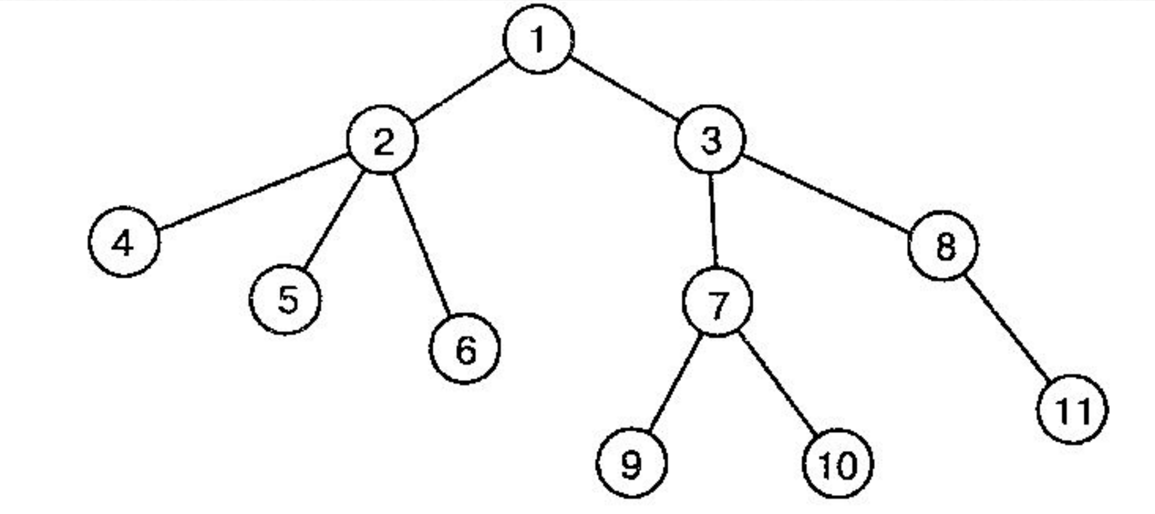
Динамические структуры данных

Прикладное программирование

Деревья – основные понятия

Связной граф без циклов называется деревом. Пример дерева приведен на рисунке.



Деревья – основные понятия

- Следующие определения дерева эквивалентны. Граф G=(V, E), где |V| =N и |E| =M является деревом, если:
- G – связный граф и M=N-1;
- G – ациклический граф и M=N-1;
- любые две несовпадающие вершины графа G соединяет единственная простая цепь;
- G – ациклический граф, обладающий тем свойством, что если какую-либо пару несмежных вершин соединить ребром, то полученный граф будет содержать ровно один цикл.

Деревья – основные понятия

- Терминология, связанная с понятием дерева:
- корнем дерева называют единственную вершину, находящуюся вверху «перевернутого» дерева;
- самые нижние вершины дерева называют листьями;
- вершину называют внутренней, если она не является ни корнем и ни листом;
- о вершине, которая находится непосредственно над другой вершиной, говорят, что она – родитель (предок), а вершина, которая расположена непосредственно под другой вершиной, называется потомком.

Деревья – основные понятия

- На рисунке вершина с номером 1 – корень; вершины с номерами 4, 5, 6, 9, 10, 11 – листья; вершины 2, 3, 7, 8 – внутренние; вершина 3- родитель вершины 7; вершина 8 – потомок вершины 3.

Деревья – основные понятия

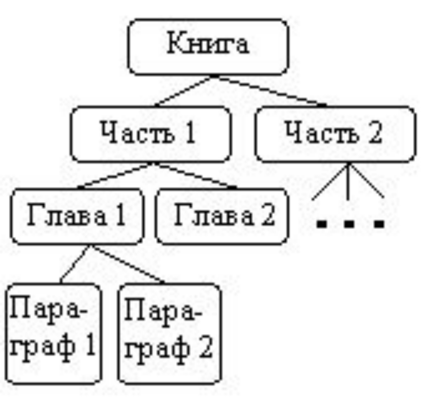
Считают, что корень дерева расположен на первом уровне. Его потомки находятся на втором уровне и т. д. Максимальный уровень какой-либо вершины дерева называется глубиной или высотой дерева. Число потомков вершины называется ее степенью. Максимальное значение этих степеней есть степень дерева. Степень дерева на рисунке, приведенном выше, равна трем.

