Московский авиационный институт
(национальный исследовательский университет)

Институт информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: И. С. Глушатов
Преподаватель: А. А. Кухтичев
Группа: М8О-207Б-19
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 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

AVL - дерево - структурный тип данных, являющийся сбалансированным по высоте двоичным деревом поиска, где высоты левого и правого поддерева отличаются не более, чем на единицу. AVL - дерево было придумано советскими учеными Георгием Адельсон-Вельским и Евгением Ландисом в 1962 году.

Балансировка дерева осуществляется алгоритмом левого или правого поворота относительно родительского и дочернего узлов на условии заданных правил. Так же выделяют отдельно большие правые и левые повороты, которые являются комбинацией малых.

Поиск, вставка и удаление в сбалансированном двоичном дереве производится за $O(log_2 n)$ независимо от порядка входных данных, а расход памяти пропорционален количеству элементов (т.е. $O(n)$).

2 Исходный код

На каждой непустой строке входного файла располагаются команды 5 видов:

- + word number
- - word
- word
- ! Save /path/to/file
- ! Load /path/to/file

Поэтому для реализации словаря понадобилось написать небольшой парсер входных команд. К тому же использование std::string было запрещено, так что пришлось с нуля написать mystd::string, дабы было удобно работать со строками.

```
namespace NMystd {
    struct TString {
        char str[256] {'\0'};
        int sizeString = 0;

        TString() = default;
        TString(const char* c) {
            while (c[sizeString] != '\0') {
                str[sizeString] = c[sizeString];
                ++sizeString;
            }
        }
        TString(const TString& c) {
            *this = c;
        }
        ~TString() = default;
        int Size();
        TString& operator=(const TString& rvl);
        TString& operator=(const char* c);
        bool operator==(const TString& rvl) const;
        bool operator!=(const TString& rvl) const;
        bool operator<(const TString& rvl) const;
        bool operator>(const TString& rvl) const;
        char& operator[](const int i);

    };
}
```

После парсинга команды, следует работа с деревом, куда мы, собственно, сохраняем данные, удаляем их или ищем. И при необходимости сохраняем и загружаем дерево по указанному пути. В качестве полей узел AVL - дерева хранит указатели на левое и правое поддеревья, ключ и значение, а так же максимальную высоту его поддеревьев.

```
struct TAVLTree {
    TAVLTree *left = 0;
    TAVLTree *right = 0;

    NMystd::TString key;
    unsigned long long value;
    unsigned int h;

    TAVLTree () = default;
```

```
    TAVLTree (const NMystd::TString& k, const unsigned long long v);
    ~TAVLTree () = default;
    void RemoveTree(TAVLTree *head);

    TAVLTree* RightRotate(TAVLTree *head);
    TAVLTree* LeftRotate(TAVLTree *head);
    TAVLTree* Balancing(TAVLTree *head);
    TAVLTree* MinRight(TAVLTree* head);
    TAVLTree* RemoveMin(TAVLTree* head);

    TAVLTree* Insert(TAVLTree *head, const NMystd::TString& key, const unsigned long
        long *value);
    TAVLTree* Remove(TAVLTree *head, const NMystd::TString& key);
    bool Search(TAVLTree *head, const NMystd::TString& key);

    void Print(const TAVLTree *head, unsigned int n);
    void Save(const TAVLTree *head, FILE *s);
    TAVLTree* BinInsert(TAVLTree *head, const NMystd::TString& key, const unsigned long
         long value);
    TAVLTree* Load(FILE *s);

    unsigned int Height(const TAVLTree *head);
    int Balance(const TAVLTree *head);
    void FixHeight(TAVLTree *head);
};
```

Главный файл *main.cpp* представляет из себя ввод команд из стандартного входного потока с помощью цикла *while* и метода *std* :: *cin*. Далее следует парсинг команды и определенное действие с деревом.

```
int main() {

  std::ios::sync_with_stdio(false);
  std::cin.tie(0);
  std::cout.tie(0);

  NMystd::TString c;
  TAVLTree *tree = 0;

  while (std::cin >> c) {

    if (c == "+") {

      unsigned long long value;
      std::cin >> c >> value;
      ToLower(&c);
      tree = tree->Insert(tree, c, &value);

    } else if (c == "-") {

      std::cin >> c;
      ToLower(&c);
      tree = tree->Remove(tree, c);

    } else if (c == "!") {
      std::cin >> c;
      if (c == "Save") {
        std::cin >> c;
        FILE *op = fopen(c.str, "w");
        if (op == nullptr) {
          std::cout << "ERROR: can not open file\n";
        } else {
```

```
33          tree->Save(tree, op);
34          fclose(op);
35          std::cout << "OK\n";
36        }
37      } else {
38        std::cin >> c;
39        FILE *op = fopen(c.str, "r");
40        if (op == nullptr) {
41          std::cout << "ERROR: can not open file\n";
42        } else {
43          tree->RemoveTree(tree);
44          tree = tree->Load(op);
45          fclose(op);
46          std::cout << "OK\n";
47
48        }
49      }
50    } else {
51      ToLower(&c);
52      tree->Search(tree, c);
53    }
54  }
55  tree->RemoveTree(tree);
56  return 0;
57
58 }
```

