## 4. Деревья

Деревья - широко распространенные структуры данных в информатике и программировании, которые представляют собой иерархические структуры в виде набора связанных узлов.

**Дерево** – это структура данных, представляющая собой совокупность элементов, образующих иерархическую структуру(рис. 4.1). Каждый элемент дерева называется *вершиной (узлом) дерева*. Вершины дерева соединены направленными дугами, которые называют *ветвями дерева*. Начальный узел дерева называют *корнем дерева*, ему соответствует нулевой уровень. *Листьями дерева* называют вершины, в которые входит одна ветвь и не выходит ни одной ветви.

Каждое дерево обладает следующими свойствами:

1. существует узел, в который не входит ни одной дуги (корень);
2. в каждую вершину, кроме корня, входит одна дуга.

Деревья особенно часто используют на практике при изображении различных иерархий. Например, популярны генеалогические деревья.

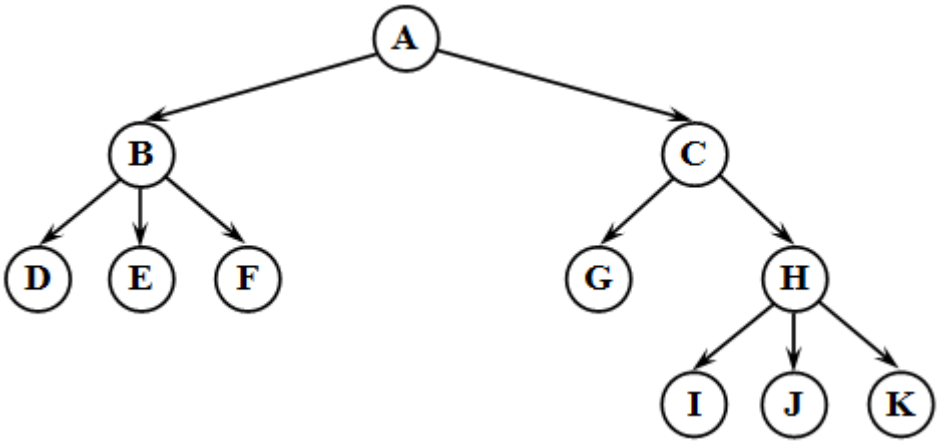


**Рис. 4.1.** Дерево

Все вершины, в которые входят ветви, исходящие из одной общей вершины, называются *потомками*, а сама вершина – *предком*. У каждого предка может быть несколько потомков. *Уровень* потомка на единицу превосходит уровень его предка. Уровень корня равен нулю. Корень дерева не имеет предка, а листья дерева не имеют потомков.

**Высота** *(глубина)* дерева определяется количеством уровней, на которых располагаются его вершины. Высота пустого дерева равна нулю, высота дерева из одного корня – единице. На первом уровне дерева может быть только одна вершина – корень дерева, на втором – потомки корня дерева, на третьем – потомки потомков корня дерева и т.д.

**Поддерево** – часть древовидной структуры данных, которая может быть представлена в виде отдельного дерева.

**Степенью вершины** в дереве называется количество дуг, которое из нее выходит. **Степень дерева** равна максимальной степени вершины, входящей в дерево. При этом листьями в дереве являются вершины, имеющие степень нуль. По величине степени дерева различают два типа деревьев:

- двоичные – степень дерева не более двух;
- сильноветвящиеся – степень дерева произвольная.

**Упорядоченное дерево** – это дерево, у которого ветви, исходящие из каждой вершины, упорядочены по определенному критерию.

Деревья являются рекурсивными структурами, так как каждое поддерево также является деревом. Таким образом, дерево можно определить как рекурсивную структуру, в которой каждый элемент является:

- либо пустой структурой;
- либо элементом, с которым связано конечное число поддеревьев.

Действия с рекурсивными структурами удобнее всего описываются с помощью рекурсивных алгоритмов.

Списочное представление деревьев основано на элементах, соответствующих вершинам дерева. Каждый элемент дерева, степень которого выше двух, имеет поле данных и два поля указателей: указатель на начало списка потомков вершины и указатель на следующий элемент в списке потомков текущего уровня. При таком способе представления дерева обязательно следует сохранять указатель на вершину, являющуюся корнем дерева.

