МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования «Санкт-Петербургский политехнический
университет Петра Великого»

Институт компьютерных наук и кибербезопасности

Высшая школа технологий искусственного интеллекта

Направление: 02.03.01 «Математика и компьютерные науки»

Отчет по лабораторной работе по дисциплине «Дискретная математика»
Реализация двоичного кода Грея, мультимножеств на его основе и операций над ними

Студент,
группы 5130201/20001 _____ Якунин Д. Д.

Преподаватель _____ Востров А. В.

«_____»_____ 2023г.

Санкт-Петербург, 2023

Содержание

Введение

Необходимо реализовать программу генерации бинарного кода Грея для заполнения универсума заданной пользователем разрядности. На основе универсума формируются два мультимножества двумя способами заполнения: вручную и автоматически, способ по ходу выполнения выбирает пользователь. Мощность множеств также задает пользователь. В результате на экран выводятся результаты всех действий над множествами: пересечения, объединения, дополнения мультимножеств, разность, арифметическое сложение, арифметическая разность, арифметическое произведение, симметрическая разность. Кроме того, требуется реализовать защиту от некорректного пользовательского ввода.

1 Математическое описание

1.1 Множества

Множество — это любая определённая совокупность объектов. Объекты, из которых составлено множество, называются его элементами. Элементы множества различны и отличимы друг от друга. Как множествам, так и элементам можно давать имена или присваивать символьные обозначения.

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.

Множество, не содержащее элементов, называется пустым множеством и обозначается $\varnothing$.

1.2 Мультимножества

Пусть $X = \{x_1, ..., x_n\}$ — некоторое (конечное) множество и пусть $a_1, ..., a_n$ — неотрицательные целые числа. Тогда мультимножеством $\widehat{X}$ (над множеством X) называется совокупность элементов множества X, в которую элемент x_i входит a_i раз, $a_i \geqslant 0$.

1.3 Операции над мультимножествами

1.3.1 Дополнение мультимножества

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.

$\overline{A} = U - A = \{k\overline{A}x | kUx - kAx, \forall x \in U\}$

1.3.2 Объединение мультимножеств

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:

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

1.3.3 Пересечение мультимножеств

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:

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

1.3.4 Разность мультимножеств

Разностью мультимножеств называется мультимножество, состоящее из элементов, одновременно присутствующих в A и в $\overline{B}$, а кратность каждого элемента равна минимальной кратности соответствующих элементов в мультимножествах A и $\overline{B}$:
$C = A \backslash \overline{B} = \{min(k_i x_i, k_j x_j)\}, k_i x_i \in A, k_j x_j \in \overline{B}$

1.3.5 Симметрическая разность мультимножеств

Симметрической разностью мультимножеств A и B называется мультимножество, состоящее из тех элементов мультимножества A и B, кратности которых различны. Кратность каждого элемента результирующего множества равна модулю разности кратностей соответствующих элементов в вычитаемых мультимножествах:
$C = A \triangle B = (A \cup B) \backslash (A \cap B) = \{kx | kx = |k_i x_i - k_j x_j|\}, k_i x_i \in A, k_j x_j \in B$

1.3.6 Арифметическая разность мультимножеств

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:
$C = A - B = \{kx | kx = k_i x_i - k_j x_j, \text{ если } kx > 0, \text{ иначе } kx = 0\}, k_i x_i \in A, k_j x_j \in B$

1.3.7 Арифметическая сумма мультимножеств

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:
$C = A + B = \{kx | kx = k_i x_i + k_j x_j\}, k_i x_i \in A, k_j x_j \in B$

1.3.8 Арифметическое произведение мультимножеств

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:
$C = A * B = \{kx | kx = k_i x_i * k_j x_j\}, k_i x_i \in A, k_j x_j \in B$

1.4 Код Грея

Двоичным кодом Грея порядка n называется последовательность всех 2^n n-битных кодов, в которой любые два соседних кода различаются ровно в одном разряде.

Реализация приведена в Листинге 1.

