МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт Компьютерных наук и кибербезопастности

Высшая школа технологий искусственного интеллекта

Направление 02.03.01 Математика и компьютерные науки

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ 1 КУРСА ДИСКРЕТНОЙ МАТЕМАТИКИ:

Реализация бинарного кода Грея и всевозможные операции над множествами

Обучающийся: _____ Санько В. В.

Руководитель: _____ Востров А. В.

«_____»_____ 20___г.

Санкт-Петербург, 2024

Содержание

Введение

Цель работы состояяла в том, чтобы реализовать программу генерации бинарного кода Грея для заполнения универсума (заданной пользователем разрядности). На основе универсума формируются два мультимножества двумя способами заполнения: вручную и автоматически (выбирает пользователь). Мощность множеств также задает пользователь. Программа должна иметь реализацию таких операций как объединение, пересечение, дополнение, разность, симметрическую разность, арифметическую разность, арифметическое умножение и деление. В результате на экран выводятся результаты всех действий над множествами. Кроме того, требуется реализовать защиту от некорректного пользовательского ввода.

1 Математическое описание

1.1 Код Грея

Код Грея — двоичный код, иначе зеркальный код, он же код с отражением, в котором две «соседние» кодовые комбинации различаются только цифрой в одном двоичном разряде. Иными словами, расстояние Хэмминга между соседними кодовыми комбинациями равно 1.

Пример того, как числа записываются в десятичной системе счисления и как они записываются в бинарном коде Грея:

0 | 000

1 | 001

2 | 011

3 | 010

4 | 110

5 | 111

6 | 101

7 | 100

1.2 Множества

Множество — одно из ключевых понятий математики, представляющее собой набор, совокупность каких-либо (вообще говоря любых) объектов — элементов этого множества. Два множества равны тогда и только тогда, когда содержат в точности одинаковые элементы.

Пример того, как выглядит множество: X = $x_1,\ x_2,\ x_3,\ ...,\ x_n$

1.3 Мультимножества

Пусть $\tilde{X} = \langle a_1(x_1), ..., a_n(x_n)\rangle$ – мультимножество над множеством $X = \{x_1, ..., x_n\}$. Тогда число a_i называется *показателем* элемента x_i, множество X – *носителем* мультимножества $\tilde{X}$, число $m = a_1 + ... + a_n$ – *мощность* мультимножества $\tilde{X}$, а множество $\underline{\tilde{X}} = \{x_i \in X | a_i > 0\}$ называется *составом* мультимножества $\tilde{X}$.

Пример Пусть $\tilde{X} = [a^0 b^3 c^4]$ –мультимножество над множеством $X = \{a, b, c\}$. Тогда $\underline{\tilde{X}} = \{b, c\}$.

Мультимножество $\tilde{X} = \langle a_1(x_1), ..., a_n(x_n)\rangle$ над множеством $X = \{x_1, ..., x_n\}$ называется *индикатором*, если $\forall i \in 1..n (a_i = 0 \vee a_i = 1)$.

Пример Мультимножество $\tilde{X} = \langle b; c\rangle$ является индикатором над множеством $X = \{a, b, c\}$, причем $\underline{\tilde{X}} = \{b, c\}$.

Над мультимножествами определены следующие операции: объединение, пересечение, разность, арифметическая разность, дополнение, симметрическая разность, арифметическая сумма, арифметическое произведение.

1.4 Операции над множествами

1.4.1 Объединение

Для двух мультимножеств A и B, *обьединение* (A ∪ B) будет мультимножеством, содержащим все элементы, присутствующие в A и B, и их кратности будут равны максимуму их кратностей в A и B.

$$C = A \cup B = \{k^{max\{m_{i1},m_{i2}\}} : k^{m_{i1}} \in A \vee k^{m_{i2}} \in B\}$$

1.4.2 Пересечение

Для двух мультимножеств A и B, *пересечение* (A ∩ B) будет мультимножеством, содержащим элементы, которые присутствуют в обоих мультимножествах, причем кратность каждого элемента в пересечении будет минимумом его кратности в A и B.

$$C = A \cap B = \{k^{min\{m_{i1},m_{i2}\}} : k^{m_{i1}} \in A \wedge k^{m_{i2}} \in B\}$$

1.4.3 Дополнение

Дополнение мультимножества относительно универсума представляет собой операцию, в результате которой получается мультимножество, содержащее элементы универсума, не входящие в исходное мультимножество, и их кратности соответствуют кратностям в универсуме.

Дополнение мультимножества A относительно универсума U обозначается как $\overline{A}$ и содержит элементы, присутствующие в U, но отсутствующие в A, с кратностями, соответствующими кратностям в U, то есть кратность элемнтов будет равна сумме кратностей соответсвубщих элементов в универсуме U и мультимножестве A.

$$\overline{A} = \{k^{m_{iU}-m_{i1}} : k^{m_{i1}} \in A \wedge k^{m_{iU}} \in U\}$$

1.4.4 Разность

Для двух мультимножеств A и B, *разность* обозначается как A\B. Результатом этой операции будет мультимножество, содержащее элементы, которые одновременно присутствуют в A и $\overline{B}$, и кратность каждого такого элемента будет равна минимальной кратности мультимножеств A и $\overline{B}$.

$$C = A\backslash B = A \cap \overline{B} = \{k^{min\{m_{i1},m_{i2}\}} : k^{m_{i1}} \in A \wedge k^{m_{i2}} \in \overline{B}\}$$

