

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: Е. С. Пищик
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б
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 $2^{64} - 1$. 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 параметром 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, отсутствие прав записи и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.

Вариант дерева: AVL-дерево.

1 Описание

Требуется написать реализацию AVL-дерева. AVL-дерево - дерево, каждый узел которого соответствует условию: модуль разности высоты левого поддерева и правого поддерева ≤ 1 . Чтобы поддерживать данное условие нужно уметь считать баланс каждого узла, баланс = высота левого поддерева - высота правого поддерева, при вставке, удалении требуется учитывать условие AVL-дерева и при нарушении делать перебалансировку дерева при помощи поворотов.

2 Исходный код

TAvl.cpp	
TAvlNode* TAvl::RotateLeft(TAvlNode* node)	Левый поворот
TAvlNode* TAvl::RotateRight(TAvlNode* node)	Правый поворот
TAvlNode* TAvl::RotateRightLeft(TAvlNode* node)	Правый-левый поворот
TAvlNode* TAvl::RotateLeftRight(TAvlNode* node)	Левый-правый поворот
TAvlNode* TAvl::ReBalance(TAvlNode* node)	Перебалансировка
TAvlNode* TAvl::InsertPrint(TAvlNode* node, TData k, uint64_t v, bool const& benchmark_flag)	Вставка в дерево и печать результата
TAvlNode* TAvl::RemoveMin(TAvlNode* node, TAvlNode* tmp)	Удаление минимального
TAvlNode* TAvl::RemovePrint(TAvlNode* node, TData k, bool const& benchmark_flag)	Удаление из дерева и печать результата
TAvlNode* TAvl::Find(TAvlNode* node, TData k)	Поиск в дереве
void TAvl::TreeDelete(TAvlNode* node)	Удаление дерева
TAvlFinal.cpp	
void TAvlFinal::Upper(TData& str)	Приведение строки к верхнему регистру
void TAvlFinal::Save(std::ostream& os, TAvlNode const* node)	Сохранение дерева
TAvlNode* TAvlFinal::Load(std::istream& is, TAvlNode const* node)	Загрузка дерева
bool TAvlFinal::FileSave(TData& fileName)	Сохранение дерева в файл
bool TAvlFinal::FileLoad(TData& fileName)	Загрузка дерева из файла
void TAvlFinal::DInsert()	Финальная функция вставки в дерево
void TAvlFinal::DRemove()	Финальная функция удаления из дерева
void TAvlFinal::DFind(TData const& k)	Финальная функция поиска в дереве
void TAvlFinal::LoadSave()	Финальная функция для сохранения/загрузки дерева

TData.cpp	
TData::TData()	Конструктор по умолчанию
TData::TData(char const* str)	Конструктор с одним параметром
TData::TData(TData const& str)	Конструктор копирования
TData& TData::operator=(TData const& str)	Оператор присваивания
void TData::Move(char* str)	Перемещение данных
void TData::Swap(TData& str)	Обмен данных
void TData::PushBack(char const& symb)	Добавление в конец вектора
char* TData::end()	Указатель за конец вектора, неконстантная версия
char const* TData::end() const	Указатель за конец вектора, константная версия
std::ostream& operator<<(std::ostream& os, TData const& str)	Перегрузка оператора вывода
std::istream& operator>>(std::istream& is, TData& str)	Перегрузка оператора ввода
bool operator<(TData const& lhs, TData const& rhs)	Перегрузка оператора <
bool operator>(TData const& lhs, TData const& rhs)	Перегрузка оператора >
bool operator==(TData const& lhs, TData const& rhs)	Перегрузка оператора ==
bool operator!=(TData const& lhs, TData const& rhs)	Перегрузка оператора !=

```

1 struct TAvlNode
2 {
3     TData key_;
4     uint64_t val_;
5     uint64_t h_;
6     TAvlNode* l_;
7     TAvlNode* r_;
8     TAvlNode() : key_(), val_(), h_(1), l_(nullptr), r_(nullptr) {};
9     TAvlNode(TData key, uint64_t val) : key_(key), val_(val), h_(1),
10    l_(nullptr), r_(nullptr) {};
11 };
12
13 struct TAvl
14 {
15     TAvlNode* root_;
16     uint64_t Height(const TAvlNode* node) {return node != nullptr ? node->h_ : 0;}
17     uint64_t Balance(const TAvlNode* node) {return Height(node->l_) - Height(node->r_);}
18     void ReHeight(TAvlNode* node) {node->h_ = std::max(Height(node->l_),
19     Height(node->r_)) + 1;}
20     TAvlNode* RotateLeft(TAvlNode*);

