Лабораторная работа № 2 по курсу дискретного анализа: словарь

Выполнила студентка группы 08-208 МАИ *Понагайбо Анастасия.*

Условие

1. Постановка задачи: 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.

2. Вариант: 1, АВЛ-дерево.

Метод решения

- Реализовать АВЛ-дерево с функциями вставки, удаления, поиска, сохранения и загрузки.

- При считывании команды проверяем первый символ. Если первый символ '+', то считываем строку и число и вызываем функцию вставки в дерево. Если при вставке баланс дерева стал равен 2 или -2, то вызываем функцию балансировки. Если вставка прошла успешно, то выводим строку "OK". Если слово уже есть в словаре, то выводим "Exist".

- Если первый символ '-', то считываем строку и вызываем функцию удаления. Если баланс дерева стал равен 2 или -2, то вызываем функцию балансировки. Если удаление прошло успешно, то выводим строку "OK". Если такого слова в словаре нет, то выводим "NoSuchWord".

- Если первый символ '!', то считываем следующую строку. Если она равна "Save", то сохраняем дерево в файл. Если "Load", то загружаем дерево из файла. Если операция прошла успешно, то выводим строку "OK". Иначе выводим сообщение об ошибке.

- В остальных случаях вызываем функцию поиска слова в словаре. Если слово найдено, то выводим строку "OK" и соответствующий слову ключ. Иначе выводим строку "NoSuchWord".

Описание программы

Типы данных:
struct node - структура узла, хранящая ключ (*key*), номер (*value*), баланс (*balance*)

и ссылки на левый и правый узлы (*left* и *right*).

i64 - unsigned long long.

Основные функции:

node* RotateToLeft(node* root) - левый поворот относительно *root*.
node* RotateToRight(node* root) - правый поворот относительно *root*.
node* FixBalance(node* root) - балансировка дерева с корнем *root*.
node* Insert(char* k, i64 vl, node* root) - вставка ключа k с номером *vl* в дерево с корнем *root*.
void MinNode(node* root, node* mN) - поиск в дереве с корнем *root* элемента с минимальным ключом и копирование его в mN.
node* RemoveMin(node* root) - удаление минимального элемента дерева с корнем *root*.
node* Remove(char* k, node* root) - удаление элемента с ключом k в дереве с корнем *root*.
void Find(char* k, node* root) - поиск элемента с ключом k в дереве с корнем *root*.
void Serialise(node* root, FILE* f) - сохранение дерева с корнем *root* в файле *f*.
node* Deserialise(FILE* f) - загрузка дерева из файла *f*.
node* DeleteTree(node* root) - удаление дерева с корнем *root*.

Дневник отладки

Runtime error at test 04.t, got signal 11

Добавлена дополнительная проверка баланса, под ключ выделяется динамический массив.

Wrong answer at test 06.t

Добавлена проверка на корректность файла.

Runtime error at test 12.t, got signal 11

Исправлена ошибка при загрузке дерева (удаление пустого дерева).

Тест производительности

```
Количество введенных данных: 10,000:
AVL-TREE: 24129 ms.
     MAP: 65833 ms.

Количество введенных данных: 100,000:
AVL-TREE: 160925 ms.
     MAP: 302040 ms.

Количество введенных данных: 1,000,000:
AVL-TREE: 2198439 ms.
     MAP: 4109039 ms.
```

Выводы

АВЛ-дерево удобно использовать, если поиск по дереву происходит чаще, чем вставка и удаление, так как после них приходится часто производить повороты для обеспечения сбалансированности, что снижает быстродействие. Таким образом, использовать АВЛ-дерево для словаря выгодно, если поиск по нему происходит гораздо чаще, чем удаление и вставка.
Основные трудности возникли при написании функций балансировки и загрузки дерева.