1.4.5 Симметрическая разность

Симметрическая разность мультимножеств A и B обозначается как A$\triangle$B и представляет собой мультимножество, содержащее элементы, которые присутствуют в A или B, но не в обоих одновременно, то есть состоит их тех элементов мультимножества A и B, кратности которых различны. Кратность результирующего множества равно модулю разности кратностей этих элементов в A и B.

$$C = A \triangle B = \{k^{|m_{i1} - m_{i2}|} : k^{m_{i1}} \in A \wedge k^{m_{i2}} \in B\}$$

1.4.6 Арифметическая разность

Для двух мультимножеств A и B, *арифметическая разность* обозначается как A $-$ B . Результатом данной операции является мультимножество, состоящее из элементов мультимножества A, кратность которых превышает кратность этих же элементов в мультимножестве B. Кратность каждого элемента результирующего мультимножества равна разности кратностей элементов мультимножеств A и B. Но разность не может быть меньше нуля.

$$C = A - B = \{k^{max\{0, m_{i1} - m_{i2}\}} : k^{m_{i1}} \in A \vee k^{m_{i2}} \in B\}$$

1.4.7 Арифметическое произведение и деление

Для двух мультимножеств A и B, *арифметическое произведение* обозначается как A $*$ B. Результатом данной операции является мультимножество, состоящее из элементов, которые присутствуют в каждом из мультимножеств, и их кратность равна произведению кратностей соответствующих элементов в перемножаемых мультимножествах. Но произведение не может превышать кратности в универсуме.

$$C = A * B = \{k^{min\{m_{i1} * m_{i2}, m_{iU}\}} : k^{m_{i1}} \in A \wedge k^{m_{i2}} \in B \wedge k^{m_{iU}} \in U\}$$

Арифметическое деление мультимножеств A и B представляет собой операцию, обозначаемую как A $\div$ B, которая эквивалентна разности мультимножества A и пересечения мультимножества B. Результатом этой операции является мультимножество, состоящее из элементов мультимножества A, кратность которых превышает кратность этих же элементов в мультимножестве B. Кратность каждого элемента результирующего мультимножества равна разности кратностей элементов мультимножеств A и B, но не меньше нуля.

$$C = A \div B = \{k^{max\{0, m_{i1} - m_{i2}\}} : k^{m_{i1}} \in A \wedge k^{m_{i2}} \in B\}$$

1.4.8 Декартово произведение A и B

Декартово произведение мультимножеств A и B, обозначаемое как A×B, представляет собой множество, которое содержит все возможные упорядоченные пары элементов, где первый элемент принадлежит мультимножеству A, а второй элемент принадлежит мультимножеству B. Декартово произведение мультимножества A и B состоит из всех пар (a,b), где a принадлежит мультимножеству A и b принадлежит мультимножеству B.

$$C = A * B = \{(a, b) : a \in A \wedge b \in B\}$$

2 Особенности реализации

2.1 Структура MultiSetElement

Структура содержит 2 переменных типа int, первая из которых отвечает за сам элемент, а вторая хранит кратность этого элемента. Также структура содержит реализацию перегрузки оператора сравнения == . И также расписан конструктор копирования по умолчанию и конструктор содержащий элемент и его кратность на входе(см.рис1).

```
struct MultiSetElement {
    int element;
    int multiplicity;

    MultiSetElement(int elem, int mult) : element(elem), multiplicity(mult) {}\
    MultiSetElement() : element(0), multiplicity(0) {}\
    bool operator==(const MultiSetElement& other) const {
        return element == other.element && multiplicity == other.multiplicity;
    }
};
```

Рис. 1: структура MultisetElement

2.2 Универсум

Универсум создается при помощи цикла for где первым элементам будет 0, а последним $(1 << n) - 1$, что ровняется максимальному значению заданной разрядности (см.рис2). Каждый элемент имеет свою кратность, которая генерерується с помощью функции $generateRandomMultiplicity()$. Элементы хранятся в мультимножестве setU упорядочено друг за другом. Для того, чтобы отсортировать элементы в порядке кода грея необходимо воспользоваться всопмогательной функцией $convertToGrayOrder(setU)$.

После сортировки заполняется двумерный динамический массив box элементом и соответствующей кратностью.

Вход: разрядность.

Выход: Мультимножество и двумерный массив, которые являются представлением универсума.

```
// Создание универсума в виде бинарного кода Грея
cout << endl;
cout << "Универсум (в виде бинарного кода Грея): " << endl;
int randomMultiplicity;
int** box = new int* [1 << n];
for (int i = 0; i < (1 << n); i++) {
    randomMultiplicity = generateRandomMultiplicity();
    setU.insert(MultiSetElement(toGrayCode(i), randomMultiplicity));
}
multiset<MultiSetElement> grayU = convertToGrayOrder(setU);
for (const MultiSetElement& element : grayU) {
    cout << toGrayCodeString(element.element, n) << "|" << element.multiplicity << endl;
    //cout << element.element << "|" << element.multiplicity << endl;
    box[element.element] = new int[2];
    box[element.element][0] = element.element;
    box[element.element][1] = element.multiplicity;
}
cout << endl;
```

Рис. 2: Создание универсума

2.3 Функции используемые в программе

toGrayCode: на вход подается число в десятичной системе счисления, которое нужно перевести в код грея. Описание: Функция принимает десятичное число (decimal) и возвращает его бинарное представление в коде Грея. Реализация: Использует операцию исключающего ИЛИ между числом и его сдвигом вправо на один бит (см.рис3).

