МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Нижегородский государственный университет им. Н.И. Лобачевского»**

**Национальный исследовательский университет**

**Институт информационных технологий, математики и механики**

**Кафедра математического обеспечения и суперкомпьютерных технологий**

**ЛАБОРАТОРНАЯ РАБОТА**

**«Структура хранения множества»**

**Выполнил:** студент группы 381706-2

Чирков Роман Дмитриевич

\_\_\_\_\_\_\_\_\_\_\_ Подпись

**Руководитель:**

Ассистент кафедры МОСТ,

Шестакова Наталья Валерьевна

\_\_\_\_\_\_\_\_\_\_\_ Подпись

Нижний Новгород

2018

Оглавление

[Введение 2](#_Toc502266332)

[Постановка задачи 3](#_Toc502266333)

[Руководство пользователя 4](#_Toc502266335)

[Руководство программиста 6](#_Toc502266336)

[Заключение 16](#_Toc502266338)

[Литература 17](#_Toc502266339)

[Приложение 18](#_Toc502266340)

# Введение

*Теория множеств* — это раздел логики и математики, в рамках которого изучаются классы (множества) элементов произвольной природы. Множество при этом понимается как произвольная совокупность определённых и различимых объектов, мысленно объединённых в единое целое и называемых его элементами. Методы теории множеств широко используются во всех областях современной *математики* и *математической логики*; они имеют принципиальное значение для вопросов обоснования математики логическими средствами.

Активное применение аппарата теории множеств в современной науке приводит к необходимости создания соответствующих программных решений. Вместе с тем лишь в отдельных языках программирования предусмотрены встроенные средства для работы с множествами. Программная реализация множества может выполняться различными способами (в соответствии с требованиями конкретной задачи или с общих позиций) и обычно тесно связана с использованием битовых операций в выбранном языке программирования. Данная работа посвящена изучению одного из возможных подходов к хранению и обработке множеств.

# Постановка задачи

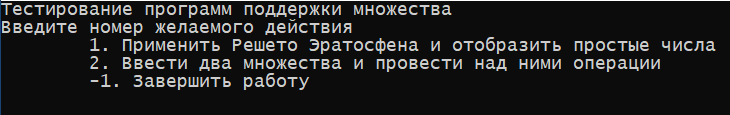
Я ставлю перед собой несколько главных учебно-практических задач:

1. Создать класс битового поля и реализовать в нём следующие функции:
   * установить бит (в единицу);
   * очистить бит (в ноль);
   * получить значение бита;
   * сравнить два битовых поля;
   * выполнить операцию “логическое или” для двух битовых полей;
   * выполнить операцию “логическое и” для двух битовых полей;
   * выполнить операцию “логическое отрицание” для битового поля.
2. Создать класс множества и реализовать в нём следующие функции:
   * включение элемента в множество;
   * исключение элемента из множества;
   * проверка наличия элемента в множестве;
   * сравнение множеств
   * сложение множеств;
   * пересечение множеств;
   * разность множеств;
   * копирование множества;
   * вычисление максимальной мощности множества.

# Руководство пользователя

Программа предназначена демонстрации работы класса множества.

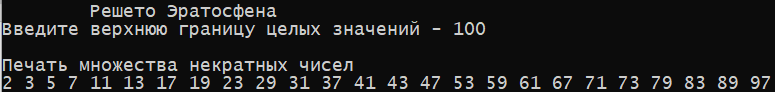
Управление осуществляется через меню. Пользователю следует ввести число 1, 2 или -1 для выбора соответствующего действия.



* Применить Решето Эратосфена и отобразить простые числа

Пункт используется демонстрации работы функции, отображающей простые числа до заданного верхнего значения.

