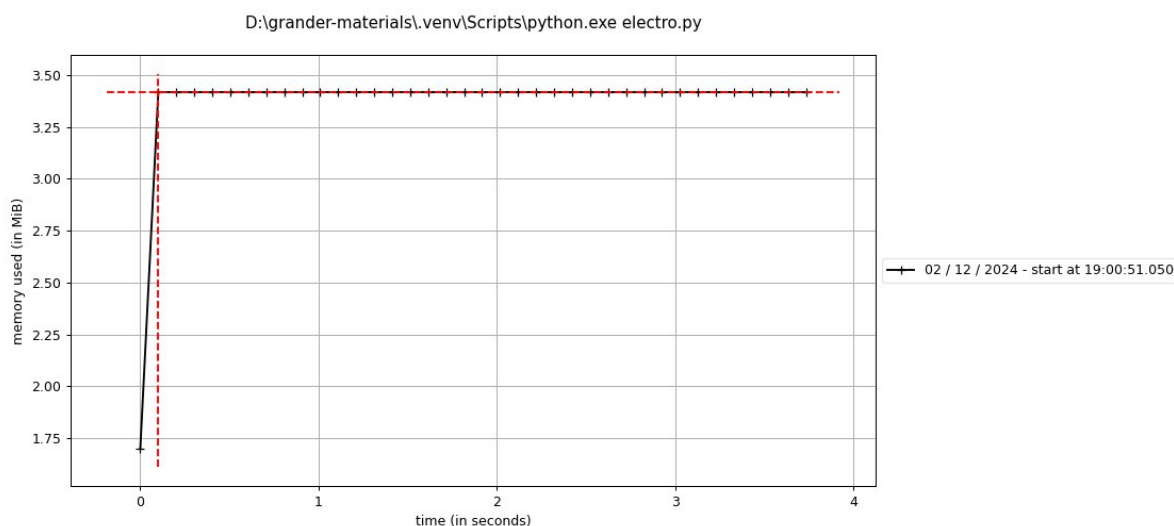


Рисунок 2.1.1 – Результат анализа `memory_profiler`

line_profiler — это инструмент для профилирования производительности кода на уровне строк в Python. Он позволяет измерять время выполнения каждой строки кода, что полезно для выявления узких мест в производительности и оптимизации кода. Результат анализа представлен на Рисунке 2.1.2.

Total time: 0.377938 s
File: electro.py
Function: solve at line 115

Line #	Hits	Time	Per Hit	% Time	Line Contents
=====					
115					@line_profile
116					def solve(self) -> None:
117					...
118					Запуск алгоритма
119					...
120	1	15171.7	15171.7	4.0	self.create_population()
121	1	0.6	0.6	0.0	history = []
122	1	0.4	0.4	0.0	max_iter = 20
123	21	18.8	0.9	0.0	for i in range(max_iter):
124	20	12898.2	644.9	3.4	_, best_value, best_x = self.calculate_best()
125	20	32.3	1.6	0.0	history.append(best_value)
126	40	7125.6	178.1	1.9	print(
127	40	448.1	11.2	0.1	f'Текущее лучшее значение: {round(best_value, 4)}'
128	20	777.5	38.9	0.2	f' в точке {list(map(lambda x: round(x, 4), best_x))}'
129)
130	20	4328.8	216.4	1.1	print(f'Итерация: {i + 1}')
131	20	72295.3	3614.8	19.1	self.local_search()
132	20	232650.5	11632.5	61.6	force = self.calculate_force()
133	20	32190.3	1609.5	8.5	self.move_particles(force)

Рисунок 2.1.2 – Результат анализа line_profiler

2.1.1 Внесённые ошибки

В код программы внесены три ошибки:

- в функции вычисления функции Растригина добавлена задержка на 0.3 секунды (Рисунок 2.1.3);
- в функцию вычисления лучшего результата добавлено лишнее заполнение массива (Рисунок 2.1.4);
- что-то там (Рисунок 2.1.5).

```
def rastrigin(x: NDArray[np.float32]) -> float:
    ...
    Вычисление функции Растригина
    ...
    sleep(0.3)
    return float(np.sum(x**2 - 10 * np.cos(2 * np.pi * x) + 10))
```

Рисунок 2.1.3 – Задержка в функции Растригина

```
if self.s is None:
    self.s = [elem for elem in self.x for _ in range(2)]
else:
    self.s = [elem for elem in self.s for _ in range(2)]
for i in range(len(self.s)):
    self.s.extend([1, 1, 1])
values = np.array([rastrigin(x) for x in self.x])
best_value = float(np.min(values))
```

Рисунок 2.1.4 – Лишние заполнения массива


```

if abs(values[i] - best_value) > 1e-3:
    alpha = 1
    while alpha > 0.001:
        alpha = np.random.uniform()

```

Рисунок 2.1.5 – Неэффективное вычисление случайного числа

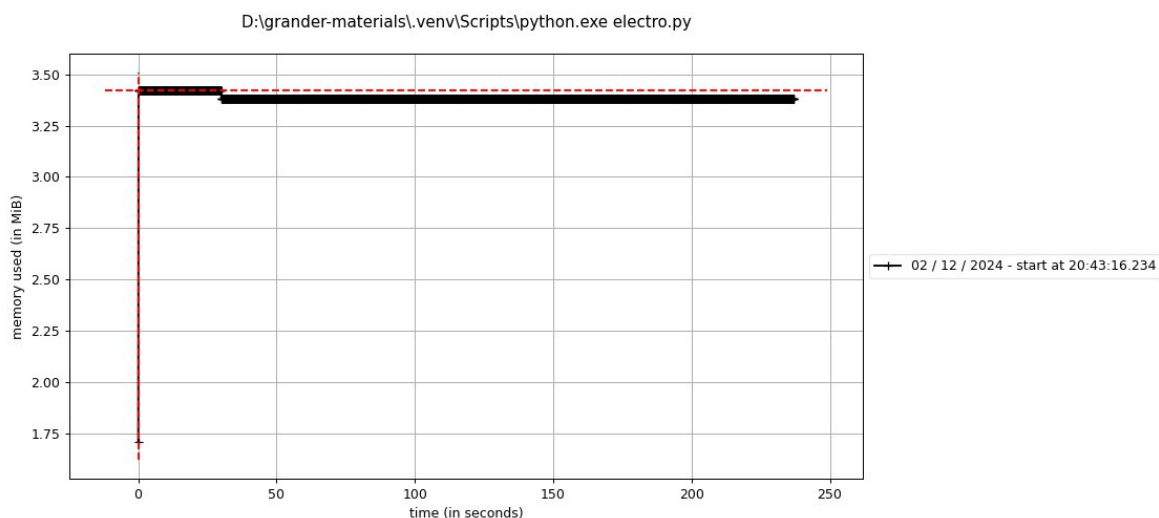


Рисунок 2.1.6 – Результат анализа memory_profiler

Total time: 269.863 s
File: electro.py
Function: solve at line 121

Line #	Hits	Time	Per Hit	% Time	Line Contents
121					@line_profile
122					def solve(self) -> None:
123					...
124					Запуск алгоритма
125					...
126	1	9019208.2	9e+06	3.3	self.create_population()
127	1	1.3	1.3	0.0	history = []
128	1	0.6	0.6	0.0	max_iter = 5
129	6	8.6	1.4	0.0	for i in range(max_iter):
130	5	44948453.2	9e+06	16.7	_, best_value, best_x = self.calculate_best()
131	5	25.4	5.1	0.0	history.append(best_value)
132	10	1751.4	175.1	0.0	print(
133	10	144.3	14.4	0.0	f'Текущее лучшее значение: {round(best_value, 4)}'
134	5	364.8	73.0	0.0	f' в точке {list(map(lambda x: round(x, 4), best_x))}'
135)
136	5	938.9	187.8	0.0	print(f'Итерация: {i + 1}')
137	5	125749707.2	3e+07	46.6	self.local_search()
138	5	45014713.2	9e+06	16.7	force = self.calculate_force()
139	5	45128069.3	9e+06	16.7	self.move_particles(force)

