

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

Высшая школа технологий и искусственного интеллекта

Направление: 02.03.01 Математика и компьютерные науки

Лабораторная работа № 1 по дисциплине
Дискретная Математика

«Генерация бинарного кода Грея и операции над мультимножествами»

Студент,

группа 5130201/40001

_____ Анари М.

Преподаватель,

_____ Востров А. В.

«_____» _____ 2025 г.

Санкт-Петербург, 2025

Содержание

1	Введение	3
2	Математическое описание	4
2.1	Множества и мультимножества	4
2.2	Операции над мультимножествами	4
2.3	Арифметические операции	5
2.4	Взвешенная арифметика в коде Грея	5
2.5	Алгоритм генерации бинарного кода Грея	6
3	Особенности реализации	7
3.1	Структуры данных	7
3.2	Основные функции и методы	7
3.2.1	Генерация кода Грея	7
3.2.2	Инициализация универсума	8
3.2.3	Операции над мультимножествами	8
3.2.4	Арифметические операции	8
3.2.5	Создание мультимножеств	9
3.3	Функции для взвешенной арифметики кода Грея	9
4	Результаты работы	10
4.1	Сценарий 1: Корректный ввод	10
4.2	Сценарий 2: Некорректный ввод разрядности	11
4.3	Сценарий 3: Некорректный ввод кратности	11
4.4	Сценарий 4: Деление на ноль	12
4.5	Сценарий 5: Демонстрация всех операций	12
4.6	Сценарий 6: Взвешенная арифметика кода Грея	13
5	Заключение	14
5.1	Реализованные возможности	14
5.2	Достоинства реализованного алгоритма	14
5.3	Недостатки и ограничения	14
5.4	Масштабирование	15
5.5	Дополнительная работа: взвешенная арифметика	15
A	Приложение	17
A.1	Заголовочный файл <code>funcs.h</code>	17
A.2	Реализация функций <code>funcs.cpp</code>	19
A.3	Точка входа <code>main.cpp</code>	33

1 Введение

В данной лабораторной работе реализована программа для генерации бинарного кода Грея заданной разрядности и выполнения операций над мультимножествами. Программа позволяет пользователю выбрать способ заполнения мультимножеств (вручную или автоматически), задать их мощности и выполнить различные операции: объединение, пересечение, разность, симметрическую разность, дополнение, а также арифметические операции.

Бинарный код Грея представляет собой систему счисления, в которой два соседних значения отличаются только в одном бите. Это свойство делает код Грея особенно полезным в цифровой электронике, коммуникациях и алгоритмах, где важно минимизировать количество изменений при переходе между состояниями. В контексте данной работы код Грея используется для генерации универсума мультимножеств, обеспечивая систематический и эффективный способ перечисления всех возможных элементов.

Мультимножества являются важным обобщением понятия множества в дискретной математике, позволяя элементам иметь кратность больше единицы. Это делает их особенно полезными для моделирования реальных ситуаций, где элементы могут повторяться с различной частотой. В программе реализованы как теоретико-множественные операции (объединение, пересечение, разность), так и арифметические операции, что позволяет проводить комплексный анализ данных, представленных в виде мультимножеств.

Программа также включает систему валидации входных данных, обеспечивающую корректность работы с мультимножествами и предотвращающую ошибки при выполнении операций. Пользовательский интерфейс спроектирован таким образом, чтобы сделать работу с программой интуитивно понятной как для начинающих, так и для опытных пользователей.

Основные задачи работы:

- Реализация алгоритма генерации бинарного кода Грея
- Создание универсума мультимножеств на основе сгенерированного кода
- Реализация операций над мультимножествами
- Обеспечение защиты от некорректного пользовательского ввода
- Создание удобного пользовательского интерфейса

2 Математическое описание

2.1 Множества и мультимножества

Множество — это совокупность различных элементов, где каждый элемент может принадлежать множеству только один раз.

Мультимножество — это обобщение понятия множества, где элементы могут повторяться. Формально мультимножество M над универсумом U можно определить как функцию $M : U \rightarrow \mathbb{N}_0$, где $M(x)$ — кратность элемента x в мультимножестве.

2.2 Операции над мультимножествами

Для мультимножеств A и B над универсумом U определены следующие операции:

1. **Объединение:** Операция объединения создает новое мультимножество, содержащее максимальную кратность каждого элемента из исходных мультимножеств. Это означает, что если элемент присутствует в обоих мультимножествах, то в результате будет взята наибольшая кратность.

$$(A \cup B)(x) = \max(A(x), B(x))$$

2. **Пересечение:** Операция пересечения формирует мультимножество, содержащее минимальную кратность каждого элемента, присутствующего в обоих исходных мультимножествах. Элементы, отсутствующие хотя бы в одном из мультимножеств, не включаются в результат.

$$(A \cap B)(x) = \min(A(x), B(x))$$

3. **Разность:** Операция разности удаляет из первого мультимножества элементы, присутствующие во втором, с учетом их кратности. Если кратность элемента во втором мультимножестве больше или равна кратности в первом, элемент полностью удаляется.

$$(A \setminus B)(x) = \max(0, A(x) - B(x))$$

4. **Симметрическая разность:** Эта операция объединяет элементы, которые присутствуют только в одном из мультимножеств, исключая общие элементы. Она представляет собой объединение двух разностей мультимножеств.

$$A \Delta B = (A \setminus B) \cup (B \setminus A)$$

5. **Дополнение:** Операция дополнения создает мультимножество, содержащее все элементы универсума, которые отсутствуют в исходном мультимножестве, с их макси-

мально возможной кратностью.

$$\overline{A}(x) = \begin{cases} \max_cardinality(x), & \text{если } A(x) = 0 \\ 0, & \text{если } A(x) > 0 \end{cases}$$

2.3 Арифметические операции

1. **Сумма:** Вычисляет общее количество элементов в мультимножестве, суммируя кратности всех элементов. Эта операция дает представление о "размере" мультимножества с учетом повторений.

$$\sum_{x \in U} A(x)$$

2. **Разность:** Вычисляет разность между суммами кратностей двух мультимножеств, при этом результат всегда неотрицательный. Если разность получается отрицательной, возвращается ноль.

