Міністерство освіти і науки України
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені ІВАНА ФРАНКА
Факультет прикладної математики та інформатики

Кафедра програмування

ЛАБОРАТОРНА РОБОТА № 9

Бітова множина

з курсу “Алгоритми та структури даних”

Виконала:

Студентка групи ПМІ-13

Демко Сніжана Іванівна

Львів – 2024

Мета: ознайомитись з бітовою множиною та розробити клас бітової множини, який дозволяє виконувати операції додавання, видалення, перевірки наявності елементів, а також об’єднанню, перетину та різниці множин. Здобути навички роботи з бітовими операціями та їх використанням.

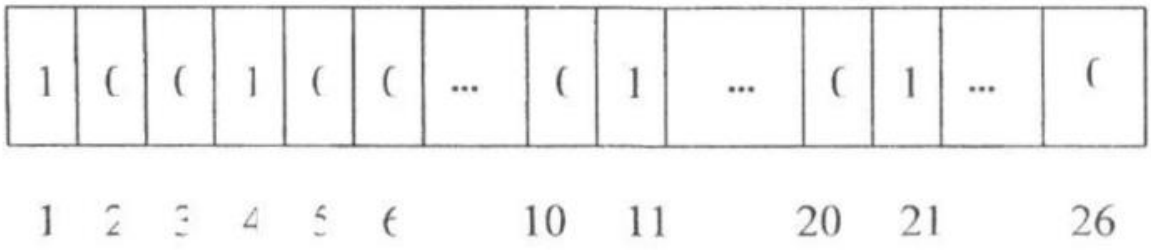
Множина – це невпорядкована сукупність деяких об'єктів, які називають елементами множини і над якими не визначено жодного відношення.

Бітова множина - це структура даних, яка використовує послідовність бітів для представлення множини елементів, де кожен біт відповідає певному елементу множини. Коли біт установлений в 1, це означає, що елемент належить множині, а коли біт установлений в 0, то елемент відсутній у множині.

Бітові множини дозволяють ефективно виконувати операції об'єднання, перетину, різниці та перевірки належності, оскільки ці операції можна виконувати за допомогою операцій порозрядного І, АБО, НЕ та зсуву бітів.

У процесі відображення множин у бітовий вектор усі можливі елементи множини треба перенумерувати, наприклад від 1 до N. У пам’яті виділяється поле з N бітів і належність елемента до множини відзначається встановленням в 1 і-того біт, якщо і-тий можливий елемент присутній у множині. Інакше цей біт встановлюється в 0.

Наприклад, для множини малих букв латинського алфавіту
вектор може складатись з 26 бітів, а множину А можна зобразити так:



(на цій схемі ми бачим, що до множини А належать елементи: ‘d’, ‘u’, ‘k’, ‘a’)

Практичне застосування

Бітові множини знаходять широке застосування у програмуванні:

- В алгоритмах оптимізації для представлення підмножин або різниць між множинами.

- В алгоритмах пошуку для швидкого виявлення відвіданих чи невідвіданих елементів.
- В алгоритмах графів для представлення множин вершин, наприклад, множини сусідів кожної вершини.
- В операціях з пам'яттю для ефективного зберігання бінарних даних.

Реалізація:

Я створила Клас CharSet, який призначений для роботи з бітовими множинами символів. Основна функціональність цього класу полягає у додаванні, видаленні та перевірці належності символів до множини, а також у виконанні операцій об'єднання, перетину та різниці між множинами.

Методи та функції класу.

