МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
федеральное государственное бюджетное образовательное учреждение высшего образования
«Тольяттинский государственный университет»

Кафедра «Прикладная математика и информатика»
(наименование)

02.03.03 Математическое обеспечение и администрирование информационных систем
(код и наименование направления подготовки / специальности)

Мобильные и сетевые технологии
(направленность (профиль) / специализация)

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА (БАКАЛАВРСКАЯ РАБОТА)

на тему Анализ и построение на основе теории графов математической модели угроз безопасности

Обучающийся В. С. Фадеев _____
(Инициалы Фамилия) (личная подпись)

Руководитель к.т.н., доцент, Н. А. Сосина
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Консультант к.т.н., доцент, А.В. Егорова
(ученая степень (при наличии), ученое звание (при наличии), Инициалы Фамилия)

Тольятти 2025

Аннотация

Выпускная квалификационная работа посвящена разработке математической модели угроз безопасности на основе теории графов. Актуальность исследования обусловлена ростом количества и сложности информационных угроз, а также необходимостью формализации и количественного анализа уязвимостей в сложных информационных системах.

В первой главе работы изложено теоретическое обоснование выбора графов в качестве основы для моделирования угроз. Проведён анализ предметной области, описана структура организации, где проходила практика, и выделены ключевые элементы инфраструктуры, подверженные рискам. Также рассмотрены базовые типы графов.

Следующая вторая глава посвящена построению математической модели. Представлены методы оценки вероятностей реализации угроз, включая последовательные, параллельные и мостовые структуры. Введены формулы расчёта вероятности реализации угроз для выявления критических узлов системы. В рамках этой модели сформулированы критерии оценки наиболее уязвимых участков, влияющих на устойчивость всей системы.

Практическая часть выпускной квалификационной работы включает программную реализацию модели на языке C++ в среде Visual Studio 2022 с использованием Windows API. Программа реализует построение графа угроз, расчёт вероятностей на основе заданных параметров, определение критических узлов и визуализацию полученных данных.

Разработан также программный модуль, который позволяет формализовать и количественно оценивать риски информационной безопасности с помощью графового анализа. Этот инструмент даёт возможность специалистам по безопасности принимать обоснованные решения на основе представленного в работе математического аппарата и визуализированных структур угроз.

Abstract

This graduate thesis is devoted to the development of a mathematical model for security threats based on graph theory. The relevance of the research stems from the increasing number and complexity of information threats, as well as the need for formalization and quantitative analysis of vulnerabilities in complex information systems.

The first chapter of the thesis presents the theoretical justification for using graphs as the foundation for threat modeling. It includes an analysis of the subject area, a description of the organizational structure where the internship took place, and identifies key infrastructure elements exposed to risks. Additionally, fundamental types of graphs are examined.

The second chapter focuses on constructing the mathematical model. Methods for assessing threat realization probabilities are presented, including sequential, parallel, and bridge structures. Formulas for calculating threat probabilities are introduced, along with the use of partial derivatives to identify critical system nodes. Within this model, evaluation criteria are formulated to determine the most vulnerable segments affecting the overall system's resilience.

The practical part of the thesis involves the software implementation of the model in C++ using Visual Studio 2022 and Windows API. The program constructs a threat graph, calculates probabilities based on given parameters, identifies critical nodes, and visualizes the results.

The outcome of this work is a software module that enables the formalization and quantitative assessment of information security risks through graph analysis. This tool allows security professionals to make informed decisions based on precise mathematical methods and visualized threat structures.

Оглавление

Введение 5
Глава 1 Анализ предметной области 7
1.1Описание компании 7
1.2 Теория графов в задачах безопасности 9
1.3 Методы представления модели на основе графов 11
1.4 Алгоритмы обхода графа 13
1.5 Типы структур в модели на основе графов модели угроз 17
1.6 Математическая модель угроз безопасности 23
1.7 Оценка критических узлов системы 25
Глава 2 Проектирование программной реализации модели 28
2.1 Выбор средств разработки 28
2.2 Выбор языка программирования 30
2.3 Описание средств для решения поставленных задач 31
Глава 3 Реализация программы 34
3.1 Разработка программы 34
3.2 Анализ и моделирование угроз на тестовой инфраструктуре 40
Заключение 50
Список используемой литературы 51

Введение

Рост количества и сложности угроз информационной безопасности в современных вычислительных системах требует новых подходов к их анализу и моделированию. Особую значимость приобретает возможность формализовать поведение угроз, выявлять уязвимые участки инфраструктуры и оценивать возможные риски не на интуитивном, а на строго математическом уровне. С этой точки зрения теория графов представляет собой мощный инструмент, позволяющий построить точную модель угроз и визуализировать связи между элементами системы. Именно поэтому использование модели на основе графов в задачах информационной безопасности становится всё более актуальным, особенно в условиях возрастающей цифровизации и усложнения архитектур информационных систем.

Актуальность настоящей выпускной квалификационной работы заключается в необходимости разработки инструментов формального анализа угроз с использованием современных методов математического моделирования. Математическое описание угроз с помощью графов позволяет выявлять критические узлы, прослеживать возможные сценарии развития атак и рассчитывать вероятность реализации тех или иных рисков. Это особенно важно при построении адаптивных систем обеспечения информационной безопасности, которые должны реагировать на новые типы угроз в режиме реального времени.

Объектом исследования в данной работе выступают угрозы информационной безопасности, возникающие в сложных распределённых информационных системах.

Предметом исследования являются математические методы анализа и моделирования этих угроз на основе теории графов, включая построение графов состояний, вероятностное моделирование переходов и вычисление показателей риска.

Цель работы заключается в разработке математической модели угроз

безопасности на основе теории графов и её программной реализации с возможностью визуализации структуры угроз и расчёта ключевых показателей риска. Модель должна быть ориентирована на практическое применение в системах анализа уязвимостей и поддержки принятия решений в сфере информационной безопасности.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести анализ предметной области и выявить ключевые аспекты, требующие формализации;
- исследовать и обосновать применимость модели на основе графов модели к описанию угроз;
- построить математическую модель на основе графов, учитывающую вероятностные характеристики переходов между состояниями системы;
- выбрать средство разработки и язык программирования для дальнейшей реализации
- реализовать программное средство, позволяющее визуализировать угрозы, рассчитывать риски и определять критические узлы;
- провести тестирование модели на различных сценариях и оценить её применимость для анализа реальных и гипотетических угроз.

Глава 1 Анализ предметной области

1.1 Описание компании

Сфера деятельности площадки прохождения практики охватывает разработку программного обеспечения и информационных систем для последующей интеграции в электронную коммерцию.

Программное обеспечение, создаваемое компанией «ВорлдИнтерТех РУС», служит связующим звеном между компаниями и потребителями, предоставляя разнообразные сервисы.

«Разработчик программного обеспечения занимает функциональную позицию в организации, отвечая за создание ПО, основываясь на техническом задания. Этот отдел занимается составлением технического задания на основе требований заказчика и разработкой программного обеспечения в соответствии с утвержденным ТЗ [14]».

Организационная структура «ВорлдИнтерТех РУС» представлена на рисунке 1:

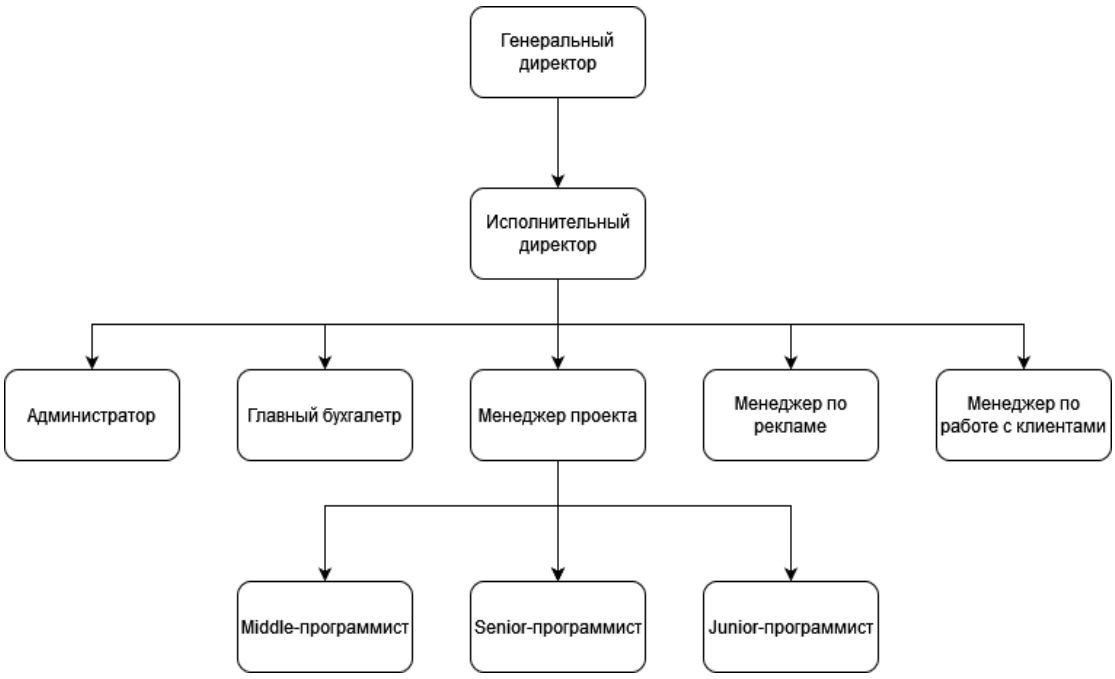


Рисунок 1 – Структурная схема базы «ВорлдИнтерТех РУС»

В подразделении отдела разработки – входят следующие сотрудники:

- менеджер проекта;
- senior-программист;
- middle-программист;
- junior-программист.

Функции сотрудников представлены на рисунке 2.

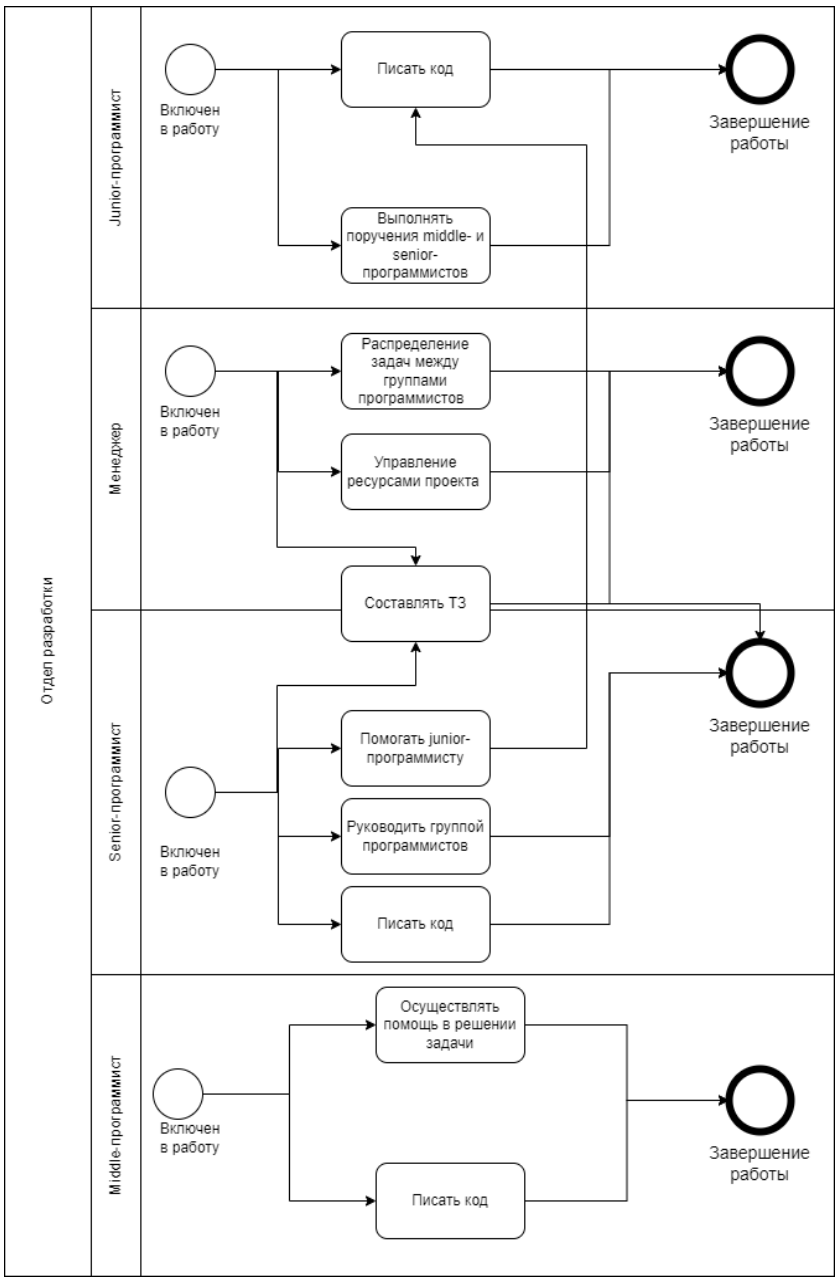


Рисунок 2 – Функции, выполняемые сотрудниками отдела разработок

Рассмотрим роли и функции сотрудников отдела разработки, которые представлены на рисунке 2.

