

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(а): студент(ка) группы _____
_____/ Хохлов А.Д./
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	5
2.3 «Решето Эратосфена»	6
3 Руководство программиста	7
3.1 Описание алгоритмов	7
3.1.1 Битовые поля	7
3.1.2 Множества	9
3.1.3 «Решето Эратосфена»	10
3.2 Описание программной реализации	11
3.2.1 Описание класса TBitField	11
3.2.2 Описание класса TSet	14
Заключение	18
Литература	19
Приложения	20
Приложение А. Реализация класса TBitField	20
Приложение Б. Реализация класса TSet.....	22

Введение

В современном мире информационных технологий большую роль играют операции с большими объемами данных. Одной из важных операций является эффективная работа с битовыми множествами. Битовое множество - структура данных, которая позволяет компактно хранить и оперировать набором битов. Такая структура данных может быть использована в различных областях: от алгоритмов сжатия данных до обработки больших объемов информации.

Актуальность и применяемость данной лабораторной работы состоит в том, что она позволяет познакомиться с основными принципами работы с битовыми множествами и реализовать их на языке программирования C++. Использование битовых множеств позволяет существенно ускорить операции с большими объемами данных и снизить требования по памяти.

Таким образом, данная лабораторная работа является актуальной и полезной для студентов и специалистов в области информационных технологий, которые имеют необходимость эффективно работать с битами и битовыми множествами.

1 Постановка задачи

Целью данной лабораторной работы является создание битового множества на языке программирования C++. В рамках работы необходимо разработать классы TBitField и TSet, которые будут предоставлять функциональность для работы с битовыми полями и множествами. Основной задачей является реализация основных операций с битовыми множествами, таких как установка, сброс, проверка наличия бита, а также операции с множествами, такие как объединение, пересечение и разность.

Задачи данной лабораторной работы:

1. Разработка класса TBitField, который будет предоставлять функциональность для работы с битовыми полями.
2. Реализация основных операций с битовыми полями, включая установку, сброс и проверку наличия определенного бита.
3. Определение класса TSet, который будет предоставлять функциональность для работы с множествами
4. Реализация основных операций с множествами, включая объединение, пересечение и разность.
5. Проверка и демонстрация работы разработанных классов с помощью приложений для работы с битовыми полями, множествами и решением задачи "Решето Эратосфена".
6. Написание отчета о выполненной лабораторной работе, включая описание алгоритмов, программной реализации и результатов работы.

2 Руководство пользователя

2.1 Приложение для демонстрации работы битовых полей

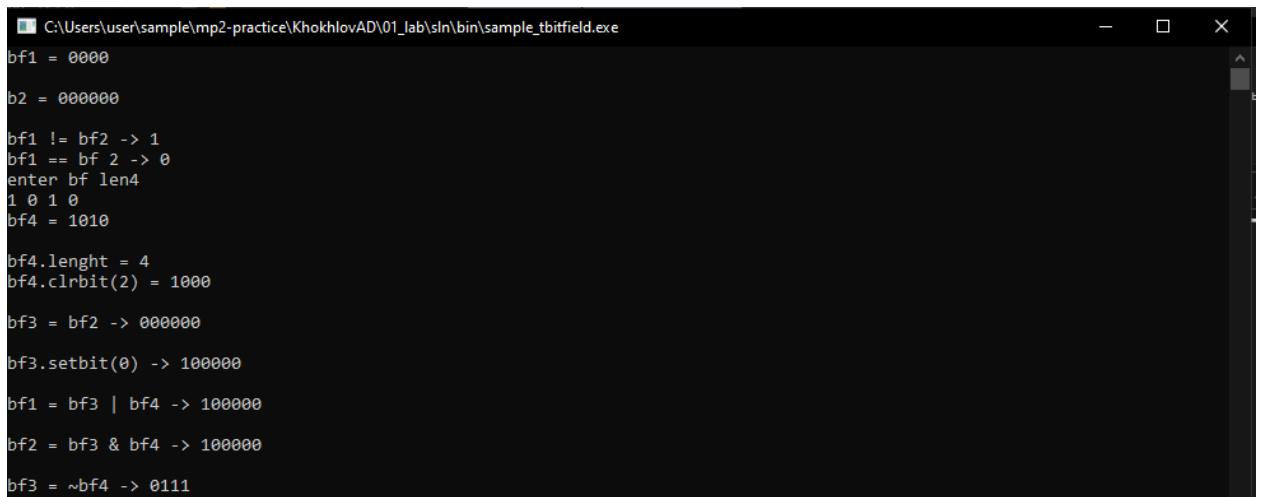
1. Запустите приложение с названием sample_tbitfield.exe. В результате появится окно, показанное ниже (рис. 1).



```
C:\Users\user\mp2-practice\KhokhlovAD\01_lab\sln\bin\sample_tbitfield.exe
bf1 = 0000
b2 = 000000
bf1 != bf2 -> 1
bf1 == bf 2 -> 0
enter bf len4
```

