Лекция 16 Двоичные деревья (окончание)

16.1. Операции над двоичными деревьями.

16.1.1. Ключевое слово **`typedef`** позволяет определять новые имена типов данных. Синтаксис: **`typedef`** *`тип новое_имя_типа`*. (1)

Например, в функциях обработки двоичного дерева постоянно использовался тип **`struct BT_node`**, причем такой тип описывал как *корень* двоичного дерева (т.е. все двоичное дерево), так и его *узел*. Конструкция (1) позволяет ввести два новых имени для типа:

```
struct BT_node {
   int key;
   struct BT_node *left;
   struct BT_node *right;
   struct BT_node *parent;
}
```

Имеем:

```
typedef struct BT_node *BT; /* BT – новое имя типа
                               "указатель на struct
                               BT_node" */
typedef struct BT_node *node; /* node – еще одно новое
                               имя типа "указатель на
                               struct BT_node" */
```

Теперь можно написать новые объявления функций обработки двоичного дерева:

```
node BTsearch(BT T, int k);
node BTmin(BT T);
node BTmax(BT T);
node BTsucc(node n);
node BTpred(node n);
void BTinsert(BT T, node n);
node BTdelete(BT T, node n);
```

16.1.2. Новое имя типа, определенное с помощью **`typedef`**, можно использовать в других **`typedef`** в позиции *`тип`* для определения еще более новых имен типа.

16.1.3. Конструкция **`typedef`** не вводит новых типов данных (в отличие от конструкции **`struct`**). Она вводит новые имена для старых типов данных, что может существенно упростить написание кода и сделать его более понятным.

16.2. Определение функции **`void BTdelete(BT T, node n);`**

16.2.1. ***Алгоритм***.

На входе: указатель на дерево *T* (фактически это указатель на корень *root* дерева *T*) и указатель на узел *n* дерева *T*.

На выходе: двоичное дерево *T* с удаленным узлом *n* (ключи нового дерева по-прежнему упорядочены).

Необходимо рассмотреть три случая: (1) у узла *n* нет детей (листовой узел); (2) у узла *n* только один ребенок; (3) у узла *n* два ребенка.

(1) просто удаляем узел *n*;

(2) вырезаем узел *n*, соединив единственного ребенка узла *n* с родителем узла *n*.

(3) находим *succ*(*n*) и удаляем его, поместив ключ *succ*(*n*) в узел *n*.

Случаи (1) и (2) можно объединить.

Шаг 1: если у n меньше двух детей (либо левый, либо правый ребенок отсутствует), удаляем n, иначе удаляем $succ(n)$; устанавливаем указатель y на удаляемый узел.

Шаг 2: находим ребенка удаляемого узла (на него указывает y) и устанавливаем на него указатель x (если у удаляемого узла нет детей, имеет значение *NULL*).

Шаг 3: подвешиваем ребенка y (указатель x) к родителю y; если у y нет родителя (y – корень дерева), новым корнем дерева становится x; устанавливаем в соответствующем поле родителя указатель на x, полностью исключая y из дерева.

Шаг 4: если удаляемый узел $succ(n)$, заменяем данные узла n на данные узла $succ(n)$.

16.2.2. ***Программа на Си***.

```
node BTdelete(BT T, node n) {
        node x, y;

        /* y – указатель на удаляемый узел n */
        if(n->left == NULL || n->right == NULL) y = n;
        else y = succ(n);

        /* x – указатель на ребенка y, либо NULL */
        if(y->left != NULL) x = y->left;
        else x = y->right;

        /* если x – ребенок y, вырезаем y */
        if(x != NULL) x->parent = y->parent;

        /* если у y нет родителя (y – корень дерева) */
        /* новым корнем дерева становится x */
        if(y->parent = NULL) T = x;
        else {

        /* x присоединяется к y->parent с требуемой стороны */
             if(y == y->parent->left) y->parent->left = x;
             else y->parent->right = x;
        }

        /* если удалялся не узел n, а succ(n), необходимо */
        /* заменить данные узла n на данные узла succ(n) */
        if(y != n) n->key = y->key;
        ........................

        /* функция возвращает указатель удаленного узла, что */
        /* дает возможность использовать этот узел в других */
        /* структурах, либо очистить занимаемую им память */
        return y;
}
```

Среднее время выполнения функции **BTdelete** $O(h)$, где h – высота дерева T.

16.3. Построение двоичного дерева поиска.

16.3.1. Пусть имеется множество из m ключей: k_0, k_1, …, k_{m-1}. Произведем разбиение этого множества на три подмножества B_1, B_2, B_3, причем B_2 состоит ровно из одного элемента, а B_1 и B_3 могут быть пустые. Разбиение произведем таким образом, что все ключи из B_1 меньше ключа из B_2, а все ключи из B_3 больше (или равны) ключа из B_2.