Каждый элемент дерева, степень которого не выше двух, имеет поле данных и два поля указателей: указатель на левого потомка вершины и указатель на правого потомка.

Для того, чтобы выполнить операцию над всеми вершинами дерева, необходимо все его вершины просмотреть. Такая задача называется *обходом дерева*.

При обходе все вершины дерева должны посещаться в определенном порядке. Существует несколько способов обхода всех вершин дерева. Выделим три наиболее часто используемых способа обхода дерева (рис. 4.2):

- прямой;
- симметричный;
- обратный.

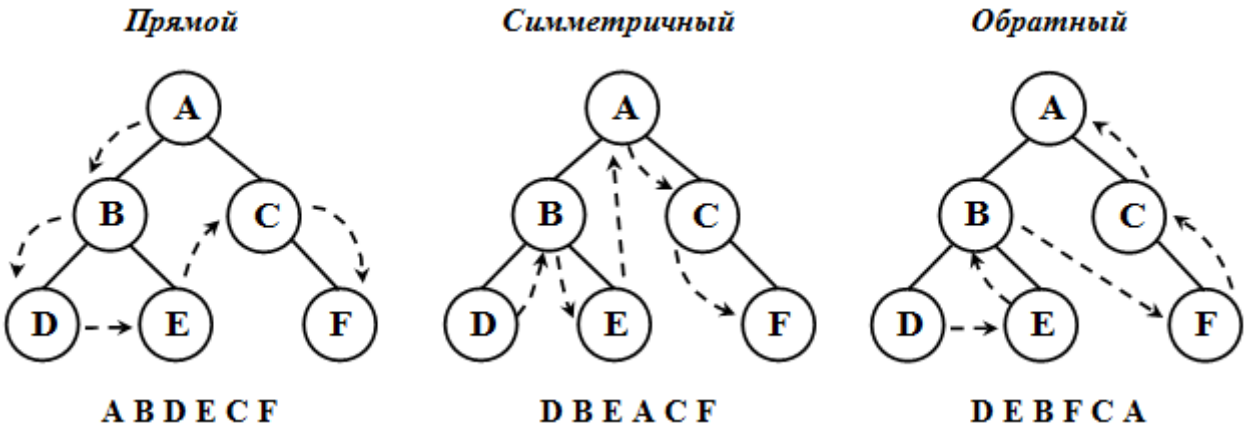


**Рис. 4.2.** Обходы деревьев

Существует большое многообразие древовидных структур данных. Выделим самые распространенные из них: бинарные (двоичные) деревья, красно-черные деревья, АВЛ-деревья и т.д.

Путь в дереве – это список вершин, в котором соседние вершины пути соединены ребрами.

Важным свойством дерева является наличие только одного пути, соединяющего 2 вершины. Несвязанный набор вершин называется бором. Обычно бор упорядочен.

Отметим, что сильно ветвящееся дерево и бор можно представить в виде бинарного дерева.

## Бинарные деревья

**Бинарное (двоичное) дерево** – это динамическая структура данных, представляющая собой дерево, в котором каждая вершина имеет не более двух потомков (рис. 4.3). Таким образом, бинарное дерево состоит из элементов, каждый из которых содержит информационное поле и *не более двух* ссылок на различные бинарные поддеревья. На каждый элемент дерева имеется ровно одна ссылка.

