

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

_____ Адиатуллин Т.Р

Доцент

_____ Востров А.В

«_____» _____ 20 __ г.

Санкт-Петербург, 2025

Содержание

Введение	4
1 Постановка задач	5
2 Математическое описание	6
2.1 Множества	6
2.2 Мультимножества	6
2.3 Бинарный код Грея	6
2.4 Операции над мультимножествами	7
2.4.1 Теоретико-множественные операции	7
2.4.2 Арифметические операции	8
3 Особенности реализации	9
3.1 Структура проекта	9
3.2 Структуры данных	9
3.2.1 Внутренняя структура класса Multiset	9
3.3 Интерфейс класса Multiset	10
3.4 Реализация ключевых функций	10
3.4.1 Функция generateGrayCode(int n)	10
3.4.2 Метод fillUniverse(int n)	11
3.4.3 Метод fillByHand()	12
3.4.4 Метод fillAutomaticly(int desiredCardinality)	13
3.4.5 Метод print(const string& str)	14
4 Реализация операций над мультимножествами	16
4.1 Теоретико-множественные операции	16
4.1.1 Метод Union(const Multiset& other)	16
4.1.2 Метод Intersection(const Multiset& other)	16
4.1.3 Метод Complement()	17
4.1.4 Метод Diff(const Multiset& other)	17
4.1.5 Метод SimmDiff(const Multiset& other)	18
4.2 Арифметические операции	18
4.2.1 Оператор + (арифметическая сумма)	19
4.2.2 Оператор - (арифметическая разность)	19
4.2.3 Оператор * (арифметическое произведение)	20
4.2.4 Оператор / (арифметическое деление)	20
4.3 Вспомогательный метод recount()	21
5 Результаты работы программы	22
5.1 Главное меню программы	22
5.2 Генерация универсума	22
5.3 Заполнение множеств вручную	23
5.4 Автоматическое заполнение множеств	23

5.5	Выполнение операций над множествами	24
5.5.1	Теоретико-множественные операции	24
5.5.2	Арифметические операции	25
5.6	Обработка некорректного ввода	27
5.7	Проверка граничных случаев	28
Заключение		29
6	Список литературы	32

Введение

В рамках данной лабораторной работы была разработана программа на языке C++, которая предназначена для создания и выполнения различных операций над мультимножествами. Для формирования элементов мультимножеств использовался бинарный код Грея, что позволило обеспечить их уникальность и логическую последовательность внутри сформированного универсума

1 Постановка задач

Цель данной лабораторной работы заключается в практической реализации алгоритмов дискретной математики с использованием языка программирования C++.

В рамках работы необходимо:

1. Разработать алгоритм генерации универсума мультимножеств на основе бинарного кода Грея заданной разрядности.
2. Спроектировать и реализовать структуру данных для хранения мультимножеств, обеспечивающую эффективную работу с элементами и их кратностями.
3. Реализовать два способа формирования мультимножеств из универсума:
 - ручной — с последовательным вводом элементов пользователем;
 - автоматический — с указанием требуемой мощности мультимножества.
4. Запрограммировать основные операции над мультимножествами:
 - теоретико-множественные — объединение, пересечение, дополнение, разность и симметрическую разность;
 - арифметические — сложение, вычитание, умножение и деление.
5. Создать удобный и устойчивый к ошибкам интерфейс для взаимодействия пользователя с программой..
6. Провести тестирование программы на различных сценариях использования, включая граничные случаи и некорректный ввод данных.

Результатом работы станет функциональная программа, демонстрирующая генерацию универсума, заполнение мультимножеств и выполнение различных операций над ними, а также подробный отчёт, описывающий теоретические основы, особенности реализации и результаты тестирования разработанного программного обеспечения.

2 Математическое описание

2.1 Множества

Множество — это фундаментальное понятие в математике, представляющее собой неупорядоченную совокупность уникальных элементов. В отличие от мультимножеств, элементы множества различны и отличимы друг от друга. Иными словами, множество можно рассматривать как частный случай мультимножества, в котором кратность каждого элемента равна единице. Операции над множествами, такие как объединение, пересечение и разность, могут быть обобщены и для мультимножеств, с учётом количества повторений элементов.