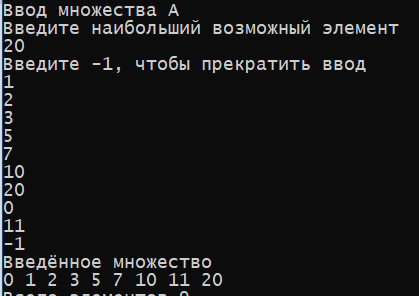
Пользователю предлагается ввести верхнюю границ целых значений. Затем после ввода, печатаются через пробел все простые числа не превосходящие заданную границу



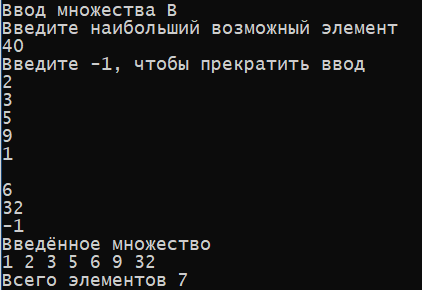
* Ввести два множества и провести над ними операции

Пункт используется для проведения операций над двумя введёнными множествами.

Пользователю предлагается ввести множество A.



Затем – множество B.

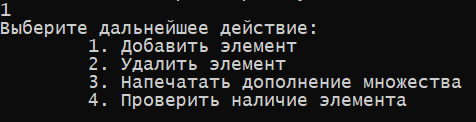


После ввода предлагается выбрать номер дальнейшего действия.

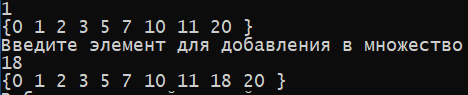
Далее показаны примеры выполнения операций.

1. Операции над A

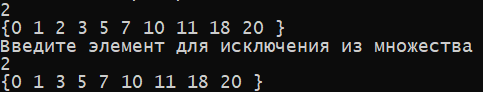
При выборе этого пункта появляется меню выбора операции.



Добавить элемент.



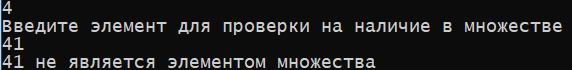
Удалить элемент.



Напечатать дополнение множества. За унивёрс принимается множество целых чисел от нуля до верхней границы множества.

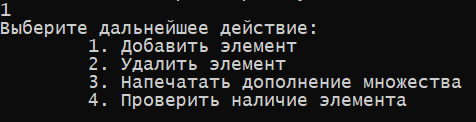


Проверить наличие элемента.

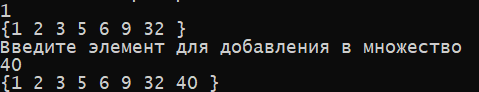


2. Операции над B.

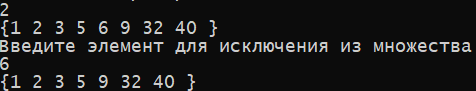
При выборе этого пункта появляется меню выбора операции.



Добавить элемент.

****

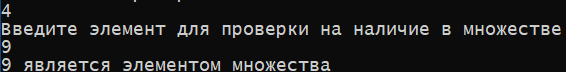
Удалить элемент.



Напечатать дополнение множества. За унивёрс принимается множество целых чисел от нуля до верхней границы множества.

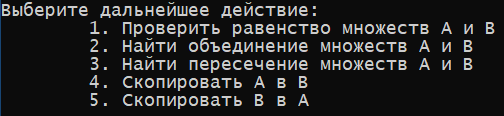


Проверить наличие элемента.



3. Операции над обоими множествами

При выборе этого пункта появляется меню выбора операции.



Проверить равенство множеств А и B.



Найти объединение множеств А и B.



Найти пересечение множеств А и B.



Скопировать A в B.



Скопировать B в A.



-1. Завершить работу c этими множествами.

# Руководство программиста

*Структура программного комплекса*

*Структура данных*

*Описание использованных алгоритмов*

Ниже представлены алгоритмы некоторых функций и процедур.

Битовое поле

* Получение индекса числа, в котором записан бит с номером n.

Описание

Функция получает в качестве параметра число n и возвращает результат побитового сдвига вправо на 5.

Алгоритм

Если n < 0 вернуть -1

index = n >> 5

Вернуть index

* Получение маски памяти.