Листинг 1. Псевдокод генерации кодов Грея

```
1  Вход: n >= 0
```

```
2   Выход: B (array [1..n] of 0,1)
3
4   for i from 1 to n do:
5     B[i]:=0
6   end for
7   yield B
8
9   for i from 1 to 2^n-1 do:
10    p:= Q(i)
11    B[p]:=1-B[p]
12    yield B
13  end for
14
15  Q:
16  q:=1, j:=i
17  while (j mod 2 = 0) do:
18    j:=j/2
19    q:=q+1
20  end while
21  return q
```

2 Особенности реализации

2.1 Класс universum

Класс предназначен для хранения данных о универсуме, на основе которого будут создаваться мультимножества.

2.1.1 Поля класса

- int bitDepth - разрядность кодов Грея
- int size - количество кодов Грея
- std::vector<unsigned short> cardinality - кратность элементов универсума
- std::vector<std::vector<bool> > uni - коды Грея

2.1.2 Конструктор класса

При создании объекта типа universum вызывается конструктор, делающий следующее:

- считывание разрядности кодов Грея и запись в переменную bitDepth
- вычисление количества кодов ($2^{bitDepth}$) и запись в перменную size
- заполнение массива uni кодами Грея через метод gray_code_generator()
- заполнение массива cardinality рандомными числами от 1 до 10

Реализация приведена в Листинге 2.

Листинг 2. Конструктор класса universum

```
1  universum::universum() {
2    newUni();
3  }
4
5  void universum::newUni() {
6    uni.resize(0);
7    cardinality.resize(0);
8    this->bitDepth = intInput("Введите разрядность кодов Грея
       [0; 20]: ", 20, 0);
9    size = pow(2, bitDepth);
10
11   gray_code_generator();
12
13   for (int i = 0; i < size; i++) {
14     cardinality.push_back(rand() % 9 + 1);
15   }
16 }
```

2.1.3 Метод gray_code_generator

Вход: объект типа universum

Выход: заполненный массив std::vector<std::vector<bool> > uni

Данный метод заполняет кодами Грея соответствующий массив в объекте типа universum. Создается вектор размера bitDepth, заполняется нулями и добавляется в массив uni. Далее добавляются все остальные элементы, начиная со второго. Новые эдементы генерируются по принципу инверсии одного бита предыдущего кода. Для получения номера нужного бита используется метод Q, описанный далее.

Реализация генерации кодов приведена в Листинге 3.

Листинг 3. метод gray_code_generator()

```
void universum::gray_code_generator() {
  vector<bool> B(bitDepth, 0);
  uni.push_back(B);

  for (int i = 1; i < size; i++) {
    int ind = Q(i);
    ind = bitDepth - ind;
    B[ind] = 1 - B[ind];
    uni.push_back(B);
  }
}
```

2.1.4 Метод Q

Вход: int i - номер кода для генерации

Выход: int q - номер бита для инверсии

Вспомогательный метод для метода gray_code_generator. Используется для подсчета номера бита для инверсии в коде Грея.

Реализация приведена в Листинге 4.

Листинг 4. метод Q

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

2.2 Класс multiset

Класс предназначен для хранения данных о мультимножестве.

2.2.1 Поля класса

- universum* pUniversum - указатель на универсум - основу множества
- bool bitDep - используется для определения пустоты множества
- std::vector<short> cardinality - кратность элементов множества

2.2.2 Конструктор класса

При создании объекта типа multiset вызывается конструктор, делающий следующее:

- проверка, не является ли универсум пустым множеством
- сохранение указателя на универсум
- заполнение множества автоматически, либо вручную

Заполнение множества автоматически:
Сначала пользователь выбирает, сколько будет элементов ненулевой кратности во множестве, а потом они заполняются случайными ненулевыми кратностями, не большими соответствующей кратности в универсуме.
Заполнение множества вручную:
Пользователь поэлементно вводит кратности, не большие соответствующей кратности в универсуме.
Реализация приведена в Листинге 5.