2.2 Мультимножества

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

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

В данной работе элементы мультимножеств представлены в виде кодов Грея, где разрядность n определяет размер универсума, равный 2^n элементов.

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 ($i = 0, 1, \dots, 2^n - 1$), \oplus — операция побитового исключающего «ИЛИ» (XOR), и $\gg 1$ — побитовый сдвиг вправо на один разряд.

Пример: Для $n = 3$ генерируется последовательность из $2^3 = 8$ кодов Грея:

i (десятичное)	i (двоичное)	$G(i)$ (код Грея)
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

2.4 Операции над мультимножествами

2.4.1 Теоретико-множественные операции

Для мультимножеств определяются следующие логические операции:

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

$$C = A \cup B = \{ \max(a_i, b_i) \cdot x_i \mid x_i \in U \},$$

где a_i — кратность элемента x_i в A , b_i — кратность в B , U — универсум.

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

$$C = A \cap B = \{ \min(a_i, b_i) \cdot x_i \mid x_i \in U \}.$$

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

$$C = A \setminus B = \{ \max(a_i - b_i, 0) \cdot x_i \mid x_i \in U \}.$$

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

$$C = A \Delta B = \{ |a_i - b_i| \cdot x_i \mid x_i \in U \}.$$

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

$$C = \overline{A} = U \setminus A = \{ \max(u_i - a_i, 0) \cdot x_i \mid x_i \in U \},$$

где u_i — кратность элемента x_i в универсуме U .

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, ограниченной сверху кратностью в универсуме:

$$C = A + B = \{ \min(a_i + b_i, u_i) \cdot x_i \mid x_i \in U \}.$$

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

$$C = A - B = \{ \max(a_i - b_i, 0) \cdot x_i \mid x_i \in U \}.$$

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

$$C = A \times B = \{ \min(a_i \cdot b_i, u_i) \cdot x_i \mid x_i \in U \}.$$

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

$$C = A \div B = \begin{cases} \min \left(\left\lfloor \frac{a_i}{b_i} \right\rfloor, u_i \right) \cdot x_i, & \text{если } b_i \neq 0 \\ 0 \cdot x_i, & \text{если } b_i = 0 \end{cases}$$

Примечание: Во всех операциях, если кратность элемента становится равной нулю, он исключается из результирующего мультимножества (не хранится явно).

3 Особенности реализации

Программная часть проекта реализована на языке C++ с применением стандартной библиотеки шаблонов (STL). Код программы структурирован по отдельным файлам, каждый из которых отвечает за определённый участок функционала, что обеспечивает модульность системы и упрощает её дальнейшее сопровождение и развитие.

3.1 Структура проекта

Проект имеет следующую структуру:

- `Gray.hpp` и `Gray.cpp`: содержат функцию генерации бинарного кода Грея.
- `Multiset.hpp` и `Multiset.cpp`: содержат описание и реализацию класса `Multiset`, который служит основной структурой для хранения данных мультимножеств.
- `UI.hpp` и `UI.cpp`: отвечают за пользовательский интерфейс и взаимодействие с пользователем. Здесь реализовано основное меню программы и обработка пользовательского ввода.
- `main.cpp`: точка входа в программу, инициализация объектов и запуск пользовательского интерфейса.
- `Makefile`: автоматизация процесса сборки проекта.

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.

3.2.1 Внутренняя структура класса `Multiset`

Класс содержит следующие ключевые поля:

- `map<string, int> elements`: Основная структура данных для хранения мультимножества. Контейнер `std::map` хранит пары «ключ-значение», где **ключ** — строка с бинарным кодом Грея, а **значение** — целое число, представляющее кратность этого элемента. Использование `std::map` обеспечивает упорядоченное хранение и эффективный доступ к элементам с логарифмической сложностью.

- **int totalCardinality:** Поле для хранения общей мощности мультимножества (суммы всех кратностей). Это позволяет получать мощность за константное время $O(1)$, избегая повторных итераций по контейнеру.
- **static Multiset Universum:** Статическое поле класса, хранящее универсальное множество, общее для всех экземпляров класса **Multiset**.