Описание

Получает в качестве параметра число n и возвращает смещение единицы на n & 31 влево.

Алгоритм

Mask = 1 << (n & 31)

Вернуть Mask

* Получение бита

Описание

Получает в качестве параметра число n и возвращает смещение единицы на n & 31 влево.

Алгоритм

Mask = 1 << (n & 31)

* Вернуть Mask
* Установка бита

Описание

Получает маску для бита, записывает в элемент поля результат «битового или» для маски и числа, в котором хранится заданный бит.

Алгоритм

bit = ПолучитьБит n

Если bit == 1 или n < 0 или n > Битовой длины поля вернуться

index = Получить индекс памяти n

mask = Получить маску памяти n

pMem[index] = pMem[index] | mask

* Очистка бита

Описание

Получает маску для бита, записывает в элемент поля результат «битового и» для обратной маски и числа, в котором хранится заданный бит.

Алгоритм

bit = ПолучитьБит n

Если bit == 1 или n < 0 или n > Битовой длины поля вернуться

index = Получить индекс памяти n

mask = Получить маску памяти n

pMem[index] = pMem[index] & битовое дополнение mask

Множество

* Получение индекса числа, в котором записан бит с номером n.

Описание

Функция получает в качестве параметра число n и возвращает результат побитового сдвига вправо на 5.

Алгоритм

Если n < 0 вернуть -1

index = n >> 5

Вернуть index

# Заключение

В заключение, хочется отметить, что на данный момент существует много методов сортировки. Все они заслуживают внимания и исследования. В той или иной ситуации могут применяться не только разные алгоритмы, но и модификации одного и того же алгоритма сортировки.

Если память жёстко ограничена, можно использовать сортировки с квадратичной сложностью, как например сортировка выбором.

Если требуется выполнить сортировку максимально быстро любыми средствами, подойдет пирамидальная сортировка.

Или если код программы должен не слишком сильно усложняться, но при этом сортировка должна быть достаточной быстрой, лучше всего использовать сортировку Шелла и ей подобные.

В ходе работы были решены и достигнуты все поставленные задачи и цели.

# Литература

1. Аналитический портал «Гуманитарные технологии». Теория множеств – Режим доступа: https://gtmarket.ru/concepts/7073– Загл. с экрана.

# Приложение

Main.cpp

#include <locale.h>

#include <iostream>

#include "TSet.h"

using namespace std;

void menu();

void primeNumbers();

void operations();

void oneSet(int n, TSet &S);

int main()

{

setlocale(LC\_ALL, "Rus");

cout << "Тестирование программ поддержки множества" << endl;

menu();

}

void menu()

{

int input = 0;

while (input != -1) {

cout << "Введите номер желаемого действия" << endl

<< "\t1. Применить Решето Эратосфена и отобразить простые числа" << endl

<< "\t2. Ввести два множества и провести над ними операции" << endl

<< "\t-1. Завершить работу" << endl;

cin >> input;

switch (input) {

case 1:

primeNumbers();

break;

case 2:

operations();

break;

default: input = -1;

}

}

}

void primeNumbers()

{

int n, m, k;

cout << " Решето Эратосфена" << endl;

cout << "Введите верхнюю границу целых значений - ";

cin >> n;

TSet s(n + 1);

for (m = 2; m <= n; m++)

s.InsElem(m);

for (m = 2; m \* m <= n; m++)

if (s.IsMember(m))

for (k = 2 \* m; k <= n; k += m)

if (s.IsMember(k)) s.DelElem(k);

cout << endl << "Печать множества некратных чисел" << endl << s << endl;

}

void operations()

{

int n;

cout << "Ввод множества A" << endl;

cout << "Введите наибольший возможный элемент" << endl;

cin >> n;

TSet A(n + 1);

cin >> A;

cout << "Введённое множество" << endl;

cout << A << endl;

cout << "Всего элементов " << A.GetPower() << endl << endl;

cout << "Ввод множества B" << endl;

cout << "Введите наибольший возможный элемент" << endl;

cin >> n;