Junior-программист. Его деятельность ограничена из-за недостатка опыта и знаний в области разработки программного обеспечения. Он занимается выполнением небольших задач по поддержке и улучшению уже существующих сервисов, а также написанием и выполнением тестов для ПО. Junior-программист может также брать на себя более сложные задания с целью повышения своей квалификации. Обычно он сильно зависит от более опытных коллег из-за своей неопытности.

Middle-программист. Это самостоятельный специалист, обладающий достаточным опытом в разработке программного обеспечения, что позволяет ему справляться с более сложными задачами. Middle-программист также курирует младших коллег и оказывает им помощь в решении возникающих проблем

Senior-программист. Это высококвалифицированный специалист с обширным опытом разработки, который хорошо разбирается в своей области. Он отвечает за исправление сложных ошибок, разработку новых сервисов и программных продуктов, а также проверку кода, написанного младшими программистами.

Менеджер проекта. Этот специалист управляет ресурсами проекта, совместно с Senior-программистом устанавливает сроки выполнения задач и разрабатывает технические задания, взаимодействуя с заказчиком.

1.2 Теория графов в задачах безопасности

Существуют различные типы графов, отличающиеся характером связей между вершинами. В ориентированных графах рёбра имеют направление, что означает возможность перехода только в одном направлении от исходной вершины к целевой. Такой подход эффективен для моделирования причинно-следственных связей при анализе угроз (рисунок 3). «В неориентированных

графах связи между вершинами двусторонние, что подходит для описания симметричных отношений [11]».

Особое значение имеют взвешенные графы, где «каждому ребру присваивается числовое значение, например, вероятность реализации угрозы или оценка ущерба [10]». Это позволяет количественно оценивать риски безопасности (рисунок 4).

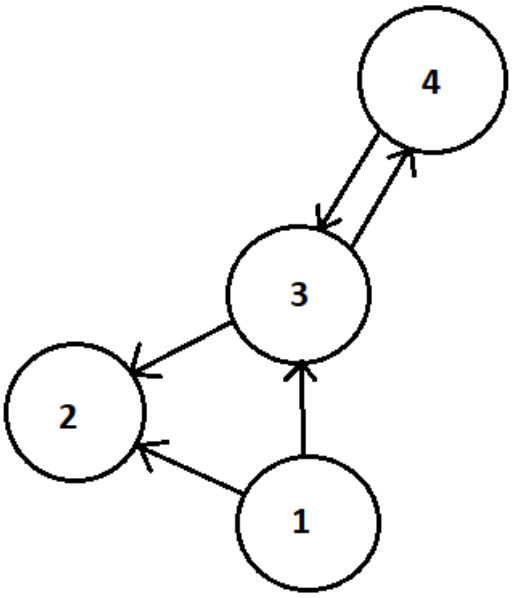


Рисунок 3 – Ориентированный граф

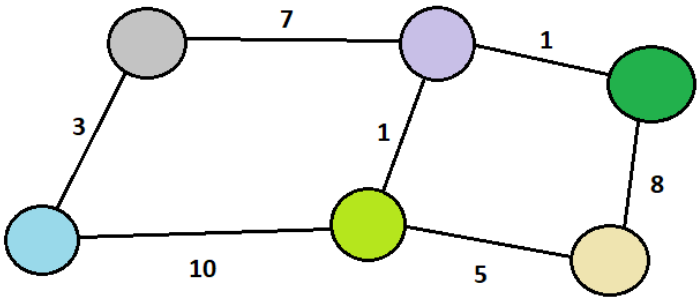


Рисунок 4 – Взвешенный граф

1.3 Методы представления модели на основе графов

«Для представления графов в вычислительных системах используются различные методы. Матрица смежности – квадратная матрица размером |V|×|V|, где элементы a_{ij} указывают наличие и вес связи между вершинами i и j [13]». Этот метод обеспечивает быстрый доступ к информации о смежности, но требует значительных объёмов памяти для крупных графов (рисунки 5 и 6).

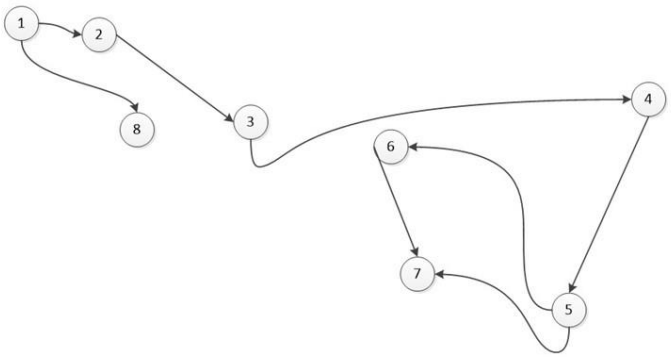


Рисунок 5 – Неориентированный граф

L	1	2	3	4	5	6	7	8
1	0	1	0	0	0	0	0	1
2	0	0	1	0	0	0	0	0
3	0	0	0	1	0	0	0	0
4	0	0	0	0	1	0	0	0
5	0	0	0	0	0	1	1	0
6	0	0	0	0	0	0	1	0
7	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0

Рисунок 6 – Матрица смежности

При работе со строкой матрицы, имеется элемент, из которого выходит ребро, в данном случае вершина 1 входит в вершину 2 и 8 (рисунок 6). При работе со столбцом, рассматриваются те ребра, которые входят в данную вершину. В вершину 1 ничего не входит, значит матрица верна.

Альтернативный подход – списки смежности (рисунки 7 и 8), где для

каждой вершины хранится перечень связанных с ней вершин. Данный метод более экономно использует память, особенно для разреженных графов, что характерно для моделей угроз безопасности.

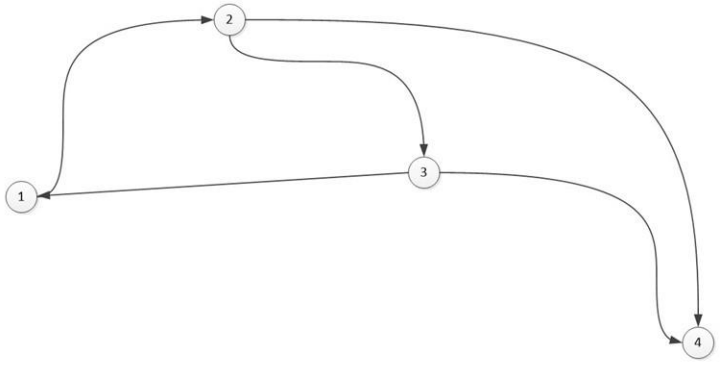


Рисунок 7 – Ориентированный граф

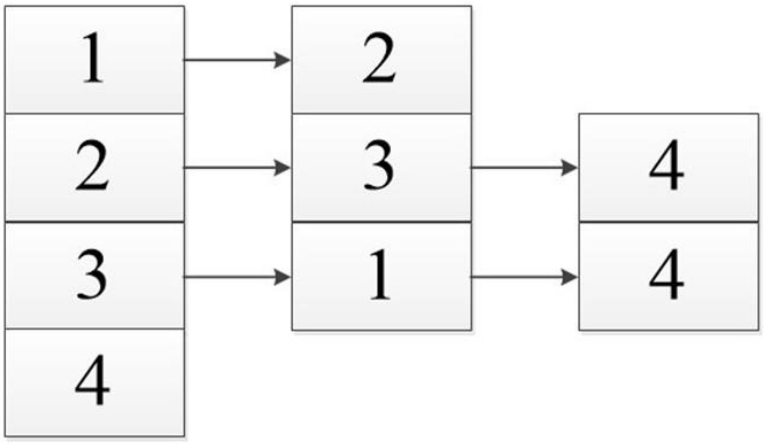


Рисунок 8 – Ориентированный граф в виде списка

«Неважно в каком порядке располагать ссылку, так как рассматривается смежность относительно первой ячейки, все остальные ссылки указывают лишь на связь с ней, а не между собой [18]». Так как существует смежность, то может понадобится дублирование вершин. Поэтому «сумма длин всех списков считается как: $2|E|$. Объем памяти: $O(|V|+|E|)$. [15]».

В Взвешенном графе вместо обозначающим наличие связи между вершинами, хранится вес ребра (рисунок 9).

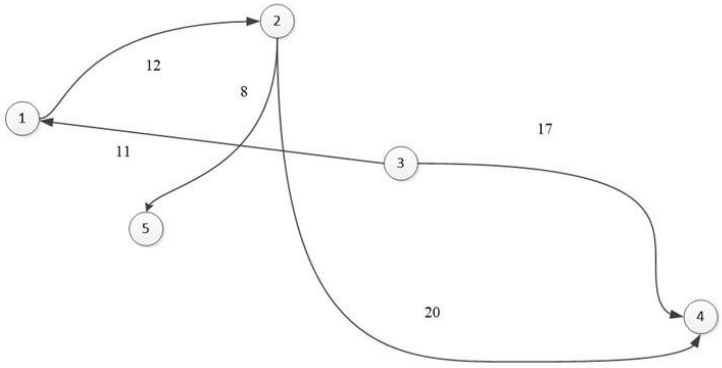


Рисунок 9 – Взвешенный граф

I	1	2	3	4	5
1	0	12	0	0	0
2	0	0	0	20	8
3	11	0	0	17	0
4	0	0	0	0	0
5	0	0	0	0	0

Рисунок 10 – Взвешенный граф в виде матрицы смежности

В ячейках указывается веса ребра, а в местах, где отсутствует связь 0 или -∞ (рисунок 10).

1.4 Алгоритмы обхода графа

Поиск в глубину (Depth-First Search) – рекурсивный алгоритм, использующий стек для посещения вершин. Алгоритм поиска в глубину основывается на принципе «углубления». Суть этого метода заключается в последовательном движении от исходной вершины по направлению к соседним вершинам до тех пор, пока не будет достигнута конечная вершина

или конечный элемент пути. В случае, если достигнут конец пути, но не найдена целевая вершина, процесс возвращается к предыдущей точке разветвления (или узлу) и продолжается по следующему возможному пути. Этот подход позволяет эффективно исследовать структуру графа, углубляясь в его ветви, и обеспечивает возможность поиска решения. Блок-схема работы DFS алгоритма представлена на рисунке 14.

Есть ориентированный граф, который выглядит так (рисунок 11):

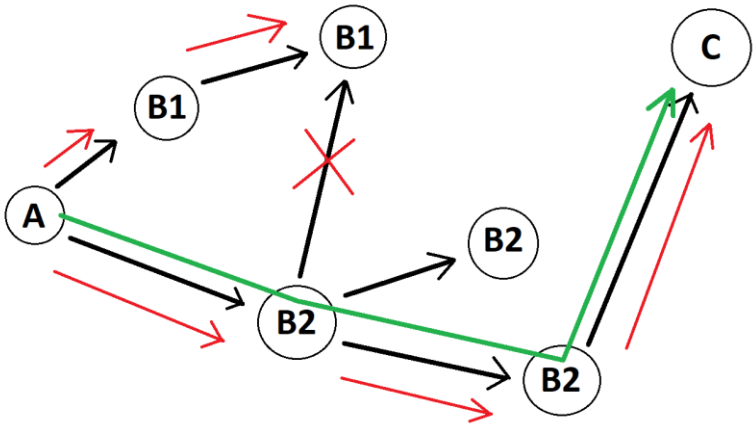


Рисунок 11 – DFS алгоритм в ориентированном графе

Начальной точкой считается A и требуется найти вершину C. Применяя DFS, исследуется один из возможных путей, двигается по нему до конца и, если не обнаружил t, возвращается и исследует другой путь.