Вход: число в десятичной системе счисления.

Выход: число в двоичной системе счисления.

toGrayCodeStrin: Описание: Функция принимает десятичное число (decimal) и разрядность (n), возвращая его бинарное представление в коде Грея в виде строки с заданным количеством разрядов. Реализация: Внутренне использует toGrayCode, конвертирует результат в bitset и возвращает строку (см.рис3).

Вход: десятичное число и разрядность.

Выход: строка, хранящее поэлементно двоичное число.

grayToDecimal: Описание: Функция принимает бинарное представление числа в коде Грея (binary) и возвращает его десятичное значение. Реализация: Использует операцию исключающего ИЛИ между текущим битом и предыдущим для восстановления оригинального значения.

Вход: двоичное число.

Выход: десичное число.

```cpp
// Функция для перевода числа в бинарный код Грея
int toGrayCode(int decimal) {
    return decimal ^ (decimal >> 1);
}

// Функция для преобразования числа в строку бинарного кода Грея с заданной разрядностью n
string toGrayCodeString(int decimal, int n) {
    bitset<32> binary(toGrayCode(decimal));
    return binary.to_string().substr(32 - n);
}

// Функция для преобразования бинарного кода Грея в десятичное значение
int grayToDecimal(const std::bitset<10>& binary) {
    std::bitset<10> gray = binary;
    for (size_t i = 1; i < binary.size(); i++) {
        gray[i] = binary[i] ^ binary[i - 1];
        //cout << gray[i] << " " << binary[i] << " " << binary[i - 1] << endl;
    }
    return static_cast<int>(gray.to_ulong());
}
```

Рис. 3: Основные функции

generateRandomMultiplicity: Описание: Генерирует случайное число (кратность) от 1 до 50. Реализация: Использует генератор случайных чисел и равномерное распределение (см.рис5).

Вход: диапозон кратности чисел.

Выход: кратность элемента.

compareByGrayCode: Описание: Компаратор для сравнения двух элементов по их кодам Грея. Реализация: Использует *toGrayCode* для получения кодов и сравнения их (см.рис5).

Вход: два мультимножества.

Выход: значение true или false.

convertToGrayOrder:

Описание: Преобразует мультимножество элементов в порядок, определенный кодом Грея. Реализация: Копирует элементы в вектор, сортирует вектор с использованием компаратора *compareByGrayCode*, затем создает мультимножество из отсортированного вектора (см.рис4).

Вход: ссылка на мультимножество.

Выход: отсортированное мультимножество.

```cpp
multiset<MultiSetElement> convertToGrayOrder(const multiset<MultiSetElement>& binarySet) {
    // Копируем элементы в вектор
    std::vector<MultiSetElement> grayOrderVector(binarySet.begin(), binarySet.end());

    // Сортируем вектор с использованием компаратора, соответствующего порядку кода Грея
    std::sort(grayOrderVector.begin(), grayOrderVector.end(), compareByGrayCode);

    // Создаем мультимножество из отсортированного вектора
    multiset<MultiSetElement> grayOrderSet(grayOrderVector.begin(), grayOrderVector.end());

    return grayOrderSet;
}
```

Рис. 4: Основные функции

```cpp
struct MultiSetElement {
    int element;
    int multiplicity;

    MultiSetElement(int elem, int mult) : element(elem), multiplicity(mult) {}\
    MultiSetElement() : element(0), multiplicity(0) {}\
    bool operator==(const MultiSetElement& other) const {
        return element == other.element && multiplicity == other.multiplicity;
    }
};

bool operator<(const MultiSetElement& lhs, const MultiSetElement& rhs) {
    return lhs.element < rhs.element;
}

int generateRandomMultiplicity() {
    std::random_device rd;  // Источник случайных чисел
    std::mt19937 gen(rd()); // Генератор случайных чисел
    std::uniform_int_distribution<int> dist(1, 50);

    return dist(gen);
}

bool compareByGrayCode(const MultiSetElement& a, const MultiSetElement& b) {
    // Реализуйте сравнение элементов по коду Грея
    return toGrayCode(a.element) < toGrayCode(b.element);
}
```

Рис. 5: Основные функции

2.4 Объединение

В этом алгоритме мы используем мультимножества (std::multiset) для хранения множеств A и B, а также для хранения результата объединения (unionSet). Мы проходимся по каждому элементу мультимножеств A и B и для каждого элемента в setA: Если элемент уже есть в unionSet, увеличиваем его кратность и удаляем старую версию элемента из unionSet. Вставляем элемент в unionSet, если ранее его там не было. То же самое делается для элементов в setB (см.рис6).

Вход: два мультимножества.