TSet B(n + 1);

cin >> B;

cout << "Введённое множество" << endl;

cout << B << endl;

cout << "Всего элементов " << B.GetPower() << endl << endl;

int input = 0;

while (input != -1) {

cout << "Выберите дальнейшее действие:" << endl

<< "\t1. Операции над A" << endl

<< "\t2. Операции над B" << endl

<< "\t3. Операции над обоими множествами" << endl

<< "\t-1. Завершить работу c этими множествами" << endl;

cin >> input;

if ((input == 1) || (input == 2)) {

cout << endl << "Выберите дальнейшее действие:" << endl

<< "\t1. Добавить элемент" << endl

<< "\t2. Удалить элемент" << endl

<< "\t3. Напечатать дополнение множества" << endl

<< "\t4. Проверить наличие элемента" << endl;

cin >> n;

if (input == 1)

oneSet(n, A);

else

oneSet(n, B);

}

if (input == 3) {

cout << endl << "Выберите дальнейшее действие:" << endl

<< "\t1. Проверить равенство множеств А и B" << endl

<< "\t2. Найти объединение множеств А и B" << endl

<< "\t3. Найти пересечение множеств А и B" << endl

<< "\t4. Скопировать A в B" << endl

<< "\t5. Скопировать B в A" << endl;

cin >> n;

switch (n) {

case 1:

if (A == B)

cout << "A == B" << endl;

else

cout << "A != B" << endl;

break;

case 2:

cout << "A + B = {" << A + B << "}" << endl;

break;

case 3:

cout << "A \* B = {" << A \* B << "}" << endl;

break;

case 4:

B = A;

cout << "Множество B после копирования множества A = {" << B << "}" << endl;

break;

case 5:

A = B;

cout << "Множество A после копирования множества B = {" << A << "}" << endl;

break;

default:

cout << "Введено неверное значение" << endl;

}

}

}

}

void oneSet(int n, TSet &S)

{

int k;

switch (n) {

case 1:

cout << "{" << S << "}" << endl;

cout << "Введите элемент для добавления в множество" << endl;

cin >> k;

S = S + k;

cout << "{" << S << "}" << endl;

break;

case 2:

cout << "{" << S << "}" << endl;

cout << "Введите элемент для исключения из множества" << endl;

cin >> k;

S = S - k;

cout << "{" << S << "}" << endl;

break;

case 3:

cout << "{" << S << "}" << endl;

cout << "Дополнение множества = {" << ~S << "}" << endl;

break;

case 4:

cout << "Введите элемент для проверки на наличие в множестве" << endl;

cin >> k;

if (S.IsMember(k))

cout << k << " является элементом множества" << endl;

else

cout << k << " не является элементом множества" << endl;

break;

default:

cout << "Введено неверное значение" << endl;

}

}

TBitField.h

#pragma once

#include <string>

#include <iostream>

using namespace std;

typedef unsigned int TELEM;

class TBitField

{

private:

int BitLen;

int MemLen;

TELEM \*pMem;

public:

TBitField();

TBitField(const int \_BitLen);

TBitField(const TBitField &bf);

//Методы реализации

int GetMemIndex(const int n) const;

TELEM GetMemMask(const int n) const;

//Доступ к битам

TELEM operator[](int n);

int GetBitLen() const;

int GetMemLen() const;

void SetBit(const int n);

void ClearBit(const int n);

int GetBit(const int n) const;

//Битовые операции

int operator==(const TBitField &bf);

TBitField& operator=(const TBitField &bf);

TBitField operator|(const TBitField &bf);

TBitField operator&(const TBitField &bf);

TBitField operator~();

friend istream &operator>>(istream &istr, TBitField &bf);

friend ostream &operator<<(ostream &ostr, const TBitField &bf);

~TBitField();

};

TBitField.cpp

#include "TBitField.h"

TBitField::TBitField()

{

BitLen = 0;

MemLen = 0;

pMem = NULL;

