НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ім. Ігоря СІКОРСЬКОГО»
НАВЧАЛЬНО-НАУКОВИЙ ФІЗИКО-ТЕХНІЧНИЙ ІНСТИТУТ

Звіт з дисципліни «Прикладні алгоритми»

ВАРІАНТ 4

РЕАЛІЗАЦІЯ МНОЖИН ЗА ДОПОМОГОЮ ТИПУ SPARSESET

Виконала студентка
групи ФІ-33
Зварунчик Аріна Олександрівна

Київ — 2025

Зміст

1 Мета роботи

Опанувати способи представлення множин та їх ефективної реалізації, проаналізувати швидкість роботи операцій над множинами у заданій реалізації: за допомогою типу `SparseSet`.

2 Опис структури класу `SparseSet`

Клас `SparseSet` реалізує множину за допомогою двох масивів: одного для збереження елементів множини (`dense`) та іншого для збереження індексів елементів у `dense` (`sparse`).

Універсумом виступає множина цілих невідємних чисел.Реалізація підтримує значення `maxVal` та `capacity` не менші за 65536.

Також ініціалізуються такі параметри:

- `maxVal` — максимальне значення елементів множини;
- `capacity` — максимальний розмір множини;
- `n` — поточна кількість елементів у множині.

2.1 Методи класу `SparseSet`

- public int search(int x) :
  - перевіряє наявність елемента x у множині, повертає його індекс у `dense` або -1, якщо елемент відсутній.
  - складність: $O(1)$
- public void insert(int x):
  - додає елемент x до множини, якщо його ще немає і не перевищено `capacity`.
  - складність: $O(1)$
- public void delete(int x) :
  - видаляє елемент x з множини, замінюючи його останнім елементом у `dense`.
  - складність: $O(1)$
- public void clear() :
  - робить множину порожньою.
  - складність: $O(1)$
- public SparseSet union(SparseSet s1) :
  - повертає нову множину, яка містить усі елементи поточної множини та `s1`.
  - складність: $O(n)$

- public SparseSet intersection(SparseSet s2) :
  - повертає нову множину, що містить елементи, які є в обох множинах.
  - складність: $O(n)$
- public SparseSet setDifference(SparseSet s3) :
  - повертає нову множину, що містить елементи поточної множини, яких немає у `s3`.
  - складність: $O(n)$
- public SparseSet symDifference(SparseSet s4) :
  - повертає нову множину, що містить елементи, які належать лише одній із двох множин.
  - складність: $O(n)$
- public boolean isSubset(SparseSet s5) :
  - перевіряє, чи всі елементи поточної множини належать `s5`; якщо поточна порожня множина, повертає true.
  - складність: $O(n)$
- public int size() :
  - повертає кількість елементів у множині.
  - складність: $O(1)$

3 Опис структури класу `Main`

Клас `Main` виконує роль демонстраційного та тестового класу для перевірки роботи типу даних `SparseSet`.

3.1 Методи класу `Main`

- public static void main(String args[]) :
  - створює дві множини `setA` та `setB` за допомогою методу `createRandomSet`.
  - демонструє роботу методів класу `SparseSet`: `insert, delete, union, intersection, setDifference, symDifference, isSubset`.
  - виводить усі результати на екран за допомогою методу `printSet`.
- private static SparseSet createRandomSet(int maxVal, int capacity, int n) :
  - створює нову множину `SparseSet` з заданими параметрами.

- заповнює множину n випадковими числами в діапазоні від 0 до `maxVal`.
- повертає створену множину.

- private static void printSet(SparseSet s) :
  - виводить елементи множини.

4 Опис структури класу `Experiments`

Клас `Experiments` використовується для експериментального визначення часу роботи операцій, реалізованих через клас `SparseSet`, у залежності від розміру множини.

4.1 Методи класу `Experiments`

- public static void main(String[] args) :
  - задає параметри експерименту: масив розмірів множин `sizes`, `maxVal`, `capacity` та кількість експериментів `exper`.
  - для кожного розміру множини генерує випадкову множину за допомогою `generateRando`
  - вимірює час пошуку елементів, що належать множині, та тих, яких немає.
  - вимірює час виконання операції об'єднання множин `union`.
  - виводить результати експериментів у консоль.
  - створює графік результатів через клас `ResultsChart`.
- private static SparseSet generateRandomSet(int maxVal, int capacity, int size) :
  - створює порожню множину `SparseSet` з параметрами `maxVal` і `capacity`.
  - заповнює множину `size` унікальними випадковими числами від 0 до `maxVal`.
  - повертає створену множину для використання в експерименті.

