Языки программирования

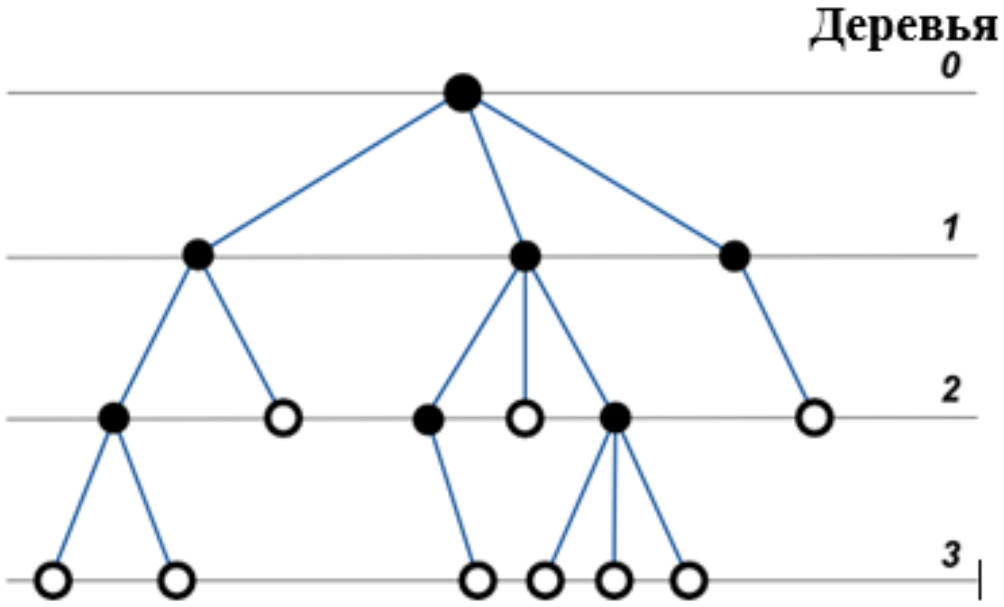
Лекция 12

ПМИ 2 курс

Демяненко Я.М.

ЮФУ 2024

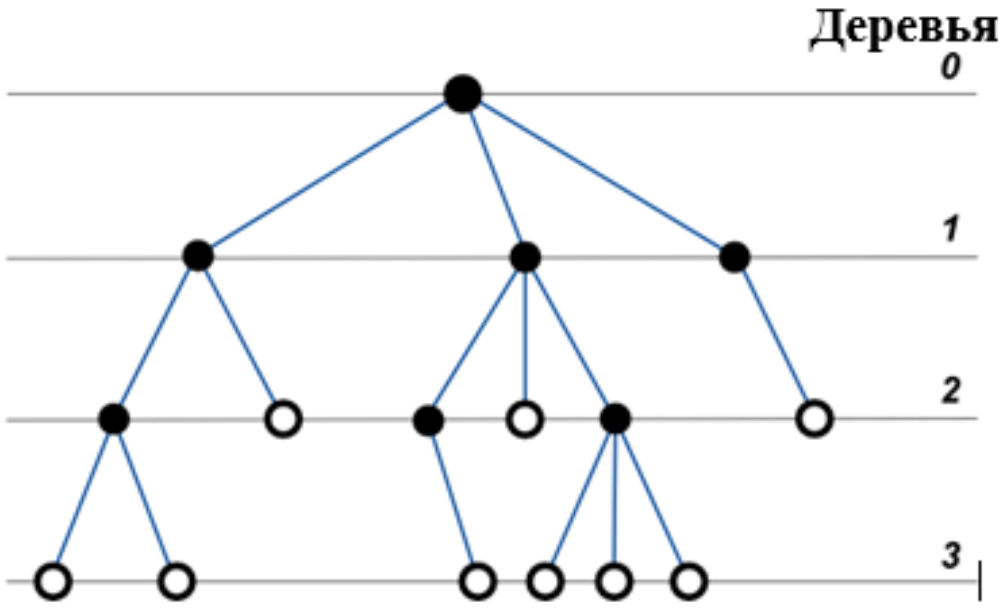
Деревья



Деревом назовем совокупность узлов, называемых **вершинами** дерева, соединенных между собой ребрами, называемыми **ветвями**

