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

ОТЧЕТ ПО УЧЕБНОЙ ПРАКТИКЕ

«Множества но основе битовых полей»

Выполнил: студент группы 381706-1
Митягина Дарья Сергеевна
_____ Подпись

Научный руководитель:
ассистент каф. МОСТ ИИТММ
_____ Лебедев И.Г

Нижний Новгород
2018.

Оглавление

1. Введение

Цель данной лабораторной работы: разработка структуры данных для хранения множеств с использованием битовых полей.

В процессе разработки программного обеспечения программист определяет не только алгоритм, но и перечень данных, необходимых для решения задачи. В зависимости от типа данных и выполняемых с ними действий программист выбирает наиболее подходящую структуру для хранения и обработки данных.

Понятие структуры данных можно определить как множество элементов данных и множество связей между ними. При описании структуры данных определяется набор возможных действий и порядок доступа к данным.

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. Битовые поля применяются для максимально полной упаковки информации, если не важна скорость доступа к этой информации.

Битовые поля могут быть полезны по разным причинам, а именно:

1. Если память ограничена, то в одном байте можно хранить несколько булевых переменных (принимающих значения ИСТИНА и ЛОЖЬ);

2. Некоторые устройства передают информацию о состоянии, закодированную в байте в одном или нескольких битах;

3. Для некоторых процедур шифрования требуется доступ к отдельным битам внутри байта.

Хотя для решения этих задач можно успешно применять побитовые операции, битовые поля могут придать вашему коду больше упорядоченности (и, возможно, с их помощью удастся достичь большей эффективности).

2. Постановка задачи:

Выполнение работы предполагает решение следующих задач:

1. Реализация класса битового поля TBitField согласно заданному интерфейсу.

2. Реализация класса множества TSet согласно заданному интерфейсу.

3. Обеспечение работоспособности тестов и примера использования.

4. Реализация нескольких простых тестов на базе Google Test.

5. Публикация исходных кодов в личном репозитории на GitHub.

3. Руководство пользователя:

Примером использования классов битового поля и множества служит решение задачи поиска простых чисел с помощью алгоритма "Решето Эратосфена".

При запуске программы происходит следующее:

1. Запрашивается верхняя граница целых значений n ;

2. Далее происходит поиск и подсчет простых чисел ;

3. На экран выводится множество некратных чисел и простые числа от 0 до n.

Затем существует два варианта работы программы:

1) **Использование класса множеств**

a) пользователь задает наибольший элемент множества s ($s > 0$);

b) пользователь вводит набор чисел (от 0 до заданного s);

c) Результат: на экран будет выведено полученное множество и его битовая строка.

2) **Использование Битовых полей**

a) пользователь задает размер битовой строки s;

b) пользователь вводит строку;

c) Результат: на экран будет выведена полученная битовая строка и множество чисел.

4. Руководство программиста

4.1. Описание структуры программы

Созданы два класса: TSet и TBitField.

В лабораторной работе содержатся следующие модули:

- tset

Этот модуль состоит из заголовочного файла tset.h и файла, содержащего его реализацию tset.cpp. tset.h отвечает за определение интерфейса класса TSet.

Реализованы следующие функции: конструкторы, деструктор для класса TSet, методы доступа к отдельным битам, перегружены теоретико-множественные операторы (такие как сравнение, присваивание, объединение, разность, пересечение элементов и т.д.), операторы ввода/вывода.

- tbitfield

Этот модуль состоит из заголовочного файла tbitfield.h и файла, содержащего его реализацию tbitfield.cpp. tbitfield.h отвечает за определение интерфейса класса TBitField.

Реализованы следующие функции: конструкторы, деструктор для класса TBitField, методы доступа к отдельным битам, перегружены битовые операторы (такие как !=, ==, |, &, ~), операторы ввода/вывода, прописаны методы для получение индекса элемента, получение маски.

- Пример использования

Пример использования содержится в файле sample_prime_numbers.cpp.

В нем рассмотрен алгоритм поиска простых чисел ("Решето Эратосфена").

К проекту sample_prime_numbers подключается статическая библиотека set, которая содержит оба модуля: tbitfield и tset. Далее приводится пример использования алгоритма "Решето Эратосфена" для класса TBitField. Для TSet действия аналогичны.

- Тесты

В test_tbitfield.cpp прописаны 29 тестов, в test_tset.cpp - 25 тестов. Назначение тестов: проверить каждый метод из классов TBitField, TSet.

4.2. Описание структур данных

Класс TSet

Класс TSet является шаблонным классом.