Здесь алгоритм двигается по пути B1 к ближайшей вершине и видит, что это не конец пути, поэтому переходит к следующей вершине.

Дальше он достиг конца B1, но не нашел C, поэтому возвращается в A и двигается по второму пути.

Достигнув ближайшей к точке A вершины пути B2 алгоритм видит три возможных направления для дальнейшего движения. Поскольку вершину, венчающую первое направление, он уже посещал, то двигается по второму.

Здесь вновь достиг конца пути, но не нашел С, поэтому возвращается назад. Следует по третьему пути и, наконец, достигает искомой вершины С.

Так работает алгоритм в глубину. Двигается по определенному пути до конца. Если конец пути – это искомая вершина, то задача выполнена. Если нет, возвращается назад и двигается по другому пути до тех пор, пока не исследует все варианты.

Поиск в ширину (Breadth-First Search, BFS) — итеративный алгоритм, использующий очередь и основанный на принципе «расширения». В отличие от поиска в глубину, BFS последовательно исследует всех соседей текущей вершины, переходя от одного уровня графа к другому. Такой подход обеспечивает полное покрытие графа и применяется для поиска кратчайших путей в невзвешенных графах и задач, требующих рассмотрения всех возможных решений на каждом шаге (рисунок 12).

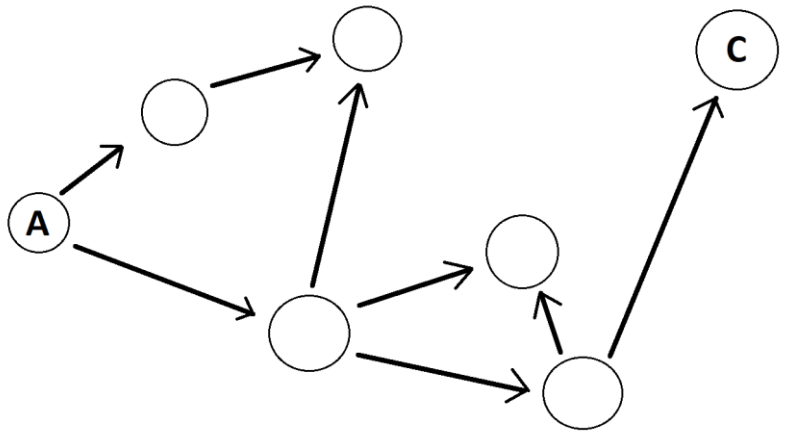


Рисунок 12 – BFS алгоритм в невзвешенном графе

Вместо следования по пути, поиск в ширину подразумевает посещение ближайших к А соседей за одно действие (шаг), затем посещение соседей и так до тех пор, пока не будет обнаружено С (рисунок 13).

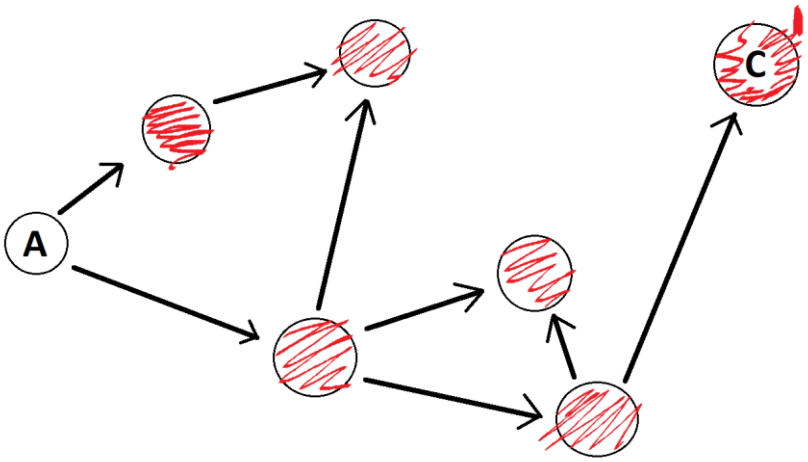


Рисунок 13 – работа BFS алгоритма в невзвешенном графе

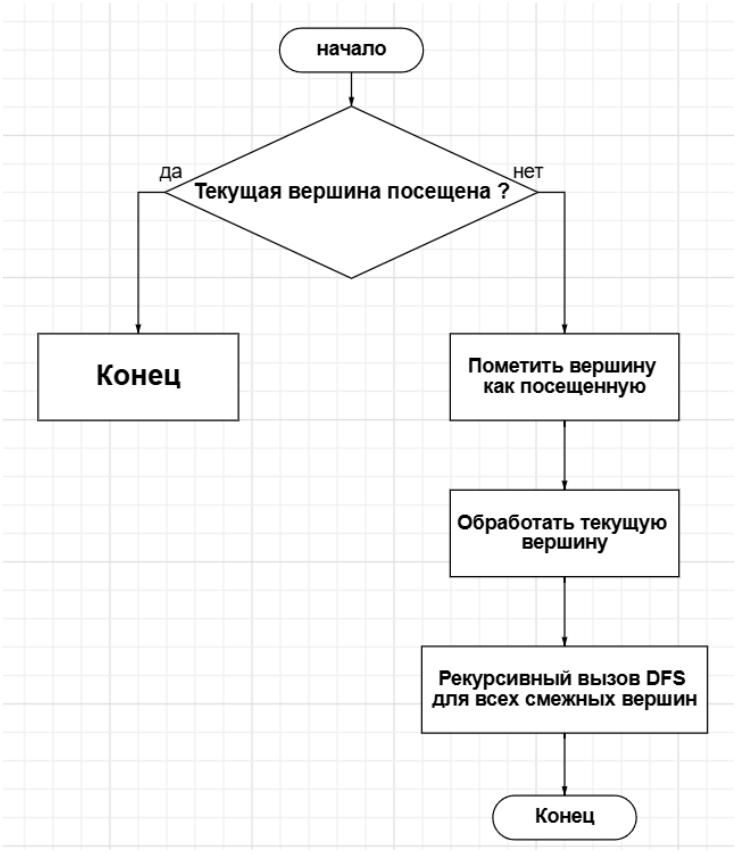


Рисунок 14 – Блок-схема алгоритма DFS:

1.5 Типы структур в модели на основе графов модели угроз

При анализе рисков удобно использовать подход, основанный на последовательностях событий. Это помогает визуализировать, как угрозы влияют на систему на разных этапах. Например, можно представить этапы проекта в виде узлов графа, где A, B, C… – состояния системы (например, «нормальная работа», «обнаружена уязвимость»); стрелки между узлами – угрозы, которые переводят систему из одного состояния в другое.

«Если для нарушения безопасности требуется выполнение нескольких этапов (например, взлом защиты переходит в утечку данных), такая последовательность называется цепочкой угроз [15]». Для удобства каждую угрозу можно обозначить двоичным флагом:

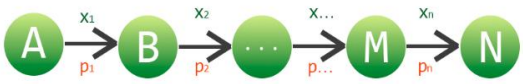
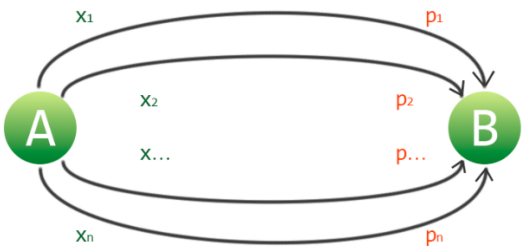
- 0 – угроза не реализована;
- 1 – угроза активна.

Если реализация общей угрозы – события X = (нарушение защищенности системы) требует прохождения нескольких этапов (состояний), то такая «последовательность угроз представима в виде последовательной цепи (сети) событий $X_i, i = (\overline{1 \dots n})$ [2]». Для обозначения факта выполнения или невыполнения угрозы воспользуемся бинарной кодировкой, где значение 0 будет означать что угроза не реализовалась, 1 – угроза реализовалась.

Вероятность реализации угрозы X_i обозначим как $P(X_i) = p_i$, а вероятность нарушения безопасности системы (НБС) – как *P(X)*.

«Угрозы, способные возникать одновременно и вызывать одинаковое состояние системы, можно описать с помощью параллельной сети [9]». В таблице 1 приведены формулы для расчета вероятностей в зависимости от структуры сети, включая общий случай для системы из n элементов.

Таблица 1 Формулы сети и вычисления вероятности для заданной структуры

Схема	Формула сети	Формула вероятности
Последовательная структура сети из n элементов		
	$X = X_1 \cdot X_2 \dots \cdot X_n = \prod_{i=1}^{n} X_i.$	$P(X) = \prod_{i=1}^{n} p_i.$
Параллельная структура сети из n элементов		
	$X = X_1 + X_2 \dots + X_n = \sum_{i=1}^{n} X_i.$	$P(X) = 1 - \prod_{i=1}^{n} (1 - p_i).$

«Когда угрозы реализуются через различные уязвимости системы, это создаёт сложную структуру, которую можно назвать «смешанной» (рисунок 15). Она включает в себя как последовательные, так и параллельные элементы [9]». При этом методы расчёта вероятностей для каждого участка сети остаются прежними и опираются на данные из таблицы 1. Однако, чем сложнее становится структура сети, тем более громоздкими и сложными оказываются формулы для вычисления вероятностей.

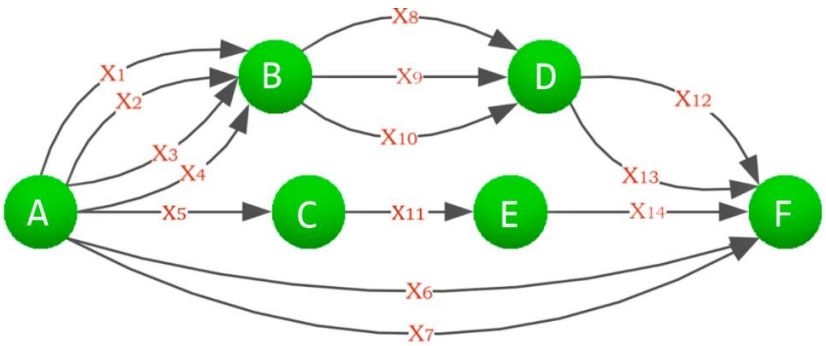


Рисунок 15 – Система, со смешанной структурой сети.

В процессе анализа угроз могут возникать ситуации, когда переход системы в определённое состояние возможен несколькими путями. Такую структуру называют «мостовой». Пример элементарной мостовой структуры показан на рисунке 16.

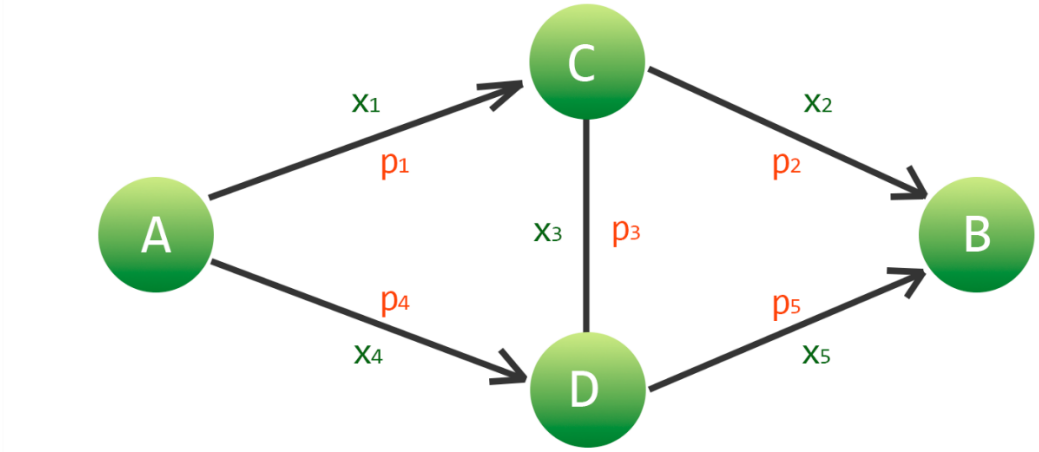


Рисунок 16 – Схема мостовой структуры

Вероятность реализации события AB = (Угроза от состояния A до состояния B реализована, т. е. НЗС) для мостовой схемы рассчитывается по формуле полной вероятности:

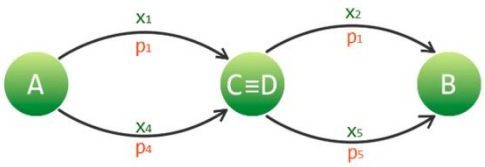
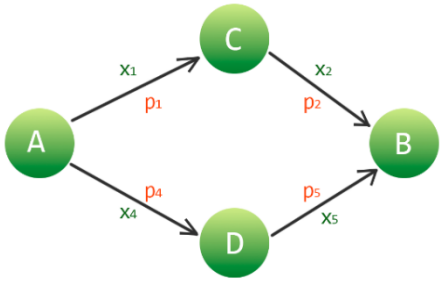
$$P(X) = P(AB)\sum_{i=1}^{2} P(AB|H_i) = P[(X_1 + X_4)\cdot(X_2 + X_5)]\cdot P(X_3) + \\ + P[(X_1X_2) + (X_4X_5)]\cdot P(\overline{X_3}). \quad (1)$$

Здесь H_1 и H_2 это гипотезы:

- H_1 означает, что угроза X_3 была реализована;
- H_2 означает, что угроза X_3 не была реализована.