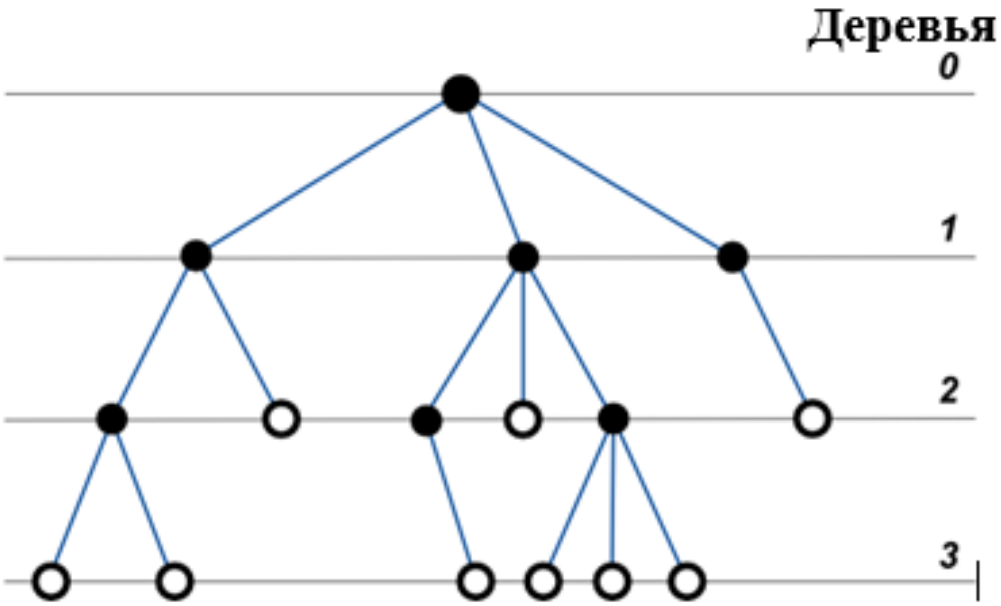
- Единственная вершина, которая не имеет вершин-предков, называется **корнем** дерева
- Вершины, не имеющие потомков, называют **листьями** дерева (*терминальными узлами)* , а совокупность всех листьев образует крону дерева

Уровни дерева



- Уровень узла — длина пути от корня дерева до этого узла
- Корень дерева имеет нулевой уровень
- Каждая вершина нижнего уровня соединяется ровно с одной вершиной предыдущего уровня

Глубина и высота



Номер максимального уровня называется **глубиной** дерева.

Высота узла — это максимальная длина нисходящего пути от этого узла к самому нижнему листу.
Высота корневого узла равна высоте всего дерева.

Глубина дерева равна высоте корневого узла.

Рекурсивное определение дерева

Деревья можно определить двумя способами: рекурсивным и нерекурсивным.
Рекурсивное определение позволяет компактно описывать алгоритмы для работы с деревьями.

Дерево или пусто или состоит из корня и нуля или более непустых поддеревьев.
Каждое поддерево является деревом.

<дерево> ::= <пусто>|
<корень> <список поддеревьев>
<список поддеревьев> ::= <дерево>|
<список поддеревьев> <дерево>

Бинарные деревья

Дерево называется **бинарным** (двоичным), если каждая его вершина имеет не более двух потомков

<бинарное дерево> ::=<пусто>|
<корень>
<левое поддерево>
<правое поддерево>

<левое поддерево> ::=<бинарное дерево>

<правое поддерево>::=<бинарное дерево>

Полным называют БД, у которого каждая вершина, не являющаяся листом, имеет ровно двух потомков, и все листья находятся на последнем уровне

Сбалансированные деревья

Идеально сбалансированным называется дерево, у которого для каждой вершины выполняется требование: число вершин в левом и правом поддеревьях различается не более чем на 1.

