

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

Дискретная математика
ОТЧЁТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №1

Реализация программы генерации бинарного кода Грея и операций над
мультимножествами на его основании.

Студент,
группы 5130201/30002

_____ Михайлова А. А.

Преподаватель

_____ Востров А. В.

«_____» _____ 2024 г.

Санкт-Петербург, 2024

Содержание

Введение	4
1 Математическое описание	5
1.1 Множества	5
1.2 Мультимножества	5
1.3 Операции над мультимножествами	5
1.3.1 Объединение	5
1.3.2 Пересечение	5
1.3.3 Дополнение	6
1.3.4 Разность	6
1.3.5 Симметрическая разность	6
1.3.6 Арифметическая сумма	6
1.3.7 Арифметическая разность	6
1.3.8 Арифметическое произведение	7
1.3.9 Арифметическое деление	7
1.4 Бинарный код Грея	7
2 Реализация программы	8
2.1 Функция menu	8
2.2 Функция start	9
2.3 Функция multiset	11
2.4 Функция result	12
2.5 Функция arifmdel	14
2.6 Функция arifmpr	15
2.7 Функция arifmrzn	15
2.8 Функция arifmsumm	16
2.9 Функция symrasnost	17
2.10 Функция rasnost	17
2.11 Функция dopolnenye	17
2.12 Функция peresechenye	18
2.13 Функция obyedinenie	18
2.14 Функция autofeel	19
2.15 Функция handfeel	21
2.16 Функция fillSetAAutomatically	21
2.17 Функция fillSet	22
2.18 Функция isnum	23
2.19 Функция printnull	24
2.20 Функция randomgenerate	24
2.21 Функции sup и gray	25
2.22 Функция sumVector	26
2.23 Функция printm	26

2.24 Функция print	26
3 Результаты работы программы	28
Заключение	33
Список использованной литературы	35

Введение

Лабораторная работа заключается в реализации программы генерации бинарного кода Грея для заполнения универсума мультимножеств (заданной пользователем разрядности). На основе универсума формируются два мультимножества двумя способами заполнения - вручную и автоматически (выбирает пользователь). Мощности множеств задает пользователь. В результате на экран выводятся результаты действий над множествами – объединения, пересечения, разности, дополнения, симметрической разности, арифметической суммы, разности, произведения и деления. Кроме того, реализована защита от некорректного пользовательского ввода.

Лабораторная работа выполнена на языке C++ в среде разработки XCode.

1 Математическое описание

1.1 Множества

Множество – любая определённая совокупность объектов.

Элементы множества – объекты, из которых составлено множество. Они различны и отличимы друг от друга.

Как множествам, так и элементам можно давать имена или присваивать символьные обозначения. Обычно множества обозначают прописными буквами латинского алфавита, а элементы множеств — строчными буквами.

Множество, не содержащее элементов, называется пустым. Обозначение: \emptyset .

Обычно в конкретных рассуждениях элементы всех рассматриваемых множеств берутся из некоторого одного, достаточно широкого множества U (своего для каждого случая), которое называется универсальным множеством (или универсумом).

1.2 Мультимножества

Мультимножеством $\hat{X} = \langle a_1(x_1), \dots, a_n(x_n) \rangle$ называется совокупность элементов множества X , в которую элемент x_i входит a_i раз, $a_i \geq 0$. Число a_i называется показателем (кратностью) элемента x_i .

1.3 Операции над мультимножествами

1.3.1 Объединение

Объединением мультимножеств A и B называется мультимножество, состоящее из всех элементов, которые присутствуют хотя бы в одном из мультимножеств, и кратность каждого элемента равна максимальной кратности соответствующих элементов в объединяемых мультимножествах.

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

1.3.2 Пересечение

Пересечением мультимножеств A и B называется мультимножество, состоящее из всех элементов, которые одновременно присутствуют в каждом из мультимножеств, и кратность каждого элемента равна минимальной кратности соответствующих элементов в пересекаемых мультимножествах.

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

1.3.3 Дополнение

Дополнением мультимножества A до универсума U называется мультимножество, состоящее из тех элементов, кратность которых равна разности кратностей соответствующих элементов в универсуме U и дополняемом мультимножестве A .

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

1.3.4 Разность

Разность мультимножеств A и B представляет собой новое мультимножество, состоящее из всех элементов, которые присутствуют в мультимножестве A , но отсутствуют в мультимножестве B . При этом учитывается кратность элементов.

$$C = A \setminus B = A \cap \bar{B} = \{ \min(a_i(x_i), a_j(x_j)) \}, a_i(x_i) \in A, a_j(x_j) \in U \setminus B$$

1.3.5 Симметрическая разность

Симметрической разностью двух множеств A и B является множество, которое включает в себя элемент только если он принадлежит ровно одному из данных множеств A и B . Кратность каждого элемента результирующего множества равна модулю разности кратностей соответствующих элементов в вычитаемых мультимножествах.

$$C = A \Delta B = \{ \min(\max(a_i(x_i), a_j(x_j)), u_k(x_k) - \min(a_i(x_i), a_j(x_j))) \}, a_i(x_i) \in A, a_j(x_j) \in B, u_k(x_k) \in U$$

1.3.6 Арифметическая сумма

Арифметической суммой мультимножеств A и B называется мультимножество, состоящее из всех элементов, которые присутствуют хотя бы в одном из мультимножеств, и кратность каждого элемента равна сумме кратностей соответствующих элементов в складываемых мультимножествах.

