САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ им. ПЕТРА ВЕЛИКОГО
Институт компьютерных наук и технологий
Высшая школа искусственного интеллекта

Отчет по лабораторной работе:

Реализация алгоритма генерации бинарного кода Грея, мультимножеств и операций над ними на языке C++.

Работу выполнил:
Тимофеев А.С.
студент группы 3530201/10001

Проверил:
Востров А.В.

Санкт-Петербург - 2022 г.

Оглавление

Глава 1

Алгоритм генерации бинарного кода Грея и операции над мультимножествами.

1.1. Введение.

В лабараторной работе требуется реализовать генерацию бинарного кода Грея для создания универсума **U** с мощностью **N** .
Также требуется реализовать создания двух мультимножеств (Мощность предоставляется выбрать пользователю) и заполнить их. Для заполнения мультимножеств, предоставляется выбор режима: *автоматический* и *ручной* ввод.
Требуется реализовать защиту некорректного ввода и ввести операции над мультимножествами.

1.2. Алгоритм генерации бинарного кода Грея.

Коды Грея - бинарные коды, где соседи отличаются друг от друга на 1 бит.
ВХОД: n $\geq$ 0
ВЫХОД: U (Множество являющиеся универсумом с мощностью 2^n)
Алгоритмическая сложность: O(2^n)

Алгоритм генерации состоит из двух функций:

- Основная функция *create_universum* описанная выше, возвращающая универсум.
- Дополнительная функция *Q* осуществляющая инверсию бита в следующем коде Грея.

Функция **Q** - определения индекса изменяемого бита:
ВХОД: i - номер подможества
ВЫХОД: p (Номер изменяемого разряда)
Алгоритмическая сложность: O($\log n$)

Листинг 1.1: Реализация Алгоритма для моей программы

```
void create_universum(int n,MyMultiSet& U){
  string tmp = "";
  srand(unsigned int(time(0)));
  if (n == 0) {
    U.MultiSet.push_back(make_pair("empty",0));
    return;
  }
  for (int i = 0; i < n; i++) {
    tmp += '0';
  }
  U.MultiSet.push_back(make_pair(tmp, rand() % 9 + 1));
  for (int i = 1; i <= pow(2, n) - 1; i++) {
    int p = q(i) - 1;
    tmp[p] = (1 - (tmp[p] - '0')) + '0';
    U.MultiSet.push_back(make_pair(tmp, rand() % 9 + 1));
  }
  std::sort(U.MultiSet.begin(), U.MultiSet.end());
}
```

Листинг 1.2: Функция Q для подсчета индекса бита для инверсии

```
int q(int i ) {
  int q = 1;
  int j = i;

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

1.3. Реализация Мультимножества.

Для реализации *мультимножества* я решил написать свой класс с единственным полем *MultiSet*. Для хранения данных я выбрал **vector<pair<string,int>>**, потому что удобно добавлять элементы и для данной структуры в языке C++ уже есть функция сортировки, которую я непосредственно использую в программе.

1.4. Основные режимы работы.

1. Задание универсума и его генерация с помощью вышеописанных алгоритмов с мощностью 2^n.
2. Задание двух мультимножеств (*ручной* или *автоматический* ввод).
3. Операция объединение.
4. Операция пересечение.
5. Операция дополнения.
6. Операция разности.
7. Операция симметрической разности.
8. Операция арифметической суммы.
9. Операция арифметической разности.
10. Операция прямого произведения множеств.
11. Вывод всех множеств заведенных пользователем и результат последнего действия над ними.

1.5. Особенности реализации операций над множествами.

Операция объединение:
ВХОД: A,B (Мультимножества)
ВЫХОД: C (Мультимножество которое эквивалетно: A $\vee$ B)
Алгоритмическая сложность: O($|\mathbf{A}| * |\mathbf{B}|$)

Операция пересечение:
ВХОД: A,B (Мультимножества)
ВЫХОД: C (Мультимножество которое эквивалетно: A $\wedge$ B)
Алгоритмическая сложность: O($|\mathbf{A}| * |\mathbf{B}|$)

Операция дополнение:
ВХОД: A, U (Мультимножество и универсум)
ВЫХОД: C (Мультимножество которое эквивалетно: (A))
Алгоритмическая сложность: O($|\mathbf{A}| * |\mathbf{U}|$)

Для операций: **Разность**, **симметрическая разность** используются алгоритмы приведенные выше. Основываясь на определении операций

Арифметические операции реализованы по определению.

Используется вспомогательная функция **isMstEmpt()**, которая определяет пустое ли множество:
ВХОД: A (Мультимножество)
ВЫХОД: Логическое значение
Алгоритмическая сложность: O(1)

Листинг 1.3: Функция объединения множеств