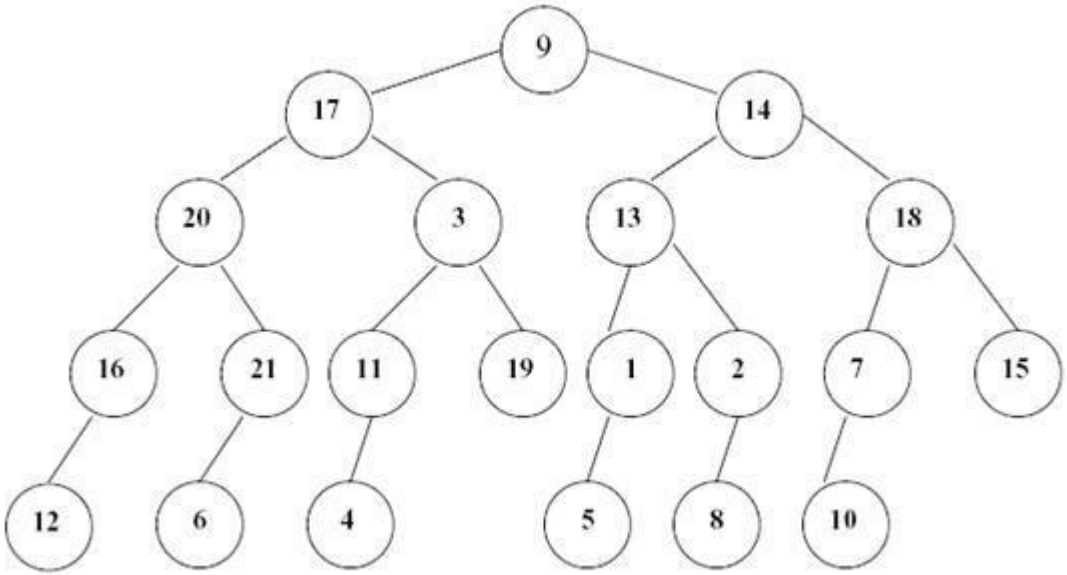
Однако идеальную сбалансированность довольно трудно поддерживать. В некоторых случаях при добавлении или удалении элементов может потребоваться значительная перестройка дерева, не гарантирующая логарифмической сложности.

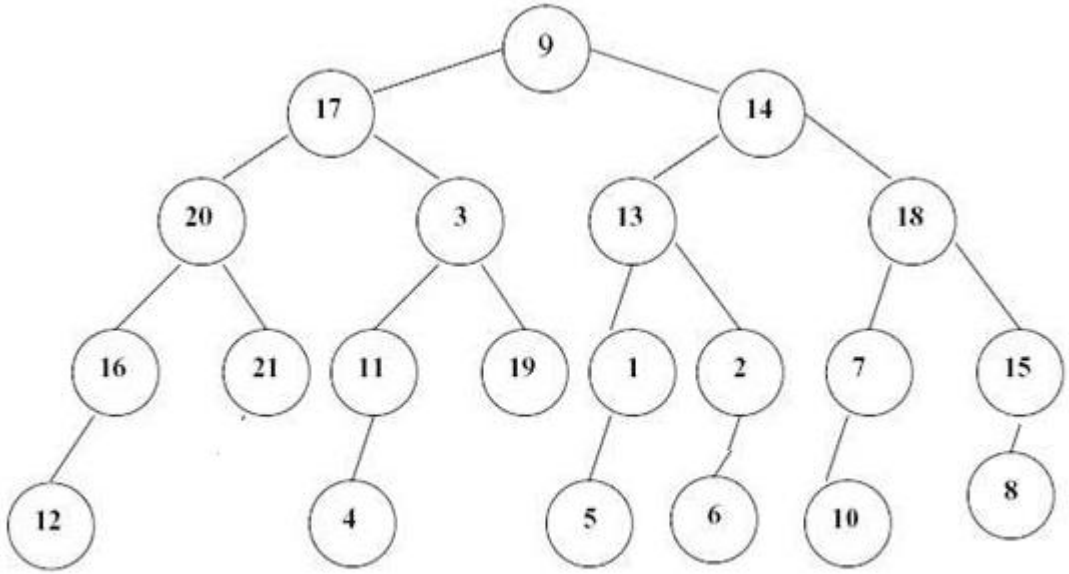
В 1962 году два советских математика: Г.М. Адельсон-Вельский и Е.М. Ландис — ввели менее строгое определение сбалансированности и доказали, что при таком определении можно написать программы добавления и/или удаления, имеющие логарифмическую сложность и сохраняющие дерево сбалансированным.