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

1.3.7 Арифметическая разность

Арифметической разностью мультимножеств A и B называется мультимножество, состоящее из тех элементов мультимножества A , кратность которых превышает кратность соответствующих элементов в мультимножестве B . Кратность каждого элемента результирующего множества равна разности кратностей соответствующих элементов в вычитаемых мультимножествах.

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

1.3.8 Арифметическое произведение

Арифметическим произведением мультимножеств A и B называется мультимножество, состоящее из элементов, которые одновременно присутствуют в каждом из мультимножеств, и их кратность равна произведению кратностей соответствующих элементов в перемножаемых мультимножествах.

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

1.3.9 Арифметическое деление

Арифметическое деление между первым и вторым мультимножеством

$$C(x) = \begin{cases} \frac{n_A(x)}{n_B(x)}, & \text{если } n_B(x) \neq 0 \\ 0, & \text{иначе} \end{cases}$$

где $n_A(x)$ - количество вхождений элемента x в множество A , а $n_B(x)$ - количество вхождений элемента x в множество B .

1.4 Бинарный код Грея

Бинарный код Грея - двоичный код, где следующее число в коде получается из предыдущего сменой одного символа. Например для чисел длиной 2 бита последовательность будет: 00 01 11 10.

Ознакомиться с алгоритмом можно на рисунке 1.

Рис. 1: Построение бинарного кода Грея

2 Реализация программы

Вся программа реализована в файле `main.cpp` с использованием библиотек `vector`, `string`, `algorithm` и `random`.

1. `vector < vector < int > > uni` – хранит элементы юниверсума
2. `vector < int > kratnost` – хранит кратности юниверсума
3. `vector < int > setA` – хранит кратности мультимножества A
4. `vector < int > setB` – хранит кратности мультимножества B

2.1 Функция `menu`

Вход: ссылка на юниверсум, кратности юниверсума, ссылка на кратности мультимножества A , ссылка на кратности мультимножества B

Выход: ответ на пользовательский ввод

Функция `menu` выводит на экран две опции: ввод разрядности юниверсума (нажатие `'r'`) и выход из программы (нажатие `'h'`). Бесконечный цикл запрашивает ввод символа и обрабатывает его: при вводе `'r'` вызывается функция `start` с переданными векторами для дальнейшей обработки, при вводе `'h'` выводится сообщение о выходе, и программа завершается с помощью `exit(0)`. В случае ввода других символов выводится сообщение об ошибке, и пользователь повторно запрашивается на ввод.

Ознакомиться с кодом можно в Листинг 1.

Листинг 1: Функция `menu`

```
void menu(vector<vector<int>> & uni, vector<int> kratnost, vector<
    int> &setA, vector<int> &setB){
cout << "Меню\n";
cout << "Если вы хотите ввести разрядность юниверсума, нажмите 'r'
    '\n";
cout << "Если вы хотите выйти из программы, нажмите 'h'\n";
char symbol;
while (true) {
    cin >> symbol;
    switch (symbol) {
        case 'r':
            start(uni, kratnost, setA, setB);
            break;
        case 'h':
            cout << "Выход из программы...\n";
            exit(0);
            break;
```



```

        default:
            cout << "Введён символ не из списка. Попробуйте
снова.\n";
            break;
    }
}
}

```

2.2 Функция start

Вход: ссылка на юниверсум, кратности юниверсума, ссылка на кратности мультимножества A, ссылка на кратности мультимножества B

Выход: заполненные юниверсум и вектор кратностей юниверсума

Функция start отвечает за ввод разрядности юниверсума и последующую обработку введенных данных. В начале функция запрашивает ввод значения разрядности и проверяет его на корректность с помощью функции isnum, обеспечивая, что ввод будет целым положительным числом. В случае некорректного ввода пользователь получает сообщение об ошибке и повторно запрашивается на ввод. Если введенное значение разрядности равно нулю, автоматически выводится сообщение о возможности выхода из программы или перезапуска, после чего пользователь может вводить символы 'q' для выхода или 'a' для перезапуска, что приводит к очищению векторов и возврату в меню. Если же разрядность положительна, функция генерирует юниверсум с помощью функции gray, заполняет кратности юниверсума с использованием функции randomgenerate, а также корректно изменяет размеры векторов setA и setB. Далее рассчитывается сумма элементов вектора kratnost, после чего осуществляется вывод всех данных с помощью функции print, и, наконец, выполняется функция multiset для дополнительной обработки.