$$\sum_{x \in U} A(x) - \sum_{x \in U} B(x) \text{ (неотрицательная)}$$

3. **Произведение:** Вычисляет произведение всех кратностей элементов мультимножества. Эта операция может быть полезна для анализа распределения элементов в мультимножестве.

$$\prod_{x \in U} A(x)$$

4. **Деление:** Выполняет целочисленное деление суммы кратностей первого мультимножества на сумму кратностей второго. При делении на ноль возвращается ноль с выводом соответствующего сообщения об ошибке.

$$\left\lfloor \frac{\sum_{x \in U} A(x)}{\sum_{x \in U} B(x)} \right\rfloor \text{ (целочисленное)}$$

2.4 Взвешенная арифметика в коде Грея

В дополнение к стандартным операциям над мультимножествами рассматривается *взвешенная арифметика* на основе бинарного кода Грея. Идея состоит в том, чтобы каждому элементу универсума, представленному кодом Грея $g \in G_n$, сопоставить целое число через обратное преобразование Грея и использовать кратности элементов как веса.

Преобразование: Пусть $\text{grayToInt}(g)$ — функция преобразования кода Грея в неотрицательное целое число. Тогда *взвешенная сумма* мультимножества A определяется как

$$\text{WSum}(A) = \sum_{g \in U} A(g) \cdot \text{grayToInt}(g).$$

Аналогично определяются взвешенные разность, произведение и целочисленное деление

на уровне агрегированных значений:

$$\begin{aligned} \text{WDiff}(A, B) &= \max(0, \text{WSum}(A) - \text{WSum}(B)), \\ \text{WProd}(A) &= \prod_{g \in U} (\text{grayToInt}(g))^{A(g)}, \\ \text{WDiv}(A, B) &= \begin{cases} \left\lfloor \frac{\text{WSum}(A)}{\text{WSum}(B)} \right\rfloor, & \text{если } \text{WSum}(B) > 0, \\ 0, & \text{иначе.} \end{cases} \end{aligned}$$

Пример. Для $n = 3$ коды Грея $\{000, 001, 011, 010, 110, 111, 101, 100\}$ соответствуют значениям $\{0, 1, 2, 3, 4, 5, 6, 7\}$. Если $A(001) = 2$ и $A(110) = 1$, то

$$\text{WSum}(A) = 2 \cdot 1 + 1 \cdot 4 = 6.$$

2.5 Алгоритм генерации бинарного кода Грея

Бинарный код Грея — это система счисления, в которой два соседних значения различаются только в одном разряде.

Рекурсивный алгоритм генерации:

1. Для $n = 1$: $G_1 = \{0, 1\}$
2. Для $n > 1$:
 - Создать G_{n-1} рекурсивно
 - Добавить префикс '0' к каждому элементу G_{n-1}
 - Добавить префикс '1' к каждому элементу G_{n-1} в обратном порядке

Формула: $G_n = \{0 + g : g \in G_{n-1}\} \cup \{1 + g : g \in \text{reverse}(G_{n-1})\}$

Пример генерации для $n = 3$:

1. $G_1 = \{0, 1\}$
2. $G_2 = \{00, 01\} \cup \{11, 10\} = \{00, 01, 11, 10\}$
3. $G_3 = \{000, 001, 011, 010\} \cup \{110, 111, 101, 100\} = \{000, 001, 011, 010, 110, 111, 101, 100\}$

Как видно из примера, каждый переход между соседними кодами изменяет только один бит, что является ключевым свойством кода Грея.

3 Особенности реализации

3.1 Структуры данных

Программа использует следующие основные структуры данных:

- `vector<string> universe` — хранит элементы универсума (коды Грея)
- `map<string, int> universeCardinality` — хранит максимальную кратность для каждого элемента универсума
- `map<string, int> multiset1, multiset2` — хранят мультимножества в виде пар (элемент, кратность)

3.2 Основные функции и методы

3.2.1 Генерация кода Грея

Функция: `vector<string> generateGrayCode(int n)`

Входные параметры:

- `n` — разрядность кода Грея

Выходные данные:

- `vector<string>` — вектор строк, содержащий все коды Грея заданной разрядности

Описание: Рекурсивно генерирует все коды Грея для заданной разрядности, используя алгоритм построения по предыдущему коду.

Обоснование выбора структуры данных: Использование `vector<string>` для хранения кодов Грея является оптимальным решением по следующим причинам:

- **Эффективность доступа:** Вектор обеспечивает константное время доступа к элементам по индексу $O(1)$
- **Удобство итерации:** Поддержка range-based for loops и итераторов для удобного обхода всех элементов
- **Динамическое управление памятью:** Автоматическое управление памятью без необходимости ручного выделения/освобождения
- **Совместимость с STL:** Легкая интеграция с алгоритмами стандартной библиотеки C++
- **Строковое представление:** Использование `string` для хранения бинарных кодов позволяет легко манипулировать отдельными битами, добавлять префиксы и выполнять конкатенацию
- **Читаемость:** Строковое представление делает код более понятным и удобным для отладки

3.2.2 Инициализация универсума

Функция: `void initializeUniverse()`

Входные параметры: Нет

Выходные данные: Нет (изменяет состояние объекта)

Описание: Генерирует универсум на основе кода Грея и случайным образом назначает максимальную кратность каждому элементу (от 1 до 10).

3.2.3 Операции над мультимножествами

Объединение: `map<string, int> unionMultisets(const map<string, int>& m1, const map<string, int>& m2)`

Пересечение: `map<string, int> intersectionMultisets(const map<string, int>& m1, const map<string, int>& m2)`

Разность: `map<string, int> differenceMultisets(const map<string, int>& m1, const map<string, int>& m2)`

Симметрическая разность: `map<string, int> symmetricDifferenceMultisets(const map<string, int>& m1, const map<string, int>& m2)`

Дополнение: `map<string, int> complementMultiset(const map<string, int>& multiset)`

Входные параметры:

- `m1`, `m2` — мультимножества для операций
- `multiset` — мультимножество для дополнения

Выходные данные:

- `map<string, int>` — результирующее мультимножество

Описание: Все операции реализованы с учетом ограничений максимальной кратности элементов.