Деревья – основные понятия

- Деревья в программировании используются значительно чаще, чем графы. Так, на построении деревьев основаны многие алгоритмы сортировки и поиска. Компиляторы в процессе перевода программы с языка высокого уровня на машинный язык представляют фрагменты программы в виде деревьев, которые называются синтаксическими.

Деревья – основные понятия

- Деревья естественно применять всюду, где имеются какие-либо иерархические структуры, т.е. структуры, которые могут вкладываться друг в друга. Примером может служить оглавление книги.



Деревья – основные понятия

- Суть двоичных деревьев, широко распространенных в программировании, следует из названия. Степень дерева равна двум. Вершина (узел) дерева может иметь не более двух потомков, их называют левыми и правыми.

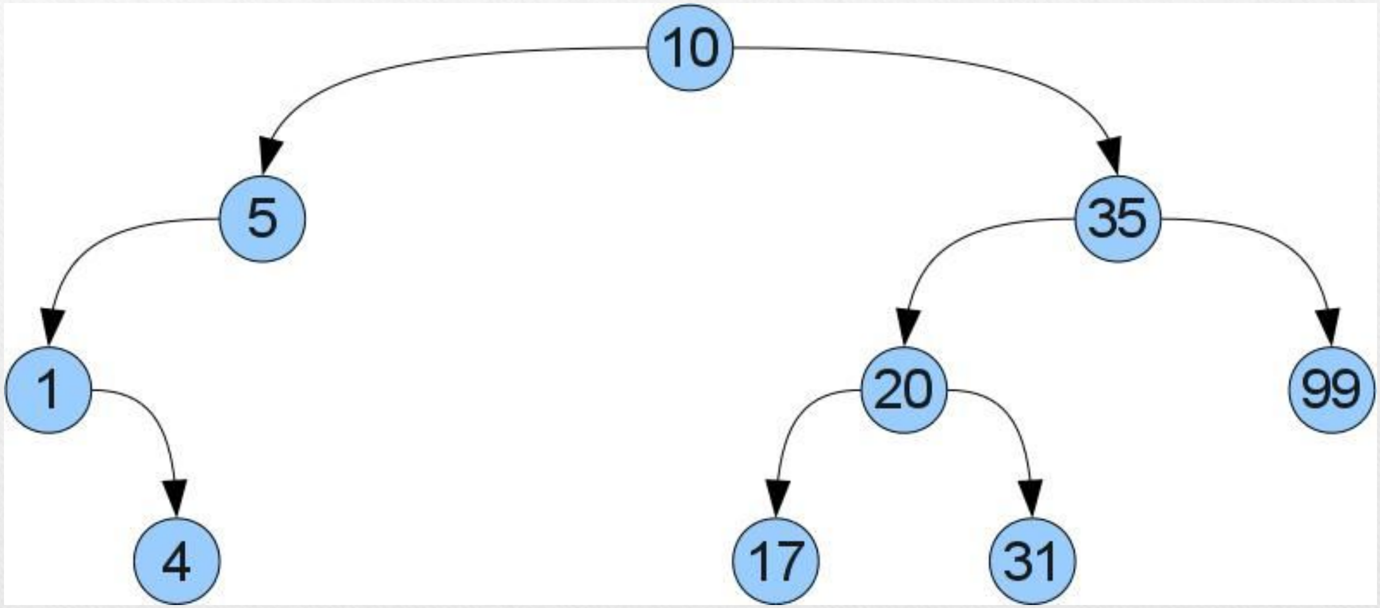
Деревья – основные понятия

Двоичные (бинарные) деревья поиска (подкласс двоичных деревьев) характеризуются тем, что значение информационного поля, связанного с вершиной дерева, больше любого соответствующего значения из левого поддерева и меньше, чем содержимое любого узла его правого поддерева.