Ознакомиться с кодом можно в Листинг 2.

Листинг 2: Функция start

```

void start(vector<vector<int>> & uni, vector<int> kratnost, vector
<int> &setA, vector<int> &setB){
    string str = "";
    int n = 0;
    cout << "Введите разрядность юниверсума:\n";
    cin >> str;
    for(;;){
        if(isnum(str)){
            n = atoi(str.c_str ());
            break;
        }
        else{

```

```

        cout << "Разрядность_юниверсума_может_быть_только_
целым_положительным_числом,_повторите_ввод.\n";
        cin >> str;
    }
}
if (n == 0){
    printnull();
    cout << "Если_вы_хотите_выйти_из_программы,_введите_'q'\n"
;
    cout << "Для_перезапуска_программы,_введите_'a'\n";
    char a;
    while (true) {
        cin >> a;
        switch (a) {
            case 'q':
                cout << "Выход_из_программы...\n";
                exit(0);
                break;
            case 'a':
                uni.clear();
                setA.clear();
                setB.clear();
                kratnost.clear();
                menu(uni, kratnost, setA, setB);
                break;
            default:
                cout << "Введён_символ_не_из_списка,_
повторите_попытку.\n";
                break;
        }
    }
}
else{
    uni = gray(n);
    kratnost = randomgenerate(n);
    setA.resize(1 << n);
    setB.resize(1 << n);
    int sumkrat = sumVector(kratnost);
    print(uni, kratnost, sumkrat);
    multiset(uni, kratnost, setA, setB, n);
}
}

```

2.3 Функция multiset

Вход: ссылка на универсум, кратности универсума, ссылка на кратности мультимножества A, ссылка на кратности мультимножества B

Выход: заполненные мультимножества A и B

Функция использует бесконечный цикл для обработки ввода пользователя. Если пользователь вводит 'h', вызывается функция handfeel, которая позволяет пользователю вручную заполнить мультимножества, после чего устанавливается флаг isA в значение false, чтобы перейти к мультимножеству B. При вводе 'a' вызывается функция autofeel, ответственной за автоматическое заполнение мультимножеств, также с установкой флага isA в false. Если пользователь вводит 'o', происходит вызов функции result, которая выводит результаты работы с мультимножествами. Если вводится любой другой символ, функция сообщает об ошибке и предлагает повторить попытку.

Ознакомиться с кодом можно в Листинг 3.

Листинг 3: Функция multiset

```
void multiset(vector<vector<int>> & uni, vector<int> kratnost,
    vector<int> &setA, vector<int> &setB, int n){
    cout << "Сформируем 2 мультимножества.\n";
    cout << "Для заполнения мультимножеств вручную введите 'h'.\n";
    cout << "Для заполнения мультимножеств автоматически введите 'a'\n";
    char symbol;
    bool isA = true;
    while (true) {
        cin >> symbol;
        switch (symbol) {
            case 'h':
                handfeel(uni, kratnost, setA, setB, isA, n);
                isA = false;
                break;
            case 'a':
                autofeel(uni, kratnost, setA, setB, isA, n);
                isA = false;
                break;
            case 'o':
                result(uni, kratnost, setA, setB, n);
                break;
            default:
                cout << "Введён символ не из списка, повторите\n";
                попытку.\n";
                break;
        }
    }
```

```

}
}

```

2.4 Функция result

Вход: ссылка на юниверсум, кратности юниверсума, ссылка на кратности мультимножества A, ссылка на кратности мультимножества B,

Выход: вывод результатов операций над мультимножествами

Функция result обрабатывает два мультимножества и выводит результаты различных операций над ними. В завершение пользователь получает инструкции по выходу или перезапуску программы, а ввод обрабатывается в цикле, позволяя очищать множества и повторно запускать операции.

Ознакомиться с кодом можно в Листинг 4.

Листинг 4: Функция result

```

void result(vector<vector<int>> & uni, vector<int> kratnost,
    vector<int> &setA, vector<int> &setB, int n){
vector<int> res(1 << n, 0);
int sum = 0;
cout << "Мультимножество A\n";
sum = sumVector(setA);
print(uni, setA, sum);
cout << "Мультимножество B\n";
sum = sumVector(setB);
print(uni, setB, sum);
res = obyedinenie(setA, setB, n);
sum = sumVector(res);
cout << "Объединение A и B\n";
printm(uni, res, sum);
res = peresechenye(setA, setB, n);
sum = sumVector(res);
cout << "Пересечение A и B\n";
printm(uni, res, sum);
res = dopolnenye(kratnost, setA, n);
sum = sumVector(res);
cout << "Дополнение A\n";
printm(uni, res, sum);
res = dopolnenye(kratnost, setB, n);
sum = sumVector(res);
cout << "Дополнение B\n";
printm(uni, res, sum);
vector<int> dopA = dopolnenye(kratnost, setA, n);
vector<int> dopB = dopolnenye(kratnost, setB, n);

