ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ "НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ НИЖЕГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМ. Н.И. ЛОБАЧЕВСКОГО"

Отчёт по лабораторной работе №1 по учебной дисциплине «Алгоритмы и структуры данных»

Выполнил:

Студент Щербаков Н.А.

группы 3824Б1ФИ2

Нижний Новгород – 2025 г.

1.Постановка задачи

Цель данной работы - разработка структуры данных для хранения множеств (TSet) с использованием битовых полей (TBitField), а также освоение фрэймворка для разработки автоматических тестов Google Test.

2.Описание программной реализации

2.1.Класс TBitField

Класс предназначен для представления и управления битовым полем — структуры данных, хранящей последовательность бит (0 и 1) с возможностью установки, очистки, чтения и выполнения логических операций над ними.

Поля класса:

- **int** BitLen - длина битового поля, то есть максимальное количество битов.
- **TELEM** *pMem - память для представления битового поля.
- **int** MemLen - количество элементов в pMem для представления битового поля.

Методы класса:

1) TBitField(**int** len)

Функционал: Конструктор. Создаёт битовое поле. Инициализирует битовое поле длиной len и устанавливает все биты в 0.

Параметры: len - длина битового поля (в битах).

Сложность: O(n), где n - количество элементов в массиве pMem.

2) TBitField(**const TBitField&** bf)

Функционал: Конструктор копирования. Создаёт копию существующего битового поля bf.

Параметры: bf - копируемый объект класса TBitField.

Сложность: O(bf.MemLen), так как копируем каждый элемент массива bf.pMem в массив pMem в цикле от 0 до bf.MemLen.

3) ~TBitField()

Функционал: Деструктор. Освобождает память, выделенную под битовое поле.

4) **int** GetMemIndex(**const int** n) **const**

Функционал: Возвращает индекс элемента массива pMem, содержащего бит с номером n.

Параметры: n - номер бита в битовом поле.

Возвращаемое значение: int, индекс элемента массива pMem, содержащего бит с номером n.

5) **TELEM** GetMemMask(**const int** n) **const**

Функционал: Создаёт маску для доступа к конкретному биту с номером n.

Параметры: n - номер нужного бита.

Возвращаемое значение: TELEM, битовая маска.

6) **int** GetLength(**void**) **const**

Функционал: Возвращает длину битового поля.

Возвращаемое значение: int, максимальное количество бит в битовом поле поле.

7) **void** SetBit(**const int** n)

Функционал: Устанавливает бит с номером n в 1.

Параметры: n - номер нужного бита.

8) **void** ClrBit(**const int** n)

Функционал: Устанавливает бит с номером n в 0.

Параметры: n - номер нужного бита.

9) **int** GetBit(**const int** n) **const**

Функционал: Возвращает значение бита с номером n (0 или 1).

Параметры: n - номер нужного бита.

Возвращаемое значение: int, значение бита с номером n (0 или 1).

10) **int** operator==(**const TBitField&** bf) **const**

Функционал: Сравнивает два битовых поля на равенство.

Параметры: bf - битовое поле для сравнения.

Возвращаемое значение: int, 1 - битовые поля равны равны, 0 - битовые поля не равны.

Сложность: O(MemLen), так как сравниваем каждый элемент массива pMem с каждым элементом массива bf.pMem в цикле от 0 до MemLen, где MemLen равен bf.MemLen.

11) **int** operator!=(**const TBitField&** bf) **const**

Функционал: Сравнивает два битовых поля на неравенство.

Параметры: bf - битовое поле для сравнения.

Возвращаемое значение: int, 0 - битовые поля равны равны, 1 - битовые поля не равны.

Сложность: O(MemLen), так как вызываем перегруженный оператор сравнения битовых полей (*this == bf), у которого данная сложность.

12) **TBitField&** operator=(**const TBitField&** bf)

Функционал: Оператор присваивания. Копирует данные из одного битового поля в другое.

Параметры: bf - источник данных для копирования.

Возвращаемое значение: TBitField&, ссылка на текущий объект.

Сложность: O(bf.MemLen), так как копируем каждый элемент массива bf.pMem в массив pMem в цикле от 0 до bf.MemLen.

