МИНОБРНАУКИ РОССИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

«САНКТ-ПЕТЕРБУРГСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ ПЕТРА ВЕЛИКОГО»

Институт Компьютерных наук и кибербезопасности
Высшая школа технологий искусственного интеллекта
Направление 02.03.01 Математика и компьютерные науки

Отчет по лабораторной работе №1

Реализация двоичного кода Грея, мультимножеств на его основе и операций над мультимножествам

Обучающийся: _____ Гладков И.А.

Руководитель: _____ Востров А.В.

«_____»_____20____г.

Санкт-Петербург, 2024

Содержание

Введение

Задача лабораторной работы заключается в реализации программы, генерирующей код Грея для заполнения универсума разрядностью, заданной пользователем. На основе универсума создаются два мультимножества A и B. Их заполнение производится либо автомотически, либо вручную, в зависимомости от выбора пользователя. Мощность множеств задает пользователь. Как итог на экран выводятся универсум, множества A и B, а также возможность произвести операции на данными множествами: объединение, пересечение, разность, арифметическая разность, дополнение, симметрическая разность, арифметическая сумма и арифметическое произведение.

Лабораторная работа реализована на языке C++, в среде разработки Visual Studio 2022.

1 Математическое описание

1.1 Множество

Множество – это математический объект, представляющий собой совокупность уникальных элементов, неупорядоченных между собой. Элементы множества могут быть числами, буквами, символами или любыми другими объектами. В математике множества часто обозначают заглавными буквами, например, A или B.

Множество может быть пустым, обозначается ∅. Так же существует универсум, множество, содержащее все объекты и все множества. Обозначает U. Пример множества: A = {3, 6, 1, 7, 4}

1.2 Мультимножество

Пусть $\tilde{X} = \langle a_1(x_1), ..., a_n(x_n)\rangle$ – мультимножество над множество $X = \{x_1, ..., x_n\}$. 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 $\tilde{X} = [a^0b^3c^4]$ –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.

∘ 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.

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

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

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

∘ Для двух мультимножеств A и B, *разность* обозначается как A\B. Результатом этой операции будет мультимножество, содержащее элементы, которые одновременно при-

сутствуют в A и $\overline{B}$, и кратность каждого такого элемента будет равна минимальной кратности мультимножеств A и $\overline{B}$.

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

∘ *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 $\overline{\text{A}}$ 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.

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

∘ *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.

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

∘ 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.

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

∘ Для двух мультимножеств A и B, *арифметическая сумма* обозначается как A + B. Результатом данной операции является мультимножество, состоящее из всех элементов мультимножеств A и B, кратность каждого элемента равна сумме кратностей в складываемых мультимножествах. Но сумма кратностей не может превышать кратности в универсуме.

$$C = A - B = \{k^{min\{m_{i1} + m_{i2}, m_{iU}\}} : k^{m_{i1}} \in A \vee k^{m_{i2}} \in B, k^{m_{iU}} \in U\}$$

∘ 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.

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

1.3 Бинарный код Грея

Код Грея представляет собой последовательность двоичных чисел, в которой каждое следующее число отличается от предыдущего всего лишь одним битом. Формируется относительно заданной разрядности при помощи особого алгорима.

Пример кода Грея разрядности 2:

00, 01, 10, 11

Ниже, в **листинге 1**, представлен алгоритм, генерурующий последовательность всех подмножеств n-элементного множества, где каждое следующее подмножество получается из предыдущего удалением и добавлением в точности одного элемента. Вход: n >= 0 - мощность множества. Выход: последовательность кодов множества B

```
Вход: n >= 0 - мощность множества.
Выход: последовательность кодов множества B
B: array[1..n] of 0..1 {Битовая шкала для представления подмножеств}
for i from 1 to n do: B[i]:= 0 for {инициализация}
yeild B{пустое множество}
for i from 1 to 2^n - 1 do:
p:= Q[i] {Определение номера элемента, подлежащего добавлению или удалению}
B[p]:= 1 - B[p] {Добавление или удаление элемента}
yeild B {очередное подмножество}
end for

```

Листинг 1. Построение бинарного кода Грея

Ниже, в **листинге 2**, представлена функция, с помощью которой определяется номер разряда, подлежащего изменению, возвращает количество нулей на конце двоичной записи числа i, увеличенное на 1

Вход: i - номер подмножества

Выход: номер измеряемого разряда.

```
Вход: i - номер подмножества
Выход: номер измеряемого разряда
q:= 1
j:= i
while (j % 2 = 0) do
```

```
6   j:= j/2
7   q:= q + 1
8   end while
9   return q
10
```

Листинг 2. Функция определения номера измеряемого разряда

2 Особенности реализации

2.1 Класс GrayCode

Класс GrayCode – основной класс в моей лабораторной работе, он отвечает за универсум и код Грея. Он является базовым классом для класса содержащего мультимножество.

Класс содержит переменные:

- int **n** – разрядность кода Грея
- int **k** – количесво чисел в коде Грея $k = 2^n$
- int **Count** – количество различных элементов
- vector <int> **Codes** – массив содержаший код Грея
- vector <int> **Krat** – массив сожержащий кратность каждого элемента кода Грея

2.1.1 Конструктор с параметром GrayCode

Вход: кратность универсума.

Выход: сформированный универсум.

Переменной **n** присваивается значение передваемой переменной ***a***, переменная **k** считается по формуле $k = 2^n$ с помощью *for*, **Codes** увеличивается до размера k × n с помощью метода *resize* (далее будет понятно почему). После создается код Грея по созданному мною алгориму:
(Напишем элементы кода Грея разрядности 3 друг под другом)
000
001
010
011
100
101
110
111

Если посмотреть на каждый столбец (их всего 3 в данном случае), заметим что в первом "0"сменяется на "1"(а также "0"на "1"в дальнейшем) через **k**/2, то есть 8/2=4, во втором в два раза чаще, то есть через 4/2=2 числа, и так далее. для этого создается переменная **buf** = **k**/2, который будет использоваться и зменяться в алгоритме. При помощи этой перемнной и нескольких проверок в *for* будет ставится либо "0 либо "1". Также все эти столбцы записываются попорядку в **Codes**, то есть "000011110011001101010101"для этого нужно было создать массив размером k × n.

