Московский авиационный институт
(национальный исследовательский университет)

Факультет прикладной математики и физики

Кафедра вычислительной математики и программирования

Лабораторная работа №2 по курсу «Дискретный анализ»

Студент: Д.Г. Кривенко
Преподаватель: А.А. Кухтичев
Группа: 80-206
Дата:
Оценка:
Подпись:

Москва, 2016

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.

Вариант структуры : АВЛ – дерево.

1 Описание

Требуется реализовать АВЛ-дерево.
АВЛ–дерево – сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на единицу[1]. Так как в ходе работы с деревом его высота может измениться, а баланс нарушиться, применяют такую оперцию как балансировка вершины.
Балансировкой вершины называется операция, которая в случае разницы высот левого и правого поддеревьев равной 2, изменяет связи предок – потомок в поддереве данной вершины так, что разница становится меньше либо равной 1, иначе ничего не меняет[1]. Указанный результат получается вращениями поддерева данной вершины.
Существует несколько типов вращений: малое левое вращение, малое правое вращение, большое правое вращение, большое левое вращение. Большое вращение – это комбинация правого и левого малых вращений. Опишем псевдокодом малый левый поворот[2]:

```
1  function rotateLeft(Node a):
2   Node b = a.right
3   a.right = b.left
4   b.left = a
5   height adjustment
```

Опишем псевдокодом большой левый поворот[2]:

```
1   function bigRotateLeft(Node a):
2   rotateRight(a.right)
3   rotateLeft(a)
```

Малое правое и большое правое вращение определяются симметрично малому левому и большому левому вращению.
Алгоритм добавления вершины[1]:

1. Проход по пути поиска, пока не убедимся, что ключа в дереве нет.
2. Включение новой вершины в дерево.
3. «Отступления» назад по пути поиска и проверка в каждой вершине показателя сбалансированности. Если необходимо — балансировка.

Алгоритм удаления вершины:

1. Если вершина – лист, то удалим её и вызовем балансировку всех её предков в порядке от родителя к корню.
2. Найдём самую близкую по значению вершину в поддереве наибольшей высоты (правом или левом).

3. Переместим найденную вершину на место удаляемой вершины.
4. Вызовем процедуру удаления найденной вершины.

Операция поиска выполняется аналогично операции поиска в бинарном дереве.

2 Исходный код

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.

Создадим структуру *elem* в которой будем хранить ключь и значение. Создадим структуру *node* – езел дерева, который содержит указатели на правого и левого потомка $left$ и $right$, показатель балансировки *bal* и поле *data*

```
1  struct elem{
2      char* str;
3      size_t value;
4  };
5  struct node{
6      elem data;
7      int bal;
8      node* left;
9      node* right;
10     node(elem k) { data = k; left = right = NULL; bal = 0;}
11     ~node(){};
12 };
```

Опишем функциии, необходимые для работы с деревом.

main.cpp	
node* RightRotation(node* root)	Малый правый поворот
node* LeftRotation(node* root)	Малый левый поворот
node* Balance(node* root)	Балансировка дерева
int Add(node** rootp, elem val)	Добавление элемента в дерерво
node* Find(node** rootp, char val[])	Поиск элемента по значению
node* FindMin(node** rootp)	Нахождение минимального элемента
int Remove(node** rootp, char val[])	Удаление элемента
void Delete(node** rootp)	Удаление всего дерева
void Serialization(node** rootp, FILE* f)	Сериализация
node* Deserialisation(FILE* f)	Десериализация

3 Консоль

```
dmitriy@dmitriy-desktop:~$ g++ -pedantic -Wall -Werror -std=c++11 LR2.cpp
dmitriy@dmitriy-desktop:~$ ./a.out
+ wtrttNwtwt 63561513514367057 92
+ wtrttNwtw 113553462804448542 72
+ wtrttNwt 69203299096657264 64
+ wtrttNw 399671756802097152
+ wtrttN 11243149194066984960
+ wtrtt 5245798614692528128
+ wtrt 12751358511042461696
+ wtr 1814494297426624512
+ wt 9695799558577651712
+ w 3437433912018599936
OK
OK
OK
OK
OK
OK
OK
OK
OK
OK
w
OK: 3437433912018599936
- w
OK
w
NoSuchWord
+ wt 2342
Exist
! Save f1
OK
+ w 324
OK
w
OK: 324
! Load f2
ERROR: Couldn't create file
! Load f1
```

```
OK
w
NoSuchWord
```

4 Тест производительности

Тест производительности представляет из себя следующее: реализованное АВЛ–дерево сравнивается со стандартным контейнером map. В ходе тестирования производятся запросы различного характера. Время измеряется в тактах процессора.
Выпоняются 100 запросов различного характера(удаление, добавление и тд).

```
dmitriy@dmitriy-desktop:~$ g++ benchmark.cpp
dmitriy@dmitriy-desktop:~$./a.out
AVL time: 1643
MAP time: 944
```

Выполняютс 10^6 запросов различного характера(удаление, добавление и тд).

```
dmitriy@dmitriy-desktop:~$ ./a.out
AVL time: 7184635
MAP time: 20553412
```

Заметим, что реализованная структура и стандартный контейнер работают практически за одинаковое время.

5 Выводы

Выполнив вторую лабораторную работу по курсу «Дискретный анализ», я изучил АВЛ – дерево, а так же научился его реализовывать. Выполняя данную работу я впервые столкнулся с операциями сериализации и десериализация. В процессе написания соответствующих функций я подробно изучил работу с бинарными файлами в C++: функции fopen, fwrite, fread и др. Так же разробрал и написал алгоритмы сериализации и десериализация для АВЛ – дерева

Список литературы

[1] *АВЛ – дерево – Википедия.*
URL: hhttps://ru.wikipedia.org/wiki/АВЛ-дерево

[2] *АВЛ – дерево – Викиконспекты.*
URL: http://neerc.ifmo.ru/wiki/index.php?title=АВЛ-дерево