}

TBitField::TBitField(const int \_BitLen)

{

BitLen = \_BitLen;

MemLen = (BitLen + 31) >> 5;

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++) {

pMem[i] = 0;

}

}

TBitField::TBitField(const TBitField & bf)

{

BitLen = bf.GetBitLen();

MemLen = bf.GetMemLen();

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++) {

pMem[i] = bf.pMem[i];

}

}

int TBitField::GetMemIndex(const int n) const

{

if (n < 0) return -1;

int index = n >> 5;

return index;

}

TELEM TBitField::GetMemMask(const int n) const

{

TELEM Mask;

Mask = 1 << (n & 31);

return Mask;

}

TELEM TBitField::operator[](int n)

{

return pMem[n];

}

int TBitField::GetBitLen() const

{

return BitLen;

}

int TBitField::GetMemLen() const

{

return MemLen;

}

void TBitField::SetBit(const int n)

{

int bit = GetBit(n);

if ((bit == 1) || (n < 0) || (n > BitLen)) return;

int index = GetMemIndex(n);

int mask = GetMemMask(n);

pMem[index] = pMem[index] | mask;

}

void TBitField::ClearBit(const int n)

{

int bit = GetBit(n);

if ((bit == 0) || (n < 0) || (n > BitLen)) return;

int index = GetMemIndex(n);

int mask = GetMemMask(n);

pMem[index] = pMem[index] & ~mask;

}

int TBitField::GetBit(const int n) const

{

int index = GetMemIndex(n);

int mask = GetMemMask(n);

if (!(pMem[index] & mask))

return 0;

else

return 1;

}

int TBitField::operator==(const TBitField & bf)

{

if ((BitLen != bf.GetBitLen()) || (MemLen != bf.GetMemLen())) return 0;

for (int i = 0; i < MemLen; i++) {

if (pMem[i] != bf.pMem[i]) return 0;

}

return 1;

}

TBitField & TBitField::operator=(const TBitField & bf)

{

if (this != &bf) {

BitLen = bf.GetBitLen();

MemLen = bf.GetMemLen();

pMem = new TELEM[MemLen];

for (int i = 0; i < MemLen; i++) {

pMem[i] = bf.pMem[i];

}

}

return \*this;

}

TBitField TBitField::operator|(const TBitField & bf)

{

int len = BitLen;

if (BitLen < bf.BitLen)

len = bf.BitLen;

TBitField tmp(len);

for (int i = 0; i < MemLen; i++)

{

tmp.pMem[i] = pMem[i];

}

for (int i = 0; i < bf.MemLen; i++) {

tmp.pMem[i] |= bf.pMem[i];

}

return tmp;

}

TBitField TBitField::operator&(const TBitField & bf)

{

int len = BitLen;

if (BitLen < bf.BitLen)

len = bf.BitLen;

TBitField tmp(len);

for (int i = 0; i < MemLen; i++)

{

tmp.pMem[i] = pMem[i];

}

for (int i = 0; i < bf.MemLen; i++) {

tmp.pMem[i] &= bf.pMem[i];

}

return tmp;

}

TBitField TBitField::operator~()

{

TBitField tmp(BitLen);

for (int i = 0; i < MemLen; i++) {

tmp.pMem[i] = ~pMem[i];

}

return tmp;

}

TBitField::~TBitField()

{

delete[] pMem;

}

istream & operator>>(istream & istr, TBitField & bf)

{

char buf[256];

istr >> buf;

TBitField field = TBitField(strlen(buf));

for (int i = 0; buf[i] != 0; i++) {

if (buf[i] != '0')

field.SetBit(i);

else

field.ClearBit(i);

}

bf = field;

return istr;

}

ostream & operator<<(ostream & ostr, const TBitField & bf)

{

int len = bf.GetBitLen();

for (int i = 0; i < len; i++) {

if (bf.GetBit(i))

ostr << '1';

else

ostr << '0';

}

return ostr;

