МИНОБРНАУКИ РОССИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
"САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО"
Институт компьютерных наук и технологий
Направление **02.03.01** : Математика и компьютерные науки

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ
«Инспекция кода»

Исполнитель: _____ Яшнова Дарья Михайловна
группа 5130201/20002

Руководитель: _____ Курочкин Михаил Александрович

«___»_____ 2025г

Санкт-Петербург, 2025

Содержание

Введение

В процессе разработки программного обеспечения, помимо выполнения основной задачи, заключающейся в реализации функциональности, предусмотренной спецификацией, важной целью является обеспечение высокого качества созданного программного решения.

Тестирование программного обеспечения представляет собой процесс проверки соответствия фактического поведения программы ожидаемому, который осуществляется на основе заранее определенного набора тестов. Однако на практике ни один из методов тестирования не способен выявить абсолютно все дефекты в программном продукте. Это обусловлено ограниченностью ресурсов проекта, таких как бюджет, временные рамки и количество персонала, выделенного на выполнение задач, включая тестирование. Тем не менее, грамотно организованный процесс тестирования позволяет обнаружить большинство дефектов, что способствует их своевременному устранению и, как следствие, повышению качества программного обеспечения.

В рамках данной лабораторной работы применяется метод ручного тестирования, который представляет собой проверку программного обеспечения, осуществляемую специалистами без использования специализированных автоматизированных инструментов.

Следует отметить, что ручное тестирование не заменяет автоматизированное тестирование, а используется в сочетании с ним, что позволяет выявлять дефекты программного обеспечения на более ранних этапах разработки.

1 Постановка задачи

Задача: выполнить инспекцию кода, выступая в роли разработчика программного обеспечения.

Для достижения данной цели необходимо выполнить следующие шаги:

- Ознакомиться с основными методами ручного тестирования

- Провести анализ кода посредством инспекции, изучив его на наличие ошибок и недочётов.

- Внести необходимые исправления в программу, основываясь на рекомендациях, предоставленных специалистом по тестированию.

2 Описание методов инспекции кода

Ручное тестирование программного обеспечения — это процесс проверки кода и его функциональности без использования автоматизированных инструментов. Несмотря на стремительное развитие технологий и появление мощных средств автоматизированного тестирования, ручные методы остаются важной частью процесса обеспечения качества программного обеспечения. Их применение позволяет обнаружить ошибки на ранних этапах разработки, повысить надежность кода и улучшить взаимодействие внутри команды разработчиков.

2.1 Инспекция кода

Инспекция — это формализованный процесс анализа кода группой специалистов. Задача инспекции — обнаружение ошибок, а не их исправление. Основные этапы инспекции:

1. Подготовка. Координатор раздает участникам листинг программы и спецификацию проекта для предварительного изучения.

2. Заседание.

- Разработчик объясняет логику работы программы, а остальные участники задают вопросы и анализируют код.

- Используются контрольные списки распространенных ошибок.

3. Итоги. Все найденные ошибки фиксируются, после чего разработчик исправляет их самостоятельно.

Преимущества:

- Выявление до 70% ошибок, связанных с проектированием и кодированием.

- Обучение команды через анализ чужих ошибок и стилей программирования.

- Повышение командного духа и качества взаимодействия.

Недостатки: - Требует времени и ресурсов.

- Меньшая эффективность при поиске высокоуровневых ошибок (например, на этапе анализа требований).

2.2 Сквозной просмотр кода (Walkthrough)

Сквозной просмотр — это менее формализованный процесс, чем инспекция. Группа участников (обычно 3-5 человек) "проигрывает"логику программы, используя тестовые данные. Основные этапы:

1. Подготовка. Участники получают материалы для изучения. Тестировщик готовит набор тестовых данных.

2. Проведение.

- Тестировщик вводит тестовые данные, а участники вручную отслеживают изменения состояния программы.

- Задаются вопросы программисту, выявляются ошибки в логике и проектировании.

3. Итоги. Ошибки фиксируются, а разработчик вносит исправления.

Преимущества:

- Участники лучше понимают логику программы.