```

```

res = rasnost(setA, dopB, n);
sum = sumVector(res);
cout << "Разность A и B\n";
printm(uni, res, sum);
res = rasnost(setB, dopA, n);
sum = sumVector(res);
cout << "Разность B и A\n";
printm(uni, res, sum);
vector<int> obAB = obyedinenie(setA, setB, n);
vector<int> perAB = peresechenye(setA, setB, n);
vector<int> dopperAB = dopolnenye(kratnost, perAB, n);
res = symrasnost(obAB, dopperAB, n);
sum = sumVector(res);
cout << "Симметрическая разность A и B\n";
printm(uni, res, sum);
res = arifmsumm(setA, setB, kratnost, n);
sum = sumVector(res);
cout << "Арифметическая сумма A и B\n";
printm(uni, res, sum);
res = arifmravn(setA, setB, n);
sum = sumVector(res);
cout << "Арифметическая разность A и B\n";
printm(uni, res, sum);
res = arifmravn(setB, setA, n);
sum = sumVector(res);
cout << "Арифметическая разность B и A\n";
printm(uni, res, sum);
res = arifmpr(setA, setB, kratnost, n);
sum = sumVector(res);
cout << "Арифметическое произведение A и B\n";
printm(uni, res, sum);
res = arifmdel(setA, setB, n);
sum = sumVector(res);
cout << "Арифметическое деление A и B\n";
printm(uni, res, sum);
res = arifmdel(setB, setA, n);
sum = sumVector(res);
cout << "Арифметическое деление B и A\n";
printm(uni, res, sum);
cout << "Если вы хотите выйти из программы, введите 'q'\n";
cout << "Для перезапуска программы, введите 'a'\n";
char a;
while (true) {

```

```

cin >> a;
switch (a) {
    case 'q':
        cout << "Выход из программы...\n";
        exit(0);
        break;
    case 'a':
        uni.clear();
        setA.clear();
        setB.clear();
        kratnost.clear();
        menu(uni, kratnost, setA, setB);
        break;
    default:
        cout << "Введён символ не из списка, повторите
попытку.\n";
        break;
}
}
}

```

2.5 Функция arifmdel

Вход: ссылка на кратности мультимножества A, ссылка на кратности мультимножества B

Выход: мультимножество result - результат операции

Функция arifmdel реализует арифметическое деление мультимножеств. Возвращает новый вектор, который является результатом операции. В вектор добавляется либо результат деления кратностей, либо 0.

Ознакомиться с кодом можно в Листинг 5.

Листинг 5: Функция arifmdel

```

vector<int> arifmdel(vector<int> &setA, vector<int> &setB, int n){
vector<int> result(1 << n, 0);
for(int i = 0; i < setA.size(); i++){
    int n = setA[i] / setB[i];
    if(n < 0){
        result[i] = 0;
    }
    else{
        result[i] = n;
    }
}
}

```

```
return result;
}
```

2.6 Функция arifmpr

Вход: ссылка на кратности мультимножества A, ссылка на кратности мультимножества B

Выход: мультимножество result - результат операции

Функция arifmpr реализует арифметическое произведение мультимножеств. Возвращает новый вектор, который является результатом операции. В вектор добавляется либо результат произведения кратностей, либо максимальная кратность элемента в универсуме.

Ознакомиться с кодом можно в Листинг 6.

Листинг 6: Функция arifmpr

```
vector<int> arifmpr(vector<int> &setA, vector<int> &setB, vector<
    int> kratnost, int n){
vector<int> result(1 << n, 0);
for(int i = 0; i < setA.size(); i++){
    int n = setA[i] * setB[i];
    if(n > kratnost[i]) {
        result[i] = kratnost[i];
    }
    else{
        result[i] = n;
    }
}
return result;
}
```

2.7 Функция arifmrazn

Вход: ссылка на кратности мультимножества A, ссылка на кратности мультимножества B

Выход: мультимножество result - результат операции

Функция arifmrazn реализует арифметическую разность мультимножеств. Возвращает новый вектор, который является результатом операции. В вектор добавляется либо результат разности кратностей, либо 0.

Ознакомиться с кодом можно в Листинг 7.

Листинг 7: Функция arifmrazn

```
vector<int> arifmrazn(vector<int> &setA, vector<int> &setB, int n)
{
```

```

vector<int> result(1 << n, 0);
for(int i = 0; i < setA.size(); i++){
    int n = setA[i] - setB[i];
    if(n < 0){
        result[i] = 0;
    }
    else{
        result[i] = n;
    }
}
return result;
}

```

2.8 Функция arifmsumm

Вход: ссылка на кратности мультимножества А, ссылка на кратности мультимножества В

Выход: мультимножество result - результат операции

Функция arifmsumm реализует арифметическую сумму мультимножеств. Возвращает новый вектор, который является результатом операции. В вектор добавляется либо результат суммы кратностей, либо максимальная кратность элемента юниверсума.

Ознакомиться с кодом можно в Листинг 8.