3.3 Интерфейс класса Multiset

Класс предоставляет следующие публичные методы:

```

1 class Multiset {
2 private:
3     map<string, int> elements;
4     int totalCardinality = 0;
5     void recount();
6
7 public:
8     static Multiset Universum;
9
10    Multiset() : totalCardinality(0) {}
11    int getCardinality() const;
12    const map<string, int>& getElements() const;
13
14    void fillUniverse(int n);
15    void fillByHand();
16    void fillAutomaticly(int n);
17
18    bool isEmpty() const;
19    void print(const string& str) const;
20
21    Multiset Union(const Multiset& other) const;
22    Multiset Intersection(const Multiset& other) const;
23    Multiset Complement() const;
24    Multiset Diff(const Multiset& other) const;
25    Multiset SimmDiff(const Multiset& other) const;
26
27    Multiset operator+(const Multiset& other) const;
28    Multiset operator-(const Multiset& other) const;
29    Multiset operator*(const Multiset& other) const;
30    Multiset operator/(const Multiset& other) const;
31 };

```

Листинг 1: Объявление класса Multiset

3.4 Реализация ключевых функций

3.4.1 Функция generateGrayCode(int n)

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

Вход:

- `int n`: Разрядность кода Грея.

Выход:

- `vector<string>`: Вектор строк с кодами Грея.

Алгоритм: Функция использует формулу $G(i) = i \oplus (i \gg 1)$ для вычисления i -го кода Грея, затем преобразует результат в двоичную строку заданной длины n .

```
1 static inline int grayOf(int x) { return x ^ (x >> 1); }
2
3 vector<string> generateGrayCode(int n){
4     vector<string> gray;
5     int total = 1 << n; // 2^n
6     for (int i = 0; i < total; ++i) {
7         int g = grayOf(i);
8         string code;
9         for (int j = n - 1; j >= 0; --j) {
10             code += ((g >> j) & 1) ? '1' : '0';
11         }
12         gray.push_back(code);
13     }
14     return gray;
15 }
```

Листинг 2: Реализация `generateGrayCode()`

3.4.2 Метод `fillUniverse(int n)`

Назначение: Заполняет универсум мультимножеств на основе кодов Грея разрядности n , присваивая каждому элементу случайную кратность.

Вход:

- `int n`: Разрядность для генерации универсума.

Выход:

- `void`: Метод модифицирует статическое поле `Universe`.

```
1 void Multiset::fillUniverse(int n) {
2     elements.clear();
3     totalCardinality = 0;
4
5     if (n <= 0) {
6         cout << "... \n";
7         return;
8     }
9 }
```

```

10 vector<string> grayCodes = generateGrayCode(n);
11 random_device rb;
12 mt19937 gen(rb());
13 uniform_int_distribution<> dist(1, 50);
14
15 for (const auto& code : grayCodes) {
16     int car = dist(gen);
17     elements[code] = car;
18     totalCardinality += car;
19 }
20 }

```

Листинг 3: Реализация fillUniverse()

3.4.3 Метод fillByHand()

Назначение: Позволяет вручную заполнить мультимножество, запрашивая у пользователя кратность для каждого элемента универсума.

Вход: Нет параметров (взаимодействие с пользователем через консоль).

Выход:

- void: Метод модифицирует объект мультимножества.

Особенности: Метод включает проверку корректности ввода и обработку исключений при вводе некорректных данных.

```

1 void Multiset::fillByHand() {
2     elements.clear();
3     totalCardinality = 0;
4
5     if (Universum.isEmpty()) {
6         cout << "....\n";
7         return;
8     }
9
10    for (const auto& pair : Universum.getElements()) {
11        const string& code = pair.first;
12        const int maxCardinality = pair.second;
13
14        while (true) {
15            cout << "... " << code
16                 << " (max: " << maxCardinality << "): ";
17            int currentCardinality;
18
19            if (!(cin >> currentCardinality)) {
20                cin.clear();
21                cin.ignore(numeric_limits<streamsize>::max(),
22                          '\n');
23                cerr << "....\n";
24                continue;
25            }

```

```

26         try {
27             if (currentCardinality < 0) {
28                 throw invalid_argument("...");
29             }
30             if (currentCardinality > maxCardinality) {
31                 throw out_of_range("... "
32                                     +
33                                     to_string(maxCardinality));
34             }
35             if (currentCardinality > 0) {
36                 elements[code] = currentCardinality;
37                 totalCardinality += currentCardinality;
38             }
39             break;
40         } catch (const exception& e) {
41             cerr << e.what() << "\n";
42         }
43     }
44 }
45 }
46 }

```

Листинг 4: Реализация fillByHand()

3.4.4 Метод fillAutomaticly(int desiredCardinality)

Назначение: Автоматически заполняет мультимножество, выбирая случайные элементы из универсума до достижения заданной мощности.

