

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический университет
Петра Великого»

Институт компьютерных наук и кибербезопасности
Направление: 02.03.01 Математика и компьютерные науки

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1
по дисциплине Дискретная математика

**Реализация генерации бинарного кода Грея и операций
над мультимножествами на его основе.**

Студент,
группы 5130201/40003

_____ Четвергов И.С

Доцент

_____ Востров А.В

«_____» _____ 20__ г.

Санкт-Петербург, 2025

Содержание

Введение	3
1 Постановка задач	4
2 Математическое описание	5
2.1 Множества	5
2.2 Мультимножества	5
2.3 Бинарный код Грея	5
2.4 Операции над множествами и мультимножествами	5
2.4.1 Логические (теоретико-множественные) операции	5
2.4.2 Арифметические операции	6
3 Особенности реализации	8
3.1 Структура проекта	8
3.2 Структуры данных	8
3.3 Интерфейс класса <code>Multiset</code>	8
3.4 Реализация методов класса <code>Multiset</code>	10
3.4.1 Функция <code>toBinary(int number, int length)</code>	10
3.4.2 Функция <code>generateGrayCode(int n)</code>	10
3.4.3 Метод <code>fillRandomly(int n)</code>	11
3.4.4 Метод <code>fillManually(const Multiset& universe)</code>	12
3.4.5 Метод <code>fillAutomatically(const Multiset& universe, int desiredCardinality)</code>	13
3.4.6 Метод <code>unionWith()</code>	15
3.4.7 Метод <code>difference()</code> и оператор <code>operator-</code>	16
3.4.8 Метод <code>intersectWith()</code> и оператор <code>operator&()</code>	17
3.4.9 Метод <code>symmetricDifference()</code> и оператор <code>operator^()</code>	17
3.4.10 Метод <code>complement()</code>	18
4 Результаты работы программы	23
4.1 Генерация универсума	23
4.2 Заполнение и операции над множествами	23
4.3 Демонстрация работы исключений	26
Заключение	27
5 Список литературы	29

Введение

В данной лабораторной работе реализована программа на языке программирования C++, предназначенная для генерации и выполнения операций над мультимножествами. Основой для генерации элементов мультимножеств послужил бинарный код Грея, что обеспечило уникальность и последовательную взаимосвязь всех элементов в созданном универсуме.

1 Постановка задач

Целью данной лабораторной работы является практическая реализация алгоритмов дискретной математики на языке программирования C++.

В рамках работы необходимо:

1. Реализовать генерацию универсума мультимножеств на основе бинарного кода Грея заданной разрядности.
2. Спроектировать и реализовать структуру данных для хранения мультимножеств, способную эффективно работать с элементами и их кратностями.
3. Реализовать два способа заполнения мультимножеств на основе универсума:
 - Вручную, с пошаговым вводом данных.
 - Автоматически, с заданием желаемой мощности.
4. Запрограммировать основные операции над мультимножествами:
 - Теоретико-множественные: объединение, пересечение, дополнение, разность, симметрическая разность.
 - Арифметические: сложение, разность, произведение, деление.
5. Разработать удобный и отказоустойчивый пользовательский интерфейс для взаимодействия с программой.

Результатом работы станет программа, демонстрирующая генерацию и операции над мультимножествами, а также отчёт, описывающий теоретические основы и особенности реализации.

2 Математическое описание

2.1 Множества

Множество — это фундаментальное понятие в математике, представляющее собой неупорядоченную совокупность различных элементов. В отличие от мультимножеств, каждый элемент в обычном множестве может встречаться только один раз. Таким образом, можно сказать, что множество является частным случаем мультимножества, где кратность каждого элемента равна 1. Основные операции над множествами, такие как объединение, пересечение и разность, также применимы к мультимножествам, но с учётом кратностей элементов.

2.2 Мультимножества

Мультимножество \hat{X} (над множеством X) называется совокупность элементов множества X , в которую элемент x_i входит $a_i \geq 0$. Мультимножество обозначается одним из следующих способов:

$$\hat{X} = [x_1^{a_1}, \dots, x_n^{a_n}] = \langle x_1, \dots, x_n; \dots; x_n, \dots, x_n \rangle = \langle a_1(x_1), \dots, a_n(x_n) \rangle.$$

2.3 Бинарный код Грея

Бинарный код Грея (отражённый бинарный код) — это такая последовательность двоичных чисел, в которой два соседних числа отличаются друг от друга только одним разрядом. Для генерации бинарного кода Грея заданной разрядности n используется формула:

$$G(i) = i \oplus (i \gg 1) \quad (1)$$

где $G(i)$ — i -е число в коде Грея, i — обычное бинарное представление числа, \oplus — операция побитового исключающего «ИЛИ», и $\gg 1$ — побитовый сдвиг вправо. Элементы мультимножеств в данной работе представлены в виде кодов Грея, где разрядность n определяет количество элементов в универсуме, равное 2^n .