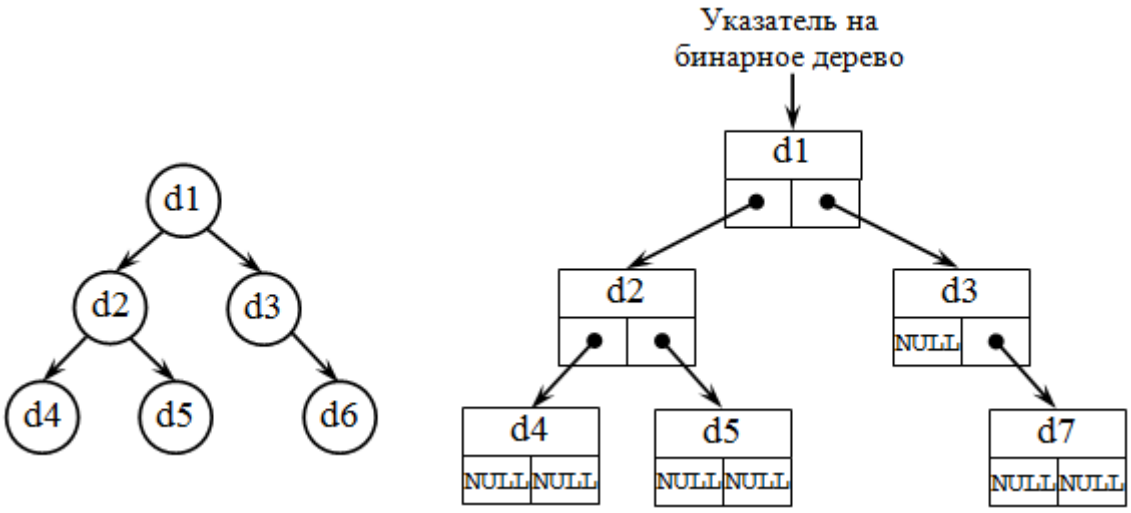


**Рис. 4.3.** Бинарное дерево и его организация

Каждая вершина бинарного дерева является структурой, состоящей из четырех видов полей. Содержимым этих полей будут соответственно:

- информационное поле (ключ вершины);
- служебное поле (их может быть несколько или ни одного);
- указатель на левое поддерево;
- указатель на правое поддерево.

По степени вершин бинарные деревья делятся на (рис. 4.4):

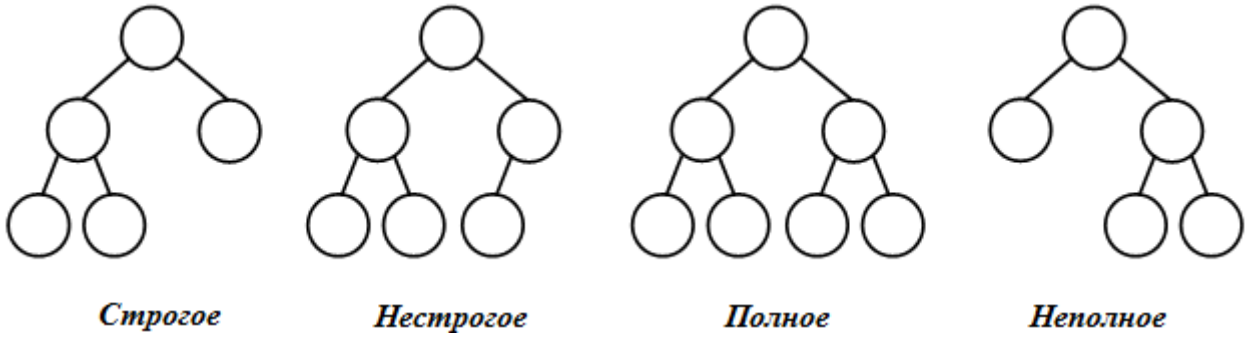


**Рис. 4.4**

- *строгие* – вершины дерева имеют степень ноль (у листьев) или два (у узлов);
- *нестрогие* – вершины дерева имеют степень ноль (у листьев), один или два (у узлов).

В общем случае у бинарного дерева на `k`-м уровне может быть до $2^{k-1}$ вершин. Бинарное дерево называется *полным*, если оно содержит только полностью заполненные уровни. В противном случае оно является *неполным*.

Дерево называется *сбалансированным*, если длины всех путей от корня к внешним вершинам равны между собой. Дерево называется *почти сбалансированным*, если длины всевозможных путей от корня к внешним вершинам отличаются не более, чем на единицу.

Бинарное дерево может представлять собой пустое множество. Бинарное дерево может выродиться в список (рис. 4.5).