Вход:

- `int desiredCardinality`: Желаемая мощность мультимножества.

Выход:

- `void`: Метод модифицирует объект мультимножества.

Алгоритм: Метод случайным образом выбирает элементы из универсума и увеличивает их кратность до тех пор, пока общая мощность не достигнет заданного значения.

```

1 void Multiset::fillAutomaticly(int desiredCardinality) {
2     elements.clear();
3     totalCardinality = 0;
4
5     if (Universum.isEmpty()) {
6         cout << "....\n";
7         return;
8     }
9
10    int universeCardinality = Universum.getCardinality();
11

```

```

12     if (desiredCardinality < 0 || desiredCardinality >
13         universeCardinality) {
14         cerr << "... "
15             << universeCardinality << ".\n";
16         return;
17     }
18     if (desiredCardinality == 0) {
19         cout << "....\n";
20         return;
21     }
22
23     vector<string> codes;
24     for (const auto& [code, _] : Universum.getElements()) {
25         codes.push_back(code);
26     }
27
28     random_device rd;
29     mt19937 gen(rd());
30     uniform_int_distribution<> dist(0, (int)codes.size() - 1);
31     map<string, int> temp;
32
33     int added = 0;
34     while (added < desiredCardinality) {
35         const string& code = codes[dist(gen)];
36         if (temp[code] < Universum.getElements().at(code)) {
37             temp[code]++;
38             ++added;
39         }
40     }
41
42     elements = move(temp);
43     totalCardinality = desiredCardinality;
44 }

```

Листинг 5: Реализация fillAutomatically()

3.4.5 Метод print(const string& str)

Назначение: Выводит содержимое мультимножества в консоль с указанием кодов Грея и их кратностей.

Вход:

- `const string& str`: Заголовок для вывода.

Выход:

- `void`: Вывод осуществляется в стандартный поток вывода.

```

1 void Multiset::print(const string& str) const {
2     cout << str;

```

```

3     if (elements.empty()) {
4         cout << "...)\n";
5         return;
6     }
7
8     for (auto& [code, count] : elements)
9         cout << code << " : " << count << endl;
10
11     cout << "...: " << totalCardinality << endl;
12 }

```

Листинг 6: Реализация print()

4 Реализация операций над мультимножествами

В данном разделе приводится описание реализации теоретико-множественных и арифметических операций над мультимножествами.

4.1 Теоретико-множественные операции

4.1.1 Метод Union(const Multiset& other)

Назначение: Выполняет операцию объединения двух мультимножеств. Для каждого элемента универсума кратность в результате устанавливается равной максимальной из кратностей в исходных множествах.

Вход:

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

Выход:

- `Multiset`: Новое мультимножество — результат объединения.

```
1 Multiset Multiset::Union(const Multiset& other) const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.getElements()) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         int cntB = other.elements.count(el) ?
6             other.elements.at(el) : 0;
7         result.elements[el] = max(cntA, cntB);
8     }
9     result.recount();
10    return result;
}
```

Листинг 7: Реализация Union()

4.1.2 Метод Intersection(const Multiset& other)

Назначение: Реализует операцию пересечения мультимножеств. Кратность каждого элемента в результате равна минимуму из кратностей в исходных множествах.

Вход:

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

Выход:

- `Multiset`: Новое мультимножество — результат пересечения.


```

1 Multiset Multiset::Intersection(const Multiset& other) const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.getElements()) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         int cntB = other.elements.count(el) ?
6             other.elements.at(el) : 0;
7         result.elements[el] = min(cntA, cntB);
8     }
9     result.recount();
10    return result;
11 }

```

Листинг 8: Реализация Intersection()

4.1.3 Метод Complement()

Назначение: Вычисляет дополнение мультимножества относительно универсума. Кратность каждого элемента в результате равна разности между кратностью в универсуме и кратностью в исходном множестве.

