Домашнее задание №4

Д.А. Першин

27 ноября 2014 г.

1 Словесное описание алгоритма

Пусть имеется входной массив a длины n, содержащий вершины бинарного дерева поиска в preorder порядке. Необходимо вывести дерево в inorder и postorder порядке. Будем использовать следующее утверждение: первая вершина a_1 во входной последовательности являеся корнем (очевидно из определения preorder порядка). Для данной вершины (корня) существует левое и правое поддеревья $a_l = [a_2, a_{k-1}]$ и $a_r = [a_k, a_{n-1}]$, найдем индекс k. Из определения бинарного дерева поиска очевидно, что это будет первая вершина, большая или равная (из условия) чем корень (такое k: $a_k >= a_1$). Для поиска k будем испльзовать алгоритм *All nearest greater values* (будет описан ниже).

В итоге получаем индексы корня, правого поддерева, левого поддерева. Для вывода дерева в inorder порядке выводим сначало левое поддерево, затем текущий корень, затем правое поддерево, для postorder - левое поддерево, правое поддерево, корень. Данный алгритм выполняестя рекурсивно для правого и левого поддерева, пока поддерево не будет состоять из одного элемента без поддеревьев (листа). В таком случае просто выводим лист.

Алгоритм *All nearest greater values* - алгоритм поиска ближайшего првого соседа, большего или равного, чем текущий. Алгоритм работает следующим образом: начинаем обход массива с конца, создадим стек s, а также результирующий массив r. Если текущий элемент a_i менше или равен элемету на вершине стека s, то для текущего элемента он будет большем или равным ближайшим соседом, $r[i] = k$, где k - индекс элемента на вершине стека, далее кладем текущий элемент в стек. Если текущий элемент больше, чем элемент на вершине стека, начинаем извлекать значания из стека, пока не найдем большее или равное значение и выполняем предыдущий пункт, или пока стек не опустеет, тогда для текущего элемента не существует такого соседа - $r[i] = None$, кладем текущий элемент в стек. Алгоритм выполняеся для $i = n - 1...0$. Несложно заметить, что каждый элемент кладется в стек и извлекается не более одого раза, следовательно временная сложность $O(n)$; если массив - строго возрастающая последовательность, то глубина стека может достигнуть n, следовательно сложность по памяти - $O(n)$.

Алгоритм:

1. Для каждого элемента a_iвходного массива a длиный n найдем ближайшего большего или равного соседа:

   (a) создадим пустой стек s (вершина - $top(s)$, глубина - $size(s)$), результирующий массив r длины n, занчание a_i будем записывать в стек как пару чисел (i, a_i);

   (b) для всех a_i пока $size(s) \neq 0$ и $a_i > top(s)$, извлекаем значение из стека;

   (c) если $size(s) = 0$, то $r_i = None$, кладем a_i на стек;
   иначе $r_i = index(top(s))$, кладем a_i на стек;

   (d) переходим к пункту b

2. $left = 0$, $right = n - 1$.

3. Для массива $[a_{left}, a_{right}]$ a_{left} - корень поддерева, $pivot = r_{left}$.

4. - если $left = right$, то:
   $print(a_{left})$
   $return$
   - для вывода $inorder$ порядка:
   рекурсивно пункт 3 для $left = left + 1$, $right = pivot - 1$
   $print(a_{left})$
   рекурсивно пункт 3 для $left = pivot$, $right = right$
   $return$
   - для вывода $postorder$ порядка:
   рекурсивно пункт 3 для $left = left + 1$, $right = pivot - 1$
   рекурсивно пункт 3 для $left = pivot$, $right = right$
   $print(a_{left})$
   $return$

2 Доказательство корректнсти

Предположим, что элемент a_{left} не является корнем текущего поддерева, но данное условие противоречит определению $preorder$ порядка (сначала печатаеся корень поддерева, а затем правое и левое поддеревья), следовательно a_{left} действительно является корнем текущего поддерева. Предположим, что $p = r_i$ не является первой вершиной правого поддерева, тогда существует вершина $k > p$, такая что $a_k < a_{left}$, но данное условие противоречит определению бинарного дерева поиска, следовательно r_i является первой вершиной правого поддерева.

3 Асимптотические оценки

В результате получаем сложность по памяти $O(n)$, так как мы используем входной массив длины n, стек для поиска ближайшего большего или равного соседа, глубина которого может достинать в среднем $\log(n)$, в худшем случае - n, а также рекурсивный алгоритм для вывода массива в inorder и postorder порядке, глубина рекурсии также в среднем - $\log(n)$, в худшем случае - n. Сложность по времени равна $O(n)$, так как мы используем рекурсивный алгоритм вывода дерева - в среднем $O(\log(n))$, в худшем случае - $O(n)$, поиск ближайшего большего или равного соседа - $O(n)$.