}

Tset.h

#pragma once

#include "TBitField.h"

class TSet

{

private:

int MaxPower; // максимальная мощность множества

TBitField BitField; // битовое поле для хранения хар-го вектора

public:

TSet();

~TSet();

TSet(int mp);

TSet(const TSet &s); // конструктор копирования

TSet(const TBitField &bf); // конструктор преобразования типа

operator TBitField(); // преобразование типа к битовому полю

// доступ к битам

int GetMaxPower(void) const; // максимальная мощность множества

int GetPower(void) const; // мощность множества

void InsElem(const int n); // включить элемент в множество

void DelElem(const int n); // удалить элемент из множества

int IsMember(const int n) const; // проверить наличие элемента в множестве

// теоретико-множественные операции

int operator== (const TSet &s); // сравнение

TSet& operator=(const TSet &s); // присваивание

TSet operator+ (const int n); // включение элемента в множество

TSet operator- (const int n); // удаление элемента из множества

TSet operator+ (const TSet &s); // объединение

TSet operator\* (const TSet &s); // пересечение

TSet operator~ (void); // дополнение

friend istream &operator>>(istream &istr, TSet &set);

friend ostream &operator<<(ostream &ostr, const TSet &set);

};

TSet.cpp

#include "TSet.h"

TSet::TSet()

{

MaxPower = 0;

TBitField BitField = TBitField();

}

TSet::~TSet()

{

}

TSet::TSet(int mp)

{

MaxPower = mp;

BitField = TBitField(MaxPower);

}

TSet::TSet(const TSet & s)

{

MaxPower = s.MaxPower;

BitField = s.BitField;

}

TSet::TSet(const TBitField & bf)

{

MaxPower = bf.GetBitLen();

BitField = bf;

}

TSet::operator TBitField()

{

return BitField;

}

int TSet::GetMaxPower(void) const

{

return MaxPower;

}

int TSet::GetPower(void) const

{

int count = 0;

for (int i = 0; i < MaxPower; i++) {

if (IsMember(i))

count++;

}

return count;

}

void TSet::InsElem(const int n)

{

if (n > MaxPower) { //УДАЛИТЬ В ПРИЛОЖЕНИИ

MaxPower = n;

BitField = TBitField(MaxPower) | BitField;

}

BitField.SetBit(n);

}

void TSet::DelElem(const int n)

{

BitField.ClearBit(n);

}

int TSet::IsMember(const int n) const

{

return BitField.GetBit(n);

}

int TSet::operator==(const TSet & s)

{

if (BitField == s.BitField)

return 1;

else return 0;

}

TSet & TSet::operator=(const TSet & s)

{

MaxPower = s.MaxPower;

BitField = s.BitField;

return \*this;

}

TSet TSet::operator+(const int n)

{

TSet tmp = \*this;

tmp.InsElem(n);

return tmp;

}

TSet TSet::operator-(const int n)

{

TSet tmp = \*this;

tmp.DelElem(n);

return tmp;

}

TSet TSet::operator+(const TSet & s)

{

TSet tmp;

tmp.BitField = BitField | s.BitField;

tmp.MaxPower = tmp.BitField.GetBitLen();

return tmp;

}

TSet TSet::operator\*(const TSet & s)

{

TSet tmp;

tmp.BitField = BitField & s.BitField;

tmp.MaxPower = tmp.BitField.GetBitLen();

return tmp;

}

TSet TSet::operator~(void)

{

TSet tmp(\*this);

tmp.BitField = ~BitField;

return tmp;

}

istream & operator>>(istream & istr, TSet & set)

{

cout << "Введите -1, чтобы прекратить ввод" << endl;

int buf;

do {

scanf("%d", &buf);

if (buf == -1) break;

if (buf > set.MaxPower)

cout << "Элемент не входит в множество" << endl;

set = set + buf;

} while (buf != -1);

return istr;

}

ostream & operator<<(ostream & ostr, const TSet & set)

{

for (int i = 0; i < set.BitField.GetBitLen(); i++) {

if (set.IsMember(i))

ostr << i << " ";

}

return ostr;

}