Рис. 1. Основное окно программы

2. Это окно показывает работу основных функций работы с битовыми полями (сравнение на равенство, присваивание, длина, установка/очистка бита, сложение, пересечение, отрицание). Для продолжения введите значение битового поля длины 4. В результате будет выведено (рис. 2). Для выхода нажмите любую клавишу.



```
C:\Users\user\sample\mp2-practice\KhokhlovAD\01_lab\sln\bin\sample_tbitfield.exe
bf1 = 0000
b2 = 000000
bf1 != bf2 -> 1
bf1 == bf 2 -> 0
enter bf len4
1 0 1 0
bf4 = 1010
bf4.lenght = 4
bf4.clrbit(2) = 1000
bf3 = bf2 -> 000000
bf3.setbit(0) -> 100000
bf1 = bf3 | bf4 -> 100000
bf2 = bf3 & bf4 -> 100000
bf3 = ~bf4 -> 0111
```

Рис. 2. Основное окно программы

2.2 Приложение для демонстрации работы множеств

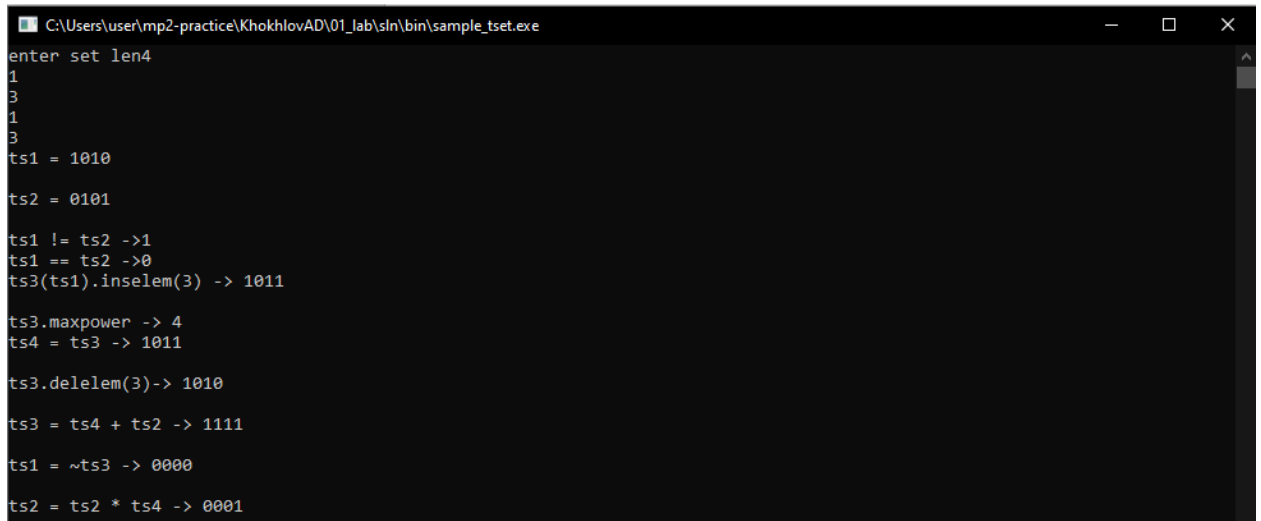
1. Запустите приложение с названием sample_tset.exe. В результате появится окно, показанное ниже (рис. 3).



```
C:\Users\user\mp2-practice\KhokhlovAD\01_lab\sln\bin\sample_tset.exe
enter set len4
```

Рис. 3. Основное окно программы

2. Введите значение множества длины 4. Будет выведено следующее (рис. 4). Это окно показывает работу основных функций работы с множествами (сравнение на равенство, добавление элемента, длина, удаление элемента, сложение, пересечение, отрицание). Для завершения программы нажмите любую клавишу.