Рисунок 2.1.7 – Результат анализа line_profiler

2.2 Анализ кода на языке C

Для динамического анализа кода на языке C использованы два динамических анализатора – Valgrind и DynamoRIO.

Valgrind — это мощный инструмент для динамического анализа программ, написанных на языках C, C++ и других языках, которые могут быть скомпилированы в исполняемые файлы для архитектуры x86. Valgrind предоставляет несколько инструментов для анализа использования памяти, поиска утечек памяти, профилирования производительности и других аспектов качества кода. Результат анализа представлен на Рисунке 2.2.1.

```
grander@grander-arch ~/grander-materials/Тестирование/Практики/prac5 main* ↓ 11s
> valgrind ./main
==48736== Memcheck, a memory error detector
==48736== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==48736== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==48736== Command: ./main
==48736==
Введите высоту матрицы: 2
Введите ширину матрицы: 2
Введите строку
1
1
Введите строку
2
2
Результат ввода
1 1
2 2
Введите строку для вставки
3
3
Результат выполнения
3 3
1 1
2 2
==48736==
==48736== HEAP SUMMARY:
==48736==    in use at exit: 0 bytes in 0 blocks
==48736== total heap usage: 7 allocs, 7 frees, 2,112 bytes allocated
==48736==
==48736== All heap blocks were freed -- no leaks are possible
==48736==
==48736== For lists of detected and suppressed errors, rerun with: -s
==48736== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Рисунок 2.2.1 – Результат анализа Valgrind

DynamoRIO — это мощная платформа для динамического инструментирования программ, которая позволяет вставлять инструкции в исполняемый код во время выполнения программы. DynamoRIO поддерживает программы на языках C, C++ и других, которые могут быть скомпилированы в исполняемые файлы для архитектуры x86 и x86-64. Она

предоставляет гибкий и мощный механизм для анализа и модификации кода на лету, что делает её полезной для различных задач, таких как профилирование, отладка, оптимизация и анализ безопасности. Результат анализа представлен на Рисунке 2.2.2.

```
grander@grander-arch ~/grander-materials/Тестирование/Практики/prac5 main* ↓
● > ~/Загрузки/DynamoRIO-Linux-11.0.0/bin64/drrun -t drmemory -- ./main
~Dr.M~ Dr. Memory version 2.6.20005
Введите высоту матрицы: 2
Введите ширину матрицы: 2
Введите строку
1
1
Введите строку
2
2
Результат ввода
1 1
2 2
Введите строку для вставки
3
3
Результат выполнения
3 3
1 1
2 2
~Dr.M~
~Dr.M~ NO ERRORS FOUND:
~Dr.M~      0 unique,      0 total unaddressable access(es)
~Dr.M~      0 unique,      0 total uninitialized access(es)
~Dr.M~      0 unique,      0 total invalid heap argument(s)
~Dr.M~      0 unique,      0 total warning(s)
~Dr.M~      0 unique,      0 total,      0 byte(s) of leak(s)
~Dr.M~      0 unique,      0 total,      0 byte(s) of possible leak(s)
~Dr.M~ ERRORS IGNORED:
~Dr.M~      2 unique,      2 total,    2048 byte(s) of still-reachable allocation(s)
~Dr.M~      (re-run with "-show_reachable" for details)
~Dr.M~ Details: /home/grander/Загрузки/DynamoRIO-Linux-11.0.0/drmemory/drmemory/logs/DrMemory-main.46575.000/results.txt
```

Рисунок 2.2.2 – Результат анализа DynamoRIO

2.2.1 Внесённые ошибки

В код программы внесены следующие ошибки:

1. Убрана очистка памяти после завершения программы (Рисунок 2.2.3).
2. Добавлено обращение к неинициализированному участку памяти (Рисунок 2.2.4).
3. Сделана очистка незанятой памяти (Рисунок 2.2.5).