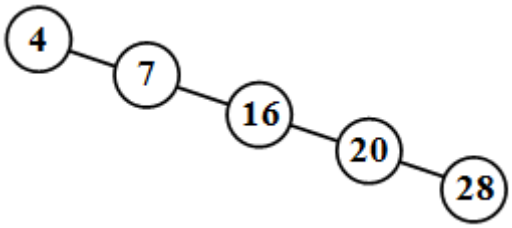


**Рис. 4.5.** Список как частный случай бинарного дерева

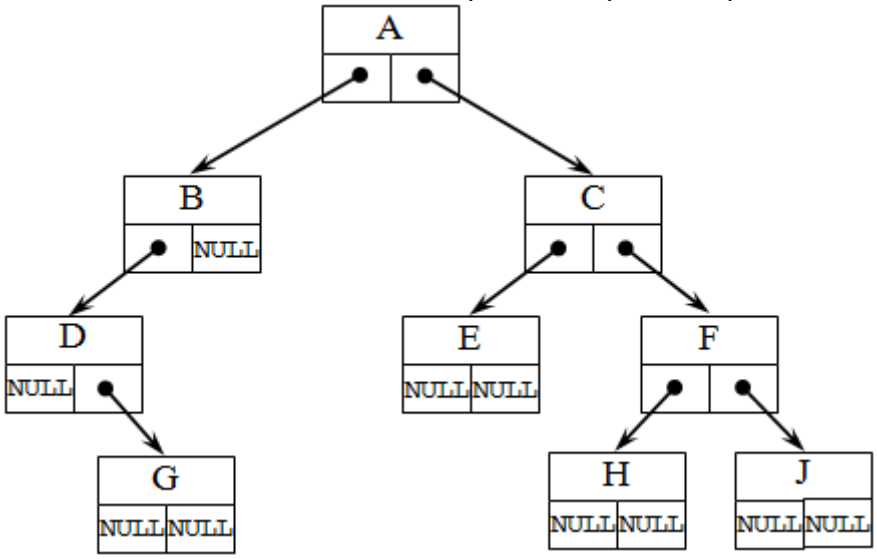


**Рис. 4.6.** Адресация в бинарном дереве

Бинарные деревья могут применяться для поиска данных в специально построенных деревьях (базы данных), сортировки данных, вычислений арифметических выражений, кодирования (метод Хаффмана) и т.д.

## Бинарное дерево. Характеристики.

**Высота** – максимальный уровень в дереве.
**Длина пути** от корня до узла соответствует уровню узла.
**Длина внутреннего пути** – сумма длин путей до всех узлов дерева(их иногда называют внутренними).
**Внешний узел** – обозначение позиции вставки нового узла в бинарном дереве.

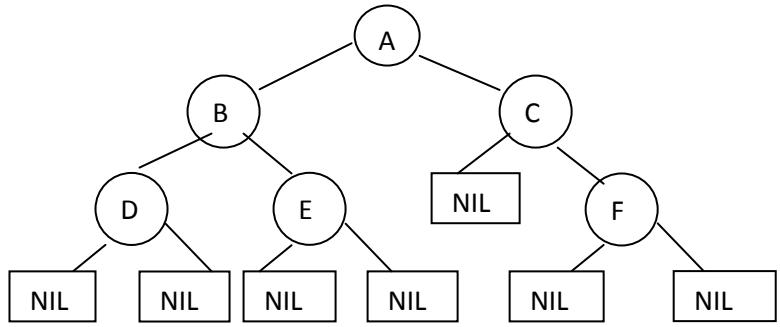

Рис. 4.7

На данном рисунке внешние узлы помечены NIL.

**Длина внешнего пути** – сумма длин путей до всех внешних узлов дерева.

Высота бинарного дерева с N узлами не меньше Ln N и не больше N-1.

В дальнейшем для простоты будем работать с бинарными деревьями поиска. Отличительной особенностью которых является то, что ключевые поля отца, левого и правого сыновей связаны соотношениями: левый_сын(ключ) меньше отец(ключ); Правый_сын(ключ) больше отец(ключ).
Описание узла бинарного дерева выглядит следующим образом:

```
struct node {
            int data;//информационное поле
            int count; //служебное поле
            node *left;//адрес левого поддерева
            node *right;//адрес правого поддерева
           };
```

Основными операциями, осуществляемыми с бинарными деревьями, являются:

- создание бинарного дерева;
- печать бинарного дерева;
- обход бинарного дерева;
- вставка элемента в бинарное дерево;
- удаление элемента из бинарного дерева;
- проверка пустоты бинарного дерева;
- удаление бинарного дерева.
- поиск в бинарном дереве.

