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. Т. Бахарев
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

Москва, 2018

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.

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 на запись и т.п.), программа должна вывести строку, начинающуюся с «ERROR:» и описывающую на английском языке возникшую ошибку.
Вариант задания: `АВЛ-дерево.`

1 Описание решения

АВЛ-дерево - дерево, у которого выстота левого поддерева любого узла отличается от высоты правого не более чем на 1. Оно является частным случаем бинарного дерева. Для чего нужно соблюдать балансировку? При определенных "плохих"данных дерево может вырождаться в линейный список. Тогда в худшем случае, добавление, поиск и удаление будут производиться за линейное время. Но если использовать балансировку, то мы будем иметь классическое дерево – дерево с большой "кроной". Такая структуризация данных уменьшит время работы с деревом – все приведенные выше операции будут производиться за $O(logn)$. Самое интересное, что балансировка работает за $O(1)$ – иными словами, мы всегда будем иметь сбалансированное дерево, вне зависимости от количества входных данных.

Опишем структуру узла дерева:

```
1 struct TNode
2 {
3     TNode* Left;
4     TNode* Right;
5     TNode* Parent;
6     char* Key;
7     unsigned long long int Number;
8     short Balance;
9 };
```

Left – указатель на левое поддерево данного узла

Right – указатель на правое поддерево данного узла

Parent – указатель на родительский узел данного узла

Key – указатель на массив, который является ключом

Number – значение ключа

Balance – баланс данного узла дерева

В целом, процедуры добавления и удаления элементов ничем не отличаются от таких же операций в бинарном дереве. Только вот в AVL-дереве после каждых действий с деревом необходимо подниматься вверх от удаленного или добавленного элемента и вычислять балансы родительских узлов. Изначально баланс равен 0. Если в какой то вершине нарушена балансировка - т.е. баланс равен 2 или -2, то производятся повороты дерева вокруг этой вершины. Перейдем к более подробному описаню алгоритма действий с AVL-деревом.

Вставка в AVL-дерево

▶ Вставляем элемент как в обычное дерево поиска

▶ Поднимаемся вверх и пересчитываем баланс: пришли из левого поддерева – прибавим 1 к балансу, из правого – вычтем 1

▶ Баланс $= 0$ – высота не изменилась – балансировка окончена

▶ Баланс $= \pm\ 1$ – высота изменилась, продолжаем подъем

▶ Баланс $= \pm\ 2$ – инвариант нарушен, делаем соответствующие повороты, если получаем новый баланс, равный 0, то останавливаемся, иначе ($\pm\ 1$) – продолжаем подъем

Удаление из AVL-дерева

▶ Удаляем элемент как в обычном дереве поиска

▶ Поднимаемся вверх от удаленного элемента и пересчитываем баланс: пришли из левого поддерева – вычтем 1 из баланса, из правого – прибавим 1

▶ Баланс $= \pm\ 1$ – высота не изменилась – балансировка окончена

▶ Баланс $= 0$ – высота уменьшилась – продолжаем

▶ Баланс $= \pm\ 2$ – инвариант нарушен, делаем соответствующие повороты, если получаем новый баланс, равный 0, то продолжаем подъем, иначе – останавливаемся

Лемма

Пусть m_h – минимальное число вершин в AVL-дереве высоты h. Тогда $m_h = F_{h+2} - 1$, где F_h – h-ое число Фибоначчи.