```

~/grander-materials/Тестирование/Практики/prac5 main* ↑ 22s
● .venv > clang main.c -o main && valgrind ./main
==219755== Memcheck, a memory error detector
==219755== Copyright (C) 2002-2024, and GNU GPL'd, by Julian Seward et al.
==219755== Using Valgrind-3.24.0 and LibVEX; rerun with -h for copyright info
==219755== Command: ./main
==219755==
Введите высоту матрицы: 2
Введите ширину матрицы: 2
==219755== Use of uninitialised value of size 8
==219755==    at 0x48E2C9B: _itoa_word (_itoa.c:183)
==219755==    by 0x48EDCDD: __printf_buffer (vfprintf-process-arg.c:155)
==219755==    by 0x48EFE3: __vfprintf_internal (vfprintf-internal.c:1544)
==219755==    by 0x48E3C72: printf (printf.c:33)
==219755==    by 0x1093E2: main (in /home/grander/grander-materials/Тестирование/Практики/prac5/main)
==219755==
==219755== Conditional jump or move depends on uninitialised value(s)
==219755==    at 0x48E2CAC: _itoa_word (_itoa.c:183)
==219755==    by 0x48EDCDD: __printf_buffer (vfprintf-process-arg.c:155)
==219755==    by 0x48EFE3: __vfprintf_internal (vfprintf-internal.c:1544)
==219755==    by 0x48E3C72: printf (printf.c:33)
==219755==    by 0x1093E2: main (in /home/grander/grander-materials/Тестирование/Практики/prac5/main)
==219755==
==219755== Conditional jump or move depends on uninitialised value(s)
==219755==    at 0x48E99D: __printf_buffer (vfprintf-process-arg.c:186)
==219755==    by 0x48EFE3: __vfprintf_internal (vfprintf-internal.c:1544)
==219755==    by 0x48E3C72: printf (printf.c:33)
==219755==    by 0x1093E2: main (in /home/grander/grander-materials/Тестирование/Практики/prac5/main)
==219755==
==219755== Conditional jump or move depends on uninitialised value(s)
==219755==    at 0x48EE9D6: __printf_buffer (vfprintf-process-arg.c:208)
==219755==    by 0x48EFE3: __vfprintf_internal (vfprintf-internal.c:1544)
==219755==    by 0x48E3C72: printf (printf.c:33)
==219755==    by 0x1093E2: main (in /home/grander/grander-materials/Тестирование/Практики/prac5/main)
==219755==
==219755== Invalid read of size 8
==219755==    at 0x109458: main (in /home/grander/grander-materials/Тестирование/Практики/prac5/main)
==219755==    Address 0x4a7f8d0 is 0 bytes after a block of size 16 alloc'd
==219755==    at 0x48447A8: malloc (vg_replace_malloc.c:446)
==219755==    by 0x109381: main (in /home/grander/grander-materials/Тестирование/Практики/prac5/main)
==219755==

```

Рисунок 2.2.6 – Результат анализа Valgrind