- Выявляются не только ошибки, но и слабые места программы.

Недостатки:

- Ограниченная скорость проверки из-за ручного анализа.

- Требуется участие нескольких человек, что увеличивает затраты.

2.3 Проверка за столом (Desk Checking)

Этот метод представляет собой индивидуальный анализ кода программистом. Разработчик самостоятельно вычитывает код, проверяет его с помощью контрольного списка или прогоняет тестовые данные вручную.

Преимущества:

- Простота и доступность.

- Не требует организации группы.

Недостатки:

- Низкая эффективность из-за субъективности и ограниченности одного человека.

- Нарушение принципа тестирования, согласно которому программист не должен проверять собственный код.

2.4 Важность использования контрольных списков

Контрольные списки — это инструмент, облегчающий выявление ошибок. Они включают типовые ошибки, которые часто встречаются в коде. Например:

- Обращения к неинициализированным переменным.

- Выход индексов за пределы массива.
- Неправильное использование арифметики целых чисел.
- Ошибки в логике циклов или ветвлений.
- Несоответствие типов данных при передаче параметров между модулями.

Использование таких списков позволяет систематизировать процесс анализа и повысить его эффективность.

2.5 Рецензирование

Рецензирование — это процесс оценки программного кода другими программистами. Цель — получение обратной связи о стиле программирования, удобстве сопровождения и качестве кода. Участники оценивают программы по заранее подготовленным критериям и предоставляют свои замечания.

Преимущества:
- Объективная оценка кода.
- Повышение квалификации программистов через анализ чужих решений.

Недостатки:
- Не направлено на выявление ошибок.
- Требует участия значительного числа людей для обеспечения анонимности.

3 Технология инспекции кода

3.1 Состав группы

Заседание происходило в следущем составе:
- секретарь Чурова Софья, исполняющая роль координатора;
- специалист по тестированию Брезгина Ольга;
- программист Яшнова Дарья, она же проектировщик программы.

3.2 Код программы

Программа должна реализовывать разрезание графа, проверку принадлежности вершин к одной компоненте связности.

Дан неориентированный граф. Над ним в заданном порядке производят операции следующих двух типов:

del — разрезать граф, то есть удалить из него ребро;

ask — проверить, лежат ли две вершины графа в одной компоненте связности.

Известно, что после выполнения всех операций типа del рёбер в графе не осталось. Найдите результат выполнения каждой из операций типа ask.

Входные данные

Первая строка содержит три целых числа, разделённые пробелами — количество вершин графа n, количество рёбер m и количество операций k (1<=n<=250, 0<=m<=500, m<=k<=1000).

Следующие m строк задают рёбра графа; i-я из этих строк содержит два числа ui и vi (1<=ui, vi<=n), разделённые пробелами — номера концов i-го ребра. Вершины нумеруются с единицы; граф не содержит петель и кратных рёбер.

Далее следуют k строк, описывающих операции. Операция типа del задаётся строкой "del u v"(1<=u,v<=n), которая означает, что из графа удаляют ребро между вершинами u и v. Операция типа ask задаётся строкой "ask u v"(1<=u,v<=n). Гарантируется, что каждое ребро графа встретится в операциях типа del ровно один раз.

Входное значение n	Входное значение m	Входное значение k	Описание e (ребра)	Описание q (запросы)	Ожидаемый результат
5	4	3	(1,2), (2,3), (3,4), (4,5)	ask 1 3, del 3 4, ask 4 5	YES, YES останов программы
6	3	2	(1,2), (2,3), (4,5)	ask 1 3, ask 4 6	YES, NO останов программы
4	2	2	(1,2), (3,4)	ask 1 4, del 1 2	NO останов программы
7	7	3	(1,2), (2,3), (3,4), (4,5), (5,6), (6,7), (1,7)	ask 1 4, del 6 7, ask 1 7	YES, YES останов программы
Число < 0	Число < 0	Число < 0	Некорректные значения для ребер	Некорректные запросы	Ошибка ввода

Рис. 1: Спецификация

Выходные данные

Для каждой операции ask выведите на отдельной строке слово "YES если две указанные вершины лежат в одной компоненте связности, и "NO"в противном случае.

