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: И. О. Ильин
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-19

Дата: 27.11.2020
Оценка:
Подпись:

Москва, 2021

Лабораторная работа №1

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.

Структура данных: AVL-дерево.

1 Описание

Необходимо написать собственную реализацию AVL-дерева с возможностью сохранения в файл и загрузки из него, а также произвести сравнение скорости работы с *std::map*. Ключами являются строки, длинной до 256 символов, а значениями – числа от 0 до $2^{64} - 1$.

Как сказано в [1]: «AVL-дерево представляет собой бинарное дерево поиска со сбалансированной высотой: для каждого узла x высота левого и правого поддеревьев x отличается не более чем на 1».

Для того, чтобы данное условие выполнялось, необходимо при вставке и удалении вычислять баланс узла – разность между высотой левого поддерева и высотой правого поддерева, и при необходимости производить перебалансировку дерева при помощи поворотов. При этом, вычисление баланса узла и его перебалансировка выполняются за константное время.

В процессе поиска, вставки и удаления в AVL-дереве рассматриваются $O(h)$ вершин, где h – высота дерева. Также не более чем для $O(h)$ вершин может потребоваться балансировка. Тогда перечисленные операции будут производится за $O(log_2 n)$ операций, где n – кол-во элементов в дереве.

Для сохранения и загрузки дерева будем использовать КЛП-обход в глубину. Таким образом, данные операции будут выполнены за $O(n)$.

2 Исходный код

string.hpp	
TString()	Конструктор по умолчанию
TString(const char* str)	Конструктор от массива char
TString(const TString& str)	Конструктор копирования
TString(char* & str, size$_t s, size_t cap$)	Конструктор, выполняющий перемещение массива char в объект
TString(TString&& str) noexcept	Конструктор перемещения от другой строки
$\sim TString()$	Деструктор
char* begin() const	Итератор на начало
char* end() const	Итератор за конец
const char* begin() const	Константный итератор на начало
const char* end() const	Константный итератор за конец
void Swap(TString& str)	Метод, производящий обмен данными между двумя строками
size_t Size() const	Метод получения размера строки
const char* Buffer() const	Метод получения поля, содержащего массив char
void ToUpperCase()	Метод, приводящий строку в верхнему регистру
char& operator[ ](size_t idx)	Метод получения ссылки на элемент по индексу
const char& operator[ ](size_t idx) const	Метод получения константной ссылки на элемент по индексу
TString& operator=(TString const& another)	Оператор копирующего присваивания
TString& operator=(TString&& another) noexcept	Оператор перемещения
friend std::istream& operator»(...)	Оператор ввода из потока
friend std::ostream& operator«(...)	Оператор вывода в поток
bool operator<(...) bool operator>=(...) bool operator>(...) bool operator<=(...) bool operator==(...) bool operator!=(...)	Операторы сравнения

```cpp
#pragma once

#include <iostream>
#include <cstring>

class TString {
private:
    char* buffer;
    size_t size;
    size_t capacity;

public:
    TString();
    TString(const char* str);
    TString(const TString& str);
    TString(char* & str, size_t s, size_t cap);
    TString(TString&& str) noexcept;
    ~TString();

    char* begin();
    char* end();
    const char* begin() const;
    const char* end() const;

    void Swap(TString& str);
    size_t Size() const;
    const char* Buffer() const;

    void ToUpperCase();

    char& operator[](size_t idx);
    const char& operator[](size_t idx) const;
    TString& operator=(TString const& another);
    TString& operator=(TString&& another) noexcept;

    friend std::ostream& operator<<(std::ostream& os, const TString& lhs);
  friend std::istream& operator>>(std::istream& is, TString& lhs);
};

bool operator<(const TString& lhs, const TString& rhs);
bool operator>(const TString& lhs, const TString& rhs);
bool operator==(const TString& lhs, const TString& rhs);
bool operator!=(const TString& lhs, const TString& rhs);
```