Листинг 5. Конструктор класса multiset

```
1  void multiset::newMultiset(universum* pUniversum) {
2    if (pUniversum->getBitDepth() == 0) {
3      bitDep = 0;
4      return;
5    }
6
7    bitDep = 1;
8    this->pUniversum = pUniversum;
9    bool flag;
10   flag = intInput("\n\nВыберите, как заполнить множество: [0] -
       Вручную, [1] - Автоматически\n", 1);
11   cardinality.resize(pUniversum->getCardinality().size());
12
13   if (flag) {
14     cout << "\nКакая будет мощность множества? (Максимум " <<
         pUniversum->getSize() << ")\n";
15     int card;
16     card = intInput("", pUniversum->getSize());
```

```
17      for (int i = 0; i < card; i++) {
18        int ind;
19        do {
20          ind = rand() % pUniversum->getSize();
21        } while (cardinality[ind] != 0);
22        cardinality[ind] = (rand() % (pUniversum->getCardinality()
              [ind])) + 1;
23      }
24    }
25    else {
26      int multiplicity;
27      for (int i = 0; i < pUniversum->getSize(); i++) {
28        cout << "Введите кратность элемента " << i << " - ";
29        for (bool j : pUniversum->getUni()[i])
30        std::cout << j;
31        cout << ", она не должна превышать " << pUniversum->
              getCardinality()[i] << "\n";
32        cardinality[i] = intInput("", pUniversum->getCardinality()
              [i]);
33      }
34    }
35  }
```

2.2.3 Метод complement

Вход: объект класса multiset

Выход: объект класса multiset, противоположный исходному

Метод реализует операцию дополнения множества путем поэлементного вычитания кратностей множества из универсума.

Результат вычитания записывается в соответствующий элемент кратности другого множества.

В случае, когда универсум является пустым множеством, этот и следующий методы операций над мультимножествами будут выводить соответсвующее сообщение "<blank>".

Реализация приведена в Листинге 6.

Листинг 6. Метод complement

```
1  void multiset::complement(multiset A) {
2    if (bitDep == 0) {
3      return;
4    }
5
6    for (int i = 0; i < pUniversum->getSize(); i++) {
7      cardinality[i] = pUniversum->getCardinality()[i] - A.
           cardinality[i];
8    }
9  }
```

2.2.4 Метод unionS

Вход: 2 объекта класса multiset

Выход: объект класса multiset - результат объединения множеств

Метод реализует операцию объединения множеств: для каждого элемента входящих множеств ищется с элемент с максимальной кратностью и записывается в соответствующую кратность выходного элемента.

Реализация приведена в Листинге 7.

Листинг 7. Метод unionS

```
void multiset::unionS(multiset A, multiset B) {
  if (bitDep == 0) {
    return;
  }

  for (int i = 0; i < pUniversum->getSize(); i++) {
    cardinality[i] = max(A.cardinality[i], B.cardinality[i]);
  }
}
```

2.2.5 Метод intersection

Вход: 2 объекта класса multiset

Выход: объект класса multiset - результат пересечения множеств

Метод реализует операцию пересечения множеств: для каждого элемента входящих множеств ищется элемент с минимальной кратностью и записывается в соответствующую кратность выходного элемента.

Реализация приведена в Листинге 8.

Листинг 8. Метод intersection

```
void multiset::intersection(multiset A, multiset B) {
  if (bitDep == 0) {
    return;
  }

  for (int i = 0; i < pUniversum->getSize(); i++) {
    cardinality[i] = min(A.cardinality[i], B.cardinality[i]);
  }
}
```

2.2.6 Метод difference

Вход: 2 объекта класса multiset

Выход: объект класса multiset - результат разности множеств

Метод реализует операцию разности множеств: в соответствующую кратность выходного элемента записывается минимум между кратностями соответствующих элементов первого входного и дополненного второго множеств.

Реализация приведена в Листинге 9.