На рис.1 представлена спецификация программы.

```
#include <iostream>
#include <vector>
#include <map>

using namespace std;

vector<int> p;
vector<int> siz;
vector<int> points;
int n, m;

void init() {
    p.resize(n);
    points.resize(n);
    siz.resize(n);
    for (int i = 0; i < n; ++i) {
        p[i] = i;
        siz[i] = 1;
        points[i] = 0;
    }
}

int getHead(int i) {
    while (p[i] != i) {
        i = p[i];
    }

    return i;
}
int get(int x, int y) {
    return getHead(x) == getHead(y);
}

void join(int i, int j) {
    int head_i = getHead(i);
    int head_j = getHead(j);
    if (head_i == head_j) {
        return;
    }

    if (siz[head_i] < siz[head_j]) {
        p[head_i] = head_j;
```

```
43          points[head_i] = points[head_i] - points[head_j];
44          siz[head_j] += siz[head_i];
45      }
46      else {
47          p[head_j] = head_i;
48          points[head_j] = points[head_j] - points[head_i];
49          siz[head_i] += siz[head_j];
50      }
51  }
52
53  int main() {
54      int k;
55      cin >> n >> m >> k;
56      init();
57      vector<pair<int, int>> e(m);
58      for (auto& i : e)
59      {
60          cin >> i.first >> i.second;
61          i.first--;
62          i.second--;
63      }
64      vector<pair<int, pair<int, int>>> q(k);
65      map<pair<int, int>, bool> mp;
66      for (int i = 0; i < k; i++)
67      {
68          string s;
69          cin >> s;
70          if (s == "ask") {
71              int x, y;
72              cin >> x >> y;
73              x--, y--;
74              q[i] = make_pair(0, make_pair(x, y));
75          }
76          else
77          {
78              int x, y;
79              cin >> x >> y;
80              x--, y--;
81              q[i] = make_pair(1, make_pair(x, y));
82              if (x > y)swap(x, y);
83              mp[make_pair(x, y)] = 1;
84          }
85      }
86
87      for (int i = 0; i < m; i++) {
88          if (e[i].first > e[i].second)swap(e[i].first, e[i].second);
89          if (mp.find(e[i]) == mp.end()) join(e[i].first, e[i].second);
90      }
91
92      vector<string> s;
93
94      for (int i = k - 1; i >= 0; i--) {
95          if (q[i].first == 1) {
96              join(q[i].second.first, q[i].second.second);
97          }
98          else {
99              if (get(q[i].second.first, q[i].second.second)) {
100                 s.push_back("YES\n");
101             }
102             else s.push_back("NO\n");
```

```
103             }
104         }
105 
106         reverse(s.begin(), s.end());
107 
108         for (auto i : s)cout << i;
109 
110         return 0;
111     }
```

4 Список вопросов и ответов

1. Используются ли переменные с неинициализированными значениями?

   Ответ: Да, n, m, k.

2. Выходят ли индексы за границы массива?

   Ответ: Да.

   Комментарий: В коде используется декрементация индексов ‘x–‘ и ‘y–‘ для приведения их к нумерации с нуля. Это может привести к выходу за границы, если входные значения ‘x‘ или ‘y‘ равны 0.

3. Используются ли нецелочисленные индексы?

   Ответ: Нет.

4. Вычисляются ли адреса битовых строк? Передаются ли битовые строки в качестве аргументов?

   Ответ: Нет.

5. Участвуют ли в вычислениях неарифметические переменные?

   Ответ: Нет.

6. Выполняются ли вычисления с использованием данных разного типа?

   Ответ: Нет.

7. Выполняются ли вычисления с переменными разной длины?

   Ответ: Нет, все переменные имеют одинаковую длину (целочисленные).

8. Присваиваются ли переменным значения типа данных большего размера?

   Ответ: Нет, присвоения данных большего размера не происходит.

9. Возможны ли переполнение или потеря точности в процессе промежуточных вычислений?

   Ответ: Нет.

10. Возможно ли деление на нуль?