Описание двоичного дерева

- struct node
- {
- int info; //Информационное поле
- node *l, *r;//Левая и Правая часть дерева
- };
- node * tree=NULL; //Объявляем переменную, тип которой структура Дерево

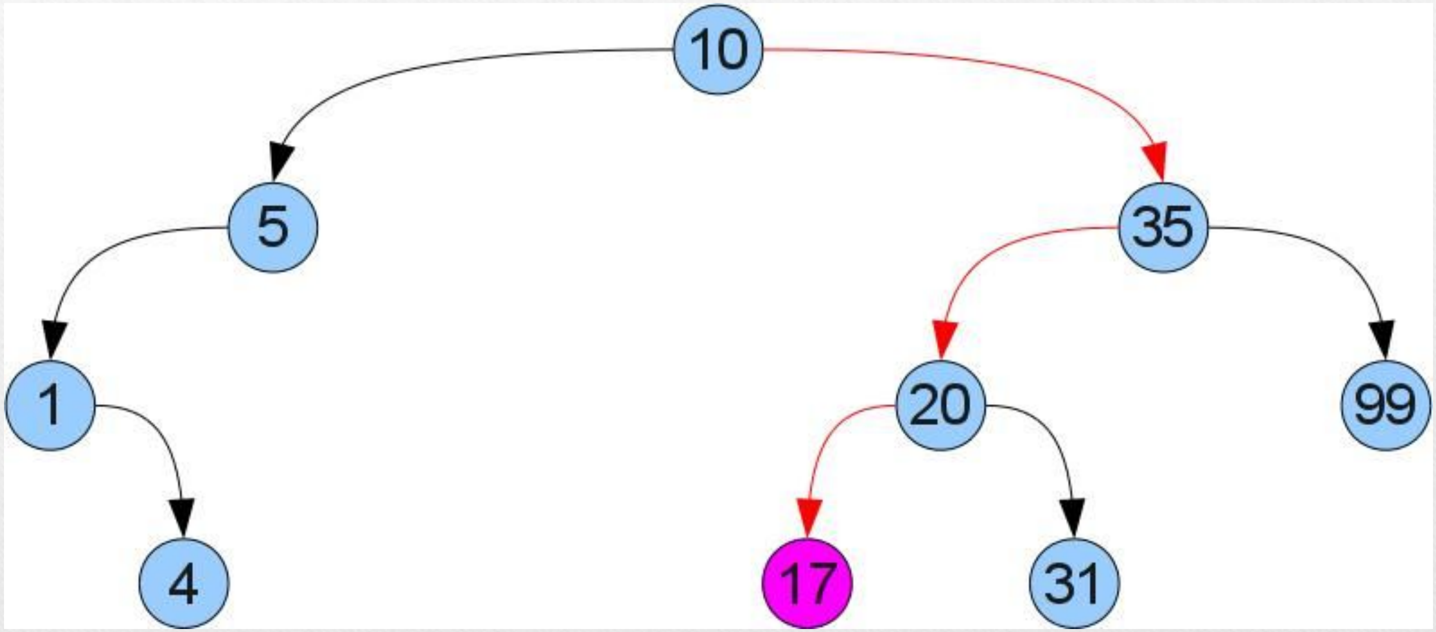
Вид двоичного дерева



Вид двоичного дерева

- **Что же нам дает такое упорядочивание? То, что мы легко можем отыскать требуемый ключ x в дереве! Просто сравним x со значением в корне. Если они равны, то мы нашли требуемое. Если же x меньше (больше), то он может оказаться только в левом (соответственно правом) поддереве.**

Вид двоичного дерева



Основные операции с деревом

- Основные операции:
- вставка элемента в дерево;
- удаление элемента из дерева;
- обход дерева.

Добавление элемента в дерево

- То, что указано в описании - это структура, описывающая звено дерева. По ходу выполнения программы Звенья будут создаваться и дописываться к существующим по указанным правилам. Сама эта структура – это фундамент очень простого бинарного дерева.