Листинг 8: Функция arifmsumm

```

vector<int> arifmsumm(vector<int> &setA, vector<int> &setB, vector
    <int> kratnost, int n){
vector<int> result(1 << n, 0);
for(int i = 0; i < setA.size(); i++){
    int n = setA[i] + setB[i];
    if(n > kratnost[i]) {
        result[i] = kratnost[i];
    }
    else{
        result[i] = n;
    }
}
return result;
}

```


2.9 Функция symrasnost

Вход: ссылка на кратности мультимножества объединения мультимножеств А и В, ссылка на кратности мультимножества дополнения пересечения мультимножеств А и В

Выход: мультимножество result - результат операции

Функция symrasnost реализует симметрическую разность мультимножеств. Возвращает новый вектор, который является результатом операции. Операция реализована с помощью функций obyedinenie, peresechenye, dopolnenye.

Ознакомиться с кодом можно в Листинг 9.

Листинг 9: Функция symrasnost

```
vector<int> symrasnost(vector<int> &obAB, vector<int> &dopperAB,
    int n){
vector<int> result (1<<n, 0);
result = peresechenye(obAB, dopperAB, n);
return result;
}
```

2.10 Функция rasnost

Вход: ссылка на кратности мультимножества 1, ссылка на кратности мультимножества дополнения мультимножества 2

Выход: мультимножество result - результат операции

Функция rasnost реализует разность мультимножеств. Возвращает новый вектор, который является результатом операции. Операция реализована с помощью функции dopolnenye.

Ознакомиться с кодом можно в Листинг 10.

Листинг 10: Функция rasnost

```
vector<int> rasnost(vector<int> &setA, vector<int> &dopB, int n){
vector<int> result (1<<n, 0);
result = peresechenye(setA, dopB, n);
return result;
}
```

2.11 Функция dopolnenye

Вход: ссылка на кратности универсума, ссылка на кратности мультимножества

Выход: мультимножество result - результат операции

Функция dopolnenye реализует дополнение мультимножества. Возвращает новый вектор, который является результатом операции. В вектор добав-

ляется максимум из результата разности кратностей юниверсума и мульти-множества и 0.

Ознакомиться с кодом можно в Листинг 11.

Листинг 11: Функция `dopolnenye`

```
vector<int> dopolnenye(vector<int> &kratnost, vector<int> &set,
    int n){
vector<int> result (1<<n, 0);
for(int i = 0; i < set.size(); i++){
    result[i] = max(kratnost[i] - set[i], 0);
}
return result;
}
```

2.12 Функция `peresechenye`

Вход: ссылка на кратности мультимножеств

Выход: мультимножество `result` - результат операции

Функция `peresechenye` реализует пересечение мультимножества. Возвращает новый вектор, который является результатом операции. В вектор добавляется минимум из кратностей мультимножеств.

Ознакомиться с кодом можно в Листинг 12.

Листинг 12: Функция `peresechenye`

```
vector <int> peresechenye(vector<int> &setA, vector<int> &setB,
    int n){
vector<int> result (1<<n, 0);
for(int i = 0; i < setA.size(); i++){
    result[i] = min(setA[i], setB[i]);
}
return result;
}
```

2.13 Функция `obyedinenie`

Вход: ссылка на кратности мультимножеств

Выход: мультимножество `result` - результат операции

Функция `obyedinenie` реализует объединение мультимножества. Возвращает новый вектор, который является результатом операции. В вектор добавляется максимум из кратностей мультимножеств.

Ознакомиться с кодом можно в Листинг 13.

Листинг 13: Функция obyedinenie

```
vector<int> obyedinenie(vector<int> &setA, vector<int> &setB, int
    n){
vector<int> result (1<<n, 0);
for(int i = 0; i < setA.size(); i++){
    result[i] = max(setA[i], setB[i]);
}
return result;
}
```

2.14 Функция autofeel

Вход: ссылка на юниверсум, кратности юниверсума, ссылка на кратности мультимножества А, ссылка на кратности мультимножества В, bool значение

Выход: автоматически заполненные мультимножества

Функция autofeel предназначена для автоматического заполнения мультимножества с пользовательским вводом. Она принимает на вход флаг isA, указывающий на то, обрабатывается ли множество А или В. В зависимости от значения 'isA', она запрашивает ввод количества ненулевых элементов для соответствующего множества. Пользователь вводит количество, и функция проверяет корректность значения: оно должно быть целым числом в диапазоне от 0 до максимального значения, определяемого размером вектора кратностей. Если ввод некорректен, выводится ошибка, и пользователь может повторить попытку. После успешного ввода функция автоматически заполняет указанное множество с помощью функции fillSetAAutomatically и сообщает об успешном заполнении. Если заполняется множество А, пользователю предлагается выбрать способ заполнения множества В (вручную или автоматически). Если же заполняется множество В, появляется сообщение о завершении заполнения и предложении увидеть результаты операций над мультимножествами.