2.4 Операции над множествами и мультимножествами

Для множеств и мультимножеств определяются следующие операции:

2.4.1 Логические (теоретико-множественные) операции

1. **Объединение** ($A \cup B$): Кратность элемента равна максимуму из кратностей в исходных мультимножествах:

$$C = A \cup B = \{x \mid x \in A \vee x \in B\},$$

$$C = A \cup B = \{ \max(a_i x_i, a_j x_j) \}, \quad a_i x_i \in A, \quad a_j x_j \in B.$$

2. **Пересечение** ($A \cap B$): Кратность элемента равна минимуму из кратностей:

$$C = A \cap B = \{ x \mid x \in A \wedge x \in B \},$$

$$C = A \cap B = \{ \min(a_i x_i, a_j x_j) \}, \quad a_i x_i \in A, \quad a_j x_j \in B.$$

3. **Разность** ($A \setminus B$): Кратность элемента равна разности кратностей, но не может быть отрицательной:

$$C = A \setminus B = \{ x \mid x \in A \wedge x \notin B \},$$

$$C = A \setminus B = A \cap \overline{B} = \{ \min(a_i x_i, a_j x_j) \}, \quad a_i x_i \in A, \quad a_j x_j \in \overline{B}.$$

4. **Симметрическая разность** ($A \Delta B$): Кратность элемента равна модулю разности кратностей:

$$C = A \Delta B = (A \cup B) \setminus (A \cap B) = \{ x \mid (x \in A \wedge x \notin B) \vee (x \notin A \wedge x \in B) \},$$

$$C = A \Delta B = \{ a_x \mid a_x = |a_i x_i - a_j x_j| \}, \quad a_i x_i \in A, \quad a_j x_j \in B.$$

5. **Дополнение** (\overline{A}): Дополнение мультимножества A относительно универсума U задаётся как:

$$C = \overline{A} = U \setminus A = \{ x \mid x \notin A \}$$

$$C = \overline{A} = \{ \max(a_i x_i - a_j x_j, 0) \}, \quad a_i x_i \in U, \quad a_j x_j \in A.$$

2.4.2 Арифметические операции

Эти операции используют арифметические правила для вычисления кратности результирующих элементов.

1. **Арифметическая сумма** ($A + B$): Кратность элемента равна сумме кратностей, но не превышает кратность в универсуме:

$$C = A + B = \{ x \mid x \in A \vee x \in B \},$$

$$C = A + B = \{ \min(a_i x_i + a_j x_j, u_k x_k) \}, \quad a_i x_i \in A, \quad a_j x_j \in B, \quad u_k x_k \in U.$$

2. **Арифметическая разность** ($A - B$): Кратность элемента равна разности кратностей, но не может быть отрицательной:

$$C = A - B = \{ x \mid x \in A \wedge x \in B \},$$

$$C = A - B = \{ \max(a_i x_i - a_j x_j, 0) \}, \quad a_i x_i \in A, \quad a_j x_j \in B.$$

3. **Арифметическое произведение ($A \times B$):** Кратность элемента равна произведению кратностей, ограниченному сверху кратностью в универсуме:

$$C = A \times B = \{x \mid x \in A \wedge x \in B\},$$

$$C = A \times B = \{\min(a_i x_i \cdot a_j x_j, u_k x_k)\}, \quad a_i x_i \in A, \quad a_j x_j \in B, \quad u_k x_k \in U.$$

4. **Арифметическое деление ($A \div B$):** Кратность элемента равна целой части от деления кратностей:

$$C = A \div B = \{x \mid x \in A \wedge x \in B\},$$

$$C = A \div B = \begin{cases} \left\{ \left\lfloor \frac{a_i x_i}{a_j x_j} \right\rfloor \right\}, & a_i \in A, \quad a_j x_j \in B, \text{ при } a_j \neq 0 \\ 0, & \text{при } a_j = 0. \end{cases}$$

5. **Арифметическое деление ($A \div B$):** Кратность элемента равна целой части от деления кратностей, ограниченной сверху кратностью в универсуме, и описывается следующей системой:

$$C = A \div B = \begin{cases} \left\{ \min \left(\left\lfloor \frac{a_i x_i}{a_j x_j} \right\rfloor, u_k x_k \right) \right\}, & \text{при } a_j x_j \neq 0 \\ 0, & \text{при } a_j x_j = 0 \end{cases} \quad a_i x_i \in A, \quad a_j x_j \in B$$

Примечание: Если кратность элемента в делителе $a_j x_j$ равна нулю, то кратность этого элемента в множестве C равна нулю:

3 Особенности реализации

Программная реализация проекта выполнена на языке программирования C++ с использованием стандартной библиотеки шаблонов (STL). Весь код был разделён на несколько файлов, каждый из которых отвечает за свой функционал, что обеспечивает модульность и упрощает поддержку.

3.1 Структура проекта

Проект имеет следующую структуру:

- `Multiset.h` и `Multiset.cc`: содержат описание и реализацию класса `Multiset`, который служит основной структурой для хранения данных.
- `Core.h` и `Core.cc`: содержат вспомогательные функции, такие как генерация бинарного кода Грея (`generateGrayCode`) и преобразование чисел в бинарную строку (`toBinary`)
- `UIHandler.h` и `UIHandler.cc`: отвечают за пользовательский интерфейс и взаимодействие с пользователем. Здесь реализовано основное меню программы и обработка пользовательского ввода.
- `IO.h` и `IO.cc`: содержат функции ввода-вывода, включая безопасный ввод целых чисел с проверкой на некорректные данные.
- `Exceptions.h`: содержит определения пользовательских классов исключений для обработки ошибок, что делает программу более надёжной.

3.2 Структуры данных

Для эффективного представления и управления данными мультимножеств в проекте был разработан специализированный класс `Multiset`. Он инкапсулирует в себе все необходимые данные и операции, предоставляя высокоуровневый интерфейс. Внутренняя структура этого класса основана на ассоциативном контейнере `std::map`, который идеально подходит для данной задачи.