avl.hpp	
TAvlTree()	Конструктор дерева
$\sim TAvlTree()$	Деструктор

int GetHeight(const TAvlNode* node) const	Высота узла node
int GetBalance(const TAvlNode* node) const	Возвращает баланс узла
TAvlNode* RotateRight(TAvlNode* node)	Правый поворот относительно узла node
TAvlNode* RotateLeft(TAvlNode* node)	Левый поворот относительно узла node
TAvlNode* Balance(TAvlNode* node)	Выполняет банасировку узла
void RecountHeight(TAvlNode* node)	Пересчитывает высоту в узле
TAvlNode* InsertInNode(TAvlNode* node, TString key, uint64_t value, const bool& printResult)	Рекурсивная вставка узла в дерево
TAvlNode* RemoveMin(TAvlNode* node, TAvlNode* curr)	Удаляет мин. вершину из правого поддерева curr дерева node и меняет вершину node на мин. вершину, возвращает правое поддерево node
TAvlNode* SubRemove(TAvlNode* node, const TString& key, const bool& printResult)	Рекурсивный метод удаления вершины с ключом key из дерева node
void SubDeleteTree(TAvlNode* node)	Рекурсивное удаление дерева
void SubSave(std::ostream& os, const TAvlNode* node)	Рекурсивная функция сохранения
TAvlNode* SubLoad(std::istream& is)	Рекурсивная загрузка из файла
void DfsPrint(const TAvlNode* node, const int& depth) const	Рекурсивная печать дерева
TAvlNode* Find(const TString& key) const	Поиск узла по ключу
void Insert(TString key, uint64_t value, const bool& printResult)	Вставка узла в дерево
void Remove(const TString& key, const bool& printResult)	Удаление из дерева узла с ключом key
void DeleteTree()	Удаление дерева
void Save(const TString& path, const bool& printResult)	Сохранение дерева в файл
void Load(const TString& path, const bool& printResult)	Загрузка дерева из файла
void Print() const	Печать дерева

```cpp
#pragma once

#include "string.hpp"

struct TAvlNode {
    TString key;
    uint64_t value;
    int height;

    TAvlNode* left;
    TAvlNode* right;

    TAvlNode();
    TAvlNode(TString k, uint64_t val, int h = 1);
    ~TAvlNode();
};

class TAvlTree {
private:
    TAvlNode* root;

    int GetHeight(const TAvlNode* node) const;
    int GetBalance(const TAvlNode* node) const;

    TAvlNode* RotateRight(TAvlNode* node);
    TAvlNode* RotateLeft(TAvlNode* node);
    TAvlNode* Balance(TAvlNode* node);
    void RecountHeight(TAvlNode* node);

    TAvlNode* InsertInNode(TAvlNode* node, TString key, uint64_t value, const bool&
        printResult);
    TAvlNode* RemoveMin(TAvlNode* node, TAvlNode* curr);
    TAvlNode* SubRemove(TAvlNode* node, const TString& key, const bool& printResult);
    void SubDeleteTree(TAvlNode* node);
    void SubSave(std::ostream& os, const TAvlNode* node);
    TAvlNode* SubLoad(std::istream& is);
    void DfsPrint(const TAvlNode* node, const int& depth) const;

public:
    TAvlTree();
    ~TAvlTree();
    TAvlNode* Find(const TString& key) const;
    void Insert(TString key, u_int64_t value, const bool& printResult);
    void Remove(const TString& key, const bool& printResult);
    void DeleteTree();
    void Save(const TString& path, const bool& printResult);
    void Load(const TString& path, const bool& printResult);
    void Print() const;
};
```

3 Консоль

```
MacBook-Pro:da_lab_02 mr-ilin$ make all
g++ -std=c++17 -pedantic -Wall -O2 -c main.cpp -o main.o
g++ -std=c++17 -pedantic -Wall -O2 -c avl.cpp -o avl.o
g++ -std=c++17 -pedantic -Wall -O2 -c string.cpp -o string.o
g++ -std=c++17 -pedantic -Wall -O2 main.o avl.o string.o -o solution
MacBook-Pro:da_lab_02 mr-ilin$ ./solution
+ a 1
OK
+ A 2
Exist
+ aa 18446744073709551615
OK
aa
OK: 18446744073709551615
A
OK: 1
-A
OK
a
```

4 Тест производительности

Тест производительности представляет собой вставку 1 млн случайных пар ключ - значение, поиск каждого элемента и удаление, причем порядок элементов при вставке, поиске и удалении различный. Будем сравнивать описанную реализацию AVL-дерева со словарем *std::map*.

```
MacBook-Pro:da_lab_02 mr-ilin$ make benchmark
g++ -std=c++17 -pedantic -Wall -O2 -c benchmark.cpp -o benchmark.o
g++ -std=c++17 -pedantic -Wall -O2 -c string.cpp -o string.o
g++ -std=c++17 -pedantic -Wall -O2 -c avl.cpp -o avl.o
g++ -std=c++17 -pedantic -Wall -O2 benchmark.o string.o avl.o -o benchmark
MacBook-Pro:da_lab_02 mr-ilin$ ./benchmark <tests/one_million_shuffled.t
----------------------------
Insert std::map time = 0.420017s
Insert avl tree time = 0.496773s

Search std::map time = 0.277718s
Search avl tree time = 0.265392s

Delete std::map time = 0.352363s
Delete avl tree time = 0.577042s
----------------------------
```

Можем сделать вывод, что написанная реализация немного уступает при вставке и удалении *std::map*, но выигрывает в поиске.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я многому научился.

Разобрался с AVL-деревом, поворотами и балансировкой. Также, во время поиска утечек памяти, понял свои ошибки в выделении и освобождении памяти, соответственно научился использовать *valgrind* для их поиска и устранения. Научился использовать *std::move( )* для перемещения данных (позволил оптимизировать вставку узла в дерево).

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] *АВЛ-дерево — Вики университета ITMO.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево` (дата обращения: 20.11.2020).