Структура, схема и соответствующие формулы для сети в случае реализации или отсутствия угрозы X_3 приведены в таблице 2.

Таблица 2 Структура, схема и формула сети при выполнении/не выполнении мостовой угрозы

Схема	Формула сети	Гипотеза
	$X(AB\|H_1) = (X_1 + X_4) \cdot$ $\cdot (X_2 + X_5).$	H_1 – угроза X_3 была реализована (X_3 = 1)
	$(AB\|H_2) = (X_1X_2)+$ $+(X_4X_5).$	H_2 – угроза X_3 не была реализована (X_3 = 0)

Для вычисления вероятности нарушения безопасности системы (НБС) в мостовой структуре используется следующая формула:

$$P(AB) = (P[(X_1 + X_4) \cdot (X_2 + X_5)] \cdot P(P_3) + P[(X_1X_2) + (X_4X_5)] \cdot P(\overline{X_3}). \quad (2)$$

Для расчетов применяются стандартные формулы теории вероятностей:

- формула сложения вероятностей (для суммы событий):

$$P(M + N) = P(M) + P(N) - P(M) \cdot P(N). \quad (3)$$

- формула умножения вероятностей (для произведения независимых событий):

$$P(M \cdot N) = P(M) \cdot P(N). \quad (4)$$

Используя эти формулы, получаем итоговое выражение:

$$P(X_1)P(X_2) + P(X_4)P(X_5) + P(X_1)P(X_3)P(X_5) + \\ +P(X_2)P(X_3)P(X_4) - P(X_1)P(X_2)P(X_3)(X_4) - \\ -P(X_1)P(X_2)P(X_3)P(X_5) - P(X_1)P(X_2)P(X_4)P(X_5) - \\ -P(X_1)P(X_3)(X_4)P(X_5) - P(X_2)P(X_3)P(X_4)P(X_5) + 2 \cdot \\ \cdot P(X_1)P(X_2)P(X_3)P(X_4)P(X_5). \quad (5)$$

Поскольку некоторые угрозы могут быть реализованы или пропущены, выполнение цепочек угроз зависит от предыдущих этапов. «Полученное выражение позволяет выявить наиболее уязвимые участки сети. Например, можно определить, какая угроза, будучи устранённой, предотвратит выполнение всей цепочки [9]».

Для определения влияния каждой угрозы на общий риск используется анализ частных производных. Элемент с наибольшим значением частной производной становится ключевым объектом для дальнейшего анализа и устранения.

При последовательной структуре системы, для вычисления частной производной по каждой из угроз необходимо воспользоваться следующей формулой:

$$\frac{\partial P(X)}{\partial p_j} = \prod_{i \neq j} p_i = \frac{1}{p_j} \prod_{i=1}^{n} p_i = \frac{P(X)}{pe_j}. \quad (6)$$

Согласно формуле (13), можно сделать вывод, что в системах с последовательной структурой наиболее эффективно предотвращать угрозы с наименьшей вероятностью реализации

Устранение такой угрозы приводит к разрыву всей цепочки, что значительно снижает общий риск.

Для систем с параллельной структурой используется другая формула для вычисления частных производных:

$$\frac{\partial P(X)}{\partial p_j} = \frac{\partial P(X)}{\partial q_j} \cdot \frac{\partial q_j}{\partial p_j} = -\frac{\partial P(X)}{\partial q_j} = \prod_{i \neq j} q_i = \frac{1}{q_j} \prod_{i=1}^{n} q_i = \frac{1-P(X)}{q_j}. \quad (7)$$

Согласно формуле (14), в системах с параллельной структурой наиболее эффективно сосредоточиться на устранении угроз с наибольшей вероятностью реализации p_j. Снижение вероятности или полное устранение такой угрозы значительно уменьшает общий риск, так как она оказывает наибольшее влияние на систему.

Решим задачу параллельной и последовательной структуры сети. Схемы сети аналогичны схемам в таблице 1. Зададим вероятности реализации угроз случайными значениями от 0 до 1.

$$P(X_1) = p_1 = 0.5;$$
$$P(X_2) = p_2 = 0.7.$$

Для последовательной структуры частные производные будут равны:

$$\frac{\partial P(X)}{\partial p_1} = p_2 = 0.7;$$
$$\frac{\partial P(X)}{\partial p_2} = p_1 = 0.5.$$

Из этих результатов следует, что эффективнее предотвращать угрозу с наименьшей вероятностью $p_1 = 0.5$, так как её устранение разрывает всю цепочку.

Для параллельной структуры частные производные рассчитываются следующим образом:

$$\frac{\partial P(X)}{\partial p_1} = 1 - p_2 = 0.3;$$
$$\frac{\partial P(X)}{\partial p_2} = 1 - p_1 = 0.5.$$

Здесь более эффективно бороться с угрозой, имеющей наибольшую вероятность $p_2 = 0.7$, так как она сильнее влияет на общий риск.

Особый интерес представляет «мостовая» угроза X_3. Для её анализа частные производные в общем виде выглядят так:

$$\frac{\partial P(X)}{\partial p_1} = p_2 + p_3p_5 - p_2p_3p_4 - p_2p_3p_5 - p_2p_4p_5 - p_3p_4p_5 + 2p_2p_3p_4p_5;$$

$$\frac{\partial P(X)}{\partial p_2} = p_1 + p_3p_4 - p_1p_3p_4 - p_1p_3p_5 - p_1p_4p_5 - p_3p_4p_5 + 2p_1p_3p_4p_5;$$

$$\frac{\partial P(X)}{\partial p_3} = p_1p_5 + p_2p_4 - p_1p_2p_4 - p_1p_2p_5 - p_1p_4p_5 - p_2p_4p_5 + 2p_1p_2p_4p_5;$$

$$\frac{\partial P(X)}{\partial p_4} = p_5 + p_2p_3 - p_1p_2p_3 - p_1p_2p_5 - p_1p_3p_5 - p_2p_3p_5 + 2p_1p_2p_3p_5;$$

$$\frac{\partial P(X)}{\partial p_5} = p_4 + p_1p_3 - p_1p_2p_3 - p_1p_2p_4 - p_1p_3p_4 - p_2p_3p_4 + 2p_1p_2p_3p_4.$$

После проведения ряда экспериментов с различными значениями вероятностей для мостовой угрозы были получены разнообразные результаты.

Это указывает на то, что «невозможно установить универсальную закономерность для определения наиболее значимой угрозы [9]». Каждый случай требует индивидуального анализа. В рамках данной работы были рассмотрены основные типы сетевых структур, а также выведены базовые формулы, которые позволяют выявить ключевые закономерности. Например, в последовательных структурах эффективнее предотвращать угрозы с наименьшей вероятностью, а в параллельных – с наибольшей.

1.6 Математическая модель угроз безопасности

В рамках анализа информационной безопасности систем актуальной задачей является построение формализованной математической модели угроз. В данной главе представлена модель (рисунок 17), основанная на аппарате теории графов, позволяющая оценивать вероятность реализации угроз, выявлять критические узлы системы и принимать обоснованные меры по снижению рисков. В таблице 3 указаны полные формулы вероятностей реализации угроз при различных структурах сети.

Формализация модели. Для моделирования угроз используется

ориентированный взвешенный граф

$$G = (V, E, w), \tag{8}$$

где V – множество вершин, представляющих состояния системы;

$E \subset V \cdot V$– множество ориентированных рёбер, отражающих возможные переходы между состояниями;

$w: E \to [0,1]$.

Каждое ребро $(v_i, v_j) \in E$ имеет вес $p_{ij} = w(v_i, v_j)$, отражающий вероятность перехода.

$\mathrm{P}(A \to B) = P(X_3) \cdot P_1 + \big(1 - P(X_3)\big) \cdot P_2$.

Общая вероятность достижения критического состояния:

$$P_{\text{риск}} = \sum_{\pi \in \text{Путь}(v_0, v_c)} \prod_{(v_i, v_j) \in \pi} w(v_i, v_j). \tag{9}$$

Оценка критических узлов:

$$\frac{\partial P}{\partial p_{ij}}. \tag{10}$$

Таблица 3 – Вероятности реализации угроз при разных структурах сети

Структура	Формула	Пример (числовой)
Последовательная	$P = \prod p_i.$ (11)	$0{,}5 \cdot 0{,}7 = 0.35$
Параллельная	$P = 1 - \prod (1 - p_i).$ (12)	$1 - 0{,}5 \cdot 0{,}7 = 0{,}65$
Мостовая	$P = P(X_3)P_1 + \big(1 - P(X_3)\big)P_2.$ (13)	Зависит от P_1, P_2, $P(X_3)$

Рисунок 17 – Блок-схема математической модели угроз безопасности

1.7 Оценка критических узлов системы

Критические узлы системы определяются через анализ частных производных вероятностей:

$$\frac{\partial P}{\partial v_i}, \tag{11}$$

где P – общая вероятность угрозы.

v_i – вероятность реализации конкретной уязвимости.

«Чем выше значение частной производной $\frac{\partial P}{\partial v_i}$, тем больше влияние узла i на общую вероятность угрозы. Таким образом, узлы с наибольшими значениями частных производных считаются критическими [12]».

Рассмотрим граф на рисунке 18:

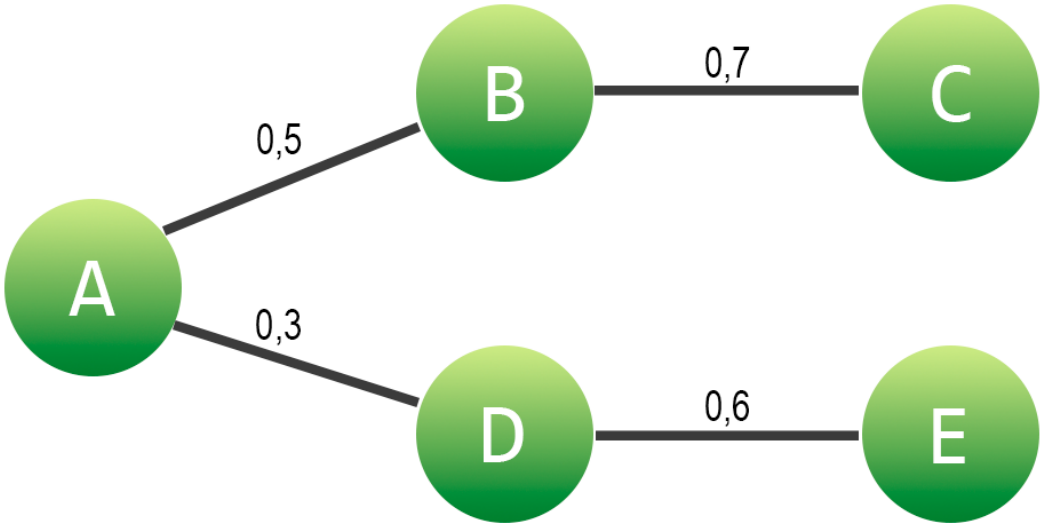


Рисунок 18 – граф с последовательными цепями

Последовательная цепь A в B в C: $P_{\text{посл}} = p_1 \cdot p_2 = 0.5 \cdot 0.7 = 0.35$;

Последовательная цепь A в D в E: $P_{\text{посл}} = p_3 \cdot p_4 = 0.3 \cdot 0.6 = 0.18$;

Параллельные цепи A в B и A в D:

$$P_{\text{паралл}} = 1 - (1 - p_1) \cdot (1 - p_3). \quad (12)$$

Общая вероятность вычисляется по формуле (22) для параллельных событий. Подставляем значения вероятностей:

$$P_{\text{паралл}} = 1 - (1 - 0.5) \cdot (1 - 0.3) = 1 - 0.5 \cdot 0.7 = 1 - 0.35 = 0.65.$$

«Критические узлы системы – это вершины графа, которые наиболее уязвимы для атак. Они определяются через: Высокую вероятность реализации угрозы; Большое количество входящих или исходящих рёбер; Влияние на общую вероятность угрозы (через частные производные) [4]».