Листинг 9. Метод difference

```
void multiset::difference(multiset A, multiset B) {
  if (bitDep == 0) {
    return;
  }

  multiset notB(pUniversum, 1);
  notB.addition(B);
  intersection(A, notB);
}
```

2.2.7 Метод simDifference

Вход: 2 объекта класса multiset

Выход: объект класса multiset - результат симметрической разности множеств

Метод реализует операцию симметрической разности множеств: в соответствующую кратность выходного элемента записывается модуль разности кратностей соответствующих элементов первого и второго входных множеств.

Реализация приведена в Листинге 10.

Листинг 10. Метод simDifference

```
void multiset::simDifference(multiset A, multiset B) {
  if (bitDep == 0) {
    return;
  }

  for (int i = 0; i < pUniversum->getSize(); i++) {
    cardinality[i] = abs(A.cardinality[i] - B.cardinality[i]);
  }
}
```

2.2.8 Метод arifSum

Вход: 2 объекта класса multiset

Выход: объект класса multiset - результат алгебраической суммы множеств

Метод реализует операцию алгебраической суммы множеств: в соответствующую кратность выходного элемента записывается сумма кратностей соответствующих элементов первого и второго входных множеств. Если она больше кратности универсума, то вместо неё записывается последняя.

Реализация приведена в Листинге 11.

Листинг 11. Метод arifSum

```
  void multiset::arifSum(multiset A, multiset B) {
    if (bitDep == 0) {
      return;
    }

```

```
6       for (int i = 0; i < pUniversum->getSize(); i++) {
7         cardinality[i] = A.cardinality[i] + B.cardinality[i];
8         if (cardinality[i] > pUniversum->getCardinality()[i])
9         cardinality[i] = pUniversum->getCardinality()[i];
10      }
11    }
```

2.2.9 Метод arifDifference

Вход: 2 объекта класса multiset

Выход: объект класса multiset - результат алгебраической разности множеств

Метод реализует операцию алгебраической разности множеств: в соответствующую кратность выходного элемента записывается разность кратностей соответствующих элементов первого и второго входных множеств. Если она меньше 0, то вместо неё записывается 0.

Реализация приведена в Листинге 12.

Листинг 12. Метод arifDifference

```
1  void multiset::arifDifference(multiset A, multiset B) {
2    if (bitDep == 0) {
3      return;
4    }
5
6    for (int i = 0; i < pUniversum->getSize(); i++) {
7      cardinality[i] = A.cardinality[i] - B.cardinality[i];
8      if (cardinality[i] < 0) cardinality[i] = 0;
9    }
10 }
```

2.2.10 Метод arifMultiplication

Вход: 2 объекта класса multiset

Выход: объект класса multiset - результат алгебраического произведения множеств

Метод реализует операцию алгебраической разности множеств: в соответствующую кратность выходного элемента записывается произведение кратностей соответствующих элементов первого и второго входных множеств. Если оно больше кратности в универсуме, то вместо него записывается кратность соответствующего элемента универсума.

Реализация приведена в Листинге 13.

Листинг 13. Метод arifMultiplication

```
1  void multiset::arifMultiplication(multiset A, multiset B) {
2    if (bitDep == 0) {
3      return;
4    }
5
```

```
6     for (int i = 0; i < pUniversum->getSize(); i++) {
7       cardinality[i] = A.cardinality[i] * B.cardinality[i];
8       if (cardinality[i] > pUniversum->getCardinality()[i])
9       cardinality[i] = pUniversum->getCardinality()[i];
10    }
11  }
```

2.3 Функции вне классов

2.3.1 Функция intInput

Вход: сообщение для вывода, максимальное возможное число, минимальное возможное число

Выход: введенное значение int

Вспомогательная функция проверки корректности ввода пользователя. В случае, если кроме числа пользователем будет введено что-либо ещё, то она попросит его повторить ввод. Реализована через методы cin.peek, cin.clear, и cin.ignore.

Реализация приведена в Листинге 14.