Добавление элемента в дерево

- Заполнение простого бинарного дерева разделено на три основные части, а именно – Если дерево пустое, то надо создать первый элемент. Этот элемент будет корнем дерева. После создания первого элемента надо выделить память для возможных ветвей.
- Если дерево что-то содержит, то проверяется условие, согласно которому мы размещаем в дереве элементы.

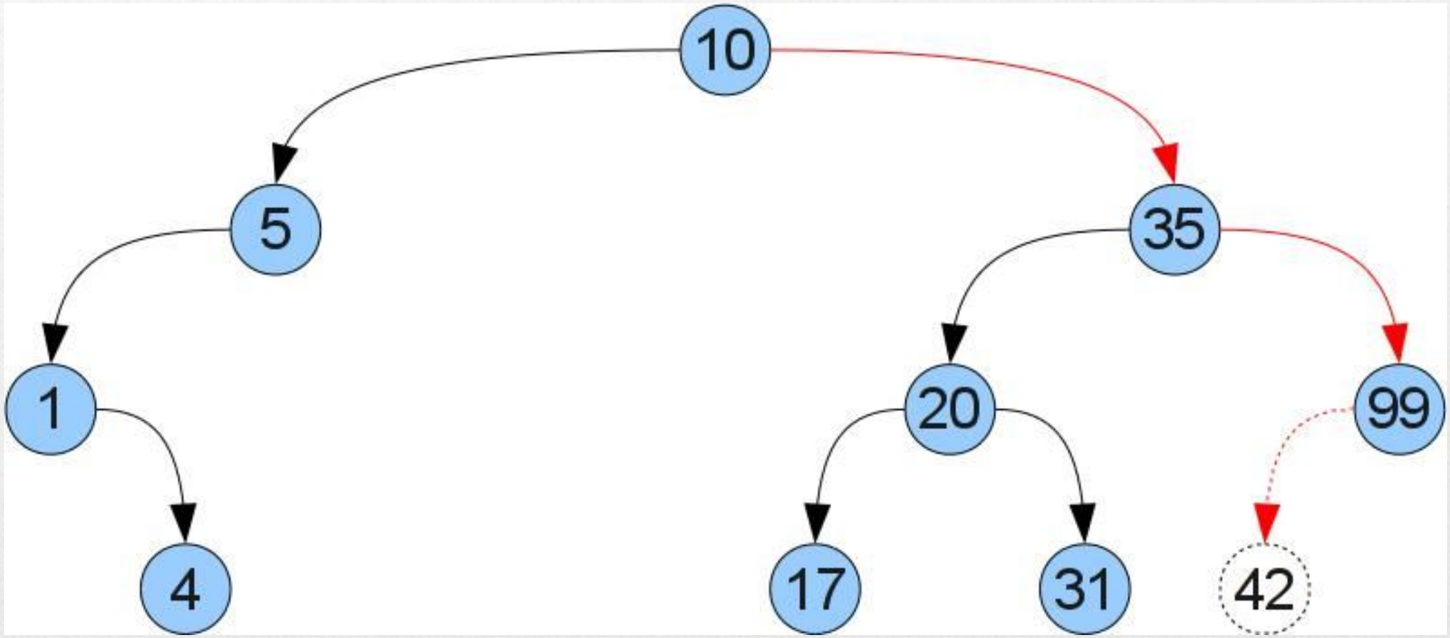
Добавление элемента в дерево

- Бинарное дерево – это упорядоченное дерево, каждая вершина которого имеет не более двух поддеревьев, причем для каждого узла выполняется правило: в левом поддереве содержатся только ключи, имеющие значения, меньшие, чем значение данного узла, а в правом поддереве содержатся только ключи, имеющие значения, большие, чем значение данного узла.

Добавление элемента в дерево

- При всем этом, если элемент, который мы хотим поместить в дерево больше чем корневой, то с помощью рекурсивного вызова функции происходит последовательное перемещение элемента в правую часть.
- Если записываемый элемент меньше чем корневой, то выполняется такая же перестановка, только в левую сторону.