Потом **Krat** увеличивается с помощью метода *resize* до размера **k**. Затем туда записывается случайное значение кратности для каждого элемента в коде Грея. Код конструктора представлен в **листинге 3**, ниже.

```
GrayCode::GrayCode(int a) {
  n = a;
  if (a == 0)
  k = 0;
  else
  k = 2;
  for (int i = 0; i < n - 1; i++) {
    k *= 2;
  }
  Count = k;
  Codes.resize(k * n);
  int buf = k / 2;
  for (int m= 0; m < n; m++) {
    static int i = 0;
    static int p = 1;
    int f = 0;
    for (i; i < k * p; i++) {
      if (f < buf)
      Codes[i] = 0;
      else
      Codes[i] = 1;
      f++;
      if (f == buf * 2)
      f = 0;
    }
    buf = buf / 2;
    p++;
  }
  Krat.resize(k);
  srand(time(0));
  max = 0;
  for (int i = 0; i < k; i++) {
    Krat[i] = rand() % 99 + 1 ;
    max += Krat[i];
  }
}
```

Листинг 3. Конструктор GrayCode

2.1.2 Методы Getk, Getn, GetCodes, GetKrat, Print

Метод *Getk*, возвращает значение переменной **k**, то есть количесво чисел в коде Грея $k = 2^n$. Этот метод необходим в формировании мультимножества: используется в циклах for, чтоб не было выхода за границы массива.

Вход: получение количества чисел в коде Грея.

Выход: количесво чисел в коде Грея.

```
int GrayCode::Getk() {
  return k;
}
```

Листинг 4. Метод Getk

Метод *Getn()* возвращает значение переменной **n**, то есть разрядность кода Грея. Используется также при формировании мультимножеств: когда у пользователя спрашивают какого кратности будет то или иное число, которое показывается в строке припомощи данного метода.

Вход: получение разрядности кода Грея.

Выход: разрядность кода Грея.

```
int GrayCode::Getn() {
  return n;
}
```

Листинг 5. Метод Getn

Метод *GetCodes()* возвращает массив **Codes**, содержащий код Грея. Необходим в формировании мультимножеств, так как класс Multi не создает заного код Грея, а берет их класса GrayCode.

Вход: получение массива, содержащего код Грея.

Выход: массив, содержащий код Грея.

```
vector <int> GrayCode::GetCodes() {
  return Codes;
}
```

Листинг 6. Метод GetCodes

Метод *GetKrat()* массив **Krat**, содержащий крастность элементов в универсуме. Необходим при создании мультимножеств, для корректного заполнения. В мульти множестве кратность элемента не должна превышать кратности в универсуме. Метод используется для проверок данного правила.

Вход: получение массива, содержащего кратность элементов в универсуме.

Выход: массив, содержащий крастность элементов в универсуме.

```
vector <int> GrayCode::GetKrat() {
  return Krat;
}
```

Листинг 7. Метод GetKrat

Метод *Print()* Выводит на экран универсум. Так как весь код Грея представлен в одном массиве, необходимо проходить по массиву с определенным шагом и выводить по одному символу.

Вход: намерение увидеть универсум на консоли

Выход: печать универсума на консоль

```
void GrayCode::Print() {
  cout << "\n  УНИВЕРСУМ\n\n";

  if (n == 0) {
    printf("Пуст..\n");
  }
  else {
    for (int i = 0; i < k; i++) {
      if (Krat[i] != 0) {
        cout << "   ";
        for (int j = 0; j < k * n; j += k) {
          cout << "  " << Codes[i + j];
        }

        cout << "  [" << i << "]" << " :  " << Krat[i] << "\n";
      }
    }
  }
}
```

Листинг 8. Метод Print

2.2 Класс Multi

Класс Multi отвечает за мультимножества. Данный класс наследник от класса GrayCode. В нем создается мультимножество на основе универсума. Экземпляры данного класса будут различными мультимножествами, то есть один экземпляр отвечает за одно мультимножество. Он содержит переменные:

- int **Count** – количесво различных элементов в мультимножестве
- vector <int> **elems** – массив содержаший кратность для каждого числа
- GrayCode* **base** – указатель на экземпляр класса GrayCode содержащий универсум
- vector<bool **Show** – массив флагов, показывающие, выбран ли тот или иной элемент универсума в мультимножестве.

2.2.1 Конструктор с параметрами Multi

Вход: Количество различных элементов, указатель на экземпляр класса с универсумом и переменная, показывающая, будет ли этот экземпляр класса входным мультимножеством или резельтатом операций над входными мультимножествами.

Выход: Заготовка для формирования универсума.

Переменной **Count** присваивается передаваемое значение a, то есть мощность мультимножества. Далее указателю **base** присваивается адресс экземпляра класса GrayCode. После массив **elems** увеличивается до размера количества различных чисел в универсуме. Кратность каждого равна нулю. Затем пользователь решает, будет он выбирать различные элементы мультимножества в ручную или автоматически. Если пользователь выбирает заполнить вручную, то он вводит по одному элементы, если автоматически, то элементы выбираются случайно. Код конструктора представлен ниже, в **листинге 10**.

```
1  Multi::Multi(int a, GrayCode* b, bool flag) {
2    Count = a;
3    base = b;
4    elems.resize(base->Getk(), 0);
5    Show.resize(base->Getk(), 0);
6
7    if (base->Getk() != 0 && flag) {
8
9      regex powerful("^[0-9]{0,50}$");
10
11     printf("Хотите ли вы конкретные ненулевые значения - [1], или же их выбрать автомат
       ически - [0]?\n");
12     string s;
13     while (true) {
14       cin >> s;
15       if (s == "0" or s == "1")
16       break;
```

```
17        else printf("\nВведите либо [1] либо [0]\n");
18      }
19      if (stoi(s) == 1) {
20        printf("\nВводите по одному элементы\n");
21        for (int i = 0; i < Count; i++) {
22          cin >> s;
23          if (regex_match(s, powerful) && stoi(s) >= 0 && stoi(s) < this->base->Getk()) {
24            if (Show[stoi(s)] == 1) {
25              printf("Ну зачем же вы пишете повторяющийся элемент?\n");
26              i--;
27            }
28            else
29            Show[stoi(s)] = 1;
30          }
31          else {
32            std::cout << "Ошибка ввода!\n";
33            i--;
34
35          }
36        }
37      }
38      else {
39        srand(time(0));
40        for (int i = 0; i < Count; i++) {
41          int r = rand() % base->Getk();
42          if (Show[r] == 1)
43          i--;
44          else {
45            Show[r] = 1;
46          }
47        }
48      }
49    }
50  }
```

Листинг 9. Конструктор Multi

2.2.2 Методы Avto(), Hand(), Print(string s)

∘ Метод *Avto()* заполняет массив**elems** случайными кратностостями элементы мультимножества. Проходится по массиву, заполняя значения кратности рандомно, при этом проводится проверка, чтобы кратность была не больше чем в универсуме.

Вход: намерение сформировать мультимножество автоматически.

Выход: сформированное мультимножество.

```
1     void Multi::Avto() {
2       srand(time(0));
```

```
    for (int i = 0; i < base->Getk(); i++) {
      if (Show[i] != 0) {
        int r = rand() % base->GetKrat()[i];
        elems[i] = r;
      }
    }
}

```

Листинг 10. Метод Avto

∘ Метод *Hand()* заполняет массив **elems** кратностями элементов, вводимыми с консоли, пользователем. В цикле каждый раз у пользователя спрашивают какая кратность будут у того или иного числа, и производится проверка на корректность пользовательского ввода и на то, чтобы вводимая кратность не превышала кратности в универсуме.