Листинг 14. Функция intInput

```
int intInput(const char* msg, int maxi, int mini) {
  cout << msg;
  int val;
  while (true) {
    cin >> val;
    if (cin.peek() != '\n' or val > maxi or val < mini) {
      cin.clear();
      cin.ignore(cin.rdbuf()->in_avail());
      printf("\nНекоректное значение, повторите ввод\n");
    } else break;
  }
  return val;
}
```

2.3.2 Функция help

Вход:-

Выход: вывод сообщения с возможными командами

Вспомогательная функция вывода возможных для ввода комманд.

Реализация приведена в Листинге 15.

Листинг 15. Функция help

```
void help() {
  cout << "\n\n[0] - выход" << '\n';
  cout << "[1] - help" << '\n';
  cout << "[2] - вывести универсум" << '\n';
  cout << "[3] - вывести A" << '\n';
  cout << "[4] - вывести B" << '\n';
  cout << "[5] - дополнение A" << '\n';
  cout << "[6] - дополнение B" << '\n';
  cout << "[7] - объединение A и B" << '\n';
  cout << "[8] - пересечение A и B" << '\n';
  cout << "[9] - A\\B" << '\n';
  cout << "[10] - B\\A" << '\n';
  cout << "[11] - симметрическая разность A и B" << '\n';
```

```
14    cout << "[12] - симметрическая разность B и A" << '\n';
15    cout << "[13] - A-B" << '\n';
16    cout << "[14] - B-A" << '\n';
17    cout << "[15] - A*B" << '\n';
18    cout << "[16] - задать новый универсум" << '\n';
19 }
```

3 Результаты работы программы

После запуска программы пользователю предлагается задать разрядность кодов Грея и выбрать, как будет задаваться каждое множество. В случае некорректного ввода будет выведено "Некоректное значение, повторите ввод"(Рис.1).

```
Введите разрядность кодов Грея [0; 20]: 2

Universum:
0) code:        00      cardinality: 9
1) code:        01      cardinality: 6
2) code:        11      cardinality: 4
3) code:        10      cardinality: 4


Выберите, как заполнить множество: [0] - Вручную, [1] - Автоматически
0
Введите кратность элемента %0 - 00, она не должна превышать 9
10

Некоректное значение, повторите ввод
9
Введите кратность элемента %1 - 01, она не должна превышать 6
4
Введите кратность элемента %2 - 11, она не должна превышать 4
2
Введите кратность элемента %3 - 10, она не должна превышать 4
3


Выберите, как заполнить множество: [0] - Вручную, [1] - Автоматически
1

Какая будет мощность множества? (Максимум 4)
3

A
multiset:
0) code:        00      cardinality: 9
1) code:        01      cardinality: 4
2) code:        11      cardinality: 2
3) code:        10      cardinality: 3

B
multiset:
0) code:        00      cardinality: 0
1) code:        01      cardinality: 4
2) code:        11      cardinality: 4
3) code:        10      cardinality: 3
```

Рис. 1. Ввод универсума и множеств

После задания множеств выводится меню с возможными командами (Рис.2).