5 Опис структури класу `ResultsChart`

Клас `ResultsChart` використовується для візуалізації результатів експериментів, виконаних над множинами `SparseSet`.

5.1 Методи класу `ResultsChart`

- public ResultsChart(int[] sizes, double[] timeIn, double[] timeNotIn, double[] timeUnion) :
  - конструктор класу, який створює та відображає графіки результатів експериментів.
  - викликає методи `createSearchChart` та `createUnionChart`.

- private void createSearchChart(int[] sizes, double[] timeIn, double[] timeNotIn) :
  - формує набір даних для графіка часу виконання операції `search`(коли еоемент належить множині і коли ні).
  - створює лінійний графік із підписами осей та легендою.
  - відображає графік за допомогою методу `showChart`.
- private void createUnionChart(int[] sizes, double[] timeUnion) :
  - формує набір даних для графіка часу виконання операції об'єднання множин `union`.
  - створює лінійний графік із підписами осей та легендою.
  - відображає графік за допомогою методу `showChart`.
- private void showChart(JFreeChart chart, String title) :
  - створює вікно `JFrame` із графіком `chart` та заголовком `title`.
  - встановлює розмір вікна, позицію на екрані та робить його видимим.

6 Тестування швидкості

6.1 Результати методу `union`

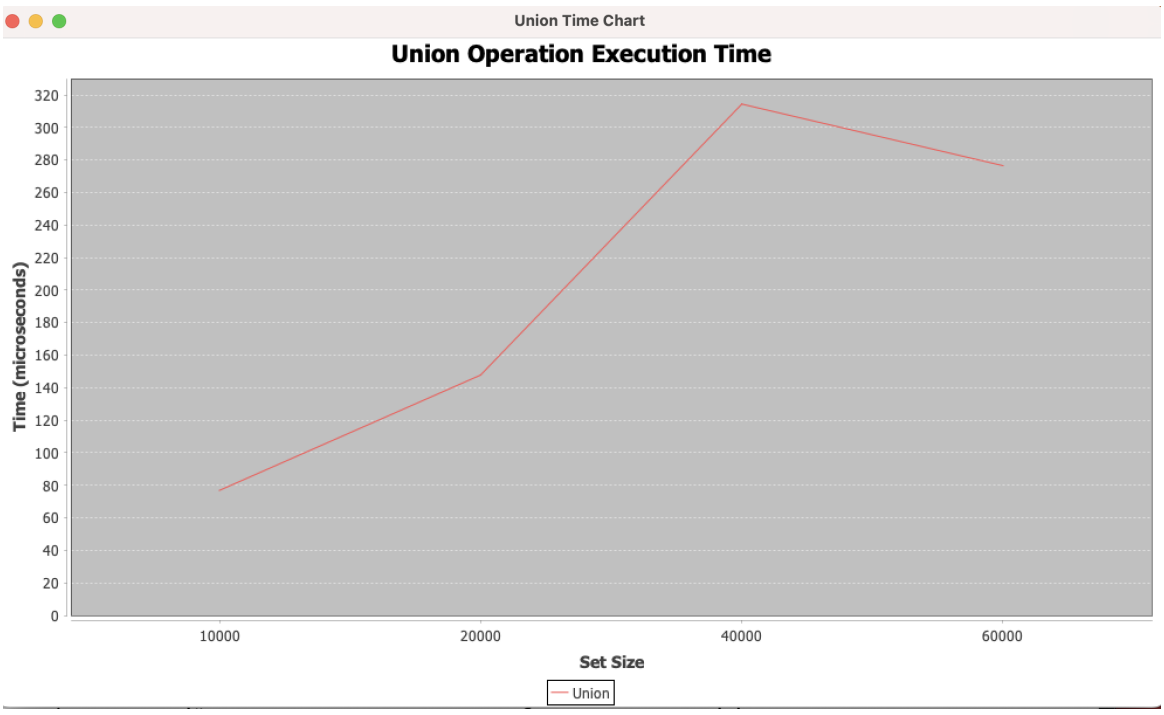

Рис. 1: Порівняння продуктивності операції `union` для різних розмірів множин.