Рассмотрим предыдущий граф, где вершина B является критической, так как она имеет высокую вероятность перехода в состояние C (p2=0.7), а вершина D также может быть критической, если вероятность перехода в E (p4=0.6) значительна. Для более точного определения критических узлов можно использовать анализ частных производных, как было описано в разделе 7. Для вершины B: $\frac{\partial P}{\partial p_2} = p_1 = 0.5$.

Изменение вероятности p2 (переход B в C, *B* в *C*) существенно влияет на общую вероятность угрозы.

Для вершины D: $\frac{\partial P}{\partial p_4} = p_3 = 0.3$.

Влияние D меньше, чем у B, оно всё же значимо, особенно если вероятность p3 увеличивается.

В первой главе проведён анализ предметной области, обоснован выбор теории графов для моделирования угроз безопасности. Рассмотрены различные типы графов (ориентированные, неориентированные, взвешенные) и методы их представления (матрица смежности, списки смежности). На основании формул для расчёта вероятностей реализации угроз в последовательных, параллельных, смешанных и мостовых структурах разработана математическая модель и выполнена формализация.

Глава 2 Проектирование программной реализации модели

2.1 Выбор средств разработки

Для эффективной реализации программной части дипломной работы был проведён сравнительный анализ популярных интегрированных сред разработки (IDE), подходящих для создания приложений под Windows. Рассматривались следующие среды: Visual Studio 2022, CLion, Code::Blocks и Eclipse CDT.

Visual Studio 2022 выделяется своей полной интеграцией с операционной системой Windows, что обеспечивает максимальную совместимость с Windows API и оптимизацию под платформу. «Среда обладает широким набором инструментов для разработки, отладки и тестирования приложений [17]». Её удобный и интуитивно понятный интерфейс, а также встроенные средства профилирования и анализа кода делают Visual Studio предпочтительным выбором для проектов любой сложности.

CLion, разработанная компанией JetBrains, представляет собой кроссплатформенную IDE с мощным интеллектуальным редактором кода и системой анализа. Она отлично подходит для кроссплатформенной разработки и имеет интеграцию с системами сборки CMake, что облегчает управление зависимостями. Однако, в сравнении с Visual Studio, CLion менее оптимизирована для работы с Windows API, что ограничивает её использование при создании нативных оконных приложений под Windows.

Code::Blocks – это лёгкая и простая IDE с открытым исходным кодом, поддерживающая различные компиляторы. «Она обладает минималистичным интерфейсом и сравнительно ограниченным набором встроенных функций, что может усложнить работу над крупными проектами [20]». Однако, благодаря своей простоте и открытости, «Code::Blocks часто используется для образовательных целей и небольших проектов [21]».

Eclipse CDT является расширением популярной платформы Eclipse для разработки на C и C++. «Она кроссплатформенная и поддерживает множество плагинов, расширяющих функциональность [1]». Тем не менее, для полноценной работы с Windows API требуется установка дополнительных компонентов и настройка, что увеличивает время подготовки среды к работе.

Для оценки рассмотренных средств разработки были выбраны следующие критерии: доступность среды, необходимость установки дополнительных плагинов для работы с Windows API, простота настройки, удобство интерфейса и наличие встроенных шаблонов и инструментов для ускорения разработки. Сравнительный анализ перечисленных критериев показан в таблице 4.

Таблица 4 – Сравнительный анализ средств разработки

Среда разработки	Доступность	Плагин для C++	Разработка без плагина	Простота настройки	Удобство среды	Наличие готовых шаблонов
Visual Studio 2022	+	-	+	+	+	+
CLion	-	-	-	+	+	+
Code::Blocks	+	+	-	+	-	-
Eclipse CDT	+	+	-	-	-	-

По результатам анализа выбор был сделан в пользу Visual Studio 2022, поскольку данная среда максимально полно соответствует требованиям работы. Она обеспечивает широкий функционал из коробки, не требует дополнительной настройки и установки плагинов для полноценной работы. Высокий уровень интеграции с операционной системой позволяет сосредоточиться на решении прикладных задач, а удобный интерфейс и набор готовых инструментов значительно ускоряют процесс разработки.

2.2 Выбор языка программирования

В процессе разработки программного обеспечения важным этапом является выбор языка программирования. На сегодняшний день существует множество языков, среди которых выделяются Java, Python, C# и C++. В данной работе акцент будет сделан на сравнении C++ с другими языками, а именно Java и Python.

«Java, созданный компанией Sun Microsystems в 1995 году, зарекомендовал себя как один из самых популярных языков программирования, особенно в области разработки корпоративных и мобильных приложений [16]». Его ключевыми преимуществами являются «кроссплатформенность, обеспечиваемая виртуальной машиной Java (JVM), автоматическое управление памятью и обширная стандартная библиотека [16]». Эти особенности делают Java универсальным инструментом, подходящим для создания различных приложений, от веб-сервисов до сложных корпоративных систем. Однако, несмотря на свои достоинства, Java может быть критикован за меньшую скорость работы по сравнению с C++ и более сложный синтаксис.

Python, в свою очередь, известен своей простотой и лаконичностью. «Созданный в конце 1980-х годов, он стал особенно популярным в области научных исследований, анализа данных и разработки веб-приложений [8]». Его синтаксис позволяет быстро разрабатывать прототипы и упрощает чтение и поддержку кода. Однако Python может уступать в производительности C++ и Java, особенно когда дело касается задач, требующих высокой скорости обработки данных.

«C++, разработанный Бьёрном Страуструпом в начале 1980-х годов, представляет собой расширение языка C и поддерживает как процедурное, так и объектно-ориентированное программирование [12]». Одним из его главных преимуществ является высокая производительность и возможность низкоуровневого управления памятью, что делает его идеальным для

системного программирования и разработки высоконагруженных приложений. Однако, сложность синтаксиса и необходимость управления памятью могут стать препятствием для новичков.

Для удобства проведем сравнение языков программирования, обращая внимание на ключевые аспекты, такие как быстродействие, лаконичность кода, распространенность на рынке и доступность документации. Подробные результаты этого анализа представлены в таблице 5.

Таблица 5 – Сравнение языков программирования

Язык программирования	Скорость работы	Компактность кода	Распространенность на рынке	Доступность документации
Java	+	-	+	+
Python	-	+	+	+
C++	+	-	+	+

Каждый из языков имеет свои сильные и слабые стороны. На этапе выбора языка для разработки приложения было принято решение остановиться на C++, учитывая его высокую производительность и возможности для низкоуровневого программирования. Хотя Java и Python предлагают множество удобств и обладают высокой популярностью, их относительная медлительность и сложности в некоторых аспектах разработки оказали влияние на принятое решение.

2.3 Описание средств для решения поставленных задач

Для разработки программного обеспечения, направленного на анализ угроз безопасности, выбрана среда разработки Visual Studio и язык программирования C++. Данная среда обеспечивает мощный инструментарий для создания, отладки и тестирования приложений, что делает её идеальной для реализации серьезных проектов, связанных с анализом данных и алгоритмами.

Технологии, используемые в работе, охватывают как стандартные библиотеки C++ для работы с контейнерами и алгоритмами, так и специализированные библиотеки, которые облегчают задачу моделирования и анализа графов. Это позволит создать эффективную программную модель, способную обрабатывать сложные взаимосвязи между состояниями системы и угрозами.

Для хранения данных о состоянии системы и угрозах будет использован стандартный образец классов и структур, которые позволяют организовать информацию в удобочитаемом формате. Каждая угроза будет представлена в виде объекта с параметрами, такими как вероятность реализации и потенциальный ущерб. Это упростит манипуляции с данными и сделает алгоритмы более гибкими.

Для представления графов и их визуализации в программе будет использоваться структура данных, основанная на списках смежности. Этот метод позволяет эффективно хранить и обрабатывать данные о графах, обеспечивая быстрый доступ к смежным вершинам. «Списки смежности позволяют динамически изменять структуру графа, что особенно важно для моделирования изменяющихся угроз [22]».

Важным аспектом работы станет реализация алгоритмов обхода графов, таких как поиск в глубину (Depth-First Search) и поиск в ширину (Breadth-First Search). Эти алгоритмы будут использованы для анализа структуры графа и определения критических узлов, что является ключевым моментом для оценки уязвимости системы. На основе данных о вероятностях угроз будет производиться расчет рисков для каждого узла графа.

Для визуализации результатов анализа будет применяться библиотека для построения графиков, что поможет пользователям наглядно увидеть распределение рисков и критические точки в системе. Это обеспечит более понятное восприятие данных и позволит пользователям лучше понять потенциальные угрозы.

После разработки архитектуры приложения будет создана диаграмма

классов, которая отразит структуру программы и взаимосвязи между основными компонентами. Это поможет упростить восприятие системы и улучшить организацию кода.

Во второй главе проведён детальный сравнительный анализ современных средств разработки, включающий оценку таких сред, как CLion, Code::Blocks и Eclipse CDT, по ключевым критериям: производительность, удобство интерфейса, интеграция с Windows API и поддержка языка C++. В ходе исследования было установлено, что CLion, несмотря на мощные возможности рефакторинга, демонстрирует меньшую эффективность при работе с Windows API, а Code::Blocks и Eclipse CDT требуют значительной дополнительной настройки для полноценной реализации проекта. В результате комплексной оценки была выбрана среда Visual Studio 2022, которая обеспечивает наилучшую интеграцию с Windows API, обладает встроенными инструментами для работы с большими проектами и предлагает мощные средства отладки. Язык программирования C++ был предпочтён Java и Python благодаря его высокой производительности, низкоуровневым возможностям и оптимальной поддержке вычислительно сложных алгоритмов анализа графов. Разработанная архитектура программы включает несколько ключевых модулей: модель данных для хранения информации о вершинах и рёбрах графа, генератор инфраструктуры для создания тестовых сценариев, алгоритмы расчёта рисков на основе вероятностных моделей, а также подсистемы визуализации графа и поиска критических узлов. Особое внимание при проектировании было уделено оптимизации алгоритмов обхода графа и вычисления вероятностей, что позволяет системе эффективно обрабатывать графы с тысячами вершин. Для обеспечения высокой производительности были применены такие методы, как кэширование промежуточных результатов и параллельные вычисления. Реализованные механизмы масштабирования позволяют адаптировать систему для работы с различными объёмами данных - от небольших корпоративных сетей до сложных распределённых инфраструктур.

Глава 3 Реализация программы

3.1 Разработка программы

Для начала я хочу ознакомится, как теория графов помогает моделировать угрозы информационной безопасности. Мы разберем четыре ключевые структуры анализа рисков и посмотрим, как они реализуются на практике. Для наглядности я буду сопровождать объяснение фрагментами кода и демонстрацией работы программы.

```
class ThreatModelAnalyzer {
private:
    vector<char> nodes;
    vector<vector<Edge>> graph;

    // ...

public:
    ThreatModelAnalyzer() {
        // 1. Создание узлов A-J
        for (char c = 'A'; c <= 'J'; ++c) nodes.push_back(c);  // <-- Здесь!

        // 2. Создание рёбер
        for (size_t i = 0; i < nodes.size(); ++i) {
            vector<Edge> edges;
            bool has_connection = false;

            // 3. 50% вероятность соединения с последующими узлами
            for (size_t j = i + 1; j < nodes.size(); ++j) {
                if (connection_dist(gen) < 0.5) {  // <-- 50% вероятность здесь!
                    edges.push_back({
                        nodes[j],                  // Целевой узел
                        weight_dist(gen)           // Вероятность 0.5-1.0
                        });
                    has_connection = true;
                }
            }

            // 4. Автоматическое соединение со следующим узлом, если связей нет
            if (!has_connection && i < nodes.size() - 1) {  // <-- Здесь!
                edges.push_back({
                    nodes[i + 1],
                    weight_dist(gen)
                    });
            }

            graph.push_back(edges);
        }

        // 5. Очистка рёбер последнего узла (опционально)
        if (!graph.empty()) graph.back().clear();  // <-- Последний узел без связей
    }
```

Рисунок 19 – Фрагмент кода с заданными условиями

В моем коде создаётся направленный граф из узлов (A-J), где каждое ребро имеет случайную вероятность (0.5-1.0). Вероятность соединения между узлами – 50%. Если у узла нет связей, он автоматически соединяется со следующим. Реализованы методы для расчёта. Полный фрагмент данного кода показан на рисунке 19.