```
1  MyMultiSet unionOfMst(MyMultiSet A, MyMultiSet B)
2  {
3
4    if (isMstEmpt(A)) {
5      return B;
6    }
7    if (isMstEmpt(B)) {
8      return A;
9    }
10   MyMultiSet C;
11   int i = 0;
12   for (int i = 0; i < A.MultiSet.size(); i++) {
13     bool myflag = false;
14     for (int j = 0; j < B.MultiSet.size(); j++) {
15       if (A.MultiSet[i].first == B.MultiSet[j].first) {
16         myflag = true;
17
18         C.MultiSet.push_back(make_pair(A.MultiSet[i].first, max(A.MultiSet[i
       ].second, B.MultiSet[j].second)));
19         B.MultiSet[j].first = "empty";
20         break;
21       }
22     }
```

```
23      if (!myflag) {
24        C.MultiSet.push_back(A.MultiSet[i]);
25      }
26    }
27    while (i < B.MultiSet.size()) {
28      if (B.MultiSet[i].first != "empty") {
29        C.MultiSet.push_back(B.MultiSet[i]);
30      }
31      i++;
32    }
33    return C;
34  }
35  }
36  }
```

Листинг 1.4: Функция пересечения множеств

```
MyMultiSet intersectionOfMst(MyMultiSet& A, MyMultiSet& B)
{
  if (isMstEmpt(A)) {
    return A;
  }
  if (isMstEmpt(B)) {
    return B;
  }
  MyMultiSet C;
  for (int i = 0; i < A.MultiSet.size(); i++) {
    for (int j = 0; j < B.MultiSet.size(); j++) {
      if (A.MultiSet[i].first == B.MultiSet[j].first) {
        C.MultiSet.push_back(make_pair(A.MultiSet[i].first, min(A.MultiSet[i
 ].second, B.MultiSet[j].second)));
      }
    }
  }
  if (C.MultiSet.empty())C.MultiSet.push_back(make_pair("empty", 1));
  return C;
}
```

Листинг 1.5: Функция дополнения множеств

