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: Т. Б. Даутов
Преподаватель: А. А. Кухтичев
Группа: М8О-206Б-22
Дата:
Оценка:
Подпись:

Москва, 2024

Лабораторная работа №1

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.

Вариант дерева: AVL дерево.

1 Описание

Требуется реализовать структуру AVL дерева, на основе которой создать структуру словаря.

AVL (Адельсон-Вельского и Ландиса) дерево - бинарное дерево поиска (BST), для каждой вершины которого выполняется условие, что модуль разности высот левого и правого поддеревьев не превосходит 1. Т.е. поддерживается сбалансированность.

Такое дерево решает главную проблему "наивных"деревьев поиска - потенциальную "деградацию"до линейного времени доступа к содержимому. Сохранение баланса достигается проверкой условия и выполнением операции балансировки при необходимости. Балансировка узлов проводится после операций вставки и удаления (т.е. операций, изменяющих состояние и структуру дерева) и представляет из себя повороты вокруг рёбер: левого, правого и их перекрёстных комбинаций (т.н. "больших"правого и левого поворотов).

2 Исходный код

Сначала был реализован шаблонный класс AVL дерева $TAVLTree$. Он содержит публичные методы поиска, вставки и удаления элементов: $Find$, $Insert$, $Remove$. Внутри класса описана приватная структура $TNode$ - структура узла дерева, содержащая значение вершины, высоту поддерева, начинающегося с этого узла, а также указатели на левое и правое поддеревья. В секции $private$ также реализованы статические методы для работы (преимущественно рекурсивной) с узлами дерева.

Затем была написана структура $TKeyValue$, содержащая соответствующие заданию ключ и значение (т.е. строку (в виде объекта класса $std :: string$) и целочисленного значения $uint64\_t$). Кроме того, внутри неё описаны операторы сравнения, неоходимые для дерева.

Также была описана вспомогательная структура данных очередь - $TQueue$, которая была применена для поуровневого обхода AVL дерева. Реализована на принципе односвязного списка.

Наконец, был реализован класс словаря $TDictionary$, расширяющий (наследующий) класс $TAVLTree$ с параметром шаблона $TKeyValue$. В него добавлены публичные методы $Load$ и $Save$, реализующие загрузку и сохранение содержимого словаря в файл.

main.cpp	
void MakeLower(std::string& s)	Функция преобразования строки в нижний регистр

```
struct TKeyValue final {
  std::string key;
  uint64_t value;
}

template <class T>
class TAVLTree {
  public:
    const T& TryFind(const T& v) const;
    bool Insert(const T& v);
    bool Remove(const T& v);
    void Clear();
  protected:
    struct TNode final {
      T value;

      int8_t height = 1;
      TNode* left = nullptr;
      TNode* right = nullptr;
    };

    TNode* root;

    static void calculate_height(TNode* node);
    static int8_t update_height(TNode* node);
    static int8_t calculate_bfactor(TNode* node);
    static TNode* rotate_left(TNode* node);
    static TNode* rotate_right(TNode* node);
    static TNode* rebalance(TNode* node);
    static TNode* find(TNode* node, const T& v);
    static TNode* find_min(TNode* node);
    static TNode* remove_min(TNode* node);
    static TNode* insert(TNode* node, const T& v);
    static TNode* remove(TNode* node, const T& v);
    static void clear(TNode* node);
};

struct TDictionary : public TAVLTree<TKeyValue> {
  void Save(const std::string& fname);
  void Load(const std::string& fname);
};
```

3 Консоль

```
timur@workstation$ gcc -pedantic -Wall -std=c11 main.c -o lab1
timur@workstation$ cat input.txt
+ a 1
+ A 2
+ bbb 18446744073709551615
bbb
A
-A
a
timur@workstation$ ./lab1 <input.txt
OK
Exist
OK
OK: 18446744073709551615
OK: 1
OK
NoSuchWord
```

4 Тест производительности

Тест производительности представляет из себя следующее: генерируется случайный большой набор действий-запросов: вставка, удаление, поиск элементов. Затем сравнивается время, потраченное на один набор действий каждой из структур: $std :: map$ и $TAVLTree$. Размер теста составлял 18000 строк.

```
timur@workstation$ g++ -std=c++2a benchmark.cpp -o bench
timur@workstation$ ./bench <bench.txt
[std::map] = 27ms
[AVLTree] = 34ms
```

Таким образом несложно заметить видим, что стандартная структура библиотеки C++ немного "выигрывает"у реализованного мной AVL-дерева. Данная разница может быть обусловлена особенностями реализации обеих структур.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я изучил и реализовал структуру AVL дерева и применил его для создания структуры словаря на языке C++. AVL является сбалансированным бинарным деревом посика, имеющее логарифмическую алгоритмическую сложность доступа к элементам. Сохранения оптимального состояния достигается с помощью простых поворотов, имеющих константную временную сложность.

В ходе выполнения сталкивался с проблемами превышения времени исполнения и ошибкой времени выполнения. Они были решены доработкой кода, отвечающего за балансировку и сохранения данных словаря в файл.

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] *АВЛ-дерево — Университет ИТМО.*
URL: `https://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево` (дата обращения: 18.04.2024).

[3] *АВЛ-деревья — Хабр.*
URL: `https://habr.com/ru/articles/150732/` (дата обращения: 18.04.2024).