

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский
Нижегородский государственный университет им. Н.И. Лобачевского»
(ННГУ)

Институт информационных технологий, математики и механики

ЛАБОРАТОРНАЯ РАБОТА

на тему:
«Битовые поля и множества»

Выполнил: студент группы 3822Б1ФИ2

_____/Миронов А. И./
Подпись

Проверил: к.т.н, доцент каф. ВВиСП

_____/Кустикова В.Д./
Подпись

Нижний Новгород
2023

Содержание

Введение.....	3
1 Постановка задачи.....	4
2 Руководство пользователя.....	5
2.1 Приложение для демонстрации работы битовых полей	5
2.2 Приложение для демонстрации работы множеств	7
2.3 «Решето Эратосфена».....	8
3 Руководство программиста	9
3.1 Описание алгоритмов	9
3.1.1 Битовые поля	9
3.1.2 Множества	12
3.1.3 «Решето Эратосфена»	15
3.2 Описание программной реализации	16
3.2.1 Описание класса TBitField	16
3.2.2 Описание класса Tset	21
Заключение	25
Литература	26
Приложения	27
Приложение А. Реализация класса TBitField	27
Приложение Б. Реализация класса TSet.....	35

Введение

В современном программировании и анализе данных часто возникает необходимость работы с большими объемами информации и эффективным представлением данных. Битовые поля и множества являются одним из инструментов, которые позволяют компактно хранить и манипулировать множествами элементов. Битовые поля и множества позволяют представить множество элементов в виде битовых векторов, где каждый бит соответствует наличию или отсутствию элемента в множестве. Такое представление позволяет существенно сократить объем памяти, занимаемый множеством, и ускорить операции над ними

1 Постановка задачи

Цель работы: изучение и практическое применение концепции битовых полей и множеств.

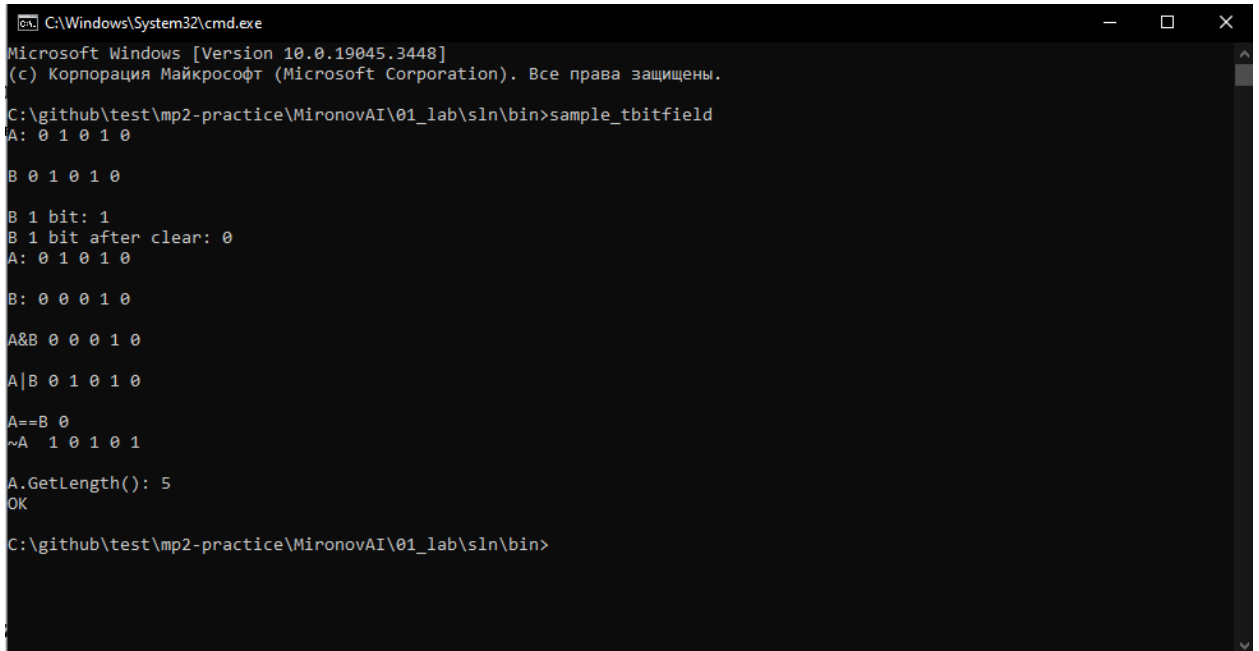
Задачи:

1. Изучить теоретические основы битовых полей и множеств.
2. Разработать программу, реализующую операции над битовыми полями и множествами.
3. Провести эксперименты с различными наборами данных.
4. Проанализировать полученные результаты и сделать выводы о преимуществах и ограничениях использования битовых полей и множеств.

2 Руководство пользователя

2.1 Приложение для демонстрации работы битовых полей

1. Запустите приложение с названием sample_tbitfield.exe. В результате появится окно, показанное ниже (**Ошибка! Источник ссылки не найден.**).



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.19045.3448]
(c) Корпорация Майкрософт (Microsoft Corporation). Все права защищены.