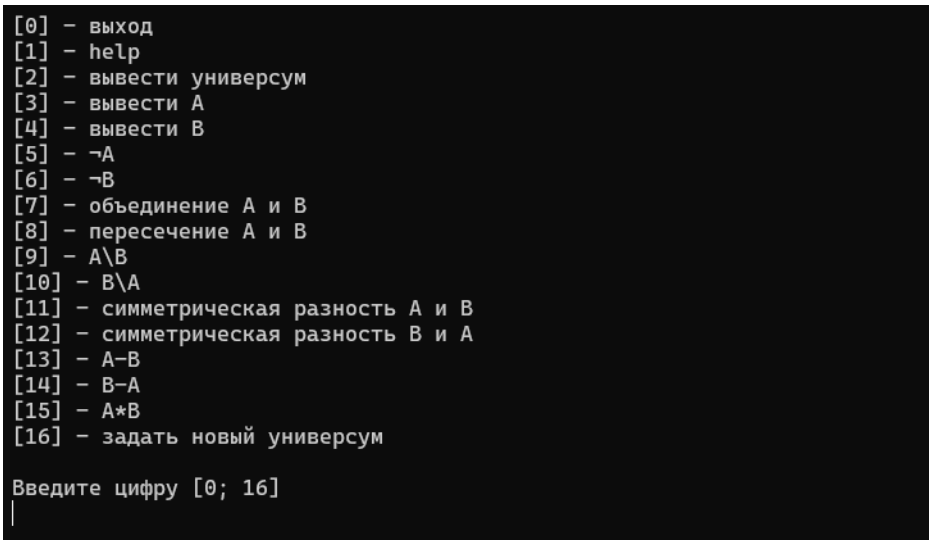

Рис. 2. Меню команд

При вводе соответствующих значений будут выводиться результаты выполнения операций над множествами (Рис.3, Рис.4, Рис.5).