Добавление элемента в дерево



Добавление элемента в дерево

- void add_node(int x, Node *&MyTree) //Функция добавления звена в дерево
- {
- if (NULL==MyTree) //Если дерева нет, то добавляем первый корневой элемент
- {
- MyTree=new Node; //Выделяем память под звено дерева

Добавление элемента в дерево

- MyTree->x=x; //Записываем данные в звено
- MyTree->l=MyTree->r=NULL; //Подзвенья инициализируем пустотой во избежание ошибок
- }
- }

Добавление элемента в дерево

```
if (x<MyTree->x)      //Если нововведенный элемент x меньше чем элемент x из корня дерева, уходим влево
        { if (MyTree->l!=NULL) add_node(x, MyTree->l); //При помощи рекурсии помещаем элемент на свободный участок
else //Если элемент получил свой участок, то
  { MyTree->l=new Node; //Выделяем память левому подзвену.
```

Добавление элемента в дерево

```
MyTree->l->l=MyTree->l->r=NULL; //У левого подзвена будут свои левое и правое подзвенья, инициализируем их пустотой
        MyTree->l->x=x; //Записываем в левое подзвено записываемый элемент
                }
            }
```

Добавление элемента в дерево

- if (x>MyTree->x) //Если нововведенный элемент x больше чем элемент x из корня дерева, уходим вправо

{ if (MyTree->r!=NULL) add_node(x, MyTree->r); //При помощи рекурсии заталкиваем элемент на свободный участок

- else //Если элемент получил свой участок, то
- { MyTree->r=new Node; //Выделяем память правому подзвену.
-

Добавление элемента в дерево

- MyTree->r->l=MyTree->r->r=NULL; //У правого подзвена будут свои левое и правое подзвенья, инициализируем их пустотой
- MyTree->r->x=x; //Записываем в правое подзвено записываемый элемент
- }
- }
- }

Обход дерева

```
void show(Node *&Tree) //Функция обхода
{
   if  (Tree!=NULL) //Пока не встретится пустое звено
  {
       show(Tree->l); //Рекурсивная функция для вывода левого поддерева
    cout<<Tree->x; //Отображаем корень дерева
        show(Tree->r); //Рекурсивная функци для вывода правого поддерева
  }
}
```