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>sample_tbitfield
A: 0 1 0 1 0

B 0 1 0 1 0

B 1 bit: 1
B 1 bit after clear: 0
A: 0 1 0 1 0

B: 0 0 0 1 0

A&B 0 0 0 1 0

A|B 0 1 0 1 0

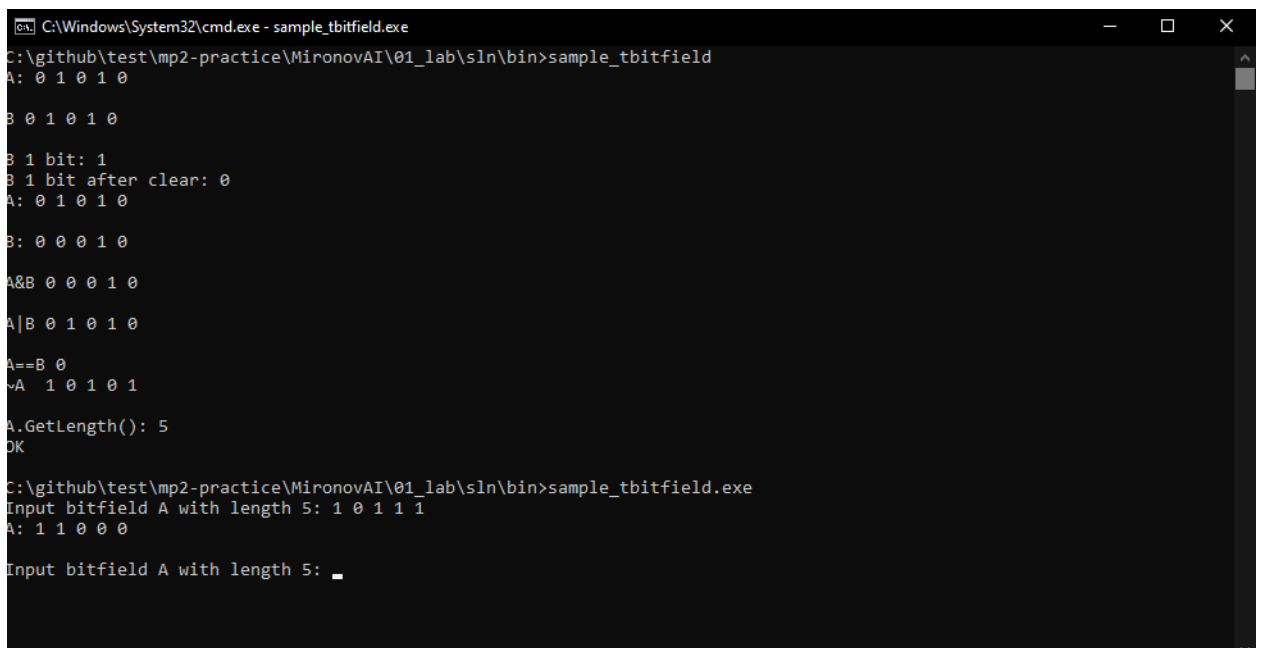
A==B 0
~A 1 0 1 0 1

A.GetLength(): 5
OK

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>
```

Рис. 1. Основное окно программы

2. Затем вам будет предложено ввести 2 битовых поля длины 5 (рис 2).



```
C:\Windows\System32\cmd.exe - sample_tbitfield.exe
C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>sample_tbitfield
A: 0 1 0 1 0

B 0 1 0 1 0

B 1 bit: 1
B 1 bit after clear: 0
A: 0 1 0 1 0

B: 0 0 0 1 0

A&B 0 0 0 1 0

A|B 0 1 0 1 0

A==B 0
~A 1 0 1 0 1

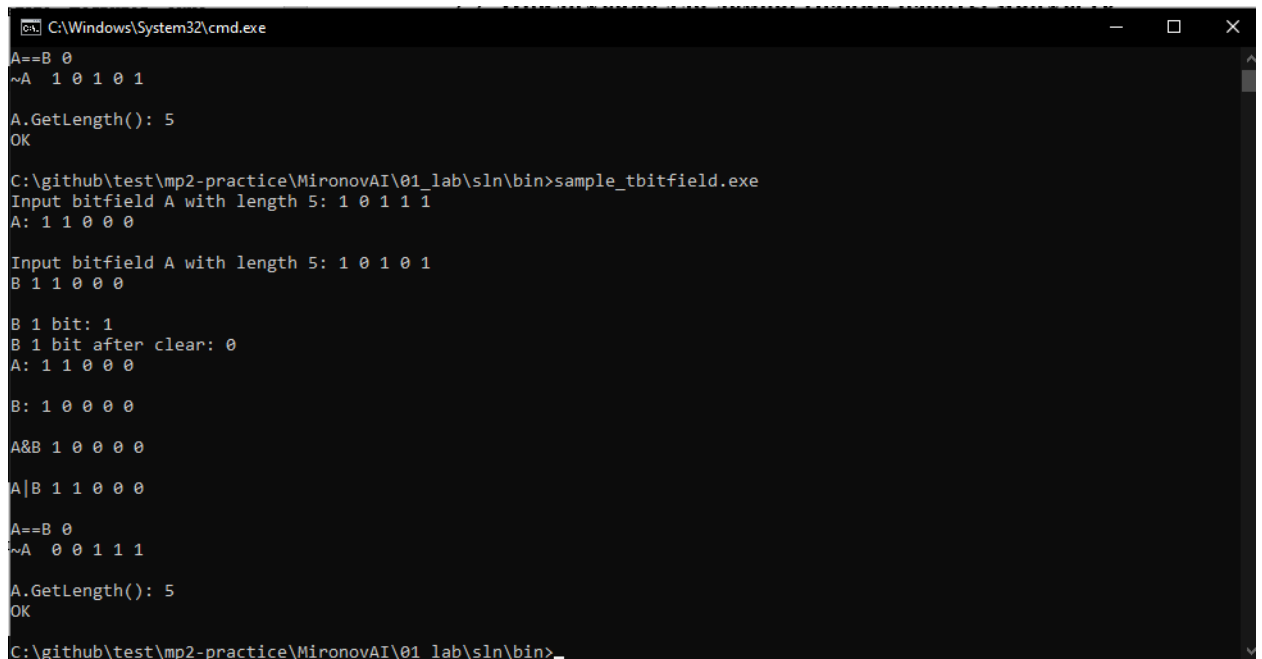
A.GetLength(): 5
OK

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>sample_tbitfield.exe
Input bitfield A with length 5: 1 0 1 1 1
A: 1 1 0 0 0

Input bitfield A with length 5: _
```

Рис. 2. Ввод битовых полей

3. После ввода множества нулей и /или единиц, будет выведены результаты соответствующих операций и функций (рис 3).



