Федеральное агентство по образованию Российской Федерации

Государственное образовательное учреждение

высшего профессионального образования

Нижегородский государственный университет им. Н.И. Лобачевского

Институт информационных технологий, математики и механики

Отчёт по лабораторной работе

Множества на основе битовых полей

Выполнил:

студент института ИТММ гр. 381908-4

Иссаракура Н.А.

Проверил:

ассистент каф. МОСТ, ИИТММ

Лебедев И.Г.

Содержание

[Введение 3](#_Toc59364219)

[Постановка задачи 4](#_Toc59364220)

[Руководство пользователя 5](#_Toc59364221)

[Руководство программиста 6](#_Toc59364222)

[Описание структуры программы 6](#_Toc59364223)

[Описание структур данных 6](#_Toc59364224)

[Описание алгоритмов 9](#_Toc59364225)

[Заключение 12](#_Toc59364226)

[Литература 13](#_Toc59364227)

[Приложения 14](#_Toc59364228)

# Введение

Множество — это структура данных, эквивалентная множествам в математике.

Множество может состоять из различных элементов, порядок элементов в множестве не определен. В множество можно добавлять и удалять элементы, можно перебирать элементы множества, можно выполнять операции над множествами (объединение, пересечение, разность). Можно проверять принадлежность элементу множества.

В программирование множества помогают решать задачи различного характера. Например, есть множество всех работников предприятия и множество работников, прошедших ежемесячную проверку. Тогда с легкостью возможно найти тех, кто из работников не прошел проверку.

# Постановка задачи

Задача конкретной работы — разработка структуры данных для внедрения битовых полей с использованием множеств, кроме того, нужно было освоить инструменты разработки программного обеспечения, такие как Git и Google Test.

Используя шаблон, содержащий интерфейсы классов битового поля и множества (h-файлы), готовый набор тестов для каждого из указанных классов, пример использования класса битового поля и множества для решения задачи поиска простых чисел с помощью алгоритма "Решето Эратосфена", нужно было написать программу, решающую следующие задачи:

* Реализация класса битового поля TBitField согласно заданному интерфейсу,
* Реализация класса множества TSet согласно заданному интерфейсу,
* Обеспечение работоспособности тестов и примера использования,
* Реализация нескольких простых тестов на базе Google Test,
* Публикация исходных кодов в личном репозитории на GitHub.

# Руководство пользователя

Пользователю нужно запустить файл sample\_prime\_numbers.exe.

Откроется консольное приложение для тестирования программ поддержки битового поля на основе решета Эратосфена.

Программа запросит пользователя ввести верхнюю границу целых значений.

После ввода числа, программа выполнит действия и выведет результат в консоль.

Для повторного выполнения потребуется перезапустить программу.

# Руководство программиста

## **Описание структуры программы**

#include "tbitfield.h" – подключение файла TBitField.h, описывающего работу с битовыми полями.

#include "tset.h" – подключение файла TSet.h, описывающего работу с множествами, на основе битовых полей.

#include <iomanip> - файл описывает манипуляторы при работе с потоковыми операциями.

Код классов в приложении.

## 

## **Описание структур данных**

Программа состоит из нескольких основных классов – tbitfield, tset.

**tbitfield.cpp, заголовок tbitfield.h.**

Реализованы конструктор с параметром, конструктор копирования, деструктор.

Содержит поля:

* BitLen для хранения длины битового поля,
* TELEM \*pMem для предоставления памяти битовому полю,
* MemLen для хранения количества элементов битового поля.

Реализованы методы:

* GetMemIndex – получение индекса,
* GetMemMask – маска для бита,
* GetLength – получение длины,
* SetBit – установка бита,
* ClrBit – очистка бита,
* GetBit – получение бита.

Реализованы операции:

* operator== - оператор сравнения,
* operator!= - оператор сравнения,
* operator= - оператор присваивания,
* operator| - операция «ИЛИ»,
* operator& - оператор «И»,
* operator~ - оператор отрицания,
* &operator>> -оператор ввода,
* &operator<< - оператор вывода.

