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. Почечура
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2022

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: `AVL-дерево`.

1 Описание

Требуется написать реализацию $AVL-$дерева. Идея его заключается в том, что у каждой вершины разница высот между левым и правым деревом по модулю не превосходит единицы. Также это дерево является бинарным деревом поиска. При вставке и удалении элементов всегда нужно проверять баланс вершин, у которых он мог измениться, и при необходимости делать балансировку с помощью поворотов. Все действия очень похожи на действия с бинарным деревом.

2 Исходный код

Все функции схожи с функциями для бинарного дерева поиска. Отличием является балансировка, которая требует дополнительного написания функций и внесения новых шагов в функции. Также в структуру ноды я добавил поле для баланса и ссылку на предка, чтобы было легче совершать балансировку. Алгоритм поворотов взят прямиком из книги Кормена: в зависимости от типа поворота (правый или левый) левый или правый ребёнок текущей ноды становится на её место, а нода занимает место ребёнка (функции $LeftRotate$ и $RightRotate$). Повороты используются в функции $BalancingForInsert$, когда в ноде баланс равен 2 или -2. После произведённых поворотов баланс становится нулём и дерево снова может считаться $AVL-$деревом. В добавлении (функция $Insert$) я производил изменение баланса только после того, как вставил элемент, поэтому взаимодействие с балансами идёт через родителей. Условие остановки ребалансировки для добавления: баланс одной из нод равен нулю, либо дошли до корня. В это случае возвращаем $true$ и больше балансы не трогаем. Примерно тот же алгоритм и в удалении. Только здесь я уже по ходу поиска ноды я меняю балансы, и при необходимости возвращаю их назад (если ниже баланс уже скорректирован). Такой подход связан с множествами особенностей удаления. В $BalancingForDelete$ мы рассматриваем больше случаев, чем в $BalancingForInsert$(опять же, из-за особенностей удаления). В функции $GoingDeeper$ идёт поиск замены текущей ноде (самый левый элемент правого дерева). Там также происходит ребалансировка, и если выполнены условия конца ребалансировки (для удаления это: стоим в корне, либо баланс текущей ноды равен 1 или -1) возвращаем $true$ и больше балансы не трогаем. В функции $Save$ открываем файл с указанным названием на вывод и с помощью очереди печатаем всё дерево в файл (ключи и значение). Псевдо-бфс. В функции $Load$ мы открываем файл с указанным названием на поток ввода, считываем данные и добавляем их в дерево. Перед считываем и добавлением чистим всё дерево.

lab2.cpp	
void DestroyTAVLTreeNode(TAVLTree Node *x)	Удаление ноды дерева
void DestroyTAVLTree(TAVLTreeNode *x)	Удаление всего дерева
TAVLTree::∼TAVLTree()	Деструктор дерева
void TAVLTree::LeftRotate(TAVLTree Node *x)	Левый поворот
void TAVLTree::RightRotate(TAVLTree Node *y)	Правый поворот
void TAVLTree::Insert(string key, unsigned long long value, bool &status)	Добавление в дерево

bool TAVLTree::BalancingFor Insert(TAVLTreeNode *node)	Балансировка для добавления
bool TAVLTree::Insert(string key, unsigned long long value, TAVLTreeNode *node, bool &status)	Добавление в дерево элемента, если корень уже есть
void PrintAVLNodeElems(TAVLTreeNode *x, int h)	Печать всех элементов дерева
void TAVLTree::Print()	Печать дерева
bool TAVLTree::Search(string key, TAVLTreeNode * node)	Поиск ключа в дереве
void TAVLTree::Search(string key)	Поиск
bool TAVLTree::BalancingForDelete (TAVLTreeNode* node)	Балансировка для удаления
bool TAVLTree::GoingDeeper(TAVLTree Node *node, string& key, unsigned long long& value)	Поиск замены удаляемой ноде
void TAVLTree::Delete(string key)	Удаление ноды
bool TAVLTree::Delete(string key, TAVLTreeNode *node)	Удаление ноды, если она не корень
void TAVLTree::Save(string filename)	Запись дерева в файл
void TAVLTree::Load(string filename)	Загрузка дерева из файла

```
1  struct TAVLTreeNode {
2      string key;
3      unsigned long long value;
4      short balance;
5      TAVLTreeNode *left;
6      TAVLTreeNode *right;
7      TAVLTreeNode *parent;
8      TAVLTreeNode(string key1, unsigned long long value1, TAVLTreeNode *left1,
           TAVLTreeNode *right1, TAVLTreeNode *parent1, short balance1) {
9          key = key1;
10         value = value1;
11         left = left1;
12         right = right1;
13         parent = parent1;
14         balance = balance1;
15     }
16 };
17
18 class TAVLTree {
19 public:
20     TAVLTree() {}
21     ~TAVLTree();
```

```
    void Insert(string key, unsigned long long value, bool &status);
    void Search(string key);
    void Delete(string key);
    void Print();
    void Save(string filename);
    void Load(string filename);
private:
    bool Insert(string key, unsigned long long value, TAVLTreeNode * node, bool &status);
    bool Search(string key, TAVLTreeNode * node );
    void LeftRotate(TAVLTreeNode *node);
    void RightRotate(TAVLTreeNode *node);
    bool BalancingForInsert(TAVLTreeNode *node);
    bool BalancingForDelete(TAVLTreeNode *node);
    bool GoingDeeper(TAVLTreeNode* node, string& key, unsigned long long& value);
    bool Delete(string key, TAVLTreeNode * node);
private:
    TAVLTreeNode *root = nullptr;
};
```

3 Консоль

```
root@DESKTOP-5HM2HTK:~# cat <test
+ a 1
+ A 2
+ aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
18446744073709551615
aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
A
-A
a
root@DESKTOP-5HM2HTK:~# g++ lab2.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <test
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
root@DESKTOP-5HM2HTK:~#
```

4 Тест производительности

```
root@DESKTOP-5HM2HTK:~# g++ -pedantic -Wall benchmark.cpp
root@DESKTOP-5HM2HTK:~# ./a.out <tests/1.t
Count of lines 100000
map ms=106756
avl ms=110746
root@DESKTOP-5HM2HTK:~#
```

Как видно, $AVL-$дерево работает примерно за то же время, что и map, т.к. это структуры, использующие практически один и тот же алгоритм (как и красно-чёрное дерево). Длина пути по элементам в данных структурах при выполнении определённой операции не привысит логарифма от количества всех элементов. Различие могло получится лишь из-за неидельного написания $AVL-$дерева.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я разобрался, как работает $AVL-$дерево и научился его реализовывать. Поворот - довольно интеерсное действие, которое нашло применение в балансировке $AVL-$дерева. Придумывать алгоритм для балансировки и реализовывать его было очень затягивающе. Попрактиковался работать с указателями в языке $C++$.

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] *AVL-дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/АВЛ-дерево` (дата обращения: 19.04.2022).