Для описания алгоритмов этих основных операций используется следующее объявление:

```
struct BinaryTree{
      int Data; //поле данных
      BinaryTree* Left; //указатель на левый потомок
      BinaryTree* Right; /указатель на правый потомок
};
. . . . . .
BinaryTree* BTree = NULL;
```

Приведем функции перечисленных основных операций при работе с бинарным деревом.

```
//добавление узла в  бинарное дерево
void Add_Binary_Tree(BinaryTree** Node, int key){
  BinaryTree* ptr;//вспомогательный указатель
  int fl=1;
if(&Node == NULL) { ptr= new BinaryTree;
                    ptr->Data=key;
ptr-> Left= ptr-> Right=NULL;
```

```
                    &Node= ptr;
                      }
else{ptr=&Node;
  while (fl==1) {
    if(ptr ->Data > key)  if(ptr-> Left==NULL) { ptr-> Left= new BinaryTree;
                                                 ptr= ptr-> Left;
                                                 ptr->Data=key;
                                                 ptr-> Left= ptr-> Right=NULL;
                                                 fl=0;
                                                }
                                          else ptr= ptr-> Left;
                          else
                       if(ptr ->Data < key)  if(ptr-> Right ==NULL)
                                                  { ptr-> Right = new BinaryTree;
                                                    ptr= ptr-> Right;
                                                     ptr->Data=key;
                                                     ptr-> Left= ptr-> Right=NULL;
                                                     fl=0;
                                                     }
                                                else ptr=ptr->Right;
                                 else fl=2;
                    }
}



//прямой обход бинарного дерева
void PreOrder_BinaryTree(BinaryTree* Node){
BinaryTree* ptr;
ptr=Node;
int fl=0;
int uv=-1;
BinaryTree*st[n];//n- определяет размер стека
while(fl==0){
  if (ptr== NULL) if (uv==-1) fl=1;
                          else{ptr=st[uv--];
                               ptr=ptr->Right;}
     else{ <обработка узла>
            st[++uv]=ptr;
            ptr=ptr->Left;}
}
}
//симметричный обход бинарного дерева
Реализовать самостоятельно на основе прямого обхода

//печать бинарного дерева
void Print_BinaryTree(BinaryTree* Node){
  int i;
BinaryTree* ptr;

int fl=0;
int uv;
BinaryTree*st[n];//n- определяет размер стека
while(fl==0){
  if (ptr== NULL) if (uv==-1) fl=1;
                          else{ptr=st[uv--];
                               ptr=ptr->Right;}
     else{ printf ("%4ld", ptr->Data);
           if(ptr->Left!=NULL) printf ("%4ld", ptr-> Left ->Data);else printf(("----");
```

```
            if(ptr-> Right!=NULL) printf ("%4ld", ptr-> Right ->Data); else printf(("----");
            cout << endl;
             st[uv++]=ptr;
             ptr=ptr->Left;}
}
.// поиск в бинарном дереве
void Find_Node_BinaryTree(BinaryTree* Node,int Data, BinaryTree** Ad){
BinaryTree* ptr;
 (*Ad)=NULL;
  if ( Node != NULL ){
    if (Node->Data == Data) (*Ad)=Node;
        else{ptr=Node;
              while((!Ad)&&(ptr))if(ptr->Data==Data) (*Ad)=Node;
                                       else if(ptr->Data>Data) ptr=ptr->Left;
                                               else ptr=ptr->Right;}
  }
//удаление вершины из бинарного дерева
void Delete_Node_BinaryTree(BinaryTree** Node,int Data){
BinaryTree *Node1;

  if ( (*Node) != NULL ){
    if ((*Node)->Data == Data){
      BinaryTree* ptr = (*Node);
      if ( (*Node)->Left == NULL && (*Node)->Right == NULL ) (*Node) = NULL;
      else if ((*Node)->Left == NULL) (*Node) = ptr->Right;
      else if ((*Node)->Right == NULL) (*Node) = ptr->Left;
      else {
        Node1 = ptr->Right;
        BinaryTree * ptr1;
        ptr1 = Node1;
        while (*ptr1 != NULL) {Node1=ptr1;
          ptr1 = ptr1->Left;}
        (*ptr1) = ptr->Left;
      }
      delete(ptr);
      Delete_Node_BinaryTree(Node,Data);
    }
    else {
      Delete_Node_BinaryTree(&((*Node)->Left),Data);
      Delete_Node_BinaryTree(&((*Node)->Right),Data);
    }
  }
}

//проверка пустоты бинарного дерева
bool Empty_BinaryTree(BinaryTree* Node){
  return ( Node == NULL ? true : false );
}
```