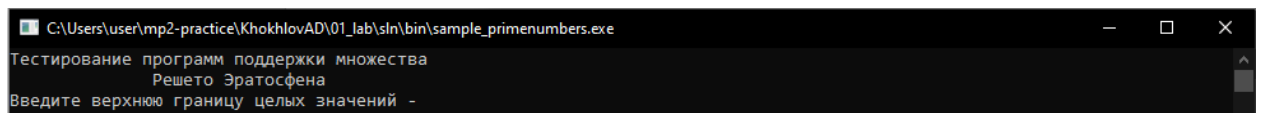


```
C:\Users\user\mp2-practice\KhokhlovAD\01_lab\sln\bin\sample_tset.exe
enter set len4
1
3
1
3
ts1 = 1010
ts2 = 0101
ts1 != ts2 -> 1
ts1 == ts2 -> 0
ts3(ts1).inselem(3) -> 1011
ts3.maxpower -> 4
ts4 = ts3 -> 1011
ts3.delelem(3)-> 1010
ts3 = ts4 + ts2 -> 1111
ts1 = ~ts3 -> 0000
ts2 = ts2 * ts4 -> 0001
```

Рис. 4. Основное окно программы

2.3 «Решето Эратосфена»

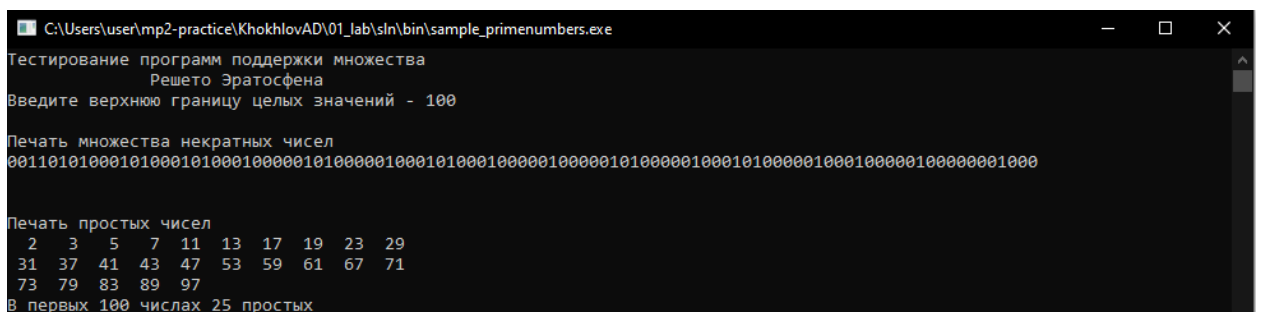
3. Запустите приложение с названием sample_tbitfield.exe. В результате появится окно, показанное ниже (рис. 5).



```
C:\Users\user\mp2-practice\KhokhlovAD\01_lab\sln\bin\sample_primenumbers.exe
Тестирование программ поддержки множества
Решето Эратосфена
Введите верхнюю границу целых значений -
```

Рис. 5. Основное окно программы

4. Введите верхнюю границу целых значений для поиска простых чисел. В результате выведено множество в битовом представлении. Ниже выведен список простых чисел (рис. 6). Для завершения программы нажмите любую клавишу.



```
C:\Users\user\mp2-practice\KhokhlovAD\01_lab\sln\bin\sample_primenumbers.exe
Тестирование программ поддержки множества
Решето Эратосфена
Введите верхнюю границу целых значений - 100
Печать множества некратных чисел
0011010100010100010100010000010100000100000101000001000001010000010001010000010000001000
Печать простых чисел
 2  3  5  7 11 13 17 19 23 29
31 37 41 43 47 53 59 61 67 71
73 79 83 89 97
В первых 100 числах 25 простых
```

Рис. 6. Основное окно программы

3 Руководство программиста

3.1 Описание алгоритмов

3.1.1 Битовые поля

Битовые множества представляют собой набор чисел, где каждый бит является элементом с индексом, соответствующим этому биту. Эти структуры обеспечивают удобный доступ к отдельным битам данных и позволяют создавать объекты с некротной байту длиной, что экономит память благодаря более компактному распределению данных.

