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

«Статический анализ кода»

Исполнитель: _____ Яшнова Дарья Михайловна
группа 5130201/20002

Руководитель: _____ Курочкин Михаил Александрович

«___»_____ 2025г

Санкт-Петербург, 2025

Содержание

Введение

Цель работы - проанализировать код программы «Разрезание графа, проверка принадлежности вершин к одной компоненте связности» с целью улучшения качества кода. Необходимо изучить инструмент PVS-Studio и провести статический анализ кода при помощи PVS-Studio.

1 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.

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.

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

Программа написана на языке C++ 14, в среде Visual Studio 2022.

```
#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);
```

```
16      for (int i = 0; i < n; ++i) {
17          p[i] = i;
18          siz[i] = 1;
19          points[i] = 0;
20      }
21  }
22
23  int getHead(int i) {
24      while (p[i] != i) {
25          i = p[i];
26      }
27
28      return i;
29  }
30  int get(int x, int y) {
31      return getHead(x) == getHead(y);
32  }
33
34  void join(int i, int j) {
35      int head_i = getHead(i);
36      int head_j = getHead(j);
37      if (head_i == head_j) {
38          return;
39      }
40
41      if (siz[head_i] < siz[head_j]) {
42          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 }
```

2 Статический анализ кода приложения

PVS-Studio — это коммерческий инструмент для статического анализа исходного кода, разработанный компанией "Program Verification Systems". Он помогает разработчикам находить ошибки, потенциальные уязвимости и дефекты в коде на языках C, C++, C#, Java и Python. Этот анализатор активно используется в разработке как для десктопных, так и для серверных приложений, и особенно полезен для крупных проектов, где ручной поиск ошибок становится слишком трудоёмким.

2.1 Основные возможности PVS-Studio

PVS-Studio предоставляет широкий набор функций для анализа кода, которые помогают разработчикам улучшить качество программного обеспечения. Вот основные возможности, которые предлагает этот инструмент:

- Анализ и поиск ошибок

  PVS-Studio позволяет выявлять широкий спектр ошибок, включая:

  - Опечатки в коде: Например, случайное использование = вместо ==, или ошибки в логических выражениях.

  - Неправильное использование переменных: Использование неинициализированных переменных, утечки памяти, доступ к освобождённой памяти.

  - Ошибки многопоточности: Проблемы с синхронизацией, гонки данных, некорректное использование потоков.

  - Неопределённое поведение: Код, который может вести себя непредсказуемо в зависимости от компилятора или платформы.

  - Потенциальные уязвимости: Выявление мест в коде, которые могут быть эксплуатированы злоумышленниками (например, SQL-инъекции, переполнение буфера).

- Анализ многопоточного кода

- Удобные отчёты

  PVS-Studio генерирует отчёты с подробным описанием найденных ошибок:

  - Категоризация дефектов: ошибки разделяются на категории (например, критические, предупреждения, улучшения).

  - Описание проблем: инструмент предоставляет объяснение каждой ошибки, примеры неправильного и правильного кода.

  - Фильтрация: возможность фильтровать предупреждения по важности, типу ошибок или файлам.

  - Форматы отчётов: поддержка различных форматов (HTML, XML, JSON), что удобно для интеграции с внешними системами.

- Проверка соответствия стандартам

2.2 Виды ошибок

2.2.1 Ошибки, связанные с опечатками

Опечатки — одна из наиболее распространённых причин ошибок. PVS-Studio умеет находить такие проблемы, как:

– Использование оператора `=` вместо `==` в условиях.
– Ошибки в именах переменных или функций.
– Логические ошибки в выражениях.

2.2.2 Использование неинициализированных переменных

Инструмент обнаруживает случаи, когда переменная используется до её инициализации.

2.2.3 Утечки памяти

PVS-Studio помогает выявлять утечки памяти, возникающие из-за забытых вызовов `delete` или `free`.

2.2.4 Доступ к невалидной памяти

Инструмент находит попытки обращения к освобождённой памяти, выход за пределы массива и другие подобные ошибки.

2.2.5 Ошибки многопоточности

PVS-Studio помогает выявлять проблемы многопоточности:

– Гонки данных (data races).
– Deadlock'и.
– Неправильное использование примитивов синхронизации.