**Прошитые деревья**

Эффективность обхода дерева рекурсивными и нерекурсивными алгоритмами может быть увеличена, если использовать указатели на отсутствующие поддеревья для хранения в них адресов узлов преемников, которые надо посетить при заданном порядке обхода бинарного дерева. Такой указатель называется нитью или прошитой связью.. Его следует отличать от указателей на  левого и правого потомков. Операция, заменяющая указатели NULL на нити, называется прошивка. Она может выполняться по-разному. Если нити заменяют указатели NULL в узлах с пустыми правыми

поддеревьями, при просмотре в симметричном порядке, то бинарное дерево называется симметрично прошитым справа. Похожим образом может быть определено бинарное дерево, симметрично прошитое слева: дерево, в котором каждый левый NULL указатель изменен так, что он содержит нить – связь к предшественнику данного узла при просмотре в симметричном порядке. Симметрично прошитое бинарное дерево – это такое дерево, которое симметрично прошито слева и справа. Однако левая прошивочная нить не дает тех преимуществ, что правая прошивочная нить. На рис. 6.1 показано дерево, симметрично прошитое справа. На нем пунктирными линиями обозначены прошивочные нити.

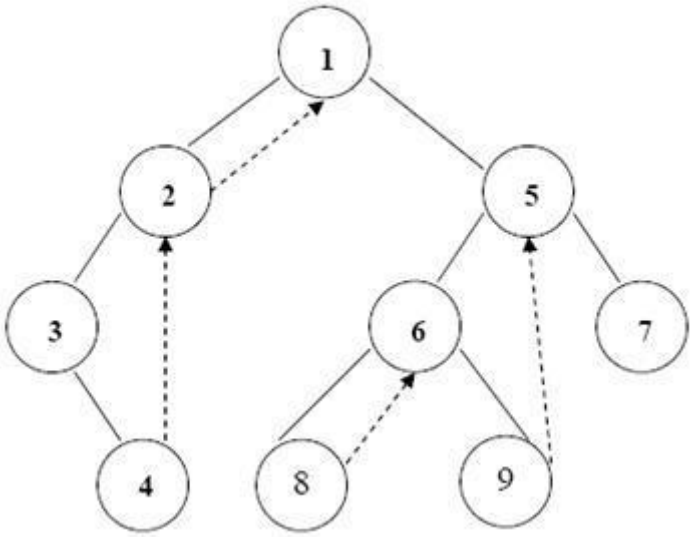


Рис. 4.8 – Симметрично прошитое справа бинарное дерево

Также используются бинарные деревья, прямо прошитые справа и слева. В них NULL правые и левые указатели узлов заменены соответственно на их преемников и предшественников при прямом порядке обхода. Прошитые деревья эффективно обходятся  без использования стека.

Поскольку нужно каким-то образом отличать обычную связь от прошивочной нити, каждому узлу добавляется два однобитовых (логических) поля: ltag и rtag. Если значение поля true, соответствующее поле связи является обычной связью, в случае значения false – прошивочной нитью.

Рассмотрим вставку новой вершины слева от заданной в симметрично прошитое бинарное дерево (рис. 4.7). На рис. 4.8 показано результирующее дерево