13) **TBitField** operator|(**const TBitField&** bf)

Функционал: Побитовое ИЛИ (OR) двух битовых полей.

Параметры: bf - второе битовое поле.

Возвращаемое значение: TBitField, новый объект класса TBitField - результат побитового ИЛИ (OR).

Сложность: O(max{MemLen, bf.MemLen}), так как выбирается битовое поле с максимальным количеством битов, создаётся объект класса битового поля res, у которого res.MemLen = max{MemLen, bf.MemLen}. Затем каждый элемент массива res.pMem инициализируется в двух циклах, которые в сумме проходят через все элементы от 0 до max{MemLen, bf.MemLen}.

14) **TBitField** operator&(**const TBitField&** bf)

Функционал: Побитовое И (AND) двух битовых полей.

Параметры: bf - второе битовое поле.

Возвращаемое значение: TBitField, новый объект класса TBitField – результат побитового И (AND).

Сложность: O(max{MemLen, bf.MemLen}), так как выбирается битовое поле с максимальным количеством битов, создаётся объект класса битового поля res, у которого res.MemLen = max{MemLen, bf.MemLen}. Затем каждый элемент массива res.pMem инициализируется в двух циклах, которые в сумме проходят через все элементы от 0 до max{MemLen, bf.MemLen}.

15) **TBitField** operator~(**void**)

Функционал: Побитовое отрицание (NOT)битового поля. Инвертирует все биты битового поля.

Возвращаемое значение: TBitField, новый объект класса TBitField с инвертированными битами - результат побитового отрицания (NOT).

Сложность: O(MemLen), так как создаём объект класса битового поля res, которого res.MemLen = MemLen, применяем побитовое отрицание к каждому элементу массива res.pMem в цикле от 0 до MemLen.

16) **std::istream&** operator>>(**std::istream&** istr, **TBitField&** bf)

Функционал: Оператор ввода. Считывает битовое поле из потока в виде строки из 0 и 1.

Параметры: istr - входной поток, bf - объект для записи битового поля из потока.

Возвращаемое значение: std::istream&, ссылка на входной поток.

Сложность: O(bf.BitLen), так как выполняется цикл от 0 до bf.BitLen, записывая биты в битовое поле.

17) **std::ostream&** operator<<(**std::ostream&** ostr, **const TBitField&** bf)

Функционал: Оператор вывода. Выводит битовое поле в поток в виде строки 0 и 1.

Параметры: ostr - выходной поток, bf - выводимое битовое поле.

Возвращаемое значение: std::ostream&, ссылка на выходной поток.

Сложность: O(bf.BitLen), так как выполняется цикл вывода битового поля от 0 до bf.BitLen.

2.2.Класс TSet

Класс TSet реализует конечное множество на основе характеристического вектора, представленного классом TBitField. Каждый элемент множества соответствует одному биту в битовом поле: 1 — элемент присутствует, 0 — отсутствует. Класс предоставляет операции вставки/удаления элементов, тестирования принадлежности, а также стандартные теоретико-множественные операции (объединение, пересечение, дополнение).

Поля класса:

- **int** MaxPower – максимальная мощность множества (размер универса).
- **TBitField** BitField - битовое поле, хранящее характеристический вектор множества.

Методы класса:

1) TSet(**int** mp)

Функционал: Конструктор. Инициализирует множество с максимальной мощностью mp. Битовое поле BitField будет иметь длину mp.

Параметры: mp - максимальная мощность множества (количество возможных элементов).

Сложность: O(n) - за счёт инициализации битового поля BitField, где n – количество элементов в массиве BitField.pMem.

2) TSet(**const TSet&** s)

Функционал: Конструктор копирования. Создаёт новое множество - копию множества s (включая копию внутреннего битового поля).

Параметры: s - исходное множество (откуда идёт копирование).

Сложность: O(s.BitField.MemLen) - так как копируем каждый элемент массива s.BitField.pMem в массив BitField.pMem в цикле от 0 до s.BitField.MemLen (конструктор копирования класса TBitField).

3) TSet(**const TBitField&** bf)

Функционал: Конструктор преобразования типа. Создаёт множество на основе переданного битового поля TBitField. MaxPower устанавливается равным длине битового поля bf.