```
C:\Windows\System32\cmd.exe
A==B 0
~A 1 0 1 0 1

A.GetLength(): 5
OK

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>sample_tbitfield.exe
Input bitfield A with length 5: 1 0 1 1 1
A: 1 1 0 0 0

Input bitfield A with length 5: 1 0 1 0 1
B 1 1 0 0 0

B 1 bit: 1
B 1 bit after clear: 0
A: 1 1 0 0 0

B: 1 0 0 0 0

A&B 1 0 0 0 0

A|B 1 1 0 0 0

A==B 0
~A 0 0 1 1 1

A.GetLength(): 5
OK

C:\github\test\mp2-practice\MironovAI\01_lab\sln\bin>
```

Рис. 3. Результат тестирования функций класса TBitField

2.2 Приложение для демонстрации работы множеств

1. Запустите приложение с названием sample_tset.exe. В результате появится окно, показанное ниже (рис 4).

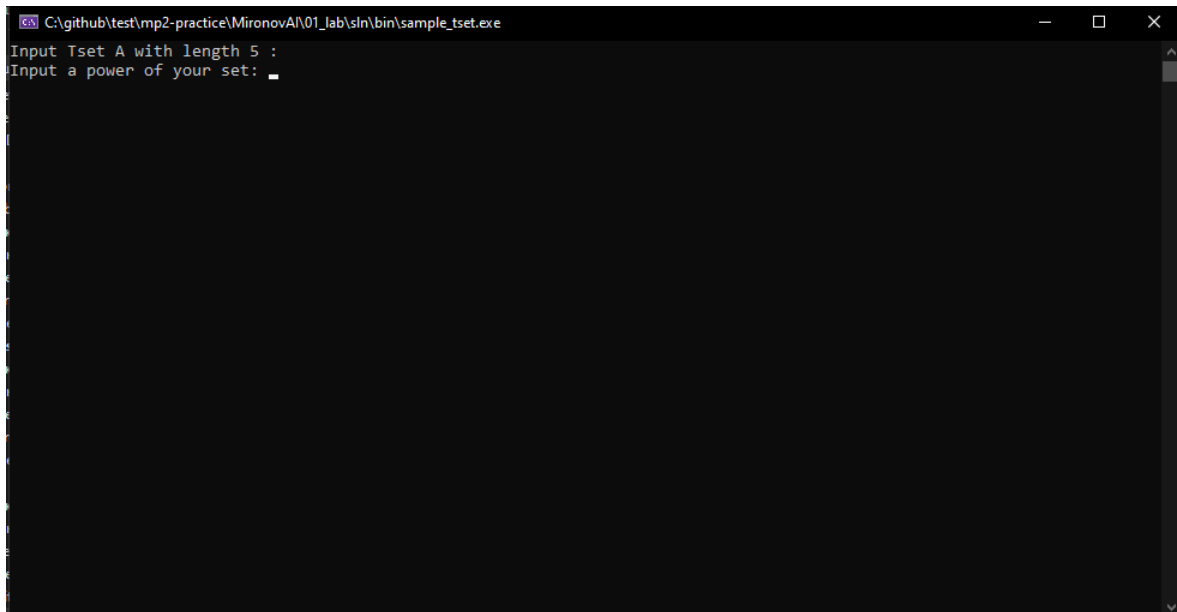


Рис. 4. Основное окно программы

2. Затем вам будет предложено ввести свое множество. Необходимо ввести сначала количество чисел в множестве, программа будет ждать N чисел. Причём они не должны превышать 5. После ввода множества нулей и /или единиц, будет выведены результаты соответствующих операций и функций (рис 5).

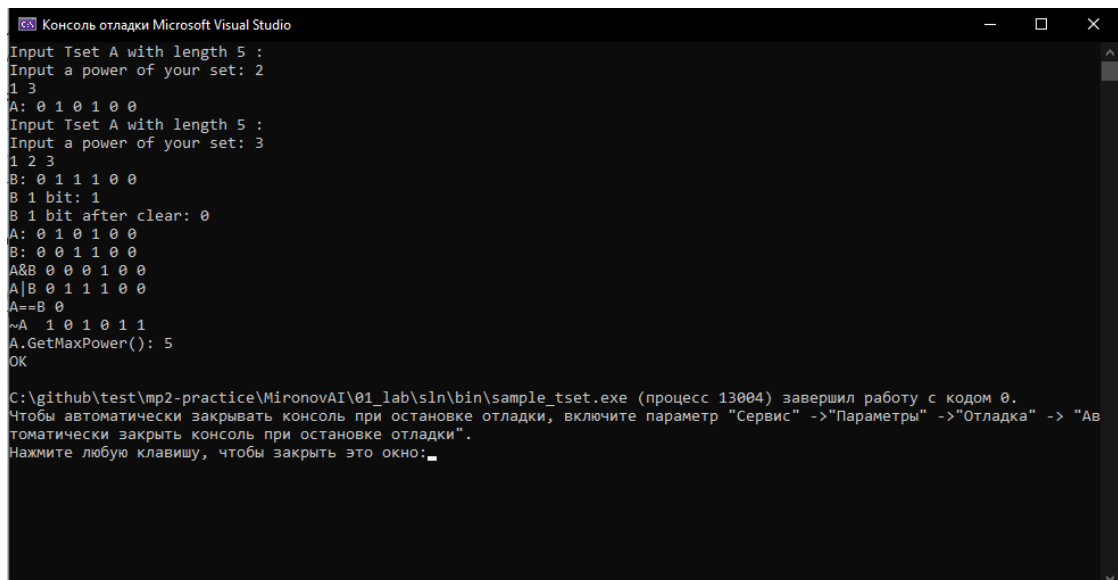


Рис. 5. Результат тестирования класса **TSet**

2.3 «Решето Эратосфена»

1. Запустите приложение с названием sample_tset.exe. В результате появится окно, показанное ниже (рис 6).

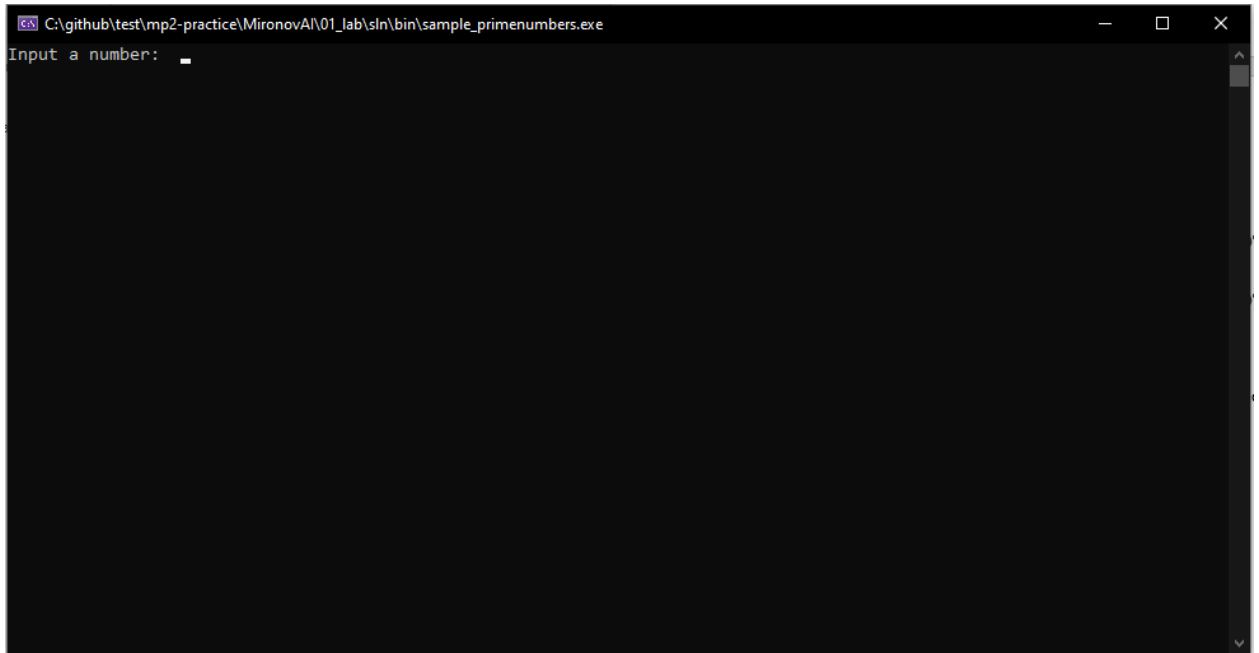


Рис. 6. Основное окно программы

2. Затем вам будет нужно ввести целое положительное число. После чего программа выведет простые числа на отрезке до введенного числа и их количество (рис 7).

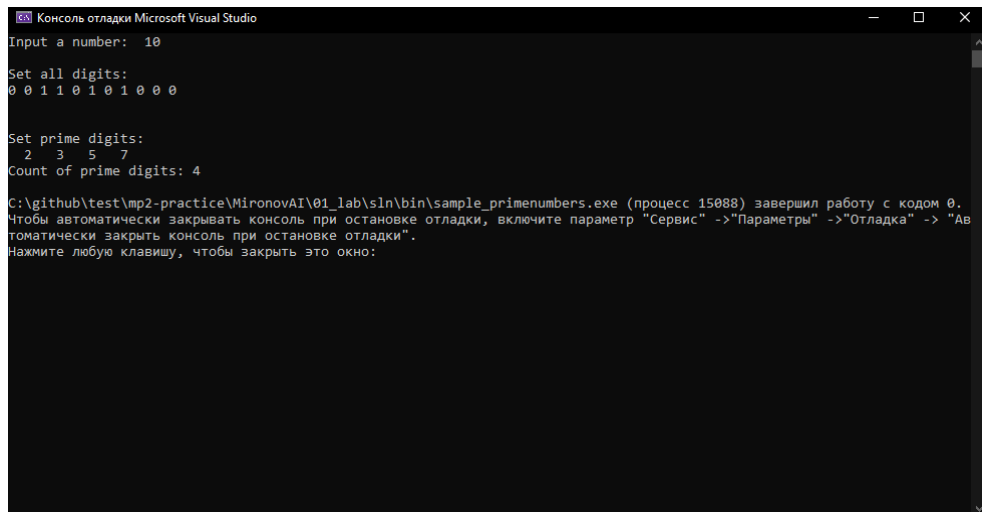


Рис. 7. Результат работы алгоритма «Решето Эратосфена»

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Битовые поля

Начало работы указано в пункте [2.1 этого документа](#)

Описание методов и полей класса в пункте [3.2.1 этого документа](#)

Битовые поля представляют собой набор чисел, каждый бит которых интерпретируется элементом, равным индексом бита. Битовые поля обеспечивают удобный доступ к отдельным битам данных. Они позволяют формировать объекты с длиной, не кратной байту, что в свою очередь позволяет экономить память, более плотно размещая данные.

Битовое поле хранится в виде класса с полями: массив беззнаковых целых чисел, каждое из которых имеет размер 32 бита, максимальный элемент(количество битов), количество беззнаковых целых чисел, которые образуют битовое поле.

Пример битового поля длины 5: 01100

Битовое поле поддерживает операции объединения, пересечения, дополнение (отрицание), сравнения, ввода и вывода.

Операция объединения:

Операция возвращает экземпляр класса, каждый бит которого равен 1, если он есть хотя бы в 1 классе, которые объединяем, и 0 в противном случае.

Пример:

0	1	1	0	1	1
1	0	0	0	1	1
1	1	1	0	1	1

Операция пересечения:

Операция возвращает экземпляр класса, каждый бит которого равен 1, если он есть в каждом классе, и 0 в противном случае.

Пример:

0	1	1	0	1	1
1	0	0	0	1	1
0	0	0	0	1	1

Операция дополнения (отрицания):

Операция возвращает экземпляр класса, каждый бит которого равен 0, если он есть в исходном классе, и 1 в противном случае.

Пример:

0	1	1	0	1	1
1	0	0	1	0	0

Операция ввода битового поля из консоли:

Операция позволяет ввести битовое поле из консоли. Для этого необходимо в консоль ввести 1 или 0 столько раз, какова длина вводимого битового поля.

Пример:

0	0	0	0	0	0
---	---	---	---	---	---

После ввода:

0	1	1	0	1	1
---	---	---	---	---	---

Операция вывода битового поля в консоль:

Операция вывода позволяет вывести битовое поле в консоль

Также в битовом поле можно установить или отчистить бит

Пример:

0	1	1	0	1	1
---	---	---	---	---	---

Результат установление 1 бита:

1	1	1	0	1	1
---	---	---	---	---	---

Пример:

0	1	1	0	1	1
---	---	---	---	---	---

Результат удаления 2 бита:

0	0	1	0	1	1
---	---	---	---	---	---

Замечание: Введенный бит должен быть больше 0 и меньше длины битового поля

Операции сравнения

Операция равенства выведет 1 , если два битовых поля равны, или каждые их биты совпадают, 0 в противном случае. Операция , обратная операции равенства, выведет 0, если хотя бы два бита совпадают, 1 в противном случае.

3.1.2 Множества

Начало работы указано в пункте [2.2 этого документа](#)

Описание методов и полей класса в пункте [3.2.2 этого документа](#)

Множества представляют собой набор целых положительных чисел. В данной лабораторной работе множество реализовано при помощи битового поля, соответственно каждый бит которых интерпретируется элементом, равным индексом бита. Битовые поля обеспечивают удобный доступ к отдельным битам данных. Они позволяют формировать объекты с длинной, не кратной байту, что в свою очередь позволяет экономить память, более плотно размещая данные. Создание множества через битовые поля может сильно сократить использование памяти.

Множество поле хранится в виде класса с полями: Битовое поле, максимальный элемент множества.

Пример множества максимальной длины 5: $A: \{1, 2, 3, 5\}$

Множество поддерживает операции объединения, пересечения, дополнение (отрицание), сравнения, ввода и вывода.

Операция объединения с множеством:

Операция возвращает экземпляр класса, содержащий все уникальные элементы из двух классов.

Пример:

$$A = \{1, 2, 3\}$$

$$B = \{1, 3, 5\}$$

Результат объединения множеств $A \cup B$:

$$A \cup B = \{1, 2, 3, 5\}$$

Операция пересечения с множеством:

Операция возвращает экземпляр класса, содержащий все уникальные элементы, которые находятся в каждом классе.

Пример:

$$A = \{1, 2, 3\}$$

$$B = \{1, 3, 5\}$$

Результат пересечения множеств $A \cap B$:

$$A \cap B = \{1, 3\}$$

Операции объединения и пересечения также работают с целым числом, они будут эквивалентны функциям добавления или удаления элемента в (из) множество (множества)

Операция дополнения (отрицания):

Операция возвращает экземпляр, класса содержащий элементы, которых нет в исходном классе, не большие максимальному элементу.

Пример:

$$A = \{1, 2, 3\}$$

Результат объединения множеств $\sim A$:

$$\sim A = \{4, 5\}$$

Оп

Операция ввода множества из консоли:

Операция позволяет ввести множество из консоли. Для этого необходимо в консоль ввести сначала количество элементов в множестве, а дальше сами элементы, причём элементы не могут превышать максимального элемента множества.

Пример:

$$A = \{ \}$$

После ввода:

$$A = \{1, 2, 3\}$$

Операция вывода битового поля в консоль:

Операция вывода позволяет вывести множество в консоль

Также в множестве можно добавить или убрать элемент.

Пример:

$$A = \{1, 2, 3\}$$

Результат добавления 4:

$$A = \{1, 2, 3, 4\}$$

Пример:

$$A = \{1, 2, 3\}$$

Результат удаления 2:

$$A = \{1, 3\}$$

Замечание: Введенное число должно быть больше 0 и меньше максимального элемента множества

3.1.3 «Решето Эратосфена»

Начало работы указано в пункте [2.3 текущего документа](#)

Данный алгоритм реализован двумя способами, при помощи классов **TSet** и **TBitField**. Изначально алгоритм будет использовать класс TBitField. Для того, чтобы использовать реализацию с использованием TSet, необходимо убрать два слеша перед **#define USE_SET** (

Сначала необходимо ввести целое положительное число, до которого будет . После чего алгоритм заполняет все элементы классов равными 1.

После чего алгоритм , начинает перебирать все числа от 2 до N.

1. Если это число есть в нашем множестве, то мы переходим к шагу 2, иначе к шагу 3.
2. Это число, и дальше все кратные ему числа удаляются из нашего множества.
3. Выбирается следующее число. Если это число больше N, то алгоритм заканчивается, иначе – переход к шагу 1.

После чего выводятся все числа от 1 до N с идентификатором (0, если оно не простое, 1 если простое), затем выводятся все простые числа и их количество.

3.2 Описание программной реализации

3.2.1 Описание класса TBitField