Вход: Нет параметров (используется статический универсум).

Выход:

- Multiset: Новое мультимножество — дополнение текущего.

```

1 Multiset Multiset::Complement() const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.elements) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         result.elements[el] = max(0, cntU - cntA);
6     }
7     result.recount();
8     return result;
9 }

```

Листинг 9: Реализация Complement()

4.1.4 Метод Diff(const Multiset& other)

Назначение: Реализует теоретико-множественную разность мультимножеств. Кратность элемента в результате равна разности кратностей, но не может быть отрицательной.

Вход:

- const Multiset& other: Вычитаемое мультимножество.

Выход:

- Multiset: Новое мультимножество — результат разности.

```

1 Multiset Multiset::Diff(const Multiset& other) const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.getElements()) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         int cntB = other.elements.count(el) ?
6             other.elements.at(el) : 0;
7         result.elements[el] = max(0, cntA - cntB);
8     }
9     result.recount();
10    return result;
11 }

```

Листинг 10: Реализация Diff()

4.1.5 Метод SimmDiff(const Multiset& other)

Назначение: Реализует операцию симметрической разности мультимножеств. Кратность элемента равна абсолютному значению разности кратностей.

Вход:

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

Выход:

- `Multiset`: Новое мультимножество — результат симметрической разности.

```

1 Multiset Multiset::SimmDiff(const Multiset& other) const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.getElements()) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         int cntB = other.elements.count(el) ?
6             other.elements.at(el) : 0;
7         result.elements[el] = abs(cntA - cntB);
8     }
9     result.recount();
10    return result;
11 }

```

Листинг 11: Реализация SimmDiff()

4.2 Арифметические операции

В отличие от теоретико-множественных операций, арифметические операции выполняют вычисления с кратностями элементов, используя стандартные арифметические правила. Все операции ограничены сверху кратностью элементов в универсуме.

4.2.1 Оператор + (арифметическая сумма)

Назначение: Реализует арифметическое сложение мультимножеств. Кратность элемента в результате равна сумме кратностей, ограниченной сверху кратностью в универсуме.

Вход:

- `const Multiset& other`: Второе слагаемое.

Выход:

- `Multiset`: Новое мультимножество — результат сложения.

```
1 Multiset Multiset::operator+(const Multiset& other) const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.getElements()) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         int cntB = other.elements.count(el) ?
6             other.elements.at(el) : 0;
7         result.elements[el] = min(cntA + cntB, cntU);
8     }
9     result.recount();
10    return result;
}
```

Листинг 12: Реализация `operator+()`

4.2.2 Оператор - (арифметическая разность)

Назначение: Выполняет арифметическое вычитание мультимножеств. Кратность элемента равна разности кратностей, но не может быть отрицательной.

Вход:

- `const Multiset& other`: Вычитаемое.

Выход:

- `Multiset`: Новое мультимножество — результат вычитания.

```
1 Multiset Multiset::operator-(const Multiset& other) const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.getElements()) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         int cntB = other.elements.count(el) ?
6             other.elements.at(el) : 0;
7         result.elements[el] = max(0, cntA - cntB);
8     }
9     result.recount();
10    return result;
}
```

Листинг 13: Реализация `operator-()`

4.2.3 Оператор * (арифметическое произведение)

Назначение: Реализует арифметическое умножение мультимножеств. Кратность элемента равна произведению кратностей, ограниченному сверху кратностью в универсуме.

Вход:

- `const Multiset& other`: Второй множитель.

Выход:

- `Multiset`: Новое мультимножество — результат умножения.

```
1 Multiset Multiset::operator*(const Multiset& other) const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.getElements()) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         int cntB = other.elements.count(el) ?
6             other.elements.at(el) : 0;
7         result.elements[el] = min(cntA * cntB, cntU);
8     }
9     result.recount();
10    return result;
}
```

Листинг 14: Реализация `operator*`()

4.2.4 Оператор / (арифметическое деление)

Назначение: Выполняет целочисленное деление кратностей мультимножеств. Кратность элемента равна целой части от деления, ограниченной сверху кратностью в универсуме. При делении на ноль кратность устанавливается в 0.

