Київський національний університет імені Тараса Шевченка
Факультет комп’ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Алгоритми та складність

Лабораторна робота №2
«Splay Tree»

Варіант 20. Тип даних T1

Виконав студент 2-го курсу
Групи ІПС-21

Шевнюк Михайло Олексійович

2025

Завдання

Реалізувати **splay-дерево** для статичної множини раціональних чисел (лінійний порядок очевидний). Список операцій:

створення порожнього дерева	**O(1)**
`insert(x)`	амортизовано **O(log n)**
`contains(x)` / `access`	амортизовано **O(log n)**
`extractMin()`	амортизовано **O(log n)**
друк дерева `print()`	**O(n)**

Теорія

Splay-дерево (Д. Слейтор, Р. Тар'ян, 1985) — це бінарне пошукове дерево, у якому після кожної операції доступу виконується серія ротацій (Zig, Zig-Zig, Zig-Zag), що піднімає відвіданий вузол у корінь. Така **самобалансування без додаткової інформації** забезпечує амортизовану `O(log n)` складність для всіх базових операцій.

Особливість: після кожної `insert`, `search` або `extractMin` коренем стає елемент, з яким працювали, тож часто-використовувані ключі знаходяться близько до кореня (ефект кешування).

Алгоритм побудови

1 Вставка

Звичайний BST-прохід, вставляємо новий вузол у лист.

Виконуємо `splay(newNode)` — послідовність Zig/Zig-Zig/Zig-Zag ротацій до кореня.

2 Пошук / contains

Йдемо як у BST, запам'ятовуємо останній відвіданий вузол. *Splay* знайдений (або останній) вузол.

3 Вилучення мінімуму

Йдемо ліворуч до найменшого ключа.

`splay(min)` — стає коренем.

Корінь видаляється; праве піддерево стає новим коренем.

Складність

- Insert, constains, extractMin - найгірший випадок O(n), амортизований O(log(n))
- Пам'ять – O(n).

Мова програмування

C++

Модулі програми

1. **`Rational`** — дріб у скороченій формі, оператори `< ==`, `<<`.
2. **`SplayTree<T, Comp>`** — шаблон контейнера (компаратор за замовчуванням `std::less<T>`). Публічні методи: `insert`, `contains`, `extractMin`, `print`.
3. Візуалізація — ASCII-дерево, повернуте на 90°.
4. **`main()`** — демонструє роботу на множині раціональних чисел і цілих.

Інтерфейс користувача

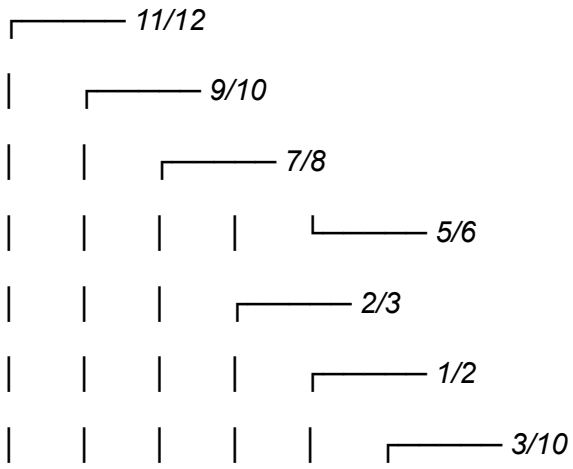
Користувач не вводить дані вручну; демонстраційний набір слів заданий у коді. При бажанні можна зчитати набір із файлу.