По горизонтальній осі позначено розмір множини (n), а по вертикальній — середній час виконання операції у мікросекундах. **Аналіз результатів:**

Операція об'єднання `union` демонструє зростання часу виконання зі збільшенням розміру множин. На графіку видно, що для невеликих множин час виконання становить близько 80−150 мікросекунд, а при збільшенні розміру до 40,000–60,000 елементів час підвищується до 280−310 мікросекунд. Це свідчить про лінійну залежність часу від ромір-у множини, що відповідає алгоритмічній складності $O(n)$ операції `union`.

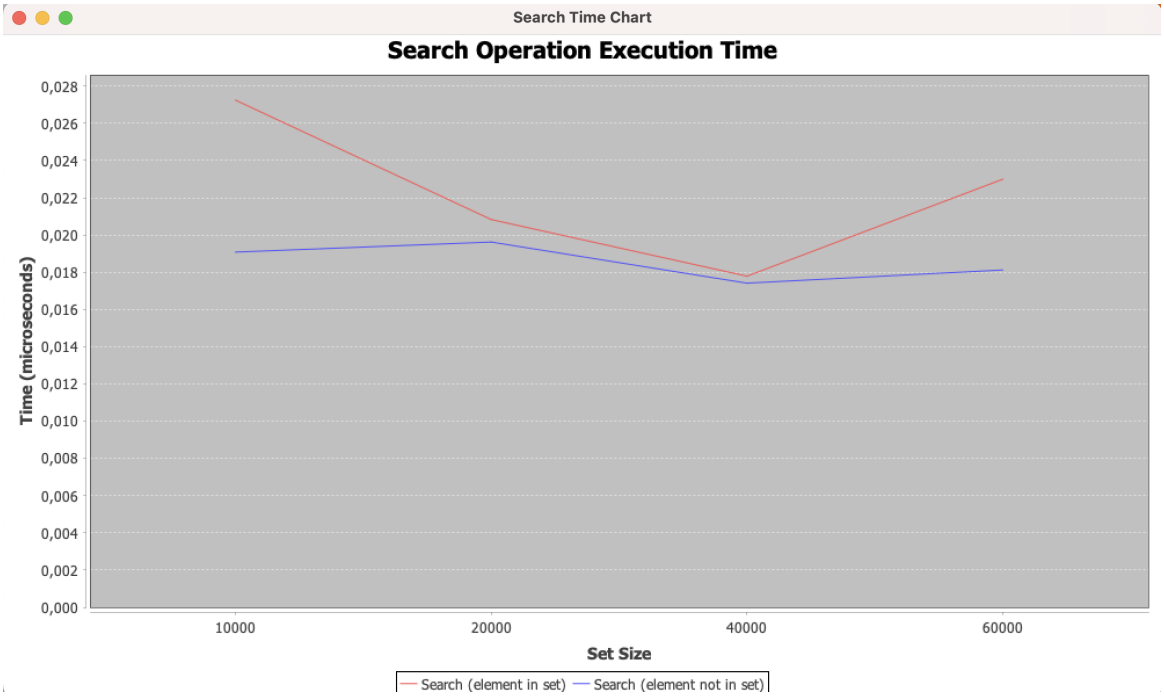

Рис. 2: Порівняння продуктивності операції `search`(у залежності,коли елемент належить множині,а коли ні) для різних розмірів множин.

Аналіз результатів: На графіку представлено результати для двох випадків: - пошук елемента, який належить множині (червона лінія), - пошук елемента, який не належить множині (синя лінія).

Як видно з результатів, середній час пошуку елемента практично не залежить від розміру множини й залишається майже сталим. Це відповідає алгоритмічній складності $O(1)$, характерній для структури даних `SparseSet`.

У середньому: - пошук елемента, який належить множині займає близько 0.02−0.027 мікросекунд, - пошук елемента, який не належить множині трохи повільніший (близько 0.018−0.02 мікросекунд).

Ця різниця пояснюється додатковими перевірками, необхідними для перевірки чи належить елемент множині.

7 Мій репозиторій:

https://github.com/arinacoola/AppliedAlgorithms