Вход: намерение сформировать мультимножество пользовательским вводом

Выход: сформированное мультимножество

```
void Multi::Hand() {

  cout << "\n-------------------------------------";
  cout << "\nВы решили заполнить массив вручную\n\n";

  srand(time(0));

  for (int i = 0; i < base->Getk(); i++) {
    if (Show[i] != 0) {
      cout << "Какая кратность будет у этого числа? ";
      for (int j = 0; j < base->Getn(); j++) {
        cout << base->GetCodes()[i + j * base->Getk()]  ;
      }
      cout <<" ["<<i<<"]: "<< base->GetKrat()[i]<<"\n";
      string bit;
      regex powerful("^[0-9]{0,50}$");
      while (true) {
        cin >> bit;
        if (regex_match(bit, powerful) && stoi(bit) >= 0 && stoi(bit) <= this->base->GetKrat()[i]) {
          break; // Ввод соответствует регулярному выражению, выходим из цикла
        }
        else {
          std::cout << "Ошибка ввода!\n";
        }
      }
      elems[i] = stoi(bit);
    }
```

```
28      }
29
```

Листинг 11. Метод Hand

∘ Метод *Print(string s)* выводит на экран мультимножество и название *string s*, которое вводится в параметрах метода. Вывод на экран аналогичный универсуму, только перед ним пишется название универсума.

Вход: название мультимножества

Выход: вывод в консоль мультимножество с его названием.

```
1     void Multi::Print(string s) {
2       printf("\n");
3       cout << "\n    "<<s<<"\n\n";
4       bool flag = 1;
5       for (int i = 0; i < base->Getk(); i++) {
6         if (Show[i] != 0) {
7           cout << "   ";
8           for (int j = 0; j < base->Getk() * base->Getn(); j += base->Getk()) {
9             cout << "  " << (base->GetCodes().at(i+j));
10            flag = 0;
11          }
12
13          cout << "  [" << i << "]"<< " :  " << elems[i] << "\n";
14        }
15      }
16      if (flag)
17      printf("Пустое множество\n");
18      printf("\n\n");
19
```

Листинг 12. Конструктор Print

2.2.3 Метод Unification – объединение мультимножеств A и B

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции объединения мультимножеств A и B.

Функция выполняет операцию объединения двух мультимножеств A и B, то есть находит максимум из двух кратностей одинаковых элементов в двух мультимножествах передаваемых через параметры функции. Максимальная кратность элемента записывается в **this->elems[i]**, то есть в экземпляр класса, в котором выполняется данный метод.

```
1   void Multi::Unification(Multi& a, Multi& b) { //объединение
2     for (int i = 0; i < base->Getk(); i++) {
3       this->elems[i] = a.elems[i] > b.elems[i] ? a.elems[i] : b.elems[i];
4       if (elems[i] != 0)
```

```
      Show[i] = 1;
    }
  }

```

Листинг 13. Метод Unification

2.2.4 Метод Intersection – пересечение мультимножеств A и B

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции пересечения мультимножеств A и B.

Функция выполняет операцию пересечения двух мультимножеств A и B, то есть находит минимум из двух кратностей одинаковых элементов в двух мультимножествах переваемых через параметры функции. Минимальная кратность элемента записывается в **this->elems[i]**, то есть в экземпляр класса, в которм выполняется данный метод.

```
void Multi::Intersection(Multi& a, Multi& b) { //Пересечение
  for (int i = 0; i < base->Getk(); i++) {
    this->elems[i] = a.elems[i] < b.elems[i] ? a.elems[i] : b.elems[i];
    if (elems[i] != 0)
    Show[i] = 1;
  }
}
```

Листинг 14. Метод Intersection

2.2.5 Метод DifferenceA – разность мультимножеств A и B

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции разности мультимножеств A и B.

Функция выполняет операцию разности двух мультимножеств A и B (A\B = $A \cap \overline{B}$), то есть определяет минимальную кратность соответствующих элементов в мультимножествах A и $\overline{B}$, и записывает результат в результирующее мультимножество.

```
void Multi::DifferenceA(Multi& a, Multi& b){ // разность множеств A и B
  for (int i = 0; i < base->Getk(); i++) {
    this->elems[i] = a.elems[i] < (base->GetKrat()[i]-b.elems[i]) ? a.elems[i] : base->
  GetKrat()[i] - b.elems[i];
    if (elems[i] != 0)
    Show[i] = 1;
  }
}
```

Листинг 15. Метод DifferenceA

2.2.6 Метод DifferenceB – разность мультимножеств B и A

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции разности мультимножеств B и A.

Функция выполняет операцию разности двух мультимножеств B и A (B\A = $\cap$), то есть определяет минимальную кратность соответствующих элементов в мультимножествах B и , и записывает результат в результирующее мультимножество.

```
1  void Multi::DifferenceB(Multi& a, Multi& b){ //разность множеств B и A
2    for (int i = 0; i < base->Getk(); i++) {
3      this->elems[i] = (base->GetKrat()[i]-a.elems[i]) < b.elems[i] ?base->GetKrat()[i]-
     a.elems[i] : b.elems[i];;
4      if (elems[i] != 0)
5      Show[i] = 1;
6    }
7  }
```

Листинг 16. Метод DifferenceB

2.2.7 Метод AdditionA – дополнение мультимножества A

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции дополнения мультимножества A.

Функция выполняет операцию дополнения мультимножества A, то есть находит разности кратностей элементов в универсуме и в мультимножестве A.

```
1  void Multi::AdditionA(Multi& a, Multi& b){ //Дополнение множества A
2    for (int i = 0; i < base->Getk(); i++) {
3      this->elems[i] =  a.base->GetKrat().at(i)-a.elems[i]>0 ? a.base->GetKrat()[i] - a.
     elems[i] : 0;
4      if (elems[i] != 0)
5      Show[i] = 1;
6    }
7  }
```

Листинг 17. Метод AdditionA

2.2.8 Метод AdditionB – дополнение мультимножества B

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции дополнения мультимножества B.

Функция выполняет операцию дополнения мультимножества B, то есть находит разности кратности элементов в универсуме и в мультимножестве B.

```
1  void Multi::AdditionB(Multi& a, Multi& b) { //Дополнение множества B
2    for (int i = 0; i < base->Getk(); i++) {
```

```
3       this->elems[i] = b.base->GetKrat().at(i) - b.elems[i] > 0 ? b.base->GetKrat()[i] -
      b.elems[i] : 0;
4       if (elems[i] != 0)
5       Show[i] = 1;
6     }
7   }
```

Листинг 18. Метод AdditionB

2.2.9 Метод Sim_Difference – симметрическая разность мультимножеств A и B

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции симметрической разности мультимножеств A и B.