**tset.cpp, заголовок tset.h**

Реализованы конструктор с параметром, конструктор копирования, конструктор преобразования типа.

Содержит поля:

* MaxPower для хранения максимальной мощности множества,
* TBitField BitField – битовое поле для хранения характеристического вектора.

Реализует методы:

* GetMaxPower – получение максимальной мощности множества,
* InsElem – включение элемента в множество,
* DelElem – удаление элемента из множества,

Реализует операции:

* operator== - оператор сравнения,
* operator!= - оператор сравнения,
* operator= - оператор присваивания,
* operator+ - операция объединение с элементом,
* operator- - операция разность с элементом,
* operator+ - операция объединение множеств,
* operator\* - операция пересечение множеств,
* operator~ - операция дополнения,
* &operator>> -оператор ввода,
* &operator<< - оператор вывода.

# Описание алгоритмов

Создание множества:

* Инициализируем битовое поле размером, равным мощности множества,
* Выделяем память,
* Заполняем элементы нулями.

Добавление элемента в множетсво:

* Инициализируем битовое поле,
* Передаем элемент в класс битового поля,
* На основе элемента получаем индекс и маску,
* Используя побитовое «ИЛИ», присваиваем по полученному индексу, полученную маску.

Удаление элемента из множества:

* Передаем элемент в класс битового поля,
* На основе элемента получаем индекс и маску,
* Используя побитовое «И», присваиваем по полученному индексу, полученную маску, предварительно применив к маске побитовую инверсию.

Эксперименты

Результат выполнения операции присваивание:

Graphical user interface, text

Description automatically generated

Результат выполнения операции сравнения:

Graphical user interface, text

Description automatically generated

Результат операции “ИЛИ”:

Graphical user interface, text

Description automatically generated

Результат операции “И”:

Text

Description automatically generated

Результат операции отрицание:

Graphical user interface, text

Description automatically generated

Результат объединения множеств:

Text

Description automatically generated

Результат операции сравнения множеств:

Text

Description automatically generated

Результат операции пересечения множеств:

Text

Description automatically generated

Результат операции дополнение:

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

# Заключение

При выполнении работы были получены навыки работы с Google Tests и CMake и реализации множества на основе битовых полей.

# Литература

* Столлингс, В. Структурная организация и архитектура компьютерных систем, 5-е изд.: Пер. с англ. — М.: Издательский дом «Вильямс», 2002. — 896 с.: ил. — Парал. тит. англ.
* Johnson M. Superscalar Microprocessor Design. — Englewood Cliff, New Jersey: Prentice Hall, 1991.
* Касперски К. Техника оптимизации программ. Эффективное использование памяти. — СПб.: БХВ-Петербург, 2003. — 464 с.: ил.
* Stone H. High performance Computer Architecture. — Reading, MA: Addison-Wesley, 1993.
* Tullsen D.M., Eggers S.J. Effective Cache Prefetching on a Bus-Based Multiprocessor. — ACM Transactions on Computer Systems, pp. 57-88, Feb 1995.
* Chandra D., Guo F., Kim S., Solihin Y. Predicting inter-thread cache contention on a chip multi-processor architecture. — Proceedings of the 11th International Symposium on High Performance Computer Architecture (HPCA), pp. 340–351, Feb 2005.
* Press W., Teukolsky S., Vetterling W., Flannery B. Numerical Recipes in C. The Art of Scientific Computing. Second Edition. — Cambridge University Press, 1992.
* Камаев А.М., Сиднев А.А., Сысоев А.В. Об одном подходе к анализу эффективности приложений // Труды 50-й научной конференции МФТИ «Современные проблемы фундаментальных и прикладных наук»: Часть I. Радиотехника и кибернетика. - М.: МФТИ, 2007.
* Debugging and performance monitoring. Intel® 64 and IA-32 Architectures Software Developer’s Manual. Volume 3B: System Programming Guide, Part 2. May 2007. — [http://www.intel.com/products/processor/manuals/]
* Юнаковский А.Д. Начала вычислительных методов для физиков. – Н. Новгород: ИПФ РАН, 2007.

