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: Я. С. Поскряков
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2019

Лабораторная работа №2

Задача:

Требуется 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 длинной 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 дожна вывести строку «OK», усли 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.
- $! \; Load \; /path/to/file$ – загрузить словарь из файла. Предполагается, что файл был заранее 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.

Вариант дерева: Красно-чёрное дерево.

1 Описание

Требуется написать реализацию красно-чёрного дерева. Красно-черное дерево является одним из самобалансирующихся двоичных деревьев. Используется для организации сравнимых данных, таких как тексты, числа. Красно-чёрное дерево - двочиное дерево поиска, в котором каждый элемент может быть либо красным, либо черным. При этом, корень у данного дерева всегда черный, листья также считаются черными.

1. Узел либо красный, либо чёрный.
2. Корень — чёрный. (В других определениях это правило иногда опускается. Это правило слабо влияет на анализ, так как корень всегда может быть изменен с красного на чёрный, но не обязательно наоборот).
3. Все листья (NIL) — чёрные.
4. Оба потомка каждого красного узла — чёрные.
5. Всякий простой путь от данного узла до любого листового узла, являющегося его потомком, содержит одинаковое число чёрных узлов.

Взглянем на операции, проводимые с Красно-чёрным деревом:

1. Поиск. Такой же, как и в обычном бинарном дереве поиска.
2. Вставка. Вставка нового элемента производится только в листья ровно тем же самым способом, что и вставка в бинарное дерево поиска. Затем производится восстановительная ребалансировка красно-чёрного дерева.
3. Удаление. При удалении элемента с нелистовыми потомками, мы ищем либо минимальный в правом поддереве, либо максимальный в левом, и ставим найденный элемент на место удаляемого. Затем выполняем восстановительную ребалансировку дерева.
4. Запись в файл. Начиная с корня: затем размер, затем пары ключ-значение, все данные разделяются пробелами. Далее данная операция проиводится для левого, а затем правого поддерева
5. Чтение из файла. Вводим данные в том же порядке, в котором они были записаны предыдущей операцией, таким образом дерево восстанавливается ровно с той же структурой.
6. Алгоритм ребалансировки дерева.

2 Исходный код

Входные данные, имеющие описанные выше формат, интерпретируются, после чего вызывается одна из пяти функций, соответствующая запросу. Функции возвращают численный результат операции, по которому формируется вывод.

Класс $TRBtree$:

```
class TNode {
public:
    TNode() {};
    TNode(bool ecolor, unsigned long long int eval, char *ekey) {
        leftson = nullptr;
        rightson = nullptr;
        parent = nullptr;
        color = ecolor;
        value = eval;
        key = ekey;
    }
    ~TNode() {
        delete[] key;
        key = nullptr;
        parent = nullptr;
        leftson = nullptr;
        rightson = nullptr;
    }
    TNode * leftson;
    TNode * rightson;
    TNode * parent;
    bool color; // 0 == RED, 1 == BLACK
    char * key;
    unsigned long long int value;
};

class TRbTree {
public:
    TNode * root;
    TNode * nil;
    TRbTree() {
        nil = new TNode(1,0,nullptr);
        nil->color = 1;
        nil->key = 0;
        nil->leftson = nullptr;
        nil->rightson = nullptr;
        nil->value = 0;
        nil->parent = nullptr;
        root = nil;
    }

    void LeftRotate(TNode * x);
```

```
43      void RightRotate(TNode * x);
44      void Push(TNode * elem_new);
45      void FixPush(TNode * current);
46      void Print_Tree(TNode * p, int level);
47      TNode * FindMin(TNode * current);
48      TNode * Find(char * s);
49      int Delete(TNode * z);
50      void FixDelete(TNode * x);
51      void Translplant(TNode *u, TNode * v);
52
53      void Save(char * name);
54      void Save(std::ofstream& output, TNode* root);
55      void Reborn();
56      void Load(char * name);
57      void Load(std::ifstream& input, TNode *& root);
58
59  };
```

