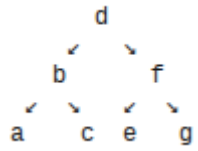
Несмотря на то, что бывает много различных типов деревьев, двоичные деревья играют особую роль, так как в отсортированном состоянии позволяют очень быстро выполнять вставку, удаление и поиск. Каждый элемент двоичного дерева состоит из информационной части и указателей на левый и правый элементы. В некотором смысле двоичное дерево является особым видом связанного списка. Элементы можно вставлять, удалять и извлекать в любом порядке. Кроме того, операция извлечения не является разрушающей. Несмотря на то, что деревья легко представить в воображении, в теории программирования с ними связан ряд сложных задач. Большинство функций, работающих с деревьями, рекурсивны, поскольку дерево по своей сути является рекурсивной структурой данных. Другими словами, каждое поддерево, в свою очередь, является деревом. Процесс поочередного доступа к каждой вершине дерева называется обходом (вершин) дерева (tree traversal). Существует три порядка обхода дерева: обход симметричным способом, или симметричный обход (inorder), обход в прямом порядке, прямой обход, упорядоченный обход, обход сверху, или обход в ширину (preorder) и обход в обратном порядке, обход в глубину, обратный обход, обход снизу (postorder). При симметричном обходе обрабатывается сначала левое поддерево, затем корень, а затем правое поддерево. При прямом обходе обрабатывается сначала корень, затем левое поддерево, а потом правое. При обходе снизу сначала обрабатывается левое поддерево, затем правое и, наконец корень. Последовательность доступа при каждом методе обхода показана:

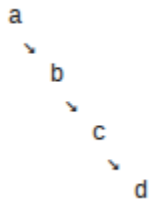


Симметричный обход	**a b c d e f g**
Прямой обход	**d b a c f e g**
Обход снизу	**a c b e g f d**

```
struct tree {
  char info;
  struct tree *left;
  struct tree *right;
};


void inorder(struct tree *root)
{
  if(!root) return;

  inorder(root->left);
  if(root->info) printf("%c ", root->info);
  inorder(root->right);
}

void preorder(struct tree *root)
{
  if(!root) return;

  if(root->info) printf("%c ", root->info);
  preorder(root->left);
  preorder(root->right);
}
```

```
void postorder(struct tree *root)
{
  if(!root) return;

  postorder(root->left);
  postorder(root->right);
  if(root->info) printf("%c ", root->info);
}
```

Если вы запускали программу печати дерева, вы, вероятно, заметили, что некоторые деревья являются *сбалансированными* (balanced), т.е. каждое поддерево имеет примерно такую же высоту, как и остальные, а некоторые деревья очень далеки от этого состояния. Например, дерево **a⇒b⇒c⇒d** выглядит следующим образом:



В этом дереве нет левых поддеревьев. Такое дерево называется *вырожденным*, поскольку фактически оно выродилось в линейный список. В общем случае, если при построении дерева вводимые данные являются случайными, то получаемое дерево оказывается близким к сбалансированному. Если же информация предварительно отсортирована, создается вырожденное дерево. (Поэтому иногда при каждой вставке дерево корректируют так, чтобы оно было сбалансированным, но этот процесс довольно сложен)

К сожалению, удалить вершину дерева не так просто, как отыскать. Удаляемая вершина может быть либо корнем, либо левой, либо правой вершиной. Помимо того, к вершине могут быть присоединены поддеревья (количество присоединенных поддеревьев может равняться 0, 1 или 2). Процесс переустановки указателей подсказывает рекурсивный алгоритм, приведенный ниже:

```
struct tree *dtree(struct tree *root, char key)
{
  struct tree *p,*p2;

  if(!root) return root; /* вершина не найдена */

  if(root->info == key) { /* удаление корня */
    /* это означает пустое дерево */
    if(root->left == root->right){
      free(root);
      return NULL;
    }
    /* или если одно из поддеревьев пустое */
    else if(root->left == NULL) {
      p = root->right;
      free(root);
      return p;
    }
    else if(root->right == NULL) {
      p = root->left;
      free(root);
      return p;
```

```
    }
    /* или есть оба поддерева */
    else {
      p2 = root->right;
      p = root->right;
      while(p->left) p = p->left;
      p->left = root->left;
      free(root);
      return p2;
    }
  }
  if(root->info < key) root->right = dtree(root->right, key);
  else root->left = dtree(root->left, key);
  return root;
}
```

Необходимо также следить за правильным обновлением указателя на корень дерева, описанного вне данной функции, поскольку удаляемая вершина может быть корнем. Лучше всего с этой целью указателю на корень присваивать значение, возвращаемое функцией `dtree()`:

```
root = dtree(root, key);
```

Двоичные деревья — исключительно мощное, гибкое и эффективное средство. Поскольку при поиске в сбалансированном дереве выполняется в худшем случае $\log_2 n$ сравнений, оно намного лучше, чем связанный список, в котором возможен лишь последовательный поиск.