Последовательная структура. Этот метод рассчитывает общую вероятность успешного прохождения всех этапов атаки последовательно, как если бы злоумышленник должен был преодолеть каждую защиту одну за другой.

```
100     double sequentialStructure() const {
101         double product = 1.0;
102         for (const auto& edges : graph) {
103             for (const auto& edge : edges) {
104                 product *= edge.probability;
105             }
106         }
107         return product;
108     }
```

Рисунок 20 – Фрагмент кода с последовательной структурой

Метод показывает, насколько сложно реализовать сложную многоступенчатую атаку. Даже при относительно высоких шансах на каждом этапе, их последовательное преодоление резко снижает общую вероятность успеха (рисунок 20).

Для реализации этого метода я использовал формулу последовательной структуры, а именно формулу вероятностей (1 таблица).

Параллельная структура. Этот метод рассчитывает общую вероятность успешного прохождения атаки через хотя бы один из доступных путей, как если бы злоумышленник мог выбрать наиболее уязвимую защиту или использовать несколько независимых уязвимостей одновременно (рисунок 21).

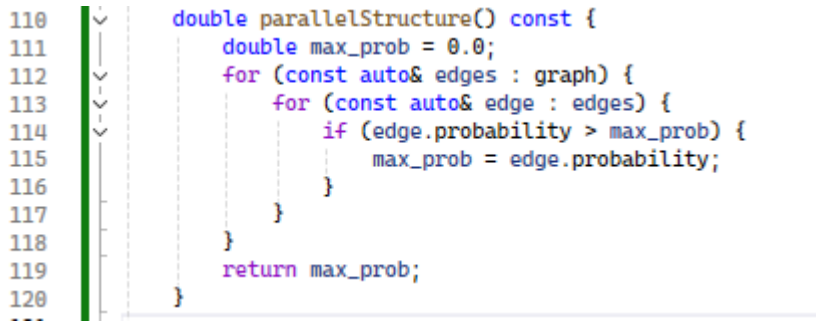


Рисунок 21 – Фрагмент года с параллельной структурой

Каждый новый путь атаки увеличивает общий шанс успеха, даже если отдельные вероятности малы. «В параллельной системе добавление новых компонентов повышает уязвимость, тогда как в последовательной – снижает [22]».

Для реализации этого метода я использовал формулу параллельной структуры, формула вероятностей (1 таблица).

Смешанная структура. Этот метод рассчитывает общую вероятность успеха атаки в системе, где сочетаются последовательные и параллельные элементы. Например, злоумышленник должен преодолеть несколько этапов защиты (последовательно), но на некоторых этапах есть альтернативные пути обхода (параллельно) (рисунок 22).

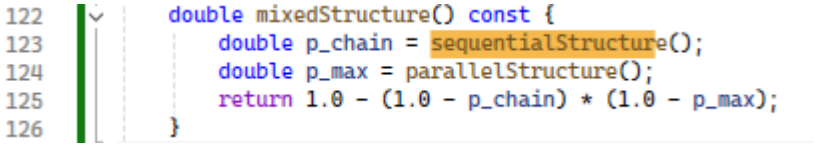


Рисунок 22 – Фрагмент кода с смешанной структурой

Вероятность вычисляется по формуле:

$$P_{\text{смеш}} = 1 - \left(1 - P_{\text{послед}}\right) \cdot \left(1 - P_{\text{паралл}}\right), \quad (13)$$

где

$P_{паралл}$ – вероятность успеха последовательной структуры,

$P_{паралл}$ – вероятность успеха параллельной структуры.

Метод учитывает, что злоумышленник может атаковать как через сложную многоэтапную цепочку, так и через отдельные слабые места.

Смешанная структура отражает сложные реальные системы, где успех атаки возможен через несколько независимых стратегий. Корректная реализация формулы позволяет точнее оценивать риски, учитывая все возможные векторы атак.

Мостовая структура. Этот метод рассчитывает вероятность успеха атаки в системе с двумя независимыми путями, которые могут пересекаться или взаимодействовать. Мостовая структура часто используется для моделирования сложных систем, где успех зависит от комбинации нескольких компонентов (рисунок 23).

```
128     double bridgeStructure() const {
129         if (graph.size() < 5) throw logic_error("Недостаточно узлов");
130         double p1 = graph[0].empty() ? 0.0 : graph[0][0].probability;
131         double p2 = graph[1].empty() ? 0.0 : graph[1][0].probability;
132         double p3 = graph[3].empty() ? 0.0 : graph[3][0].probability;
133         double p4 = graph[4].empty() ? 0.0 : graph[4][0].probability;
134         return (p1 * p2) + (p3 * p4) - (p1 * p2 * p3 * p4);
135     }
136 };
```

Рисунок 23 – Фрагмент кода с мостовой структурой

Вероятность вычисляется по формуле:

$$P_{мост} = P_1 \cdot P_2 + P_3 \cdot P_4 - P_1 \cdot P_2 \cdot P_3 \cdot P_4, \quad (14)$$

где

P_1, P_2 – вероятности первого пути (например, узлы $A \rightarrow B$),

P_3, P_4 – вероятности второго пути (например, узлы $C \rightarrow D$).

Формула учитывает:

– успех хотя бы одного из двух путей (логическое «ИЛИ»);

– корректировку на пересечение путей (чтобы не учитывать их совместный успех дважды).

Метод подходит для систем, где атака может быть выполнена через разные комбинации уязвимостей.

Вычитание $P_1 \cdot P_2 \cdot P_3 \cdot P_4$ исключает двойной учёт сценария, когда оба пути успешны одновременно.

Мостовая структура полезна для моделирования систем с альтернативными путями атаки, учитывая их взаимодействие. Текущая реализация кода требует доработки для большей гибкости, но корректно отражает базовый принцип объединения вероятностей. Код вывода результата показан на рисунке 24, а результат запуска всего кода показан на рисунке 25.

```
138   template<typename Func>
139   void measurePerformance(const string& name, Func func) {
140       auto start = high_resolution_clock::now();
141       try {
142           double result = func();
143           auto end = high_resolution_clock::now();
144           stringstream ss;
145           ss << MAGENTA << "╔" << string(TOTAL_WIDTH - 2, '=') << "╗\n"
146               << "║ " << BLUE << left << setw(TOTAL_WIDTH - 4) << name << MAGENTA << " ║\n"
147               << "╠" << string(TOTAL_WIDTH - 2, '-') << "╣\n"
148               << "║ " << RESET << "Результат: " << GREEN;
149
150           if (fabs(result) < 1e-4) {
151               ss << scientific << setprecision(6);
152           }
153           else {
154               ss << fixed << setprecision(10);
155           }
156           ss << result;
157
158           ss << string(6, ' ') << MAGENTA << " ║\n"
159               << "║ " << RESET << "Время: " << YELLOW << setw(22)
160               << duration_cast<microseconds>(end - start).count() << " мкс"
161               << string(8, ' ') << MAGENTA << " ║\n"
162               << "╚" << string(TOTAL_WIDTH - 2, '=') << "╝\n" << RESET;
163           cout << ss.str();
164       }
165       catch (const exception& e) {
166           cerr << RED << "\n[ОШИБКА] " << e.what() << RESET << endl;
167       }
168   }
169
```

Рисунок 24 – Фрагмент кода вывода данных

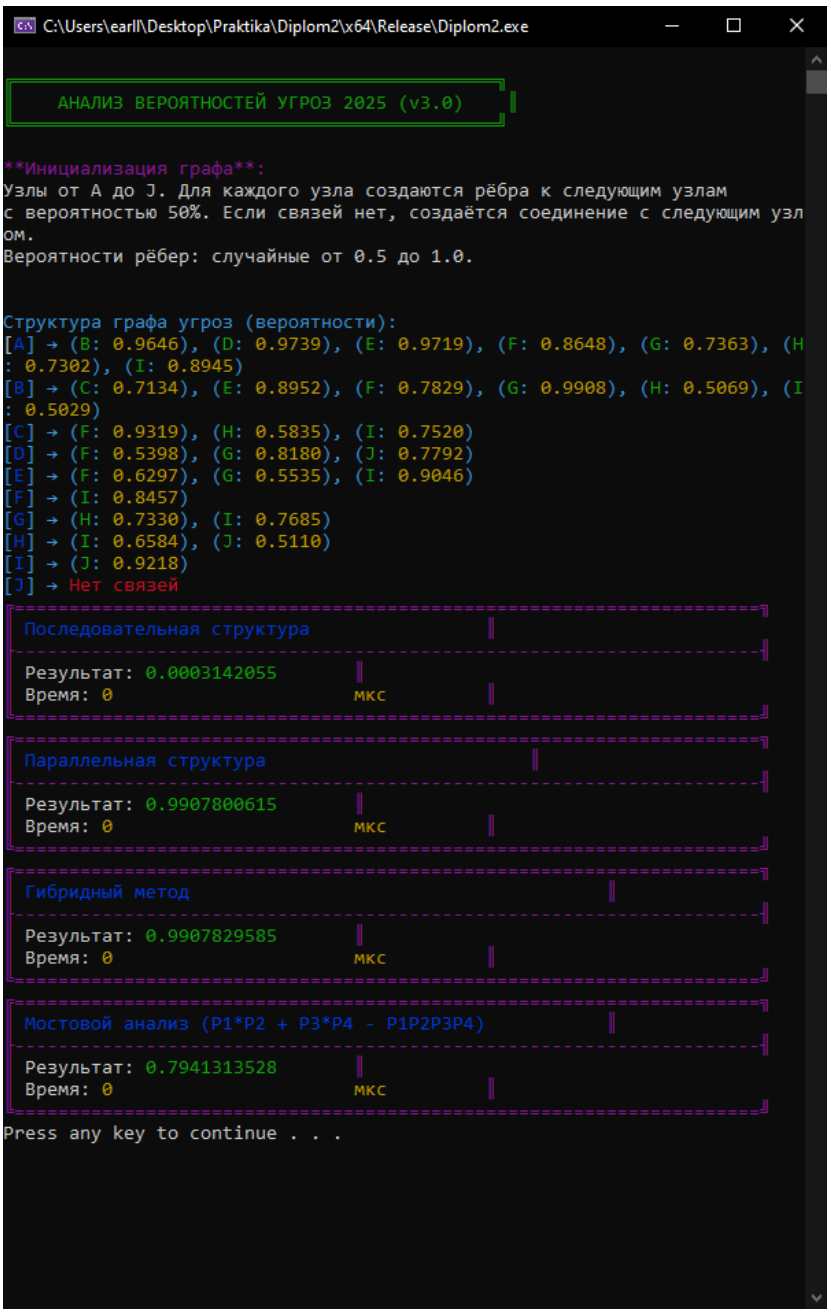


Рисунок 25 – Результат

3.2 Анализ и моделирование угроз на тестовой инфраструктуре

В данном разделе рассматривается применение разработанной математической модели к анализу информационной безопасности в условиях типовой ИТ-инфраструктуры. Современные инфраструктуры включают множество компонентов: веб-серверы, базы данных, сетевые устройства и

