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: В. П. Будникова
Преподаватель: А. А. Кухтичев
Группа: М8О-209Б-19

Дата:
Оценка:
Подпись:

Москва, 2020

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 структурах данных:: B-дерево.

Тип ключа: `Регистронезависимую последовательность букв английского алфавита длиной не более 256`

Тип значения: `Числа от` 0 `до` $2^{64} - 1$.

Формат входных данных и результата: 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.

1 Описание

Как сказано в [1]: «B-дерево (по-русски произносится как Би-дерево) — структура данных, дерево поиска. С точки зрения внешнего логического представления, сбалансированное, сильно ветвистое дерево. Часто используется для хранения данных во внешней памяти.». Для B-Дерева существует значение t, которое называется степенью B-Дерева. «Здесь t — параметр дерева, не меньший 2.» [1] В элементах(узлах) B-Дерева содержатся ключи и ссылки для другие узлы или листья. Количество ключей в элементе больше, чем $t-1$ и меньше, чем $2*t-1$, а количество ссылок всегда на один элемент больше, чем количество элементов, т. е. оно не может быть больше, чем $2*t$. Исключением является корень - количество элементов в корне больше 1 и меньше, чем $2*t-1$. Листья B-Дерева всегда находятся на одной высоте. Элементы(ключи) узла всегда отсортированы, причем элементы, находящиеся в потомке, ссылка на который лежит между элементами X и Y должны быть больше X и меньше Y. B-Дерево в данной реализации не содержит повторяющиеся элементы.

Описание реализации:
Сравнение элементов происходит путем сравнения их ключей. Для сравнения строк, их записи и проведения над ними операций реализован класс TString. Корень, узлы и листья B-дерева реализованы с помощью структуры TList, где хранятся массивы элементов и массивы указателей, также хранятся значения размеров(в конкретный момент) этих массивов. Также реализованы методы вставки, удаления, поиска элемента в массиве(или ссылок в массиве ссылок). Элемент реализован с помощью структуры TElem, где хранятся ключ и значения.

Поиск:
Проверяем, находится ли элемент в корне. Если да - то поиск завершается успехом. Если нет, нужно найти такой интервал (X , Y), что искомый элемент больше X и меньше Y и искать в потомке, на которого указывает ссылка между этими элементами. Если такого интервала нет, то это значит, что искомый элемент больше всех элементов в корне, искать в самом правом потомке. Далее рекурсивно ищем нужный нам элемент в потомке. Если мы дошли до листа и в нем не содержится данный элемент, то поиск завершается неудачей.

Добавление ключа:
Добавление элемента происходит только в лист, причем его размер должен быть строго меньше, чем $2*t-1$, иначе при вставке элементов будет больше, чем $2*t-1$, а это противоречит определению B-Дерева. Поэтому, когда мы спускаемся к нужному листу(действия аналогичные поиску) каждый раз будем проверять размер потомка. Если потомков нет и размер его родителя строго меньше, чем $2*t-1$, то мы просто вставляем элемент(учитывая то, что элементы должны быть отсортированы). Если нет потомков, а размер родителя равен $2*t-1$, то выталкиваем средний элемент к родителю и рекурсивно запускаем вставку от родителя - это может произойти

только в том случае, если родитель является корнем, так как во всех остальных узлы с помощью данного алгоритма будут разгружены. Если Длина корня меньше $2 * t - 1$ и у него есть потомки, то смотрим на потомка. Если его длина равна $2 * t - 1$, то разгружаем его как указанно выше, и рекурсивно добавляем в него ключ.

