

# **Курс «Алгоритмы и алгоритмические языки»**

## **Лекция 18**

## ***Двоичные деревья поиска***

- ◇ Проблема: организовать хранилище данных, которое позволяет хранить большие объемы данных и предоставляет возможность быстро находить и модифицировать данные.
- ◇ *Хранилище данных* обеспечивает пользователю *интерфейс*, в котором определены *словарные операции*: *search* (найти, иногда называется *fetch*), *insert* (вставить) и *delete* (удалить).
- ◇ Варианты решения – деревья поиска, хеширование
- ◇ *Двоичное дерево* – набор узлов, который:
  - ◆ либо пуст (пустое дерево),
  - ◆ либо разбит на три непересекающиеся части:
    - узел, называемый *корнем*,
    - двоичное дерево, называемое *левым поддеревом*, и
    - двоичное дерево, называемое *правым поддеревом*.
- ◇ Двоичное дерево не является частным случаем обычного дерева, хотя у этих структур много общего. Основные отличия:
  - (1) Пустое дерево является двоичным деревом, но не является обычным деревом.
  - (2) Двоичные деревья  $(A(B, NULL))$  и  $(A(NULL, B))$  различны, а обычные деревья – одинаковы.

## Двоичные деревья поиска

- ◆ Структура для представления узла двоичного дерева поиска:

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

- ◆ Ключи в двоичном дереве поиска хранятся с соблюдением свойства упорядоченности:

Пусть  $x$  – произвольный узел двоичного дерева поиска.

Если узел  $y$  принадлежит левому поддереву, то

$$key[y] \leq key[x],$$

если  $y$  находится в правом поддереве узла  $x$ , то

$$key[y] \geq key[x].$$

## ***Двоичные деревья поиска: поиск узла***

◇ **На входе:** искомый ключ  $k$  и указатель  $root$  на корень поддерева, в котором производится поиск.

**На выходе:** указатель на узел с ключом  $key$  (если такой узел есть), либо пустой указатель NULL.

```
struct BT_node *Btsearch (struct BT_node *root, int k)
{
    if (! root || root->key == k)
        return root;
    if (k < root->key)
        return Btsearch (root->left, k);
    else
        return Btsearch (root->right, k);
}
```

## Двоичные деревья поиска: поиск узла

◇ Итеративная версия поиска.

```
struct BT_node *Btsearch (struct BT_node *root, int k)
{
    struct BT_node *p = root;

    while (p && p->key != k)
        if (k < p->key)
            p = p->left;
        else
            p = p->right;
    return p;
}
```

◇ Среднее время поиска  $O(h)$ , где  $h$  – высота дерева.

## ***Двоичные деревья поиска: минимум и максимум***

- ◆ **На входе:** указатель *root* на корень поддерева.  
**На выходе:** указатель на узел с минимальным ключом *k*.

```
struct BT_node *Btmin (struct BT_node *root)
{
    struct BT_node *p = root;
    while (p->left)
        p = p->left;
    return p;
}
```

- ◆ Среднее время выполнения  $O(h)$ , где  $h$  – высота дерева.

## Двоичные деревья поиска: следующий элемент

◆ На входе: указатель *node* на узел дерева.

На выходе: указатель на следующий за *node* узел дерева.

```
struct BT_node *Btsucc (struct BT_node *node) {
    struct BT_node *p = node, *q;
    /* I случай: правое поддерево узла не пусто */
    if (p->right)
        return Btmin (p->right);
    /* II случай: правое поддерево узла пусто,
       идем по родителям до тех пор, пока не найдем
       родителя, для которого наше поддерево левое */
    q = p->parent;
    while (q && p == q->right) {
        p = q;
        q = q->parent;
    }
    return q;
}
```

◆ Среднее время выполнения  $O(h)$ , где  $h$  – высота дерева.