2.2.6 Логические ошибки

Инструмент выявляет логические ошибки, включая:

– Условия, которые всегда истинны или ложны.
– Некорректные циклы.
– Ошибки в выражениях.

2.2.7 Неопределённое поведение

Инструмент выявляет код, который может вести себя по-разному в зависимости от компилятора, платформы или окружения.

2.2.8 Неиспользуемый или избыточный код

Инструмент находит:

- Неиспользуемые переменные, функции и параметры.
- Лишние вызовы функций.
- Код, который не влияет на результат выполнения программы.

Анализатор содержит 5 видов диагностических правил:

- General (GA) - диагностики общего плана. Основной набор диагностических правил PVS-Studio.
- Optimization (OP) - диагностики микрооптимизации. Указания по повышению эффективности и безопасности кода.
- 64-bit (64) - диагностики, позволяющие выявлять специфические ошибки, связанные с разработкой 64-битных приложений, а также переносом кода с 32-битной на 64-битную платформу.
- Customers' Specific (CS) - узкоспециализированные диагностики, разработанные по просьбам пользователей. По умолчанию этот набор диагностик отключен.
- MISRA - диагностики, разработанные в соответствии со стандартом MISRA (Motor Industry Software Reliability Association). По умолчанию этот набор диагностик отключен.

2.3 Уровни предупреждений и наборы диагностических правил

PVS-Studio разделяет все предупреждения по 3 уровням достоверности: High, Medium и Low. Также некоторые сообщения относятся к особой категории Fails. Рассмотрим эти уровни подробнее:

* High (1) - включает в себя предупреждения с максимальным уровнем достоверности. Такие предупреждения часто указывают на ошибки, требующие немедленного исправления.
* Medium (2) - содержит менее достоверные предупреждения, на которые все же стоит обратить пристальное внимание.
* Low (3) - предупреждения с минимальным уровнем достоверности, указывающие на несущественные неточности в коде. Среди таких предупреждений обычно велик процент ложных срабатываний.
* Fails - внутренние сообщения анализатора, информирующие о возникновении каких-то проблем при работе. В группу Fails попадают сообщения об ошибках анализатора (например, сообщения с кодами V001, V003 и т.п.), а также любой необработанный вывод вспомогательных программ, используемых самим анализатором во время анализа (препроцессор, командный процессор cmd), выдаваемый ими в stdout/stderr. Например, в группу Fails может попасть сообщение препроцессора об ошибках препроцессирования, доступа к файлам (файл не существует или заблокирован антивирусом) и т.п.

2.4 Запуск PVS-Studio

Перед запуском необходимо настроить плагин на обнаружение ошибок. Есть несколько опций для отображения выявляемых ошибок. На рис.2 представлена конфигурация, которая максимально полно покроет возможные случаи ошибки в программе.