    Ответ: Нет.

11. Критичны ли погрешности, возникающие из-за использования двоичных представлений чисел при вычислениях?

    Ответ: Нет, так как вычисления производятся только с целыми числами, где двоичные представления не влияют на точность.

12. Корректно ли используются атрибуты памяти, адресуемой с помощью ссылок и указателей?

    Ответ: Код не использует явных указателей.

13. Согласуются ли описания структуры данных в разных процедурах?

    Ответ: Да.

14. Не выходят ли значения переменных за логически обоснованные пределы?

    Ответ: Выходят.

    Комментарий: Возможен выход индексов за пределы массива, если значения ‘x‘ и ‘y‘ из входных данных не соответствуют ожидаемым диапазонам (1 <= x, y <= n).

15. Понятны ли приоритеты операций?

    Ответ: Приоритеты операций понятны.

16. Существуют ли ошибки завышения или занижения значения индекса на единицу при индексации массивов?

    Ответ: Да.

    Комментарий: Возможны ошибки, связанные с декрементацией индексов (‘x- -‘, ‘y- -‘). Если входные данные содержат минимальные значения (например, 1), после декрементации они становятся 0, что корректно для нумерации с нуля. Однако, если входные данные уже нумеруются с нуля, это приведет к ошибке.

17. Соблюдены ли правила наследования объектов?

    Ответ: Наследование в коде отсутствует.

18. Все ли переменные объявлены?

    Ответ: Да, все переменные объявлены.

19. Понятны ли последствия использования атрибутов по умолчанию?

    Ответ: Атрибуты по умолчанию в коде не используются.

20. Есть ли попытки сравнения несопоставимых переменных?

    Ответ: Нет, все сравнения выполняются между переменными совместимых типов.

21. Есть ли попытки сравнения переменных разного типа?

    Ответ: Нет, все сравнения выполняются между переменными одного типа (целочисленные индексы и флаги).

22. Правильно ли инициализированы массивы и строки?

    Ответ: Да.

23. Корректно ли используются операторы сравнения?

    Ответ: Да.

24. Правильно ли определены размеры, типы и классы памяти?

    Ответ: Да.

25. Корректно ли используются булевы выражения?

    Ответ: Да, булевы выражения используются корректно.

26. Согласуется ли инициализация с классом памяти?

    Ответ: Да.

27. Корректно ли используются результаты сравнения в булевых выражениях?

    Ответ: Да.

28. Имеются ли переменные с похожими именами?

    Ответ: Да.

    Комментарий: Переменные ‘x‘, ‘y‘, ‘n‘, ‘m‘, ‘k‘ имеют короткие имена, что может привести к путанице.

29. Корректно ли сравниваются дробные величины?

    Ответ: В коде отсутствуют дробные величины.

30. Понятны ли приоритеты операций?

    Ответ: Да, приоритеты операций понятны.

31. Понятен ли способ разбора булевых выражений компилятором?

    Ответ: Да.

32. Может ли значение индекса в переключателе превысить число переходов?

    Ответ: Нет.

33. Правильно ли заданы атрибуты файлов?

    Ответ: Код не работает с файлами.

34. Будет ли завершен каждый цикл?

    Ответ: Да, все циклы в коде имеют конечные условия.

35. Будет ли завершена программа?

    Ответ: Да, программа завершится.

36. Есть ли цикл, который может не выполниться из-за входных условий?

    Ответ: Да.

    Комментарий: Циклы могут не выполниться, если их начальные условия не соответствуют их диапазону. Цикл ‘for (int i = k - 1; i >= 0; i–)‘ не выполнится, если ‘k <= 0‘.

37. Корректны ли инструкции OPEN?

    Ответ: Инструкции ‘OPEN‘ отсутствуют, так как код не работает с файлами.

38. Согласуются ли спецификации формата с инструкциями ввода-вывода?

    Ответ: Да. Код использует стандартный ввод и вывод через ‘cin‘ и ‘cout‘, спецификации формата не требуются.

39. Соответствует ли размер буфера размеру записи?