Битовые множества реализованы в виде класса с полями, включающими массив беззнаковых целых чисел, каждое размером 32 бита. Также присутствуют максимальное количество битов и количество беззнаковых целых чисел, формирующих битовое множество.

Пример битового множества длиной 5: 10110

Операции, такие как объединение, пересечение, дополнение (отрицание), сравнение, ввод и вывод, поддерживаются для битовых множеств.

Операция объединения:

Возвращает множество, где каждый бит равен 1, если он присутствует хотя бы в одном из объединяемых множеств, и 0 в противном случае.

Пример:

0 1 0 0 1 0

1 0 0 0 1 1

1 1 0 0 1 1

Операция пересечения:

Возвращает множество, где каждый бит равен 1, если он присутствует в каждом из множеств, и 0 в противном случае.

Пример:

0 1 1 0 1 1

1 0 0 0 0 1

0 0 0 0 0 1

Операция дополнения (отрицания):

Возвращает множество, где каждый бит равен 0, если он присутствует в исходном множестве, и 1 в противном случае.

Пример:

0 1 0 0 1 1

1 0 1 1 0 0

Операция ввода битового множества из консоли:

Позволяет ввести битовое множество из консоли, где необходимо ввести 1 или 0 столько раз, сколько длина вводимого битового множества.

Пример:

0 0 0 0 0 0

После ввода:

0 1 1 0 0 1

Операция вывода битового множества в консоль:

Позволяет вывести битовое множество в консоль.

Также возможно установить или сбросить бит в битовом множестве.

Пример:

0 1 1 0 1 1

Результат установки 1 бита:

1 1 1 0 1 1

Пример:

0 1 1 0 1 1

Результат сброса 2 бита:

0 0 1 0 1 1

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

Операции сравнения:

Операция сравнения на равенство возвращает 1, если два битовых множества идентичны, и 0 в противном случае. Операция сравнения на неравенство возвращает 0, если хотя бы два бита не совпадают, и 1 в противном случае.

3.1.2 Множества

Множества представляют собой набор уникальных положительных чисел. В рамках данной лабораторной работы множество реализовано через использование битовых полей, где каждый бит служит элементом, соответствующим индексу бита. Благодаря битовым полям обеспечивается легкий доступ к отдельным битам данных, позволяя формировать объекты с длиной, не кратной байту. Это, в свою очередь, способствует более эффективному использованию памяти.

Множество хранится в виде класса с полями: битовое поле и максимальный элемент множества.

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

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

Операция объединения с множеством:

Возвращает множество, содержащее все уникальные элементы из двух множеств.

Пример:

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

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

Результат объединения множеств $A \cup B$:

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

Операция пересечения с множеством:

Возвращает множество, содержащее все уникальные элементы, которые присутствуют в обоих множествах одновременно.

Пример:

...

$$A = \{2, 3, 5\}$$

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

Результат пересечения множеств $A \cap B$:

$$A \cap B = \{3, 5\}$$

Операция дополнения (отрицания):

Возвращает множество, содержащее элементы, которых нет в исходном множестве и не превышают максимального элемента.

Пример:

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

Результат дополнения множества $\sim A$:

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

Операция ввода множества из консоли:

Позволяет ввести множество из консоли. Для этого необходимо ввести элементы, при этом они не могут превышать максимального элемента множества.

Пример:

$A = \{\}$

После ввода:

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

Операция вывода битового поля в консоль:

Позволяет вывести множество в консоль.

Также в множество можно добавить или удалить 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»

Данный алгоритм реализован двумя способами, при помощи классов TSet.