Функция выполняет операцию симметрической разности двух мультимножеств A и B (A\B), то есть находит кратность элементов, находящихся либо в мультимножестве A, либо в мультимножестве B, но не находящихся одновременно в A и B. То есть если кратность в A больше кратности в B, то выполняется разность кратностей из A и B, и если кратность в B больше кратности в A, то выполняется разность кратностей B и A, а иначе кратность равна нулю.

```
1   void Multi::Sim_Difference(Multi& a, Multi& b){ //Симметрическая разность множеств A и
      B
2     for (int i = 0; i < base->Getk(); i++) {
3       this->elems[i] = a.elems[i] - b.elems[i] > 0  b.elems[i] - a.elems[i] > 0 ?
4       (a.elems[i] - b.elems[i] > 0? a.elems[i] - b.elems[i] : b.elems[i] - a.elems[i] ):
      0;
5       if (elems[i] != 0)
6       Show[i] = 1;
7     }
8   }
9
```

Листинг 19. Метод Sim_Difference

2.2.10 Метод Ar_DifferenceA – арифметическая разность мультимножеств A и B

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции арифметической разности мультимножеств A и B.

Функция выполняет операцию арифметической разности мультимножеств A и B. Производится разность кратности элемента мультимножества A и мультимножества B. Итог записывается в результирующее мультимножество, но если разность получается отрицательная, то записывается ноль.

```
1   void Multi::Ar_DifferenceA(Multi& a, Multi& b) { //Арифметическая разность множеств A и
       B
2     for (int i = 0; i < base->Getk(); i++) {
3       this->elems[i] = a.elems[i] - b.elems[i] > 0 ? a.elems[i] - b.elems[i] : 0;
```

```
4       if (elems[i] != 0)
5       Show[i] = 1;
6     }
7   }
8
```

Листинг 20. Метод Ar_DifferenceA

2.2.11 Метод Ar_DifferenceB – арифметическая разность мультимножеств B и A

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции арифметической разности мультимножеств B и A.

Функция выполняет операцию арифметической разности мультимножеств B и A. Производится разность кратности элемента мультимножества B и мультимножества A. Итог записывается в результирующее мультимножество, но если разность получается отрицательная, то записывается ноль.

```
1   void Multi::Ar_DifferenceB(Multi& a, Multi& b) { //Арифметическая разность множеств B и
        A
2     for (int i = 0; i < base->Getk(); i++) {
3       this->elems[i] = b.elems[i] - a.elems[i] > 0 ? b.elems[i] - a.elems[i] : 0;
4       if (elems[i] != 0)
5       Show[i] = 1;
6     }
7   }
8
```

Листинг 21. Метод Ar_DifferenceB

2.2.12 Метод Ar_Summ – арифметическая сумма мультимножеств A и B

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции арифметической суммы мультимножеств A и B.

Функция выполняет операцию арифметической разности мультимножеств A и B. Производится сумма кратности элемента мультимножества A и мультимножества B. Итог записывается в результирующее мультимножество, но если сумма получается больше кратности в универсуме, то записывается кратность элемента универсума.

```
1   void Multi::Ar_Summ(Multi & a, Multi & b){//арифметическая сумма A и B
2     for (int i = 0; i < base->Getk(); i++) {
3       this->elems[i] = b.elems[i] + a.elems[i] < base->GetKrat()[i] ? b.elems[i] + a.
      elems[i] : base->GetKrat()[i];
4       if (elems[i] != 0)
5       Show[i] = 1;
6     }
7   }
```

Листинг 22. Метод Ar_Summ

2.2.13 Метод Ar_Prod – арифметическое произведение мультимножеств A и B

Вход: ссылка на мультимножество A и ссылка на мультимноество B.

Выход: результат операции арифметического произведения мультимножеств A и B.

Функция выполняет операцию арифметического произведения мультимножеств A и B. Производится произведение кратности элемента мультимножества A и мультимножества B. Итог записывается в результирующее мультимножество, но если произведение получается больше кратности в универсуме, то записывается кратность элемента универсума.

```
1  void Multi::Ar_Prod(Multi & a, Multi & b){//арифметическое произведенеие A и B
2    for (int i = 0; i < base->Getk(); i++) {
3      this->elems[i] = b.elems[i] * a.elems[i] < base->GetKrat()[i] ? b.elems[i] * a.
     elems[i] : base->GetKrat()[i];
4    }
5  }
```

Листинг 23. Метод Ar_Prod

2.3 Функция main и проверка на корректность пользовательского ввода

Весь интерфейс вывода на экран прописан в функции *main*, также там используется проверка на корректность при каждом вводе пользователя, чтобы нельзя было ввести несответствующее значение. Используются регулярные выражения (regex) бля проверки ввода определенных символов, в данной лабораторной это цифры. Также вместе с регулярными выражениями используется проверка диапазона чисел, изменяющаяся от места запроса ввода данных пользователя. Пример использования в лабораторной ниже, в **листинге 25**.

Вход: желание ограничить пользовательский ввод.

Выход: ограничение пользовательского ввода.

```
regex powerful("^[0-9]{0,50}$");
//создание универсума
string buf;

cout << "Введите разрядность УНИВЕРСУМА (целое число 0-10)\n";
while (true) {
  cin >> buf;
  if (regex_match(buf, powerful) && stoi(buf) >= 0 && stoi(buf) <= 10) {
    break; // Ввод соответствует регулярному выражению, выходим из цикла
  }
  else {
    std::cout << "\nОшибка ввода! Ведите число от 0 до 10\n\n";
  }
}
int n = stoi(buf);
GrayCode m(n);
```

Листинг 24. Пример корректности пользовательского ввода

В начале у пользователя запрашивается разрядность универсума, после выводится универсум.

Затем предлагается заполнить мультимножество A, вручную или автоматически. Следующим шагом необходимо решить сколько будет уникальных элементов в мультимножестве. Далее предоставляется выбор заполнения этих элементов: вручную, либо автоматически. Пользователь выбирает, и происходит заполнение мультимножества различными элементами. При первом запросе, если было выбрано автоматическое запонение, то кратности к уникальным числам унивесума выбираются автоматически, не превышая кратности в универсуме. Если же было выбрано ручное заполнение, то программа будет запрашивать кратность от пользователя для каждого уникального элемента мультимножества. После на экран выводится сформированное мультимножество A.

Аналогичные действия производятся с мультимножеством B.

Следующим шагом создается экземпляр класса *Multi*, оно будет являтся результирующим мультимножеством операций между A и B. Затем пользователю предоставляется возможность выполнить различные действия с мультимножествами A и B, а также выйти из программы. В зависимости от пользователя, выполняется тот или иной метод класса *Multi* в экземпляре класса,

созданным чуть ранее (результирующее мультимножество).

3 Результаты работы

Ниже будут представлены илюстрации, показывающие результат проделанной работы, и то как эту программу видит пользователь.

