Лабораторна робота 4

Тема: Ефективні алгоритми реалізації АТД Множина.
Ціль: Засвоїти метод реалізації множин на різноманітних структурах даних. Реалізація множин двоїчними векторами, впорядкованими списками, Методи хешування.
Опорні знання: Мови програмування Паскаль, С. Поняття АТД та реалізації АТД. АТД Множина.
Завданння: Ознайомитися з теоретичним матеріалом та виконати завдання, визначені в розділі Хід роботы, підготувати відповіді на конетрольні запитання, оформити протокол виконання роботи.

Хід роботи
Завдання 1
1.Реалізувати АТД Множина на основі двоїчного вектора.

```
class SetUsingBinaryVector:
    def __init__(self, size):
        # Ініціалізація об'єкта - задання розміру та створення бінарного вектора
        self.size = size
        self.vector = [0] * size

    def add_element(self, element):
        # Додавання елементу до множини (встановлення біта в 1)
        self.vector[element] = 1

    def remove_element(self, element):
        # Видалення елементу з множини (встановлення біта в 0)
        self.vector[element] = 0

    def is_member(self, element):
        # Перевірка, чи є елемент частиною множини
        print(f"Is {element} a member of the set? {self.vector[element] == 1}")
        return self.vector[element] == 1

    def union(self, other_set):
        # Об'єднання двох множин (взяття максимального значення для кожного біта)
        result_set = SetUsingBinaryVector(self.size)
        for i in range(self.size):
            result_set.vector[i] = max(self.vector[i], other_set.vector[i])
        print("Union Result:")
        print(result_set.vector)
        return result_set

    def intersection(self, other_set):
        # Перетин двох множин (взяття мінімального значення для кожного біта)
        result_set = SetUsingBinaryVector(self.size)
        for i in range(self.size):
            result_set.vector[i] = min(self.vector[i], other_set.vector[i])
        print("Intersection Result:")
        print(result_set.vector)
        return result_set

    def difference(self, other_set):
        # Різниця двох множин (віднімання одного біта від іншого, з урахуванням не від'ємних
значень)
        result_set = SetUsingBinaryVector(self.size)
        for i in range(self.size):
            result_set.vector[i] = max(0, self.vector[i] - other_set.vector[i])
        print("Difference Result:")
```

```
        print(result_set.vector)
        return result_set

# Приклад використання
set1 = SetUsingBinaryVector(5)
set2 = SetUsingBinaryVector(5)

set1.add_element(1)
set1.add_element(3)
set2.add_element(2)
set2.add_element(3)

set1.is_member(3)

union_result = set1.union(set2)
intersection_result = set1.intersection(set2)
difference_result = set1.difference(set2)
```

Результатом виконання буде:

```
Is 3 a member of the set? True
Union Result:
[0, 1, 1, 1, 0]
Intersection Result:
[0, 0, 0, 1, 0]
Difference Result:
[0, 1, 0, 0, 0]
```

2.Оцінити складність, переваги і недоліки реалізації множин двоїчними векторами:

Складність:
Додавання та видалення елементів: O(1).
Об'єднання, перетин та різниця: O(n), де n - розмір множини.
Переваги:
Простота реалізації.
Можливість представлення множини з великою кількістю елементів.
Недоліки:
Займає пам'ять для всіх можливих елементів, що може бути невиправдано для розріджених множин.

Завдання 2.
3.Реалізувати АТД Множина на основі впорядкованих векторів.

```
class SetUsingOrderedVector:
    def __init__(self):
        # Ініціалізація об'єкта - створення порожнього впорядкованого вектора
        self.vector = []

    def add_element(self, element):
        # Додавання елементу до множини, якщо його ще немає, та впорядкування вектора
        if element not in self.vector:
            self.vector.append(element)
            self.vector.sort()
        print("Add Element Result:")
        print(self.vector)
```

```
    def remove_element(self, element):
        # Видалення елементу з множини, якщо він є у векторі
        if element in self.vector:
            self.vector.remove(element)
        print("Remove Element Result:")
        print(self.vector)

    def is_member(self, element):
        # Перевірка, чи є елемент частиною множини
        print(f"Is {element} a member of the set? {element in self.vector}")
        return element in self.vector

    def union(self, other_set):
        # Об'єднання двох множин (впорядкування за допомогою set та перетворення у
вектор)
        result_set = SetUsingOrderedVector()
        result_set.vector = sorted(list(set(self.vector + other_set.vector)))
        print("Union Result:")
        print(result_set.vector)
        return result_set

    def intersection(self, other_set):
        # Перетин двох множин (впорядкування за допомогою set та перетворення у вектор)
        result_set = SetUsingOrderedVector()
        result_set.vector = sorted(list(set(self.vector) & set(other_set.vector)))
        print("Intersection Result:")
        print(result_set.vector)
        return result_set

    def difference(self, other_set):
        # Різниця двох множин (впорядкування за допомогою set та перетворення у вектор)
        result_set = SetUsingOrderedVector()
        result_set.vector = sorted(list(set(self.vector) - set(other_set.vector)))
        print("Difference Result:")
        print(result_set.vector)
        return result_set

# Приклад використання
set1 = SetUsingOrderedVector()
set2 = SetUsingOrderedVector()

set1.add_element(3)
set1.add_element(1)
set2.add_element(2)
set2.add_element(3)

set1.is_member(3)

union_result = set1.union(set2)
intersection_result = set1.intersection(set2)
difference_result = set1.difference(set2)
```