Удаление:

Для удаления обратная ситуация, при поиске удаляемого элемента нужно каждый раз убеждаться, что в узле элементов не меньше, чем $t-1$, структура B-Дерева нарушится. Удаление происходит только из листов. Если у корня нет потомков(в данном случае корень и есть лист), то удаляем из корня. Если мы дошли до нужного листа, то, при наличии искомого элемента, мы его удаляем и удаление завершается успехом. Если мы не нашли нужный нам для удаления элемент, то удаление завершается неудачей. Если у нужного нам потомка элементов меньше, чем $t - 1$, то смотрим на его братьев. Если в одном из братьев элементов будет больше, чем $t - 1$, то воруем у него самый ближний к нашему потомку элемент, при этом ставим этот элемент на место родительского, от которого мы пошли в потомок, а родительский элемент добавляем в наш потомок. Если у обоих братьев элементов по $t-1$(или наш потомок крайних и у брата $t - 1$), то объединяем двух братьев и переходим в удаление от объединенного корня.

Сохранение в файл и чтение из него:

При сохранении в файл сначала записывается число - количество элементов в узле, потом записываются пары ключ-значения всех элементов узла, далее функция сохранения в файл вызывается для всех потомков узла с 1 го по последний. когда мы доходим до листа, мы записывается размер листа, потом все элементы диста(пары ключ-значение), а потом записываем фиктивное значение - -1 ровно столько раз, сколько у нас ссылок «в никуда», т.е. на один больше, чем количество элементов в листе. Фиктивные значения нужны для того, чтобы при считывании понять, что мы спустились к листу и дальше(после всех фиктивных элементов) начинается новый узел(или лист). Функция чтения возвращает указатель на структуру узла дерева. При чтении мы считываем сначала значение длины узла, потом считываем пары, записывая из значения в узел, далее ровно на один раз больше вызываем функцию считывания, и присваиваем ее значение потомкам этого узла. Если мы считали -1, то возвращаем nullptr, если наша функция возвратила nullptr, то ма не записываем потомком листу.

2 Исходный код

Константы	
Length, Length2, Length3	Константы, задающие степень, максимальную и минимальную длину для узла
struct TString	
int Size()	Функция, для вывода текущей длины строки
TString& operator=(const TString &rvl)	Перегрузка операторов = для TString
TString& operator=(const char* c)	
bool operator==(const TString &rvl) const;	Перегрузка оператора == для TString
bool operator!=(const TString &rvl) const;	Перегрузка оператора != для TString
bool operator<(const TString &rvl) const;	Перегрузка оператора < для TString
bool operator>(const TString &rvl) const;	Перегрузка оператора > для TString
char& operator[](const int i)	Перегрузка оператора доступа к элементу [] для TString
std::istream& operator»(std::istream &in, TString &a)	Перегрузка ввода для TString
std::ostream& operator«(std::ostream &out, const TString &a)	Перегрузка вывода для TString
struct TElem	
TString key unsigned long long str	Ключ и значение
bool operator<(const TElem &elem)	Перегрузка оператора < для TElem
bool operator>(const TElem &elem)	Перегрузка оператора > для TElem
bool operator==(const TElem &elem)	Перегрузка оператора == для TElem
struct TList	
TElem mas[Length2] int masSize = 0 TList * ref[Length2 + 1] int refSize = 0	Элементы структуры struct TList: Массив с элементами, с ссылками и размеры массивов в текущий момент
void PushBackMas(const TElem &elem) void PushBackRef(TList * list)	Функции для вставки элемента в массив
void PushBackMasSort(const TElem &elem)	Функция для вставки в массив в правильном порядке
void ClearMas() void ClearRef()	Функции обнуления размера массивов
bool FindMas(const TElem &elem, int &index)	Функуия поиска элемента в массиве