Выход: мультимножество, являющийся объединением двух мультимножеств.

```cpp
multiset<MultiSetElement> unionSet;

// Вставляем элементы из setA
for (const MultiSetElement& elementA : setA) {
    MultiSetElement elementToInsert = elementA;
    // Если элемент уже есть в unionSet, добавляем его кратности
    auto it = unionSet.find(elementA);
    if (it != unionSet.end()) {
        elementToInsert.multiplicity += it->multiplicity;
        unionSet.erase(it);
    }
    unionSet.insert(elementToInsert);
}

// Вставляем элементы из setB
for (const MultiSetElement& elementB : setB) {
    MultiSetElement elementToInsert = elementB;
    // Если элемент уже есть в unionSet, добавляем его кратности
    auto it = unionSet.find(elementB);
    if (it != unionSet.end()) {
        elementToInsert.multiplicity += it->multiplicity;
        unionSet.erase(it);
    }
    unionSet.insert(elementToInsert);
}
```

Рис. 6: Объединение мультимножеств

2.5 Пересечение

Создается пустое мультимножество intersectionSet, которое будет содержать элементы, входящие в оба мультимножества A и B. Затем проходимся по элементам мультимножества setA. Для каждого элемента проверяется, существует ли такой элемент в мультимножестве setB. Если элемент присутствует в setB, то находится наименьшее значение из двух кратностей множества A и B. Этот элемент добавляется в intersectionSet с кратностью равной наименьшему значению двух кратностей (см.рис7).

Вход: два мультимножества.

Выход: мультимножество, являющийся пересечением двух мультимножеств.

```
// Пересечение мультимножеств A и B
multiset<MultiSetElement> intersectionSet;
for (const MultiSetElement& element : setA) {
    auto foundInB = setB.find(element);
    if (foundInB != setB.end()) {
        // Находим минимум из кратностей элемента в A и B
        int minMultiplicity = min(element.multiplicity, foundInB->multiplicity);
        intersectionSet.insert(MultiSetElement(element.element, minMultiplicity));
    }
}
```

Рис. 7: Пересечение мультимножеств

2.6 Дополнение

Создается мультимножество complementA, которое будет содержать элементы, принадлежащие универсальному мультимножеству setU, но не принадлежащие мультимножеству setA - это результат операции дополнения мультимножеств. Мы проходимся по всем элементам универсального мультимножества setU и, если элемент не встречается в мультимножестве setA, то добавляем его в complementA. Если же такой элемент встречается, то от кратности в универсуме этого элемента отнимается кратность элемента множества A, в complementA сохраняем значение элемента и в кратность записываем результат вычитания кратностей (см.рис8). Таким образом, complementA будет содержать дополнение мультимножества A относительно универсального мультимножества U.

Вход: два мультимножества.

Выход: мультимножество, являющийся дополнением двух мультимножеств.

```
// Дополнение A (A')
int complementMultiplicity;
multiset<MultiSetElement> complementA;
for (const MultiSetElement& element : setU) {
    complementMultiplicity = element.multiplicity;
    //auto foundInA = std::find(grayA.begin(), grayA.end(), element);
    auto foundInA = setA.find(element);
    if (foundInA != setA.end()) {
        complementMultiplicity -= foundInA->multiplicity;
    }
    if (complementMultiplicity > 0) {
        complementA.insert(MultiSetElement(element.element, complementMultiplicity));
    }
}
```

Рис. 8: Дополнение мультимножеств

2.7 Разность

Создается мультимножество differenceAB, которое будет содержать элементы, принадлежащие мультимножеству setA, но не принадлежащие мультимножеству setB - это результат операции разность мультимножеств. Мы проходимся по всем элементам мультимножества setA и, если элемент не встречается в мультимножестве setB, то добавляем его в differenceAB с учетом кратности мультимножества A. Если элемент есть и в A, и в B, вычитаем кратности и добавляем только положительные результаты (см.рис9). Таким образом, differenceAB будет содержать разность мультимножеств A и B.

Вход: два мультимножества.

Выход: мультимножество, являющийся разностью двух мультимножеств.

```
// Разность A \ B
multiset<MultiSetElement> differenceAB;
for (const MultiSetElement& element : setA) {
    auto foundInB = setB.find(element);
    if (foundInB == setB.end()) {
        // Если элемент есть в A, но отсутствует в B, добавляем его с учётом кратности из A
        differenceAB.insert(MultiSetElement(element.element, max(0, element.multiplicity)));
    }
    else {
        // Если элемент есть и в A, и в B, вычитаем кратности и добавляем только положительные результаты
        int differenceMultiplicity = element.multiplicity - foundInB->multiplicity;
        if (differenceMultiplicity > 0) {
            differenceAB.insert(MultiSetElement(element.element, differenceMultiplicity));
        }
    }
}
```

Рис. 9: Разность мультимножеств

2.8 Симметрическая разность

Создается мультимножество symmetricDifference, которое будет содержать элементы, удовлетворяющие условиям симметрической разности мультимножеств A и B. Мы проходимся по всем элементам мультимножества setA и если находим одинакомые элементы берем наибольшую кратность и наименьшую, в symmetricDifference добавляем этот элемент с кратностью равной разности наибольшего и наименьшего значения. В обратном случае добавляем элементы, которые не встречаются в мультимножестве setB. Затем мы проходимся по всем элементам мультимножества setB и добавляем элементы, которые не встречаются в мультимножестве setA. Это позволяет нам создать мультимножество symmetricDifference, содержащее симметрическую разность мультимножеств A и B (см.рис10).

Вход: два мультимножества.