```
class TBitField
{
private:
    int BitLen;
    TELEM *pMem;
    int MemLen;
    int GetMemIndex(const int n) const;
    TELEM GetMemMask (const int n) const;
public:
    TBitField(int len);
    TBitField(const TBitField &bf);
    ~TBitField();

    // доступ к битам
    int GetLength(void) const;
    void SetBit(const int n);
    void ClrBit(const int n);
    int GetBit(const int n) const;

    int operator==(const TBitField &bf) const;
    int operator!=(const TBitField &bf) const;
    const TBitField& operator=(const TBitField &bf);
    TBitField operator|(const TBitField &bf);
    TBitField operator&(const TBitField &bf);
    TBitField operator~(void);

    friend istream &operator>>(istream &istr, TBitField &bf);
    friend ostream &operator<<(ostream &ostr, const TBitField
&bf);
};
```

Назначение: представление битового поля.

Поля:

BitLen – длина битового поля – максимальное количество битов.

pMem – память для представления битового поля.

MemLen – количество элементов для представления битового поля.

Методы:

```
int GetMemIndex(const int n) const;
```

Назначение: получение индекса элемента в памяти...

Входные параметры:

n – номер бита.

Выходные параметры:

Номер элемента в памяти.

TELEM GetMemMask (const int n) const;

Назначение: Получение битовой маски

Входные параметры:

n – номер бита

Выходные параметры:

Битовая маска типа

TBitField(int len) ;

Назначение: конструктор с параметром, выделение памяти

Входные параметры:

len – длина битового поля

Выходные параметры:

Отсутствуют

TBitField(const TBitField &bf) ;

Назначение: конструктор копирования. Создание экземпляра класса на основе другого экземпляра

Входные параметры:

bf – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры:

Отсутствуют

~TBitField() ;

Назначение: деструктор. Отчистка выделенной памяти

Входные и выходные параметры отсутствуют

int GetLength(void) const;

Назначение: получение длины битового поля

Входные параметры отсутствуют

Выходные параметры: длина битового поля

void SetBit(const int n)

Назначение: установить бит = 1

Входные параметры:

n - номер бита, который нужно установить

Выходные параметры отсутствуют

void ClrBit(const int n);

Назначение: отчистить бит (установить бит = 0)

Входные параметры:

n - номер бита, который нужно отчистить

Выходные параметры отсутствуют

int GetBit(const int n) const;

Назначение: вывести бит (узнать бит)

Входные параметры:

n - номер бита, который нужно вывести (узнать)

Выходные параметры: бит (1 или 0, в зависимости есть установлен он, или нет)

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

Назначение: оператор сравнения. Сравнить на равенство 2 битовых поля

Входные параметры:

bf – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

int operator!=(const TBitField &bf) const;

Назначение: оператор сравнения. Сравнить на равенство 2 битовых поля

Входные параметры:

bf – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

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

Назначение: оператор присваивания. Присвоить экземпляру ***this** экземпляр **&bf**

Входные параметры:

bf – битовое поле, которое мы присваиваем

Выходные параметры: ссылка на экземпляр класса **TBitField**, ***this**

TBitField operator|(const TBitField &bf);

Назначение: оператор побитового «ИЛИ»

Входные параметры:

bf – битовое поле

Выходные параметры: Экземпляр класса , который равен { ***this | bf** }

TBitField operator&(const TBitField &bf);

Назначение: оператор побитового «И»

Входные параметры:

bf – битовое поле, с которым мы сравниваем

Выходные параметры: Экземпляр класса , который равен { ***this & bf** }

TBitField operator~(void);

Назначение: оператор инверсии

Входные параметры отсутствуют

Выходные параметры: Экземпляр класса, каждый элемент которого равен {**~*this**}, т.е. если *i* бит исходного экземпляра будет равен 1 , то на выходе он будет иметь 0.

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

Назначение: оператор ввода из консоли

Входные параметры:

istr – буфер консоли

bf – класс, который нужно ввести из консоли

Выходные параметры:

Ссылка на буфер (поток) **istr**

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

Назначение: оператор вывода из консоли

Входные параметры:

istr – буфер консоли

bf — класс, который нужно вывести в консоль

Выходные параметры: Ссылка на буфер (поток) **istr**

3.2.2 Описание класса Tset