## Двоичные деревья поиска: вставка

- ◆ **На входе:** указатель *root* на корень дерева и указатель *node* на новый узел, у которого есть значение ключа, а все поля с указателями имеют значение *NULL*.

```
void Btinsert (struct BT_node *root,
              struct BT_node *node) {
    struct BT_node *p, *q, *s;
    p = root, q = NULL;
    while (p) {
        q = p;
        p = (node->key < p->key) ? p->left : p->right;
    }
    node->parent = q;
    /* подвешиваем node к корню, либо к q слева или справа*/
    if (q == NULL)
        root = node;
    else if (node->key < q->key)
        q->left = node;
    else
        q->right = node;
}
```



## ***Двоичные деревья поиска: удаление***

- ◇ **На входе:** 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*.

## ***Двоичные деревья поиска: удаление***

- ◇ Шаг 1: если у  $n$  меньше двух детей, удаляем  $n$ , иначе удаляем  $succ(n)$ ; устанавливаем указатель  $u$  на удаляемый узел.
- ◇ Шаг 2: находим ребенка удаляемого узла и устанавливаем на него указатель  $x$ .
- ◇ Шаг 3: подвешиваем ребенка  $u$  (указатель  $x$ ) к родителю  $u$ ; если у  $u$  нет родителя, новым корнем дерева становится  $x$ ; устанавливаем в соответствующем поле родителя указатель на  $x$ , полностью исключая  $u$  из дерева.
- ◇ Шаг 4: если удаляемый узел  $succ(n)$ , заменяем данные узла  $n$  на данные узла  $succ(n)$ .

## Двоичные деревья поиска: удаление

```
struct BT_node BTdelete (struct BT_node **root,  
                        struct BT_node *n) {  
    struct BT_node *x, *y;  
    /* y - указатель на удаляемый узел n */  
    y = (! n->left || ! n->right) ? n : BT_succ (n);  
    /* x - указатель на ребенка y, либо NULL */  
    x = y->left ? y->left : y->right;  
    /* если x - ребенок y, вырезаем y */  
    if (x)  
        x->parent = y->parent;  
    /* если у y нет родителя, новым корнем дерева становится x */  
    if (! y->parent)  
        *root = x;  
    else {  
        /* x присоединяется к y->parent с требуемой стороны */  
        if (y == y->parent->left)  
            y->parent->left = x;  
        else  
            y->parent->right = x;  
    }  
    <...>  
}
```