Дерево считается **АВЛ сбалансированным** (сокращения от фамилий Г.М. Адельсон-Вельский и Е.М. Ландис), если для каждой вершины выполняется требование: высота левого и правого поддеревьев различаются не более, чем на 1.

Не всякое АВЛ-сбалансированное дерево идеально сбалансировано, но всякое идеально сбалансированное дерево АВЛ-сбалансировано.

Сбалансированное дерево?





Реализация бинарных деревьев

Реализуются в виде двусвязных нелинейных списков

```
struct elem{
  int inf;
  elem* lt,*rt;
};

typedef elem* t_ptr;
```

Рекурсивные алгоритмы для бинарных деревьев (порядок обхода)

- Инфиксный (ЛКП)
- Префиксный (КЛП)
- Постфиксный (ЛПК)

При описании алгоритма там, где стоит К выполняется действие алгоритма, там, где стоит Л или П выполняется рекурсивный вызов

- **Печать (КЛП)**

```
void printKLR(t_ptr t) {
  if (t!=nullptr) {
    cout<<t->inf<<" ";
    printKLR(t->lt);
    printKLR(t->rt);
  }
}
```

- **Печать (ЛКП)**

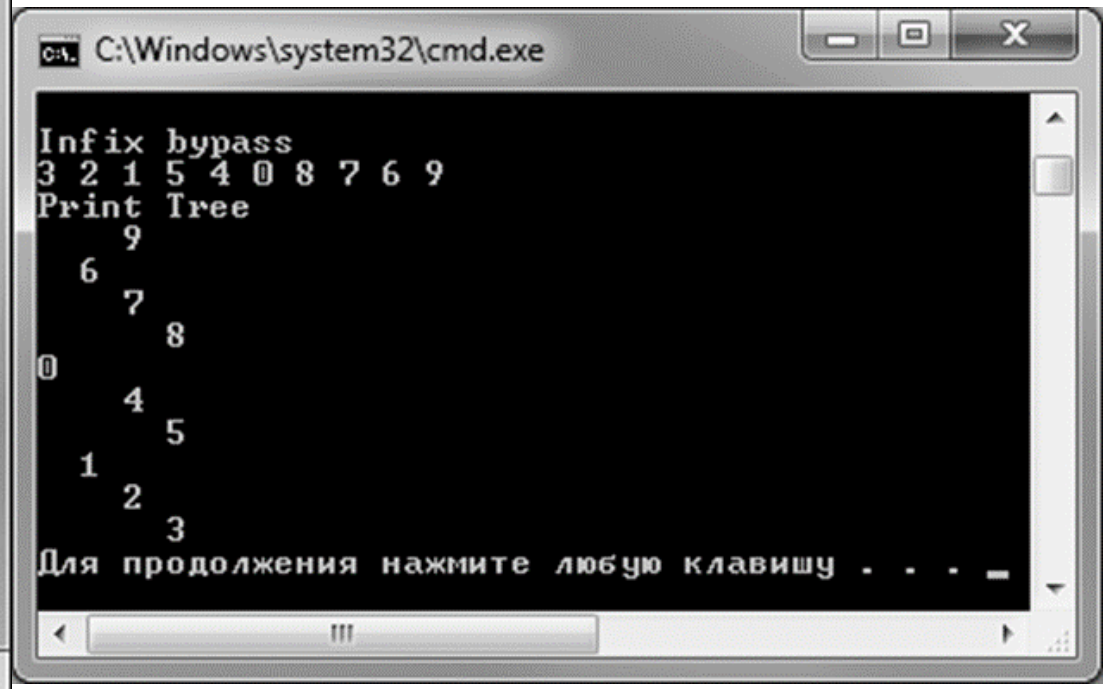
```
void printLKR(t_ptr t) {
  if (t!= nullptr) {
    printLKR(t->lt);
    cout<<t->inf<<" ";
    printLKR(t->rt);
  }
}
```

