Лабораторная работа № 2 по курсу Дискретный Анализ. Сбалансированные деревья

Выполнил студент группы М8О-207Б-21 МАИ *Друхольский Александр.*

Условие

Кратко описывается задача:

1. Реализовать декартово дерево с возможностью поиска, добавления и удаления элементов. 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.

2. Вариант задания: Декартово дерево

Метод решения

Декартово дерево - это структура данных, объединяющая в себе бинарное дерево поиска и бинарную кучу. Дерево строится, опираясь на пары (key, priority) - ключ и приоритет. То есть по ключу - это бинарное дерево, а по приоритету - куча.

Основные операции, которые применимы для этого дерева - split и merge.

SPLIT: разрезать исходное дерево T по ключу k. Возвращать она будет такую пару деревьев $\langle$T1,T2$\rangle$, что в дереве T1 ключи меньше k, а в дереве T2 все остальные: split(T,k)$\rightarrow\langle$T1,T2$\rangle$.

MERGE: с помощью этой операции можно слить два декартовых дерева в одно. Причём, все ключи в первом(левом) дереве должны быть меньше, чем ключи во втором(правом). В результате получается дерево, в котором есть все ключи из первого и второго деревьев: merge(T1,T2)$\rightarrow$T

Удаление: удаляем нужную вершину a на её место ставим merge от её левого и правого сына

Вставка: разрезаем дерево по ключу k, затем представляем k как отдельное дерево и сливаем его с первым деревом от split. Результирующее дерево сливаем со вторым деревом от split.

Замечание: приоритеты генерируем рандомом.

Описание программы

1. Структура *Pair*, которая нужна для вывода результата split -> T1, T2.
2. Структура *node*, которая содержит указатели на правого и левого ребёнка, ключ, приоритет, значение.
3. Основные функции
   1.1 $Pair\ TreeSplit(node*\ tree,\ char*\ key)$ разбивает дерево по ключу, возвращает пару деревьев
   1.2 $node*\ TreeMerge(node*\ left,\ node*\ right)$ сливает два дерева, возвращает указатель на новое дерево.
   1.3 $void\ TreeInsert(node*\ tree,\ char*\ key,\ unsigned\ long\ long\ value)$ осуществляет вставку элемента в дерево.
   1.4 $void\ TreeDelete(node*\ tree,\ char*\ key)$ осуществляет поиск и удаление нужной вершины
   1.5 $node*\ Create_tree(char*\ key,\ unsigned\ long\ long\ value)$ создаёт вершину.
3. Дополнительные функции: поиск в дереве, длина строки, копирование строки, печать дерева (для тестирования).
4. Функция $int\ main(void)$ состоит из ввода данных, вызова функций работы с деревом.

Все структуры сделаны динамическими, чтобы удовлетворять условию по ограничению памяти.

Дневник отладки

WA - 5 Проблема в алгоритме с удалением.

RE - 12 Исправил выделение памяти под строки в структуре node.

WA - 7 Исправил ошибки при удалении корня.

Тест производительности

Замерялось время выполнения, включая ввод данных.

1. Вставка

№	Кол-во вершин	Время, с
1	1000	0,009
2	10000	0,038
3	100000	0,380
4	1000000	14,215

2. Вставка + поиск

№	Кол-во вершин	Время, с
1	1000	0,010
2	10000	0,046
3	100000	0,791
4	1000000	25,635

Каждый последующий тест больше предыдущего в 10 раз, что делает результат замера более наглядным.

Выводы

Я познакомился с декартовым деревом и научилс с ним работать. Это достаточно простая структура представления информации, но тем не менее полезная. Средняя глубина вершины при случайных приоритетах будет равна O(log n). Недостатки заключаются в большом объёме памяти, так как требуется хранить приоритеты, а так же времени доступа к вершинам, так как высота дерева может оказаться (маловероятно) линейной от количества элементов, что мы пытаемся предотвратить генерируя рандомные приоритеты.