4.*Оцінити складність, переваги і недоліки реалізації множин двоїчними векторами.*

Складність:
Додавання та видалення елементів: O(n), де n - розмір впорядкованої множини.
Об'єднання, перетин та різниця: O(m + n), де m і n - розміри множин.
Переваги:
Відсутність зайвого використання пам'яті для можливих елементів.

Недоліки:
Додавання та видалення елементів може бути повільним для великих впорядкованих множин

Результатом виконання буде:

```
Add Element Result:
[3]
Add Element Result:
[1, 3]
Add Element Result:
[2]
Add Element Result:
[2, 3]
Is 3 a member of the set? True
Union Result:
[1, 2, 3]
Intersection Result:
[3]
Difference Result:
[1]
```

Завдання 3.

5.Реалізувати задачу про базу даних з голосування депутатів.

```
class VotingDatabase:
    def __init__(self):
        # Ініціалізація бази даних - словник для зберігання голосів депутатів
        self.deputies_votes = {}

    def vote(self, deputy_id, decision):
        # Метод для голосування депутата з ідентифікатором deputy_id
        # decision може приймати значення 'for', 'against' або 'abstain'
        if deputy_id not in self.deputies_votes:
            self.deputies_votes[deputy_id] = decision
            print(f"Deputat {deputy_id} voted {decision}.")
        else:
            # Якщо депутат вже проголосував, вивести повідомлення
            print(f"Deputat {deputy_id} has already voted.")

    def get_decision(self, deputy_id):
        # Метод для отримання рішення (голосу) депутата за його ідентифікатором deputy_id
        if deputy_id in self.deputies_votes:
            decision = self.deputies_votes[deputy_id]
            print(f"Decision of Deputat {deputy_id}: {decision}")
            return decision
        else:
            # Якщо депутат ще не голосував, повернути None
            print(f"Deputat {deputy_id} has not voted yet.")
            return None

# Приклад використання
```

```
voting_db = VotingDatabase()

voting_db.vote(1, 'for')
voting_db.vote(2, 'against')
voting_db.vote(1, 'abstain')  # Спроба повторного голосування

decision_1 = voting_db.get_decision(1)
decision_2 = voting_db.get_decision(2)
decision_3 = voting_db.get_decision(3)  # Запит про депутата, який ще не голосував
```

Результатом виконання буде:

```
Deputat 1 voted for.
Deputat 2 voted against.
Deputat 1 has already voted.
Decision of Deputat 1: for
Decision of Deputat 2: against
Deputat 3 has not voted yet.
```

6.*Оцінити складність, переваги і недоліки реалізації словників методами*

Складність:
Додавання та видалення рішень: O(1).
Перевірка рішень: O(1).
Переваги:
Простота реалізації.
Швидкий доступ до рішень за ідентифікатором депутата.
Недоліки:
Може виникнути конфлікт, якщо депутат вже проголосував.
Не враховує інших аспектів голосування, таких як таємне голосування.

Контрольні запитання:

1. Перелічити основні операції АТД Множина:
   - Додавання елементу.
   - Видалення елементу.
   - Перевірка належності елементу до множини.
   - Об'єднання множин.
   - Перетин множин.
   - Різниця множин.
2. Перелічити методи реалізації АТД Множина на основі різних структур даних:
   - На основі двоїчного вектора.
   - На основі впорядкованого вектора.
3. Перелічити основні методи обходів дерева:
   - Прямий обхід (pre-order).
   - Центральний обхід (in-order).
   - Зворотний обхід (post-order).
4. Перелічити методи реалізації АТД Дерево та порівняти їх такими критеріями:
   - Ефективність.
   - Простота реалізації.
   - Універсальність.
5. Описати постановку задачі кодування та метод Хаффмана:
   Задача кодування полягає в призначенні бітового коду кожному символу з метою оптимізації представлення даних.
   Метод Хаффмана використовує дерево, де символи представлені листями, а їх коди визначаються шляхом до листя в дереві.
6. Описати структури даних необхідні для реалізації методу Хаффмана:

Структура даних для збереження частот символів.

Min-heap для ефективного вибору вузлів з найменшою частотою.

Бінарне дерево, де листя відповідають символам, а шляхи до листя визначають їх коди.