3.3 Интерфейс класса `Multiset`

Этот класс содержит следующие ключевые поля, отражающие его структуру:

- `std::map<std::string, int> elements_`: Это основная структура данных, которая представляет мультимножество. `std::map` хранит пары «ключ-значение», где в качестве **ключа** используется строка (`std::string`) для представления уникального **бинарного кода Грея**, а в качестве **значения** — целое число (`int`), хранящее **кратность** этого элемента.

- `long long totalCardinality_`: Данное поле используется для хранения **общей мощности** мультимножества. Его наличие позволяет получать сумму кратностей всех элементов за константное время, избегая повторных итераций по контейнеру при каждом запросе.

```

1  #pragma once
2
3  #include <string>
4  #include <map>
5  #include <iostream>
6  #include <stdexcept>
7
8  class MultisetArithmetic;
9
10 class Multiset {
11 public:
12     Multiset() : totalCardinality_(0) {}
13
14     long long getCardinality() const;
15     const std::map<std::string, int>& getElements() const;
16
17     Multiset operator-(const Multiset& other) const;
18     Multiset operator/(const Multiset& other) const;
19     Multiset operator|(const Multiset& other) const;
20     Multiset operator&(const Multiset& other) const;
21     Multiset operator^(const Multiset& other) const;
22
23     void fillRandomly(int n);
24     void fillManually(const Multiset& universe);
25     void fillAutomatically(const Multiset& universe, int n);
26
27     bool isEmpty() const;
28     void print() const;
29
30     bool letUserUseOp_ = false;
31 private:
32     friend class MultisetArithmetic;
33     std::map<std::string, int> elements_;
34     long long totalCardinality_;
35
36     Multiset unionWith(const Multiset& other) const;
37     Multiset intersectWith(const Multiset& other) const;
38     Multiset difference(const Multiset& other) const;
39     Multiset symmetricDifference(const Multiset& other) const;
40     Multiset complement(const Multiset& universe) const;
41 };

```

Листинг 1: Класс Multiset.h

3.4 Реализация методов класса Multiset

В данном подразделе будет приведено описание реализации ключевых методов, сгруппированных по функциональности. Каждый метод будет сопровождаться кратким пояснением и листингом кода. Но прежде чем углубиться в детали реализации самого класса, важно рассмотреть его фундамент — вспомогательные функции, которые составляют ядро проекта и обеспечивают его надежность.

Функции Core.cc

Этот файл содержит реализацию вспомогательных функций, которые используются в основном приложении для выполнения общих задач.

3.4.1 Функция `toBinary(int number, int length)`

Назначение: Преобразует целое число `number` в его бинарное строковое представление заданной длины `length`.

Вход:

- `int number`: Исходное целое число.
- `int length`: Требуемая длина бинарной строки.

Выход:

- `std::string`: Бинарное представление числа, отформатированное до нужной длины.

3.4.2 Функция `generateGrayCode(int n)`

Назначение: Генерирует вектор, содержащий все бинарные коды Грея заданной разрядности `n`.

Вход:

- `int n`: Разрядность кода Грея.

Выход:

- `std::vector<std::string>`: Вектор строк, где каждая строка — это код Грея.

```
1 #include "Core.h"
2 #include "Exceptions.h"
3
4 #include <string>
5 #include <vector>
6 #include <stdexcept>
7 #include <limits>
8
```

```

9 std::string toBinary(int number, int length) {
10     if (number < 0) {
11         throw InvalidValueException(...);
12     }
13     if (length < 0) {
14         throw InvalidValueException(...);
15     }
16     std::string binaryString = "";
17     for (int i = length - 1; i >= 0; --i) {
18         binaryString += ((number >> i) & 1) ? '1' : '0';
19     }
20     return binaryString;
21 }
22
23 std::vector<std::string> generateGrayCode(int n) {
24     if (n < 0) {
25         throw InvalidValueException(...);
26     }
27
28     if (n > 30) {
29         throw std::out_of_range(...);
30     }
31
32     if (n == 0) {
33         return {};
34     }
35
36     std::vector<std::string> grayCodes;
37     int totalCodes = 1 << n; // 2^n
38
39     grayCodes.reserve(totalCodes);
40
41     for (int i = 0; i < totalCodes; ++i) {
42         int gray_i = i ^ (i >> 1);
43         grayCodes.push_back(toBinary(gray_i, n));
44     }
45
46     return grayCodes;
47 }

```

Листинг 2: Реализация Core.cc

Реализация методов заполнения

3.4.3 Метод fillRandomly(int n)

Назначение: Автоматически заполняет мультимножество, выступающее в роли универсума. Заполнение происходит на основе бинарного кода Грея заданной разрядности n , при этом кратность каждого элемента назначается случайным образом.

Вход:

- `int n`: Разрядность для генерации универсума.

Выход:

- `void`: Метод модифицирует внутреннее состояние объекта, не возвращая значения.

```

1 void Multiset::fillRandomly(int n) {
2     if (n < 0) {
3         throw InvalidValueException("...");
4     }
5     elements_.clear();
6     totalCardinality_ = 0;
7     std::vector<std::string> grayCodes = generateGrayCode(n);
8     if (grayCodes.empty() && n != 0) {
9         throw InvalidOperationException(...);
10    }
11    std::random_device rd;
12    std::mt19937 gen(rd());
13    std::uniform_int_distribution<> distrib(1, 100);
14    for (const auto& code : grayCodes) {
15        int cardinality = distrib(gen);
16        elements_[code] = cardinality;
17        totalCardinality_ += cardinality;
18    }
19 }
```

Листинг 3: Реализация `fillRandomly()`

3.4.4 Метод `fillManually(const Multiset& universe)`

Назначение: Позволяет вручную заполнить мультимножество, задавая кратность для каждого элемента, определённого в универсуме.