- **Печать (ЛПК)**

```
void printLRK (t_ptr t) {
  if (t!= nullptr) {
    printLRK (t->lt);
    printLRK (t->rt);
    cout<<t->inf<<" ";
  }
}
```

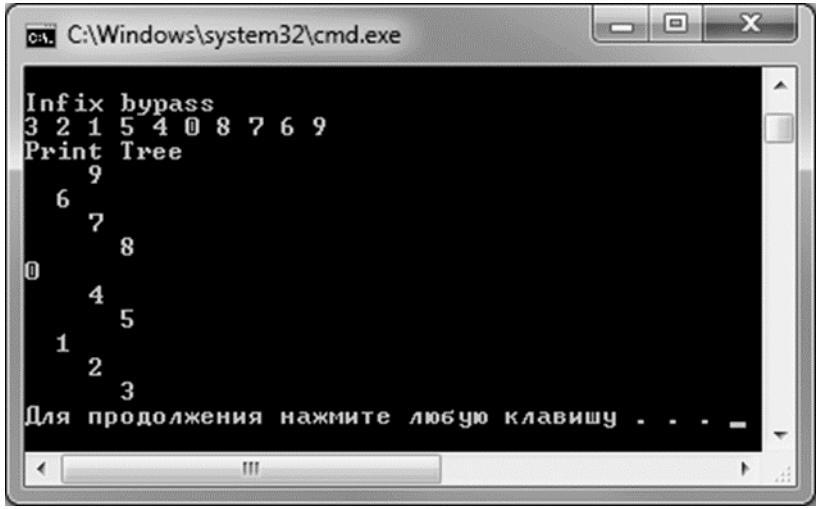
Печать с отступами для анализа структуры

```
void printTREE(t_ptr t, int n) {
  if (t!=nullptr) {
    printTREE(t->rt,n+1);
    for (int i=0; i<n; i++)
      cout<<"  ";
    cout<<t->inf<<endl;
    printTREE(t->lt,n+1);
  }
}
```



Создание идеально сбалансированного дерева

```
t_ptr create(int n) {
  t_ptr p;
  int d;
  if (n>0) {  p = new elem;
    cout<<"Another element?"; cin>> p->inf;
    d= n / 2;
    p->lt = create(d);
    p->rt = create(n-1-d);
    return p;
  }
  else return nullptr;
}
```



Удаление дерева

```
void erase(t_ptr& t) {
  if (t!=nullptr) {
    erase(t->lt);
    erase(t->rt);
    delete(t)
    t = nullptr;
  }
}
```

Пример использования

```
int main(){
  int n;
  t_ptr tree;
  cout<<"Number of elements?"; cin>>n;
  tree=create(n);
  cout<<endl<<"Infix bypass"<<endl;
  printLKR(tree);
  cout<<endl<<"Print Tree"<<endl;
  printTREE(tree,0);
  erase(tree);
  return 0;
}
```

Функция для вычисления суммы четных элементов

```
int sum(t_ptr t){
  if (!t) // проверка на nullptr
    return 0;
  if (t->inf % 2) // проверка на нечетность
    return sum(t->lt) + sum(t->rt);
  return t->inf + sum(t->lt) + sum(t->rt);
}
```

Функция для вычисления максимального элемента

```
int max(t_ptr t){
  int max1= t->inf;
  int max2;
  if (t->lt){
    max2 =max(t->lt);
    if (max2 > max1)
      max1 = max2;
  }
  if (t->rt){
    max2 = max(t->rt);
    if (max2 > max1)
      max1 = max2;
  }
  return max1;
}
```