Выход: мультимножество, являющийся симметрической разностью двух мультимножеств.

```
// Симметрическая разность A Δ B
multiset<MultiSetElement> symmetricDifference;
for (const MultiSetElement& element : setA) {
    auto foundInB = setB.find(element);
    if (foundInB != setB.end()) {
        int minMultiplicity = min(element.multiplicity, foundInB->multiplicity);
        int maxMultiplicity = max(element.multiplicity, foundInB->multiplicity);
        int result = maxMultiplicity - minMultiplicity;
        symmetricDifference.insert(MultiSetElement(element.element, result));
    }
    else
    {
        symmetricDifference.insert(MultiSetElement(element));
    }
}

for (const MultiSetElement& element : setB) {
    auto foundInB = setA.find(element);
    if (foundInB != setA.end()) {

    }
    else
    {
        symmetricDifference.insert(MultiSetElement(element));
    }
}
```

Рис. 10: Симметрическая разность мультимножеств

2.9 Арифмеитческая разность

Существует мультимножество symmetricDifferenceSet, которое будет содержать элементы, удовлетворяющие условиям арифметической разности мультимножеств A и B. Первое условие добавляет в symmetricDifferenceSet элементы из мультимножества A, которые не встречаются в мультимножестве B. Второе добавляет элементы из мультимножества A, которые также содержатся и в B. Кратность считается как разность кратности элемента из A с элементом из B, если кратность из B больше, элемент не будет добавлен в symmetricDifferenceSet. Это позволяет создать мультимножество symmetricDifferenceSet, содержащее элементы, присутствующие только

в A, представляя арифметическую разность мультимножеств (см.рис11).

Вход: два мультимножества.

Выход: мультимножество, являющийся арифметической разностью двух мультимножеств.

```
// Арифметическая разность мультимножеств A и B
multiset<MultiSetElement> symmetricDifferenceSet;
for (const MultiSetElement& element : setA) {
    auto foundInB = setB.find(element);
    if (foundInB == setB.end()) {
        // Если элемент есть в A, но отсутствует в B, добавляем его с учётом кратности из A
        symmetricDifferenceSet.insert(MultiSetElement(element.element, element.multiplicity));
    }
    else {
        // Если элемент есть и в A, и в B, вычитаем кратности и добавляем только положительные результаты
        int differenceMultiplicity = element.multiplicity - foundInB->multiplicity;
        if (differenceMultiplicity > 0) {
            symmetricDifferenceSet.insert(MultiSetElement(element.element, differenceMultiplicity));
        }
    }
}
```

Рис. 11: Арифмеитческая разность мультимножеств

2.10 Арифметическое умножение и деление

Арифметическое умножение мультимножеств A и B представляет собой операцию, в результате которой формируется мультимножество, содержащее пересечение всех элементов мультимножеств A и B. Операция, выполняемая в коде, создает новое мультимножество productSet, которое равно мультимножеству intersectionSet (см.рис12).

Вход: два мультимножества.

Выход: мультимножество, являющийся арифметическим умножением двух мультимножеств.

```
// Арифметическое умножение мультимножеств A и B (пересечение всех элементов)
multiset<int> productSet = intersectionSet;
```

Рис. 12: Арифметическое умножение мультимножеств

Арифметическое деление мультимножеств A и B представляет собой операцию, в результате которой формируется мультимножество, содержащее разность A и пересечения с B. Операция, выполняемая в коде, создает новое мультимножество divisionSet, которое равно мультимножеству differenceAB (см.рис13).

Вход: два мультимножества.

Выход: мультимножество, являющийся арифметическим делением двух мультимножеств.

```
// Арифметическое деление мультимножеств A и B (разность A и пересечения с B)
multiset<int> divisionSet = differenceAB;
```

Рис. 13: Арифметическое деление мультимножеств

2.11 Декартово произведение A и B

Создается множество cartesianProductA, которое содержит все возможные пары элементов из мультимножеств A и B. Внутри вложенных циклов происходит итерация по элементам мультимножества A и B, и для каждой пары элементов создается упорядоченная пара (pair), которая добавляется в cartesianProductA (см.рис14).

Вход: два мультимножества.

Выход: множество, содержащее все возможные пары.

```
// Декартово произведение мультимножеств A и B
set<pair<MultiSetElement, MultiSetElement>> cartesianProductA;
for (const MultiSetElement& elementA : setA) {
    for (const MultiSetElement& elementB : setB) {
        cartesianProductA.insert({ elementA, elementB });
    }
}
```

Рис. 14: Декартово произведение A и B мультимножеств

2.12 Автоматическая генерация мультимножеств

Цель данного метода заключается в создании мультимножества $setA$ с заданной мощностью $powerA$. Мультимножество создается с использованием генератора случайных чисел, обеспечивая уникальность элементов в пределах универсума. В данном фрагменте кода создается генератор случайных чисел (gen) с использованием аппаратного источника случайности ($random\_device$). $uniform\_int\_distribution$ определяет диапазон значений для генерации.

Блок кода ниже (см.рис15) осуществляет автоматическую генерацию элементов мультимножества $setA$. Генерируется случайное значение $randomElement$, затем проверяется его уникальность в $setA$. Если элемент уже присутствует, производится повторная генерация до тех пор, пока не будет сгенерирован новый элемент. Далее осуществляется поиск подходящего индекса в box и генерация случайной кратности с последующей проверкой на соответствие, если кратность больше кратности универсума, то кратность будет генерироваться до тех пор, пока не получится значение не большее, чем кратность этого же элемента в универсуме. Далее показано как

сохраняются элементы в мультимножестве.

Вход: мощность мультимножества, разрядность и унимерсум этой разрядности.