2 Исходный код

main.cpp	
void ToLowerCase(char* Buffer)	Выполняет преобразование прописных букв в строчные.
bool Insert(unsigned long long int, char*, TNode**, TNode**, size_t)	Функция вставки элемента. В функцию передается пара ключ/значение, текущий и родительский узлы, а так же длина ключа.
bool Search(char* str, TNode**, size_t)	Поиск строки любого размера в дереве, начиная с некоторого узла.
bool Remove(char*, TNode**, size_t)	Удаление элемента из дерева(если таковой имеется)
void LkpLoad(TNode**, TNode*, FILE*)	Сохранение поддерева, начиная с некоторого узла, в текстовый файл
void LkpSave(TNode*, FILE*)	Загрузить новое дерево взамен старого из указанного файла
avl_tree.cpp	
TAvlTree()	Конструктор АВЛ-дерева
~TAvlTree()	Деструктор АВЛ-дерева
void SimpleLeftRotate(TNode*)	Поворот дерева вокруг указанной вершины влево без пересчёта баланса.
void SimpleRightRotate(TNode*)	Поворот дерева вокруг указанной вершины вправо без пересчёта баланса
void SimpleDelete(TNode*)	Удаление поддерева, начиная с некоторого узла
void LeftRotation(TNode*)	Малое вращение влево вокруг данного узла с восстановлением баланса во всем дереве
void RightRotation(TNode*)	Малое вращение вправо вокруг данного узла с восстановлением баланса во всем дереве
void BigLeftRotation(TNode*)	Большое вращение влево вокруг данного узла с восстановлением баланса во всем дереве
void BigRightRotation(TNode*)	Большое вращение вправо вокруг данного узла с восстановлением баланса во всем дереве

void RecountBalanceInsert(TNode*, bool)	Функция для пересчета баланса при вставке. Вторым аргуметом идет информация о том, из какого дочернего узла мы пришли – левого или правого.
void RecountBalanceDelete(TNode*, bool)	Функция для пересчета баланса при удалении. Вторым аргуметом идет информация о том, из какого дочернего узла мы пришли – левого или правого.
void UpdateRoot()	Функция для обновления корня дерева. При поворотах, корень может перестать быть корнем

avl_tree.h

```
typedef struct TNode TNode;
struct TNode
{
    TNode* Left;
    TNode* Right;
    TNode* Parent;
    char* Key;
    unsigned long long int Number;
    short Balance;
};
class TAvlTree
{
public:
    TAvlTree();
    ~TAvlTree();
    int Height(TNode*);
    int CountBalance(TNode*);

    bool Insert(unsigned long long int, char*, TNode**, TNode**, size_t);
    bool Search(char* str, TNode**, size_t);
    bool Remove(char*, TNode**, size_t);

    void SimpleLeftRotate(TNode*);
    void SimpleRightRotate(TNode*);
    void SimpleDelete(TNode*);

    void LeftRotation(TNode*);
    void RightRotation(TNode*);
    void BigLeftRotation(TNode*);
    void BigRightRotation(TNode*);

    void RecountBalanceInsert(TNode*, bool);
    void RecountBalanceDelete(TNode*, bool);
```

```
34
35      void LkpLoad(TNode**, TNode*, FILE*);
36      void LkpSave(TNode*, FILE*);
37      void UpdateRoot();
38      TNode* Root;
39  private:
40  };
```

3 Консоль

```
alex$ make
g++ -std=c++11 -o diskran_laba_2 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -lm main.cpp avl_tree.cpp
alex$ cat tests/01.t
K
-d
+ vBlN 144
+ plvh 264
Y
E
-v
+ SHFi 744
a
B
alex$ ./*2 <tests/01.t
NoSuchWord
NoSuchWord
OK
OK
NoSuchWord
NoSuchWord
NoSuchWord
OK
NoSuchWord
NoSuchWord
alex$
```

4 Тест производительности

Сравним работу АВЛ-дерева и контейнера map. Ниже приведены листинги генератора тестов и использования std::map.

benchmark.cpp

```
void ToLowerCase(std::string& str)
{
    for(size_t i = 0; i != str.size(); ++i)
    {
        if(str[i] >= 'A' && str[i] <= 'Z')
            str[i] += 32;
    }
}
int main()
{
    std::map<std::string, unsigned long long int> lib;
    char command;
    std::string str;
    unsigned long long int val;
    std::cin.get(command);
    while(!std::cin.eof())
    {
        if(command == '+') // + akiufh 34
        {
            std::cin.get();// skip space
            std::cin>>str;
            ToLowerCase(str);
            std::cin.get();// skip space
            std::cin>>val;
            std::cin.get();// skip space
            auto it = lib.find(str);
            if((*it).first != str)// elem is not in lib
            {
                lib.insert(std::pair<std::string, unsigned long long int>(str, val));
                std::cout<<"OK\n";
            }
            else
            {
                std::cout<<"Exist\n";
            }
        }
        else if(command == '-') // - a
        {
            std::cin.get();// skip space
            std::cin>>str;
            ToLowerCase(str);
            std::cin.get();// skip space
            auto it = lib.find(str);
```

```
44              if((*it).first == str) //elem finded
45              {
46                  lib.erase(it);
47                  std::cout<<"OK\n";
48              }
49              else
50              {
51                  std::cout<<"NoSuchWord\n";
52              }
53          }
54          else
55          {
56              std::cin.unget();
57              std::cin>>str;
58              ToLowerCase(str);
59              std::cin.get();// skip space
60              auto it = lib.find(str);
61              if((*it).first == str)
62              {
63                  std::cout<<"OK: "<<(*it).second<<"\n";
64              }
65              else
66              {
67                  std::cout<<"NoSuchWord\n";
68              }
69          }
70          std::cin.get(command);
71      }
72      return 0;
73  }
```