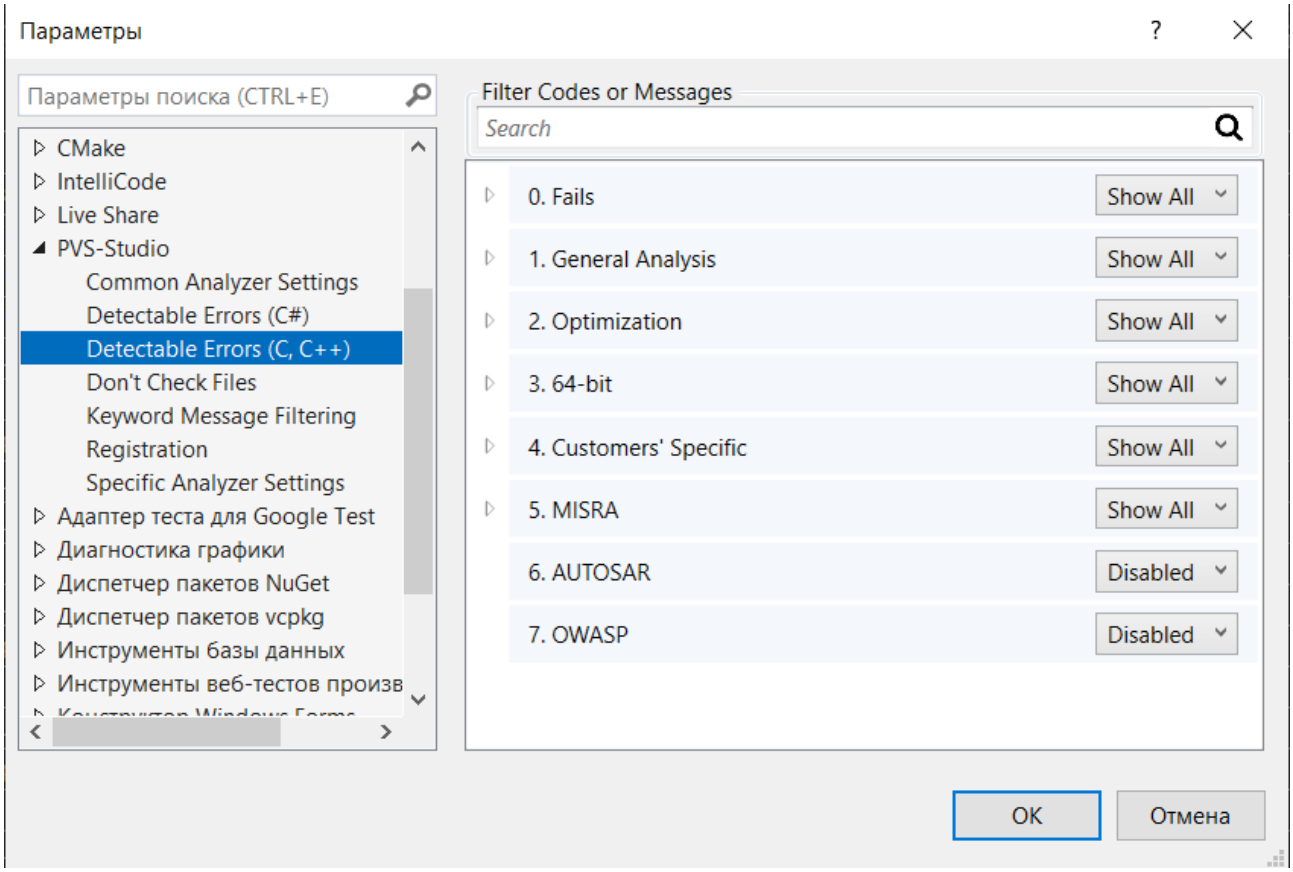


Рис. 2: Конфигурация вывода ошибок

Для запуска необходимо установить плагин PVS-Studio в среде VisualStudio и выбрать файл с кодом для анализа.