Вход:

- `const Multiset& universe`: Универсум, который содержит все возможные элементы и их максимальные кратности.

Выход:

- `void`: Метод модифицирует внутреннее состояние объекта, не возвращая значения.

```

1 void Multiset::fillManually(const Multiset& universe) {
2     elements_.clear();
3     totalCardinality_ = 0;
4     for (const auto& pair : universe.getElements()) {
5         const std::string& code = pair.first;
6         const int max_cardinality = pair.second;
7         while (true) {
8             try {
9                 int current_cardinality = readInteger(...);
```

```

10         if (current_cardinality < 0 ||
11             current_cardinality > max_cardinality) {
12             throw InvalidValueException(...);
13         }
14         if (current_cardinality > 0) {
15             elements_[code] = current_cardinality;
16             totalCardinality_ += current_cardinality;
17         }
18         break;
19     } catch (const InvalidValueException& e) {
20         std::cerr << e.what() << "\n";
21     }
22 }
23 }

```

Листинг 4: Реализация fillManually()

3.4.5 Метод fillAutomatically(const Multiset& universe, int desiredCardin

Назначение: Автоматически заполняет мультимножество, выбирая случайные элементы из универсума до тех пор, пока не будет достигнута заданная желаемая мощность.

Вход:

- `const Multiset& universe`: Универсум, из которого будут выбираться элементы для заполнения.
- `int desiredCardinality`: Желаемая мощность мультимножества.

Выход:

- `void`: Метод модифицирует внутреннее состояние объекта, не возвращая значения.

```

1 void Multiset::fillAutomatically(const Multiset& universe, int
2     desiredCardinality) {
3     elements_.clear();
4     totalCardinality_ = 0;
5     long long universeCardinality = universe.getCardinality();
6     if (desiredCardinality < 0 ||
7         desiredCardinality > universeCardinality) {
8         throw InvalidValueException(...);
9     }
10    if (desiredCardinality == 0) {
11        return;
12    }
13    std::map<std::string, int> temp_elements;
14    for (const auto& pair : universe.getElements()) {
15        temp_elements[pair.first] = 0;
16    }
17    std::random_device rd;
18    std::mt19937 gen(rd());

```

```

18     std::uniform_int_distribution<> distrib(
19         0, universe.getElements().size() - 1
20     );
21     std::vector<std::string> codes;
22     for(const auto& pair : universe.getElements()) {
23         codes.push_back(pair.first);
24     }
25     for (int i = 0; i < desiredCardinality; ++i) {
26         int randomIndex = distrib(gen);
27         const std::string& code = codes[randomIndex];
28
29         if (temp_elements[code] <
30             universe.getElements().at(code)) {
31             temp_elements[code]++;
32         } else {
33             i--;
34         }
35     }
36     for (const auto& pair : temp_elements) {
37         if (pair.second > 0) {
38             elements_[pair.first] = pair.second;
39         }
40     }
41     totalCardinality_ = desiredCardinality;
42 }

```

Листинг 5: Реализация fillAutomatically()

Операции над мультимножествами

Теоретико-множественные операции

Этот раздел описывает логику работы перегруженных операторов и базовых методов, которые были реализованы для выполнения операций над мультимножествами.

3.4.6 Метод `unionWith()`

Назначение: Выполняет операцию объединения мультимножеств. Метод проходит по элементам обоих мультимножеств и для каждого кода устанавливает кратность, равную наибольшему из значений в исходных множествах, формируя результирующее мультимножество.

Вход:

- `const Multiset& other`: Другое мультимножество для объединения.

Выход:

- `Multiset`: Новое мультимножество, являющееся результатом объединения.

```
1 Multiset Multiset::unionWith(const Multiset& other) const {
2     Multiset result;
3
4     for (const auto& [code, a_card] : this->elements_) {
5         int b_card = 0;
6         if (other.elements_.count(code)) {
7             b_card = other.elements_.at(code);
8         }
9         int res_card = std::max(a_card, b_card);
10        if (res_card > 0) {
11            result.elements_[code] = res_card;
12            result.totalCardinality_ += res_card;
13        }
14    }
15
16    for (const auto& [code, b_card] : other.elements_) {
17        if (!this->elements_.count(code)) {
18            if (b_card > 0) {
19                result.elements_[code] = b_card;
20                result.totalCardinality_ += b_card;
21            }
22        }
23    }
24
25    return result;
26 }
27
28 Multiset Multiset::operator|(const Multiset& other) const {
29     return this->unionWith(other);
30 }
```

Листинг 6: Реализация Multiset::unionWith() и operator|()

3.4.7 Метод difference() и оператор operator-()

Назначение: Реализует теоретико-множественную разность мультимножеств. Метод создает копию первого мультимножества, затем вычитает кратности элементов второго. Если в результате вычитания кратность становится меньше или равна нулю, элемент удаляется, что гарантирует неотрицательность итоговой кратности. Оператор `operator-`() служит удобной обёрткой для этого метода.