```
MyMultiSet complementOfMst(MyMultiSet& A,MyMultiSet &U)
{
  if (isMstEmpt(A)) {
    return U;
  }
  if (isMstEmpt(U)) {
    return U; //???
  }
  MyMultiSet C;
  for (int i = 0; i < U.MultiSet.size(); i++) {
    bool flag = false;
    for (int j = 0; j < A.MultiSet.size(); j++) {
      if (U.MultiSet[i].first == A.MultiSet[j].first) {
        C.MultiSet.push_back(make_pair(U.MultiSet[i].first, U.MultiSet[i].
 second - A.MultiSet[j].second));
        flag = true;
        break;
      }
    }
    if (!flag) {
      C.MultiSet.push_back(U.MultiSet[i]);
    }
  }
  return C;
}
}
```

1.6. Результат работы программы:

Результатом работы программы является отображение в консоли результата операции выбранной пользователем.

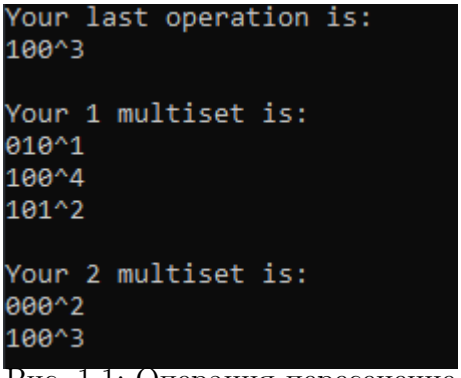

Рис. 1.1: Операция пересечение

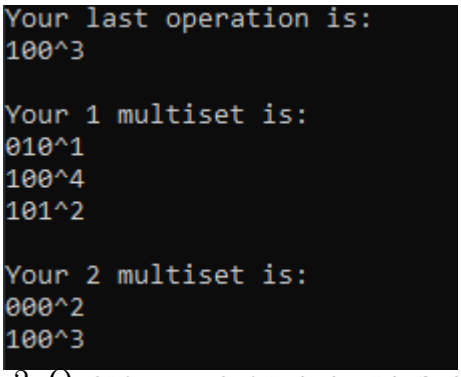

Рис. 1.2: Операция декартово произведение

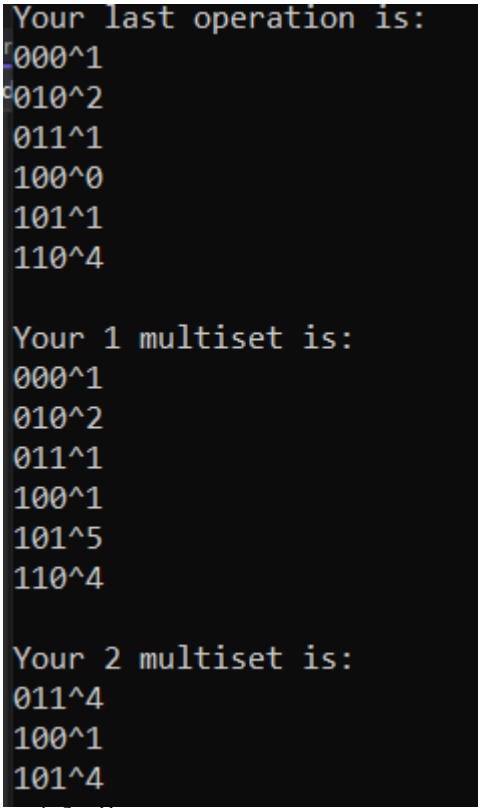

Рис. 1.3: Операция разность множеств

Глава 2
Заключение

2.1. Реализованный функционал

В моей программе реализован следующий функционал:

1. Задание универсума и его генерация с помощью вышеописанных алгоритмов с мощностью 2^n.
2. Задание двух мультимножеств (*ручной* или *автоматический* ввод).
3. Выполнение всех операций над множествами.
4. Защита от пользовательского некорректного ввода

2.2. Масштабирование

Благодаря использованию именно такой структуры данных как вектор класса MultiSet, описанного ранее, мы с легкостью можем ввести сколь угодно много новых множеств. Достаточно программисту изменить параметр при входе в программу.

Также при должном оптимизировании можно свести опепрации требующией $O(n^2)$ алгоритмической сложности $O(n * \log n)$, так как множества подаются в отсортированном виде и можно применять бинарный поиск.
В моей программе используется не самая быстрая реализации алгоритмов, так как числа не столь велики. При более больших n, разумнее использовать вариант с бинарным поиском.

Литература

[1] В.В Голенков, Н.А Гулякина, БГУИР 2010 (Мет пособие).«ЭУМК по Дискретной математике» URL: https://studfile.net/preview/1399243/page:15/; дата обращения: 15.06.2014.

[2] Ф.А Новиков. «Дискретная математика для программистов» (3-е издание)