main.cpp	
void ToLower(NMystd::TString *str)	Функция перевода строки в нижний регистр.
string.hpp	
TString() = default TString(const char* c) TString(const TString& c)	Конструкторы TPair.
int Size()	Возвращает длину строки
TString& operator=(const TString& rvl) TString& operator=(const char* c)	Операторы присваивания

bool operator==(const TString& rvl) const;
bool operator!=(const TString& rvl) const;
bool operator<(const TString& rvl) const;
bool operator>(const TString& rvl) const;
	Операторы сравнения

char& operator[](const int i)	Оператор квадратных скобок
avltree.hpp	
TAVLTree () = default; TAVLTree (const NMystd::TString& k, const unsigned long long v);	Конструкторы дерева
void RemoveTree(TAVLTree *head);	Удаление дерева
TAVLTree* RightRotate(TAVLTree *head); TAVLTree* LeftRotate(TAVLTree *head);	Правый и левый повороты

TAVLTree* Balancing(TAVLTree *head);	Функция балансировки дерева
TAVLTree* MinRight(TAVLTree* head);	Поиск минимального элемента в правом поддереве
TAVLTree* RemoveMin(TAVLTree* head);	Удаление минимального элемента
TAVLTree* Insert(TAVLTree *head, const NMystd::TString& key, const unsigned long long *value);	Вставка в дерево
TAVLTree* Remove(TAVLTree *head, const NMystd::TString& key);	Удаление из дерева
bool Search(TAVLTree *head, const NMystd::TString& key);	Поиск в дереве
void Print(const TAVLTree *head, unsigned int n);	Печать дерева
void Save(const TAVLTree *head, FILE *s);	Сохранение дерева в файл
TAVLTree* BinInsert(TAVLTree *head, const NMystd::TString& key, const unsigned long long value);	Обычная вставка в дерево без балансировки
TAVLTree* Load(FILE *s);	Загрузка дерева из файла
unsigned int Height(const TAVLTree *head);	Возвращает максимальную высоту поддеревьев
int Balance(const TAVLTree *head);	Возвращает разницу высот поддеревьев
void FixHeight(TAVLTree *head);	Перерасчет высот

3 Консоль

```
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab2$ make run
python tests_creator.py
./main.out <tests/01.t >tests/01ans.txt
./main.out <tests/02.t >tests/02ans.txt
./main.out <tests/03.t >tests/03ans.txt
./main.out <tests/04.t >tests/04ans.txt
diff tests/01ans.txt tests/01.txt
diff tests/02ans.txt tests/02.txt
diff tests/03ans.txt tests/03.txt
diff tests/04ans.txt tests/04.txt
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab2$ ./main.out <tests/03.t
OK
OK: 10839719230826938404
OK: 10839719230826938404
OK: 10839719230826938404
OK
OK
NoSuchWord
OK
NoSuchWord
OK
OK
OK
OK: 13837742014146632709
NoSuchWord
OK
NoSuchWord
OK
OK: 8779566326796827863
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab2$
```

4 Тест производительности

Тест производительности представляет из себя следующее: в каждом запуске используется 4 файла с одинаковым количеством команд (вставка, удаление, поиск). Время исполнения программы подсчитывается с помощью библиотеки std::chrono. Сначала я запускаю программу с реализованным мной AVL - деревом, а потом с std::map.

```
igor@igor-Aspire-A315-53G:~/Рабочий стол/c++/DA/lab2$ cat time_log.txt
10000 0.0455774
20000 0.0940782
40000 0.158827
80000 0.292889

10000 0.0394729
20000 0.0652158
40000 0.133483
80000 0.162831
```

Как видно, программа в std::map работает быстрее.

5 Выводы

В ходе второй лабораторной работы я понял, насколько сложно обходиться без библиотек и уже написанных в них функций и классов. Реализовывать каждый раз один и тот же тип данных или брать его из прошлых проектов и адаптировать под новый - все это слишком замедляет процесс написания кода и тормозит реализацию главного задания, отвлекает от него. Прежде чем начать написание AVL - дерева, пришлось создать класс строки, реализовать отдельную программу для тестирования и, собственно, провести эти самые тесты. В данной лабораторной работе я написал сбалансированное дерево поиска и убедился в его эффективности. Узнал, что при сравнение константных переменных оператор сравнения должен возвращать именно константное булевое значение. Заметил, что в AVL - дереве правый и левый повороты происходят за константное время, что изначально было для меня не очевидным. По какой-то причине нерекурсивный поиск в дереве оказался чуть медленне, чем рекурсивный, однако я оставил его, чтобы избавиться от лишних рекурсивных действий.

Список литературы

[1] *AVL - деревъя.*
URL: https://medium.com/@dimko1/структуры-данных-avl-дерево-7f8739e8faf9

[2] *Про операторы C++ и их переопределение*
URL: https://en.cppreference.com/w/cpp/language/operators

[3] *Wikipedia про АВЛ - деревъя*
URL: https://ru.wikipedia.org/wiki/АВЛ-дерево