Тема 4. Деревья

Бинарные деревья поиска

Определение. Бинарное дерево называется деревом поиска, если для любого узла A выполнено соотношение

```
x < A.value < y
```

где x - произвольное значение из левого поддерева, y - произвольное значение из правого поддерева.

Можно дать определение с нестрогим неравенством `x <= A.value < y` , но на практике все интересные случаи используют строгие неравенства, поэтому их и имеем в виду.

Такое определение сразу задает правила работы с деревом поиска, а именно, сам поиск значения, и операции добавления и удаления элементов так, чтобы поддерживалось определение.

Далее в этом тексте будем писать не совсем строгий код, чтобы не загромождать и иллюстрировать именно логику работы (т.е. опустим template, строгое описание типов и т.п.). А строгий код будет уже в рабочих примерах.

Возможны рекурсивные и нерекурсивные реализации.

Поиск. Хотим узнать, есть ли в дереве конкретное значение X и в каком узле оно находится

```
TreeNode * Search(TreeNode * root, const T & x)
{
    if (root == nullptr) return nullptr;
    if (x == root->value) return root;
    if (x > root->value) return Search(root->right, x);
    else  return Search(root->left, x);
}

TreeNode * Search(TreeNode * root, T & x)
{
    if (root == nullptr) return nullptr;
    TreeNode *p = root;
    while (p) {
       if (x == p->value) return p;
       p = (x > p->value) ? p->right : p->left;
    }
    return nullptr;
}
```

Эти процедуры возвращают указатель на узел дерева. Однако идеология контейнерных структур использует понятие итератора для обозначение позиции доступа к элементу. Поэтому по-хорошему здесь нужно было бы начинать поиск с позиции итератора и результат поиска возвращать как итератор. Но пока не будем усложнять код и оставим указатель на вершины делева.

Добавление. Процедура возвращает указатель на вершину всего дерева после добавления

```
TreeNode * Add(TreeNode * root, const T & x)
{
    if (root == nullptr) {
        root = new TreeNode;
```

```
        root->value = x;
    } else {                    // нет проверки на наличие x в дереве !!!
        if (x > root->value) {
            root->right = Add(root->right, x);
            root->right->parent = root;
        } else {
            root->left = Add(root->left, x);
            root->left->parent = root;
        }
    }
    return root;
 }

TreeNode * Add(TreeNode * root, const T & x)
{
    TreeNode *px = new TreeNode;
    px->value = x;
    if (root == nullptr) {
        root = px;
    } else {          // нет проверки на наличие x в дереве !!!
        for (TreeNode *p = root; p; ) {
            if (x > p->value) {
                if (p->right) { p = p->right; }
                else { p->right = px;
                       px->parent = p;
                       break;
                }
            } else {
                if (p->left) { p = p->left; }
                else { p->left = px;
                       px->parent = p;
                       break;
                }
            }
        }
    }
    return root;
 }
```

Удаление. Иллюстрация на картинках.

```
TreeNode * Remove(TreeNode * root, const T & x)
{
    TreeNode *p;
    if (root == nullptr) return nullptr;
    if (x > root->value) {
        root->right = Remove(root->right, x);
        if (root->right) root->right->parent = root;
    } else if (x < root->value) {
        root->left = Remove(root->left, x);
        if (root->left) root->left->parent = root;
    } else {
        if (root->right == nullptr) {
            p = root->left;
            delete root;
```

```
            return p;
        }
        if (root->left == nullptr) {
            p = root->right;
            delete root;
            return p;
        }
        p = SearchRightmost (root->left);  // но можно и сразу искать и удалять
        root->value = p->value;
        root->left = Remove(root->left, root->value);
        if (root->left) root->left->parent = root;
    }
    return root;
}

TreeNode * p = SearchRightmost (TreeNode *root)
{
    for ( ; root->right; root = root->right);
    return root;
}
```

Нерекурсивный вариант — в качестве самостоятельного упражнения.
Нигде не использовался указатель parent.
1 — можно его учитывать и соответственно устанавливать
2 — можно не учитывать и вообще убрать из реализации
Недостатки — дерево может выродиться в список, трудоемкость $O(N)$.