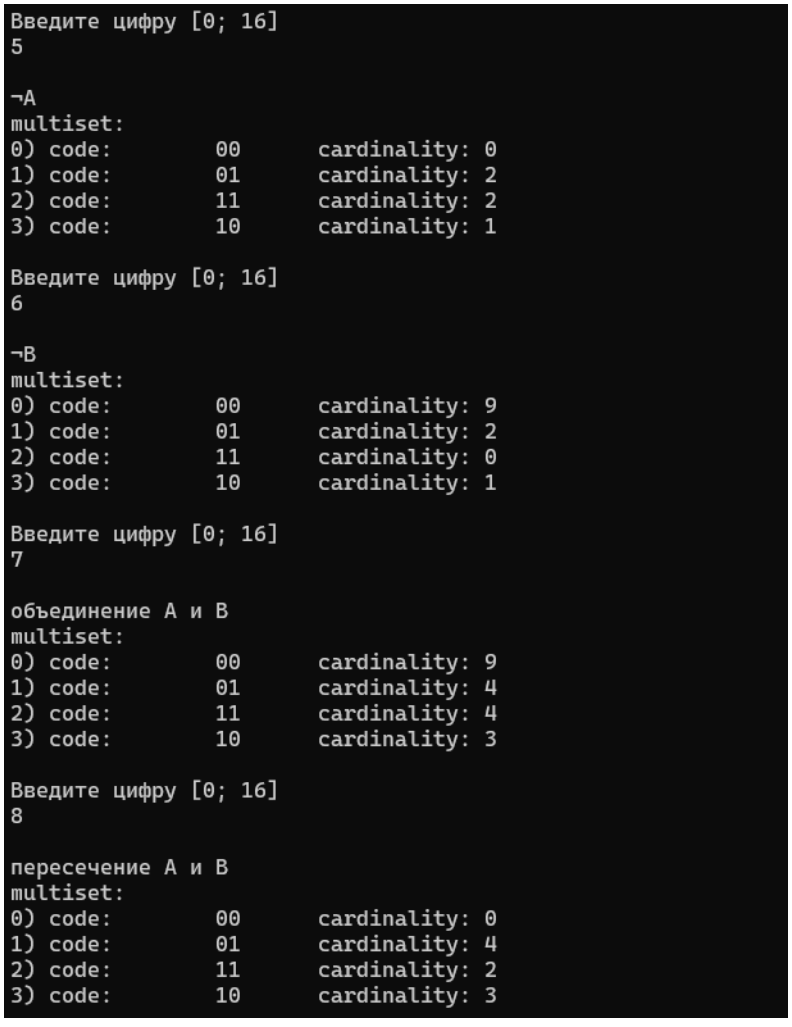

Рис. 3. Операции над множествами

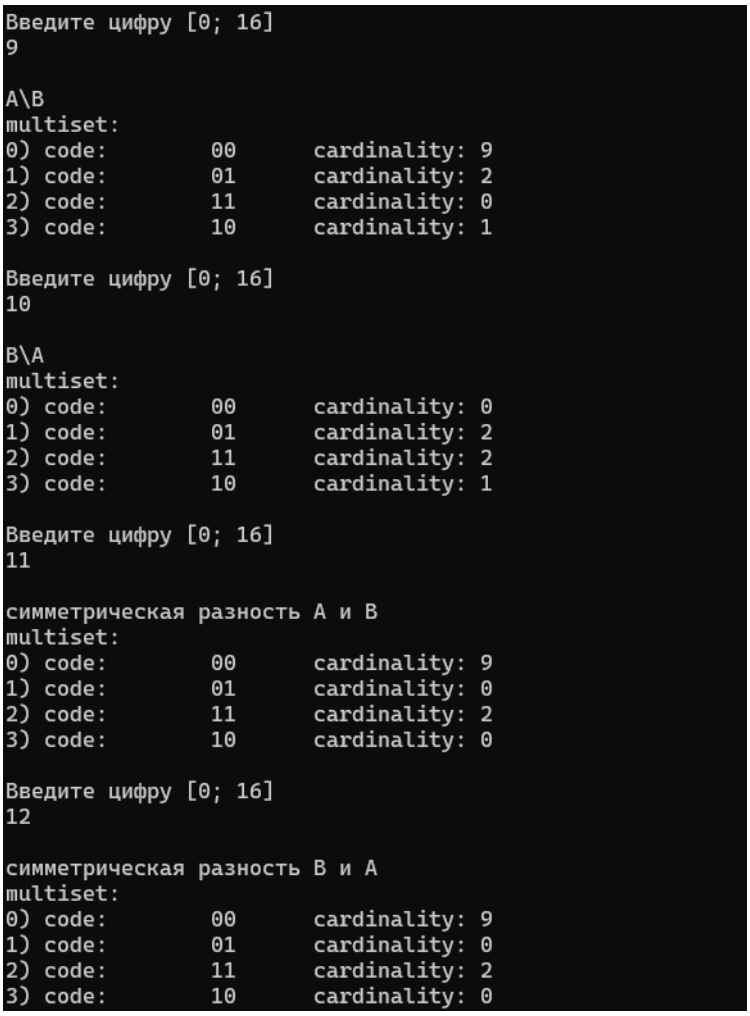

Рис. 4. Операции над множествами

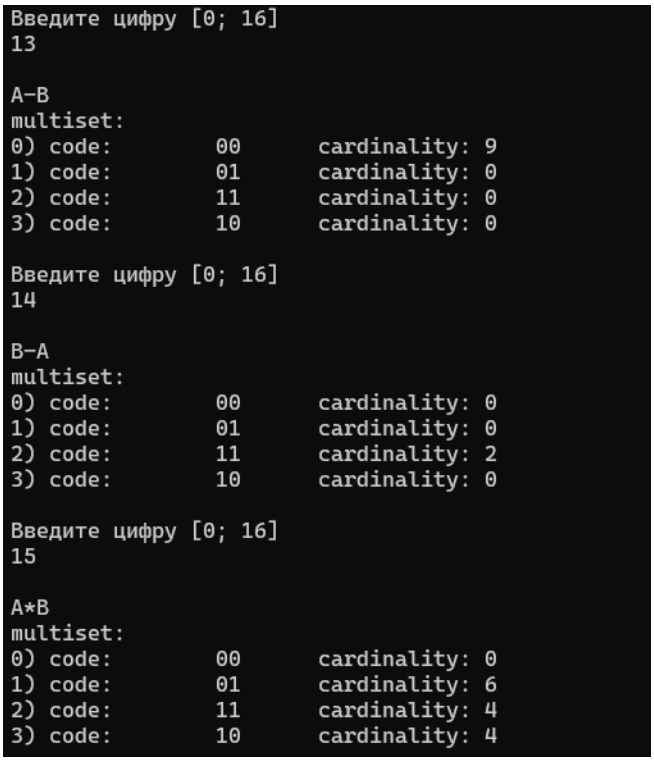

Рис. 5. Операции над множествами

При вводе числа 16 пользователю будет предложено пересоздать универсум и мультимножества (Рис.6).

