

Лабораторная работа №2 по курсу Дискретного Анализа: Структуры и деревья

Выполнил студент группы 08-208 МАИ Жерлыгин Максим Андреевич

Условие

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 264 - 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 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.

Для всех операций, в случае возникновения системной ошибки (нехватка памяти, отсутствие прав записи и т.п.), программа должна вывести строку, начинающуюся с «**ERROR:**» и описывающую на английском языке возникшую ошибку.

Вариант 4: В-Дерево.

Метод решения

Для выполнения данной лабораторной работы требуется выполнить два подзадания: реализовать само В-дерево, орагинозовать корректный ввод команд через консоль.

У б-дерева нужно релизовать функции поиска по дереву, вставки нового узла в дерево, удаления из листа, сохранения дерева в определенном формате в файл и загрузки из ранее созданного файла.

Вставка осуществляется по следующему алгоритму, сначала заопляется корень дерева до 2^*P-1 элементов (P - число сопоставленное с деревом) далее, происходит поиск места для новых ключей по алгоритму с хожим с поиском места для вставки с бинарным деревом поиска, затем, если вставка происходит в заполненный узел, тот делится на два узла по P-1 элементов а их разделителей уходит на узел выше.

Удаление - производится поиск ключа, если он находится в листе, то просто удаляется из него, иначе в зависимости от того, с какими узлами связан найденный ключ мы преобразуем ключи, путем заимствований элементов у соседних узлов, а также слияния соседних узлов в один при условии, что оба содержать P-1 элементов добиться удаления заданного элемента.

Сохранение и загрузка - схоже с сохранением и загрузкой бинарного дерева поиска: сохраняется размер узла затем характеристика - лист или не лист, а затем идут ключи, далее, если считанный узел не лист - запускаем алгоритм рекурсивно от след узла. Дерево конструируется параллельно, загрузка - аналогичный алгоритм.

Описание программы

Было разработано два класса: шаблонный для б-дерева который в параметрах шаблона принимает тип хранящегося ключа и характеристику дерева и обычный класс ключа.

Для второго были перегружены операторы сравнения, так как алгоритмы построения б-дерева основаны на сравнений значений ключей в узлах.

- `template<typename T, int P> class BTree` - Само б-дерево, T - тип ключа, P - характеристика Б-дерева.
- `template<typename, int> struct BTreeNode` - шаблон, содержится внутри класса `BTree`, обозначает один узел дерева, параметры аналогичные: T - тип ключа, P - характеристика Б-дерева.
- `BTreeNode()` - конструктор узла, создает пустой узел, памяти выделяется при вызове этого конструктора
- `BTreeNode()` - деструктор узла, удаляет узел с памятью.
- `T* BTreeSearch(T k)` - поиск в дереве ключа k, возвращает указатель на ключ при успешном поиске.
- `void BTreeSplitChild(BTreeNode<T, P>* u, int i)` - разделение узла u, по индексу i.
- `void BTreeInsertNonfull(T k)` - вспомогательный метод вставки ключа в дерево.
- `int BTreeFindKey(T k)` - поиск ключа k в текущем ключе
- `void BTreeDeleteFromLeaf(int index)` - удаление ключа из узла, который является листом.
- `void BTreeDeleteFromNonLeaf(int index)` - удаление ключа, который не является листом
- `T BTreeNodeGetPred(int index)` - достать у узла с индексом index - самый близкий слева узел хранящийся в дереве. `BTreeNodeGetSucc(int index)` - достать у узла с индексом index - самый близкий справа узел хранящийся в дереве
- `void BTreeFill(int index)` - заполнить полностью узел по индексу.
- `void BTreeBorrowFromPrev(int index)` - заимствовать элемент у след слева узла.
- `void BTreeBorrowFromPrev(int index)` - заимствовать элемент у след слева узла
- `void BTreeBorrowFromNext(int index)` - заимствовать элемент у след справа узла.
- `void BTreeMerge(int index)` - объединить узел index и index+1
- `void BTreeDeleteNode(T k)` - удаление ключа.
- `void BTreePrint(std::ostream& out, int c)` - вывод дерева.
- `void BTreeDestroy()` - удалить дерево и его потомков
- `void BTreeLoad(std::fstream& in)` - загрузить дерево из файла
- `void BTreeSave(std::fstream& out)` - сохранить дерево в файл

Дневник отладки

№	Проблема	Описание
1	Неправильный ответ на тесте 4	Некорректная реализация сохранения дерева в файл

Вывод

Б-дерево - неоднозначная структура данных.

С одной стороны по итогу завершения работы получаем довольно интересный тип данных, позволяющий хранить данные в особом порядке, при этом поиск нужной информации может быть осуществим за довольно короткое время, при использовании б-дерева переходы между кластерами данных осуществляются довольно редко, что в некоторых случаях сильно упрощает работу машине.

С другой стороны реализации данного типа данных - крайне трудная (по личным ощущениям) задач, некоторые методы требуют сразу несколько вспомогательных методов, что сильно увеличивает сложность и читаемость кода.

Основное преимущество В-Дерева в том, что жёсткий диск читает/пишет данные большими кусочками зараз (например, по 4кб), и количество потомков в вершине подбирается под это ограничение. Получается очень "невысокое" дерево, хорошо подходящее для хранения на диске. Диск сильно медленнее процессора, поэтому мы можем себе позволить хранить потомков в массиве и поддерживать их упорядоченность при вставке/удалении — всё равно перезапись делается кусками по 4кб.

Также, мы читаем не отдельные данные с разных мест, а целыми блоками. Перемещая узел дерева в оперативную память, мы перемещаем выделенный блок последовательной памяти, поэтому эта операция достаточно быстро работает.

Соответственно, мы имеем минимальную нагрузку на сервер, и при этом малое время ожидания. Эти и другие описанные преимущества позволили В-деревьям стать основой для индексов, базирующихся на деревьях в СУБД.