Выход: сформированное мультимножество.

```
else if (choice == 'A' || choice == 'a') {
    // Автоматическая генерация мультимножеств
    int randomElement;
        random_device rd;
        mt19937 gen(rd());
        uniform_int_distribution<int> dis(0, (1 << n) - 1);

        for (int i = 0; i < powerA; i++) {

            randomElement = dis(gen);

            // Проверка на наличие элемента с тем же значением в setA
            MultiSetElement tempElement(randomElement, 0);
            while (setA.find(tempElement) != setA.end()) {
                randomElement = dis(gen);
                tempElement = MultiSetElement(randomElement, 0);
            }

            int j = 0;
            while (j < (1 << n) && box[j][0] != randomElement) {
                j++;
            }

            randomMultiplicity = generateRandomMultiplicity();
            while (box[j][1] <= randomMultiplicity) {
                randomMultiplicity = generateRandomMultiplicity();
            }
```

```
        MultiSetElement elem;
        elem.element = randomElement;
        elem.multiplicity = randomMultiplicity;
        setA.insert(elem);
        //cout << toGrayCodeString(elem.element, n) << "|" << elem.multiplicity << endl;
    }
```

Рис. 15: Автоматическая генерация мультимножества A

2.13 Ввод мультимножеств вручную

В первую очередь осуществляется проверка на вводимые значения (см.рис16), проверка абсолютно идентична проверки на ввод мощности мультимножеств. Затем происходит конвертация строки в значение типа *int*. Если значение меньше нуля или больше допустимого выводится предупреждение и повторяется итерация. При помощи функции *toGrayCodeString* происходит сравнение строк, когда анологичная строка найдена в универсуме мы получаем индекс под которым хранится это значение в универсуме. Далее происходит уже ранее использованная проверка на нахождение элемента с тем же значением в самом мультимножестве.

В конце осуществляется пароверка на ввод кратности, в которой предусмотрена проверка на удовлетворяющее универсуму значение кратности. Если кратность введена неверна или больше

кратности универсума, то пользователь увидит предупреждение, функция $validInput = false$ и итерация повторится вновь.

Вход: разрядность, мощность мультимножества, универсум, элементы мультимножества и кратность каждого элемента.

Выход: сформированное мультимножество.

```
    // Преобразование введенной строки в число
    bitset<10> binaryElement(el_A);
    element = binaryElement.to_ulong();
    //int elementG = toGrayCode(element);
    if (!(element >= 0 && element < (1 << n))) {
        cout << "Некорректный ввод. Введите бинарное значение разрядности n." << endl;
        i--; // повторяем итерацию
        continue;
    }
    int j = 0;
    while (j < (1 << n) && toGrayCodeString(box[j][0], n) != el_A) {
        j++;
    }
    element = j;
     //Проверка на наличие элемента с тем же значением в setA
    MultiSetElement tempElement(element, 0);
    if (setA.find(tempElement) != setA.end()) {
        cout << "Некорректный ввод. Данный элемент уже содержится в мультимножестве." << endl;
        i--; // повторяем итерацию
        continue;
    }
```

```
while (true) {
    bool validInput = true;

    for (char* c = m_A; *c != '\0'; ++c) {
        if (!isdigit(*c)) {
            validInput = false;
            break;
        }
    }

    if (validInput) {
        multiplicity = atoi(m_A);
        int j = 0;
        while (j < (1 << n) && box[j][0] != element ) {
            j++;
        }
        if (box[j][1] >= multiplicity) {
            break;
        }
    }

    cerr << "Некорректный ввод. Кратность элемента не должна превышать кратность универсума" << endl;
    cout << "Введите кратность элемента: ";
    cin >> m_A;
}
```

Рис. 16: Ручной ввод мультимножества A

2.14 Защита от некорректного ввода

Защита на корректный ввод осуществляется на все вводимые значения, в частности на разрядность, мощность и кратность множеств. Рассмотрим принцип работы защиты на примере ввода мощности множества B: $int\ powerB$ - переменная, в которой будет храниться мощность множества B. $char\ inputP\_B[10]$ - массив символов для временного хранения введенной строки с мощностью B.

Вывод приглашения для пользователя: Выводится приглашение пользователю с текстом «Введите мощность множества B (количество элементов):» .

Чтение ввода пользователя: Считывается ввод пользователя с клавиатуры и сохраняется в массив символов $char\ inputP\_B$.

Проверка валидности ввода: Запускается бесконечный цикл с помощью $while$(см.рис17) ($true$), который будет выполняться до тех пор, пока не будет получен корректный ввод. Внутри цикла проверяется каждый символ в строке $char\ inputP\_B$. Если какой-либо символ не является цифрой (с помощью $isdigit()$), то ввод считается некорректным, и устанавливается флаг $validInput$ в $false$. Если весь ввод содержит только цифры, и флаг $validInput$ остается $true$, происходит следующее: Строка $char\ inputP\_B$ преобразуется в целое число с помощью $atoi()$, и результат сохраняется в $int\ powerB$. Проверяется, что $int\ powerB$ больше или равно нулю. Если оба условия выполняются, цикл завершается с помощью break.

