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

Алгоритми та складність

Лабораторна робота №2-2

“ Реалізація B+ дерева”

Виконала студентка 2-го курсу

Групи ІПС-21

Сенечко Дана Володимирівна

Київ - 2025

Завдання

Реалізація B+ дерева. Тип даних за варіантом Т7 - комплексні числа.

Теорія

B+ дерево — тип дерева, яке подає відсортовані дані в вигляді, що дозволяє швидке додавання, отримання і видалення записів, кожен з яких ототожнений ключем. Це динамічний, багаторівневий індекс, з верхньою та нижньою межами на кількість ключів в кожному сегменті індекса (блоці або вершині). В B+ дереві, на відміну від B-дерева, всі записи зберігаються на рівні листових вузлів дерева; у внутрішніх вузлах зберігаються лише ключі.

Виділимо основні характеристики B+ дерева:

Сховище даних в листках: У B+ дереві дані зберігаються тільки в листових вузлах. Внутрішні вузли містять лише ключі, які вказують на листя, де знаходяться відповідні дані.

Порядок дерева: B+ дерево має фіксований порядок, що визначає максимальну кількість ключів, які можуть бути збережені в одному вузлі.

Упорядкованість даних: Дані у кожному листковому вузлі впорядковані за ключами. Це дозволяє швидко виконувати діапазонні запити та виконувати послідовний обхід даних.

Локальність даних: Сусідні дані часто зберігаються поруч один з одним у листових вузлах, що покращує локальність доступу до даних.

Алгоритм

Пошук

Корінь B+ дерева представляє весь діапазон значень в дереві, де кожен внутрішній вузол є підінтервалом. Нехай необхідно знайти значення *k*. Починаючи з кореня виконується пошук листа, який може містити значення *k*. На кожному вузлі необхідно з'ясувати, на який наступний внутрішній вузол необхідно слідувати. Внутрішній вузол B+ дерева має не більше ніж *b* дітей, кожен з яких являє собою окремий підінтервал. Ми обираємо відповідний вузол за допомогою пошуку у ключових значеннях вузла.

Додавання

В першу чергу необхідно знайти блок, в який необхідно додати новий запис.

- Якщо блок не повністю заповнений (кількість елементів після вставки не більше ніж b-1), то додати запис
- В іншому випадку необхідно розщепити блок
    - Додати новий блок, перемістити половину елементів з існуючого до нового
    - Додати найменший ключ та адресу з нового блоку до батьківського блоку
    - Якщо батьківський блок заповнено, аналогічно розділити його.
        - Додати середній ключ до батьківського блоку
    - Повторювати, поки батьківський блок не буде потребувати розщеплення.

- Якщо розщеплюється корінь — створити новий корінь, який має один ключ і два покажчика (значення, яке додається до кореня, видаляється з свого вузла)

В-дерева розширюється зі сторони кореня, а не зі сторони листів.

Видалення

В першу чергу необхідно знайти блок, в якому знаходиться запис, який необхідно видалити.

- Видалити запис
    - Якщо блок хоча б наполовину заповнений – завершення алгоритму
    - Якщо блок має менше елементів
        - Виконати спробу перерозподілити елементи, тобто додати до вузла елемент з брата (вузла, з яким поточний має спільного батька)
        - Якщо виконати перерозподіл не вдалось, об'єднати вузол з братом
- Якщо відбулося об’єднання, видалити запис (який вказує на видалений блок або його брата) з батьківського блоку
- Об’єднання може поширюватись на корінь, тоді відбувається зменшення висоти дерева

Складність алгоритму

Складність не перевищує $O(log_t n)$, де t - степінь дерева (мінімальна к-сть піддерев), а n - кількість елементів.

Мова реалізації алгоритму

C++

Модулі програми

- **`struct Complex;`**

  Реалізує комплексне число з операціями +, ==, < та вивід числа.

- **`class BPlusTree`**

  Клас для B+ дерева та операцій в ньому.

  - **`struct Node`** - структура вузла дерева;
  - **`void insert(const Complex& value)`** - вставка вузла;
  - **`void remove(const Complex& value)`** - видалення вузла;
  - **`void removeRecursive(Node* node, const Complex& value)`** - рекурсивна функція видалення;
  - **`Node* getParent(Node* currentNode, Node* childNode)`** - пошук батьківського вузла;
  - **`Node* findLeaf(Node* node, const Complex& value)`** - пошук листка;
  - **`void insertIntoLeaf(Node* leaf, const Complex& value)`** - вставка в листок;
  - **`void splitLeaf(Node* leaf)`** - поділ листка;
  - **`void insertIntoParent(Node* left, const Complex& key, Node* right)`** - вставка в батьківський вузол;
  - **`void splitNode(Node* node)`** - поділ вузла;
  - **`void printTree()`** - виведення дерева.

- **`int main()`**

  Головна функція програми.

Інтерфейс користувача

Вхідні дані вводяться програмно (в функції int main()). Результат виводиться в консоль.

Тестові приклади

1) Додаємо вершини з числами: 1+2i, -3+4i, 5-6i
2) Видалимо вершину -3+4i
3) Повернемо -3+4i та видалимо 1+2i

```
dunnaya@MacBook-Air-Dana lab2 % ./bplus_tree
-3+4i |
1+2i | -3+4i 5-6i |

After deleting (-3, 4):
5-6i |
1+2i | 5-6i |

After returning (-3, 4) and deleting (1, 2):
5-6i |
-3+4i | 5-6i |
```

Розберемо детальніше:

Спочатку вставляємо по порядку комплексні числа: 1+2i, -3+4i, 5-6i. Оскільки порядок дерева в даному випадку встановлено 3, максимальна кількість значень в листі - 2. Відбувається спліт листа та отримуємо перший етап виводу програми: лівий лист: [1+2i], правий лист: [-3+4i, 5-6i]. Мінімальний ключ із правого листа (-3+4i) просувається в внутрішній вузол.

Далі видаляємо внутрішній ключ -3+4i. Він замінюється на найменший ключ із правого піддерева, тобто 5-6i.

І останнім кроком повертаємо значення -3+4i та видаляємо 1+2i. Вставка -3+4i відбувається після значення 1+2i, тобто у ліве піддерево. Далі ми видаляємо 1+2i -> ліве піддерево залишається заповненим наполовину, тому на цьому все.

Висновки

B+ дерево є вдосконаленою версією B-дерева, яке широко використовується для оптимізації роботи з базами даних і файловими системами. Основні особливості B+ дерева полягають у тому, що всі дані розташовані в листових вузлах, які з'єднані між собою у вигляді зв'язаного списку, а також в тому, що всі ключі знаходяться в листових вузлах і внутрішні вузли містять лише ключі для навігації. Така структура дозволяє ефективно виконувати операції пошуку, вставки і видалення.

Використані джерела

- Wikipedia
- B+ tree visualization
- https://stackoverflow.com