Создание универсума разрядности, которую ввел пользователь, изображено на *рисунке 1.*

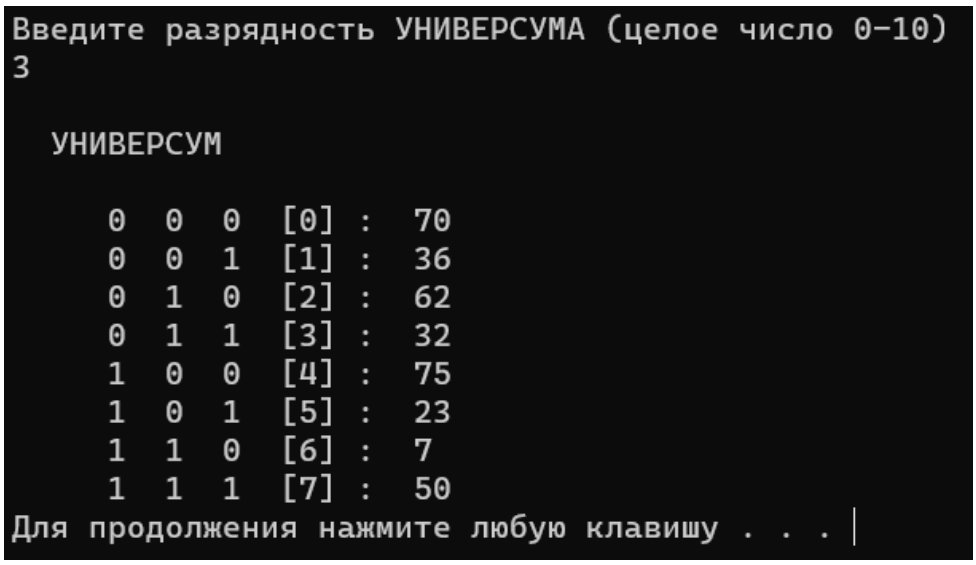

Рис. 1. Создание универсума, заданной пользователем разрядности

Формирование двух мультимножеств изображено на *рисунках 2 - 4.*

Автоматическое заполнение мультимножества A и автоматический выбор четырех различных ненулевых элементов показано на *рисунке 2.*

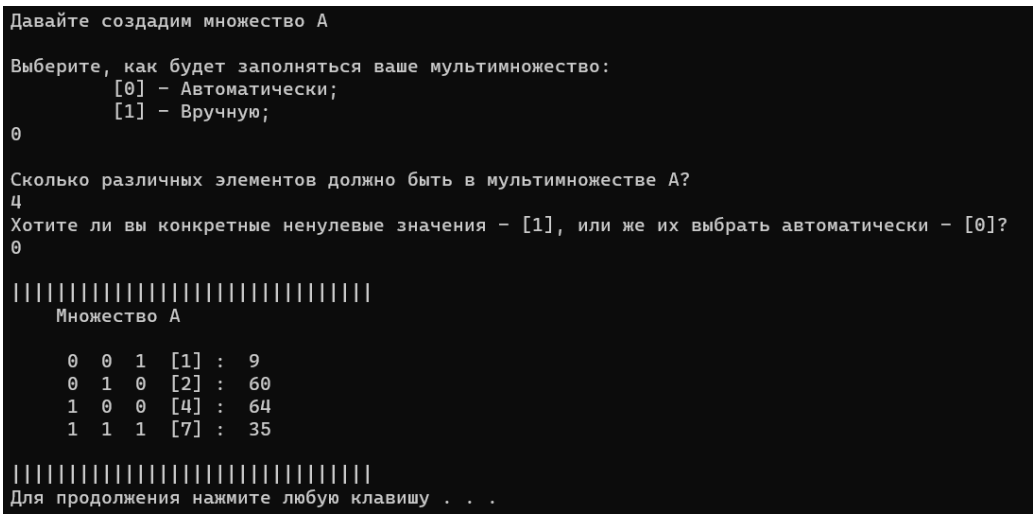

Рис. 2. Автоматическое заполнение мультимножества A и автоматический выбор четырех различных ненулевых элементов

Ввод различных ненулевых элементов в мультимножество B вручную и заполнение кратностей выбранных элементов тоже вручную можно увидеть на *рисунках 3-4*

```
------------------------------------------------
Давайте создадим множество B

Выберите, как будет заполняться ваше мультимножество:
         [0] - Автоматически;
         [1] - Вручную;
1

Сколько различных элементов должно быть в мультимножестве B?
4
Хотите ли вы конкретные ненулевые значения - [1], или же их выбрать автоматически - [0]?
1

Вводите по одному элементы
1
2
3
4
```

Рис. 3. Ввод различных ненулевых элементов в мультимножество B вручную

```
----------------------------------
Вы решили заполнить массив вручную

Какая кратность будет у этого числа? 001 [1]: 36
12
Какая кратность будет у этого числа? 010 [2]: 62
34
Какая кратность будет у этого числа? 011 [3]: 32
32
Какая кратность будет у этого числа? 100 [4]: 75
1

||||||||||||||||||||||||||||||||
    Множество B

     0  0  1  [1] :  12
     0  1  0  [2] :  34
     0  1  1  [3] :  32
     1  0  0  [4] :  1

||||||||||||||||||||||||||||||||
Для продолжения нажмите любую клавишу . . .
```

Рис. 4. Заполнение кратностей выбранных элементов вручную

Меню с возжными операциями над операциями и дополнительные действиями над множествами множествами изображено на *рисунке 5*

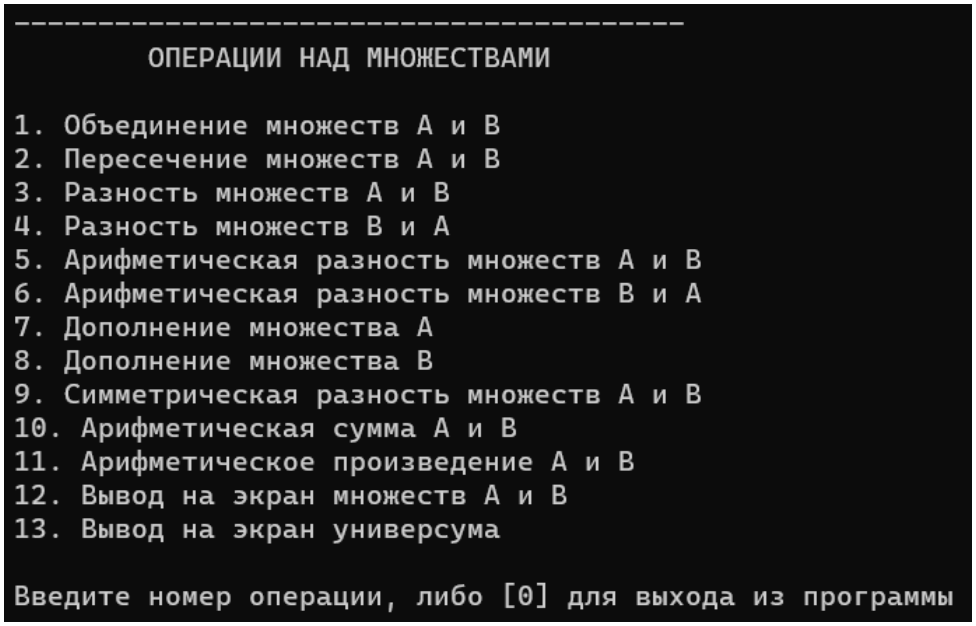

Рис. 5. Перечень операций над мультимножествами A и B

Все операции над множествами, реализованные в лабораторной работе, а также дополнительные действия показаны на рисунках 6-17

