Московский авиационный институт
(национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: А. А. Боглаев
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 $2^{64}-1$. Разным словам может быть поставлен в соответствие один и тот же номер.

Программа должна обрабатывать строки входного файла до его окончания. Каждая строка может иметь следующий формат:

+ word 34 — добавить слово «word» с номером 34 в словарь. Программа должна вывести строку «OK», если операция прошла успешно, «Exist», если слово уже находится в словаре.

- word — удалить слово «word» из словаря. Программа должна вывести «OK», если слово существовало и было удалено, «NoSuchWord», если слово в словаре не было найдено.

word — найти в словаре слово «word». Программа должна вывести «OK: 34», если слово было найдено; число, которое следует за «OK:» — номер, присвоенный слову при добавлении. В случае, если слово в словаре не было обнаружено, нужно вывести строку «NoSuchWord».

! Save /path/to/file — сохранить словарь в бинарном компактном представлении на диск в файл, указанный парамером команды. В случае успеха, программа должна вывести «OK», в случае неудачи выполнения операции, программа должна вывести описание ошибки (см. ниже).

! Load /path/to/file — загрузить словарь из файла. Предполагается, что файл был ранее подготовлен при помощи команды Save. В случае успеха, программа должна вывести строку «OK», а загруженный словарь должен заменить текущий (с которым происходит работа); в случае неуспеха, должна быть выведена диагностика, а рабочий словарь должен остаться без изменений. Кроме системных ошибок, программа должна корректно обрабатывать случаи несовпадения формата указанного файла и представления данных словаря во внешнем файле.

Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутсвие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Вариант структуры данных: B-дерево.

1 Описание

Требуется написать реализацию словаря с помощью B-дерева. Нужно реализовать основные операции (вставка, удаление, поиск) и операции работы с диском (запись данных на диск, чтение данных с диска).

Как сказано в [2]: «B-дерево (англ. B-tree) — сильноветвящееся идеально сбалансированное дерево поиска.».

Сильноветвящееся дерево - каждый узел B-дерева может иметь много дочерних узлов, для этого у B-дерева есть параметр t, называемый минимальной степенью B-дерева ($t >= 2$). Данный параметр отвечает за ветвление дерева.

Сбалансированное дерево - дерево, для которого выполняется следующее свойство: для каждой его вершины высота её двух поддеревьев различается не более чем на 1. Так как B-дерево является идеально сбалансированным, то для каждой вершины высота её двух поддеревьев будет одинакова.

Свойства B-дерева:

1. Каждый узел, кроме корня, содержит от $t-1$ до $2t-1$ ключей. Корень содержит от 1 до $2t-1$ ключей.
2. Ключи в каждом узле упорядочены по неубыванию (для быстрого доступа к ним).
3. Каждый узел дерева, кроме листьев, содержащий ключи K_1, ..., K_n, имеет $n+1$ потомка. Причем
   - Первый дочерний узел будет содержать числа в диапазоне $(-\infty, K_1)$.
   - $n+1$ дочерний узел будет содержать числа в диапазоне (K_n, ∞).
   - Дочерние узлы, находящиеся между первым и последним элменетом, будут сожержать числа в диапазоне (K_{i-1}, K_i).
4. Все листья находятся на одном уровне.

Поиск в B-дереве очень похож на поиск в бинарном дереве поиска. Отличие в том, что в бинарном дереве поиска выбирали один из двух путей, в B-дереве выбираем из интервала. Если ключ содержится в узле, то он найден, иначе продолжаем поиск.

Вставка в B-дерево сложнее, чем вставка в бинарное дерево поиска, так как нам нужно соблюдать свойства B-дерева. Нельзя создавать узел для вставки, как это принято в бинарном дереве поиска. Вместо этого, мы должны вставлять элемент

в уже существующий узел. Но при этом важно проверять, что узел не заполнился ($2t-1$ элементов в узле). Если же узел заполнен, то его нужно разбить.

Для этого у нас есть операция разбиения. Заполненный узел делим на два узла, в каждом их готорых $t-1$ ключ. Средний ключ перемещается в родительский узел, он будет разделительной точкой для двух новых узлов. Может произойти ситуация, когда родительский узел тоже будет заполнен. Его тоже нужно разделить.

Вставку в B-дерево можно осуществить за один проход от корня к листу. При проходе в поисках нужной позиции от корня к листу будем разделять все заполненные узлы. Так мы сможем гаранитировать, что при разделении какого-то узла, родительский узел не будет заполнен.