Вход:

- `const Multiset& other`: Делитель.

Выход:

- `Multiset`: Новое мультимножество — результат деления.

```
1 Multiset Multiset::operator/(const Multiset& other) const {
2     Multiset result;
3     for (auto& [el, cntU] : Universum.getElements()) {
4         int cntA = elements.count(el) ? elements.at(el) : 0;
5         int cntB = other.elements.count(el) ?
6             other.elements.at(el) : 0;
7         int div = (cntB > 0) ? (cntA / cntB) : 0;
8         result.elements[el] = min(div, cntU);
9     }
}
```

```

9     result.recount();
10    return result;
11 }

```

Листинг 15: Реализация operator/()

4.3 Вспомогательный метод recount()

Назначение: Пересчитывает общую мощность мультимножества путём суммирования всех кратностей элементов.

Примечание: Данный метод вызывается после каждой операции, изменяющей содержимое мультимножества, чтобы поддерживать корректное значение `totalCardinality`.

```

1 void recount() {
2     totalCardinality = 0;
3     for (auto& [_ , cnt] : elements)
4         totalCardinality += cnt;
5 }

```

Листинг 16: Реализация recount()

5 Результаты работы программы

Программа имеет консольный интерфейс, позволяющий пользователю выполнять все реализованные операции через меню. В данном разделе представлены примеры выполнения основных функций, демонстрирующие корректность работы программы.

5.1 Главное меню программы

При запуске программы пользователю предоставляется главное меню с следующими опциями:

1. Сгенерировать универсум мультимножеств
2. Заполнить множества А и В вручную
3. Заполнить множества А и В автоматически
4. Выполнить операции над множествами
5. Вывести множество
6. Выход

5.2 Генерация универсума

После выбора первой опции пользователь может указать разрядность кода Грея. На рисунке ниже показан пример генерации универсума с разрядностью $n = 3$, что приводит к созданию $2^3 = 8$ элементов.

Пример вывода:

Введите разрядность кода Грея: 3

Сгенерированный универсум:

000 : 23

001 : 47

011 : 12

010 : 31

110 : 19

111 : 8

101 : 34

100 : 25

Общая мощность: 199

Каждому коду Грея присваивается случайная кратность в диапазоне от 1 до 50, что определяет максимальное количество вхождений элемента в любое мультимножество на основе данного универсума.

5.3 Заполнение множеств вручную

При выборе опции ручного заполнения программа последовательно запрашивает кратность для каждого элемента универсума. Пользователь может указать значение от 0 до максимальной кратности элемента в универсуме.

Пример ручного заполнения множества А:

Заполнение множества А вручную:

Введите кратность для кода 000 (max: 23): 10

Введите кратность для кода 001 (max: 47): 20

Введите кратность для кода 011 (max: 12): 5

Введите кратность для кода 010 (max: 31): 0

Введите кратность для кода 110 (max: 19): 15

Введите кратность для кода 111 (max: 8): 8

Введите кратность для кода 101 (max: 34): 12

Введите кратность для кода 100 (max: 25): 18

5.4 Автоматическое заполнение множеств

Автоматическое заполнение позволяет создать мультимножество заданной мощности путём случайного выбора элементов из универсума.

Пример автоматического заполнения:

Введите мощность множества А (0..199): 50

Введите мощность множества В (0..199): 60

Множества А и В сгенерированы!

После генерации можно вывести содержимое множеств:

Множество А:

Множество А:

000 : 8

001 : 12

011 : 3

010 : 7

110 : 10

111 : 2

101 : 5

100 : 3

Общая мощность: 50

Множество В:

Множество В:

000 : 5

001 : 18

011 : 6
010 : 9
110 : 4
111 : 7
101 : 8
100 : 3
Общая мощность: 60

5.5 Выполнение операций над множествами

После заполнения множеств А и В становится доступным меню операций. Рассмотрим примеры выполнения различных операций.

5.5.1 Теоретико-множественные операции

1. Объединение (А В):

А В:
000 : 8
001 : 18
011 : 6
010 : 9
110 : 10
111 : 7
101 : 8
100 : 3
Общая мощность: 69

Как видно, для каждого кода выбрана максимальная кратность из двух множеств (например, для кода 001: $\max(12, 18) = 18$).