```

```

21     TAvlNode* RotateRight(TAvlNode*);
22     TAvlNode* RotateRightLeft(TAvlNode*);
23     TAvlNode* RotateLeftRight(TAvlNode*);
24     TAvlNode* ReBalance(TAvlNode*);
25     TAvlNode* InsertPrint(TAvlNode*, TData, uint64_t);
26     TAvlNode* RemoveMin(TAvlNode*, TAvlNode*);
27     TAvlNode* RemovePrint(TAvlNode*, TData, bool const&);
28     TAvlNode* Find(TAvlNode*, TData k);
29     void TreeDelete(TAvlNode*);
30     TAvl() : root_(nullptr) {};
31     void InsPrint(TData k, uint64_t v) {root_ = InsertPrint(root_, std::move(k), v,
        false);}
32     void DeletePrint(TData k) {root_ = RemovePrint(root_, std::move(k), false);}
33     TAvlNode* Find(TData k) {return Find(root_, std::move(k));}
34     ~TAvl() {TreeDelete(root_);}
35 };
36
37 struct TAvlFinal : public TAvl
38 {
39     private:
40         void Upper(TData&);
41         void Save(std::ostream&, TAvlNode const*);
42         TAvlNode* Load(std::istream&, TAvlNode const*);
43         bool FileSave(TData&);
44         bool FileLoad(TData&);
45     public:
46         void DInsert();
47         void DRemove();
48         void DFind(TData const&);
49         void LoadSave();
50 };
51
52 class TData
53 {
54     private:
55         size_t cap_;
56         size_t size_;
57         char* data_;
58     public:
59         TData();
60         TData(char const*);
61         TData(TData const&);
62         ~TData() { delete[] this->data_; }
63         TData& operator=(TData const&);
64         void Move(char*);
65         void Swap(TData&);
66         void PushBack(char const&);
67         char* begin() { return this->data_; }
68         char const* begin() const { return this->data_; }

```

```

69 |     char* end();
70 |     char const* end() const;
71 |     size_t Size() const { return this->size_; }
72 |     char const* Data() const { return this->data_; }
73 |     char& operator[](size_t ind) { return this->data_[ind]; }
74 |     char const& operator[](size_t ind) const { return this->data_[ind]; }
75 |
76 |     friend std::ostream& operator<<(std::ostream&, TData const&);
77 |     friend std::istream& operator>>(std::istream&, TData&);
78 | };

```

3 Консоль

```
pe4eniks$ ./solution
+ word 34
OK
word
OK: 34
-word
OK
word
NoSuchWord
+ till 35
OK
+ till 67
Exist
! Save tree.bin
OK
till
OK: 35
-till
OK
till
NoSuchWord
! Load tree.bin
OK
till
OK: 35
-till
OK
till
NoSuchWord
```


4 Тест производительности

Тест производительности представляет из себя следующее: создаем объекты `std::map` и наш `avl`. Вставляем в оба объекта по 1 млн. элементов с ключом=значению в диапазоне от 0 до 999999. Измеряем время работы для `std::map` и `avl`. Далее 1 млн. раз ищем элемент с ключом=значению=999999 и замеряем время для `std::map` и `avl`. Последний тест - 1 млн. раз удаляем значение(от 999999 до 0) из `std::map` и `avl`, замеряем время.

```
pe4eniks$ ./benchmark
Insert map time: 6.99499 seconds
Insert avl time: 12.1846 seconds
Find map time: 7.5032 seconds
Find avl time: 3.30145 seconds
Delete map time: 10.0912 seconds
Delete avl time: 9.73254 seconds
```

Как видно, что удаление в `avl` работает совсем чуть-чуть быстрее чем в `std::map`, вставка в `avl` работает значительно более медленно, чем в `std::map`, поиск в `avl` работает значительно быстрее чем в `std::map`.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я познакомился с различными структурами данных, как AVL-дерево, RB-дерево, Декартово-дерево, B-дерево, Patricia. Научился работать с AVL-деревом. Такие структуры данных хорошо подходят для хранения и обработки большого объема данных, т.к. поиск, вставка и удаление делаются за $O(\log n)$. Также важно знать как устроены эти структуры, чтобы понимать как работают некоторые стандартные контейнеры, например: `std::map` использует внутри RB-дерево.

Список литературы

- [1] AVL-деревья Habr
- [2] AVL-дерево wiki