Сначала необходимо ввести целое положительное число, до которого будет производиться поиск. После чего алгоритм заполняет все элементы класса 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);
    void SetBit(const int n);
    void ClrBit(const int n);
    int GetBit(const int n) const
    bool operator==(const TBitField &bf) const;
    bool 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 маски по индексу.

Входные параметры:

n - номер бита.

Выходные параметры:

Маска.

TBitField(int len);

Назначение: конструктор с параметром.

Входные параметры:

len – длина поля.

TBitField(const TBitField &bf);

Назначение: конструктор копирования.

Входные параметры:

bf – ссылка на копируемое поле.

void SetBit(const int n);

Назначение: установка бита.

Входные параметры:

n – индекс бита.

void ClrBit(const int n);

Назначение: очистка бита.

Входные параметры:

n – индекс бита.

int GetBit(const int n) const

Назначение: получение значения бита.

Входные параметры:

n – индекс бита.

Выходные параметры:

Значение бита.

bool operator==(const TBitField &bf) const

Назначение: перегрузка операции сравнения на равенство.

Входные параметры:

bf – ссылка на константное битовое поле.

Выходные параметры:

Результат сравнения.

```
bool operator!=(const TBitField &bf) const;
```

Назначение: перегрузка операции сравнения на неравенство.

Входные параметры:

bf – ссылка на константное битовое поле.

Выходные параметры:

Результат сравнения.

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

Назначение: перегрузка операции присваивания.

Входные параметры:

bf – ссылка на константное битовое поле.

Выходные параметры:

Ссылка на битовое поле.

```
TBitField operator|(const TBitField &bf);
```

Назначение: перегрузка операции “или”.

Входные параметры:

bf – ссылка на константное битовое поле.

Выходные параметры:

Битовое поле.

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

Назначение: перегрузка операции “и”.

Входные параметры:

bf – ссылка на константное битовое поле.

Выходные параметры:

Битовое поле.

```
TBitField operator~(void);
```

Назначение: перегрузка операции отрицания.

Выходные параметры:

Битовое поле.

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

Назначение: перегрузка потокового ввода.

Выходные параметры:

istr – ссылка на поток ввода.

bf – ссылка на константное битовое поле.

Выходные параметры:

Ссылка на поток ввода.

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

Назначение: перегрузка потокового вывода.

Входные параметры:

ostr – ссылка на поток вывода.

bf – ссылка на константное битовое поле.

Выходные параметры:

Ссылка на поток вывода.

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

Поля:

MaxPower – длина множества.

BitField – битовое поле.

Методы:

```
TSet(int mp);
```

Назначение: конструктор с параметром.

Входные параметры:

mp – длина множества.

TSet(const TSet &s);

Назначение: конструктор копирования.

Входные параметры:

s – ссылка на константное множество.

TSet(const TBitField &bf);

Назначение: конструктор преобразования типа.

Входные параметры:

bf – ссылка на константное битовое поле.

operator TBitField();

Назначение: преобразование типа к битовому полю.

int GetMaxPower(void) const;

Назначение: получение максимальной мощности множества.

Выходные параметры:

Максимальная мощность множества.

void InsElem(const int Elem);

Назначение: включение элемента в множество.

Входные параметры:

Elem – новый элемент множества.

void DelElem(const int Elem);

Назначение: удаление элемента из множества.

Входные параметры:

Elem – элемент множества.

int IsMember(const int Elem) const;

Назначение: проверка наличия элемента в множестве.

Входные параметры:

Elem – элемент множества.

Выходные параметры:

Результат проверки.

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

Назначение: перегрузка операции сравнения на равенство.

Входные элементы:

s – ссылка на константное множество.

Выходные параметры:

Результат сравнения.

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

Назначение: перегрузка операции сравнения на неравенство.

Входные элементы:

s – ссылка на константное множество.

Выходные параметры:

Результат сравнения.

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

Назначение: перегрузка операции присваивания.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Ссылка на множество.

TSet operator+ (const int Elem) ;

Назначение: перегрузка операции объединения с элементом.

Входные параметры:

Elem – новый элемент множества.