3.2.4 Арифметические операции

Сумма: `int sumMultisets(const map<string, int>& multiset)`

Разность: `int arithmeticDifferenceMultisets(const map<string, int>& m1, const map<string, int>& m2)`

Произведение: `int productMultisets(const map<string, int>& multiset)`

Деление: `int divisionMultisets(const map<string, int>& m1, const map<string, int>& m2)`

Входные параметры:

- `multiset` — мультимножество для вычислений
- `m1`, `m2` — мультимножества для операций

Выходные данные:

- `int` — результат арифметической операции

Описание: Арифметические операции выполняются над суммами кратностей элементов мультимножеств.

3.2.5 Создание мультимножеств

Ручное создание: `void createMultisetManually(map<string, int>& multiset, const string& name)`

Автоматическое создание: `void createMultisetAutomatically(map<string, int>& multiset, const string& name, int cardinality)`

Входные параметры:

- `multiset` — ссылка на мультимножество для заполнения
- `name` — имя мультимножества
- `cardinality` — мощность для автоматического создания

Выходные данные: Нет (изменяет переданное мультимножество)

Описание: Ручное создание позволяет пользователю задать кратность каждого элемента, автоматическое — случайным образом распределяет заданную мощность.

3.3 Функции для взвешенной арифметики кода Грея

В реализации используются вспомогательные функции для преобразования между кодом Грея и целыми числами, а также для выполнения взвешенных арифметических операций:

- `int grayToInt(const string& g)` — преобразует строковый код Грея в целое число с использованием поразрядного вычисления префиксной XOR-суммы.
- `string intToGray(int x, int n)` — преобразует целое число в код Грея длины n через формулу $g = x \oplus (x \gg 1)$ с последующим форматированием до n бит.
- `long long weightedSum(const map<string,int>& A)` — вычисляет $WSum(A)$ как сумму весов `grayToInt(g)` с кратностями элементов.
- `long long weightedDiff(const map<string,int>& A, const map<string,int>& B)` — возвращает $\max(0, WSum(A) - WSum(B))$.
- `long long weightedProd(const map<string,int>& A)` — считает произведение $\prod (\text{grayToInt}(g))^{A(g)}$ с проверкой переполнения при необходимости.
- `long long weightedDiv(const map<string,int>& A, const map<string,int>& B)` — выполняет целочисленное деление $\left\lfloor \frac{WSum(A)}{WSum(B)} \right\rfloor$ с защитой от деления на ноль.

Данные функции используются исключительно для иллюстрации расширенного подхода и не подменяют стандартные операции над мультимножествами; они работают на уровне агрегированных величин, где вес определяется положением элемента в последовательности Грея.

4 Результаты работы

4.1 Сценарий 1: Корректный ввод

Параметры тестирования:

- Разрядность кода Грея: 3
- Максимальная кратность универсума: 5
- Способ создания: Автоматический
- Мощность мультимножеств: 8 и 6

Ожидаемый результат: Программа должна успешно сгенерировать универсум из 8 элементов (коды Грея для 3 разрядов), создать два мультимножества и выполнить все операции.

```
=== Multiset Operations Program ===
Enter bit width for Gray code (1-10): 3
Enter maximum cardinality for universe elements (1-10): 5

Universe (Gray codes with max cardinality):
1. 000 (3)
2. 001 (5)
3. 011 (1)
4. 010 (1)
5. 110 (2)
6. 111 (3)
7. 101 (3)
8. 100 (3)

Choose creation method for Multiset 1:
1. Manual
2. Automatic
Enter choice (1-2): 2
Enter cardinality for Multiset 1: 8

Creating Multiset 1 automatically with cardinality 8:

Choose creation method for Multiset 2:
1. Manual
2. Automatic
Enter choice (1-2): 2
Enter cardinality for Multiset 2: 6

Creating Multiset 2 automatically with cardinality 6:

Multiset 1:
001: 2 (9)
010: 1 (7)
110: 1 (5)
111: 4 (6)

Multiset 2:
000: 1 (1)
001: 2 (9)
101: 1 (1)
111: 2 (6)
```

Рис. 1: Демонстрация работы программы с корректным вводом

4.2 Сценарий 2: Некорректный ввод разрядности

Параметры тестирования:

- Ввод разрядности: 15 (превышает максимально допустимое значение 10)
- Ожидаемое поведение: Запрос повторного ввода

```
Choose mode:
1. Run main program
2. Run tests
Enter choice (1-2): 1
=== Multiset Operations Program ===
Enter bit width for Gray code (1-10): 15
Invalid input! Enter a number between 1 and 10:
```

Рис. 2: Обработка некорректного ввода разрядности

4.3 Сценарий 3: Некорректный ввод кратности

Параметры тестирования:

- Ввод кратности элемента: 12 (превышает максимальную кратность элемента)
- Ожидаемое поведение: Запрос повторного ввода

```
Choose mode:
1. Run main program
2. Run tests
Enter choice (1-2): 1
=== Multiset Operations Program ===
Enter bit width for Gray code (1-10): 15
Invalid input! Enter a number between 1 and 10: 4
Enter maximum cardinality for universe elements (1-10): 5

Universe (Gray codes with max cardinality):
1. 0000 (5)
2. 0001 (4)
3. 0011 (3)
4. 0010 (3)
5. 0110 (3)
6. 0111 (4)
7. 0101 (2)
8. 0100 (4)
9. 1100 (4)
10. 1101 (3)
11. 1111 (4)
12. 1110 (5)
13. 1010 (5)
14. 1011 (3)
15. 1001 (5)
16. 1000 (3)

Choose creation method for Multiset 1:
1. Manual
2. Automatic
Enter choice (1-2): 1

Creating Multiset 1 manually:
First, set the maximum cardinality for each universe element:
Enter max cardinality for 0000 (1-10): 12
Invalid input! Enter a number between 1 and 10:
```

Рис. 3: Обработка некорректного ввода кратности

4.4 Сценарий 4: Деление на ноль

Параметры тестирования:

- Мультимножество 1: содержит элементы
- Мультимножество 2: пустое (сумма кратностей = 0)
- Ожидаемое поведение: Вывод сообщения об ошибке и возврат 0

```
=== Arithmetic Operations ===
Sum of M1: 0
Sum of M2: 3
Arithmetic Difference (M1 - M2): 0
Arithmetic Difference (M2 - M1): 3
Product of M1: 1
Product of M2: 3
Division (M1 / M2): 0
Division (M2 / M1): Division by zero error!
0
```

Рис. 4: Обработка деления на ноль

4.5 Сценарий 5: Демонстрация всех операций

Параметры тестирования:

- Разрядность: 2
- Ручное создание мультимножеств
- Демонстрация всех операций над множествами и арифметических операций

```
Multiset 1:
011: 1 (2)
110: 2 (2)

Multiset 2:
001: 2 (5)
111: 2 (5)

=== Set Operations ===
Union:
001: 2 (5)
011: 1 (2)
110: 2 (2)
111: 2 (5)

Intersection:
Empty multiset

Difference (M1 - M2):
011: 1 (2)
110: 2 (2)

Difference (M2 - M1):
001: 2 (5)
111: 2 (5)

Symmetric Difference:
001: 2 (5)
011: 1 (2)
110: 2 (2)
111: 2 (5)

Complement of M1:
000: 3 (3)
001: 5 (5)
010: 1 (1)
100: 3 (3)
101: 7 (7)
111: 5 (5)

Complement of M2:
000: 3 (3)
010: 1 (1)
011: 2 (2)
100: 3 (3)
101: 7 (7)
110: 2 (2)

=== Arithmetic Operations ===
Sum of M1: 3
Sum of M2: 4
Arithmetic Difference (M1 - M2): 0
Arithmetic Difference (M2 - M1): 1
Product of M1: 2
Product of M2: 4
Division (M1 / M2): 0
Division (M2 / M1): 1

=== Gray-weighted Arithmetic (values derived from Gray ... integer) ===
Weighted Sum of M1: 10
Weighted Sum of M2: 12
Weighted Difference (M1 - M2): -2
Weighted Difference (M2 - M1): 2
Weighted Product of M1: 32.00
Weighted Product of M2: 25.00
Weighted Division (M1 / M2): 0.83
Weighted Division (M2 / M1): 1.20
```

Рис. 5: Результаты выполнения всех операций над мультимножествами

4.6 Сценарий 6: Взвешенная арифметика кода Грея

Параметры тестирования:

- Разрядность кода Грея: 3
- Мультимножество A : несколько элементов с различными кратностями
- Мультимножество B : другой набор элементов

Ожидаемый результат: Корректный расчет $WSum(A)$, $WSum(B)$, а также $WDiff(A, B)$ и $WDiv(A, B)$ (целочисленно).

```
=== Gray-weighted Arithmetic (values derived from Gray → integer) ===  
Weighted Sum of M1: 10  
Weighted Sum of M2: 12  
Weighted Difference (M1 - M2): -2  
Weighted Difference (M2 - M1): 2  
Weighted Product of M1: 32.00  
Weighted Product of M2: 25.00  
Weighted Division (M1 / M2): 0.83  
Weighted Division (M2 / M1): 1.20
```

Рис. 6: Демонстрация взвешенной арифметики на кодах Грея

5 Заключение

5.1 Реализованные возможности

В ходе выполнения лабораторной работы была успешно реализована программа, включающая в себя:

- Алгоритм генерации бинарного кода Грея рекурсивным методом
- Систему создания мультимножеств с ограничениями по максимальной кратности элементов
- Полный набор операций над мультимножествами (объединение, пересечение, разность, симметрическая разность, дополнение)
- Арифметические операции с защитой от некорректных вычислений
- Два способа создания мультимножеств (ручной и автоматический)
- Систему валидации пользовательского ввода
- Удобный пользовательский интерфейс с меню

5.2 Достоинства реализованного алгоритма

- **Корректность:** Все операции реализованы в соответствии с математическими определениями
- **Надежность:** Обеспечена защита от некорректного ввода и ошибочных вычислений
- **Гибкость:** Поддержка различных разрядностей кода Грея (1-10)
- **Удобство использования:** Интуитивно понятный интерфейс с подсказками
- **Производительность:** Эффективные алгоритмы с временной сложностью $O(n)$ для большинства операций

5.3 Недостатки и ограничения

- **Ограниченная разрядность:** Максимальная разрядность кода Грея ограничена 10 битами
- **Память:** При больших разрядностях требуется значительный объем памяти
- **Валидация:** Некоторые типы некорректного ввода могут требовать дополнительной обработки

5.4 Масштабирование

Алгоритм может быть масштабирован для работы с большими разрядностями путем:

- Использования более эффективных структур данных
- Реализации итеративного алгоритма генерации кода Грея
- Добавления поддержки параллельных вычислений
- Оптимизации памяти для хранения больших универсумов

5.5 Дополнительная работа: взвешенная арифметика

В ходе выполнения работы была реализована дополнительная функциональность — взвешенная арифметика на кодах Грея. Данный подход позволяет оценивать мультимножества не только по кратностям, но и с учетом позиции элементов в последовательности Грея (через преобразование `grayToInt`). Это расширяет аналитические возможности программы, позволяя применять агрегированные метрики и сравнения мультимножеств на уровне числовых весов.

Преимущества: естественная интерпретация веса элемента, целочисленные операции, совместимость с существующими структурами данных. Ограничения: возможный рост значений (произведение), чувствительность к выбранному способу взвешивания.

Список литературы

- [1] Новиков Ф.А. Дискретная математика для программистов. 3-е изд. Санкт-Петербург : Питер Пресс, 2009. - 384 с. (Дата обращения 20.09.2025)
- [2] Секция "Телематика" / текст : электронный / URL: <https://tema.spbstu.ru/dismath/> (Дата обращения 20.09.2025)
- [3] Грей, Фрэнк. "Импульсно-коддовая связь." Патент США 2,632,058. 1953.
- [4] Блишард, Уэйн Д. "Теория мультимножеств." Журнал формальной логики Нотр-Дама 30.1 (1989): 36-66.

А Приложение

В данном приложении приведены исходные коды модулей программы.