Ознакомиться с кодом можно в Листинг 14.

Листинг 14: Функция autofeel

```
void autofeel(vector<vector<int>>& uni, vector<int> kratnost,
    vector<int>& setA, vector<int>& setB, bool isA, int n) {
cout << "Выбрано_заполнение_множеств_автоматически.\n";
if (isA) {
int nonZeroCount;
int totalElements = kratnost.size();
cout << "Введите_количество_ненулевых_элементов_для_множества_A_
    максимум( " << totalElements << "):_";
while (true) {
    cin >> nonZeroCount;
```

```

        if (cin.fail() || nonZeroCount < 0 || nonZeroCount >
totalElements) {
            cout << "Ошибка! Введите целое число от 0 до " <<
totalElements << ". Повторите ввод: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n'
);
        } else {
            break;
        }
    }
    fillSetAAutomatically(kratnost, setA, nonZeroCount);
    cout << "Множество A заполнено автоматически.\n";
    cout << "Выберите способ заполнения множества B ('h' - ручную, 'a'
' - автоматически): \n";
} else {
    int nonZeroCount;
    int totalElements = kratnost.size();
    cout << "Введите количество ненулевых элементов для множества B
максимум(" << totalElements << "): ";
    while (true) {
        cin >> nonZeroCount;

        if (cin.fail() || nonZeroCount < 0 || nonZeroCount >
totalElements) {
            cout << "Ошибка! Введите целое число от 0 до " <<
totalElements << ". Повторите ввод: ";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n'
);
        } else {
            break;
        }
    }
    fillSetAAutomatically(kratnost, setB, nonZeroCount);
    cout << "Множество B заполнено автоматически.\n";
    cout << "Чтобы увидеть результаты операций над мультимножества,
нажмите 'o'\n";
}
}

```

2.15 Функция handfeel

Вход: ссылка на универсум, кратности универсума, ссылка на кратности мультимножества A, ссылка на кратности мультимножества B, bool значение

Выход: мультимножества, заполненные пользователем

Функция handfeel предназначена для ручного заполнения мультимножеств. Она принимает флаг isA, указывающий, какое множество заполняется. Сначала функция информирует пользователя о выборе ручного заполнения и предупреждает, что кратности элементов мультимножества не должны превышать кратности элементов универсального множества. В зависимости от значения isA, функция предлагает пользователю ввести кратности элементов для соответствующего множества. Если isA истинно, пользователь вводит кратности для множества A, после чего выводится запрос о способе заполнения множества B (вручную или автоматически). Если же isA ложно, функция вызывает fillSet для заполнения множества B, а затем предлагает пользователю нажать 'о', чтобы увидеть результаты операций над мультимножествами.

Ознакомиться с кодом можно в Листинг 15.

Листинг 15: Функция handfeel

```
void handfeel(vector<vector<int>> & uni, vector<int> kratnost,
    vector<int> &setA, vector<int> &setB, bool isA, int n){
    cout << "Выбрано_заполнение_множеств_вручную.\n";
    cout << "Кратность_элементов_мультимножества_не_должна_превышать_
        кратности_элементов_универсума.\n";
    if(isA){
        cout << "Введите_кратности_элементов_мультимножества_A.\n"
            ;
        fillSet(kratnost, setA);
        cout << "Выберите_способ_заполнения_множества_B_('h'_-_
            вручную,_'a'_-_автоматически):_\n";
    }
    else{
        fillSet(kratnost, setB);
        cout << "Чтобы_увидеть_результаты_операций_над_
            мультимножествами_нажмите_'о'\n";
    }
}
```

2.16 Функция fillSetAutomatically

Вход: ссылка на кратности универсума, ссылка на кратности мультимножества, ссылка на количество ненулевых значений

Выход: заполненное мультимножество

Функция `fillSetAAutomatically` автоматически заполняет мультимножество заданными кратностями. Сначала создается вектор доступных индексов от вектора кратностей и перемешивает их для случайного выбора. Затем, для заранее указанного количества ненулевых значений `nonZeroCount`, функция присваивает элементам множества случайные значения от 1 до соответствующей кратности из вектора `kratnost`. Для остальных элементов множества устанавливает значение 0.

Ознакомиться с кодом можно в Листинг 16.

Листинг 16: Функция `fillSetAAutomatically`

```
void fillSetAAutomatically(const vector<int>& kratnost, vector<int>
    & setA, int nonZeroCount) {
    vector<int> availableIndices;
    for (int i = 0; i < kratnost.size(); ++i) {
        availableIndices.push_back(i);
    }
    random_device rd;
    mt19937 gen(rd());
    shuffle(availableIndices.begin(), availableIndices.end(), gen);
    for (int i = 0; i < nonZeroCount && i < availableIndices.size();
        ++i) {
        int index = availableIndices[i];
        setA[index] = 1 + rand() % kratnost[index];
    }
    for (int i = 0; i < setA.size(); ++i) {
        if (setA[i] == 0) {
            setA[i] = 0;
        }
    }
}
```

2.17 Функция `fillSet`

Вход: ссылка на кратности юниверсума, ссылка на кратности мультимножества

Выход: заполненное мультимножество

Функция `fillSet` предназначена для ручного заполнения множества. Функция перебирает каждый элемент кратностей и запрашивает у пользователя ввод кратности для соответствующего элемента множества. Внутри цикла проверяется корректность ввода: если введенное значение не является целым числом или выходит за пределы допустимого диапазона (от 0 до максимальной кратности для данного элемента), пользователю отображается сообщение

об ошибке, и он может попробовать снова. Как только ввод корректен, значение присваивается элементу множества setA, и цикл переходит к следующему элементу.