Выходные параметры:

Множество.

TSet operator- (const int Elem) ;

Назначение: перегрузка операции разницы с элементом.

Входные параметры:

Elem – элемент множества.

Выходные параметры:

Множество.

TSet operator+ (const TSet &s) ;

Назначение: перегрузка операции объединения множеств.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Множество.

TSet operator* (const TSet &s);

Назначение: перегрузка операции пересечения множеств.

Входные параметры:

s – ссылка на константное множество.

Выходные параметры:

Множество.

TSet operator~ (void);

Назначение: перегрузка операции дополнения.

Выходные параметры:

Множество.

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

Назначение: перегрузка операции потокового ввода.

Входные параметры:

istr – ссылка на поток ввода.

bf – ссылка на константное множество.

Выходные параметры:

Ссылка на поток ввода.

ostream &operator<<(ostream &ostr, const TSet &bf);

Назначение: перегрузка операции потокового вывода.

Входные параметры:

ostr – ссылка на поток вывода.

bf – ссылка на константное множество.

Выходные параметры:

Ссылка на поток вывода.

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. Сысоев А.В., Алгоритмы и структуры данных, лекция 03, 19 сентября.

Приложения

Приложение А. Реализация класса TBitField

```
#define BITS_IN_ONE_MEM (sizeof(TELEM) * 8)

TBitField::TBitField(int len)
{
    if (len < 0) throw "Negative len";
    BitLen = len;
    MemLen = (BitLen-1) / BITS_IN_ONE_MEM + 1;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; i++) pMem[i] = 0;
}

TBitField::TBitField(const TBitField& bf)
{
    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
{
    return (n / BITS_IN_ONE_MEM);
}

TELEM TBitField::GetMemMask(const int n) const
{
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    return (1 << (n % BITS_IN_ONE_MEM));
}

int TBitField::GetLength(void) const
{
    return BitLen;
}

void TBitField::SetBit(const int n)
{
    if ((n >= BitLen) || (n < 0)) throw "Negative n";
    pMem[GetMemIndex(n)] = (GetMemMask(n) | pMem[GetMemIndex(n)]);
}

void TBitField::ClrBit(const int n)
{
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    pMem[GetMemIndex(n)] &= (~GetMemMask(n));
}

int TBitField::GetBit(const int n) const
{
    if ((n > BitLen) || (n < 0)) throw "Negative n";
    if (pMem[GetMemIndex(n)] & GetMemMask(n)) return 1; else return 0;
}
```

```

const TBitField& TBitField::operator=(const TBitField& bf)
{
    if (*this == bf)
        return *this;
    delete[] pMem;
    this->BitLen = bf.BitLen;
    MemLen = bf.MemLen;
    pMem = new TELEM[MemLen];
    for (int i = 0; i < MemLen; i++) pMem[i] = bf.pMem[i];
    return *this;
}

bool TBitField::operator==(const TBitField& bf) const
{
    if (BitLen != bf.BitLen)
        return 0;
    int k = 0;
    for (int i = 0; i < MemLen; i++)
        if (pMem[i] != bf.pMem[i]) {
            return 0;
        }
    return 1;
}

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

TBitField TBitField::operator|(const TBitField& bf)
{
    int k;
    int z;
    int y;
    if (BitLen > bf.BitLen)
    {
        y = 0; // This is big
        k = BitLen;
        z = bf.BitLen;
    }
    else {
        y = 1; //bf is big
        k = bf.BitLen;
        z = BitLen;
    }
    TBitField a(k);
    for (int i = 0; i <= GetMemIndex(z); i++) a.pMem[i] = bf.pMem[i] | pMem[i];
    for (int i = (GetMemIndex(z) + 1); i < a.MemLen; i++) if (y == 1) a.pMem[i]
= bf.pMem[i]; else a.pMem[i] = pMem[i];
    return a;
}

TBitField TBitField::operator&(const TBitField& bf)
{
    int k;
    int z;
    if (BitLen > bf.BitLen) {
        k = BitLen;
        z = bf.BitLen;
    }
}