Вход:

- `const Multiset& other`: Вычитаемое мультимножество.

Выход:

- `Multiset`: Новое мультимножество, являющееся результатом разности.

```

1 Multiset Multiset::difference(const Multiset& other) const {
2     Multiset result = *this;
3     for (const auto& pair : other.elements_) {
4         const std::string& code = pair.first;
5         const int other_cardinality = pair.second;
6         if (result.elements_.count(code)) {
7             int current_cardinality = result.elements_[code];
8             int new_cardinality = current_cardinality -
9                 other_cardinality;
10            if (new_cardinality > 0) {
11                result.elements_[code] = new_cardinality;
12            } else {
13                result.elements_.erase(code);
14            }
15        }
16        result.totalCardinality_ = 0;
17        for (const auto& pair : result.elements_) {
18            result.totalCardinality_ += pair.second;
19        }
20        return result;
21    }
22
23
24 Multiset Multiset::operator-(const Multiset& other) const {
25     return this->difference(other);
26 }
```

Листинг 7: Реализация Multiset::difference() и operator-()

3.4.8 Метод intersectWith() и оператор operator&()

Назначение: Реализует операцию пересечения мультимножеств. В отличие от других операций, этот метод использует существующую логику разности, что демонстрирует эффективное повторное использование кода. Пересечение $A \cap B$ вычисляется по формуле $A \setminus (A \setminus B)$, где сначала из мультимножества A вычитается разность $A \setminus B$. В итоге остаются только те элементы, которые являются общими для обоих мультимножеств.

Вход:

- `const Multiset& other`: Другое мультимножество для пересечения.

Выход:

- `Multiset`: Новое мультимножество, являющееся результатом пересечения.

```
1 Multiset Multiset::intersectWith(const Multiset& other) const {
2     Multiset a_minus_b = this->difference(other);
3     return this->difference(a_minus_b);
4 }
5
6 Multiset Multiset::operator&(const Multiset& other) const {
7     return this->intersectWith(other);
8 }
```

Листинг 8: Реализация `Multiset::intersectWith()` и `operator&()`

3.4.9 Метод symmetricDifference() и оператор operator^()

Назначение: Реализует операцию симметрической разности мультимножеств. Вычисление основано на формуле $(A \setminus B) \cup (B \setminus A)$. Метод вычисляет разности двух мультимножеств, а затем объединяет полученные результаты с помощью метода `unionWith()`.

Вход:

- `const Multiset& other`: Другое мультимножество для операции.

Выход:

- `Multiset`: Новое мультимножество, являющееся результатом симметрической разности.

```
1 Multiset Multiset::symmetricDifference(const Multiset& other)
2     const {
3     Multiset a_minus_b = this->difference(other);
4     Multiset b_minus_a = other.difference(*this);
5     return a_minus_b.unionWith(b_minus_a);
6 }
```

```

7 Multiset Multiset::operator^(const Multiset& other) const {
8     return this->symmetricDifference(other);
9 }

```

Листинг 9: Реализация Multiset::symmetricDifference() и operator^()

3.4.10 Метод complement()

Назначение: Вычисляет дополнение текущего мультимножества относительно заданного универсального множества (**universe**) путём вычитания.

Вход:

- **const Multiset& universe:** Универсальное множество, относительно которого вычисляется дополнение.

Выход:

- **Multiset:** Новое мультимножество, являющееся результатом операции дополнения.

```

1 Multiset Multiset::complement(const Multiset& universe) const {
2     return universe.difference(*this);
3 }

```

Листинг 10: Реализация Multiset::complement()

Арифметические операции

При разработке арифметических операций над мультимножествами была обнаружена проблема с перегрузкой стандартных бинарных операторов (таких как +, -, *, /). в отличие от теоретико-множественных операций, арифметические действия требуют дополнительного параметра — универсального множества, которое задаёт верхний предел кратности для каждого элемента. поскольку перегруженный оператор может принимать только два операнда, возникла необходимость в альтернативном архитектурном решении.

Для решения этой задачи был разработан отдельный класс

MultisetArithmetic. этот класс принимает универсальное множество в своём конструкторе и содержит методы для выполнения всех необходимых арифметических операций. такой подход обеспечивает инкапсуляцию и модульность, отделяя логику вычислений от структуры данных **Multiset**.

Класс MultisetArithmetic

```

1 #pragma once
2
3 #include "Multiset.h"
4

```

```

5 class MultisetArithmetic {
6 public:
7     MultisetArithmetic(const Multiset& universe) :
8         universe_(universe) {}
9
10    Multiset sum(const Multiset& A, const Multiset& B) const;
11    Multiset product(const Multiset& A, const Multiset& B) const;
12    Multiset division(const Multiset& A, const Multiset& B)
13        const;
14    Multiset difference(const Multiset& A, const Multiset& B)
15        const;
16 private:
17     const Multiset& universe_;
18 };

```

Листинг 11: Объявление класса MultisetArithmetic

Метод sum()

Реализует арифметическую сумму мультимножеств. кратность каждого элемента в результате является суммой кратностей из исходных мультимножеств, ограниченной сверху его кратностью в универсуме.

Вход:

- `const Multiset& A`: Первое слагаемое-мультимножество.
- `const Multiset& B`: Второе слагаемое-мультимножество.

