**Теоретическая часть**

1. Типы ошибок в коде

***Синтаксическая ошибка***

*Определение:* Ошибка, возникающая при нарушении правил синтаксиса языка программирования, что мешает компиляции или интерпретации кода.

**Пример:**

python

if x > 0 # пропущена двоеточие

print("Положительное число")

***Логическая ошибка***

*Определение:* Ошибка, при которой программа работает без ошибок, но результат не соответствует ожидаемому из-за неправильной логики.

**Пример:**

python

# Ошибка в логике: условие должно быть x < 0, а не x > 0

if x > 0:

print("Отрицательное число")

***Ошибка в условии (неправильное условие)***

*Определение:* Ошибка, связанная с неправильным или некорректным условием в конструкции if, while и т.п., что приводит к неправильному выполнению блока кода.

**Пример:**

python

if x = 10 # неправильное использование оператора присваивания вместо сравнения

***Ошибка обработки граничных случаев***

*Определение:* Ошибка, возникающая при неправильной обработке крайних или особых случаев входных данных, что может привести к сбою или неправильным результатам.

**Пример:**

python

Не учитывается деление на ноль

def divide(a, b):

return a / b # при b=0 произойдет ошибка

***Отклонение от алгоритма (код делает не то, что описано)***

*Определение:* Ошибка, когда реализованный код не соответствует поставленной задаче или алгоритму, то есть логика выполнения не совпадает с ожидаемой.

**Пример**:

python

**Алгоритм:** найти максимум из двух чисел

def max\_of\_two(a, b):

return b if a > b else a # неправильно, должно быть: return a if a > b else b

2. Методы поиска ошибок

***Основные методы***

**1.Визуальный анализ кода**

- Просмотр кода для выявления очевидных ошибок, таких как синтаксические или логические ошибки, неправильное использование операторов.

**2. Сравнение с описанием алгоритма**

- Проверка соответствия реализации поставленной задаче или алгоритму, чтобы обнаружить отклонения или неправильные шаги.

**3. Ручное тестирование на конкретных примерах**

- Запуск программы с тестовыми данными, специально подобранными для проверки граничных случаев, ошибок ввода и других условий, чтобы выявить ошибки обработки граничных случаев и логические ошибки.

**Практическая часть**

**ЗАДАЧА**

def is\_perfect\_square(n):

if n < 0:

return False

if n == 0 or n == 1:

return True

low = 2

high = n // 2

while low <= high:

mid = (low + high) // 2

mid\_sq = mid \* mid

if mid\_sq == n:

return True

elif mid\_sq < n:

low = mid + 1

else:

high = mid - 1

return False

***Анализ кода функции is\_perfect\_square(n)***

1. Визуальный анализ и сравнение с алгоритмом

|  |  |  |  |
| --- | --- | --- | --- |
| Строка | Описание кода | Соответствие алгоритму | Возможные ошибки / отклонения |
| 1 | If n <= 0: | В алгоритме: если n < 0, вернуть false | Ошибка |
| 2 | Return false | - | - |
| 3 | If n == 1: | В алгоритме: если n == 0 или 1, вернуть true | Ошибка |
| 4 | Return true | - | - |
| 5 | Low = 2 | В алгоритме: диапазон от 2 до n // 2 | Верно |
| 6 | High = n | В алгоритме: high = n // 2 | Ошибка |
| 7-15 | Цикл бинарного поиска | В целом соответствует алгоритму | - |
| 8 | Mid = (low + high) // 2 | - | - |
| 9 | Mid\_sq = mid \* mid | - | - |
| 10-15 | Условия внутри цикла | В целом соответствует алгоритму | - |
| 16 | Return false | - | - |

2. Обнаруженные ошибки и отклонения

|  |  |  |
| --- | --- | --- |
| Строка | Описание ошибки / отклонения | Почему |
| Строка 1 | Условие if n <= 0: | Не соответствует спецификации, так как n – это квадрат, и должно возвращать True. Поэтому условие должно быть if n < 0: |
| Строка 3 | Условие if n == 1: | Не учитывает случай n=0, который тоже является квадратом. Следует заменить на if n == 0 or n == 1: |
| Строка 6 | High = n | Не соответствует алгоритму. Правильно: high = n // 2. Это важно для корректной работы бинарного поиска и предотвращения лишних итераций |

3. Мнимые "ошибки" (подозрительные, но корректные)

|  |  |  |
| --- | --- | --- |
| Строка | Обоснование | Почему корректно? |
| Строка 6 | High = n | Можно считать, что это не ошибка, а упрощение, хотя и не соответствует спецификации. |

4. Таблица тестовых случаев

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Входное значение | Ожидаемый результат | Фактический результат | Примечания |
| 1 | -4 | False | false (верно) | Отрицательное число |
| 2 | 0 | True | False (неверно) | Ошибка: из-за if n <= 0 возвращается False, хотя 0 — квадрат |
| 3 | 1 | True | True (верно) | — |
| 4 | 4 | True | True (верно) | — |
| 5 | 9 | True | True (верно) | — |
| 6 | 16 | True | True (верно) | — |
| 7 | 2 | False | False (верно) | — |
| 8 | 3 | False | False (верно) | — |
| 9 | 15 | False | False (верно) | — |
| 10 | 1000000 | True | True (верно) | — |
| 11 | 17 | False | False (верно) | — |

**Обнаруженные ошибки:** тесты для n=0показывают несоответствие, так как функция возвращает False, хотя по спецификации должна — True.

5. Исправленная версия функции

def is\_perfect\_square(n):

if n < 0: # Исправлено: отрицательные числа не могут быть квадратами

return False

if n == 0 or n == 1: # Объединено условие для 0 и 1

return True

low = 2

high = n // 2 # Исправлено: диапазон до n // 2

while low <= high:

mid = (low + high) // 2

mid\_sq = mid \* mid

if mid\_sq == n:

return True

elif mid\_sq < n:

low = mid + 1

else:

high = mid - 1

return False

6. Анализ и выводы

**- Наиболее серьезная ошибка**: условие if n <= 0: — оно неправильно обрабатывает случай n=0, что противоречит спецификации. Это могло привести к неправильным результатам для нулевых значений.

**- Чаще всего встречавшийся тип ошибки:** Отклонение от алгоритма — неправильное задание диапазона поиска (high = n вместо n // 2), что влияет на эффективность и точность.

**- Самый эффективный тест:** n=0, так как он выявил несоответствие в условии обработки граничных случаев.