    Ответ: Код не применяет буферы, так как использует стандартные потоки ввода-вывода.

40. Корректны ли возможные погружения в цикл?

    Ответ: Погружения в цикл корректны.

41. Имеются ли ошибки диапазона, приводящие к отклонению количества итераций от нужного значения?

    Ответ: Да.

    Комментарий: Возможны ошибки диапазона, если входные данные некорректны. Если ‘k‘ меньше 0, то цикл ‘for (int i = k - 1; i >= 0; i–)‘ не будет выполнен.

42. Открываются ли файлы перед обращением к ним?

    Ответ: Файлы в коде не используются.

43. Закрываются ли файлы после обращения к ним?

    Ответ: Файлы в коде не используются.

44. Корректно ли обрабатываются признаки конца файла?

    Ответ: Файлы в коде не используются.

45. Обрабатываются ли ошибки ввода-вывода?

    Ответ: Нет.

    Комментарий: Ошибки ввода-вывода явно не обрабатываются. Если ввод некорректен, это может привести к ошибкам выполнения.

46. Присутствуют ли в выходной информации орфографические или грамматические ошибки?

    Ответ: Нет.

47. Существуют ли точки ветвления, в которых не учтены все возможные условия?

    Ответ:Нет, все точки ветвления учитывают возможные условия.

48. Совпадает ли число формальных параметров с числом аргументов?

    Ответ: Да.

49. Совпадает ли тип возвращаемого значения с типом, объявленным в сигнатуре функций?

    Ответ: Нет.

    Комментарий: в функции get возвращаевым типом должен быть int, но возвращается выражение типа bool.

50. Совпадают ли атрибуты параметров и аргументов?

    Ответ: Да.

51. Совпадают ли единицы измерения параметров и аргументов?

    Ответ: Да.

52. Совпадает ли число аргументов, передаваемых вызываемым модулям, с числом ожидаемых параметров?

    Ответ: Да.

53. Совпадают ли атрибуты аргументов, передаваемых вызываемым модулям, с атрибутами ожидаемых параметров?

    Ответ: Да.

54. Совпадают ли единицы измерения аргументов, передаваемых вызываемым модулям, с единицами измерения ожидаемых параметров?

    Ответ: Код не использует физические единицы измерения.

55. Правильно ли заданы количество, атрибуты и порядок следования аргументов при вызове встроенных функций?

    Ответ: Да, встроенные функции ‘cin‘, ‘cout‘, ‘reverse‘, и ‘map.find‘ используются корректно.

56. Имеются ли ссылки на параметры, не связанные с текущей точкой входа?

    Ответ: Нет.

57. Делаются ли в подпрограмме попытки изменить входные аргументы?

    Ответ: Нет.

58. Согласуются ли определения глобальных переменных в разных модулях?

    Ответ: Модули отсутствуют.

59. Передаются ли константы в качестве аргументов?

    Ответ: Нет.

60. Есть ли в таблице перекрестных ссылок переменные, на которые нет ссылок?

    Ответ: Все переменные, объявленные в коде, используются, кроме массива Points.

    Комментарий: Можно полностью убрать массив points из кода, так как он не используется.

61. Совпадает ли список атрибутов с тем, который следовало ожидать?

    Ответ: Да, все переменные и параметры имеют ожидаемые типы и атрибуты.

62. Выдаются ли какие-нибудь предупреждения или информационные сообщения при компиляции?

    Ответ: Нет.

63. Осуществляется ли проверка корректности входных данных?

    Ответ: Проверка входных данных отсутствует. Значения ‘n‘, ‘m‘, ‘k‘ предполагаются корректными, но не проверяются явно.

5 Исправленный код

По результатам заседания в код были внесены следующие изменения:

- Из кода удален вектор points и все операции, связанные с ним.
- Изменены имена переменных и функций на более понятные и читаемые в коде.
- Тип выражения, возвращаемого функцией areConnected был заменен на bool, чтобы не происходило неявной конвертации.