1. **add(char a):**
    - Цей метод додає символ a до множини.
    - Він приймає символ a як параметр.
    - Внутрішньо використовується бітове представлення множини, де кожен символ відображається на один біт. Після додавання символу, відповідний біт стає 1, що вказує на те, що символ належить множині.
2. **deletee(char a):**
    - Цей метод видаляє символ a з множини.
    - Приймає символ a як параметр.
    - Він змінює відповідний біт, який представляє символ a, на 0, вказуючи на те, що символ більше не належить множині.
3. **test(char a):**
    - Цей метод перевіряє, чи належить символ a множині.
    - Приймає символ a як параметр.
    - Він перевіряє відповідний біт, що відповідає символу a, і повертає true, якщо він встановлений (1), і false в іншому випадку.
4. **association(CharSet& set):**
    - Цей метод повертає об'єднання поточної множини з множиною set.
    - Приймає іншу множину set як параметр.
    - Для кожного символу в множині, метод виконує логічне "або" з відповідними бітами символів обох множин, щоб отримати об'єднання.
5. **intersection(CharSet& set):**

- Цей метод повертає перетин поточної множини з множиною set.
- Приймає іншу множину set як параметр.
- Для кожного символу в множині, метод виконує логічне "і" з відповідними бітами символів обох множин, щоб отримати перетин.

6. **difference(CharSet& set):**
   - Цей метод повертає різницю поточної множини з множиною set.
   - Приймає іншу множину set як параметр.
   - Для кожного символу в множині, метод виконує логічне "і" НЕ з відповідними бітами символів обох множин, щоб отримати різницю.
7. **print():**
   - Цей метод виводить множину на екран.
   - Він виводить всі символи, які належать множині, разом з їх бітовим представленням.

Результати роботи програмної реалізації бітової множини:

Приклад №1

```
s1: { a b c } {00000000000000000000000111}
s2: { b c d } {00000000000000000000001110}
Is 's' in s1? No
Is 'z' in s1? No
Added 's' to s1
Added 'n' to s1
Is 's' in s1? Yes
Deleted 'c' from s1
Updated s1: { a b n s } {00000001000010000000000011}
Union of s1 and s2: { a b c d n s } {00000001000010000000001111}
Intersection of s1 and s2: { b } {00000000000000000000000010}
Difference of s1 and s2: { a n s } {00000001000010000000000001}
```

Приклад №2

```
s1: { } {00000000000000000000000000}
s2: { x y z } {11100000000000000000000000}
Is 's' in s1? No
Is 'z' in s1? No
Added 's' to s1
Added 'n' to s1
Is 's' in s1? Yes
Deleted 'c' from s1
Updated s1: { n s } {00000001000010000000000000}
Union of s1 and s2: { n s x y z } {11100001000010000000000000}
Intersection of s1 and s2: { } {00000000000000000000000000}
Difference of s1 and s2: { n s } {00000001000010000000000000}
```

Тестування

Для забезпечення коректності роботи класу CharSet, був використаний Google Test Framework. Нижче подано опис проведених тестів:

1. **Тест додавання та видалення елементів:**

Цей тест перевіряє правильність додавання та видалення символів до/з множини.

```
TEST(CharSetTest, AddTest) {
    CharSet set;
    set.add('a');
    ASSERT_TRUE(set.test('a'));
}

TEST(CharSetTest, DeleteTest) {
    CharSet set("abc");
    set.deletee('a');
    ASSERT_FALSE(set.test('a'));
}
```

2. **Тест перевірки належності:**

Цей тест перевіряє коректність функції test(), яка визначає належність символу до множини.

```
TEST(CharSetTest, TestTest) {
    CharSet set("abc");
    ASSERT_TRUE(set.test('a'));
    ASSERT_TRUE(set.test('b'));
    ASSERT_TRUE(set.test('c'));
    ASSERT_FALSE(set.test('d'));
}
```

3. **Тест операції об'єднання:**

Цей тест перевіряє правильність обчислення об'єднання двох множин.

```
TEST(CharSetTest, AssociationTest) {
    CharSet set1("abc");
    CharSet set2("bcd");
    CharSet association = set1.association(set2);
    ASSERT_TRUE(association.test('a'));
    ASSERT_TRUE(association.test('b'));
    ASSERT_TRUE(association.test('c'));
    ASSERT_TRUE(association.test('d'));
    ASSERT_FALSE(association.test('e'));
}
```