test.py

```
1  import sys
2  import random
3  import string
4
5  def get_random_key():
6      return random.choice( string.ascii_letters )
7
8  actions = ["+", "?", "-"]
9  test_file_name = "tests/01"
10 output_file = open("tests/01.t", 'w')
11 answer_file = open("tests/01.a", 'w')
12 for _ in range(2000):
13     keys = dict()
14     for _ in range(3000):
15             action = random.choice( actions )
16             if action == "+":
17                 key_1 = get_random_key()
```

```
                key_2 = get_random_key()
                key_3 = get_random_key()
                key_4 = get_random_key()
                key = key_1 + key_2 + key_3 + key_4
                value = random.randint( 1, 1000 )
                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 ) )
            elif action == "-":
                delete_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"
                    keys.pop(key)
                else:
                    answer = "NoSuchWord"
                answer_file.write( "{0}\n".format(answer))
```

benchmark.sh

```
#! /bin/bash
make
echo "Time for AVL-tree"
time ./*2 < tests/01.t > tmp
g++ -std=c++14 benchmark.cpp
echo "Time for MAP from STL"
time ./a.out < tests/01.t > tmp
```

Консоль

```
alex$ bash bench*sh
g++ -std=c++11 -o diskran_laba_2 -pedantic -Wall -Werror -Wno-sign-compare
-Wno-long-long -lm main.cpp avl_tree.cpp
Time for AVL-tree

real    0m6.525s
user    0m6.332s
sys     0m0.108s
Time for MAP from STL

real    0m26.850s
user    0m16.212s
sys     0m10.632s
alex$
```

Как видно из теста производительности, AVL-дерево выигрывает по времени у std::map почти в 4-5 раз. Для теста были сгенерированы входные файлы, 6000000 строк в каждом. В них производится добавление, удаление и поиск элементов. В основе реализиции std::map лежит красно-черное дерево. Балансировка этого дерева происходит за $O(1)$, в то время как в AVL-дереве за $O(logn)$. Производительность работы программы зависит от ее использования. Если чаще всего нам придется искать элементы, то можно использовать AVL-дерево. Если же предвидятся частые вставки и удаления, то лучше использовать красно-черное дерево, так как балансировка будет происходить быстрее, а соответственно, быстрее будет работать программа.

5 Выводы

Лабораторная работа 2 познакомила меня с новыми структурами данных – сбалансированными деревьями. Они хорошо подходят для хранения различной информации – ведь мы исключаем тот случай, когда представление данных может выродиться в линейный список. Производительность программы заметно вырастает. Так же, я узнал, что находится внутри контейнера map из библиотеки STL, закрепил навыки в поиске ошибок и их отладке, а так же в тестировании моей программы через генератор тестов.

Список литературы

[1] Томас Х. Кормен, Чарльз И. Лейзерсон, Рональд Л. Ривест, Клиффорд Штайн. *Алгоритмы: построение и анализ, 2-е издание.*

[2] *АВЛ-дерево — Википедия.*
URL: `https://ru.wikipedia.org/wiki/АВЛ-дерево` (дата обращения: 30.06.2018).

[3] *C++ Program to Implement AVL Trees*
URL: *https://www.sanfoundry.com/cpp-program-implement-avl-trees/*