Національний технічний університет України
"Київський політехнічний інститут ім. Ігоря Сікорського"

Навчально-науковий Фізико-технічний інститут
Кафедра математичних методів захисту інформації

ЛАБОРАТОРНА РОБОТА 1: РЕАЛІЗАЦІЯ МНОЖИНИ ВАРІАНТ 1

Виконав студент
групи ФІ-23
Чуй Тимофій

Київ — 2024

Опис реалізації

У варіанті 1 потрібно було використовувати як базовий тип даних список, для спрошення коду у моєму випадку він двозв'язний; на швидкість операцій це не впливає (для однозв'язного можна було б записати потрібні сусідні вершини як сторонні вказівники). Список дозволяє зберігати об'єкти різних типів, залишив основні: $int, float, char, string$; проте вони можуть бути довільним за умови можливості порівняння, їх легко можна додати за необхідності до множини допустимих (вузел визначається з використанням шаблону класу variant, тому достатньо лише дописати потрібний тип).

Базові операції, здебільшого, реалізовані стандартним способом: лінійний пошук і потрібна операція в залежності від результату. З особливостей, метод
set_insert написаний у двох варіантах, оскільки іноді можна не виконувати перевірку наявності в множині і витратити трохи менше часу на деякі додаткові опреації. З тією ж метою метод set_search повертає не булеве значення, а вказівник на знайдений вузел, який можна використовувати надалі (наприклад, у set_delete). Щодо вилучення елементу, воно не просто перенаправляє вкзаівники, а додатково вивільняє пам'ять, виділену під вузел - як наслідок, set_clear має викликати set_delete для кожного елементу. Так як пошук починається з початку списку, кожний такий виклик закінчується на першому елементі, але все ж очистка множини вимагає час $\Theta(n)$.

Додаткова логіка так само використовує лінійний пошук, який я намагався уникати, де було можливо, за рахунок set_unsafe_insert. Усі методи, котрі приймають на вхід дві множини і мають поверунти третю, залишають вхід без змін і повертають вказівник на результат. Варто сказати, що методи додатково покриті юніт-тестами, усі вони пройшли.

Безпосередньо заміри часу я проводив на випадкових множинах, згенерованих за допомогою вихра Мерсена з випадковим початковим значенням: намагався наблизити якомога ближче розподіл до рівномірного, щоб особливості генерації не впливали на результат. Для спрощення в тестові множини входять лише типи $int, float, string$ - в такому випадку менш ймовірне неспівпадіння типів даних (що дозволяє не порівнювати значення і зменшує час), але, загалом, це не має сильно вплинути. Тестових сценаріїв було 10, з розмірами множин від 25 до 1025 з кроком 100; перевірялися пошук в двох випадках і обчислення різниці (так як остання не використовує спрощення додавання елементів); кожен сценарій повторювався 1000 раз.

Код - тут; за умови наявності бібліотеки $gtest$ його достатньо для автономної компіляції.

Результати експериментів

Далі наведено графіки залежності часу виконання від розміру множини, для set_search у мікросекундах, для $set\_difference$ у мілісекундах.

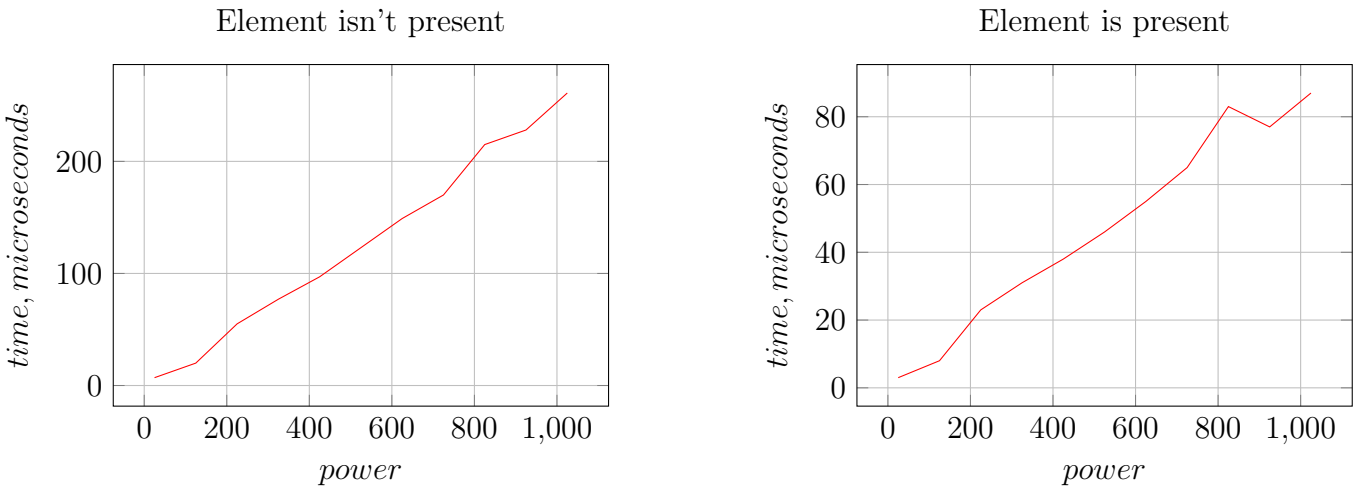

Рис. 1: Search tests

Загалом, перші виглядають очікуваним чином: пошук наявного елементу займає приблизно вдвічі менше часу, ніж відсутнього; для обох випадків залежність близька до лінійної, хоч і є невеликі відхилення, які можна пояснити відносно невеликою кількістю експериментів. Єдине питання до результатів тут - чому зі збільшенням потужності відношення часу пошуку відсутнього елементу до часу пошуку наявного повільно збільшується (починаючи від 2.3 до 3). Для перевірки я запустив додаткові тести для потужностей від 1125 до 1425 (не включені в графіки, бо не дають корисної інформації про характер росту і даремно зменешують маштаб) - та сама тенденція, пік в 1325 - 3.19. Прчина, ймовірно, у порівняння окремих елементів: оскільки вони можуть належати до різних типів, там потенційно може виникати складність більша, за лінійну.

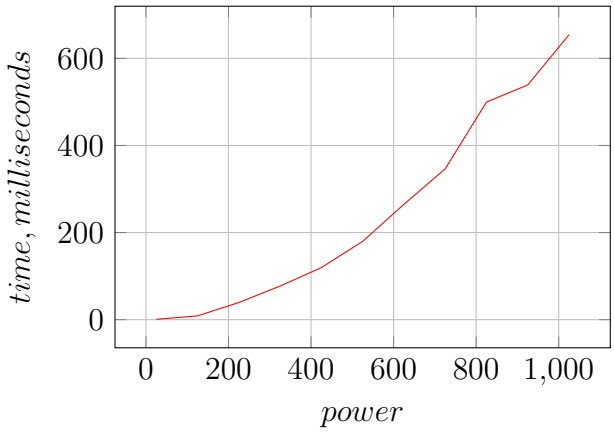

Рис. 2: Difference test

Графік для різниці явно відповідає нелінійній залажності, виглядає близько до квадратичної, як і має бути. Але з незрозумілих мені причин усі три графіки мають великий пік у точці 825. Для кожного експерименту множини незалежно генерувалися кожен раз, це не може бути особливістю структури тестової множини - чомусь при цій потужності виконувати операції складніше. Останнє зауваження стосується тільки однієї точки, тож за відсутності інших тестів (як і практичної можливості їх проведення) я вважаю результати тестів усіх операцій у високій мірі відповідними очікуваним.