Некорректный ввод: Если ввод оказывается некорректным, выводится сообщение об ошибке с помощью $cerr$, предупреждая пользователя о том, что мощность должна быть неотрицательным целым числом. Повторный запрос: Запрашивается новый ввод мощности множества B у пользователя с помощью сообщения «Введите мощность множества B: ». Цикл повторяется до тех пор, пока не будет получен корректный ввод.

```
int powerB;
char inputP_B[10];
cout << "Введите мощность множества B (количество элементов): ";
cin >> inputP_B;

while (true) {
    bool validInput = true;

    for (char* c = inputP_B; *c != '\0'; ++c) {
        if (!isdigit(*c)) {
            validInput = false;
            break;
        }
    }

    if (validInput) {
        powerB = atoi(inputP_B);
        if (powerB >= 0 && powerB <= (1 << n)) {
            break;
        }
    }

    cerr << "Некорректный ввод. Мощность должна быть неотрицательным целым числом, не больше чем 2^n(n - разрядность)." << endl;
    cout << "Введите мощность множества B: ";
    cin >> inputP_B;
}
```

Рис. 17: Ввод мощности множества B

Также реализована защита от дурака для для выбора пользователя между двумя способами заполнения множества: вручную (M) или автоматически (A) (см.рис18). Алгоритмическое описание:

Объявление переменных: $char\ choice$ - переменная для хранения выбора пользователя (M или A).

Вывод приглашения для пользователя: Выводится приглашение пользователю с текстом «Выберите способ заполнения множества (M - вручную, A - автоматически): ».

Чтение выбора пользователя: Считывается ввод пользователя с клавиатуры и сохраняется в переменной *choice*.

Проверка валидности выбора: Запускается цикл while(см.рис18), который будет выполняться до тех пор, пока пользователь не введет корректный выбор. Внутри цикла проверяется, что введенный символ приведен к нижнему регистру (*tolower*(*choice*)) не равен ни 'm', ни 'a'. Если введенный символ не равен 'm' и не равен 'a', то считается, что ввод некорректен. Если ввод некорректен, выводится сообщение об ошибке с помощью *cerr*, предупреждая пользователя о некорректном выборе. Затем пользователю предлагается ввести выбор еще раз с помощью сообщения «Выберите способ заполнения множества (M - вручную, A - автоматически): ». Цикл повторяется до тех пор, пока не будет получен корректный выбор.

Корректный выбор: Когда пользователь вводит 'm' или 'a', цикл завершается, и выбор пользователя сохраняется в переменной *choice* для дальнейшего использования в программе.

```
char choice;
cout << "Выберите способ заполнения множества (M – вручную, A – автоматически): ";
cin >> choice;

while (tolower(choice) != 'm' && tolower(choice) != 'a') {
    cerr << "Некорректный ввод. Введите 'M' для заполнения вручную или 'A' для автоматического заполнения." << endl;
    cout << "Выберите способ заполнения множества (M – вручную, A – автоматически): ";
    cin >> choice;
}
```

Рис. 18: Проверка на правильный ввод способа заполнения множества

3 Результаты работы

В результате работы было реализовано 11 различных операций над мультимножествами, результаты которых выводятся на экран в виде бинарного кода Грея (см.рис19 и рис20). У пользователя есть возможность ввести разрядность множеств, выбрать мощности для каждого из мультимножеств, а также выбрать их кратность. Множества можно заполнить вручную или же случайно сгенерировать все элементы множества. Для надежности и корректной работы программы была реализована защита на ввод любых значений.