```
class TSet
{
private:
    int MaxPower;
    TBitField BitField;
public:
    TSet(int mp);
    TSet(const TSet &s);
    TSet(const TBitField &bf);
    operator TBitField();
    int GetMaxPower(void) const;
    void InsElem(const int Elem);
    void DelElem(const int Elem);
    int IsMember(const int Elem) const;
    int operator== (const TSet &s) const;
    int operator!= (const TSet &s) const;
    const TSet& operator=(const TSet &s);
    TSet operator+ (const int Elem);
    TSet operator- (const int Elem);
    TSet operator+ (const TSet &s);
    TSet operator* (const TSet &s);
    TSet operator~ (void);
    friend istream &operator>>(istream &istr, TSet &bf);
    friend ostream &operator<<(ostream &ostr, const TSet &bf);
};
```

Назначение: представление множества чисел.

Поля:

MaxPower — максимальный элемент множества.

BitField — экземпляр битового поля, на котором реализуется множество.

Методы:

TSet(int mp);

Назначение: конструктор с параметром, выделение памяти

Входные параметры:

mp — максимальный элемент множества.

Выходные параметры:

Отсутствуют

TSet(const TSet &s);

Назначение: конструктор копирования. Создание экземпляра класса на основе другого экземпляра

Входные параметры:

s – ссылка, адрес экземпляра класса, на основе которого будет создан другой.

Выходные параметры:

Отсутствуют

~TSet();

Назначение: деструктор. Отчистка выделенной памяти

Входные и выходные параметры отсутствуют

int GetMaxPower(void) const;

Назначение: получение максимального элемента множества

Входные параметры отсутствуют

Выходные параметры: максимальный элемент множества

void InsElem(const int Elem)

Назначение: добавить элемент в множество

Входные параметры:

Elem - добавляемый элемент

Выходные параметры отсутствуют

void DelElem(const int Elem)

Назначение: удалить элемент из множества

Входные параметры:

Elem - удаляемый элемент

Выходные параметры отсутствуют

int IsMember(const int Elem) const;

Назначение: узнать, есть ли элемент в множестве

Входные параметры:

Elem - элемент, который нужно проверить на наличие

Выходные параметры: 1 или 0, в зависимости есть элемент в множестве, или нет

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

Назначение: оператор сравнения. Сравнить на равенство 2 множества

Входные параметры:

s – битовое поле, с которым мы сравниваем

Выходные параметры: 1 или 0, в зависимости равны они, или нет соответственно

int operator!=(const TSet &s) const;

Назначение: оператор сравнения. Сравнить на равенство 2 множества

Входные параметры:

s – битовое поле, с которым мы сравниваем

Выходные параметры: 0 или 1, в зависимости равны они, или нет соответственно

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

Назначение: оператор присваивания. Присвоить экземпляру ***this** экземпляр **s**

Входные параметры:

s – множество, которое мы присваиваем

Выходные параметры: ссылка на экземпляр класса **TSet**, ***this**

TSet operator+(const TSet &bf);

Назначение: оператор объединения множеств

Входные параметры:

s - множество;

Выходные параметры: Экземпляр класса, который равен { ***this | s** }

TSet operator*(const TSet &bf);

Назначение: оператор пересечения множеств

Входные параметры:

s - множество;

Выходные параметры: Экземпляр класса, который равен { ***this & s** }
}

TBitField operator~(void);

Назначение: оператор дополнение до Универса

Входные параметры отсутствуют

Выходные параметры: Экземпляр класса, каждый элемент которого равен { **~*this** }, т.е. если *i* элемент исходного экземпляра будет равен будет находится в множестве, то на выходе его не будет, и наоборот

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

Назначение: оператор ввода из консоли

Входные параметры:

istr – буфер консоли

s – класс, который нужно ввести из консоли

Выходные параметры:

Ссылка на буфер (поток) **istr**

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

Назначение: оператор вывода из консоли

Входные параметры:

istr – буфер консоли

s – класс, который нужно вывести в консоль

Выходные параметры: Ссылка на буфер (поток) **istr**

```
operator TBitField();
```

Назначение: вывод поля **BitField**

Входные параметры отсутствуют

Выходные данные: поле **BitField**

```
TSet operator+(const int Elem);
```

Назначение: оператор объединения множества и элемента. Данный оператор аналогичен метод добавления элемента в множество

Входные параметры:

Elem - число

Выходные параметры: исходный экземпляр класса, содержащий **Elem**

```
TSet operator+(const int Elem);
```

Назначение: оператор объединения множества и элемента. Данный оператор аналогичен методу удаления элемента из множества.

Входные параметры:

Elem - число

Выходные параметры: исходный экземпляр класса, не содержащий **Elem**

Заключение

В ходе выполнения работы "Битовые поля и множества" были изучены и практически применены концепции битовых полей и множеств.

Были достигнуты следующие результаты:

1. Были изучены теоретические основы битовых полей и множеств.
2. Была разработана программа, реализующая операции над битовыми полями и множествами. В ходе экспериментов была оценена эффективность работы этих операций и сравнена с другими подходами. Результаты показали, что использование битовых полей и множеств позволяет существенно сократить объем памяти и ускорить операции над множествами.
3. Были проанализированы полученные результаты и сделаны выводы о преимуществах и ограничениях использования битовых полей и множеств. Оказалось, что эти структуры данных особенно полезны при работе с большими объемами данных, где компактность представления и эффективность операций являются ключевыми факторами.

Литература

1. Википедия [https://ru.wikipedia.org/wiki/Битовое_поле].

Приложения

Приложение А. Реализация класса TBitField

```
#include "tbitfield.h"

#define size 32 //sizeof(ui)

TBitField::TBitField(int len)
{
    if (len <= 0)
        throw "len_is_LEq_zero!";

    BitLen = len;

    MemLen = (BitLen - 1) / (size) + 1;

    pMem = new TELEM[MemLen];

    // зануление битов (может из-за поведения заполнить
"мусором")

    for (int i = 0; i < MemLen; ++i)
    {
        pMem[i] = 0;
    }
}

TBitField::TBitField(const TBitField& bf) // конструктор
копирования
{
    BitLen = bf.BitLen;

    MemLen = bf.MemLen;

    pMem = new TELEM[MemLen];

    for (int i = 0; i < MemLen; ++i)
    {
        pMem[i] = bf.pMem[i];
    }
}
```

```

    }

}

TBitField::~TBitField()
{
    delete[] pMem;
}

int TBitField::GetMemIndex(const int n) const // индекс Мем для
бита n
{
    if (n < 0)
        throw "n_is_below_zero";
    return n / (size);
}

TELEM TBitField::GetMemMask(const int n) const // битовая маска
для бита n
{
    if (n > BitLen)
        throw "overload_n";
    if (n < 0)
        throw "n_is_below_zero";
    TELEM Mask = 1u << (size - n % size - 1);
    return Mask;
}