```

00Введите строку
1
1
Введите строку
2
2
Результат ввода
1 1
2 2
Введите строку для вставки
3
3
Результат выполнения
3 3
1 1
2 2
==219755==
==219755== HEAP SUMMARY:
==219755==      in use at exit: 48 bytes in 4 blocks
==219755==    total heap usage: 7 allocs, 3 frees, 2,112 bytes allocated
==219755==
==219755== LEAK SUMMARY:
==219755==    definitely lost: 24 bytes in 1 blocks
==219755==    indirectly lost: 24 bytes in 3 blocks
==219755==    possibly lost: 0 bytes in 0 blocks
==219755==    still reachable: 0 bytes in 0 blocks
==219755==    suppressed: 0 bytes in 0 blocks
==219755== Rerun with --leak-check=full to see details of leaked memory
==219755==
==219755== Use --track-origins=yes to see where uninitialised values come from
==219755== For lists of detected and suppressed errors, rerun with: -s
==219755== ERROR SUMMARY: 9 errors from 5 contexts (suppressed: 0 from 0)

```

Рисунок 2.2.7 – Результат анализа Valgrind


```

~/grander-materials/Тестирование/Практики/prac5 main* ↑ 9s
.venv > ~/Загрузки/DynamoRIO-Linux-11.0.0/bin64/drrun -t drmemory -- ./main
~~Dr.M~~ Dr. Memory version 2.6.20005
Введите высоту матрицы: 2
Введите ширину матрицы: 2
~~Dr.M~~ WARNING: application is missing line number information.
~~Dr.M~~
~~Dr.M~~ Error #1: UNINITIALIZED READ: reading register eflags
~~Dr.M~~ # 0 libc.so.6!?          +0x0      (0x00007ef60929502e <libc.so.6+0x6402e>)
~~Dr.M~~ # 1 libc.so.6!?          +0x0      (0x00007ef609295fe4 <libc.so.6+0x64fe4>)
~~Dr.M~~ # 2 libc.so.6!_IO_printf +0xb2    (0x00007ef609289c73 <libc.so.6+0x58c73>)
~~Dr.M~~ # 3 main
~~Dr.M~~ Note: @0:00:03.710 in thread 225893
~~Dr.M~~ Note: instruction: cmovs %rax -> %r12
~~Dr.M~~
~~Dr.M~~ Error #2: UNINITIALIZED READ: reading register rcx
~~Dr.M~~ # 0 libc.so.6!?          +0x0      (0x00007ef609288c9b <libc.so.6+0x57c9b>)
~~Dr.M~~ # 1 libc.so.6!?          +0x0      (0x00007ef609293cde <libc.so.6+0x62cde>)
~~Dr.M~~ # 2 libc.so.6!?          +0x0      (0x00007ef609295fe4 <libc.so.6+0x64fe4>)
~~Dr.M~~ # 3 libc.so.6!_IO_printf +0xb2    (0x00007ef609289c73 <libc.so.6+0x58c73>)
~~Dr.M~~ # 4 main
~~Dr.M~~ Note: @0:00:03.712 in thread 225893
~~Dr.M~~ Note: instruction: movzx (%r8,%rcx) -> %eax
~~Dr.M~~
~~Dr.M~~ Error #3: UNINITIALIZED READ: reading register rax
~~Dr.M~~ # 0 libc.so.6!?          +0x0      (0x00007ef609288ca8 <libc.so.6+0x57ca8>)
~~Dr.M~~ # 1 libc.so.6!?          +0x0      (0x00007ef609293cde <libc.so.6+0x62cde>)
~~Dr.M~~ # 2 libc.so.6!?          +0x0      (0x00007ef609295fe4 <libc.so.6+0x64fe4>)
~~Dr.M~~ # 3 libc.so.6!_IO_printf +0xb2    (0x00007ef609289c73 <libc.so.6+0x58c73>)
~~Dr.M~~ # 4 main
~~Dr.M~~ Note: @0:00:03.712 in thread 225893
~~Dr.M~~ Note: instruction: cmp %rax $0x0000000000000009
~~Dr.M~~
~~Dr.M~~ Error #4: UNINITIALIZED READ: reading register r12
~~Dr.M~~ # 0 libc.so.6!?          +0x0      (0x00007ef60929499a <libc.so.6+0x6399a>)
~~Dr.M~~ # 1 libc.so.6!?          +0x0      (0x00007ef609295fe4 <libc.so.6+0x64fe4>)
~~Dr.M~~ # 2 libc.so.6!_IO_printf +0xb2    (0x00007ef609289c73 <libc.so.6+0x58c73>)
~~Dr.M~~ # 3 main
~~Dr.M~~ Note: @0:00:03.714 in thread 225893
~~Dr.M~~ Note: instruction: test %r12 %r12
~~Dr.M~~
~~Dr.M~~ Error #5: UNADDRESSABLE ACCESS beyond heap bounds: reading 0x00007ef6058009f0-0x00007ef6058009f8 8 byte(s)
~~Dr.M~~ # 0 main
~~Dr.M~~ Note: @0:00:03.719 in thread 225893
~~Dr.M~~ Note: next higher malloc: 0x00007ef605800a10-0x00007ef605800a18
~~Dr.M~~ Note: refers to 0 byte(s) beyond last valid byte in prior malloc
~~Dr.M~~ Note: prev lower malloc: 0x00007ef6058009e0-0x00007ef6058009f0
~~Dr.M~~ Note: instruction: mov (%rax,%rcx,8) -> %rdi
~~Dr.M~~
~~Dr.M~~ Error #6: INVALID HEAP ARGUMENT for free 0x00007ef5f5c228c8
~~Dr.M~~ # 0 replace_free [/home/runner/work/dynamorio/dynamorio/drmemory/common/alloc_replace.c:2710]
~~Dr.M~~ # 1 main
~~Dr.M~~ Note: @0:00:03.739 in thread 225893

```

Рисунок 2.2.8 – Результат анализа DynamoRIO

