Домашнее Задание по алгоритмам №7

Павливский Сергей Алексеевич , 873

24.03.2019

Задача 1.

Реализуйте стек с помощью двух очередей.

Решение

Будем действовать так : пусть у нас есть queue1 , queue2 . Считывая элемент , последовательно получаем элементы из queue1 , заносим их в queue2 , помещаем считанный элемент в queue1 , перекладываем эллементы из queue2 обратно в queue1 . Итого : мы сохранили порядок элементов , а последний считанный элемент будет первым на выход , т.е. выполняется свойство LIFO , т.е. мы реализовали стек .

Задача 2.

Опишите способ хранения почти-полного троичного дерева в массиве. Как по номеру клетки родителя вычислить номера детей? Как по номеру ребёнка вычислить номер родителя?

Решение

Описание.

Назовем массив А. Тогда :

1. Корень - А[1];
2. Дети элемента А[i] - элементы А[3i - 1], А[3i], А[3i + 1];
3. У элемента А[i] родитель – А[$\lfloor \frac{i+1}{3} \rfloor$]

Задача 3.

Докажите, что если в бинарном дереве поиска у элемента x нет правого ребёнка и у x есть следующий за ним в порядке возрастания элемент y, то y является самым нижним предком x, чей левый дочерний узел так же является предком x или самим x.

Решение

Если в бинарном дереве поиска есть некоторый элемент a, и мы относительно него спустимся на один элемент вниз по левой ветке , а дальше будем до упора спускаться вниз по правым веткам , то мы найдем максимальный элемент , меньший данного , т.е. предшествующий ему в упорядоченном массиве элементов бинарного дерева поиска . Собственно , эта ситуация в задаче и описывается : так как у x нет правых детей , то мы дошли до упора , а значит если мы от x пойдем наверх по левым веткам до тех пор , пока не сможем свернуть направо , а дальше свернем наверх направо , то это и будет следующий за ним в порядке возрастания элемент y , т.е. утверждение верно .

Задача 4.

Покажите, что если вершина b в бинарном дереве поиска имеет две дочерние вершины, то последующая за ней вершина c не имеет левой дочерней вершины, а предшествующая ей вершина a — правой. Под предшествующей и последующей вершиной понимается, что a.key < b.key < c.key и в дереве поиска нет ключей в промежутках (a.key, b.key) и (b.key, c.key).

Решение

Аналогично задаче 3 в сочетании с информации с лекций , чтобы найти последующую после b вершину c , нужно спуститься один раз относительно b направо , а потом спускаться до упора налево ; чтобы найти предшествующую перед b вершину a , нужно спуститься один раз налево , а потом спукаться до упора направо . Спустившись до упора в обоих случаях , мы найдем вершины c и a соответственно . Тогда , по скольку мы спустились до упора , то у a нет правой дочерней вершины , а у c левой , что и требовалось доказать .

Задача 5.

Известно, что в структуре данных потребуется хранить k-элементное подмножество A n-элементного множества. После того как в структуру данных будет загружено множество A, с помощью неё будет нужно проверить принадлежит ли A элемент x. Для этого можно совершить не более t запросов к структуре: каждый запрос q представляет собой конечную строку битов, ответ на каждый запрос — один бит. Структура данных представляет собой таблицу с двумя столбцами: первый столбец состоит из всевозможных запросов q (известных заранее), а правый из битов-ответов на запрос – правый столбец формируется после загрузки в структуру множества A. Пусть s(n, k, t) – минимальное количество строк в такой таблице которое достаточно отвести под такую структуру данных.

1. Чему равно s(n, k, 1)?
2. Найдите min (s(n, k, t)), при каком t он достигается?

Решение

1. Всего нужно n строк, причем каждая строка — вопрос о принадлежности сооответствующего элемента. Если меньше, то найдутся такие элементы, по которым за 1 запрос не удастся установить их принадлежность.

2. Нужно n - 1 строк по n - 1 элемента множества , при этом максимальное количество запросов количеством в n - 1 будет приходиться на незаписанный элемент .

Если же в таблице не будет информации про какие-то 2 элемента, то они будут для нас неразличимы, и структура будет некорректной.

Задача 6.

К серверу приходят одновременно n клиентов. Для клиента i известно время его обслуживания t_i. Время ожидания клиента определяется как сумма времени обслуживания всех предыдущих клиентов и времени обслуживания его самого. К примеру, если обслуживает клиентов в порядке номеров, то время ожидания клиента i будет равно $\sum\limits_{j=1}^{i} t_j$ Постройте эффективный алгоритм, находящий последовательность обслуживания клиентов с минимальным суммарным временем ожидания клиентов.

Решение

Чем раньше по порядку обслуживания будет стоять клиент, тем больше раз он будет включен в итоговую сумму . Значит клиент с меньшим t_i должен стоять раньше в очереди . Значит клиенты должны быть отсортированны в порядке возрастания t_i . Это делается за O($n \cdot logn$) .