Выход:

- `Multiset`: Новое мультимножество, являющееся результатом арифметического сложения.

```

1 Multiset MultisetArithmetic::sum(const Multiset& A, const
2   Multiset& B) const {
3     Multiset result;
4
5     for (const auto& pair_u : universe_.getElements()) {
6         const std::string& code = pair_u.first;
7         const int universe_cardinality = pair_u.second;
8
9         int a_cardinality = 0;
10        if (A.getElements().count(code)) {
11            a_cardinality = A.getElements().at(code);
12        }
13
14        int b_cardinality = 0;
15        if (B.getElements().count(code)) {
16            b_cardinality = B.getElements().at(code);
17        }

```

```

18     int temp_cardinality = a_cardinality + b_cardinality;
19     if (temp_cardinality > universe_cardinality) {
20         temp_cardinality = universe_cardinality;
21     }
22
23     if (temp_cardinality > 0) {
24         result.elements_[code] = temp_cardinality;
25         result.totalCardinality_ += temp_cardinality;
26     }
27 }
28
29 return result;
30 }

```

Листинг 12: Реализация MultisetArithmetic::sum()

Метод product()

Реализует арифметическое произведение мультимножеств. кратность равна произведению кратностей из исходных мультимножеств, ограниченному сверху кратностью в универсуме.

Вход:

- `const Multiset& A`: Первый множитель-мультимножество.
- `const Multiset& B`: Второй множитель-мультимножество.

Выход:

- `Multiset`: Новое мультимножество, являющееся результатом арифметического произведения.

```

1 Multiset MultisetArithmetic::product(const Multiset& A, const
  Multiset& B) const {
2     Multiset result;
3
4     for (const auto& pair_a : A.getElements()) {
5         const std::string& code = pair_a.first;
6         const int a_cardinality = pair_a.second;
7
8         if (B.getElements().count(code)) {
9             const int b_cardinality = B.getElements().at(code);
10            int universe_cardinality =
                universe_.getElements().at(code);
11
12            int temp_cardinality = a_cardinality * b_cardinality;
13            if (temp_cardinality > universe_cardinality) {
14                temp_cardinality = universe_cardinality;
15            }
16
17            if (temp_cardinality > 0) {
18                result.elements_[code] = temp_cardinality;

```

```

19         result.totalCardinality_ += temp_cardinality;
20     }
21 }
22 }
23
24     return result;
25 }

```

Листинг 13: Реализация MultisetArithmetic::product()

Метод division()

Выполняет целочисленное деление кратностей мультимножеств, ограниченное сверху кратностью в универсуме. добавлена проверка деления на ноль.

Вход:

- `const Multiset& A`: Делимое-мультимножество.
- `const Multiset& B`: Делитель-мультимножество.

Выход:

- `Multiset`: Новое мультимножество, являющееся результатом деления.

```

1 Multiset MultisetArithmetic::division(const Multiset& A, const
  Multiset& B) const {
2     Multiset result;
3
4     for (const auto& pair_a : A.getElements()) {
5         const std::string& code = pair_a.first;
6         const int a_cardinality = pair_a.second;
7
8         if (B.getElements().count(code)) {
9             const int b_cardinality = B.getElements().at(code);
10            if (b_cardinality == 0) {
11                throw InvalidOperationException(...);
12            }
13
14            int universe_cardinality =
                universe_.getElements().at(code);
15            int temp_cardinality = a_cardinality / b_cardinality;
16            if (temp_cardinality > universe_cardinality) {
17                temp_cardinality = universe_cardinality;
18            }
19
20            if (temp_cardinality > 0) {
21                result.elements_[code] = temp_cardinality;
22                result.totalCardinality_ += temp_cardinality;
23            }
24        }
25    }
26 }

```

```

27     return result;
28 }

```

Листинг 14: Реализация MultisetArithmetic::division()

Метод difference()

Реализует арифметическую разность. в отличие от обычной разности множеств, допускает отрицательные кратности.

Вход:

- `const Multiset& A`: Уменьшаемое мультимножество.
- `const Multiset& B`: Вычитаемое мультимножество.

Выход:

- `Multiset`: Новое мультимножество, являющееся результатом арифметической разности.

```

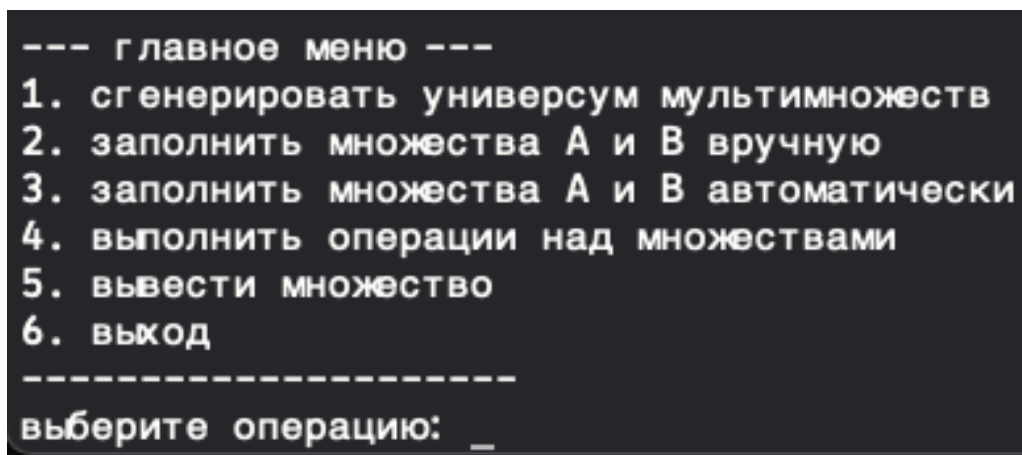
1 Multiset MultisetArithmetic::difference(const Multiset& A, const
  Multiset& B) const {
2     Multiset result = A;
3
4     for (const auto& pair : B.getElements()) {
5         const std::string& code = pair.first;
6         const int other_cardinality = pair.second;
7
8         result.elements_[code] -= other_cardinality;
9     }
10
11     result.totalCardinality_ = 0;
12     for (const auto& pair : result.elements_) {
13         result.totalCardinality_ += pair.second;
14     }
15     return result;
16 }

