РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ
ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В. Г. ШУХОВА»
(БГТУ им. В.Г. Шухова)**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

Лабораторная работа №1.3

по дисциплине: Дискретная математика
тема: «Теоретико-множественные тождества»

Выполнил: ст. группы ПВ-211
Чувилко Илья Романович

Проверили:
Рязанов Юрий Дмитриевич

Белгород 2022 г.

Цель работы: изучить методы доказательства теоретикомножественных тождеств.

Задания:

1. На рис.1 изображены круги Эйлера, соответствующие множествам A, B и C, с пронумерованными элементарными областями (не содержащими внутри себя других областей). Заштриховать элементарные области в соответствии с вариантом задания (см. табл.2).

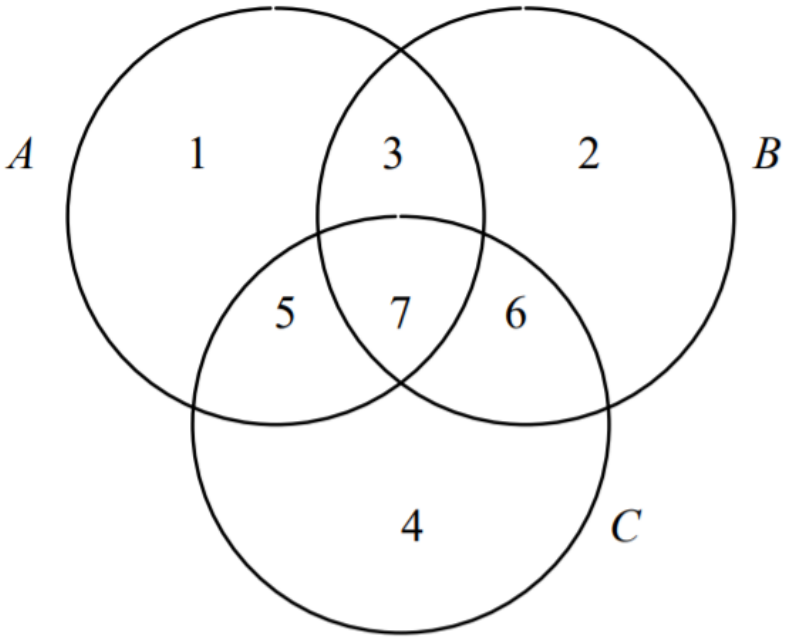

Рис.1. Круги Эйлера, соответствующие множествам *A*, *B* и *C* с пронумерованными элементарными областями

	A	B	C		
1	1	0	0	1	A-B-C
2	0	1	0	0	
3	1	1	0	1	A&B-C
4	0	0	1	1	C-A-B
5	1	0	1	0	
6	0	1	1	0	
7	1	1	1	0	
(A-B-C) U (A&B-C) U (C-A-B)					

$$(A - B - C) \cup (A \cap B - C) \cup (C - A - B) =$$
$$A \cap \neg B \cap \neg C \cup A \cap B \cap \neg C \cup C \cap \neg A \cap \neg B = A \cap \neg C \cup C \cap \neg A \cap \neg B$$

2. Написать выражение 1 над множествами A, B и C, определяющее заштрихованную область, используя операции пересечения, объединения и дополнения

$$A \cap \neg C \cup \neg A \cap \neg B \cap C$$

3. Используя свойства операций над множествами, преобразовать выражение 1 в выражение 2, не содержащее операции дополнения множества.

$$(A - C) \cup (C - A - B)$$

4. Используя свойства операций над множествами, преобразовать выражение 2 в выражение 3, не содержащее операции объединения множеств.

$$\neg(\neg(A - C) \cap \neg(C - A - B))$$

5. Используя свойства операций над множествами, преобразовать выражение 3 в выражение 4, не содержащее операции пересечения множеств

$$(A - C) \cup (C - A - B)$$

6. Доказать тождественность выражений 2 и 3 методом характеристических функций.

$$\chi_{(A-CUC-A-B)} = \chi_{A-C} + \chi_{C-A-B} - \chi_{A-C}\chi_{C-A-B} = \chi_A - \chi_A\chi_C + \chi_C - \chi_C\chi_{A-B} - (\chi_A - \chi_A\chi_C)(\chi_c - \chi_{A-B}) = \chi_A - \chi_A\chi_C + \chi_C - \chi_C(\chi_A - \chi_A\chi_B) - (\chi_A - \chi_A\chi_C)\chi_C(1 - \chi_{A-B}) = \chi_A - \chi_A\chi_C + \chi_C - \chi_A\chi_C + \chi_A\chi_B\chi_C - 0(1 - \chi_{A-B}) = \chi_A - \chi_C - \chi_A\chi_C + \chi_A\chi_B\chi_C$$

$$\chi_{\neg(\neg(A-C)\&\neg(C-A-B))} = 1 - \chi_{\neg(A-C)}\chi_{\neg(C-A-B)} = 1 - (1 - \chi_{A-C})(1 - \chi_{C-A-B}) = 1 - 1 - \chi_{A-C} + \chi_{C-A-B} - \chi_{A-C}\chi_{C-A-B} = \chi_A - \chi_A\chi_C + \chi_C - \chi_C\chi_{A-B} - (\chi_A - \chi_A\chi_C)(\chi_c - \chi_{A-B}) = \chi_A - \chi_A\chi_C + \chi_C - \chi_C(\chi_A - \chi_A\chi_B) - (\chi_A - \chi_A\chi_C)\chi_C(1 - \chi_{A-B}) = \chi_A - \chi_A\chi_C + \chi_C - \chi_A\chi_C + \chi_A\chi_B\chi_C - 0(1 - \chi_{A-B}) = \chi_A - \chi_C - \chi_A\chi_C + \chi_A\chi_B\chi_C$$