Содержит следующие 2 поля (private-часть):

int maxPower; - Максимальная мощность множества

TBitField bitField; - Битовое поле для хранения характеристического вектора

Public – часть:

1) TSet(int mp);

Конструктор инициализации, где mp – максимальная мощность множества.

2) TSet(const TSet &s); - Конструктор копирования

Принимает ссылку на объект класса TSet.

3) TSet(const TBitField &bf); - Конструктор преобразования типа

Принимает ссылку на объект класс TBitField.

4) operator TBitField(); - Преобразование типа к битовому полю

Метод возвращает битовое поле характеристического вектора.

Доступ к битам:

5) int GetMaxPower(void) const;

Возвращает максимальную мощность множества.

6) void InsElem(const int Elem);

Включает элемент в множество.

7) void DelElem(const int Elem);

Удаляет элемент из множества.

8) int IsMember(const int Elem) const;

Проверяет наличие элемента в множестве.

Теоретико-множественные операции:

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

Сравнение. Принцип работы: принимает ссылку на объект класса TSet, проверяет 2 битовых поля на равенство.

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

Сравнение. Принцип работы: принимает ссылку на объект класса TSet, проверяет 2 битовых поля на неравенство.

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

Присваивание. Принцип работы: принимает ссылку на объект класса TSet, присваивает исходное множество к пришедшему.

12) TSet operator+ (const int Elem);

Объединение с элементом. Принцип работы: принимает Elem и добавляет его к исходному множеству.

13) TSet operator- (const int Elem);

Разность с элементом. Принцип работы: принимает Elem и удаляет его из исходного множества.

14) TSet operator+ (const TSet &s);

Объединение множеств. Принимает сслыку на объект класса TSet, возвращает результирующий объект, представляющий собой объединение двух множеств.

15) TSet operator* (const TSet &s);

Пересечение множеств. Принимает сслыку на объект класса TSet, возвращает результирующий объект, представляющий собой пересечение двух множеств.

16) TSet operator~ (void);

Возвращает дополнение.

Ввод-вывод (дружественные функции):

17) friend istream &operator>>(istream &istr, TSet &bf);
18) friend ostream &operator<<(ostream &ostr, const TSet &bf);

Класс TBitField

Класс TBitField является шаблонным классом.

Содержит следующие 2 метода и 3 поля (private-часть):

int bitLen; - Длина битового поля - макс. к-во битов

TELEM *pMem; - Память для представления битового пол

int memLen; - Количество элементов Мем для представления битового поля

int GetMemIndex(const int n) const; Индекс в pMем для бита n

TELEM GetMemMask (const int n) const; - Битовая маска для бита n

Public – часть:

1) TBitField(int len);

Конструктор инициализации.

2) TBitField(const TBitField &bf);

Конструктор копирования.

3) ~TBitField();

Деструктор.

Доступ к битам:

4) int GetLength(void) const;

Получает длину (количество битов).

5) void SetBit(const int n);

Устанавливает бит (n-ый).

6) void ClrBit(const int n);

Очищает бит (n-ый).

7) int GetBit(const int n) const;

Получает значение бита (n-ого).

Битовые операции:

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

Сравнение. Принцип работы: принимает ссылку на объект класса TBitField и проверяет его на равенство с исходным.

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

Сравнение. Принцип работы: принимает ссылку на объект класса TBitField и проверяет его на неравенство с исходным.

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

Присваивание. Принцип работы: принимает ссылку на объект класса TBitField и приравнивает исходный объект к пришедшему.

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

Операция "или". Принцип работы: принимает ссылку на объект класса TBitField, возвращает результирующий объект этого же класса, полученный после применения к двум битовым полям операции «ИЛИ».

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

Операция "И". Принцип работы: принимает ссылку на объект класса TBitField, возвращает результирующий объект этого же класса, полученный после применения к двум битовым полям операции «И».

13) TBitField operator~(void);

Отрицание. Принцип работы: возвращает объект класса TBitField, полученный с помощью использования побитового отрицания для исходного объекта.

Ввод-вывод (дружественные функции):

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

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

4.3. Описание алгоритмов

В данном разделе не будут рассматриваться тривиальные методы.