Аналогично вставке, для удаления нужно проверять, что выполняются свойства B-дерева. Проще всего удалить ключ из листа, главное проверить, что в листе больше, чем $t-1$ элемент. Иначе придется брать ключ у левого или правого брата. Опять же проверяя, что для брата выполняется свойство 1. Если же и в братьях по $t-1$ элементу, то нужно объединить текущий узел и брата с родительским узлом. В новом узле станет $2t-1$ элемента. Теперь можно удалить элемент. В случае с удаление из внутреннего узла нужно найти предшественника или преемника, если предшественник или преемник содержит больше, чем $t-1$ ключ, то заменяем исходный удаляемый ключ на последний из предшественника или на первый из преемника, после чего удаляем найденный ключ из предшественника или преемника. Если же предшественник и преемник содержат по $t-1$ ключу, то нужно объединить их с родительским узлом и удалить исходный элемент.

Удаление просиходит за один проход по дереву, но при удаленни ключа из внутреннего узла может потребоваться возврат к узлу, ключ из которого был удален и замещен его предшественником или последующим за ним ключом.

Так как дерево идеально сбалансированно, опреации с деревом выполняются за $O(h)$ (h - высота дерева).

Согласно [1] : «Пусть B-дерево имеет высоту h. Корень дерева содержит как минимум один ключ, а все остальные узлы — как минимум по $t-1$ ключей. Таким образом, имеется как минимум 2 узла на глубине 1, как минимум $2t$ узлов на глубине 2, как минимум $2t^2$ узлов на глубине 3 и т.д., до глубины h, на которой имеется как минимум $2t^{h-1}$ узлов. Следовательно, число ключей n удовлетворяет следующему неравенству:

$$n >= 1 + (t-1)\sum_{i=1}^{h} 2t^{i-1} = 1 + 2(t-1)(\frac{t^h-1}{t-1}) = 2t^h - 1$$

».

Логорифмируем по основанию t:

$$h <= \log_t \frac{n+1}{2}$$

Так как t - константа, то можем записать неравенсво так:

$$h <= \log n$$

Все операции с деревом принимают сложность $O(\log n)$.

Для записи дерева в файл, спускаем по дереву от корня к листьям, при этом записываем в файл узлы, вместе с информацией о них, слева направо в бинарном представлении. То есть пишем в файл текущий узел, после записываем его дочерние узлы, в порядке их следования. Если узел пустой, то мы ничего не пишем. Для чтения дерева из файла, создаем пустой узел и записываем в него прочитанные данные, после чего по порядку читаем и записваем информацию о его дочерних узлах. Если узел был пустой, то записываем нулевой указатель. Сложность операций $O(n)$.

2 Исходный код

Узел B-дерева состоит из нескольких ключей и указательей на дочерние узлы. Для хранения ключей и указательей на дочерние узлы будем использовать динамический массив. Так как степень (характеристичекое число) t B-дерева не меняется, то для написания дерева динамического массива будет достаточно. С его помощью мы сможем легко разделить и объединить узлы, и найти нужный ключ.

Узел дерева хранит массив ключей и массив указателей на дочерние узлы.

```
1  struct TNode {
2      int n;
3      TElem* el;
4      TNode** children;
5      bool leaf;
6
7      static TElem* Search(TNode *TNode, std::string key);
8      static TElem FindSuccessor(TNode *TNode);
9      static TElem FindPredecessor(TNode *TNode);
10 };
```

Методы для узла

btree.hpp	
TElem FindSuccessor(TNode *node)	Поиск преемника, для удаления из внутреннего узла
TElem FindPredecessor(TNode *node)	Поиск предшественника, для удаления из внутреннего узла
TElem* Search(TNode *node, std::string key)	Поиск узла

Методы для B-дерева

btree.hpp	
TNode *AllocateTNode()	Выделение памяти для узла
void Deallocate(TNode *node)	Освобождение памяти, выделенной под узел
void DeleteTree(TNode *node)	Удаление дерева
bool Search(std::string key)	Поиск ключа
void SplitChild(TNode *empty_node, int i, TNode *em_node_child)	Разбиение узла
void Insert(TElem elem)	Вставка элемента в дерево
void InsertNonfull(TNode *not_full_node, TElem elem)	Вставка элемента в неполный узел

TNode *MergeTNodes(TNode *parent, TNode *left_child, TNode *right_child, int i)	Объединение узлов
bool Delete(std::string key)	Удаление элемента из дерева
bool DeleteFromTNode(TNode *node, std::string key)	Удаление элемента из узла
std::string SWV(std::string key)	Поиск со значением, для задания
void WriteInFile(TNode *node, std::ofstream& os)	Запись дерева в файл
TNode *LoadFromFile(std::ifstream& in)	Чтение дерева из файла
void Save(std::ofstream& os)	Обертка для функции записи в файл
void Load(std::ifstream& in)	Обертка для функции чтения из файла
main.c	
std::string ToLower(std::string str)	Функция для преобразования в нижний регистр

Для удобтва, напишем структуру для хранения пары ключ-значение.