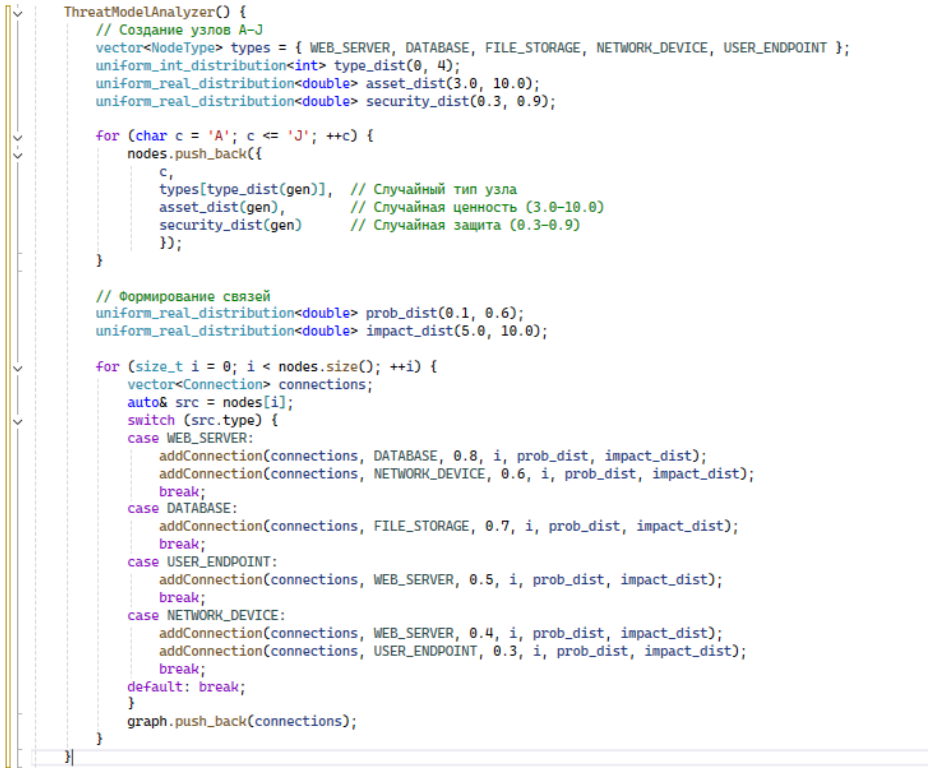
рабочие станции. «Каждый из этих элементов может стать мишенью для атак, а их взаимосвязи определяют, как угрозы распространяются внутри системы [5]».

```
59  struct Node {
60      char id;                // Уникальный идентификатор (A-J)
61      NodeType type;          // Тип узла (веб-сервер, база данных и т.д.)
62      double asset_value;     // Ценность актива (3.0-10.0)
63      double security_level;  // Уровень защиты (0.3-0.9)
64  };
65
66  struct Connection {
67      char target;            // Целевой узел (например, 'B')
68      double threat_prob;     // Вероятность угрозы (0.1-0.6)
69      double potential_impact; // Потенциальный ущерб (5.0-10.0)
70      string threat_type;     // Тип угрозы (DoS, Data Breach и т.д.)
71  };
72
```

Рисунок 26 – Фрагмент кода модели данных

Модель данных (рисунок 26) позволяет описать компоненты системы (узлы) и угрозы (связи) как объекты с параметрами. Узлы хранят информацию о ценности актива и уровне защиты, связи – о вероятности угрозы и её типе.

Пример: Моделирование веб-сервера (узел) и атаки на него (связь). Генерация инфраструктуры (рисунок 27):

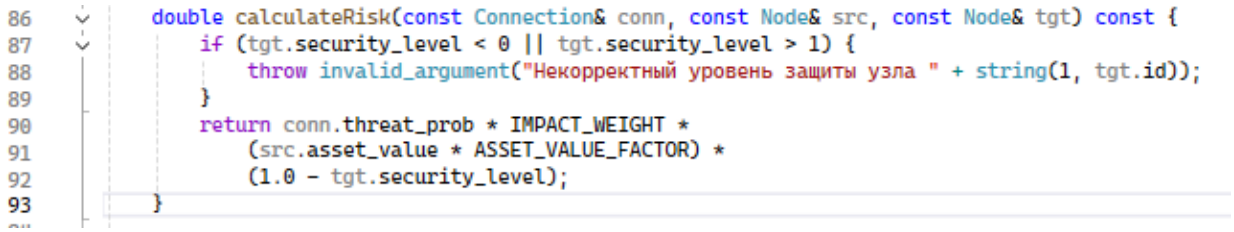


```
ThreatModelAnalyzer() {
    // Создание узлов A–J
    vector<NodeType> types = { WEB_SERVER, DATABASE, FILE_STORAGE, NETWORK_DEVICE, USER_ENDPOINT };
    uniform_int_distribution<int> type_dist(0, 4);
    uniform_real_distribution<double> asset_dist(3.0, 10.0);
    uniform_real_distribution<double> security_dist(0.3, 0.9);

    for (char c = 'A'; c <= 'J'; ++c) {
        nodes.push_back({
            c,
            types[type_dist(gen)],  // Случайный тип узла
            asset_dist(gen),        // Случайная ценность (3.0–10.0)
            security_dist(gen)      // Случайная защита (0.3–0.9)
            });
    }

    // Формирование связей
    uniform_real_distribution<double> prob_dist(0.1, 0.6);
    uniform_real_distribution<double> impact_dist(5.0, 10.0);

    for (size_t i = 0; i < nodes.size(); ++i) {
        vector<Connection> connections;
        auto& src = nodes[i];
        switch (src.type) {
        case WEB_SERVER:
            addConnection(connections, DATABASE, 0.8, i, prob_dist, impact_dist);
            addConnection(connections, NETWORK_DEVICE, 0.6, i, prob_dist, impact_dist);
            break;
        case DATABASE:
            addConnection(connections, FILE_STORAGE, 0.7, i, prob_dist, impact_dist);
            break;
        case USER_ENDPOINT:
            addConnection(connections, WEB_SERVER, 0.5, i, prob_dist, impact_dist);
            break;
        case NETWORK_DEVICE:
            addConnection(connections, WEB_SERVER, 0.4, i, prob_dist, impact_dist);
            addConnection(connections, USER_ENDPOINT, 0.3, i, prob_dist, impact_dist);
            break;
        default: break;
        }
        graph.push_back(connections);
    }
}
```

Рисунок 27 – Фрагмент кода генерации инфраструктуры

Узлы и связи отражают типичную корпоративную сеть с серверами, базами данных и рабочими станциями. Имитирует разнообразие реальных систем (разная ценность активов, уровень защиты, типы угроз). Программа сама строит граф, что упрощает тестирование гипотез. Этот граф используется для вычисления рисков, поиска уязвимых узлов и визуализации угроз. Этот код превращает абстрактные концепции теории графов в инструмент для анализа реальных угроз информационной безопасности.

Расчет риска (рисунок 28):



```
86  double calculateRisk(const Connection& conn, const Node& src, const Node& tgt) const {
87      if (tgt.security_level < 0 || tgt.security_level > 1) {
88          throw invalid_argument("Некорректный уровень защиты узла " + string(1, tgt.id));
89      }
90      return conn.threat_prob * IMPACT_WEIGHT *
91          (src.asset_value * ASSET_VALUE_FACTOR) *
92          (1.0 - tgt.security_level);
93  }
```

Рисунок 28 – Фрагмент кода расчета риска

Данный фрагмент убеждается, что уровень защиты целевого узла

(tgt.security_level) находится в диапазоне от 0 до 1. Если нет – генерирует ошибку.

Чего позволяет добиться:

– количественная оценка угроз: превращает абстрактные параметры (вероятность, ценность актива) в конкретное числовое значение риска;
– учет бизнес-логики: учитывает как технические факторы (вероятность атаки), так и бизнес-факторы (ценность актива);
– выявление уязвимостей: позволяет сравнивать риски разных узлов и связей, чтобы определить, куда направить ресурсы для усиления защиты;

Этот метод превращает абстрактные параметры инфраструктуры в конкретные числовые значения, позволяя анализировать угрозы на основе данных, а не интуиции.

Визуализация графа (рисунок 29):

```
182     void printGraph() const {
183         cout << CYAN << "\nСтруктура системы безопасности:\n" << RESET;
184         for (size_t i = 0; i < nodes.size(); ++i) {
185             const Node& n = nodes[i];
186             cout << "[" << BLUE << n.id << CYAN << "] "
187                 << left << setw(18) << type_names[n.type]
188                 << " (Ценность: " << YELLOW << fixed << setprecision(1) << n.asset_value
189                 << CYAN << ", Защита: "
190                 << GREEN << setprecision(0) << n.security_level * 100 << "%"
191                 << CYAN << ")\n";
192
193             for (const Connection& c : graph[i]) {
194                 try {
195                     double risk = calculateRisk(c, n, findNode(c.target));
196                     cout << "  → (" << GREEN << c.target << CYAN << ") "
197                         << left << setw(15) << c.threat_type
198                         << ": Риск=" << RED << setw(8) << fixed << setprecision(2) << risk
199                         << RESET << "\n";
200                 }
201                 catch (const exception& e) {
202                     cerr << RED << "Ошибка: " << e.what() << RESET << endl;
203                 }
204             }
205         }
206     }
207
```

Рисунок 29 – Фрагмент кода визуализации графа

Этот метод превращает абстрактный граф угроз в понятную текстовую

диаграмму, помогая аналитикам быстро выявлять критические точки и принимать решения по усилению защиты.

Позволяет быстро оценить структуру системы и распределение рисков. Пользователь видит, какие узлы и связи наиболее уязвимы (например, красный цвет указывает на высокий риск). «Отображает не только топологию графа, но и метаданные (ценность, защиту, тип угрозы) [19]».

Поиск критического узла (рисунок 30):

```
223     pair<char, double> findCriticalNode() const {
224         auto max_it = max_element(risk_cache.begin(), risk_cache.end(),
225             [](const pair<char, double>& a, const pair<char, double>& b) {
226                 return a.second < b.second;
227             });
228         return make_pair(max_it->first, max_it->second);
229     }
230 };
231
```

Рисунок 30 – Фрагмент кода поиска узла с максимальным риском

Этот метод превращает данные графа угроз в конкретные рекомендации, помогая эффективно распределять ресурсы для защиты инфраструктуры.

Определяет узел, который подвержен наибольшему риску, чтобы сосредоточить на нем ресурсы защиты. Учитывает все угрозы, направленные на узел, а не только отдельные связи. Позволяет ответить на вопрос: «Какой компонент системы нужно усилить в первую очередь?»

Профили угроз (рисунок 31):

```
79      const map<string, pair<double, double>> threat_profiles = {
80          {"DoS",         {0.3, 8.7}},
81          {"Data Breach", {0.2, 9.5}},
82          {"Malware",     {0.4, 7.2}},
83          {"Insider",     {0.1, 6.8}}
84      };
```

Рисунок 31 – Фрагмент кода определения параметров для разных типов угроз

Этот фрагмент кода делает модель угроз реалистичной и гибкой, позволяя:

– учитывать специфику разных типов атак;

– быстро адаптировать систему под новые угрозы;

– анализировать, какие виды угроз наиболее опасны для конкретной инфраструктуры.

«Разные типы атак (DoS, утечка данных и так далее) имеют уникальные параметры, что делает модель более реалистичной [19]». Позволяет легко добавлять новые типы угроз или корректировать существующие без изменения основной логики программы. Например, атаки типа Data Breach имеют низкую вероятность, но высокий ущерб, а DoS – более частые, но менее разрушительные.

Замер производительности (рисунок 32):

```
232     template<typename T, typename Func>
233     T measurePerformance(const string& name, Func func) {
234         auto start = high_resolution_clock::now();
235         try {
236             T result = func();
237             auto end = high_resolution_clock::now();
238 
239             stringstream ss;
240             ss << MAGENTA << "╔" << string(TOTAL_WIDTH - 2, '=') << "╗\n"
241                << "║ " << BLUE << left << setw(TOTAL_WIDTH - 4) << name << MAGENTA << " ║\n"
242                << "╠" << string(TOTAL_WIDTH - 2, '-') << "╣\n"
243                << "║ " << RESET << "Результат: " << GREEN << setw(12) << fixed << setprecision(2) << result
244                << string(18, ' ') << MAGENTA << " ║\n"
245                << "║ " << RESET << "Время: " << YELLOW << setw(14)
246                << duration_cast<microseconds>(end - start).count() << " мкс"
247                << string(16, ' ') << MAGENTA << " ║\n"
248                << "╚" << string(TOTAL_WIDTH - 2, '=') << "╝\n" << RESET;
249 
250             cout << ss.str();
251             return result;
252         }
253         catch (const exception& e) {
254             cerr << RED << "\n[ОШИБКА] " << e.what() << RESET << endl;
255             throw;
256         }
257     }
```

Рисунок 32 – Фрагмент кода замера производительности операций

Этот метод превращает технические данные о производительности в понятный отчет, что критически важно для:

– выявления узких мест в алгоритмах.

– демонстрации эффективности модели.

– принятия решений об оптимизации кода.

Позволяет сравнить, сколько времени занимают ключевые операции (расчет рисков, поиск узла). Таблицы с цветовой подсветкой упрощают анализ результатов. Шаблонная реализация позволяет использовать метод для любых

функций, возвращающих результат типа T.