// 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 > BitLen)

        throw "overload_n";

    if (n < 0)

        throw "n_is_below_zero";


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

}


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

{

    if (n > BitLen)

        throw "overload_n";

    if (n < 0)

        throw "n_is_below_zero";

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

}


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

{

    if (n > BitLen)

```

```

        throw "overload_n";

    if (n < 0)

        throw "n_is_below_zero";

    return (pMem[GetMemIndex(n)] & (GetMemMask(n))) >> (size - n
% size - 1);
}

// битовые операции

const TBitField& TBitField::operator=(const TBitField& bf) //
присваивание
{
    if (*this == bf) return *this;

    if (BitLen != bf.BitLen)
    {
        delete[] pMem;

        pMem = new TELEM[MemLen];
    }

    BitLen = bf.BitLen;
    MemLen = bf.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;
    else
    {
        for (int i = 0; i < BitLen; ++i)
        {
            if (GetBit(i) != bf.GetBit(i))
                return 0;
        }
    }
    return 1;
}

int TBitField::operator!=(const TBitField& bf) const //
сравнение
{
    return !(*this == bf);
}

TBitField TBitField::operator|(const TBitField& bf) // операция
"или"
{
    const int maxlen = max(GetLength(), bf.GetLength());
    const int minlen = min(GetLength(), bf.GetLength());
    TBitField A(maxlen);

```

```

        if (GetLength() > bf.GetLength())
            A = *this;
        else
            A = bf;

        int i = 0;
        for (; i < minlen; ++i)
        {
            if (GetBit(i) || bf.GetBit(i))
                A.SetBit(i);
        }
        return A;
    }

```

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

```

{
    const int maxlen = max(GetLength(), bf.GetLength());
    const int minlen = min(GetLength(), bf.GetLength());
    TBitField A(maxlen);
    int i = 0;
    for (; i < minlen; ++i)
    {
        if (GetBit(i) && bf.GetBit(i))
            A.SetBit(i);
    }
    return A;
}

```

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


```

{
    TBitField A(GetLength());
    for (int i = 0; i <= GetMemIndex(GetLength()); ++i)
    {
        A.pMem[i] = ~pMem[i];
    }
    return A;
}

// ВВОД/ВЫВОД

istream& operator>>(istream& istr, TBitField& bf) // ввод
{
    for (int i = 0; i < bf.GetLength(); ++i)
    {
        int val;
        istr >> val;
        if (val > bf.GetLength() || val < 0)
            throw "Wrong element ";
        bf.SetBit(val);
    }
    return itr;
}

ostream& operator<<(ostream& ostr, const TBitField& bf) // вывод
{
    for (int i = 0; i < bf.GetLength(); ++i)
    {

```

```
        ostr << bf.GetBit(i) << " ";  
    }  
    ostr << "\n";  
    return ostr;  
}
```

Приложение Б. Реализация класса TSet

```
#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()
{
    return BitField;
}

int TSet::GetMaxPower(void) const // получить макс. к-во эл-
тов
{
    return MaxPower;
}

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

```

{
    if (Elem < 0 && Elem >= MaxPower)
        throw "Out of Range";
    return BitField.GetBit(Elem);
}

void TSet::InsElem(const int Elem) // включение элемента
множества
{
    if (Elem < 0 && Elem >= MaxPower)
        throw "Out of Range";
    return BitField.SetBit(Elem);
}

void TSet::DelElem(const int Elem) // исключение элемента
множества
{
    if (Elem < 0 && Elem >= MaxPower)
        throw "Out of Range";
    return BitField.ClrBit(Elem);
}

// теоретико-множественные операции

const TSet& TSet::operator=(const TSet& s) // присваивание
{
    if (*this == s) return *this;
    MaxPower = s.GetMaxPower();
    BitField = TBitField(MaxPower);
    BitField = BitField | s.BitField; // bitfield =
s.bitfield
    return *this;
}

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

```

```

{
    if (MaxPower != s.GetMaxPower())
        return 0;

    for (int i = 0; i < MaxPower; ++i)
    {
        if (BitField.GetBit(i) != s.BitField.GetBit(i))
            return 0;
    }
    return 1;
}

int TSet::operator!=(const TSet& s) const // сравнение
{
    return !(*this == s);
}

TSet TSet::operator+(const TSet& s) // объединение
{
    TSet A(max(MaxPower, s.GetMaxPower()));
    A.BitField = BitField | s.BitField;
    return A;
}

TSet TSet::operator+(const int Elem) // объединение с
элементом
{
    if (Elem<0 && Elem>MaxPower)
        throw "Out of Range";
    TSet A(*this);
    A.BitField.SetBit(Elem);
    return A;
}

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

```

```

{
    if (Elem<0 && Elem>MaxPower)
        throw "Out of Range";
    TSet A(*this);
    A.BitField.ClrBit(Elem);
    return A;
}

TSet TSet::operator*(const TSet& s) // пересечение
{
    TSet A(max(MaxPower, s.GetMaxPower()));
    A.BitField = BitField & s.BitField;
    return A;
}

TSet TSet::operator~(void) // дополнение
{
    TSet A(MaxPower);
    A.BitField = ~BitField;
    return A;
}

// перегрузка ввода/вывода

istream& operator>>(istream& istr, TSet& s) // ввод
{
    const int x = s.MaxPower;
    for (int i = 0; i <= x; ++i)
    {
        int val; istr >> val;
        if (val > s.MaxPower)
            throw "Wrong element ";
        s.InsElem(val);
    }
    return istr;
}

```

```
}
```

```
ostream& operator<<(ostream& ostr, const TSet& s) // вывод  
{  
    const int x = s.MaxPower;  
    for (int i = 0; i <= x; ++i)  
    {  
        ostr << s.IsMember(i) << " ";  
    }  
    return ostr;  
}
```