Листинг

```
struct TElem {
    std::string key;
    uint64_t value;
};


class TBTree {
    private:
        struct TNode {
            int n;
            TElem* el;
            TNode** children;
            bool leaf;

            static TElem* Search(TNode *node, std::string key);
            static TElem FindSuccessor(TNode *node);
            static TElem FindPredecessor(TNode *node);
        };

        TNode *AllocateTNode();
        void Deallocate(TNode *node);
        void DeleteTree(TNode *node);

        void SplitChild(TNode *parent, int index, TNode* child);
        TNode *MergeTNodes(TNode *parent, TNode *left_child, TNode *right_child, int i)
            ;
```

```
26
27          void InsertNonfull(TNode *node, TElem elem);
28          bool DeleteFromTNode(TNode *node, std::string key);
29
30          void WriteInFile(TNode *node, std::ofstream& os);
31          TNode *LoadFromFile(std::ifstream& in);
32
33          TNode *root;
34          int t;
35
36      public:
37          TBTree();
38          ~TBTree();
39
40          void Insert(TElem TElem);
41          bool Delete(std::string key);
42          bool Search(std::string key);
43          void Save(std::ofstream& os);
44          void Load(std::ifstream& in);
45          std::string SWV(std::string key);
46  };
```

3 Консоль

```
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_02$ make
g++ -std=c++20 -pedantic -Wall -Wextra -Werror -g main.cpp -o lab2
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_02$ cat tests/test.t
+ English 5
! Save Subj.txt
+ PE 35304034
+ Science 5673934
Math
! Load Subj.txt
+ Math 8388923492
+ History 123456
Math
English
-History
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_02$ ./lab2 <tests/test.t
OK
OK
OK
OK
NoSuchWord
OK
OK
OK
OK: 8388923492
OK: 5
OK
```

4 Тест производительности

Сравним скорость работы B-дерева с $std :: map$. Ключи будут иметь тип $std :: string$, а значения $uint64\_t$. В $std :: map$ не реализованна работа с файлами, поэтому будем тестировать только опреации поиска, вставки и удаления. Степень дерева равна 4. Для измерений возьмем следующие наборы тестов: 1000 строк, 10000 строк, 100000 строк, 500000 строк.

```
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_02$ make benchmark
g++ -std=c++20 -pedantic -Wall -Wextra -Werror -g benchmark.cpp -o benchmark
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_02$ ./benchmark <tests/1000.t
Btree: 1.251 ms
Map: 0.673 ms
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_02$ ./benchmark <tests/10000.t
Btree: 17.181 ms
Map: 8.997 ms
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_02$ ./benchmark <tests/100000.t
Btree: 218.552 ms
Map: 117.412 ms
alex@wega:~/Рабочий стол/вуз/2course/da_labs/Lab_02$ ./benchmark <tests/500000.t
Btree: 1171.639 ms
Map: 649.296 ms
```

В основе $std :: map$ лежит красно-черное дерево. Так как мы работаем преимущественно с опреативной памятью, то красно-черное дерево будет более эффективным, потому что не требует большого числа копирований. В B-дереве мы теряем время на операциях разбиения и объединения, так как копируем элементы из одного массива в другой. Во время операций поиска, вставки и удаления мы проходим по узлам дерева циклом, что тоже занимает время.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я ознакомился со структурой данных B-дерево и реализовал ее. Также я узнал, что данная структура используется внутри многих баз данных для работы с индексами. Так как с ее помощью можно быстрее работать с информацией, которая лежит во внешней памяти.

В ходе работы столкнулся с трудностями при работе с индексами массива, справится с этим мне помог отладчик. Также возникли трудности при работе с файлами. Нужно было аккуратно написать функции для чтения из файла и записи в файл, чтобы не потерять информацию.

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] *B-дерево — Викиконспекты.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=B-дерево` (дата обращения: 09.05.2024).