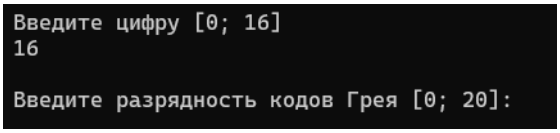

Рис. 6. Пересоздание универсума

Если изначально задан пустой универсум, то вместо результатов операций будет выводиться сообщение о пустом множестве (Рис.7, Рис.8).

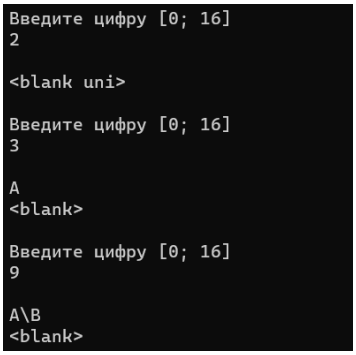

Рис. 7. Пустой универсум

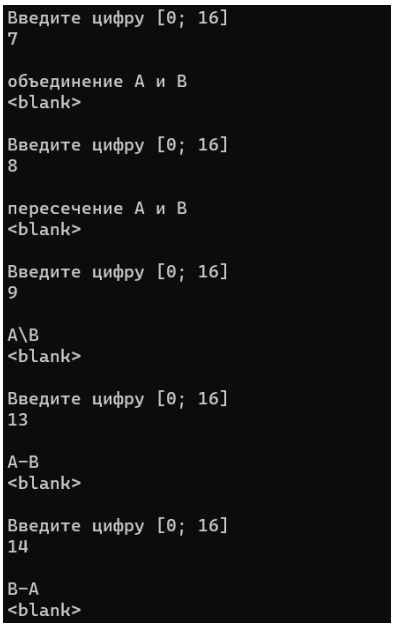

Рис. 8. Операции с пустыми множествами

При вводе 0 программа завершит свою работу (Рис.9).

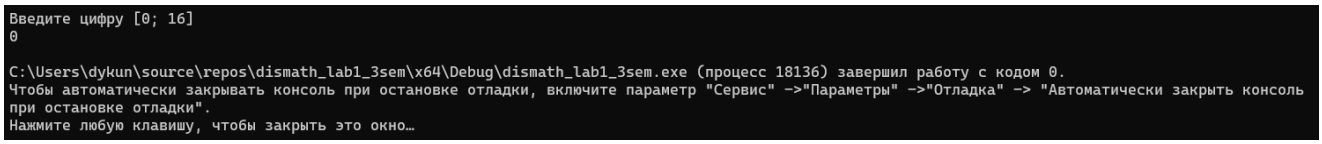

Рис. 9. Завершение работы программы

Заключение

В ходе выполнения лабораторной работы была реализована программа генерации бинарного кода Грея через специальный класс universum. На основе данного класса были создаются мультимножества, которые задаются пользователем автоматически либо вручную. Реализованы операции над множествами, позволяющие получать новые множества. Реализованные операции: дополнение, объединение, пересечение, разность, симметрическая разность, арифметическая разность, арифметическое умножение.

Достоинства реализации:

- Реализация через ООП
- Реализация операции разности через ранее реализованные операции
- Возможность работать с разными универсумами без перезапуска программы

Недостатки реализации:

- Реализация операций над множествами для пустого универсума дублируется, можно было сократить количество кода
- Нельзя работать с универсумом больше 20

Масштабируемость:

- Реализация через ООП позволяет создавать новые множества - результаты операций над старыми и работать с ними
- При задании универсума можно ограничить кратности элементов 1, тогда получатся операции над обычными множествами

Список литературы

[1] https://mydebianblog.blogspot.com/2012/12/latex.html?m=1, (дата обращения - 01.10.2023)

[2] https://studfile.net/preview/1399244/page:18/, (дата обращения - 01.10.2023)

[3] https://habr.com/ru/articles/200806/, (дата обращения - 01.10.2023)

[4] Новиков, Ф.А. ДИСКРЕТНАЯ МАТЕМАТИКА ДЛЯ ПРОГРАММИСТОВ / 3-е издание. – Питер : Питер Пресс, 2009, (дата обращения - 01.10.2023)