А.1 Заголовочный файл funcs.h

```
1 #ifndef FUNCS_H
2 #define FUNCS_H
3
4 #include <iostream>
5 #include <vector>
6 #include <string>
7 #include <map>
8 #include <set>
9 #include <algorithm>
10 #include <iomanip>
11 #include <limits>
12 #include <numeric>
13 #include <cstdlib>
14 #include <ctime>
15 #include <random>
16
17 using namespace std;
18
19 class MultisetProgram {
20 private:
21     int bitWidth;
22     vector<string> universe;
23     map<string, int> universeCardinality; // Max cardinality for
24                                         // each universe element
25     map<string, int> multiset1;
26     map<string, int> multiset2;
27 public:
28     MultisetProgram();
29
30     // Gray code generation
31     vector<string> generateGrayCode(int n);
32     void initializeUniverse();
33     void displayUniverse();
34
35     // Multiset creation and display
```

```

36 void createMultisetManually(map<string, int>& multiset, const
    string& name);
37 void createMultisetAutomatically(map<string, int>& multiset,
    const string& name, int cardinality);
38 void displayMultiset(const map<string, int>& multiset, const
    string& name);
39
40 // Set operations
41 map<string, int> unionMultisets(const map<string, int>& m1,
    const map<string, int>& m2);
42 map<string, int> intersectionMultisets(const map<string, int>&
    m1, const map<string, int>& m2);
43 map<string, int> differenceMultisets(const map<string, int>& m1
    , const map<string, int>& m2);
44 map<string, int> symmetricDifferenceMultisets(const map<string,
    int>& m1, const map<string, int>& m2);
45 map<string, int> complementMultiset(const map<string, int>&
    multiset);
46
47 // Arithmetic operations
48 int sumMultisets(const map<string, int>& multiset);
49 int arithmeticDifferenceMultisets(const map<string, int>& m1,
    const map<string, int>& m2);
50 int productMultisets(const map<string, int>& multiset);
51 int divisionMultisets(const map<string, int>& m1, const map<
    string, int>& m2);
52
53 // Gray-weighted arithmetic
54 static int grayToInt(const string& grayBits);
55 long long weightedSum(const map<string, int>& multiset) const;
56 long long weightedDifference(const map<string, int>& m1, const
    map<string, int>& m2) const;
57 long double weightedProduct(const map<string, int>& multiset)
    const;
58 double weightedDivision(const map<string, int>& m1, const map<
    string, int>& m2) const;
59
60 // Main program flow
61 void run();
62
63 // Getters for testing

```

```

64     const vector<string>& getUniverse() const { return universe; }
65     const map<string, int>& getMultiset1() const { return multiset1
        ; }
66     const map<string, int>& getMultiset2() const { return multiset2
        ; }
67     int getBitWidth() const { return bitWidth; }
68 };
69
70 // Test functions
71 void runTests();
72 void testGrayCodeGeneration();
73 void testMultisetOperations();
74 void testArithmeticOperations();
75
76 #endif // FUNCS_H

```

Листинг 1: Содержимое файла funcs.h

A.2 Реализация функций funcs.cpp

```

1 #include "funcs.h"
2 #include <cassert>
3
4 MultisetProgram::MultisetProgram() : bitWidth(0) {}
5
6 // Generate binary Gray code
7 vector<string> MultisetProgram::generateGrayCode(int n) {
8     if (n <= 0) return {" "};
9     if (n == 1) return {"0", "1"};
10
11     vector<string> prev = generateGrayCode(n - 1);
12     vector<string> result;
13
14     // Add 0 prefix to previous codes
15     for (const string& code : prev) {
16         result.push_back("0" + code);
17     }
18
19     // Add 1 prefix to reversed previous codes
20     for (int i = prev.size() - 1; i >= 0; i--) {
21         result.push_back("1" + prev[i]);
22     }

```

```

23
24     return result;
25 }
26
27 // Initialize universe with Gray codes and cardinality
28 void MultisetProgram::initializeUniverse() {
29     universe = generateGrayCode(bitWidth);
30     universeCardinality.clear();
31
32     // Ask for maximum cardinality for the universe
33     int maxUniverseCardinality;
34     cout << "Enter maximum cardinality for universe elements (1-10)
35         : ";
36     while (!(cin >> maxUniverseCardinality) ||
37         maxUniverseCardinality < 1 || maxUniverseCardinality > 10) {
38         cout << "Invalid input! Enter a number between 1 and 10: ";
39         cin.clear();
40         cin.ignore(numeric_limits<streamsize>::max(), '\n');
41     }
42
43     // Assign random cardinality (1 to maxUniverseCardinality) to
44     // each universe element
45     random_device rd;
46     mt19937 g(rd());
47     uniform_int_distribution<int> cardinalityDist(1,
48         maxUniverseCardinality);
49
50     for (const string& element : universe) {
51         universeCardinality[element] = cardinalityDist(g);
52     }
53 }
54
55 // Display universe
56 void MultisetProgram::displayUniverse() {
57     cout << "\nUniverse (Gray codes with max cardinality):\n";
58     for (size_t i = 0; i < universe.size(); i++) {
59         cout << i + 1 << ". " << universe[i] << " (" <<
60             universeCardinality[universe[i]] << ")" << endl;
61     }
62 }

```

```

59 // Manual multiset creation
60 void MultisetProgram::createMultisetManually(map<string, int>&
    multiset, const string& name) {
61     cout << "\nCreating " << name << " manually:\n";
62     multiset.clear();
63
64     // First, set cardinality for each universe element
65     cout << "First, set the maximum cardinality for each universe
        element:\n";
66     for (size_t i = 0; i < universe.size(); i++) {
67         int cardinality;
68         cout << "Enter max cardinality for " << universe[i] << "
            (1-10): ";
69         while (!(cin >> cardinality) || cardinality < 1 ||
            cardinality > 10) {
70             cout << "Invalid input! Enter a number between 1 and
                10: ";
71             cin.clear();
72             cin.ignore(numeric_limits<streamsize>::max(), '\n');
73         }
74         universeCardinality[universe[i]] = cardinality;
75     }
76
77     // Then, set multiplicity for each element
78     cout << "\nNow, set the multiplicity for each universe element
        :\n";
79     for (size_t i = 0; i < universe.size(); i++) {
80         int multiplicity;
81         cout << "Enter multiplicity for " << universe[i] << " (0 to
            " << universeCardinality[universe[i]] << ", 0 to skip):
            ";
82         while (!(cin >> multiplicity) || multiplicity < 0 ||
            multiplicity > universeCardinality[universe[i]]) {
83             cout << "Invalid input! Enter a number between 0 and "
                << universeCardinality[universe[i]] << ": ";
84             cin.clear();
85             cin.ignore(numeric_limits<streamsize>::max(), '\n');
86         }
87         if (multiplicity > 0) {
88             multiset[universe[i]] = multiplicity;
89         }