## Двоичные деревья поиска: удаление

```
struct BT_node BTdelete (struct BT_node **root,  
                        struct BT_node *n) {  
    struct BT_node *x, *y;  
    <...>  
    /* 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  $O(h)$ , где  $h$  – высота дерева.

## Построение двоичного дерева поиска.

- ◆ **Постановка задачи.** Пусть имеется множество  $K$  из  $m$  ключей:

$$K = \{k_0, k_1, \dots, k_{m-1}\}$$

Разбиение  $K$  на три подмножества  $K_1, K_2, K_3$ :

- ◆  $|K_2| = 1, |K_1| \geq 0, |K_3| \geq 0.$
- ◆  $K_2 = \{k\} \Rightarrow \forall l \in K_1: l < k$  и  $\forall r \in K_3: r \geq k$

Далее по рекурсии: разбиваем  $K_1$  на  $K_{11}, K_{12}, K_{13}$   
 $K_3$  на  $K_{31}, K_{32}, K_{33}$

и т.д. пока ключи не кончатся

- ◆ **Пример:**  $K = \{15, 10, 1, 3, 8, 12, 4\}.$   
Первое разбиение:  $\{1, 3, 4\}, \{8\}, \{15, 10, 12\};$   
второе разбиение:  $\{\{1\}\{3\}\{4\}\}\{8\}\{\{10\}\{12\}\{15\}\}.$

Получилось полностью сбалансированное двоичное дерево.

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

## Построение двоичного дерева поиска.

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

$$m = 1 + 2 + 2^2 + \dots + 2^h = 2^h - 1$$

откуда  $h = \log_2(m + 1)$ .

- ◇ Если все  $m$  ключей известны заранее, их можно отсортировать за  $O(m \cdot \log_2 m)$ , после чего построение сбалансированного дерева будет тривиальной задачей.
- ◇ Если дерево строится по мере поступления ключей, то все может быть: от линейного дерева с высотой  $O(m)$  до полностью сбалансированного дерева с высотой  $O(\log_2 m)$ .
- ◇ Пусть  $T = \{root, left, right\}$  – двоичное дерево; тогда  $h_T = \max(h_{left}, h_{right}) + 1$ .

## Деревья Фибоначчи

◆ **Числа Фибоначчи** возникли в решении задачи о кроликах, предложенном в XIII веке Леонардо из Пизы, известным как Фибоначчи.

**Задача о кроликах:** 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 в конце  $n$ -го месяца?

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

$$f_3 = f_2 + f_1 = 2.$$

По индукции можно доказать, что для  $n \geq 3$

$$f_n = f_{n-1} + f_{n-2}.$$

## Деревья Фибоначчи

◇  $n$ -е число Фибоначчи вычисляет следующая функция:

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



## Деревья Фибоначчи

### ◆ **Определение** дерева Фибоначчи

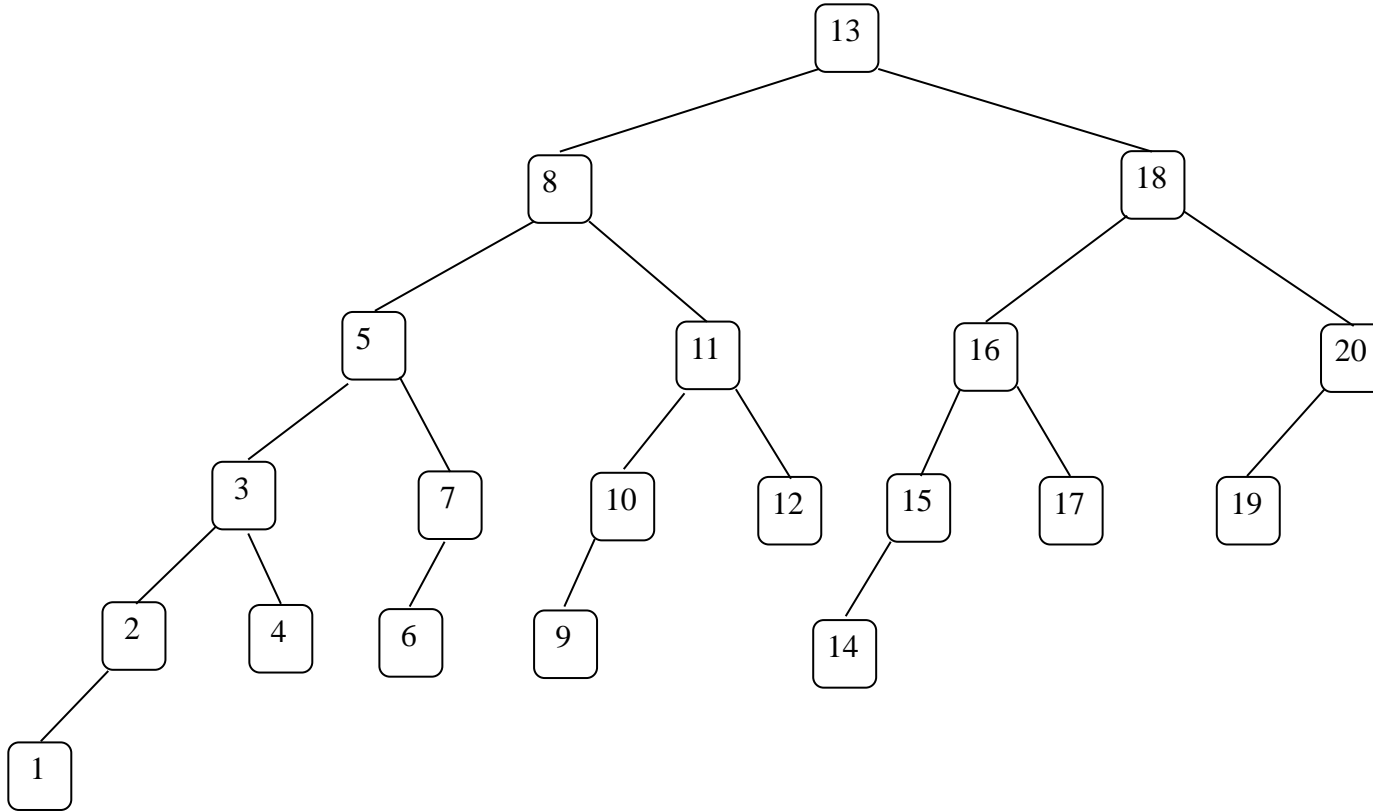
(это тоже искусственное дерево).

- (1) Пустое дерево – это дерево Фибоначчи с высотой  $h = 0$ .
- (2) Двоичное дерево, левое и правое поддереву 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 поддереву отличаются ровно на 1.

# Деревья Фибоначчи

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



## Деревья Фибоначчи

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