```

Листинг 15: Реализация MultisetArithmetic::difference()

4 Результаты работы программы

Программа имеет консольный интерфейс (рис 1), позволяющий пользователю выполнять все реализованные операции через меню. На рисунках ниже представлены примеры выполнения основных функций, демонстрирующие корректность работы программы.

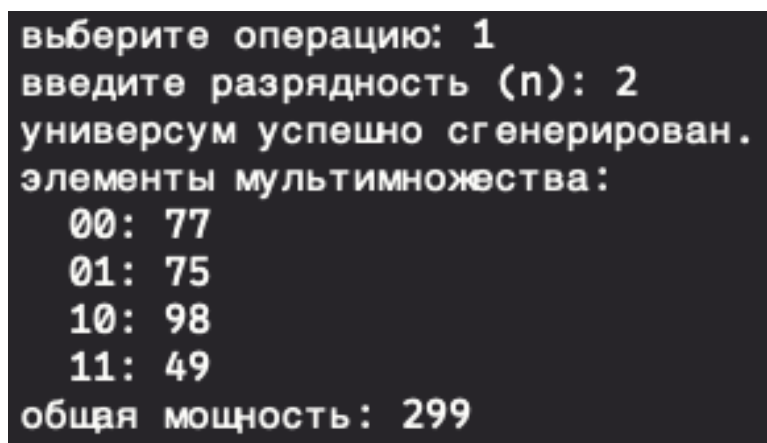


```
--- главное меню ---
1. сгенерировать универсум мультимножеств
2. заполнить множества A и B вручную
3. заполнить множества A и B автоматически
4. выполнить операции над множествами
5. вывести множество
6. выход
-----
выберите операцию: _
```

Рис. 1: Консольный интерфейс

4.1 Генерация универсума

После запуска программы пользователю предлагается выбрать операцию. Опция «1. сгенерировать универсум мультимножеств» позволяет создать универсум на основе кода Грея. Ввод разрядности $n = 2$ приведёт к генерации $2^2 = 4$ элементов. На рисунке 2 показан пример генерации универсума.



```
выберите операцию: 1
введите разрядность (n): 2
универсум успешно сгенерирован.
элементы мультимножества:
00: 77
01: 75
10: 98
11: 49
общая мощность: 299
```

Рис. 2: Генерация универсума с разрядностью 2

4.2 Заполнение и операции над множествами

Пользователь может заполнить мультимножества A и B вручную, автоматически или случайным образом. На рисунке 3 представлен пример автома-

тического заполнения множеств. На рисунке 4 приведён вывод содержимого множеств A и B , после чего доступно подменю операций (рис. 5).

```

--- автоматическое заполнение множества A ---
введите желаемую мощность для A: 123
--- автоматическое заполнение множества B ---
введите желаемую мощность для B: 224

```

Рис. 3: Автоматическое заполнение множеств A и B

<pre> --- множество A --- элементы мультимножества: 00: 31 01: 36 10: 28 11: 28 общая мощность: 123 </pre>	<pre> --- множество B --- элементы мультимножества: 00: 55 01: 56 10: 64 11: 49 общая мощность: 224 </pre>
--	--

Рис. 4: Вывод множеств A и B

```

----- подменю операций -----

1. объединение           (A | B)
2. пересечение           (A & B)
3. дополнение к A        (U \ A)
4. дополнение к B        (U \ B)
5. разность              (A \ B)
6. разность              (B \ A)
7. симм. разность        (A ^ B)
8. сумма                 (A + B)
9. арифм. разность        (A - B)
10. арифм. разность       (B - A)
11. произведение          (A * B)
12. деление               (A / B)
13. деление               (B / A)
14. в главное меню

-----

```

Рис. 5: Меню операций над множествами

На рисунках 13 представлены примеры выполнения теоретико-множественных операций: объединения (6), пересечения (7), дополнения (8, 9), разностей (10, 11) и симметрической разности (12).


```

--- результат объединения ---
элементы мультимножества:
00: 55
01: 56
10: 64
11: 49
общая мощность: 224

```

Рис. 6: Объединение А и В

```

--- результат пересечения ---
элементы мультимножества:
00: 31
01: 36
10: 28
11: 28
общая мощность: 123

```

Рис. 7: Пересечение А и В

```

--- результат дополнения ---
элементы мультимножества:
00: 46
01: 39
10: 70
11: 21
общая мощность: 176

```

Рис. 8: Дополнение к А

```

--- результат дополнения ---
элементы мультимножества:
00: 22
01: 19
10: 34
общая мощность: 75

```

Рис. 9: Дополнение к В

```

--- результат разности ---
элементы мультимножества:
(пустое множество)
общая мощность: 0

```

Рис. 10: Разность А и В

```

--- результат разности ---
элементы мультимножества:
00: 24
01: 20
10: 36
11: 21
общая мощность: 101