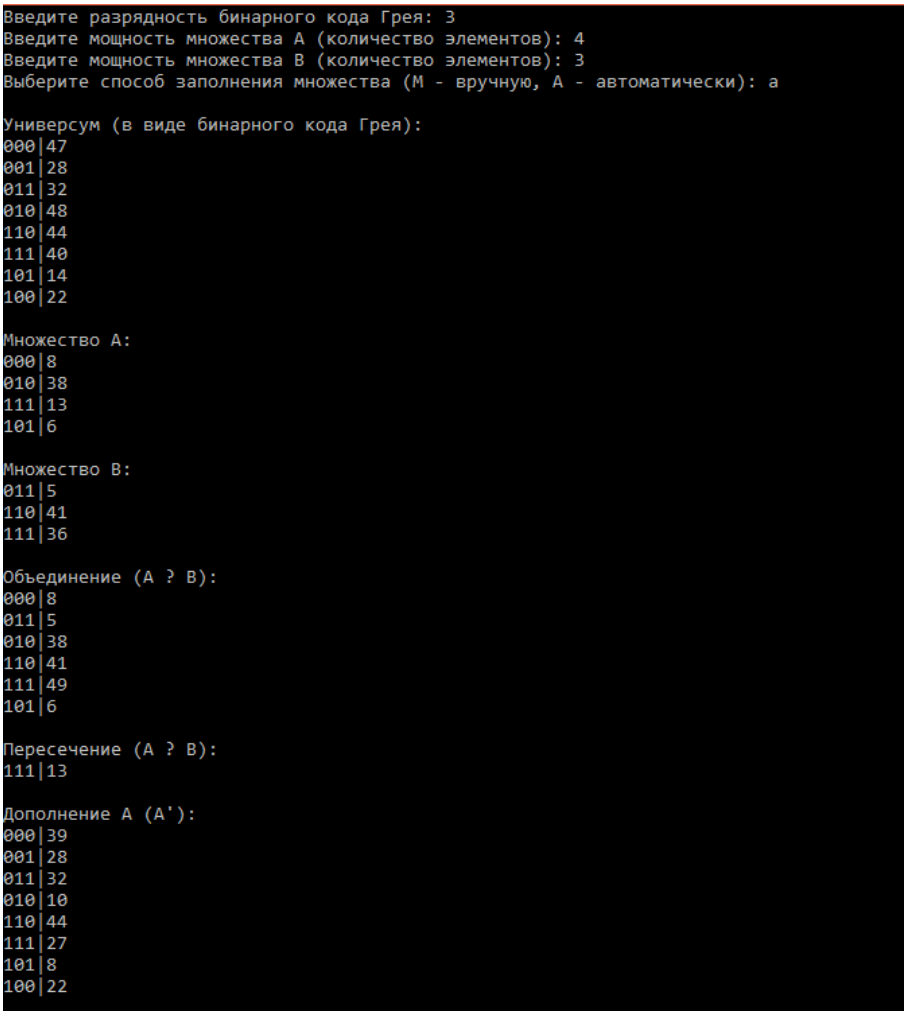

Рис. 19: Вывод на консоль результатов

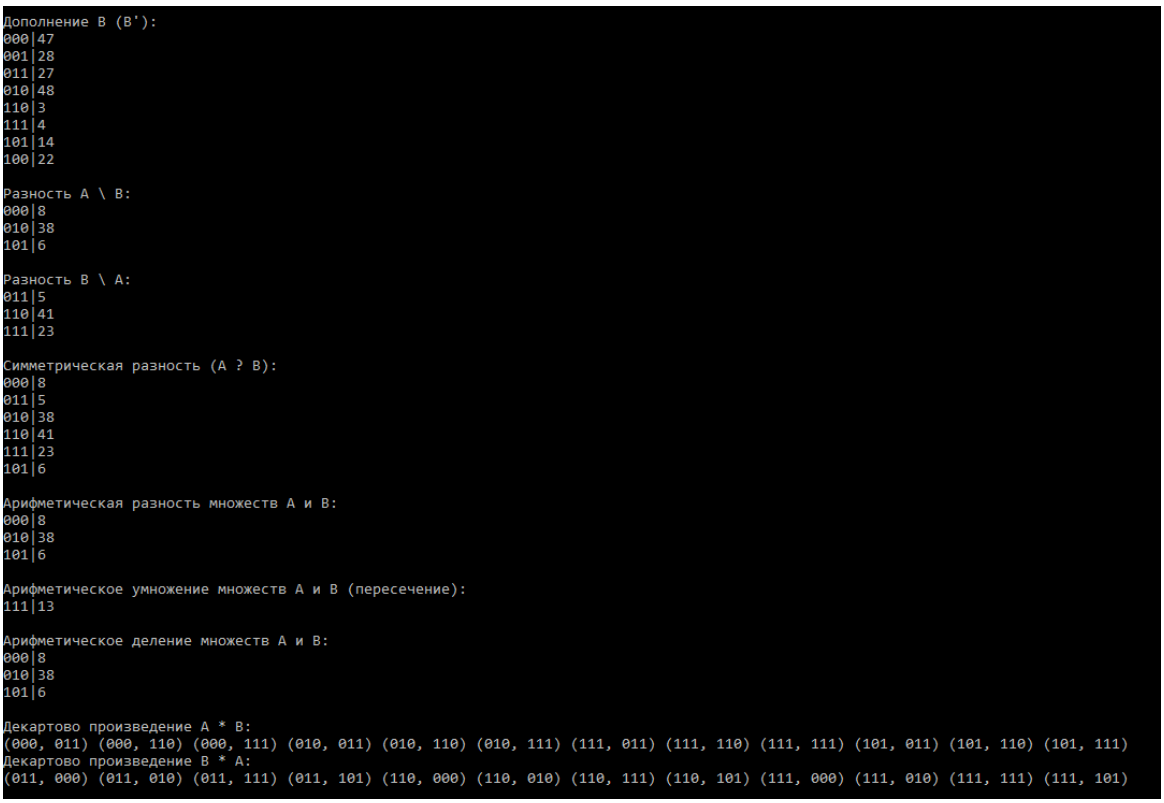

Рис. 20: Консоль

Заключение

В рамках выполнения лабораторной работы №1 была разработана программа, выполняющая ряд операций над мультимножествами, используя бинарный код Грея. Эта программа включает в себя генерацию бинарного кода Грея, создание универсума и двух мультимножеств, а также реализацию операций объединение, пересечение, дополнение, разность, симметрическую разность, арифметическую разность, арифметическое умножение и деление, а также декартово произведение двух мультимножеств. Выполнение работы позволило освоить операции над мультимножествами, применение бинарного кода Грея и овладеть навыками разработки программ для работы с мультимножествами.

Если говорить о плюсах и минусах программы, то хочется отметить следующее: минусом является структурированность, а именно: вся программа расписана в одном CPP файле, что значительно усложняет восприятие кода. Также, повтор кода касательно защиты от неккоректного ввода. И главное, использование multiset вместо vector. Плюсом является хорошая проверка вводимых данных. Еще одним плюсом, является реализация нетривиальной операции декартово произведение.

Программу можно масштабировать, к примеру дополнить операциями возведения в степень, НОК, НОД. Реализация простых операций над мультимножествами может быть основой сложного калькулятора, который будет работать с мультимножествами.

Список материалов

[1] Новиков Ф.А. Дискретная математика для программистов (начиная с 3-его издания).[последняя активация 12.02.2024]
https://stugum.files.wordpress.com/2014/03/novikov.pdf

[2] Ю.А. Фарков «Материалы по математике для Liberal Arts. Часть 1» [последняя активация 12.02.2024]
https://ion.ranepa.ru/upload/medialibrary/3ca/Matematika-dlya-LA.-CHast-1.pdf