Функция max не может быть вызвана для пустого дерева. Эта особенность учтена в основной программе следующим образом:

```
cout << "max";
if (tree)
  cout << " = " << max(tree) << endl;
else
  cout << " for empty tree not defined" << endl;
```

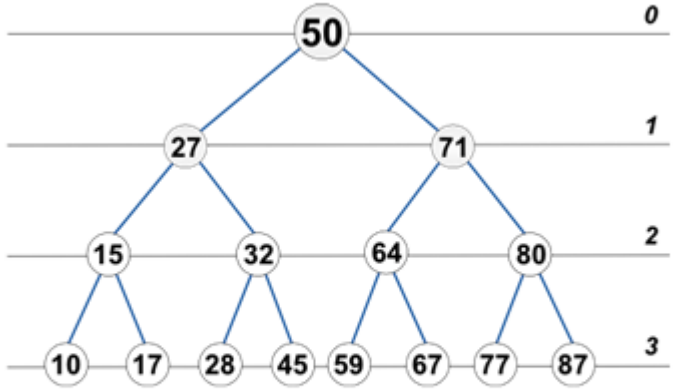
Функция для вычисления количества листьев

```
int leafsCount(t_ptr t){
  if (!t)
    return 0;
  if (!t->lt && !t->rt)
    return 1;
  return leafsCount(t->lt) + leafsCount(t->rt);
}
```

Бинарное дерево поиска

- Бинарное дерево называют бинарным деревом поиска (БДП), если для каждого узла дерева выполнено следующее: все элементы, находящиеся в левом поддереве, меньше элемента в корне, а все элементы, находящиеся в правом поддереве — больше.

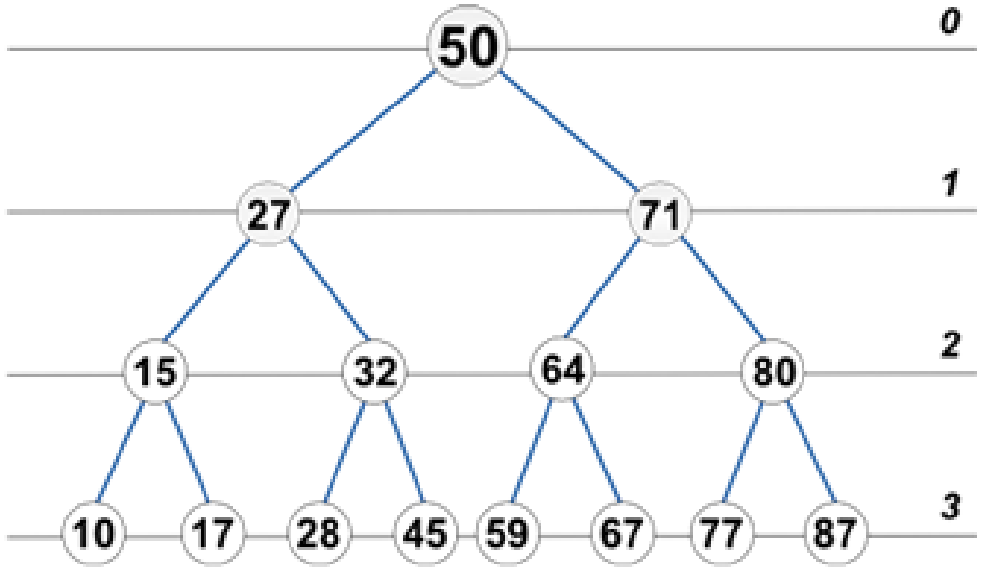
- При такой формулировке, БДП не имеет повторяющихся элементов и называется бинарным деревом поиска **без повторяющихся элементов**. Если в определении заменить все строгие неравенства на нестрогие, то БДП будет называться бинарным деревом поиска **с повторяющимися элементами**.