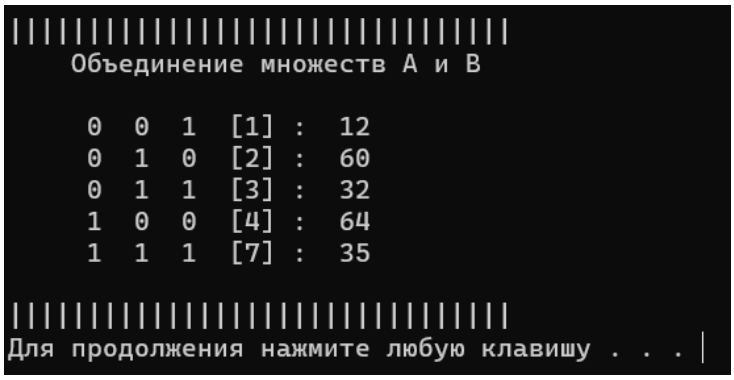

Рис. 6. Объединение мультимножеств A и B

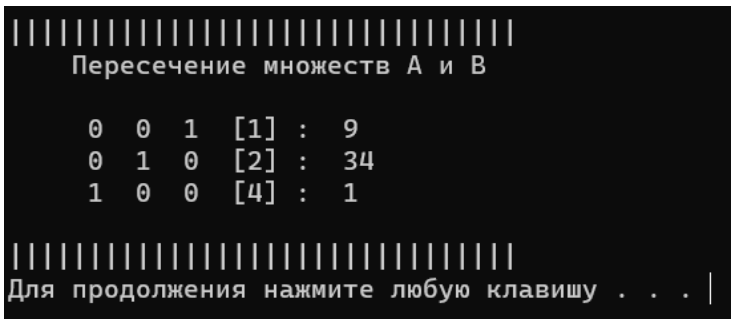

Рис. 7. Пересечение мультимножеств A и B

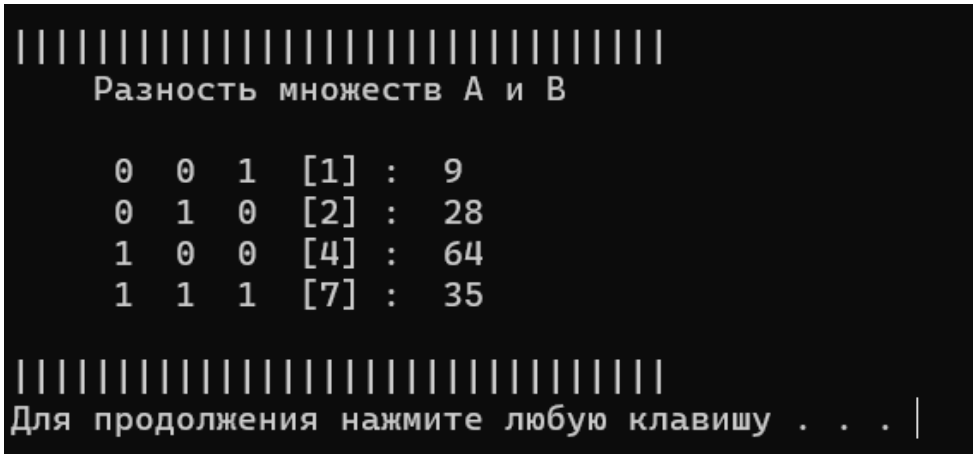

Рис. 8. Разность мультимножеств A и B

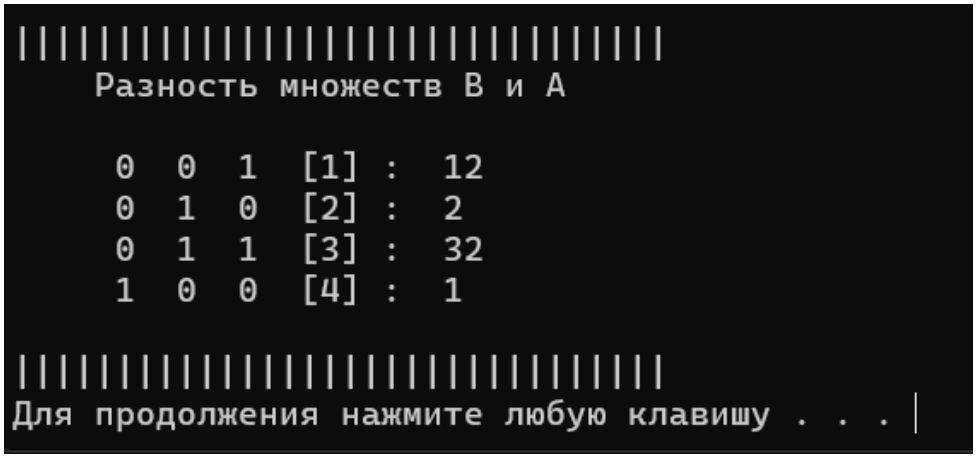

Рис. 9. Разность мультимножеств B и A

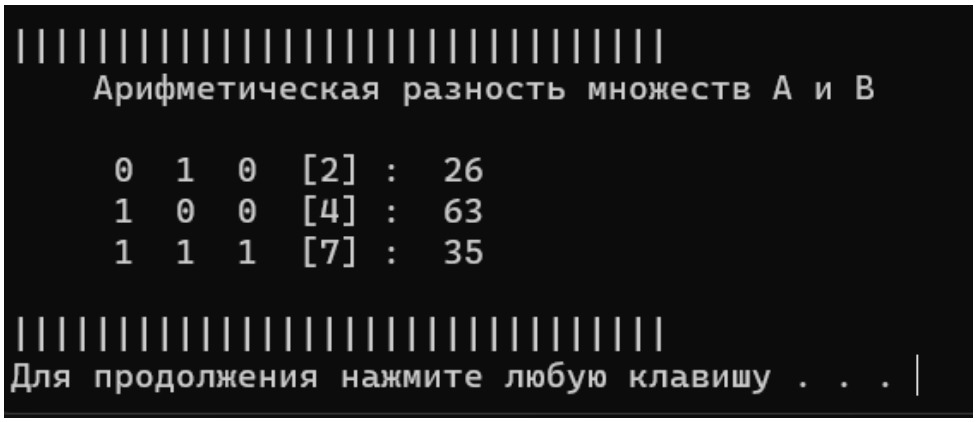

Рис. 10. Арифметическая разность мультимножеств A и B

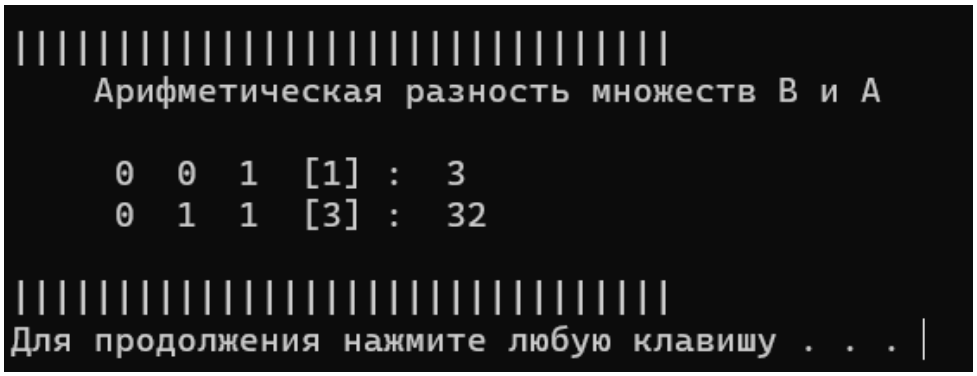

Рис. 11. Арифметическая разность мультимножеств B и A

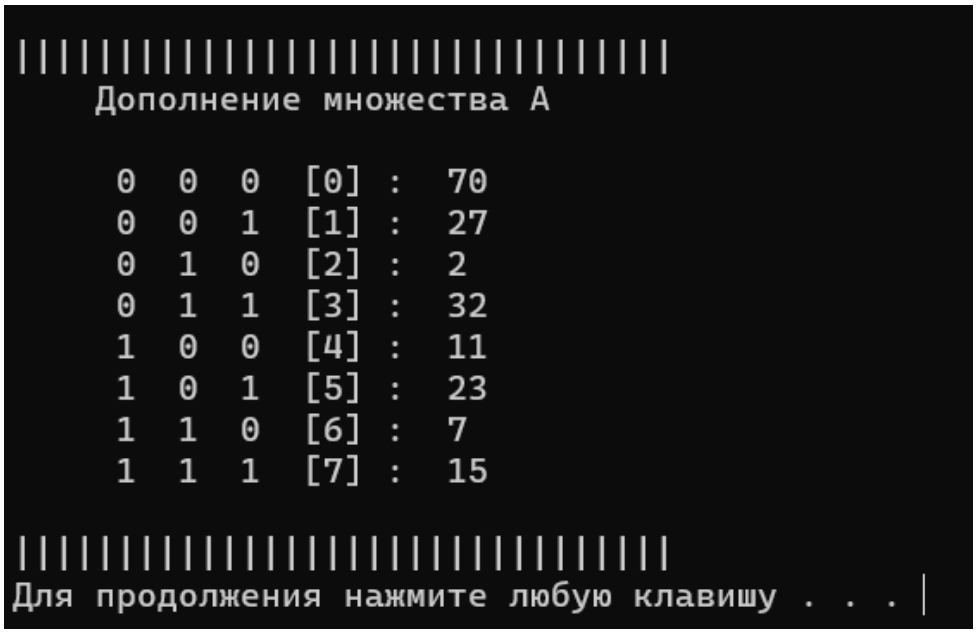

Рис. 12. Дополние мультимножества A

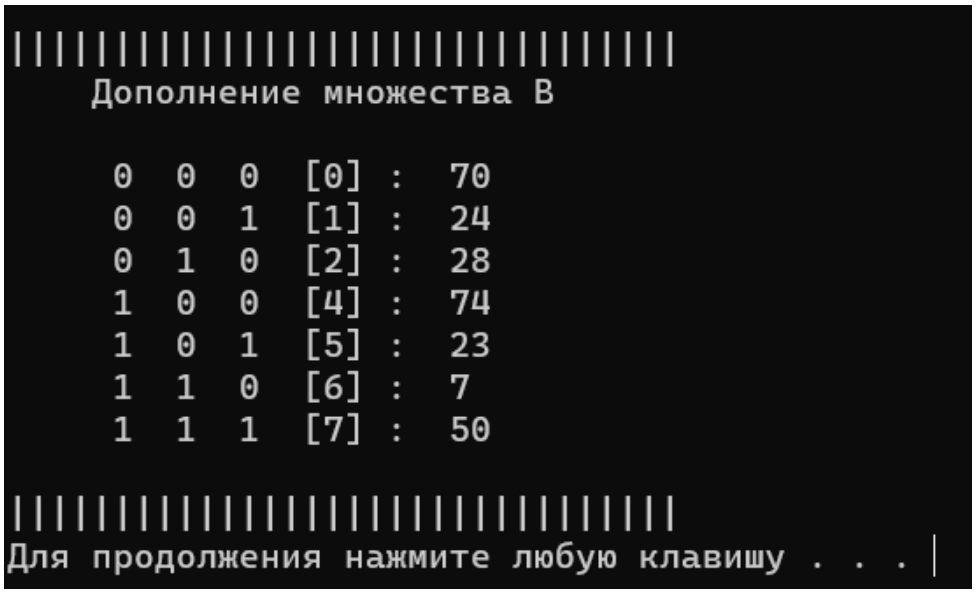

Рис. 13. Дополнение мультимножества B

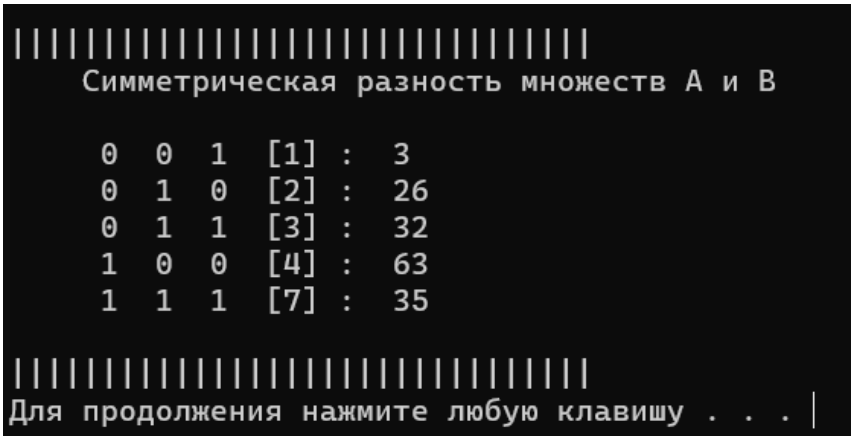

Рис. 14. Симметрическая разность мультимножеств A и B

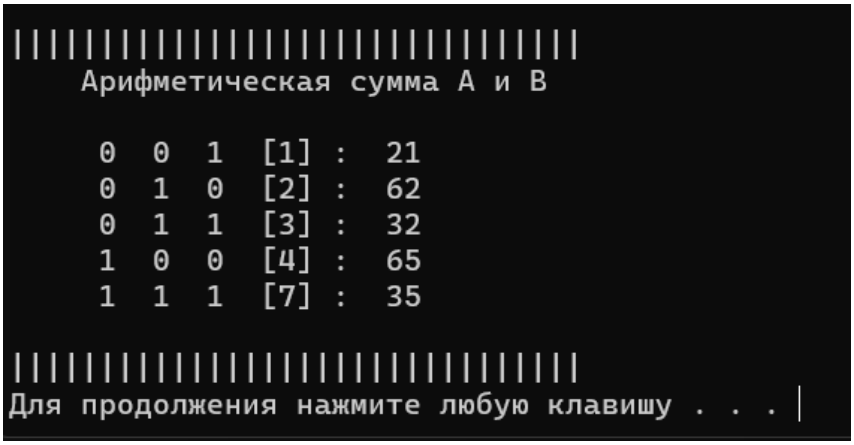

Рис. 15. Симметрическая сумма мультимножеств A и B

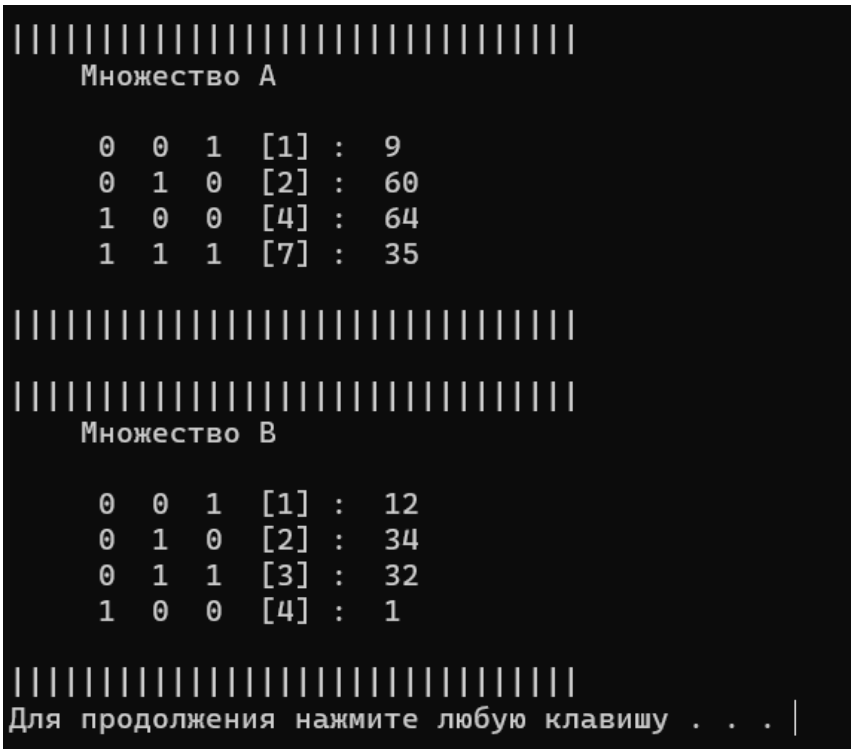