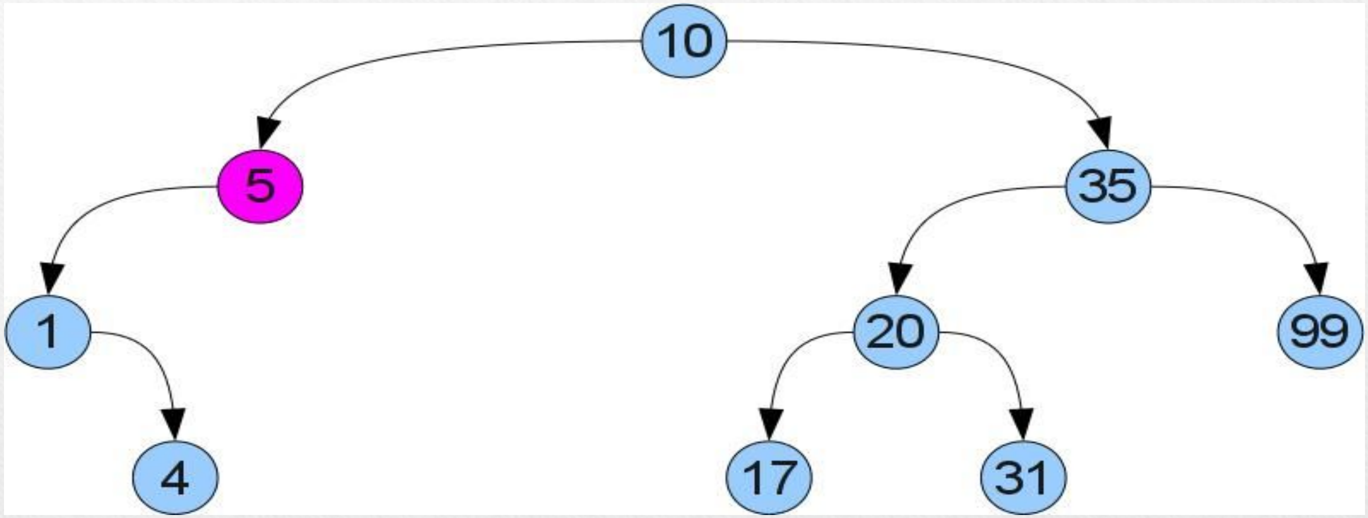
Удаление элемента из дерева

Реализация удаления элемента из дерева чуть сложнее. Если узел имеет одного потомка, то в поле ссылки родителя удаляемого элемента записывается ссылка, не равная NULL, и на этом все закончено.

Удаление элемента из дерева

- В том случае, когда у удаляемого элемента два потомка, для сохранения структуры дерева поиска на место этого элемента необходимо записать или самый правый элемент левого поддерева, или самый левый элемент правого поддерева.

