Лабораторная работа № 2 по курсу дискретного анализа: словарь

Условие

Необходимо создать библиотеку, реализующую указанную структуру данных, на основе которой разработать программу-словарь. В словаре каждому ключу, состоящему из регистронезависимой 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.

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 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.

Различия вариантов заключаются только в используемых структурах данных:
B-дерево

Метод решения

Нулевым шагом напишем аналогичную программу, работающую на основе декартова дерева, и сгенерируем большое количество случайных тестов.
Перед написание самого B-дерева был выбран наиболее простой метод работы с памятью программы, чтобы легко обрабатывать запросы Load и Save.
Идея заключается в следующем: реализуем стандартный динамический массив, в котором можно хранить ноды дерева, а в качестве ссылок на элементы дерева использовать индексы в массиве (нулевая ячейка массива будет терминирующей, т.е. являться аналогом NULL).
А для того, чтобы создавать и удалять ноды можно было за $O(1)$, в этом массиве свободные элементы будут образовывать односвязный линейный список.
Далее под ссылкой будет подразумеваться переменная типа *unsigned int* — индекс в динамическом массиве.

Описание программы

К сожалению, тестирующая система не позволила использовать модульный подход к решению поставленной задачи, поэтому весь рабочий код размещён в единственном файле *main.c*.
Структура дерева содержит 3 поля:

- root (индекс корневой ноды);
- nMemory (динамический массив, хранящий ноды дерева);
- eMemory (динамический массив, хранящий элементы дерева[1]);

Структура ноды дерева состоит из полей:

- keys_num (количество ключей хранящихся на текущий момент);
- distToBottom (до листа);
- parent (ссылка на родителя);
- keys (массив ссылок на ключи, хранящиеся в ноде);
- children (массив ссылок на потомков);

Далее следуя алгоритмам описываем основные функции дерева.

Для более простой реализации введём ещё такие вспомогательные функции, как ребалансировка, объединение узлов и их разъединение, удаления ключа и всего узла. Задача поиска фактически является обобщением случая двоичного дерева и не имеет каких-либо сложностей в реализации. При вставке нужно следить за инвариантом:

[1] значения не хранятся в виде одного массива внутри каждой ноды в целях экономии памяти

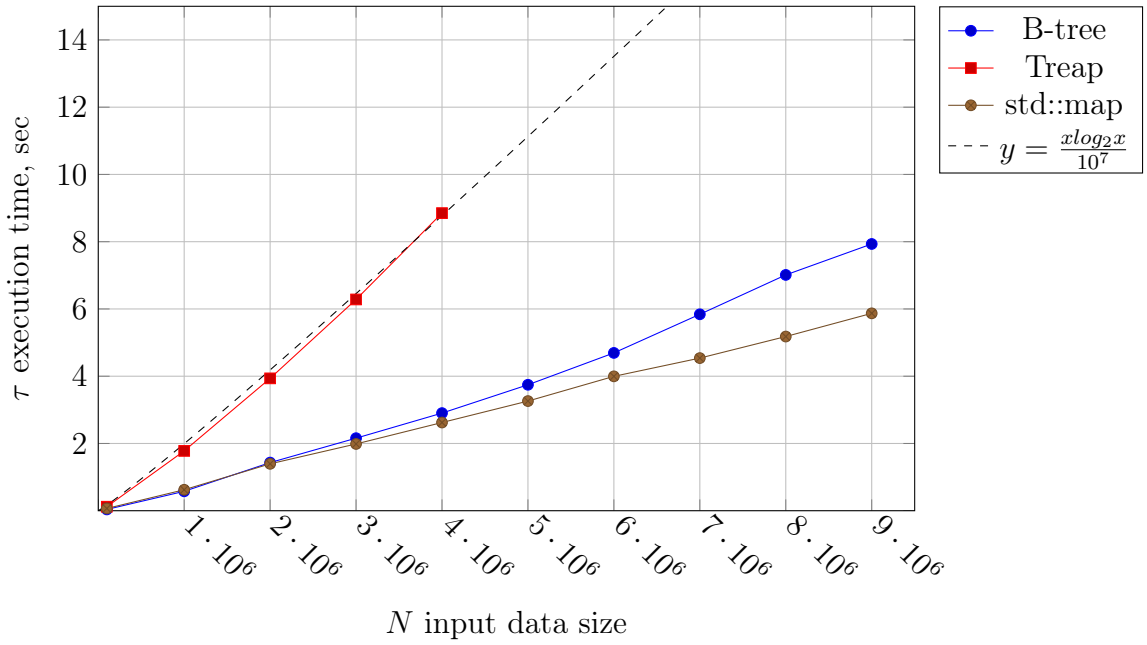
число ключей в ноде не может превысить $2 \cdot t - 1$. В случаях переполнения вершин будем делить их на две по $t - 1$ элементу, а оставшийся средний ключ отправлять к родителю. При удалении существует несколько различных случаев, но основными методами поддержания инварианта будут: перетаскивание ключей из соседних нод и слияние двух вершин дерева.

Функции загрузки и выгрузки дерева не отличаются своей сложной реализацией, так как нам требуется работать только с двумя массивами, которые просто записываются непрерывным куском байт, и ещё с одним единственным числом — ссылкой на корень.

Дневник отладки

№	Описание ошибки	Способ устранения
1	Баг при загрузке маленького дерева в уже существующее большое дерево. Использовалась лишь часть выделенной памяти, хотя утечки не происходило.	Для исправления было необходимо обрезать часть массива данных дерева функцией realloc.
2	RE-7 (оказалось, что функция fclose вызывает ошибку, при попытке закрыть нулевой указатель)	Для исправления было необходимо перенести строку с закрытием файла внутрь блока, где проверяется, был ли открыт файл.
3	WA-7	Для прохождения теста нужно было переписать обработку ошибок при попытке записи дерева в файл, чтобы программа печатала "OK"вне зависимости от успеха выполнения запроса.
4	WA-9	Для прохождения теста нужно было переписать обработку ошибок при попытке чтения дерева из файла, чтобы программа печатала "OK"вне зависимости от успеха выполнения запроса.
5	WA-11	Для прохождения теста нужно было после неудачной попытки чтения из файла очищать дерево.

Тест производительности



Выводы

В ходе выполнения лабораторной работы вспомнил принцип построения бинарного дерева поиска, изучил и реализовал структуру данных B-дерево.

B-дерево может применяться для структурирования информации на жёстком диске. Время доступа к произвольному блоку на жёстком диске велико (порядка миллисекунд), поскольку оно определяется скоростью вращения диска и перемещения головок. Поэтому важно уменьшить количество узлов, просматриваемых при каждой операции, что и позволяет сделать данная структура. Операции вставки, удаления и поиска элементов выполняются за временную сложность $O(logn)$

Основной проблемой в реализации дерева было то, что приходилось работать не с привычными указателями языка C, а использовать собственные на основе индексации массива. По этой причине возникли такие неочевидные баги, когда при обращении по указателю одновременно в левой и правой части присваивания, происходила запись в чужую память, при миграции данных во время расширения. Подобные и более простые ошибки замедляли ход написания лабораторной рабы. И всё же плюсом являлся крайне простой способ сохранения дерева в долговременную память.

Дополнительно для сравнения был реализован аналогичный словарь, только на основе $Treap$, как видно из тестов производительности, такой вариант достаточно сильно проигрывает по времени работы своим альтернативам, хотя имеет такую же временную сложность (стоит отметить, что код написать для такого дерева можно менее, чем за 1 час).