```

```

90     }
91 }
92
93 // Automatic multiset creation
94 void MultisetProgram::createMultisetAutomatically(map<string, int>&
    multiset, const string& name, int cardinality) {
95     cout << "\nCreating " << name << " automatically with
        cardinality " << cardinality << ":\n";
96     multiset.clear();
97
98     // First, set random cardinality (1-10) for each universe
        element
99     random_device rd;
100     mt19937 g(rd());
101     uniform_int_distribution<int> cardinalityDist(1, 10);
102
103     for (const string& element : universe) {
104         universeCardinality[element] = cardinalityDist(g);
105     }
106
107     // Randomly distribute cardinality among universe elements
108     vector<int> indices(universe.size());
109     iota(indices.begin(), indices.end(), 0);
110     shuffle(indices.begin(), indices.end(), g);
111
112     int remaining = cardinality;
113     for (size_t i = 0; i < indices.size() && remaining > 0; i++) {
114         int maxMultiplicity = universeCardinality[universe[indices[
            i]]];
115         int multiplicity = (i == indices.size() - 1) ? min(
            remaining, maxMultiplicity) :
116             min(rand() % (remaining + 1),
                maxMultiplicity);
117         if (multiplicity > 0) {
118             multiset[universe[indices[i]]] = multiplicity;
119         }
120         remaining -= multiplicity;
121     }
122 }
123
124 // Display multiset

```

```

125 void MultisetProgram::displayMultiset(const map<string, int>&
    multiset, const string& name) {
126     cout << "\n" << name << ":\n";
127     if (multiset.empty()) {
128         cout << "Empty multiset\n";
129         return;
130     }
131     for (const auto& pair : multiset) {
132         cout << pair.first << ": " << pair.second << " (" <<
            universeCardinality[pair.first] << ")" << endl;
133     }
134 }
135
136 // Set operations
137 map<string, int> MultisetProgram::unionMultisets(const map<string,
    int>& m1, const map<string, int>& m2) {
138     map<string, int> result;
139     set<string> allKeys;
140
141     for (const auto& pair : m1) allKeys.insert(pair.first);
142     for (const auto& pair : m2) allKeys.insert(pair.first);
143
144     for (const string& key : allKeys) {
145         int maxCardinality = universeCardinality[key];
146         int unionValue = max(m1.count(key) ? m1.at(key) : 0, m2.
            count(key) ? m2.at(key) : 0);
147         result[key] = min(unionValue, maxCardinality); // Respect
            cardinality limit
148     }
149     return result;
150 }
151
152 map<string, int> MultisetProgram::intersectionMultisets(const map<
    string, int>& m1, const map<string, int>& m2) {
153     map<string, int> result;
154     for (const auto& pair : m1) {
155         if (m2.count(pair.first)) {
156             int maxCardinality = universeCardinality[pair.first];
157             int intersectionValue = min(pair.second, m2.at(pair.
                first));

```

```

158         result[pair.first] = min(intersectionValue,
159                                   maxCardinality); // Respect cardinality limit
160     }
161     return result;
162 }
163
164 map<string, int> MultisetProgram::differenceMultisets(const map<
165 string, int>& m1, const map<string, int>& m2) {
166     map<string, int> result;
167     for (const auto& pair : m1) {
168         int maxCardinality = universeCardinality[pair.first];
169         int diff = pair.second - (m2.count(pair.first) ? m2.at(pair
170             .first) : 0);
171         if (diff > 0) {
172             result[pair.first] = min(diff, maxCardinality); //
173                 Respect cardinality limit
174         }
175     }
176     return result;
177 }
178
179 map<string, int> MultisetProgram::symmetricDifferenceMultisets(
180     const map<string, int>& m1, const map<string, int>& m2) {
181     map<string, int> diff1 = differenceMultisets(m1, m2);
182     map<string, int> diff2 = differenceMultisets(m2, m1);
183     return unionMultisets(diff1, diff2);
184 }
185
186 map<string, int> MultisetProgram::complementMultiset(const map<
187 string, int>& multiset) {
188     map<string, int> result;
189     for (const string& element : universe) {
190         int multiplicity = multiset.count(element) ? multiset.at(
191             element) : 0;
192         int maxCardinality = universeCardinality[element];
193         if (multiplicity == 0) {
194             result[element] = maxCardinality; // Complement uses
195                 actual max cardinality
196         }
197     }
198 }

```



```

191     return result;
192 }
193
194 // Arithmetic operations
195 int MultisetProgram::sumMultisets(const map<string, int>& multiset)
196 {
197     int sum = 0;
198     for (const auto& pair : multiset) {
199         sum += pair.second;
200     }
201     return sum;
202 }
203
204 int MultisetProgram::arithmeticDifferenceMultisets(const map<string
205 , int>& m1, const map<string, int>& m2) {
206     int diff = sumMultisets(m1) - sumMultisets(m2);
207     return max(0, diff); // Ensure non-negative result
208 }
209
210 int MultisetProgram::productMultisets(const map<string, int>&
211 multiset) {
212     int product = 1;
213     for (const auto& pair : multiset) {
214         product *= pair.second;
215     }
216     return product;
217 }
218
219 int MultisetProgram::divisionMultisets(const map<string, int>& m1,
220 const map<string, int>& m2) {
221     int sum2 = sumMultisets(m2);
222     if (sum2 == 0) {
223         cout << "Division by zero error!\n";
224         return 0;
225     }
226     return sumMultisets(m1) / sum2; // Integer division
227 }
228
229 // Gray-weighted arithmetic helpers
230 int MultisetProgram::grayToInt(const string& grayBits) {
231     //Gray code string to integer: prefix XOR method