# Приложения

// ННГУ, ВМК, Курс "Методы программирования-2", С++, ООП

//

// tbitfield.cpp - Copyright (c) Гергель В.П. 07.05.2001

// Переработано для Microsoft Visual Studio 2008 Сысоевым А.В. (19.04.2015)

//

// Битовое поле

#include "tbitfield.h"

TBitField::TBitField(int len)

{

if (len < 0)

throw invalid\_argument("Negative length is not allowed.");

BitLen = len;

MemLen = (len + sizeof(TELEM) \* 8 - 1) / (sizeof(TELEM) \* 8);

pMem = new TELEM[MemLen];

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

pMem[i] = 0;

}

TBitField::TBitField(const TBitField& bf) // конструктор копирования

{

BitLen = bf.GetLength();

MemLen = (BitLen + sizeof(TELEM) \* 8 - 1) / (sizeof(TELEM) \* 8);

pMem = new TELEM[MemLen];

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

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

}

TBitField::~TBitField()

{

if (pMem != NULL) {

delete[] pMem;

pMem = NULL;

MemLen = NULL;

}

}

int TBitField::GetMemIndex(const int n) const // индекс Мем для бита n

{

if (n >= 0 && n < BitLen)

return n >> 5;

else

throw - 1;

}

TELEM TBitField::GetMemMask(const int n) const // битовая маска для бита n

{

TELEM tmp = 1 << n;

return tmp;

}

// доступ к битам битового поля

int TBitField::GetLength(void) const // получить длину (к-во битов)

{

return BitLen;

}

void TBitField::SetBit(const int n) // установить бит

{

if (n < 0)

throw invalid\_argument("Can't set negative bit.");

if (n >= BitLen)

throw invalid\_argument("Can't set bit larger than the maximum bit field size.");

pMem[GetMemIndex(n)] |= GetMemMask(n % 32);

}

void TBitField::ClrBit(const int n) // очистить бит

{

if (n < 0)

throw invalid\_argument("Can't clear negative bit.");

if (n >= BitLen)

throw invalid\_argument("Can't clear bit larger than the maximum bit field size.");

pMem[GetMemIndex(n)] &= ~GetMemMask(n % 32);

}

int TBitField::GetBit(const int n) const // получить значение бита

{

if (n < 0)

throw invalid\_argument("Can't get negative bit.");

if (n >= BitLen)

throw invalid\_argument("Can't get bit larger than the maximum bit field size.");

if ((pMem[GetMemIndex(n)] & GetMemMask(n % 32)) > 0)

return 1;

else

return 0;

}

// битовые операции

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

{

if (this == &bf) return \*this;

BitLen = bf.BitLen;

MemLen = bf.MemLen;

if (bf.MemLen != MemLen)

{

MemLen = bf.MemLen;

delete pMem;

pMem = new TELEM[MemLen];

}

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

{

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

}

return \*this;

}

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

{

if (BitLen != bf.GetLength())

return 0;

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

if (pMem[i] != bf.pMem[i])

return 0;

return 1;

}

int TBitField::operator!=(const TBitField& bf) const // сравнение

{

if (BitLen != bf.GetLength())

return 1;

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

if (pMem[i] != bf.pMem[i])

return 1;

return 0;

}

TBitField TBitField::operator|(const TBitField& bf) // операция "или"

{

if (BitLen >= bf.GetLength())

{

TBitField tmp(\*this);

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

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

return tmp;

}

else

{

TBitField tmp(bf);

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

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

return tmp;

}

}

TBitField TBitField::operator&(const TBitField& bf) // операция "и"

{

TBitField tmp(BitLen);

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

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

return tmp;