bool DelMas(const TElem &elem) bool DelRef(TList * list)	Функции для удаления элемента в массивах
class TBTree	
TList * root	Корень дерева
bool Search(const TString &key, TList * list, bool print)	Метод поиска элеметна в дереве
void Insert(const TString &key, const unsigned long long &str, TList * list)	Метод вставки элемента в дерево
void Del(TString &key, TList * list)	Метод удаления элемета в дереве
void DeleteTree(TList * list)	Метод удаления всего дерева
void Save(TList * list, FILE * f)	Метод сохранения дерева в файл
void Tree(TList * list)	Метод для удланения старого дерева и присваивания нового корня
Дополнительные функции	
TList * Load(FILE * f)	Функция выгрузки дерева(корня) из файлва
void Help(TString * str)	Функция перевода в нижний регистр для коючей

```
1  int main() {
2      std::ios::sync_with_stdio(false);
3      TBTree tree;
4      TString a;
5      FILE * f;
6      unsigned long long c = 0;
7      while (std::cin >> a) {
8          if (a == "+") {
9              std::cin >> a >> c;
10             Help(&a);
11             tree.Insert(a, c);
12         } else if (a == "-") {
13             std::cin >> a;
14             Help(&a);
15             tree.Del(a);
16         } else if (a == "!") {
17             std::cin >> a;
18             if (a == "Save") {
19                 std::cin >> a;
20                 f = fopen(a.str, "wb");
21                 if (f == NULL) {
22                     std::cout << "ERROR: error opening file" << '\n';
23                 } else {
24                     tree.Save(f);
```

```
25                  std::cout << "OK" << '\n';
26                  fclose(f);
27              }
28          } else {
29              std::cin >> a;
30              f = fopen(a.str, "rb");
31              if (f == NULL) {
32                  std::cout << "ERROR: error opening file" << '\n';
33              } else {
34                  tree.Tree(Load(f));
35                  std::cout << "OK" << '\n';
36                  fclose(f);
37              }
38          }
39      } else {
40          Help(&a);
41          tree.Search(a, true);
42      }
43  }
44  return 0;
45 }
```

3 Консоль

Входные данные:
+ a 1
+ A 2
+ aa 18446744073709551615
aa
A
- A
a

```
Lera:BTree2 valeriabudnikova$ make l
g++ -std=c++11 -g -O2 -pedantic -Wall -Werror l2.cpp -o btree
Lera:BTree2 valeriabudnikova$ ./btree <01
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
```

4 Тест производительности

Сравним время работы вставки, удаления и поиска моей реализации B-Дерева с std::map, сключи и значания одинакового типа в структурах.

1000 строк:

```
Lera:BTree2 valeriabudnikova$ ./btree <0.txt
Time BTree: 0.012599 s
Lera:BTree2 valeriabudnikova$ ./map <0.txt
Time map: 0.007602 s
```

100000 строк:

```
Lera:BTree2 valeriabudnikova$ ./btree <1.txt
Time BTree: 1.6482 s
Lera:BTree2 valeriabudnikova$ ./map <1.txt
Time map: 1.1902 s
```

Как можно увидить, std::map опережает мое B-Дерева. Это происходит потому, что std::map представлено в виде красно-черново дерева, которое при вставке и удалении делает меньше копирований элементов и «переподвязки» ссылок. В моем алоритме при разгрузке узла делается кописование и переподвязка в два новых узла(или листа).

5 Выводы

Выполнив первую лабораторную работу по курсу «Дискретный анализ», я научилась реализовывать B-Дерево, операции вставки в него элементов, удаления элемента из дерева, поиска элемента по ключу. Так как в моей реализации в узле B-Дерева не хранится указательна родителя, то было трудоемко реализовывать удаление и вставку, так как каждый раз приходилось проверять потомка и потомков потомка, чтобы предотвратить нарушение структуры B-Дерева. Одной из трудностей была работа с указателями, так как при объединении узлов, или наоборот разделении требовалось либо освобождать память, либо выделять, при этом следить за чем, какие указатели указывают на потомков, а какие на уже не нужную помять. Самым трудным для меня оказалось реализовать удаление, так как необходимо проверить достаточное количество условий. Мне очень помогло представления дерева на бумаге, при реализации удаления и вставки удобно каждый шаг рисовать на бумаге, понимать, что происходить, а потом реализовывать этот шаг в программе.

Список литературы

[1] *B-дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/B-дерево` (дата обращения: 07.12.2020).

[2] 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.))