Далее движемся рекурсивно: B_1 разбивается на B_{11}, B_{12}, B_{13}, а B_3 на B_{31}, B_{32}, B_{33} и т.д. В каждой тройке сохраняется то же соотношение ключей, что и при первом разбиении.

16.3.2. Пример: 15,10,1,3,8,12,4. Первое разбиение: {1,3,4}, {8}, {15,10,12}; второе разбиение: {{1}{3}{4}}{8}{{10}{12}{15}}. Получилось полностью сбалансированное двоичное дерево, высота которого, как известно пропорциональна $\log_2 m$.

16.3.3. Дерево называется *полностью сбалансированным* (*совершенным*), если длина пути от корня до любой листовой вершины одинакова.

16.3.4. Пусть h – высота полностью сбалансированного двоичного дерева. Тогда число вершин m должно быть равно:

$$m = 1 + 2 + 2^2 + \ldots + 2^h = \sum_{i=0}^{h} 2^i = \frac{1\cdot(2^h - 1)}{2 - 1} = 2^h - 1$$

откуда $h = \log_2(m + 1)$.

16.3.5. Если все m ключей известны заранее, их можно отсортировать ($O(m\cdot\log_2 m)$), после чего построение сбалансированного дерева будет тривиальной задачей. Если же дерево строится по мере поступления ключей, то все может быть: от линейного дерева ($O(m)$) до полностью сбалансированного дерева ($O(\log_2 m)$).

Например, если идет поток ключей 1, 3, 5, 8, 10, 12, 15, то, поместив в корень дерева ключ 1, мы будем заносить все ключи справа от текущего узла и получим линейное дерево.

16.3.6. Пусть $T = \{root, left, right\}$ – двоичное дерево; тогда $h_T = \max(h_{left}, h_{right}) + 1$.

16.4. Деревья Фибоначчи

16.4.1. ***Числа Фибоначчи*** возникли в решении задачи о кроликах, предложенном в XIII веке Леонардо из Пизы, известным как Фибоначчи. **Задача**: пара новорожденных кроликов помещена на остров. Каждый месяц любая пара дает приплод – также пару кроликов. Пара начинает давать приплод в возрасте двух месяцев. Сколько кроликов будет на острове через n месяцев? В конце первого и второго месяцев на острове будет одна пара кроликов: $f_1 = 1$, $f_2 = 1$. В конце третьего месяца родится новая пара, так что $f_3 = f_2 + f_1$. По индукции можно доказать, что для $n \geq 3$ $f_n = f_{n-1} + f_{n-2}$. n-е число Фибоначчи вычисляет следующая функция:

```
int Fbn(int n) {
    if ((n == 1) || (n == 2))
       return 1;
    else {
       g = h = 1;
       for(k = 2; k < n; k++) {
            Fb = g + h;
            h = g;
            g = Fb;
       }
       return Fb;
    }
}
```

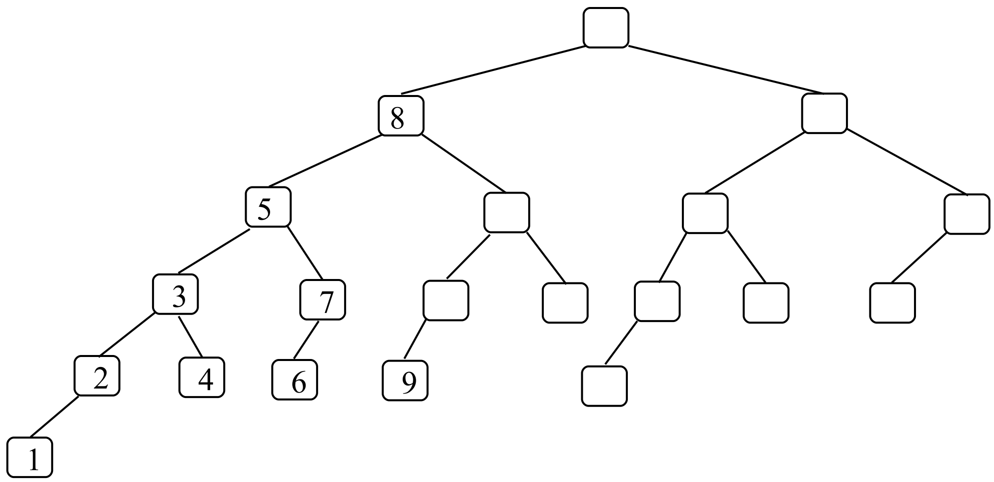
16.4.2. ***Определение*** дерева Фибоначчи (это тоже искусственное дерево).

(1) Пустое дерево – это дерево Фибоначчи с высотой $h = 0$.

(2) Двоичное дерево, левое и правое поддерево которого есть деревья Фибоначчи с высотами соответственно $h - 1$ и $h - 2$ (либо $h - 2$ и $h - 1$), есть дерево Фибоначчи с высотой h.