Рис. 16. Вывод на экран мультимножеств A и B

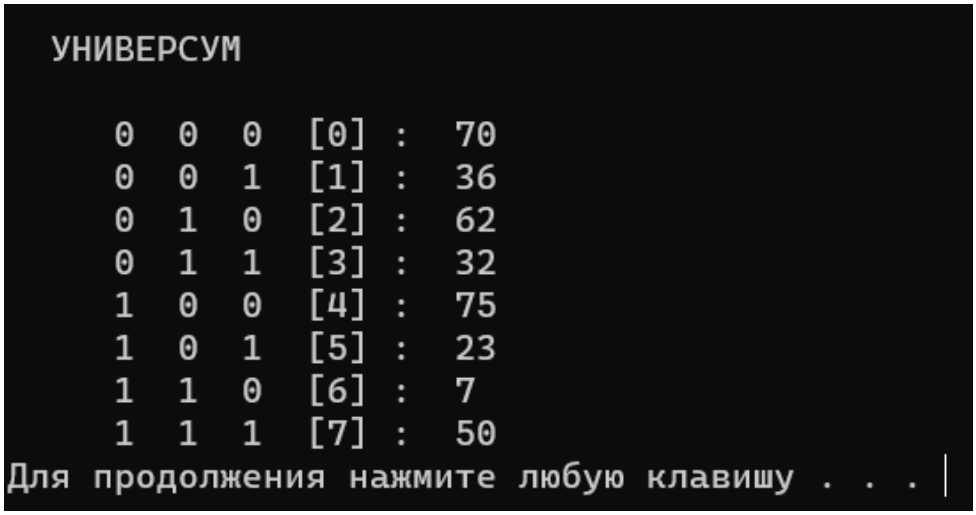

Рис. 17. Вывод на экран универсума

Заключение

В результате выполнения лабораторной работы №1 были реализованы: программа генерации основе бинарного кода Грея для заполнения универсума заданной пользователем разрядности, мультимножества и операции над ними: объединение, пересечение, разность, арифметическая разность, дополнение, симметрическая разность, арифметическая сумма, арифметическое произведение.

Из преимуществ реализованной программы, я считаю, что за счет ООП, можно использовать в других проектах классы GrayCode и Multi. Например, класс GrayCode можно использовать как генератор ланшафта, а точнее гор: уникальные элементы можно использовать как сетку, то есть квадраты которые пронумерованы попорядку, а кратность будет точкой над общим уровнем. Тогда при помощи других классов, которые будут работать с этой информацией, и както сглаживать например, а другие обрисовавать данный ландшафт, то получится, как я сяитаю, неплохая 3D модель гор.

Из недостатков я считаю, что при достаточно больших разрядностях универсума и мультимножеств будет значительное занимаение памяти, а также вывод на экран универсума и мультимножеств будет достаточно долгим. Для таких случаев стоит ограничение разрядности (оно равно 10). Также если выбрать большое количество элементов мультимножества и выбрать ручной ввод, то придется очень долго вводить по одному элементу

За счет такой реализации программы, можно без труда расширять классы GrayCode и Multi, а также совмещать с другими классами. В данном классе можно реализовать операции деления, возведения в степень, НОК, НОД и тд.

На работу было порачено примерно 15 часов. Работа была выполнена в среде разработки Visual Studio 2022 на языке c++.

Источники

1. Дискретная математика для программистов. 3-е издание. Новиков Ф.А. СПБ:.Питер,2009.