7. Доказать тождественность выражений 2 и 4 методом логических функций. Для автоматизации доказательства написать программу, которая получает и сравнивает таблицы истинности логических функций

```
int main() {
  cout << "abc\n";
  for (int a = 0; a < 2; a++) {
    for (int b = 0; b < 2; b++) {
      for (int c = 0; c < 2; c++) {
        int f1 = a && !c || c && !a && !b;
        int f2 = (a && !c) || (c & !a & !b);
        cout << a << b << c << '|' << f1 << f2 << '\n';
      }
    }
  }
}
```

8. Доказать тождественность выражений 3 и 4 теоретико-множественным методом. Для автоматизации доказательства написать программу, в которой вычисляются и сравниваются значения выражений 3 и 4 при A = {1,3,5,7}, B = {2,3,6,7} и C = {4,5,6,7}.

```
#include <iostream>
#include <stack>
#include <string>
#include <cassert>

#include "libs/UOAS/UOAS.h"

using namespace std;

// Структура, которая хранит оператор и его приоритет
struct priority {
  char operator_;
  int priority_;
};
```

```
class PolishEntry {
  private:
  string standardExpression_{};
  string polishExpression_{};
  stack<priority> operators_{};

  stack<unordered_array_set> res{};
  unordered_array_set A{};
  unordered_array_set B{};
  unordered_array_set C{};
  unordered_array_set U{};

  void setPolish() {
    for (char c : standardExpression_) {
      switch (c) {
        case '(':
          operators_.push((priority) {c, 4});
          break;
        case ')':
          while (operators_.top().operator_ != '(') {
            push(operators_.top().operator_);
          }
          operators_.pop();
          break;

        case '!':
          operators_.push((priority) {c, 1});
          break;
        case '&':
        case '^':
        case '-':
          pushLessPriority(2);
          operators_.push((priority) {c, 2});
          break;
        case 'u':
          pushLessPriority(3);
          operators_.push((priority) {c, 3});
          break;
        default:
          polishExpression_.push_back(c);
      }
    }
    while (!operators_.empty())
      push(operators_.top().operator_);
  }

  void push(char c) {
    polishExpression_.push_back(c);
    operators_.pop();
  }

  void pushLessPriority(int p) {
    while (!operators_.empty() && operators_.top().priority_ <= p)
      push(operators_.top().operator_);
  }

  void doOperator(unordered_array_set (*f)(unordered_array_set,
unordered_array_set)) {
    assert(res.size() > 1);
    unordered_array_set tmp1 = res.top();
    res.pop();
    unordered_array_set tmp2 = res.top();
    res.pop();
    res.push(f(tmp2, tmp1));
  }
```

```
void doComplement() {
    assert(!res.empty());
    unordered_array_set tmp1 = res.top();
    res.pop();
    res.push(UOAS_complement(tmp1, U));
  }

  void getUOAS() {
    for (char c : polishExpression_) {
      switch (c) {
        case '-':
          doOperator(UOAS_difference);
          break;
        case '&':
          doOperator(UOAS_intersection);
          break;
        case 'u':
          doOperator(UOAS_union);
          break;
        case '^':
          doOperator(UOAS_symmetricDifference);
          break;

        case '!':
          doComplement();
          break;
        case 'A':
          res.push(A);
          break;
        case 'B':
          res.push(B);
          break;
        case 'C':
          res.push(C);
          break;
        default:
          cout << "Unknown character: " << c;
          exit(1);
      }
    }
  }

  void setUOAS() {
    int a[] = {1, 3, 5, 7};
    int b[] = {2, 3, 6, 7};
    int c[] = {4, 5, 6, 7};
    int u[] = {1, 2, 3, 4, 5, 6, 7};

    A = UOAS_createFromArray(a, 4);
    B = UOAS_createFromArray(b, 4);
    C = UOAS_createFromArray(c, 4);
    U = UOAS_createFromArray(u, 7);
  }

  public:
  explicit PolishEntry(string &s) {
    setExpression(s);
    setUOAS();
    getUOAS();
  }
```

```
PolishEntry() {
    setUOAS();
    getUOAS();
  }

  void setExpression(string &s) {
    standardExpression_ = s;
    setPolish();
  }

  string getPolish() {
    return polishExpression_;
  }

  unordered_array_set getResult() {
    return res.top();
  }
};

int main() {
  string s1, s2;
  cin >> s1 >> s2;
  PolishEntry p1(s1);
  PolishEntry p2(s2);

  unordered_array_set a1 = p1.getResult();
  unordered_array_set a2 = p2.getResult();

  UOAS_print(a1);
  UOAS_print(a2);

  if (UOAS_isEqual(a1, a2))
    cout << "EQUAL";
  else
    cout << "NOT EQUAL";
}
```