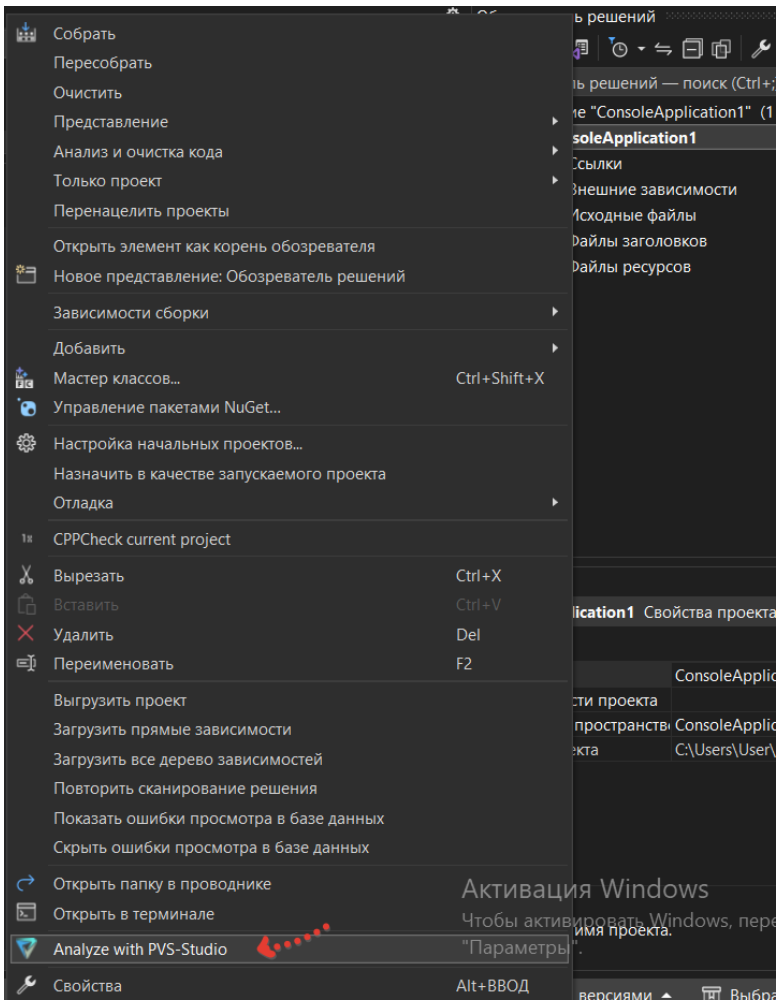


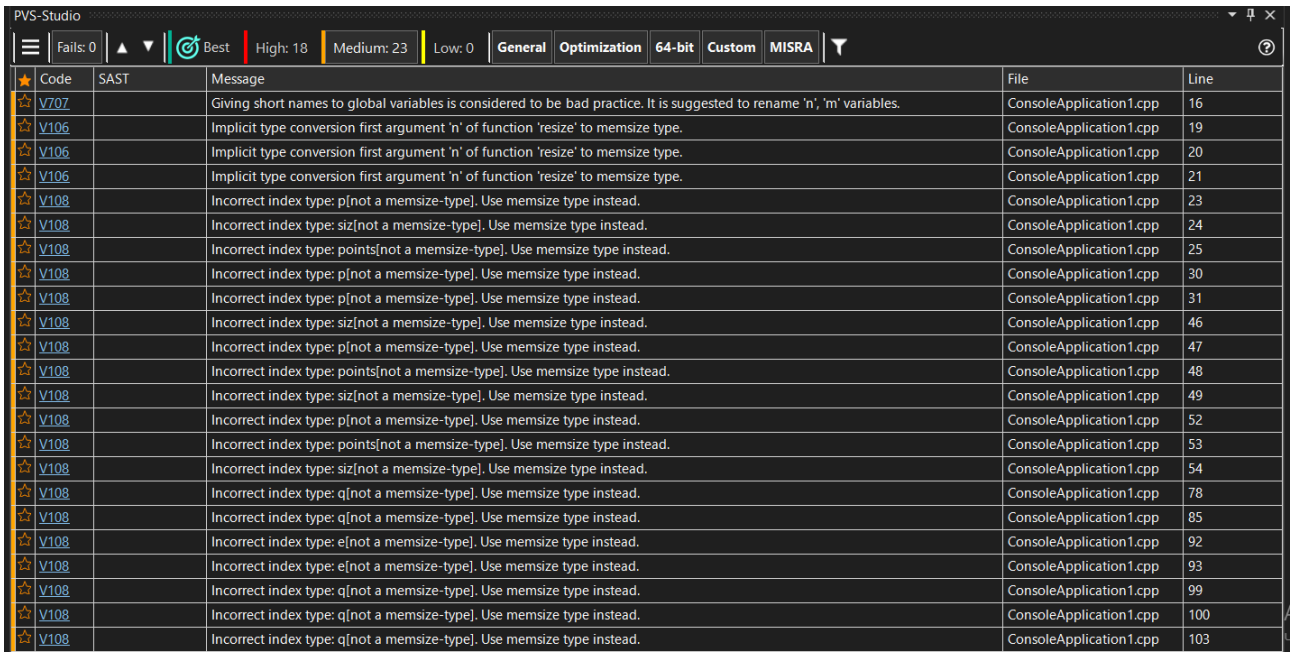
Рис. 3: Запуск инструмента PVS-Studio

При помощи интерфейса можно фильтровать выявленные предупреждения по уровню и набору правил.

