Московский авиационный институт
(национальный исследовательский университет)

Факультет компьютерных наук и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: Л. А. Постнов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №2

Задача: Необходимо создать программную библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, представляющему из себя регистронезависимую последовательность букв английского алфавита длиной не более 256 символов, поставлен в соответствие некоторый номер, от 0 до 264 - 1. Разным словам может быть поставлен в соответствие один и тот же номер.

Вариант: B-дерево.

1 Описание

B-дерево — это особый тип сбалансированного дерева поиска, в котором каждый узел может содержать более одного ключа и иметь более двух дочерних элементов. Из-за этого свойства B-дерево называют сильноветвящимся.

Как сказано в [1]: « B-деревья схожи с красно-черными деревьями в том, что все B-деревья с n узлами имеют высоту O(lg n), хотя само значение высоты Bдерева существенно меньше, чем у красно-черного дерева за счет более сильного ветвления. Таким образом, B-деревья также могут использоваться для реализации многих операций над динамическими множествами за время O(lg n).».

2 Исходный код

B-дерево состоит из следующих структур:

```
const short KEY_SIZE = 257;
const short CHARACTERISTIC_NUMBER = 5;

struct Pair {
    char key[KEY_SIZE] = "";
    unsigned long long value = 0;
};

class Node {
    friend class BTree;

private:
    Pair* data;
    Node** child;
    int n;
    bool leaf;
    int t;

    Node(int, bool);
    Node* search(char*);
    // functions for insertion
    void split_child(int, Node*);
    void insert_non_full(Pair);
    // functions for deletion
    void remove_main(char*);
    void remove_from_leaf(int);
    void remove_from_non_leaf(int);
    void steal_from_next(int);
    void steal_from_prev(int);
    void fill_up_node(int);
    void merge(int);
    // function to work with file
    void save_node(std::ofstream&);
    void destroy_node();
};

class BTree {
    Node* root;
    int t;
    void insert_from_file(Pair);
public:
    BTree(int);
    void insert(Pair);
    void remove(char*);
    unsigned long long* search(char*);
    void save_to_file(char*);
```

```
47      void load_from_file(char*);
48      ~BTree();
49  };
```

Далее кратко описана работа основных операций B-дерева:

Поиск:

-проверка на пустоту корня
-вызов функции от узла (search)
-обработка результата

Работа search(node):

-рекурсивный поиск до (не)нахождения ключа

Вставка:

-если корень пустой, просто вставляем
-проверяем на наличие ключа (search)
-если корень полный, делим его (split child)
-вызываем insert non full от корня

Работа insert non full:

-если лист, просто вставка
-если не лист, проверка на полноту ребенка, если он полный, то вызов split child
-рекурсивное продолжение

Работа split child:

-создается новый ребенок
-в него переносятся половина данных и, если это не лист, половина детей
-записываем в вершину середину полного ребенка и добавляем нового ребенка

Удаление:

-проверки на наличие ключа (пустота корня и search)
-вызов remove от корня
-если корень пустой, то уменьшаем высоту дерева

Работа remove:

-если найден и лист, вызов remove from leaf, если нет - remove from non leaf
-если не найден, смотрим, что в ребенке не менее t, если это не так, то дополняем (fill node)
-вызываем рекурсивно remove

Работа remove from leaf:

-просто удаляем ключ

Работа remove from non leaf:

-ставим на место удаляемого элемента либо максимум левого сына, либо минимум правого, вызываем удаление из листа относительно перемещенного элемента
-если ни у левого, ни у правого нельзя забрать элемент, то нужно их объединить, вызвав merge, а потом удалить из обединенной вершины

Работа fill node:
-смотрим, у какого брата можно позаимствовать элемент, в соответствии с этим вызываем borrow from next/prev
-если ни у какого, то объеиняем вершины (merge)

Работа borrow from next/prev:
-переносим ключ из корня, берем из брата ключ и ставим его в корень, переписываем детей

Работа merge:
-берем нужный ключ из корня и записываем в левую вершину
-переносим ключи и детей из правой вершины
-очищаем правую вершину

3 Консоль

```
wednees@MSI:/mnt/c/Users/leoni/OneDrive/Desktop/study/Discrete_Analysis/lab2/src$
./solution
+ a 1
+ A 2
+ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa 18446744073
709551615
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
A
-A
a

OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
```

4 Тест производительности

Тесты производительности представляют из себя следующее: словарь на основе B-дерева сравнивается со словарем из STL*(std::map)*, в основе которого лежит RB-дерево. Тестов будет три: на 10^3, 10^4 и 10^5 операций.

```
wednees@MSI:/mnt/c/Users/leoni/OneDrive/Desktop/study/Discrete_Analysis/lab2/
benchmark$ g++ benchmark.cpp -o benchmark
wednees@MSI:/mnt/c/Users/leoni/OneDrive/Desktop/study/Discrete_Analysis/lab2/
benchmark$ ./benchmark <test_1000.txt
B-tree: 0.454 ms
std::map: 0.902 ms
wednees@MSI:/mnt/c/Users/leoni/OneDrive/Desktop/study/Discrete_Analysis/lab2/
benchmark$ ./benchmark <test_10000.txt
B-tree: 7.023 ms
std::map: 7.148 ms
wednees@MSI:/mnt/c/Users/leoni/OneDrive/Desktop/study/Discrete_Analysis/lab2/
benchmark$ ./benchmark <test_100000.txt
B-tree: 88.892 ms
std::map: 88.991 ms
```

Как видно, на всех тестах, *std::map* проигрывает в скорости работы, однако с увеличение числа тестов, разрыв становится все менее существенным, так что можно сказать, что они работают одинаково быстро. И действительно, операции в обеих структурах данных работают с логарифмической сложностью. Начальный отрыв во времени работы B-дерева можно связать с тем, что на начальных этапах его не нужно балансировать и выделять дополнительную память так же часто, как в красно-черном дереве.

5 Выводы

Рабоатая над данной лабораторной работой по курсу «Дискретный анализ», я познакомился со структурой данных под названием B-дерево, реализовал ее на языке C++. Кроме того, я получил новый опыт по работе с бинарными файлами.

При выполнении данной работы возникли трудности при работе с указателями, важно было очень внимательно следить за тем, чтобы не допустить ошибки, в этом хорошо помогал отладчик gdb.

Считаю, что опыт написания сложных струкур данных, полученный при работе очень ценен и обязательно пригодится в будущем.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.* — Издательский дом «Вильямс», 2007. Перевод с английского: И. В. Красиков, Н. А. Орехова, В. Н. Романов. — 1296 с. (ISBN 5-8459-0857-4 (рус.))

[2] *Introduction of B-trees*
URL: `https://www.geeksforgeeks.org/introduction-of-b-tree-2/` (дата обращения: 01.05.2024).