```

```

        else
        {
            k = bf.BitLen;
            z = BitLen;
        }

TBitField a(k);
for (int i = 0; i <= GetMemIndex(k); i++) a.pMem[i] = pMem[i] & bf.pMem[i];
return a;
}

TBitField TBitField::operator~(void)
{
    TBitField a(*this);
    for (int i = 0; i < (a.MemLen - 1); i++) a.pMem[i] = ~(a.pMem[i]);
    for (int i = ((a.MemLen - 1) * BITS_IN_ONE_MEM); i < (a.BitLen); i++) {
        if (a.GetBit(i) == 1) a.ClrBit(i);
        else a.SetBit(i);
    }
    return a;
}

istream& operator>>(istream& istr, TBitField& bf)
{
    int tmp;
    for (int i = 0; i < bf.BitLen; i++) {
        istr >> tmp;
        if ((tmp != 0) && (tmp != 1)) {
            throw "The bit cannot take such a value";
        }

        if (tmp == 0) {
            bf.ClrBit(i);
        }
        else {
            bf.SetBit(i);
        }
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TBitField& bf)
{
    for (int i = 0; i < bf.BitLen; i++)
        if (bf.GetBit(i) == 0)
            ostr << 0;
        else ostr << 1;
    ostr << endl;
    return ostr;
}

```

Приложение Б. Реализация класса TSet

```

TSet::TSet(int mp) : BitField(mp)
{
    MaxPower = mp;
}

TSet::TSet(const TSet& s) : BitField(s.BitField)
{
    MaxPower = s.GetMaxPower();
}

```

```

}

TSet::TSet(const TBitField& bf) : BitField(bf)
{
    MaxPower = bf.GetLength();
}

TSet::operator TBitField()
{
    return BitField;
}

int TSet::GetMaxPower(void) const
{
    return MaxPower;
}

int TSet::IsMember(const int Elem) const
{
    if ((Elem < MaxPower) & (Elem >= 0))
        if (BitField.GetBit(Elem) == 1) return 1;
        else return 0; else throw "Negative Elem";
}

void TSet::InsElem(const int Elem)
{
    if ((Elem < MaxPower) & (Elem >= 0))
        BitField.SetBit(Elem); else throw "Negative n";
}

void TSet::DelElem(const int Elem)
{
    if ((Elem < MaxPower) & (Elem >= 0))
        BitField.ClrBit(Elem); else throw "Negative n";
}

const TSet& TSet::operator=(const TSet& s)
{
    if (*this == s)
        return *this;
    MaxPower = s.MaxPower;
    BitField = s.BitField;
    return *this;
}

bool TSet::operator==(const TSet& s) const
{
    return (BitField == s.BitField);
}

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

TSet TSet::operator+(const TSet& s)
{
    TSet a(BitField | s.BitField);
    return a;
}

```

```

TSet TSet::operator+(const int Elem)
{
    TSet a(*this);
    if ((Elem < MaxPower) & (Elem >= 0)) {
        a.InsElem(Elem);
    }
    else throw "Nrgative Elem";
    return a;
}

TSet TSet::operator-(const int Elem)
{
    TSet a(*this);
    if ((Elem < MaxPower) & (Elem >= 0)) {
        a.DelElem(Elem);
    }
    else throw "Nrgative Elem";
    return a;
}

TSet TSet::operator*(const TSet& s)
{
    TSet a(BitField & s.BitField);
    return a;
}

TSet TSet::operator~(void)
{
    TSet a(~BitField);
    return a;
}

istream& operator>>(istream& istr, TSet& s)
{
    for (int i = 0; i < s.MaxPower; i++) {
        s.DelElem(i);
    }
    cout << "Input your set (To finish, enter -1)" << endl;
    int i;

    while (1) {
        istr >> i;
        if (i == -1) {
            return istr;
        }
        if ((i < 0) || (i > s.MaxPower)) {
            throw "OUTOFRANGE";
        }
        s.InsElem(i);
    }
    return istr;
}

ostream& operator<<(ostream& ostr, const TSet& s)
{
    for (int i = 0; i < s.MaxPower; i++)
        if (s.IsMember(i) == 1)
            ostr << i;
    return ostr;
}

```