TBitField:

Рассмотрим некоторую характерную черту битовых полей:

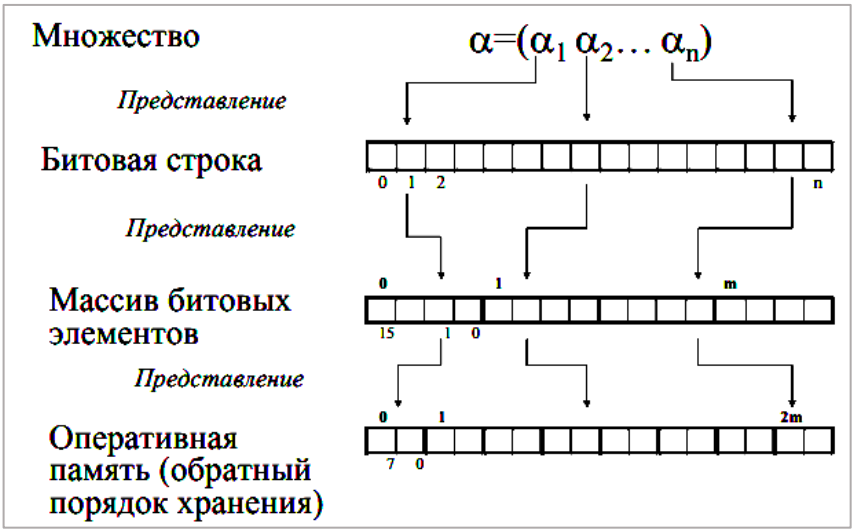

Рис. 1 – Представление битового поля

Нумерация бит в битовой строке – слева направо.

Нумерация элементов в массиве – слева направо, биты элемента – справа налево.

Байты двухбайтового элемента располагаются в ОП в обратном порядке (сначала байт с младшими битами, затем байт со старшими битами) – поддержка отображения на аппаратном уровне.

Для упрощения работы с некоторыми методами класса TBitField (такими как получить, очистить, установить бит) были реализованы вспомогательные функции. Поговорим о них подробнее:

1. GetMemMask – получение битовой маски для n-ого бита.

Маска - число, у которого в соответствующих битовому полю разрядах установлены единицы, а в остальных разрядах — нули.

Необходимая для вычисления маски формула (1.1)

$$1 \ll ((n - 1) \% (sizeof(TELEM) * 8)) \qquad (1.1)$$

2. *GetMemIndex* – получение индекса элемента elem для n-ого бита.

Таким образом, elem - целая часть от деления n на количество битов в одном элементе массива.

Для вычисления индекса требуется формула (1.2).

$$\frac{n}{\text{sizeof(TELEM)} * 8} \qquad (1.2)$$

где TELEM = int, если используем целочисленный массив.

Эти методы можно использовать в дальнейшем для реализации следующих функций:

1. SetBit(const int n) – устанавливает на n-ую позицию.

Для установки битов выполняют логическое сложение (операцию «битовое ИЛИ») числа с маской, у которой в позициях, соответствующих битовому полю, установлены единицы.

```
int i = GetMemIndex(n);

pMem[i] = pMem[i] | GetMemMask(n);
```

2. ClrBit(const int n) – очищает n-ый бит.

Для установки в один или несколько битов нулей число операцией «битовое И» умножают на маску, у которой в позициях, соответствующих битовому полю, установлены нулевые биты.

```
int i = GetMemIndex(n);

pMem[i] = pMem[i] & ~GetMemMask(n);
```

3. GetBit(const int n) – получает значение n-ого бита.

Для получения значеня бита умножают «битовую маску» на число с помощью операции «битовое И».

```
int i = GetMemIndex(n);

return (pMem[i] & GetMemMask(n));
```

На этом заслуживающие отдельного рассмотрения методы закончились.

5. Заключение

В процессе работы над этой лабораторной работой мне удалось понять принципы использования битовых полей.

Проведен *анализ задачи*:

• Понятие множества

• Операции над элементами

• Теоретико-множественные операции

Осуществлено *проектирование*:

• Конкретизация (допущения и ограничения)

• Понятие характеристического вектора

• Представление вектора в виде битовой строки

• Формирование битовой строки в виде массива

• Битовой формат элемента массива

• Выделение базового класса для реализации битовых строк

Освоена техника составления *тестов* путем самостоятельного составления нескольких из них на базе GT.

Результат: завершена разработка структуры данных для хранения множеств при помощи битовых полей. Получены новые знания и навыки.

6. Литература

Интернет-ресурсы:

1. Страница в Википедии, посвященная тема битовых полей: [https://ru.wikipedia.org/wiki/%D0%91%D0%B8%D1%82%D0%BE%D0%B2%D0%BE%D0%B5_%D0%BF%D0%BE%D0%BB%D0%B5]

2.C++ International Standard

[http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3690.pdf]

Книги:

3. Брайан В. Керниган, Деннис М. Ритчи, Язык программирования Си. Издание 3-е

4. Гергель В.П. Методические материалы по курсу «Методы программирования 2», 2015.