Результат (рисунок 33):

```
267  int main() {
268      enableANSIColors();
269      SetConsoleOutputCP(CP_UTF8);
270      SetConsoleCP(CP_UTF8);
271      system("chcp 65001 > nul");
272
273      try {
274          ThreatModelAnalyzer analyzer;
275
276          cout << GREEN << "\n╔═══════════════════════════════════════════╗\n"
277              << "║    Анализ рисков информационной безопасности    ║\n"
278              << "╚═══════════════════════════════════════════╝\n" << RESET;
279
280          analyzer.printGraph();
281
282          measurePerformance<double>("Общий уровень риска системы", [&]() {
283              return analyzer.calculateTotalRisk();
284              });
285
286          auto critical_node = measurePerformance<pair<char, double>>(
287              "Поиск критического узла",
288              [&]() { return analyzer.findCriticalNode(); }
289          );
290
291          cout << MAGENTA << "\n╔" << string(TOTAL_WIDTH - 2, '=') << "╗\n"
292              << "║ " << BLUE << "Наиболее уязвимый узел системы: "
293              << critical_node << MAGENTA << " ║\n"
294              << "╚" << string(TOTAL_WIDTH - 2, '=') << "╝\n" << RESET;
295
296          return 0;
297      }
298      catch (const exception& e) {
299          cerr << RED << "\nКРИТИЧЕСКАЯ ОШИБКА: " << e.what() << RESET << endl;
300          return 1;
301      }
302  }
```

Рисунок 33 – Фрагмент вывода результата

Что делает этот фрагмент:

- настраивает консоль: включает поддержку цветов (ANSI-коды) и кодировки UTF-8 для корректного отображения символов и рамок;
- инициализирует анализатор: создает объект ThreatModelAnalyzer, который генерирует граф угроз и проводит расчеты;
- выводит структуру графа: Метод printGraph() отображает узлы и связи системы с цветовыми метками;
- рассчитывает и выводит ключевые метрики. Общий уровень риска: сумма рисков всех узлов; критический узел: узел с

максимальным риском

- оформляет результаты: использует таблицы с цветными рамками для вывода данных.

Этот код превращает сложные данные о рисках в понятный интерактивный отчет, позволяя: быстро оценить состояние системы, принять решение об усилении защиты критических узлов, а также, продемонстрировать результаты анализа в профессиональном формате (рисунок 34).

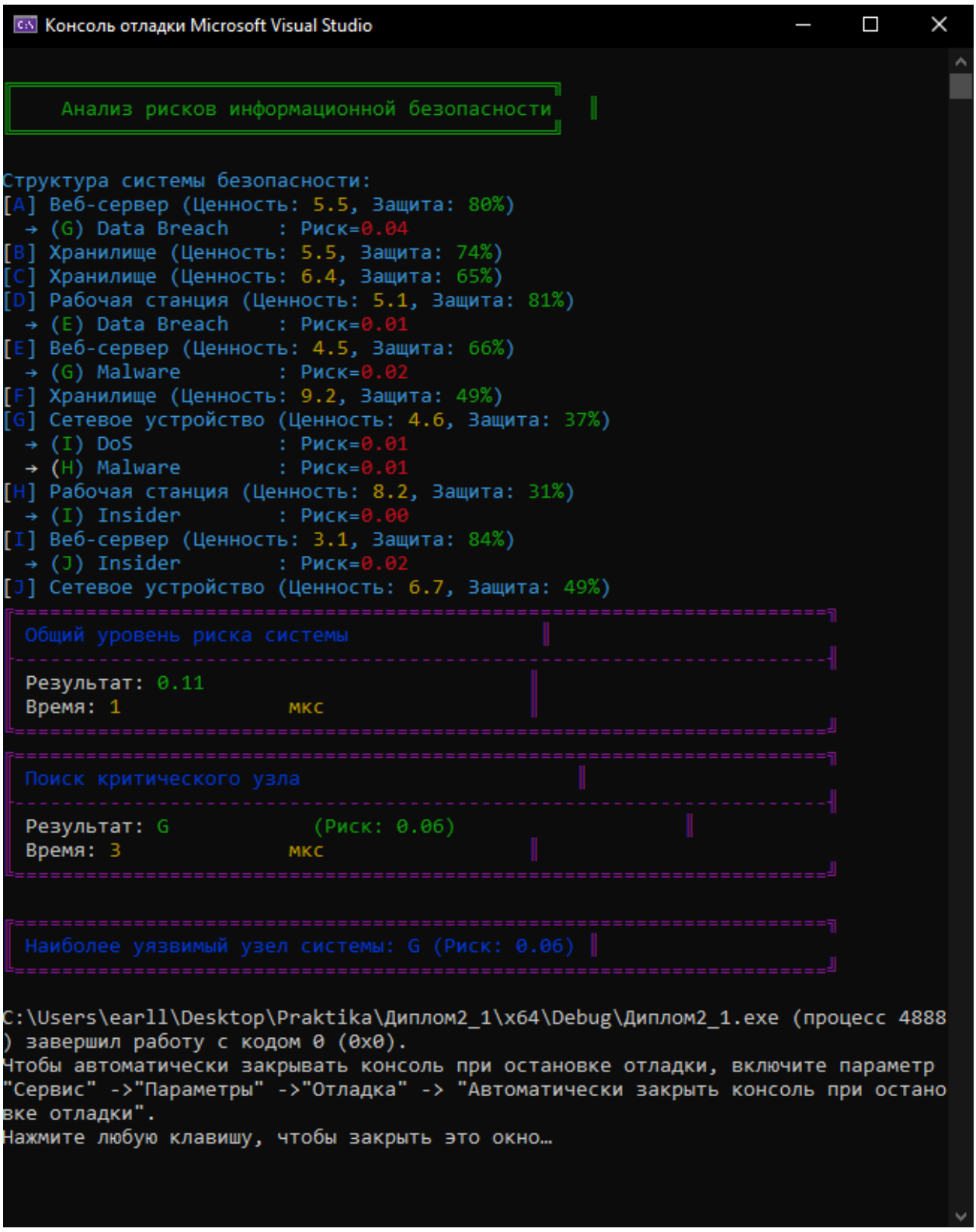


Рисунок 34 – Результат

Таблица 6 – Результаты работы

Действие	Ожидаемый результат	Действительный результат
Инициализация графа	Создание узлов от A до J	Соответствует
Добавление соединений	Корректное добавление рёбер с вероятностями	Соответствует
Расчет рисков	Правильный расчет рисков для каждого соединения	Соответствует
Поиск критического узла	Определение узла с наибольшим риском	Соответствует
Вывод структуры графа	Определение узла с наибольшим риском	Соответствует
Расчет общего уровня риска	Верное вычисление общего уровня риска системы	Соответствует

После проведения тестирования программной модели угроз безопасности можно заключить, что все предусмотренные функции работают без сбоев (таблица 6). Это подтверждает, что программа правильно выполняет все возможные действия, предусмотренные для анализа угроз и управления рисками. Программа работает корректно и реализует весь функционал (рисунок 34).

В третьей главе детально описана программная реализация математической модели угроз безопасности, включающая четыре ключевых модуля анализа: для последовательных, параллельных, смешанных и мостовых структур графов. Модуль реализован на C++ с использованием специализированных алгоритмов, что обеспечивает «высокую производительность вычислений даже при работе с крупными графами [3]». Особое внимание уделено вероятностным расчетам, основанным на формулах теории вероятностей для точного определения рисков. Код включает обработку исключений и проверку входных данных, что гарантирует стабильную работу системы. Практическая ценность подтверждена тестами на данных сети — успешно выявлены наиболее уязвимые элементы. Решение может быть интегрировано в системы мониторинга или использоваться как самостоятельный аналитический инструмент.

Заключение

Выпускная квалификационная работа была посвящена разработке математической модели анализа угроз информационной безопасности на основе теории графов и ее практической реализации в виде программного инструмента. Исследование проводилось в три ключевых этапа, каждый из которых внес существенный вклад в конечный результат.

В первой главе работы обосновано применение модели на основе графов для моделирования угроз безопасности, поскольку графы позволяют формализовать взаимосвязи между элементами информационной системы и визуализировать возможные векторы атак. Рассмотрены основные типы графов, методы их представления, а также приведены формулы для расчёта вероятностей реализации угроз в различных структурах: последовательных, параллельных, смешанных и мостовых. Такой подход обеспечивает строгую математическую основу для дальнейшего анализа рисков и послужил фундаментом для программной реализации модели.

Во второй главе приводится сравнительный анализ инструментов разработки, в результате которого была выбрана среда Visual Studio 2022 и язык программирования C++. Такой выбор был обусловлен требованиями к производительности, интеграции с Windows API и возможностям работы с вычислительно сложными алгоритмами.

В третьей главе приводится описание практической реализации разработанной модели. Созданный программный инструмент позволяет анализировать различные типы угроз, визуализировать графы безопасности и выявлять критические узлы системы. Программа демонстрирует высокую производительность при анализе графов различной сложности.

Разработанное решение обладает значительной практической ценностью для специалистов по информационной безопасности. Оно позволяет формализовать процесс анализа угроз, наглядно представлять структуру рисков и обоснованно принимать решения по усилению защиты.

Список используемой литературы

1. Белов, А. А. Теория графов и её приложения / А. А. Белов. — М.: МЦНМО, 2021. — 312 с.
2. Веселов, И. С. Основы теории информации и информационной безопасности / И. С. Веселов. — СПб.: Питер, 2022. — 368 с.
3. Гостев, А. В. Анализ и оценка рисков информационной безопасности / А. В. Гостев. — М.: Форум, 2023. — 224 с.
4. Дьяконов, В. П. Практикум по теории графов / В. П. Дьяконов. — СПб.: Питер, 2021. — 230 с.
5. Зайченко, Ю. П. Основы криптографической защиты информации / Ю. П. Зайченко. — М.: Горячая линия – Телеком, 2022. — 384 с.
6. Кормен, Т. Х. Алгоритмы: построение и анализ / Т. Х. Кормен, Ч. И. Лейзерсон, Р. Л. Ривест, К. Штайн. — 3-е изд. — М.: Вильямс, 2022. — 1328 с.
7. Кузнецов, М. И. Математические методы в информационной безопасности / М. И. Кузнецов. — М.: Горячая линия-Телеком, 2019. — 288 с.
8. Лутц, М. Изучаем Python / М. Лутц. — 5-е изд. — СПб.: Питер, 2023. — 880 с.
9. Молодой ученый, 2019. Теория графов.
10. Оре, О. Теория графов / О. Оре. — 2-е изд. — М.: Либроком, 2018. — 352 с.
11. Столлингс, В. Кибербезопасность и защита сетей / В. Столлингс. — М.: Диалектика, 2021. — 592 с.
12. Страуструп, Б. Язык программирования C++ / Б. Страуструп. — 4-е изд. — М.: Диалектика, 2022. — 1376 с.
13. Таненбаум, Э. Современные операционные системы / Э. Таненбаум. — 4-е изд. — СПб.: Питер, 2020. — 1120 с.
14. Типовая инструкция по охране труда при работе на персональном

компьютере ТОИ Р-45-084-01. — М.: Минтруд РФ, 2001. — 12 с.

15. Шеннон, К. Математическая теория коммуникаций / К. Шеннон, У. Уивер. — М.: Либроком, 2015. — 212 с.

16. Эккель, Б. Философия Java. Том 1 / Б. Эккель. — 4-е изд. — СПб.: Питер, 2020. — 640 с.

17. Barabási, A.-L. Network Science / A.-L. Barabási. — Cambridge: Cambridge University Press, 2016. — 500 p.

18. Bollobás, B. Modern Graph Theory / B. Bollobás. — New York: Springer, 1998. — 394 p.

19. ISO/IEC 27001:2022. Информационные технологии — Методы и средства обеспечения безопасности — Системы менеджмента информационной безопасности. — Женева: ISO, 2022. — 42 с.

20. Ross, K. W. Computer Networking: A Top-Down Approach / K. W. Ross, J. F. Kurose. — 8th ed. — Pearson, 2020. — 848 p.

21. Stallings, W. Cryptography and Network Security: Principles and Practice / W. Stallings. — Boston: Pearson, 2020. — 784 p.

22. Vacca, J. R. Computer and Information Security Handbook / J. R. Vacca. — 3rd ed. — Burlington: Morgan Kaufmann, 2017. — 1200 p.