```

00Введите строку
1
1
Введите строку
2
2
Результат ввода
1 1
2 2
Введите строку для вставки
3
3
Результат выполнения
3 3
1 1
2 2
~~Dr.M~~
~~Dr.M~~ Error #7: LEAK 24 direct bytes 0x00007ef605800a70-0x00007ef605800a88 + 24 indirect bytes
~~Dr.M~~ # 0 replace_realloc      [/home/runner/work/dynamorio/dynamorio/drmemory/common/alloc_replace.c:2672]
~~Dr.M~~ # 1 main
~~Dr.M~~
~~Dr.M~~ ERRORS FOUND:
~~Dr.M~~      1 unique,      1 total unaddressable access(es)
~~Dr.M~~      4 unique,      8 total uninitialized access(es)
~~Dr.M~~      1 unique,      1 total invalid heap argument(s)
~~Dr.M~~      0 unique,      0 total warning(s)
~~Dr.M~~      1 unique,      1 total,      48 byte(s) of leak(s)
~~Dr.M~~      0 unique,      0 total,      0 byte(s) of possible leak(s)
~~Dr.M~~ ERRORS IGNORED:
~~Dr.M~~      2 unique,      2 total,      2048 byte(s) of still-reachable allocation(s)
~~Dr.M~~      (re-run with "-show_reachable" for details)
~~Dr.M~~ Details: /home/grander/Зарпузки/DynamoRIO-Linux-11.0.0/drmemory/drmemory/logs/DrMemory-main.225893.000/results.txt

```

Рисунок 2.2.9 – Результат анализа DynamoRIO

2.3 Вывод

1. Анализ во время выполнения: Динамический анализ позволяет отслеживать поведение программы во время её выполнения, что полезно для выявления ошибок, которые невозможно обнаружить статическим анализом (например, ошибки, связанные с состоянием программы).

2. Профилирование и оптимизация: Инструменты динамического анализа, такие как профилировщики, помогают выявить узкие места в производительности и оптимизировать код для более эффективного выполнения.

3. Отладка и трассировка: Динамический анализ позволяет отслеживать поток выполнения программы, что полезно для отладки и понимания причин возникновения ошибок.

4. Анализ взаимодействия с внешними системами: Динамический анализ может помочь выявить проблемы, связанные с взаимодействием

программы с внешними системами (базы данных, API и т.д.), которые сложно проверить статически.

ВЫВОД

Статический анализ предоставляет возможность раннего обнаружения ошибок и улучшения качества кода на этапе разработки, что сокращает время на отладку и повышает безопасность.

Динамический анализ позволяет отслеживать поведение программы во время выполнения, выявляя ошибки и проблемы, которые невозможно обнаружить статическим анализом.

Совместное использование статического и динамического анализа обеспечивает комплексный подход к обеспечению качества и надежности программного обеспечения. Статический анализ помогает предотвратить ошибки на ранних этапах, а динамический анализ позволяет убедиться, что программа работает корректно в реальных условиях.

ПРИЛОЖЕНИЕ А

Листинг А.1 – Код файла electro.py