Ознакомиться с кодом можно в Листинг 17.

Листинг 17: Функция fillSet

```
void fillSet(vector<int> const &kratnost, vector<int> &setA) {
for (int i = 0; i < kratnost.size(); ++i) {
    int input;
    while (true) {
        cout << "Кратность_для_элемента_" << i << " "_
максимум_" << kratnost[i] << "):_";
        cin >> input;
        if (cin.fail()) {
            cout << "Ошибка!_Введите_целое_число.\n";
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max
(), '\n');

            continue;
        }
        if (input < 0 || input > kratnost[i]) {
            cout << "Кратность_не_может_быть_
негативной_или_превышать_максимальную_" << kratnost[i] << ")._
Повторите_ввод.\n";

            continue;
        }
        setA[i] = input;
        break;
    }
}
}
```

2.18 Функция isnum

Вход: константная ссылка на строку

Выход: булево значение

Функция isnum проверяет, состоит ли строка только из чисел. Функция сначала проверяет, что строка не пуста. Затем использует метод, чтобы определить, есть ли в строке символы, не входящие в набор цифр (0-9). Если таких символов нет, функция возвращает true, в противном случае возвращается false.

Ознакомиться с кодом можно в Листинг 18.

Листинг 18: Функция isnum

```
bool isnum(string const &str){
    return !str.empty() && str.find_first_not_of("0123456789")
    == string::npos;
}
```

2.19 Функция printnull

Вход: нулевая разрядность юниверсума

Выход: результат работы программы для нулевого юниверсума

Функция printnull выводит в консоль сообщение – результат программы в случае пустого множества (разрядность юниверсума 0).

Ознакомиться с кодом можно в Листинг 19.

Листинг 19: Функция printnull

```
void printnull(){
    cout << "Юниверсум_пустое_множество\n";
    cout << "Мультимножество_A_пустое_множество\n";
    cout << "Мультимножество_B_пустое_множество\n";
    cout << "Объединение_AиB_пустое_множество\n";
    cout << "Пересечение_AиB_пустое_множество\n";
    cout << "Разность_AиB_пустое_множество\n";
    cout << "Разность_BиA_пустое_множество\n";
    cout << "Симметрическая_разность_AиB_пустое_множество\n";
    cout << "Дополнение_A_пустое_множество\n";
    cout << "Дополнение_B_пустое_множество\n";
    cout << "Арифметическая_сумма_AиB_пустое_множество\n";
    cout << "Арифметическая_разность_AиB_пустое_множество\n";
    cout << "Арифметическое_произведение_AиB_пустое_множество\n";
    cout << "Арифметическое_деление_AиB_пустое_множество\n";
}
```

2.20 Функция randomgenerate

Вход: целое число

Выход: заполненная кратность юниверсума

Функция randomgenerate заполняет вектор кратностей юниверсума.

Ознакомиться с кодом можно в Листинг 20.

Листинг 20: Функция randomgenerate

```
vector <int > randomgenerate(int n)
{
    vector <int > multiplicity(1<<n, 0);
    for (int i = 0; i < 1 << n; i++)
    {
        multiplicity[i]=1+ rand() % 10;
    }
    return multiplicity;
}
```

2.21 Функции sup и gray

Вход sup: номер подмножества

Выход sup: номер изменяемого разряда

Вход gray: разрядность юниверсума

Выход gray: последовательность кодов юниверсума

Функции реализуют алгоритм построения бинарного кода Грея из раздела 1.4.

Ознакомиться с кодом можно в Листинг 21.

Листинг 21: Функции sup и gray

```
int sup(int i) {
    int q = 1;
    int j = i;
    while (j % 2 == 0)
    {
        j /= 2;
        q += 1;
    }
    return q;
}

vector <vector <int>> gray(int n){
    vector <int> B(n, 0);
    vector <vector <int >> C;
    C.push_back(B);
    for (int i = 1; i < 1<<n; i++)
    {
        int p = sup(i);
        B[p-1] = 1 - B[p-1];
        C.push_back(B);
    }
}
```

```

        return C;
    }

```

2.22 Функция sumVector

Вход: ссылка на вектор

Выход: сумма значений вектора

Функция sumVector суммирует значения вектора.

Ознакомиться с кодом можно в Листинг 22.