```

Рис. 11: Разность В и А

```

элементы мультимножества:
00: 24
01: 20
10: 36
11: 21
общая мощность: 101

```

Рис. 12: Симметрическая разность А и В

Рис. 13: Результаты выполнения теоретико-множественных операций

Аналогично, на рисунках [20](#) представлены примеры выполнения арифметических операций над мультимножествами: сумма ([14](#)), разности ([17](#), [16](#)), произведение ([15](#)), а также деление ([18](#), [19](#)).

```

--- результат арифметической суммы ---
элементы мультимножества:
00: 98
01: 18
10: 45
11: 41
общая мощность: 202

```

Рис. 14: Сумма A и B

```

--- результат арифметического произведения ---
элементы мультимножества:
00: 98
01: 18
10: 45
11: 41
общая мощность: 202

```

Рис. 15: Произведение A и B

```

--- результат арифметической разности (B - A) ---
элементы мультимножества:
00: 42
01: 0
10: 8
11: 8
общая мощность: 58

```

Рис. 16: Разность B и A

```

--- результат арифметической разности (A - B) ---
элементы мультимножества:
00: -42
01: 0
10: -8
11: -8
общая мощность: -58

```

Рис. 17: Разность A и B

```

--- результат арифметического деления (A / B) ---
элементы мультимножества:
01: 1
общая мощность: 1

```

Рис. 18: Деление A на B

```

--- результат арифметического деления (B / A) ---
элементы мультимножества:
00: 2
01: 1
10: 1
11: 1
общая мощность: 5

```

Рис. 19: Деление B на A

Рис. 20: Результаты выполнения арифметических операций над мультимножествами

4.3 Демонстрация работы исключений

Программа надёжно обрабатывает некорректный ввод. Например, при попытке сгенерировать универсум с слишком большой разрядностью или при вводе нечисловых данных, программа выводит сообщение об ошибке, но не прекращает работу, а возвращает пользователя в главное меню. Примеры работы обработчиков исключений приведены на рис. 21.

```

-----
выберите операцию: 1
введите разрядность (n): -5

multiset Error:
invalid Value - отрицательная разрядность недопустима.

выберите операцию: 2
--- заполнение множества A ---
    ведите кратность для кода 0000 (max: 41): 52

multiset Error:
invalid Value - кратность должна быть => 41
    ведите кратность для кода 0000 (max: 41): █

```

Рис. 21: Примеры срабатывания исключений

Заключение

В ходе выполнения данной лабораторной работы была успешно реализована программа на языке C++, предназначенная для работы с мультимножествами.

В результате проделанной работы были получены следующие результаты:

- Разработан и реализован эффективный алгоритм генерации универсума мультимножеств на основе бинарного кода Грея.
- Создана масштабируемая структура данных для хранения мультимножеств на основе ассоциативного контейнера `std::map`, что обеспечивает быстрый доступ к элементам по их уникальным кодам Грея.
- Реализованы два способа заполнения мультимножества, обеспечивающие как полный контроль со стороны пользователя (ручной ввод), так и удобство работы с большими объёмами данных (автоматическое заполнение).
- Успешно запрограммированы теоретико-множественные и арифметические операции над мультимножествами, включая объединение, пересечение, разность, симметрическую разность, а также арифметические сложение, разность и произведение.
- Разработан отказоустойчивый пользовательский интерфейс, надёжно обрабатывающий некорректный ввод данных с использованием механизмов исключений.

Сильные стороны

- Чёткая структура программы и разбиение на классы и модули;
- Наличие обработки ошибок через исключения;
- Инкапсуляция логики вычислений в отдельном классе `MultisetArithmetic`.

Слабые стороны

- Пересчёт мощности (`totalCardinality_`) выполняется не всегда оптимально;
- Тексты сообщений об ошибках иногда не полностью совпадают с условиями проверок.

Масштабируемость

Архитектура проекта обладает хорошей масштабируемостью.

- Легко добавлять новые операции над мультимножествами или изменять существующие, не затрагивая при этом логику других классов. Например, добавление методов для скалирования целого множества или отдельных его элементов.
- Возможно расширение на поддержку других типов ключей, (например, чисел вместо строк). Переход к **шаблонной реализации** `Multiset<T>` позволит использовать целые числа (`int`) или другие типы без изменения основной логики.
- При необходимости обработки больших объемов данных возможна замена `std::map` на `std::map`, который использует сбалансированное красно-черное бинарное дерево поиска и имеет сложность $\Theta(\log N)$ для операций поиска, вставки и удаления, на `std::unordered_map`. Использование хеш-таблицы (`std::unordered_map`) повышает **среднюю** производительность этих же операций до **константной сложности** $\Theta(1)$.
- Проект можно развивать в сторону работы с большим числом операндов (мультимножеств). Реализация через класс, позволяет вычислять такие выражения, $A \cup B \cup C$ путем композиции этих операций, исключая необходимость в создании специфических n-арных алгоритмов..

5 Список литературы

1. Сайт кафедры с учебными материалами по курсу «Дискретная математика». URL: <https://tema.spbstu.ru/dismath/> (дата обращения: 22.09.2025).
2. Новиков Ф.А. *Дискретная математика*. Учебник. Ссылка на PDF: <https://stugum.wordpress.com/wp-content/uploads/2014/03/novikov.pdf> (дата обращения: 16.09.2025).