Из определения следует, что в дереве Фибоначчи значения высот левого и правого поддерева отличаются ровно на 1.

16.4.3. ***Пример***. Дерево Фибоначчи с $h = 6$.



16.4.4. ***Теорема***.

Число вершин в дереве Фибоначчи F_h высоты h равно $m(h) = f_{h+2} - 1$.

Доказательство по индукции.

$h = 0$: $m(0) = f_2 - 1 = 0$б $m(1) = f_3 - 1 = 1$.

Шаг: по определению (16.4.2) $m(h) = m(h-1) + m(h-2) + 1$. Имеем $m(h) = (f_{h+1} - 1) + (f_h - 1) + 1 = f_{h+2} - 1$, так как согласно 16.4.1 $f_h + f_{h+1} = f_{h+2}$

16.4.5. ***Теорема***.

Пусть C_1 и C_2 таковы, что уравнение $r^2 - C_1 r - C_2 = 0$ (*) имеет два корня r_1 и r_2, $r_1 \neq r_2$. Тогда (1) из $a_n = C_1 a_{n-1} + C_2 a_{n-2}$ и начальных условий a_0 и a_1 следует $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ для $n = 1, 2, \ldots$, и (2) из $a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$ следует $a_n = C_1 a_{n-1} + C_2 a_{n-2}$.

Доказательство (2). Так как r_1 и r_2 – корни уравнения (*), то $r_1^2 = C_1 r_1 + C_2$ и $r_2^2 = C_1 r_2 + C_2$. Имеем:

$$C_1 a_{n-1} + C_2 a_{n-2} = C_1(\alpha_1 r_1^{n-1} + \alpha_2 r_2^{n-1}) + C_2(\alpha_1 r_1^{n-2} + \alpha_2 r_2^{n-2}) = \qquad (**)$$

$$= \alpha_1 r_1^{n-2}(C_1 r_1 + C_2) + \alpha_2 r_2^{n-2}(C_1 r_2 + C_2) = \alpha_1 r_1^{n-2} r_1^2 + \alpha_2 r_2^{n-2} r_2^2 = \alpha_1 r_1^n + \alpha_2 r_2^n = a_n$$

Доказательство (1). Нужно не только повторить в обратном порядке вывод (**), но и подобрать такие α_1 и α_2, чтобы $a_0 = \alpha_1 + \alpha_2$, $a_1 = \alpha_1 r_1 + \alpha_2 r_2$ (***). Рассматривая (***) как систему линейных уравнений относительно α_1 и α_2, получим:

$$\alpha_1 = \frac{a_1 - a_0 \cdot r_2}{r_1 - r_2}, \quad \alpha_2 = \frac{-a_1 + a_0 \cdot r_1}{r_1 - r_2}$$. Теорема доказана.

16.4.6. Применим теорему (16.4.5) к числам Фибоначчи: $f_n = f_{n-1} + f_{n-2}$. Корнями уравнения $r^2 - r - 1 = 0$ являются $r_1 = \dfrac{1+\sqrt{5}}{2}$ и $r_2 = \dfrac{1-\sqrt{5}}{2}$. Следовательно, согласно теореме

$$f_n = \alpha_1 \cdot r_1^n + \alpha_2 \cdot r_2^n = \alpha_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n + \alpha_2 \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n,$$

$$f_0 = \alpha_1 + \alpha_2 = 0,$$

$$f_1 = \alpha_1 \cdot \left(\frac{1+\sqrt{5}}{2}\right) + \alpha_2 \cdot \left(\frac{1-\sqrt{5}}{2}\right) = 1 \qquad ,$$

$$\alpha_1 = \frac{1}{\sqrt{5}}, \alpha_2 = -\frac{1}{\sqrt{5}}$$

Откуда

$$f_n = \frac{1}{\sqrt{5}} \cdot \left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}} \cdot \left(\frac{1-\sqrt{5}}{2}\right)^n$$

Согласно (16.4.4)

$$m(h) = f_{h+2} - 1 = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2} - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{h+2} - 1$$

Но $\left|\frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^{h+2}\right| < 1$, следовательно, $m(h) + 1 > \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^{h+2}$

Введя обозначение $\gamma = \frac{1+\sqrt{5}}{2}$ и логарифмируя обе части последнего неравенства, получаем: $h + 2 < \frac{\log_2(m+1)}{\log_2 \gamma} + \frac{\log_2 \sqrt{5}}{\log_2 \gamma}$, откуда $h < 1.44 \cdot \log_2(m+1) - 0.32$

Таким образом, мы доказали, что для деревьев Фибоначчи с числом вершин m количество сравнений в худшем случае не превышает $1.44 \cdot \log_2(m+1) - 0.32$.

16.4.7. Итак, для искусственно построенных двоичных деревьев получены хорошие оценки. Следующий шаг – научиться строить естественные деревья с такими же хорошими оценками.