```

```

228     int result = 0;
229     int bitAccumulator = 0; // current binary bit value as we scan
        MSB->LSB
230     for (char c : grayBits) {
231         int g = (c == '1') ? 1 : 0;
232         bitAccumulator ^= g; // next binary bit =
        previous XOR gray
233         result = (result << 1) | bitAccumulator;
234     }
235     return result;
236 }
237
238 long long MultisetProgram::weightedSum(const map<string, int>&
    multiset) const {
239     long long total = 0;
240     for (const auto& kv : multiset) {
241         const string& gray = kv.first;
242         int multiplicity = kv.second;
243         total += static_cast<long long>(multiplicity) * grayToInt(
            gray);
244     }
245     return total;
246 }
247
248 long long MultisetProgram::weightedDifference(const map<string, int
    >& m1, const map<string, int>& m2) const {
249     return weightedSum(m1) - weightedSum(m2);
250 }
251
252 long double MultisetProgram::weightedProduct(const map<string, int
    >& multiset) const {
253     // Product over (value(gray) ^ multiplicity). Use long double
        to handle growth.
254     long double product = 1.0L;
255     for (const auto& kv : multiset) {
256         const string& gray = kv.first;
257         int multiplicity = kv.second;
258         int value = grayToInt(gray);
259         if (value == 0 && multiplicity > 0) {
260             return 0.0L; // any zero value to positive power makes
                whole product zero

```

```

261     }
262     for (int i = 0; i < multiplicity; i++) {
263         product *= static_cast<long double>(value);
264     }
265 }
266 return product;
267 }
268
269 double MultisetProgram::weightedDivision(const map<string, int>& m1
    , const map<string, int>& m2) const {
270     long long denom = weightedSum(m2);
271     if (denom == 0) {
272         cout << "Division by zero error!\n";
273         return 0.0;
274     }
275     long long numer = weightedSum(m1);
276     return static_cast<double>(numer) / static_cast<double>(denom);
277 }
278
279 // Main menu
280 void MultisetProgram::run() {
281     cout << "=== Multiset Operations Program ===\n";
282
283     // Get bit width
284     cout << "Enter bit width for Gray code (1-10): ";
285     while (!(cin >> bitWidth) || bitWidth < 1 || bitWidth > 10) {
286         cout << "Invalid input! Enter a number between 1 and 10: ";
287         cin.clear();
288         cin.ignore(numeric_limits<streamsize>::max(), '\n');
289     }
290
291     initializeUniverse();
292     displayUniverse();
293
294     // Create multisets
295     int choice;
296     cout << "\nChoose creation method for Multiset 1:\n";
297     cout << "1. Manual\n2. Automatic\n";
298     cout << "Enter choice (1-2): ";
299     while (!(cin >> choice) || (choice != 1 && choice != 2)) {
300         cout << "Invalid input! Enter 1 or 2: ";

```

```

301     cin.clear();
302     cin.ignore(numeric_limits<streamsize>::max(), '\n');
303 }
304
305 if (choice == 1) {
306     createMultisetManually(multiset1, "Multiset 1");
307 } else {
308     int cardinality;
309     cout << "Enter cardinality for Multiset 1: ";
310     while (!(cin >> cardinality) || cardinality < 0) {
311         cout << "Invalid input! Enter a non-negative integer: "
312             ;
313         cin.clear();
314         cin.ignore(numeric_limits<streamsize>::max(), '\n');
315     }
316     createMultisetAutomatically(multiset1, "Multiset 1",
317         cardinality);
318 }
319
320 cout << "\nChoose creation method for Multiset 2:\n";
321 cout << "1. Manual\n2. Automatic\n";
322 cout << "Enter choice (1-2): ";
323 while (!(cin >> choice) || (choice != 1 && choice != 2)) {
324     cout << "Invalid input! Enter 1 or 2: ";
325     cin.clear();
326     cin.ignore(numeric_limits<streamsize>::max(), '\n');
327 }
328
329 if (choice == 1) {
330     createMultisetManually(multiset2, "Multiset 2");
331 } else {
332     int cardinality;
333     cout << "Enter cardinality for Multiset 2: ";
334     while (!(cin >> cardinality) || cardinality < 0) {
335         cout << "Invalid input! Enter a non-negative integer: "
336             ;
337         cin.clear();
338         cin.ignore(numeric_limits<streamsize>::max(), '\n');
339     }
340     createMultisetAutomatically(multiset2, "Multiset 2",
341         cardinality);

```

```

338     }
339
340     // Display multisets
341     displayMultiset(multiset1, "Multiset 1");
342     displayMultiset(multiset2, "Multiset 2");
343
344     // Perform operations
345     cout << "\n=== Set Operations ===\n";
346
347     map<string, int> unionResult = unionMultisets(multiset1,
348         multiset2);
349     displayMultiset(unionResult, "Union");
350
351     map<string, int> intersectionResult = intersectionMultisets(
352         multiset1, multiset2);
353     displayMultiset(intersectionResult, "Intersection");
354
355     map<string, int> diff1Result = differenceMultisets(multiset1,
356         multiset2);
357     displayMultiset(diff1Result, "Difference (M1 - M2)");
358
359     map<string, int> diff2Result = differenceMultisets(multiset2,
360         multiset1);
361     displayMultiset(diff2Result, "Difference (M2 - M1)");
362
363     map<string, int> symDiffResult = symmetricDifferenceMultisets(
364         multiset1, multiset2);
365     displayMultiset(symDiffResult, "Symmetric Difference");
366
367     map<string, int> comp1Result = complementMultiset(multiset1);
368     displayMultiset(comp1Result, "Complement of M1");
369
370     map<string, int> comp2Result = complementMultiset(multiset2);
371     displayMultiset(comp2Result, "Complement of M2");
372
373     cout << "\n=== Arithmetic Operations ===\n";
374     cout << "Sum of M1: " << sumMultisets(multiset1) << endl;
375     cout << "Sum of M2: " << sumMultisets(multiset2) << endl;
376     cout << "Arithmetic Difference (M1 - M2): " <<
377         arithmeticDifferenceMultisets(multiset1, multiset2) << endl;