Параметры: bf - переданное битовое поле TBitField.

Сложность: O(bf.MemLen) - вызывается конструктор копирования класса TBitField для копирования битового поля: цикл от 0 до bf.MemLen.

4) operator TBitField()

Функционал: Преобразование множества к типу TBitField битового поля. Возвращает битовое поле множества.

Возвращаемое значение: TBitField, битовое поле множества - BitField.

5) **int** GetMaxPower(**void**) **const**

Функционал: Возвращает максимальную мощность множества (размер универса).

Возвращаемое значение: int, значение MaxPower.

6) **void** InsElem(**const int** Elem)

Функционал: Включает элемент Elem в множество, устанавливая соответствующий бит в 1. Проверка допустимости Elem делегируется BitField.SetBit(), который бросает исключение при некорректном Elem.

Параметры: Elem — индекс добавляемого элемента (от 0 до MaxPower-1).

7) **void** DelElem(**const int** Elem)

Функционал: Удаляет элемент Elem из множества, устанавливая соответствующий бит в 0. Проверка допустимости Elem делегируется BitField.ClrBit(), который бросает исключение при некорректном Elem.

Параметры: Elem - индекс удаляемого элемента (от 0 до MaxPower-1).

8) **int** IsMember(**const int** Elem) **const**

Функционал: Проверяет принадлежность элемента Elem множеству. Проверка допустимости Elem делегируется BitField.GetBit(), который бросает исключение при некорректном Elem.

Параметры: Elem - проверяемый индекс.

Возвращаемое значение: int, вернёт 1 - если элемент присутствует в множестве, вернёт 0 - если элемент отсутствует в множестве.

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

Функционал: Сравнивает множества на равенство. Реализация полагается на сравнение внутренних битовых полей множеств – BitField и s.BitField.

Параметры: s - второе сравниваемое множество.

Возвращаемое значение: int, вернёт 1 - если множества равны, вернёт 0 - иначе.

Сложность: O(BitField.MemLen) - так как вызывается перегруженный оператор сравнения битовых полей BitField и s.BitField (та же сложность).

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

Функционал: Проверяет неравенство множеств (инверсия результата перегруженного operator== для множеств).

Параметры: s - второе сравниваемое множество.

Возвращаемое значение: int, вернёт 1 - если множества не равны, вернёт 0 - иначе.

Сложность: O(BitField.MemLen) – так как вызывается перегруженный оператор сравнения для множеств, который имеет данную сложность.

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

Функционал: Оператор присваивания. Копирует данные одного множества в другое.

Параметры: s - множество источник присваивания.

Возвращаемое значение: TSet&, ссылка на текущий объект.

Сложность: O(s.BitField.MemLen) - вызывается перегруженный оператор присвоения битовых полей, который имеет данную сложность.

12) **TSet** operator+(**const int** Elem)

Функционал: Возвращает новое множество, равное текущему с добавленным элементом Elem (не изменяет исходное множество). Создаётся копия текущего множества, затем вызывается InsElem(Elem).

Параметры: Elem - добавляемый элемент.

Возвращаемое значение: TSet, результат объединения с элементом – новое множество TSet.

Сложность: O(BitField.MemLen) - вызывается конструктор копирования множества, далее O(1) - добавление элемента в множество (InsElem(Elem)).

13) **TSet** operator-(**const int** Elem)

Функционал: Возвращает новое множество, равное текущему без элемента Elem (не изменяет исходное множество). Создаётся копия текущего множества и вызывается DelElem(Elem).

Параметры: Elem - удаляемый элемент.

Возвращаемое значение: TSet, результат удаления элемента - новое множество TSet.

Сложность: O(BitField.MemLen) - вызывается конструктор копирования множества, далее O(1) - удаление элемента из множество (DelElem(Elem)).

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

Функционал: Объединение множеств (не изменяет исходное множество). Определено только для множеств одного универса (одинаковой MaxPower). Проверяется совпадение универса, далее выполняется побитовое ИЛИ (OR) внутренних битовых полей BitField и s.BitField. Не затрагивает самый младший бит битовых полей.

Параметры: s - второе множество.

Возвращаемое значение: TSet, результат объединения - новое множество TSet.