4. **Тест операції перетину:**

Цей тест перевіряє правильність обчислення перетину двох множин.

```
TEST(CharSetTest, IntersectionTest) {
    CharSet set1("abc");
    CharSet set2("bcd");
    CharSet intersection = set1.intersection(set2);
    ASSERT_TRUE(intersection.test('b'));
    ASSERT_FALSE(intersection.test('a'));
    ASSERT_TRUE(intersection.test('c'));
}
```

5. **Тест операції різниці:**
   Цей тест перевіряє коректність обчислення різниці двох множин.

```
TEST(CharSetTest, DifferenceTest) {
    CharSet set1("abc");
    CharSet set2("bcd");
    CharSet difference = set1.difference(set2);
    ASSERT_TRUE(difference.test('a'));
    ASSERT_FALSE(difference.test('b'));
    ASSERT_FALSE(difference.test('c'));
}
```

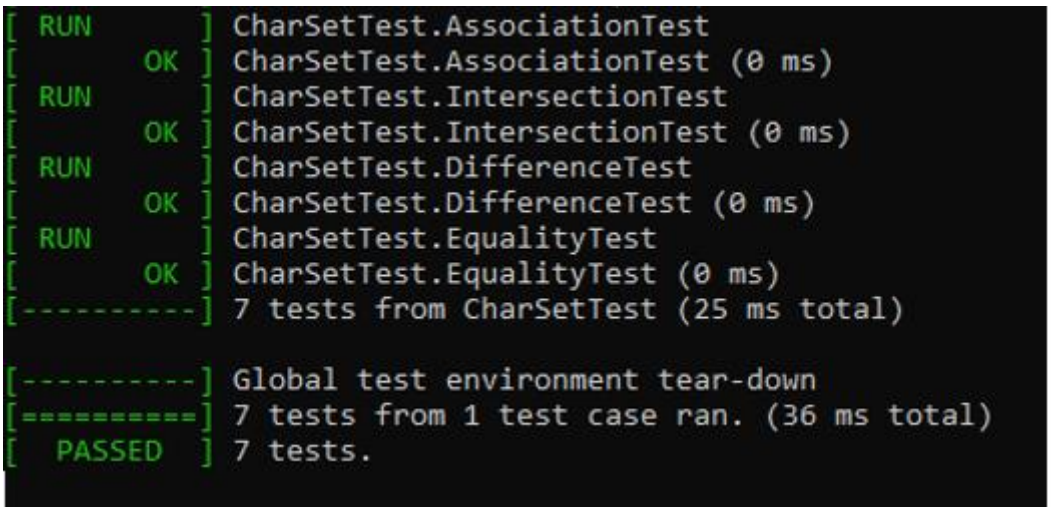
6. **Тест порівняння множин:**
   Цей тест перевіряє правильність порівняння двох множин на рівність.

```
TEST(CharSetTest, EqualityTest) {
    CharSet set1("abc");
    CharSet set2("abc");
    ASSERT_TRUE(set1 == set2);

    CharSet set3("bcd");
    ASSERT_FALSE(set1 == set3);
}
```

Результати тестів:

```
[==========] Running 7 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 7 tests from CharSetTest
[ RUN      ] CharSetTest.AddTest
[       OK ] CharSetTest.AddTest (1 ms)
[ RUN      ] CharSetTest.DeleteTest
[       OK ] CharSetTest.DeleteTest (0 ms)
[ RUN      ] CharSetTest.TestTest
[       OK ] CharSetTest.TestTest (0 ms)
```



Висновок: в ході виконання лабораторної роботи було успішно реалізовано клас для роботи з бітовими множинами символів. Були написані функції, які дозволяють виконувати операції додавання, видалення, перевірки наявності елементів, а також зробити об’єднання, перетин та різницю множин. Також було проведено тестування реалізованих функцій за допомогою Google Test Framework, яке підтвердило коректність їх роботи. Цей клас може бути корисним для різних задач, що вимагають маніпуляцій з множинами символів.