```
'''
electro.py
Реализация электромагнитного алгоритма
'''

from typing import Tuple
import numpy as np
import matplotlib.pyplot as plt
from numpy.typing import NDArray

def rastrigin(x: NDArray[np.float32]) -> float:
    '''
    Вычисление функции Растригина
    '''
    return float(np.sum(x**2 - 10 * np.cos(2 * np.pi * x) + 10))

class EMA:
    '''
    Класс, реализующий электромагнитный алгоритм
    '''

    def __init__(self, n: int):
        self.n = n
        self.population_size = 10 * n
        self.local_iter = 10
        self.scale = 0.005
        self._min = -5.12
        self._max = -self._min
        self.x: NDArray[np.float32]
        self.values = None

    def calculate_best(
        self) -> Tuple[NDArray[np.float32], float, NDArray[np.float32]]:
        '''
        Расчёт лучшего решения на итерации
        '''
        values = np.array([rastrigin(x) for x in self.x])
        best_value = float(np.min(values))
        best_x = self.x[np.where(abs(values - best_value) < 1e-3)].flatten()
        return values, best_value, best_x

    def create_population(self):
        '''
        Создание агентов в популяции
        '''
        self.x = np.vstack([
            self._min + np.random.uniform(0, 1, size=self.n) *
            (self._max - self._min) for _ in range(self.population_size)
        ])

        self.calculate_best()

    def local_search(self) -> None:
        '''
        Реализация локального поиска
        '''
```



```

search_field = self.scale * (self._max - self._min)
for k, particle in enumerate(self.x):
    cnt = 0
    while cnt < self.local_iter:
        for i in range(self.n):
            sign = np.random.randint(0, 2) * 2 - 1
            y = particle.copy()
            velocity = np.random.uniform()
            y[i] += sign * velocity * search_field
            if rastrigin(y) < rastrigin(particle):
                self.x[k] = y.copy()
                cnt = self.local_iter
                break
        cnt += 1

def calculate_force(self) -> NDArray[np.float32]:
    """
    Расчёт электромагнитной силы
    """
    values, best_value, _ = self.calculate_best()
    q = np.exp(-self.n * (values - best_value) /
               (np.sum(values - best_value)))
    force = np.zeros_like(self.x)
    for i in range(self.population_size):
        for j in range(self.population_size):
            if i != j:
                if values[j] < values[i]:
                    force[i] += (
                        ((self.x[j] - self.x[i]) /
                         np.linalg.norm(self.x[j] - self.x[i])**2) * q[i]
                        *
                        q[j])
                else:
                    force[i] += (
                        ((self.x[i] - self.x[j]) /
                         np.linalg.norm(self.x[j] - self.x[i])**2) * q[i]
                        *
                        q[j])

    return force

def move_particles(self, force) -> None:
    """
    Передвижение частиц
    """
    values, best_value, _ = self.calculate_best()
    for i in range(self.population_size):
        if abs(values[i] - best_value) > 1e-3:
            alpha = 1
            alpha = np.random.uniform()
            velocity = np.ones_like(self.x[i])
            normalized_force = force[i] / np.linalg.norm(force[i])
            for j in range(self.n):
                if force[i][j] > 0:
                    velocity[j] = self._max - self.x[i][j]
                else:
                    velocity[j] = self.x[i][j] - self._min
            self.x[i] += alpha * np.multiply(normalized_force, velocity)

def solve(self) -> None:
    """
    Запуск алгоритма
    """
    self.create_population()

```