```

```

372     cout << "Arithmetic Difference (M2 - M1): " <<
        arithmeticDifferenceMultisets(multiset2, multiset1) << endl;
373     cout << "Product of M1: " << productMultisets(multiset1) <<
        endl;
374     cout << "Product of M2: " << productMultisets(multiset2) <<
        endl;
375     cout << "Division (M1 / M2): " << divisionMultisets(multiset1,
        multiset2) << endl;
376     cout << "Division (M2 / M1): " << divisionMultisets(multiset2,
        multiset1) << endl;
377
378     // Gray-weighted arithmetic section
379     cout << "\n== Gray-weighted Arithmetic (values derived from
        Gray integer) ==\n";
380     cout << "Weighted Sum of M1: " << weightedSum(multiset1) <<
        endl;
381     cout << "Weighted Sum of M2: " << weightedSum(multiset2) <<
        endl;
382     cout << "Weighted Difference (M1 - M2): " << weightedDifference
        (multiset1, multiset2) << endl;
383     cout << "Weighted Difference (M2 - M1): " << weightedDifference
        (multiset2, multiset1) << endl;
384     cout << "Weighted Product of M1: " << fixed << setprecision(2)
        << (double)weightedProduct(multiset1) << endl;
385     cout << "Weighted Product of M2: " << fixed << setprecision(2)
        << (double)weightedProduct(multiset2) << endl;
386     cout << "Weighted Division (M1 / M2): " << fixed <<
        setprecision(2) << weightedDivision(multiset1, multiset2) <<
        endl;
387     cout << "Weighted Division (M2 / M1): " << fixed <<
        setprecision(2) << weightedDivision(multiset2, multiset1) <<
        endl;
388 }
389
390 // Test functions
391 void runTests() {
392     cout << "=== Running Tests ===\n\n";
393     testGrayCodeGeneration();
394     testMultisetOperations();
395     testArithmeticOperations();
396     cout << "\n== All Tests Completed ==\n";

```

```

397 }
398
399 void testGrayCodeGeneration() {
400     cout << "Testing Gray Code Generation...\n";
401
402     MultisetProgram program;
403
404     // Test 1-bit Gray code
405     vector<string> gray1 = program.generateGrayCode(1);
406     assert(gray1.size() == 2);
407     assert(gray1[0] == "0");
408     assert(gray1[1] == "1");
409     cout << "    1-bit Gray code: " << gray1[0] << ", " << gray1[1]
        << endl;
410
411     // Test 2-bit Gray code
412     vector<string> gray2 = program.generateGrayCode(2);
413     assert(gray2.size() == 4);
414     assert(gray2[0] == "00");
415     assert(gray2[1] == "01");
416     assert(gray2[2] == "11");
417     assert(gray2[3] == "10");
418     cout << "    2-bit Gray code: " << gray2[0] << ", " << gray2[1]
        << ", " << gray2[2] << ", " << gray2[3] << endl;
419
420     // Test 3-bit Gray code
421     vector<string> gray3 = program.generateGrayCode(3);
422     assert(gray3.size() == 8);
423     cout << "    3-bit Gray code has " << gray3.size() << "
        elements" << endl;
424
425     cout << "Gray Code Generation Tests: PASSED\n\n";
426 }
427
428 void testMultisetOperations() {
429     cout << "Testing Multiset Operations...\n";
430
431     MultisetProgram program;
432
433     // Create test multisets
434     map<string, int> m1 = {"00", 2}, {"01", 1}};

```

```

435     map<string, int> m2 = {{ "01", 3}, {"10", 1}};
436
437     // Test union
438     map<string, int> unionResult = program.unionMultisets(m1, m2);
439     assert(unionResult["00"] == 2);
440     assert(unionResult["01"] == 3);
441     assert(unionResult["10"] == 1);
442     cout << "        Union operation works correctly" << endl;
443
444     // Test intersection
445     map<string, int> intersectionResult = program.
        intersectionMultisets(m1, m2);
446     assert(intersectionResult["01"] == 1);
447     assert(intersectionResult.size() == 1);
448     cout << "        Intersection operation works correctly" << endl;
449
450     // Test difference
451     map<string, int> diffResult = program.differenceMultisets(m1,
        m2);
452     assert(diffResult["00"] == 2);
453     assert(diffResult.size() == 1);
454     cout << "        Difference operation works correctly" << endl;
455
456     cout << "Multiset Operations Tests: PASSED\n\n";
457 }
458
459 void testArithmeticOperations() {
460     cout << "Testing Arithmetic Operations...\n";
461
462     MultisetProgram program;
463
464     // Create test multiset
465     map<string, int> m = {{ "00", 2}, {"01", 3}, {"10", 1}};
466
467     // Test sum
468     int sum = program.sumMultisets(m);
469     assert(sum == 6);
470     cout << "        Sum operation: " << sum << endl;
471
472     // Test product
473     int product = program.productMultisets(m);

```



```

474     assert(product == 6);
475     cout << "        Product operation: " << product << endl;
476
477     // Test arithmetic difference
478     map<string, int> m1 = {"00", 5};
479     map<string, int> m2 = {"01", 2};
480     int diff = program.arithmeticDifferenceMultisets(m1, m2);
481     assert(diff == 3);
482     cout << "        Arithmetic difference operation: " << diff << endl
483           ;
484
485     cout << "Arithmetic Operations Tests: PASSED\n\n";
486 }

```

Листинг 2: Содержимое файла funcs.cpp

A.3 Точка входа main.cpp

```

1  #include "funcs.h"
2
3  int main() {
4      srand(time(0));
5
6      cout << "Choose mode:\n";
7      cout << "1. Run main program\n";
8      cout << "2. Run tests\n";
9      cout << "Enter choice (1-2): ";
10
11     int choice;
12     while (!(cin >> choice) || (choice != 1 && choice != 2)) {
13         cout << "Invalid input! Enter 1 or 2: ";
14         cin.clear();
15         cin.ignore(numeric_limits<streamsize>::max(), '\n');
16     }
17
18     if (choice == 1) {
19         MultisetProgram program;
20         program.run();
21     } else {
22         runTests();
23     }
24

```

```
25     return 0;  
26 }
```

Листинг 3: Содержимое файла main.cpp