Комментарии связанные с ошибками в цикле «for (int i = k - 1; i >= 0; i–)», не были исправлены, так как считается, что ошибок во входных данных нет, так как они проверяются в другой части программы.

```
1
2  #include <iostream>
3  #include <vector>
4  #include <map>
5  #include <algorithm>
6
7  using namespace std;
8
9
10 vector<int> parent;// изменено имя массива для хранения родителя элемента
11 vector<int> sizeGroup;// изменено имя массива для хранения размера множества
12 int numNodes, numEdges;
13
14 void initializeUnionFind() {
15     parent.resize(numNodes);
16     sizeGroup.resize(numNodes);
17     for (int i = 0; i < numNodes; ++i) {
18         parent[i] = i;
19         sizeGroup[i] = 1;
20     }
21 }
22
23 int findRoot(int node) {
24     while (parent[node] != node) {
25         node = parent[node];
26     }
27     return node;
28 }
29
30 bool areConnected(int node1, int node2) {
31 //тип возвращаемого значения изменен
32     return findRoot(node1) == findRoot(node2);
33 }
34
35 void unionGroups(int node1, int node2) {
36     int root1 = findRoot(node1);
37     int root2 = findRoot(node2);
38     if (root1 == root2) {
39         return;
40     }
41
42     if (sizeGroup[root1] < sizeGroup[root2]) {
43        parent[root1] = root2; //исключен массив points
44
45         sizeGroup[root2] += sizeGroup[root1];
46     } else {
47         parent[root2] = root1;
48         sizeGroup[root1] += sizeGroup[root2];
49     }
50 }
51
52 int main() {
53     int numQueries;
54     cin >> numNodes >> numEdges >> numQueries;
55     if(numNodes<=0 || numEdges<=0 || numQueries<=0){
56         return 0;
57     } \\добавлена проверка входных данных
58     initializeUnionFind();
59
60     vector<pair<int, int>> edges(numEdges);
```

```
    for (auto& edge : edges) {
        cin >> edge.first >> edge.second;
        edge.first--;
        edge.second--;
    }

    vector<pair<int, pair<int, int>>> queries(numQueries);
    map<pair<int, int>, bool> removedEdges;
    for (int i = 0; i < numQueries; i++) {
        string queryType;
        cin >> queryType;

        if (queryType == "ask") {
            int node1, node2;
            cin >> node1 >> node2;
            node1--, node2--;
            queries[i] = make_pair(0, make_pair(node1, node2));
        } else {
            int node1, node2;
            cin >> node1 >> node2;
            node1--, node2--;
            if (node1 > node2) swap(node1, node2);
            queries[i] = make_pair(1, make_pair(node1, node2));
            removedEdges[make_pair(node1, node2)] = true;
        }
    }

    for (const auto& edge : edges) {
        if (edge.first > edge.second) swap(edge.first, edge.second);
        if (removedEdges.find(edge) == removedEdges.end()) {
            unionGroups(edge.first, edge.second);
        }
    }

    vector<string> results;
    for (int i = numQueries - 1; i >= 0; i--) {
        if (queries[i].first == 1) {
            int node1 = queries[i].second.first;
            int node2 = queries[i].second.second;
            unionGroups(node1, node2);
        } else {
            int node1 = queries[i].second.first;
            int node2 = queries[i].second.second;
            if (areConnected(node1, node2)) {
                results.push_back("YES\n");
            } else {
                results.push_back("NO\n");
            }
        }
    }

    reverse(results.begin(), results.end());
    for (const auto& result : results) {
        cout << result;
    }

    return 0;
}
```

6 Заключение

При работе с кодом был приобретён опыт ручного тестирования методом инспекции, анализа программы в группе и подробного разбора с ответами на ключевые вопросы о её логике и реализации. Некоторые комментарии эксперта я приняла к сведению и внесла исправления в код.

После работы группы были исправлены недочеты:

1. Отсутствие проверок корректности входных данных
2. Неудачные названия переменных
3. Использование переменной, не влияющей на конечный результат.
4. Несовпадение типа возвращаемого значения с типом функции.

Список литературы

[1] Майерс, Г. Искусство тестирования программ. – Санкт-Петербург: Диалектика, 2012. – С. 272.