3 Консоль

Тесты генерировались следующим кодом на языке *Python*.

```
# -*- coding: utf-8 -*-
import sys
import random
import string

def get_random_key():
    return random.choice( string.ascii_letters )

if __name__ == "__main__":
    if len(sys.argv) != 2:
        print( "Usage: {0} <count of tests>".format( sys.argv[0] ) )
        sys.exit(1)

    count_of_tests = int( sys.argv[1] )

    actions = [ "+", "?" ]

    for enum in range( count_of_tests ):
        keys = dict()
        test_file_name = "tests/{:02d}".format( enum + 1 )
        with open( "{0}t.txt".format( test_file_name ), 'w' ) as output_file, \
             open( "{0}a.txt".format( test_file_name ), "w" ) as answer_file:
            for _ in range( random.randint(0, 100000) ):
                action = random.choice( actions )
                if action == "+":
                    key = get_random_key()
                    value = random.randint( 0, 100000000000000000 )
                    output_file.write("+ {0} {1}\n".format( key, value ))
                    key = key.lower()
                    answer = "Exist"
                    if key not in keys:
                        answer = "OK"
                        keys[key] = value
                    answer_file.write( "{0}\n".format( answer ) )
                elif action == "?":
                    search_exist_element = random.choice( [ True, False ] )
                    key = random.choice( [ key for key in keys.keys() ] ) if
                        search_exist_element and len( keys.keys() ) > 0 else
                        get_random_key()
                    output_file.write( "{0}\n".format( key ) )
                    key = key.lower()
                    if key in keys:
                        answer = "OK: {0}".format( keys[key] )
                    else:
                        answer = "NoSuchWord"
                    answer_file.write( "{0}\n".format( answer ) )
```

```
yar@ubuntu:~$ clang++ -pedantic -Wall -std=c++14 -Werror -Wno-sign-compare
-lm main.c -o main --some_long_argument=true
yar@ubuntu:~$ cat 01t.txt
Q
C
P
+ L 1371285559862681
+ l 10513263494525964
+ F 72038700587376717
+ h 42937126614126041
L
l
+ K 78247266927102354
yar@ubuntu:~$ ./main <01t.txt
NoSuchWord
NoSuchWord
NoSuchWord
OK
Exist
OK
OK
OK: 1371285559862681
OK: 1371285559862681
OK
yar@ubuntu:~$ cat 02t.txt
K
l
v
+ V 23646955705854650
K
+ k 42666273745435743
+ u 93883882144435257
v
u
+ r 86289566224724661
yar@ubuntu:~$ ./main <02t.txt
NoSuchWord
NoSuchWord
NoSuchWord
OK
NoSuchWord
```

```
OK
OK
OK: 23646955705854650
OK: 93883882144435257
OK
OK: 42666273745435743
```

4 Тест производительности

В тесте производительности сравнивается время работы написанного красно-чёрного дерева и стандартного контейнера map на 5 тестах в 1000000 строк, сгенерированных описанной выше программой.

```
Тест 1:
std::map    6080 ms
RBtree      6160 ms

Тест 2:
std::map    6080 ms
RBtree      5980 ms

Тест 3:
std::map    5950 ms
RBtree      6000 ms

Тест 4:
std::map    6500 ms
RBtree      6460 ms

Тест 5:
std::map    6570 ms
RBtree      6680 ms
```

5 Выводы

Для выполнения второй лабораторной работы по курсу «Дискретного анализа», я изучил большое количество литературы по теме красно-чёрных деревьев. Стоит отметить, что красно-чёрное дерево используется для реализации ассоциативных массивов в большинстве библиотек. Также, красно-чёрное дерево использует планировщик процессов в Linux, деревья заменяют очереди очередей, которые имеют приоритеты для процессов в очереди для планировщика. Если сравнивать красно-чёрное дерево с AVL, то получится следующее :

1. Поиск. Поскольку красно-чёрное дерево, в худшем случае, выше, поиск в нём медленнее.

2. Вставка. Вставка требует до 2 поворотов в обоих видах деревьев. Однако из-за большей высоты красно-чёрного дерева вставка может занимать больше времени.

3. Удаление. Удаление из красно-чёрного дерева требует до 3 поворотов, в АВЛ-дереве оно может потребовать числа поворотов до глубины дерева (до корня). Поэтому удаление из красно-чёрного дерева быстрее, чем из АВЛ-дерева.

4. Память. АВЛ-дерево в каждом узле хранит разницу высот. Красно-чёрное дерево в каждом узле хранит цвет (1 бит). Таким образом, красно-чёрное дерево может быть экономичнее. (Правда если учитывать, что в современных вычислительных системах память выделяется кратно байтам, то деревья абсолютно одинаковы)

Таким образом, красно-чёрное дерево, являющееся одним из сбалинсированных деревьев поиска, несмотря на некоторые недостатки нашло себе применение в современных реалиях.

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] *Красно-черное дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/Красно-чёрное_дерево` (дата обращения: 10.11.2018).