}

TBitField TBitField::operator~(void) // отрицание

{

TBitField tmp(\*this);

for (int i = 0; i < tmp.BitLen; i++)

{

if (tmp.GetBit(i))

tmp.ClrBit(i);

else

tmp.SetBit(i);

}

return tmp;

}

// ввод/вывод

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

{

char\* s;

istr >> s;

int len = sizeof(s) / sizeof(char) - 1;

TBitField tmp(len);

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

{

if (s[i] == '1')

tmp.SetBit(i);

else

tmp.ClrBit(i);

}

return istr;

}

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

{

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

ostr << bf.GetBit(i);

return ostr;

}

// ННГУ, ВМК, Курс "Методы программирования-2", С++, ООП

//

// tset.cpp - Copyright (c) Гергель В.П. 04.10.2001

// Переработано для Microsoft Visual Studio 2008 Сысоевым А.В. (19.04.2015)

//

// Множество - реализация через битовые поля

#include "tset.h"

TSet::TSet(int mp) : BitField(mp)

{

MaxPower = mp;

}

// конструктор копирования

TSet::TSet(const TSet& s) : BitField(s.BitField)

{

MaxPower = s.GetMaxPower();

}

// конструктор преобразования типа

TSet::TSet(const TBitField& bf) : BitField(bf)

{

MaxPower = bf.GetLength();

}

TSet::operator TBitField()

{

TBitField t(BitField);

return t;

}

int TSet::GetMaxPower(void) const // получить макс. к-во эл-тов

{

return MaxPower;

}

int TSet::IsMember(const int Elem) const // элемент множества?

{

if (Elem > MaxPower)

return 0;

return BitField.GetBit(Elem);

}

void TSet::InsElem(const int Elem) // включение элемента множества

{

BitField.SetBit(Elem);

}

void TSet::DelElem(const int Elem) // исключение элемента множества

{

BitField.ClrBit(Elem);

}

// теоретико-множественные операции

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

{

MaxPower = s.GetMaxPower();

BitField = s.BitField;

return \*this;

}

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

{

if (MaxPower == s.GetMaxPower() && BitField == s.BitField)

return 1;

else

return 0;

}

int TSet::operator!=(const TSet& s) const // сравнение

{

if (MaxPower == s.GetMaxPower() && BitField == s.BitField)

return 0;

else

return 1;

}

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

{

int tmpsize = (MaxPower >= s.GetMaxPower()) ? MaxPower : s.GetMaxPower();

TSet tmp(tmpsize);

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

if (IsMember(i))

tmp.InsElem(i);

for (int i = 0; i < s.GetMaxPower(); ++i)

if (s.IsMember(i))

tmp.InsElem(i);

return tmp;

}

TSet TSet::operator+(const int Elem) // объединение с элементом

{

int tmpsize = (MaxPower >= Elem) ? MaxPower : Elem;

TSet tmp(tmpsize);

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

if (IsMember(i))

tmp.InsElem(i);

tmp.InsElem(Elem);

return tmp;

}

TSet TSet::operator-(const int Elem) // разность с элементом

{

TSet tmp(\*this);

if (MaxPower < Elem)

tmp.DelElem(Elem);

return tmp;

}

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

{

if (MaxPower >= s.GetMaxPower()) {

TSet tmp(MaxPower);

for (int i = 0; i < s.GetMaxPower(); i++) {

if (IsMember(i) == 1 && s.IsMember(i) == 1)

tmp.InsElem(i);

}

return tmp;

}

else {

TSet tmp(s.GetMaxPower());

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

if (IsMember(i) == 1 && s.IsMember(i) == 1)

tmp.InsElem(i);

}

return tmp;

}

}

TSet TSet::operator~(void) // дополнение

{

TSet tmp(MaxPower);

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

if (!IsMember(i))

tmp.InsElem(i);

return tmp;

}

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод

{

istr >> s.BitField;

return istr;

}

ostream& operator<<(ostream& ostr, const TSet& s) // вывод

{

ostr << s.BitField;

return ostr;

}