Листинг 22: Функция sumVector

```

int sumVector(const vector<int>& vec) {
    int sum = 0;
    for (const int& num : vec) {
        sum += num;
    }
    return sum;
}

```

2.23 Функция printm

Вход: ссылка на юниверсум, кратности юниверсума, сумма кратностей юниверсума

Выход: вывод значений в консоль

Функция printm выводит элементы и кратности юниверсума.

Ознакомиться с кодом можно в Листинг 23.

Листинг 23: Функция printm

```

void printm(vector<vector<int>> const &uni, vector<int> kratnost,
    int sumkrat){
    for(int i = 0; i < uni.size(); i++){
        for(int j = uni[i].size() - 1; j >= 0; j--){
            cout << uni[i][j];
        }
        cout << "□-□кратность□" << kratnost[i] << '\n';
    }
}

```

2.24 Функция print

Вход: ссылка на юниверсум, кратности юниверсума, сумма кратностей юниверсума

Выход: вывод значений в консоль

Функция print выводит элементы, кратности юниверсума и мощность юниверсума.

Ознакомиться с кодом можно в Листинг 24.

Листинг 24: Функция print

```
void print(vector<vector<int>> const &uni, vector<int> kratnost,
int sumkrat){
    for(int i = 0; i < uni.size(); i++){
        for(int j = uni[i].size() - 1; j >= 0; j--){
            cout << uni[i][j];
        }
        cout << "□-□кратность□" << kratnost[i] << '\n';
    }
    cout << "Мощность:□" << sumkrat << '\n';
}
```

3 Результаты работы программы

После запуска программы высвечивается меню, где пользователь может ввести разрядность юниверсума или выйти из программы – рис. 2.

Рис. 2: Меню

После ввода разрядности выводится юниверсум со случайными кратностями элементов и мощность юниверсума. Далее пользователю предоставляется выбор способа для ввода множества A – рис. 3.

Рис. 3: Юниверсум и выбор способа заполнения мультимножества A

При выборе ручного заполнения пользователю предлагается ввести кратность каждого элемента мультимножества A . Далее пользователь выбирает способ ввода множества B – рис 4.

Рис. 4: Заполнение мультимножества A и выбор способа заполнения множества B

При выборе автоматического заполнения пользователю предлагается ввести количество ненулевых элементов мультимножества B . Далее пользователь может вывести результаты операций над мультимножествами. – рис 5.

Рис. 5: Заполнение мультимножества В

Выводятся заполненные мультимножества и результаты операций над ними.

Рис. 6: Мультимножества А и В

Объединение и пересечение – рис 7.

Рис. 7: Объединение и пересечение мультимножеств А и В

Дополнение – рис 8.

Рис. 8: Дополнение мультимножеств A и B

Разность и симметрическая разность – рис 9.

Рис. 9: Разность и симметрическая разность мультимножеств A и B

А также арифметические операции: сумма, разность, произведение, деление – рис 10.

Рис. 10: Арифметические операции с мультимножествами A и B

После вывода результатов операций пользователь может или выйти из программы, или перезапустить её – рис 11.

Рис. 11: Выход или перезапуск программы

Вывод, если ввести 0 в разрядность юниверсума – рис. 15.

Рис. 12: Юниверсум с разрядностью 0

Варианты вывода ошибок при некорректном вводе.

Рис. 13: Введён символ не из списка

Рис. 14: Попытка ввода буквы в разрядность юниверсума

Рис. 15: Ввод количества ненулевых элементов больший, чем элементов юни-версума

Заключение

В процессе выполнения лабораторной работы была реализована программа генерации бинарного кода Грея для заполнения юниверсума мультмножеств, на основе которой создан юниверсум и 2 мультимножества. Пользователь вводит разрядность юниверсума. Мультимножества имеют 2 способа заполнения – вручную и автоматически. С помощью множества функций реализуются все операции мультимножествами: объединение, дополнение, пересечение, разность, симметрическая разность, арифметические сумма, разность, произведение и деление и выводятся для пользователя.

Достоинства программы:

1. отсутствие утечек памяти;
2. возможность перезапуска программы без перезапуска кода.

Недостатки программы:

1. недостаточная читаемость кода из-за выполнения всей лабораторной в одном `сpp` файле;
2. использование `rand()` может привести к получению предсказуемых последовательностей чисел.

Масштабирование:

Можно внедрить графический интерфейс, что повысит доступность программы. Программа потенциально ограничена в размере векторов из-за использования 2^n . Следует рассмотреть возможность использования более эффективных структур данных - `set`, `multiset`.

Список литературы

- [1] Новиков, Ф.А. ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ПРОГРАММИСТОВ / Ф.А. Новиков. - 3-е издание. - Питер :Питер Пресс, 2009. - 384 с (Дата обращения 18.11.24).
- [2] Операции над мультимножествами. Ссылка на источник – <https://studfile.net/preview/1399244/page:18/> (Дата обращения 24.11.24).