Fails: 0 | Best | High: 18 | Medium: 23 | Low: 0 | General | Optimization | 64-bit | Custom | MISRA

Code	SAST	Message	File	Line
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	11
V2506	MISRA-CPP-6-...	The 'return' statement should be the last statement of a function.	ConsoleApplication1.cpp(...)	39
V2528	MISRA-CPP-5-...	The comma operator should not be used.	ConsoleApplication1.cpp	77
V2528	MISRA-CPP-5-...	The comma operator should not be used.	ConsoleApplication1.cpp	84
V2507	MISRA-CPP-6-...	The body of the 'if' statement should be enclosed in braces.	ConsoleApplication1.cpp	86
V2507	MISRA-CPP-6-...	The body of the 'if' statement should be enclosed in braces.	ConsoleApplication1.cpp	92
V2507	MISRA-CPP-6-...	The body of the 'if' statement should be enclosed in braces.	ConsoleApplication1.cpp	93
V2507	MISRA-CPP-6-...	The body of the 'else' branch should be enclosed in braces.	ConsoleApplication1.cpp	106
V836		Expression's value is copied at the 'i' variable declaration. The variable is never modified. Consider declaring it as a reference.	ConsoleApplication1.cpp	112
V2507	MISRA-CPP-6-...	The body of the 'for' statement should be enclosed in braces.	ConsoleApplication1.cpp	112
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	13
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	16
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	39
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	14
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	15
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	18
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	29
V2575	MISRA-CPP-7-...	The global namespace should only contain 'main', namespace declarations and extern 'C' declaration.	ConsoleApplication1.cpp	36

Рис. 4: Предупреждения уровня High



Code	SAST	Message	File	Line
V707		Giving short names to global variables is considered to be bad practice. It is suggested to rename 'n', 'm' variables.	ConsoleApplication1.cpp	16
V106		Implicit type conversion first argument 'n' of function 'resize' to memsize type.	ConsoleApplication1.cpp	19
V106		Implicit type conversion first argument 'n' of function 'resize' to memsize type.	ConsoleApplication1.cpp	20
V106		Implicit type conversion first argument 'n' of function 'resize' to memsize type.	ConsoleApplication1.cpp	21
V108		Incorrect index type: p[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	23
V108		Incorrect index type: siz[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	24
V108		Incorrect index type: points[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	25
V108		Incorrect index type: p[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	30
V108		Incorrect index type: p[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	31
V108		Incorrect index type: siz[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	46
V108		Incorrect index type: p[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	47
V108		Incorrect index type: points[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	48
V108		Incorrect index type: siz[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	49
V108		Incorrect index type: p[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	52
V108		Incorrect index type: points[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	53
V108		Incorrect index type: siz[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	54
V108		Incorrect index type: q[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	78
V108		Incorrect index type: q[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	85
V108		Incorrect index type: e[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	92
V108		Incorrect index type: e[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	93
V108		Incorrect index type: q[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	99
V108		Incorrect index type: q[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	100
V108		Incorrect index type: q[not a memsize-type]. Use memsize type instead.	ConsoleApplication1.cpp	103

Рис. 5: Предупреждения уровня Medium

2.5 Анализ выявленных предупреждений

Основные проблемы уровня medium:

1. Проблемы с читаемостью и поддерживаемостью кода (V707):

   - Использование коротких имён переменных, особенно для глобальных, делает код менее понятным и увеличивает вероятность ошибок.

2. Неявные преобразования типов (V106):

   - Неявные преобразования могут быть источником багов, особенно если они приводят к потере данных или некорректным результатам.

   - Рекомендуется использовать явное приведение типов, чтобы избежать подобных ошибок.

3. Неправильный тип индексации (V108):

   - Использование неподходящего типа для индексации массивов или контейнеров может привести к неопределённому поведению, особенно если индекс выходит за пределы допустимого диапазона.

   - Использование std::size_t для индексации — это стандартная практика, которая повышает надёжность кода.

Основные проблемы уровня hard

1. Использование глобального пространства имён (V2575):

   - Код нарушает правила хорошего стиля и может привести к конфликтам имён.