```

history = []
max_iter = 100
for i in range(max_iter):
    _, best_value, best_x = self.calculate_best()
    history.append(best_value)
    print(
        f'Текущее лучшее значение: {round(best_value, 4)}'
        f' в точке {list(map(lambda x: round(x, 4), best_x))}'
    )
    print(f'Итерация: {i + 1}')
    self.local_search()
    force = self.calculate_force()
    self.move_particles(force)

def main() -> None:
    '''
    Главная функция
    '''
    ema = EMA(2)
    ema.solve()

if __name__ == '__main__':
    main()

```

Листинг A.2 – Код файла main.c

```

#define _CRT_SECURE_NO_WARNINGS

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#include <stdlib.h>

int read_number(int* n) {
    char buffer[100];
    char *endptr;

    if (fgets(buffer, sizeof(buffer), stdin) != NULL) {
        char *newline = strchr(buffer, '\n');
        if (newline) {
            *newline = '\0';
        }

        *n = strtol(buffer, &endptr, 10);

        if (*endptr != '\0') {
            printf("Ошибка: введенная строка содержит нечисловые символы\n");
            return -1;
        }
        return 0;
    } else {
        printf("Ошибка ввода\n");
        return -1;
    }
}

int main()
{
    int n, m;
    n = 0;
    m = 0;
    while (n < 1) {

```

```

        printf("Введите высоту матрицы: ");
        if (read_number(&n) != 0) {
            return 1;
        };
    }
    while (m < 1) {
        printf("Введите ширину матрицы: ");
        if (read_number(&m) != 0) {
            return 1;
        };
    }
    int** a = (int**)malloc(n * sizeof(int*));
    if (a == NULL) {
        return -1;
    }
    for (int i = 0; i < n; i++) {
        a[i] = (int*)malloc(m * sizeof(int));
        if (a[i] == NULL) {
            for (int j = 0; j < i; j++) {
                free(a[j]);
            }
            free(a);
            return -1;
        }
    }
    for (int i = 0; i < n; i++) {
        printf("Введите строку\n");
        for (int j = 0; j < m; j++) {
            if (read_number(&a[i][j]) != 0) {
                for (int k = 0; k < i; k++) {
                    free(a[k]);
                }
                free(a);
                return 1;
            };
        }
    }
    printf("Результат ввода\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            if (j == m - 1) {
                printf("%d\n", a[i][j]);
            }
            else {
                printf("%d ", a[i][j]);
            }
        }
    }
    n += 1;
    int** b = (int**)realloc(a, n * sizeof(int*));
    if (b == NULL) {
        free(a);
        return -1;
    }
    a = b;
    a[n - 1] = (int*)calloc(1, m * sizeof(int));
    if (a[n - 1] == NULL) {
        for (int j = 0; j < n; j++) {
            free(a[j]);
        }
        free(a);
        return -1;
    }
}

```

```

    for (int i = n - 1; i > 0; i--) {
        for (int j = 0; j < m; j++) {
            a[i][j] = a[i - 1][j];
        }
    }
    printf("Введите строку для вставки\n");
    for (int j = 0; j < m; j++) {
        if (read_number(&a[0][j]) != 0) {
            for (int i = 0; i < n; i++) {
                free(a[i]);
            }
            free(a);
            return 1;
        }
    }
    printf("Результат выполнения\n");
    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++) {
            printf("%d ", a[i][j]);
        }
        printf("\n");
    }
    for (int i = 0; i < n; i++) {
        free(a[i]);
    }
    free(a);
    return 0;
}

```