2. Пересечение (А В):

А В:
000 : 5
001 : 12
011 : 3
010 : 7
110 : 4
111 : 2
101 : 5
100 : 3
Общая мощность: 41

Для каждого элемента выбрана минимальная кратность (например, для кода 001: $\min(12, 18) = 12$).

3. Дополнение к А ($U \setminus A$):

$U \setminus A$:

000 : 15

001 : 35

011 : 9

010 : 24

110 : 9

111 : 6

101 : 29

100 : 22

Общая мощность: 149

4. Разность ($A \setminus B$):

$A \setminus B$:

000 : 3

001 : 0

011 : 0

010 : 0

110 : 6

111 : 0

101 : 0

100 : 0

Общая мощность: 9

Элементы, где кратность в B больше или равна кратности в A , имеют нулевую кратность в результате.

5. Симметрическая разность ($A \oplus B$):

$A \oplus B$:

000 : 3

001 : 6

011 : 3

010 : 2

110 : 6

111 : 5

101 : 3

100 : 0

Общая мощность: 28

Кратность каждого элемента равна $|\text{кратность в } A - \text{кратность в } B|$.

5.5.2 Арифметические операции

1. Арифметическая сумма ($A + B$):

A + B:

000 : 13

001 : 30

011 : 9

010 : 16

110 : 14

111 : 8

101 : 13

100 : 6

Общая мощность: 109

Кратности складываются, но ограничиваются сверху кратностью в универсуме (например, для 111: $\min(2+7, 8) = 8$).

2. Арифметическая разность (A - B):

A - B:

000 : 3

001 : 0

011 : 0

010 : 0

110 : 6

111 : 0

101 : 0

100 : 0

Общая мощность: 9

Результат совпадает с теоретико-множественной разностью, так как обе операции используют формулу $\max(a - b, 0)$.

3. Арифметическое произведение (A * B):

A * B:

000 : 23

001 : 47

011 : 12

010 : 31

110 : 19

111 : 8

101 : 34

100 : 9

Общая мощность: 183

Кратности перемножаются и ограничиваются сверху универсумом.

4. Арифметическое деление (A / B):

A / B:

000 : 1

001 : 0
011 : 0
010 : 0
110 : 2
111 : 0
101 : 0
100 : 1
Общая мощность: 4

Выполняется целочисленное деление кратностей (например, для 000: $8 / 5 = 1$, для 110: $10 / 4 = 2$).

5.6 Обработка некорректного ввода

Программа включает механизмы обработки ошибок для обеспечения устойчивости к некорректному вводу.

Пример 1: Ввод отрицательной разрядности

Введите разрядность кода Грея: -5

Ошибка: разрядность должна быть положительной или равной нулю!

Пример 2: Ввод нечислового значения

Введите мощность множества A (0..199): abc

Ошибка ввода! Повторите.

Пример 3: Превышение максимальной кратности при ручном заполнении

Введите кратность для кода 000 (max: 23): 50

Ошибка: кратность не может превышать 23

Введите кратность для кода 000 (max: 23): 15

Пример 4: Попытка выполнить операции без генерации универсума

Универсум пуст - множества A и B будут пустыми.

Эти примеры демонстрируют, что программа корректно обрабатывает различные виды некорректного ввода и позволяет пользователю исправить ошибку без завершения работы программы.

5.7 Проверка граничных случаев

Генерация пустого универсума ($n = 0$):

Введите разрядность кода Грея: 0

Создан пустой универсум (разрядность 0)

Автоматическое заполнение с нулевой мощностью:

Введите мощность множества A (0..199): 0

Создано пустое множество.

Программа корректно обрабатывает граничные случаи, создавая пустые множества при необходимости и позволяя продолжить работу.

Заключение

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

Достигнутые результаты

В результате проделанной работы были получены следующие результаты:

- Разработан и реализован эффективный алгоритм генерации универсума мультимножеств на основе бинарного кода Грея с использованием формулы $G(i) = i \oplus (i \gg 1)$.
- Реализованы два способа заполнения мультимножеств: ручной ввод (обеспечивающий полный контроль со стороны пользователя) и автоматическое заполнение (удобное для работы с большими объёмами данных).
- Успешно запрограммированы все основные операции над мультимножествами:
 - Теоретико-множественные: объединение, пересечение, дополнение, разность, симметрическая разность.
 - Арифметические: сложение, вычитание, умножение, деление.
- Разработан отказоустойчивый пользовательский интерфейс с надёжной обработкой некорректного ввода данных через механизмы проверки типов и диапазонов значений.
- Проведено комплексное тестирование программы на различных сценариях, включая граничные случаи и некорректный ввод.