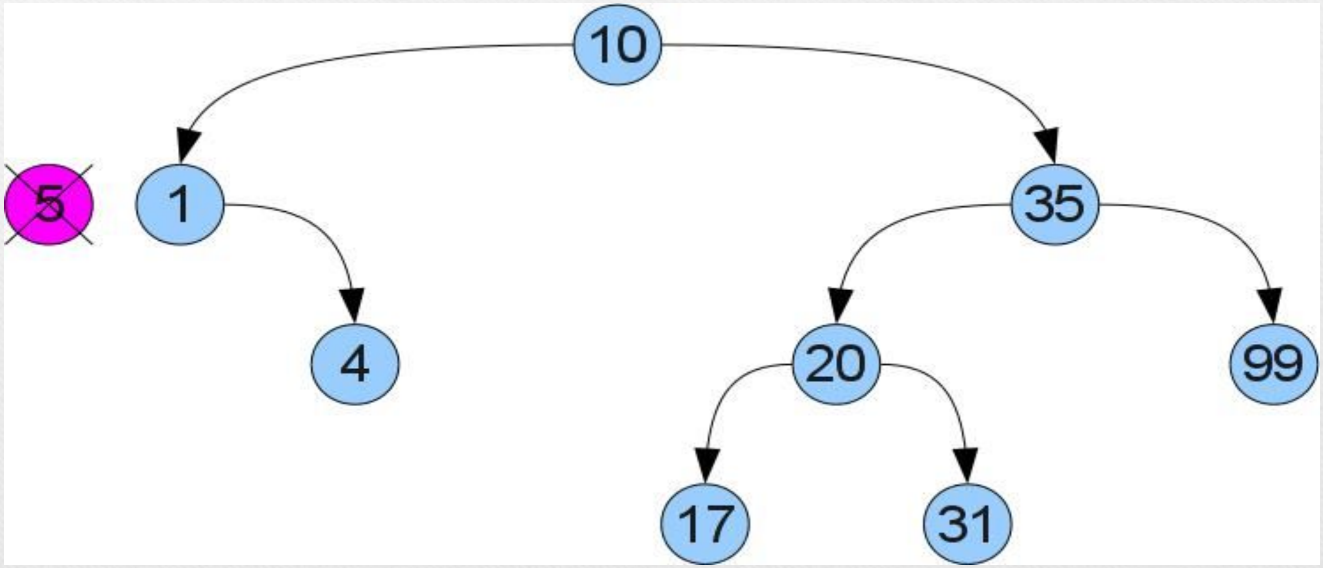
Удаление элемента из дерева



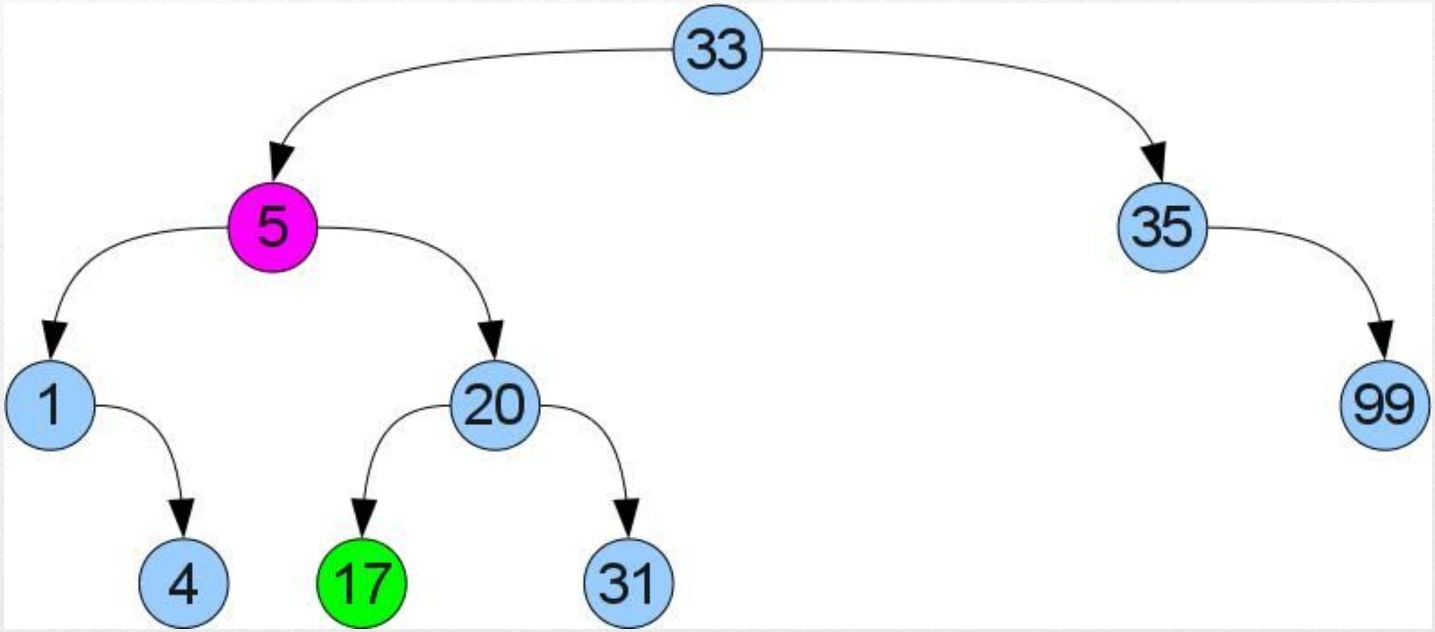
Для начала найдем нашу вершину в дереве. Теперь возникает два случая. Случай 1 (удаляем число 5):

Удаление элемента из дерева

Видно, что у удаляемой вершины нет правого потомка. Тогда мы можем убрать ее и вместо нее вставить левое поддерево, не нарушая упорядоченность:



Удаление элемента из дерева

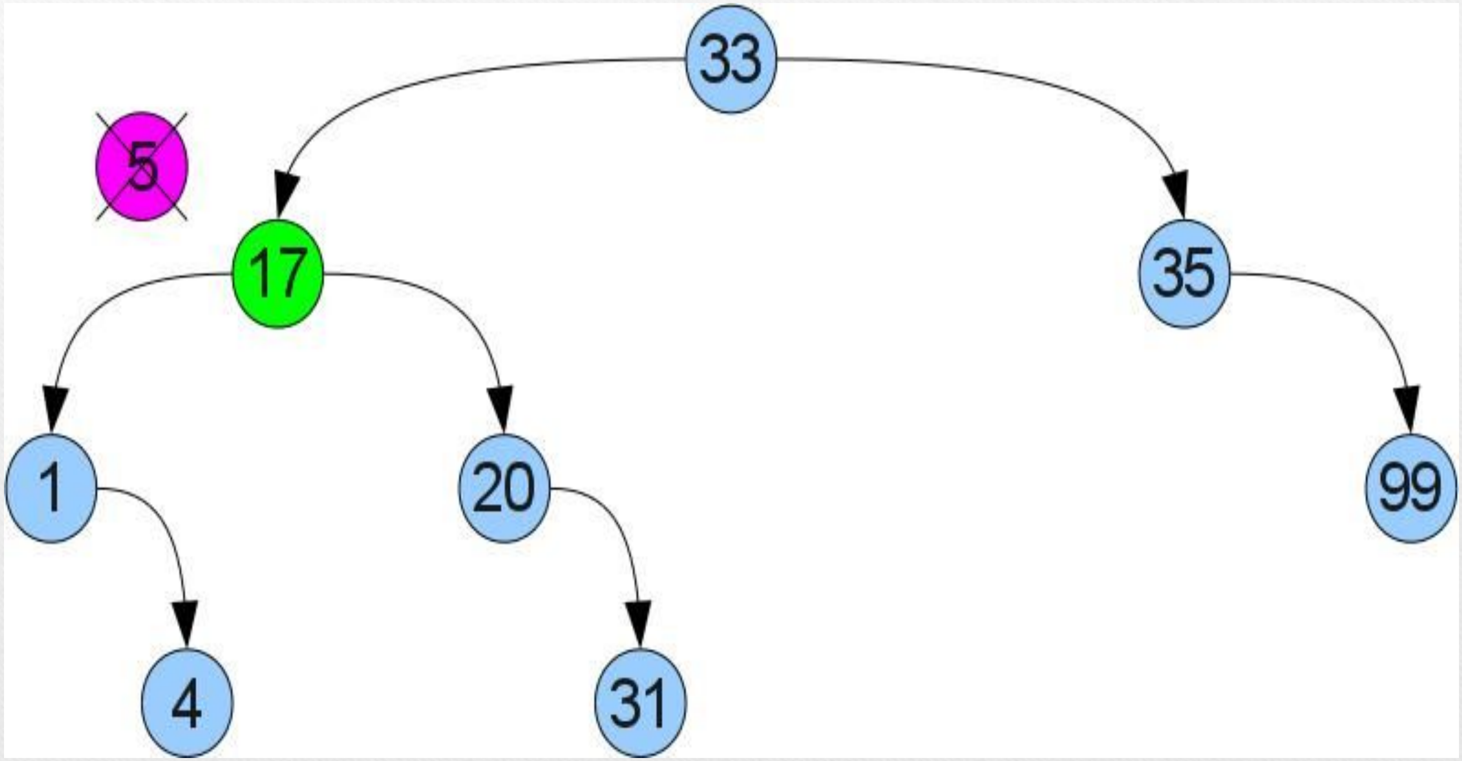


Если же правый потомок есть, на лицо случай 2 (удаляем снова вершину 5, но из немного другого дерева):

Удаление элемента из дерева

Тут так просто не получится — у левого потомка может уже быть правый потомок. Поступим по-другому: найдем в правом поддереве минимум. Ясно, что его можно найти если начать в правом потомке и идти до упора влево. Т.к у найденного минимума нет левого потомка, можно вырезать его по аналогии со случаем 1 и вставить его вместо удаляемой вершины. Из-за того что он был минимальным в правом поддереве, свойство упорядоченности не нарушится:

Удаление элемента из дерева



Удаление элемента из дерева

```
//ищем самую правую вершину левого поддерева
void del_potomka(struct BinaryTree *r, struct
    BinaryTree *q) //в качестве аргумента элемент,
    который и надо удалить
{
    if(r->right!=NULL) del_potomka(r->right, q);
    else
    {
        q->data=r->data;
        q=r;
        r=r->left;
    }
```

Удаление элемента из дерева

- /*удаление вершины из дерева*/
- void delete_element(int x, struct BinaryTree *p)
- {
-     if(p==NULL) printf("Элемента в дереве нет!\n");
-     else if(x<p->data) delete_element(x,p->left); //идём влево
-     else if(x>p->data) delete_element(x,p->right); //идём вправо

Удаление элемента из дерева

```
else //исключаем элемент
  {
      struct BinaryTree *q=p;
      if(q->right==NULL) p=q->left;//если нет правого потомка
      else if(q->left==NULL) p=q->right; //если нет левого потомка
      else del_potomka(q->left, q);//если у нас 2 потомка
       free(q);
  }
```