2. Пропуск фигурных скобок (V2507):

   - Это снижает читаемость и может привести к ошибкам при добавлении новых строк кода.

3. Использование оператора запятая (V2528):

   - Лучше разделять выражения на отдельные строки

3 Исправленный код

Некоторые предупреждения были исправлены:
- Из кода удален вектор points и все операции, связанные с ним.
- Изменены имена переменных и функций на более понятные и читаемые в коде.
- Запятые между операторами были удалены, операторы перенесены на разные строки.
- Были добавлены пропущенные фигурные скобки.

```

// This is a personal academic project. Dear PVS-Studio, please check it.

// PVS-Studio Static Code Analyzer for C, C++, C#, and Java: https://pvs-
    studio.com


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

using namespace std;
// удален неиспользуемый массив
vector<int> parent;// изменено имя массива для хранения родителя элемента
vector<int> sizeGroup;// изменено имя массива для хранения размера множества
int numNodes, numEdges;

void init() {
        parent.resize(numNodes);
    sizeGroup.resize(numNodes);
    for (int i = 0; i < numNodes; ++i) {
        parent[i] = i;
        sizeGroup[i] = 1;
    }
}

int getHead(int i) {
    while (parent[i] != i) {
        i = parent[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 (sizeGroup[head_i] < sizeGroup[head_j]) {
        parent[head_i] = head_j;
        sizeGroup[head_j] += sizeGroup[head_i];
    }
    else {
        parent[head_j] = head_i;
        sizeGroup[head_i] += sizeGroup[head_j];
    }
}
```

```
52
53  int main() {
54      int k;
55      cin >> numNodes >> numEdges >> k;
56      init();
57      vector<pair<int, int>> e(numEdges);
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;  for (int i = 0; i < k; i++)
66      {
67          string s;
68          cin >> s;
69          if (s == "ask") {
70              int x, y;
71              cin >> x >> y;
72              x-;
73              y-;// удалена запятая между операторами
74
75              q[i] = make_pair(0, make_pair(x, y));
76          }
77          else
78          {
79              int x, y;
80              cin >> x >> y;
81               x-;
82              y-;// удалена запятая между операторами
83              q[i] = make_pair(1, make_pair(x, y));
84
85             if (x > y) {
86                  swap(x, y);
87              }// добавлены фигурные скобки
88              mp[make_pair(x, y)] = 1;
89          }
90      }
91
92      for (int i = 0; i < numEdges; i++) {
93          if (e[i].first > e[i].second) {
94              swap(e[i].first, e[i].second);
95              }
96          }
97
98
99           if (mp.find(e[i]) == mp.end()) {
100
101
102             join(e[i].first, e[i].second);
103
104
105         }  // добавлены фигурные скобки
106
107     }
108
109     vector<string> s;
110
111     for (int i = k - 1; i >= 0; i--) {
```

```
112             if (q[i].first == 1) {
113                 join(q[i].second.first, q[i].second.second);
114             }
115             else {
116                 if (get(q[i].second.first, q[i].second.second)) {
117                     s.push_back("YES\n");
118                 }
119                 else s.push_back("NO\n");
120             }
121         }
122 
123         reverse(s.begin(), s.end());
124 
125         for (auto i : s)cout << i;
126 
127         return 0;
128     }
```

4 Заключение

В рамках главы был проведен статический анализ кода при помощи PVS-Studio, а также описана сама утилита. Использование PVS-Studio позволило, обнаружить ряд стилистических ошибок. PVS-Studio - это удобный инструмент, который легок в использовании. В ходе работы были найдены недочеты, которые не были найдены при инспекции кода. Однако, некоторые замечания учтенные при инспекции, не были учтены при статическом анализе. При инспекции кода был выявлен повторяющийся синтаксис в названиях функций, но при этом не было выявлено использование запятых между операторами и пропущенные фигурные скобки. Можно сделать вывод, что эти методы тестирования дополняют друг друга и эффективно использовать их совместно.

После анализа кода были исправлены недочеты:

1. Использование запятых между операторами.
2. Пропущенные фигурные скобки.
3. Использование переменной, не влияющей на конечный результат.
4. Неудачные названия переменных

Список литературы

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