Сложность: O(n + max{BitField.MemLen, s.BitField.MemLen}) - вызывается конструктор для создания результирующего множества (O(n)) и затем вызывается перегруженный оператор побитового ИЛИ для битовых полей (O(max{BitField.MemLen, s.BitField.MemLen})).

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

Функционал: Пересечение множеств (не изменяет исходное множество). Определено только для множеств одного универса (одинаковой MaxPower). Проверяется совпадение универса, далее выполняется побитовое И (AND) внутренних битовых полей BitField и s.BitField. Не затрагивает самый младший бит битовых полей.

Параметры: s - второе множество.

Возвращаемое значение: TSet, результат пересечения - новое множество TSet.

Сложность: O(n + max{BitField.MemLen, s.BitField.MemLen}) - вызывается конструктор для создания результирующего множества (O(n)) и затем вызывается перегруженный оператор побитового И для битовых полей (O(max{BitField.MemLen, s.BitField.MemLen})).

16) **TSet** operator~(**void**)

Функционал: Дополнение множества относительно данного универса (не изменяет исходное множество). Меняет присутствующие элементы на отсутствующие и наоборот (0 на 1, 1 на 0). Реализовано через побитовое отрицание (NOT) внутреннего битового поля BitField. Не затрагивает самый младший бит битового поля.

Возвращаемое значение: TSet, дополнение текущего множества – новое множество TSet.

Сложность: O(n + BitField.MemLen) – вызывается конструктор для создания результирующего множества (O(n)) и затем вызывается перегруженный оператор побитового отрицания (NOT) для битовых полей, который имеет сложность O(BitField.MemLen).

17) **std::istream&** operator>>(**std::istream&** istr, **TSet&** bf)

Функционал: Ввод множества формата: { i1 i2 i3 … } - список индексов элементов множества через пробел. Пустое множество: {}. Очищает множество, затем считывает содержимое между фигурными скобками поочерёдно (до пробелов), добавляет поочерёдно индексы элементов множества в буфер (std::string), преобразует буфер к целому числу - индексу и затем добавляет элемент в множество, соответствующий индексу. Выбрасывает исключение при неверном формате или некорректных символах.

Параметры: istr - входной поток, bf - множество для записи из потока.

Возвращаемое значение: std::istream&, ссылка на входной поток.

Сложность: Пусть n – количество символов при вводе (вместе с форматом), k – количество самих индексов при вводе. O((n-2) + k) – (n-2) символов обрабатываем, k индексов вставляем с помощью InsElem() (Сложность InsElem() – O(1), а всего элементов на вставку – k => O(k)). Без учёта сложности методов std::string.

18) **std::ostream&** operator<<(**std::ostream&** ostr, **const TSet&** bf)

Функционал: Вывод множества в формате: { i1 i2 i3 … } - пробел после первой фигурной скобки при наличии элементов, то есть при их отсутствии - {}. Итерирует индексы от 0 до MaxPower-1 и печатает те, для которых установлен бит в битовом поле bf.BitField.

Параметры: ostr - выходной поток, bf - выводимое множество.

Возвращаемое значение: std::ostream&, ссылка на выходной поток.

Сложность: O(bf.MaxPower + k) - требуется проверить каждый возможный индекс в универсе циклом (O(bf.MaxPower)), где k - количество элементов в множестве bf (bf.BitField.GetBit() - O(1)) => O(k)).

3.Краткие комментарии к тестам

Для проверки правильности работы классов TBitField и TSet были созданы тесты с использованием фреймворка Google Test, охватывающие основные операции, граничные ситуации и обработку ошибок.

3.1.Тесты класса TBitField

- **CreateBitFieldNoThrow** – Проверяет корректное создание объектов TBitField с допустимой длиной. Тест на то, что при инициализации битового поля с положительным размером и при создании копии существующего объекта не выбрасываются исключения.