Сильные стороны реализации

- **Модульность архитектуры:** Чёткая структура программы с разделением на отдельные модули (генерация кода Грея, класс мультимножества, пользовательский интерфейс) обеспечивает удобство поддержки и развития проекта.
- **Использование статического универсума:** Применение статического поля класса `Multiset::Universum` позволяет всем экземплярам мультимножеств работать с единым универсумом, что соответствует математической модели и упрощает реализацию операций.
- **Перегрузка операторов:** Реализация арифметических операций через перегрузку стандартных операторов C++ (+, -, *, /) делает код более читаемым и интуитивно понятным.

- **Надёжность:** Программа включает проверки корректности ввода и обработку ошибочных ситуаций, что предотвращает аварийное завершение при некорректном вводе.

Слабые стороны и ограничения

- **Ограничение на разрядность:** Теоретически разрядность кода Грея ограничена размером целочисленных типов и доступной памятью. На практике при больших значениях n (например, $n > 20$) размер универсума становится слишком велик для эффективной обработки.
- **Отсутствие сохранения данных:** Программа не поддерживает сохранение и загрузку мультимножеств из файлов, что ограничивает её применение для работы с постоянными данными.
- **Полная итерация по универсуму:** Во всех операциях выполняется итерация по всем элементам универсума, даже если многие из них имеют нулевую кратность. Это приводит к избыточным вычислениям для разреженных мультимножеств.

Возможности масштабирования и улучшения

Архитектура проекта обладает хорошей масштабируемостью и предоставляет возможности для дальнейшего развития:

1. **Переход к шаблонной реализации:** Обобщение класса `Multiset` до шаблонного `Multiset<T>` позволит использовать различные типы элементов (целые числа, строки, пользовательские типы) без изменения основной логики.
2. **Замена контейнера:** При необходимости обработки очень больших множеств возможна замена `std::map` (со сложностью $O(\log n)$) на `std::unordered_map` (со средней сложностью $O(1)$), что ускорит операции доступа к элементам.
3. **Добавление новых операций:** Легко добавить методы для скалирования множества (умножение всех кратностей на константу), вычисления степени мощности, проверки включения одного множества в другое и других операций.
4. **Поддержка композиции операций:** Реализация возможности вычисления сложных выражений вида $(A \cup B) \cap C$ или $((A + B) \times C)/D$ через последовательное применение операций.

5. **Сохранение и загрузка данных:** Добавление функционала сериализации мультимножеств в файлы (текстовые или бинарные) для долговременного хранения.
6. **Графический интерфейс:** Разработка GUI с визуализацией множеств и результатов операций (например, с использованием Qt или другой библиотеки).

Выводы

Реализованная программа полностью соответствует поставленным требованиям и демонстрирует практическое применение концепций дискретной математики в программировании. Использование бинарного кода Грея для генерации элементов универсума показало свою эффективность и обеспечило уникальность элементов.

Выбранная архитектура с использованием объектно-ориентированного подхода и стандартной библиотеки C++ обеспечивает баланс между производительностью, читаемостью кода и возможностями для дальнейшего развития. Программа может служить основой для более сложных систем работы с мультимножествами и быть использована в учебных целях для изучения алгоритмов дискретной математики.

Полученный опыт работы с мультимножествами, кодом Грея и реализацией математических операций в виде программного кода является ценным для дальнейшего изучения алгоритмов и структур данных.

6 Список литературы

1. Сайт кафедры с учебными материалами по курсу «Дискретная математика». URL: <https://tema.spbstu.ru/dismath/> (дата обращения: 21.10.2025).
2. Новиков Ф.А. *Дискретная математика*. Учебник. Ссылка на PDF: <https://stugum.wordpress.com/wp-content/uploads/2014/03/novikov.pdf> (дата обращения: 21.10.2025).