Деревья

Для представления различного рода иерархических данных используются деревья — для представления синтаксических структур в компиляторах, в СУБД и т.д.

Дерево — это совокупность элементов, называемых узлами (один из которых — корень), и отношений, образующих иерархическую структуру узлов.

Рекуррентное определение:

1. Узел является деревом, он же — корень дерева.
2. Если есть узел n и деревья T_1, T_2, ..., T_k — деревья с корнями n_1, n_2, ..., n_k соответственно. Тогда можно построить новое дерево, с корнем n и поддеревьями T_1, T_2, ..., T_k. Узлы n_1, n_2, ..., n_k называются сыновьями узла n.

Часто добавляют ещё нулевое дерево — дерево без узлов.

Ещё определения:

Дерево — связный ациклический граф.

Несвязный ациклический граф называют лесом.

Терминология

Путём из n_1 в n_k называют последовательность узлов n_1, ..., n_k, в котором каждый узел является родителем следующего. Длина пути — число, на единицу меньшее количества узлов, составляющих путь.

Путь нулевой длины — путь из узла к самому себе.

Узел a называется **предком** узла b, если существует путь из a в b. b в этом случае — потомок a. Каждый узел — предок и потомок самого себя. Потомок, не являющийся самим узлом, называется истинным потомком, с предком аналогично.

Узел, не имеющий истинных потомков, называется **листом**.

Поддерево какого-либо дерева — узел вместе со всеми потомками.

Высота узла — длина самого длинного пути из узла до какого-либо листа, глубина узла — длина пути от узла до корня. Высота дерева — высота корня.

Деревья бывают упорядоченными и неупорядоченными. Можно упорядочить узлы дерева, не связанные отношением предок-потомок (слева-справа).

Обходы дерева

Прямой порядок, симметричный (или внутренний) и обратный (preorder, inorder и postorder соответственно). Задаёт упорядочивание узлов.

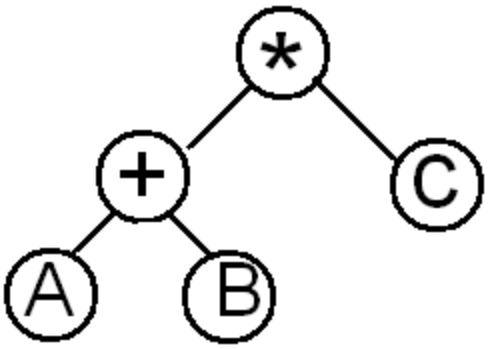
Прямой порядок — сначала корень, потом сыновья по порядку.

Симметричный порядок — сначала первый сын, затем корень, затем остальные сыновья.

Обратный порядок — сначала сыновья, потом корень.

Помеченные деревья, деревья выражений

Узлам можно сопоставить метки — значения, хранящиеся в узлах.
Дерево выражений, для (a + b) * c:



Листья — операнды, внутренние узлы — операторы. При обходе дерева выражений в прямом порядке получаем префиксную запись, в обратном — польскую, в симметричном — инфиксную.

АТД "дерево" — используется в реализации других АТД, например, множеств (как — будет рассказано позже)
Операции:

```
parent(node, tree)
leftmostChild(node, tree)
rightSibling(node, tree)
label(node, tree)
create(v, t1, ..., ti)
root(tree)
makenull(tree)
```

Пример — обход в прямом порядке:

```
void preorder(Node n)
{
   cout << label(n);
   Node child = leftmostChild(n);
   while (child != NULL)
   {
       preorder(child);
       child = rightSibling(child);
   }
}
```

Нерекурсивный обход в прямом порядке:

```
void nonRecursivePreorder(Node n)
{
   stack<Node> s;
```

```
    Node m = n;
    while (true)
    {
        if (m != NULL)
        {
            cout << label(m) << " ";
            s.push(m);
            m = leftmostChild(m);
        }
        else
        {
            if (s.empty())
                return;
            m = rightSibling(s.top());
            s.pop();
        }
    }
}
```

Реализация деревьев списком сыновей:

```
struct Node
{
   ElementType value;
   Node *sibling;
   Node *child;
};
```

Тогда:

```
Node* leftmostChild(Node *n)
{
   return n->child;
}

Node* rightSibling(Node *n)
{
   return n->sibling;
}
```

parent обычно не нужен, при рекурсивном обходе дерева родитель известен сам собой. При желании можно поле parent добавить. При желании можно переиспользовать АТД "список", но тогда каждый узел должен знать свою позицию в списке.

Двоичные деревья

Двоичные деревья — это деревья, в которых у каждого узла есть левый и правый сын, причём это принципиально разные вещи. Двоичые деревья удобно представлять следующей структурой:

```
struct BinaryTreeNode
{
   ElementType value;
   Node *leftChild;
   Node *rightChild;
};

struct BinaryTree
{
    BinaryTreeNode *root;
};
```

Представление двоичных деревьев в массиве уже было — хипсорт.

Двоичные деревья, например, хорошо подходят для реализации алгоритма Хаффмана: считаем частоты символов в тексте, берём два символа с наименьшими частотами о "объединяем их в один" с помощью дерева, сложив частоты, пока в дереве не будут все символы. Пометим рёбра до левых сыновей нулями, до правых — единицами. Путь от корня до дерева — код символа.

Двоичные деревья поиска (binary search trees, BST)

Задача — реализовать операции проверки принадлежности элемента ко множеству, вставки и удаления за O(log n).

Двоичное дерево поиска — дерево, узлы которого помечены элементами множества, над которым определено отношение полного порядка.

Определяющее свойство BST — все элементы, хранящиеся в узлах левого поддерева любого узла x, меньше элемента, содержащегося в x, а в узлах правого поддерева — больше.

Если обойти двоичное дерево в симметричном порядке, получится список элементов, упорядоченный по возрастанию.

Определение принадлежности элемента множеству:

- Если элемент равен элементу в корне, то да.
- Если меньше, то проверим принадлежность левому поддереву, если больше — проверим принадлежность правому поддереву:

Добавление:

- Если дерево пусто, заводим новый узел и делаем его корнем дерева.
- Если значение в дереве равно добавляемому элементу, то ничего добавлять не надо, если меньше — добавляем в левое поддерево, если больше — в правое:

Удаление:

- Если дерево пусто, то ок.
- если узел внутренний, то его удалить не очень тривиально. Ищем элемент для

удаления.

- Если удаляемый узел — лист, удаляем его.
- Если у узла только левый сын, удаляем узел, а на его место ставим левого сына. Если только правый сын — аналогично.
- Если есть оба сына, то среди правого поддерева находим самый маленький элемент, удаляем его оттуда и ставим на место удаляемого (как вариант — среди левого поддерева самый большой).

Для этого может быть полезна вспомогательная процедура, удаляющая минимальный элемент из дерева и возвращающая его значение.