**Доказательство** (по индукции).

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

Шаг: по определению  $m(h) = m(h-1) + m(h-2) + 1$ .

$$\text{Имеем } m(h) = (f_{h+1} - 1) + (f_h - 1) + 1 = f_{h+2} - 1,$$

$$\text{так как } f_h + f_{h+1} = f_{h+2}$$

## Деревья Фибоначчи

◇ **Теорема 2.** Пусть  $C_1$  и  $C_2$  таковы, что уравнение

$$r^2 - C_1 r - C_2 = 0 \quad (*)$$

имеет два различных корня  $r_1$  и  $r_2$ ,  $r_1 \neq r_2$ .

Тогда для

$$a_n = \alpha_1 r_1^n + \alpha_2 r_2^n$$

выполняется соотношение

$$a_n = C_1 a_{n-1} + C_2 a_{n-2} .$$

**Доказательство.**  $r_1$  и  $r_2$  – корни уравнения (\*),

$$\begin{aligned} \text{то} \quad r_1^2 &= C_1 r_1 + C_2 \\ r_2^2 &= C_1 r_2 + C_2. \end{aligned}$$

Имеем:

$$\begin{aligned} C_1 a_{n-1} + C_2 a_{n-2} &= C_1 (\alpha_1 r_1^{n-1} + \alpha_2 r_2^{n-1}) + C_2 (\alpha_1 r_1^{n-2} + \alpha_2 r_2^{n-2}) = \\ &= \alpha_1 r_1^{n-2} (C_1 r_1 + C_2) + \alpha_2 r_2^{n-2} (C_1 r_2 + C_2) = \\ &= \alpha_1 r_1^{n-2} r_1^2 + \alpha_2 r_2^{n-2} r_2^2 = \alpha_1 r_1^n + \alpha_2 r_2^n = a_n \end{aligned} \quad (**)$$

Теорема доказана.

## Деревья Фибоначчи

◇ **Теорема 3.** Пусть  $C_1$  и  $C_2$  таковы, что уравнение

$$r^2 - C_1 r - C_2 = 0 \quad (*)$$

имеет два корня  $r_1$  и  $r_2$ ,  $r_1 \neq r_2$ .

Тогда

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  $n = 1, 2, \dots$

**Доказательство.** Нужно не только повторить в обратном порядке вывод (\*\*), но и подобрать такие  $\alpha_1$  и  $\alpha_2$ , чтобы

$$a_0 = \alpha_1 + \alpha_2, \quad a_1 = \alpha_1 r_1 + \alpha_2 r_2 \quad (***)$$

Рассматривая (\*\*\*) как систему линейных уравнений относительно  $\alpha_1$  и  $\alpha_2$ , 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.

## Деревья Фибоначчи

◇ Применим доказанные теоремы к числам Фибоначчи:

$$f_n = f_{n-1} + f_{n-2}.$$

Уравнение  $r^2 - r - 1 = 0$  имеет корни

$$r_1 = \frac{1 + \sqrt{5}}{2} \qquad r_2 = \frac{1 - \sqrt{5}}{2}$$

Следовательно, согласно теореме 3

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

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

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

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

## Деревья Фибоначчи

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 теореме 1

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

$$\left| \frac{1}{\sqrt{5}} \left( \frac{1-\sqrt{5}}{2} \right)^{h+2} \right| < 1$$

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

## Деревья Фибоначчи

Обозначение  $\gamma = \frac{1 + \sqrt{5}}{2}$   $m(h) + 1 > \frac{1}{\sqrt{5}} \gamma^{h+2}$  (\*\*\*)

Логарифмируя обе части (\*\*\*) , 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

$$1,44 \cdot \log_2(m + 1) - 0,32$$