Тестовий приклад

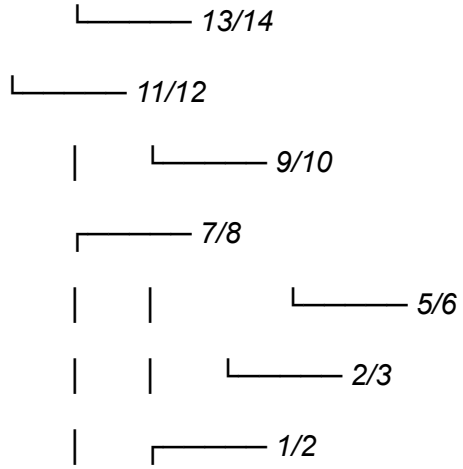
```
int main() {
    try {
        //——— Rational demo ——————————————————————————————
        vector<Rational> vec = { {n: 3, d: 10}, {n: 1, d: 2}, {n: 5, d: 6},
            {n: 7, d: 8}, {n: 2, d: 3}, {n: 9, d: 10}, {n: 11, d: 12}, {n: 13, d: 14} };
        SplayTree<Rational> ratTree;
        for (auto& r :Rational & : vec) ratTree.insert(r);

        cout << "NEW SPLAY TREE (Rational)\n";
        ratTree.print();

        cout << "EXTRACT MIN\n";
        ratTree.extractMin();
        ratTree.print();

        cout << "\n————————————————————————————————————————————————————\n\n";

        //——— int demo ——————————————————————————————————
        SplayTree<int> intTree;
        for (int i = 0; i < 10; ++i) intTree.insert(i);

        cout << "NEW SPLAY TREE (int)\n";
        intTree.print();

        cout << "EXTRACT MIN\n";
        intTree.extractMin();
        intTree.print();
    }
    catch (const exception& e) {
        cerr << "Error: " << e.what() << '\n';
        return 1;
    }
    return 0;
}
```

Результат

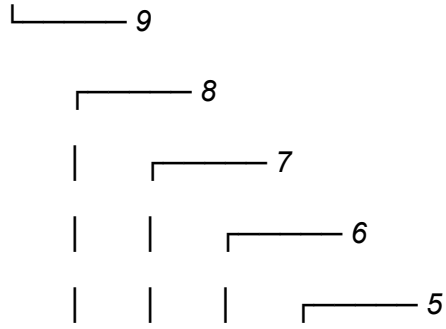
NEW SPLAY TREE (Rational)

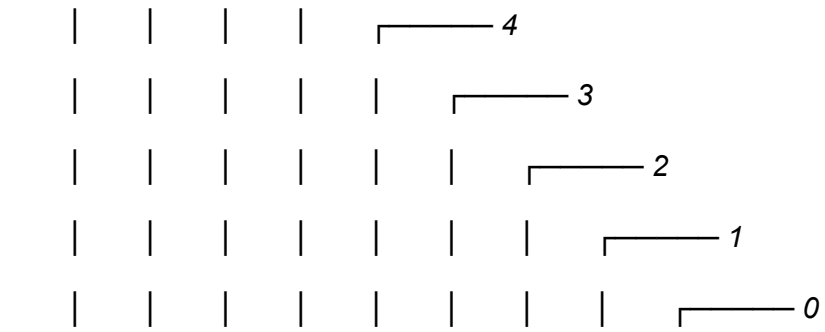


EXTRACT MIN

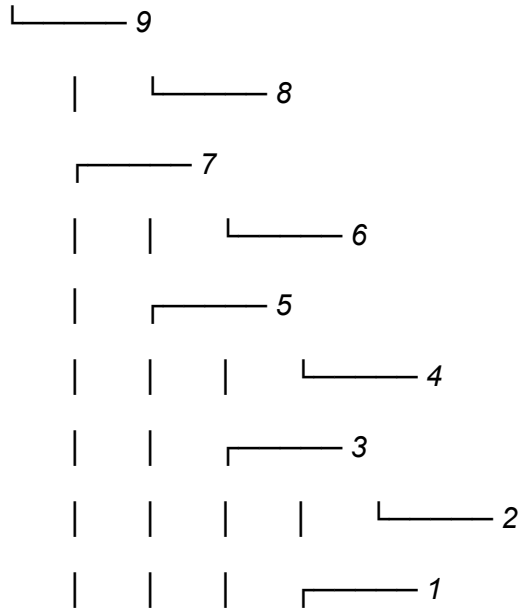


NEW SPLAY TREE (int)





EXTRACT MIN



Висновки

Splay-дерево забезпечує ті ж амортизовані `O(log n)` операції, що й червоне-чорне дерево, але не потребує додаткових полів для балансу та має властивість **динамічної оптимальності**: часті елементи автоматично знаходяться поблизу кореня.
ASCII-візуалізація спрощує налагодження та демонстрацію алгоритму.

Використана література

1. Томас Г. Кормен та ін. «Алгоритмы. Построение и анализ» (розд. 19).
2. CLRS repository — *binomial-heap* reference.
3. Sleator D., Tarjan R. *Self-Adjusting Binary Search Trees* // J. ACM, 1985.
4. Кормен Т. та ін. **Алгоритми.** Розд. 12.3 (splay-дерева).