- **CreateBitFieldAnyThrow** – Проверяет реакцию конструктора TBitField на некорректные входные данные. Создание битового поля с отрицательной или нулевой длиной должно приводить к выбросу исключения.
- **AccessToBitNoThrow** – Тестирует работу методов доступа к отдельным битам (SetBit(), ClrBit(), GetBit()). Проверяется, что установка, очистка и получение каждого бита в пределах длины поля выполняются корректно и без исключений. Дополнительно подтверждается, что после установки и очистки бит возвращается ожидаемое значение (1 или 0).
- **AccessToBitAnyThrow** – Проверяет, что при попытке обращения к биту с недопустимым индексом (меньше 0, больше длины битового поля) методы SetBit(), ClrBit() и GetBit() выбрасывают исключения.
- **BitOperationsCompareSameBitLen** – Проверяет корректность операторов сравнения (== и !=) для битовых полей одинаковой длины. Сравниваются поля, в которых совпадает или различается положение установленных битов. Также ожидается, что одинаковые поля считаются равными, а различающиеся – неравными.
- **BitOperationsCompareOtherBitLen** – Проверяет корректность сравнения битовых полей разной длины. Битовые поля, различающиеся по размеру, должны считаться неравными. После присваивания (bf2 = bf1) битовые поля становятся равными, что также проверяется в тесте.
- **BitFieldOperationsAndOrNotSameBitLen** – Тестирует работу побитовых операций |, & и ~ при одинаковой длине битовых полей. Проверяется:
  - Операция | (побитовое ИЛИ) корректно объединяет установленные биты,
  - Операция & (побитовое И) корректно пересекает установленные биты,
  - Операция ~ (побитовое отрицание) корректно инвертирует все биты.

  Сравниваются результаты с ожидаемыми эталонными битовыми полями.

- **BitFieldOperationsAndOrNotOtherBitLen** – Проверяет корректность выполнения побитовых операций | и & между битовыми полями разной длины. Тест на то, что при объединении или пересечении полей разных размеров результат формируется корректно и учитывает различие в длинах.

3.2.Тесты класса TSet

- **CreateSetNoThrow** – Проверяет создание множеств с корректной максимальной мощностью, копирование (конструктор копирования) и инициализацию множества из битового поля (конструктор преобразования типа). Проверяется, что при этих операциях исключения не выбрасываются.
- **CreateSetAnyThrow** – Проверяет реакцию конструктора TSet на некорректные значения мощности множества. Создание множества с отрицательным или нулевым размером должно вызвать исключение.
- **SetConversionToBitField** – Проверяет корректность преобразования множества в битовое поле с помощью оператора TBitField(). Создаётся множество с добавленным элементом, и ожидается, что после преобразования результат совпадает с эквивалентным битовым полем TBitField.

- **AccessToBitNoThrow** – Проверяет работу методов InsElem(), DelElem() и IsMember() при обращении к элементам в допустимых границах. Тест проходит по всем элементам множества, добавляя, удаляя и проверяя их наличие. Проверяется, что после каждой операции состояние множества соответствует ожидаемому состоянию.
- **AccessToBitAnyThrow** – Проверяет, что при попытке добавить, удалить или проверить элемент с индексом вне диапазона (отрицательный или больше MaxPower) методы множества выбрасывают исключения.
- **SetCompareSameMaxPower** – Проверяет корректность работы операторов сравнения (== и !=) для множеств с одинаковой мощностью (MaxPower). Сравниваются множества с одинаковыми и разными элементами. Ожидается, что совпадающие множества признаются равными, а различающиеся – неравными.
- **SetCompareOtherMaxPower** – Проверяет поведение операторов сравнения и присваивания для множеств с разной максимальной мощностью. Множества с разными универсами (MaxPower) считаются неравными. После присваивания одного множества другому они становятся равными (set2 = set1).
- **SetOperationsSameMaxPower** – Проверяет корректность теоретико-множественных операций для множеств из одного универса:
  - “+” с элементом – объединение с элементом,
  - “-“ с элементом – разность с элементом,
  - “+” с множеством – объединение,
  - “*” с множеством – пересечение,
  - “~” множества – дополнение.

  Сравниваются результаты операций с ожидаемыми множествами, в том числе проверяется двойное отрицание (~~set1), которое должно вернуть исходное множество (set1).

- **SetOperationsOtherMaxPower** – Проверяет, что при попытке выполнить операции объединения (“+”) и пересечения (“*”) над множествами из разных универсов выбрасываются исключения, поскольку такие операции недопустимы.