Лабораторная работа №2 по курсу дискретного анализа: сбалансированные деревья.

Выполнил студент группы М80-208Б-20 Зинин Владислав Владимирович.

Условие

Кратко описывается задача:

1. 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.

2. Необходимо реализовать дерево B-дерево.

Метод решения

Для реализации необходимого нам словаря мы должны реализовать соответствующую заданию структуру данных. В моем случае это B-дерево. Главная особенность B-дерева в том, что оно не бинарное, а сильноветвящееся и в то же время сбалансированное дерево поиска. Оно позволяет производить поиск, добавлять и удалять элементы за O(log n). B-дерево с n узлами имеет высоту O(log n), кол-во детей узлов может быть от нескольких тысяч, поскольку степень ветвления B-дерева определяется его характеристическим значение, установить которое мы можем сами. Глубина всех листьев дерева одинаковая. Также отличительной особенностью B-дерева является то, что оно «растет вверх», поскольку оно является идеально сбалансированным. Решение реализовано с помощью отдельных классов для дерева и узла дерева.

Описание программы

Программа состоит из трех файлов – main.cpp, BTree.h, BNode.h. В файле BNode.h описаны класс ноды, необходимые для неё операторы сравнения, а также вспомогательные функции:

Int Min(int left, int right); - функция, возвращающая минимальный из двух элементов

void ToLower(char* key); - функция, выполняющая перевод строки в нижний регистр.

Файл BTree.h содержит описание класса дерево, а также описание его методов:

Void Insert(map &element); - добавление элемента в дерево

Void Search(char *key); - поиск элемента в дереве

Void Delete(char *key); - удаление элемента из дерева

Void Serialize(std::ofstream &file); - загрузка дерева в бинарный файл

Void Deserialize(std::ifstream &file); - скачивание дерева из бинарного файла

Также файл BTree.h содержит вспомогательные функции:

void SearchNode(BNode* node, char* key, BNode *&result, int &pos); - бинарный поиск ноды
void DeleteTree(BNode* node); - удаление дерева
void FPrint(BNode* root); - вывод дерева
void InsertNonfull(BNode* node, map& element); - добавление в дерево после того, когда корень не насыщен
void SplitChild(BNode* parent, int pos); - разделение ноды, если её размер равен критическому (2*T – 1)
void DeleteNode(BNode* node, char* str); - удаление ноды
int SearchInCurrentNode(BNode* node, char* str); - поиск в конкретной ноде
void Merge(BNode* parent, int pos); - слияние нод
void Rebalance(BNode* node, int &pos); - если при удалении ключа встретилась вершина с минимальным размером, то эта функция выполняет преобразования для увеличения резмера (слияние или перемещение вершины из брата)
void DeleteFromCurrentNode(BNode* node, int pos); - удаляем ключ в конкретной ноде
void NodeToFile(BNode* node, std::ofstream &file); - записываем дерево в файл
void TreeFromFile(BNode* node, std::ifstream &file); - считываем дерево из файла

Дневник отладки

Сначала я пытался залить свою программу с makefile, на что мне приходил ответ с ошибкой компиляции. Я так и не разобрался, как загрузить программу на проверку с makefile, поэтому дальнейшие мои посылки были в одном файле. Далее я столкнулся с RE, с которой долго не мог разобраться. После длительных проверок своей работы я обнаружил опечатку в функции Rebalance, где я забыл указать & перед pos, из-за чего значение pos у меня не изменялось, ибо надо было передавать конкретно ссылку. Потом я столкнулся с WA. Я понял, что проблема была с “Load”, но никак не мог найти ошибку. Я решил полностью переписать функцию считывания из файла, и у меня получилось сдать.

Тест производительности

Сравним работу B-дерева со стандартным контейнером std::map из STL. Для этого создадим файл с тестами из 10, 100, 1000, 10000 и 100000 строк соответственно для команд добавления, поиска и удаления из дерева соответственно.

Тест 1. Добавление:

Кол-во строк	Std::map	BTree
10	69ms	33ms
100	251ms	184ms
1000	2344ms	2294ms
10000	21830ms	27968ms
100000	209507ms	354801ms

Тест 2. Удаление:

Кол-во строк	Std::map	BTree
10	17ms	25ms
100	211ms	321ms
1000	2010ms	3394ms
10000	20014ms	39732ms
100000	197599ms	487588ms

Тест 3. Поиск:

Кол-во строк	Std::map	BTree
10	28ms	18ms
100	195ms	177ms
1000	1867ms	1954ms
10000	19623ms	22763ms
100000	202356ms	279073ms

Заметим, что std::map на больших данных работает быстрее BTree 1,4-2 раза. Возможно, это связано с тем, что std::map реализован на основе Red-Black дерева и его сложность O(logn), тогда как сложность B-дерева – O($\log_t n$*logt).

Недочёты

По началу было множество недочетов, связанных с загрузкой и скачиванием дерева из файла, но после выполнения лабораторной работы недочетов обнаружено не было.

Выводы

В результате проведенной лабораторной работы я научился работать с B-деревом – сильноветвящимся сбалансированным деревом поиска. С помощью тестирования своей программы я убедился, что B-дерево очень хорошо подходит для хранения большого количества информации, поскольку довольно-таки быстро справляется с ней, выполняя различные команды. Однако, сравнив время работы своего B-дерева с Red-Black деревом из std::map, я выяснил, что std::map работает быстрее. Но тем не менее std::map не имеет такого настолько же полноценного функционала., как моя лабораторная работа.