Инфиксный (ЛКП) обход даст такой порядок элементов

10 15 17 27 28 32 45 50 59 64 67 71 77 80 87

Бинарное дерево поиска



Инфиксный (ЛКП) обход даст такой порядок элементов

10 15 17 27 28 32 45 50 59 64 67 71 77 80 87

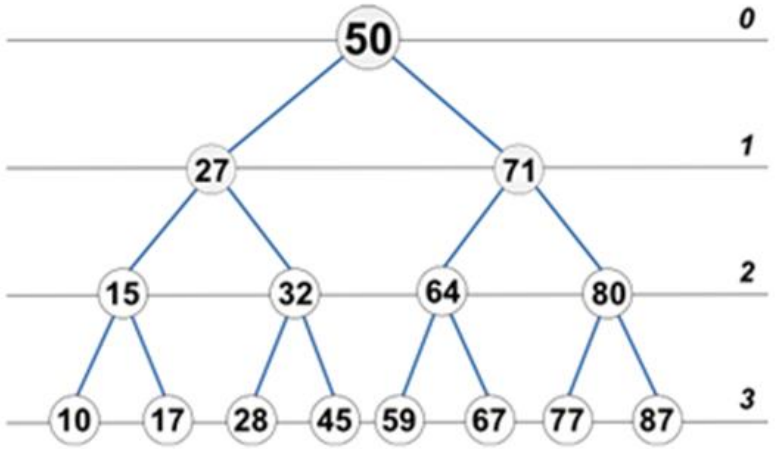
Использование дерева поиска для сортировки

```
void printLKR(t_ptr t) {
  if (t!= nullptr) {
    printLKR(t->lt);
    cout<<t->inf<<" ";
    printLKR(t->rt);
  }
}
```

Основные операции с деревом поиска

Поиск в БДП

```
t_ptr find (t_ptr  t, int a){
 if (!t) return nullptr;
 if (a==t->inf) return t;
 if (a< t->inf) return find(t->lt, a);
 return find(t->rt,a);
}
```



Инфиксный (ЛКП) обход даст такой порядок элементов

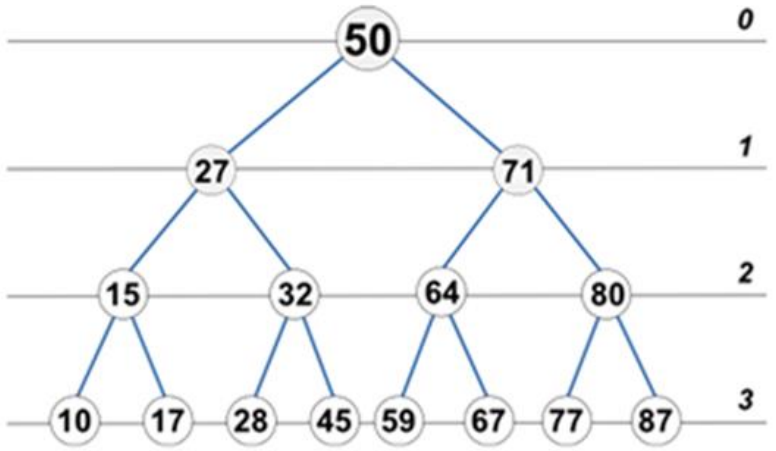
10 15 17 27 28 32 45 50 59 64 67 71 77 80 87

Добавление в БДП

```
void insert (t_ptr & t, int a){
 if (!t) {
          t = new elem;
          t->inf = a;
          t->lt = nullptr;
          t->rt = nullptr;
  }
 else {
    if (a< t->inf) insert(t->lt,a);
       else   insert(t->rt,a);
 }
}
```

Удаление элемента из дерева поиска: вначале поиск

```
void deleteEl(t_ptr &t, int a) {
  if (t != nullptr)
  if (a == t->inf)
    deleteNode(t); // узел найден - удаляем
  else if (a<t->inf)
    deleteEl(t->lt, a); // рекурсия
  else
    deleteEl(t->rt, a); // рекурсия
}
```



Инфиксный (ЛКП) обход даст такой порядок элементов

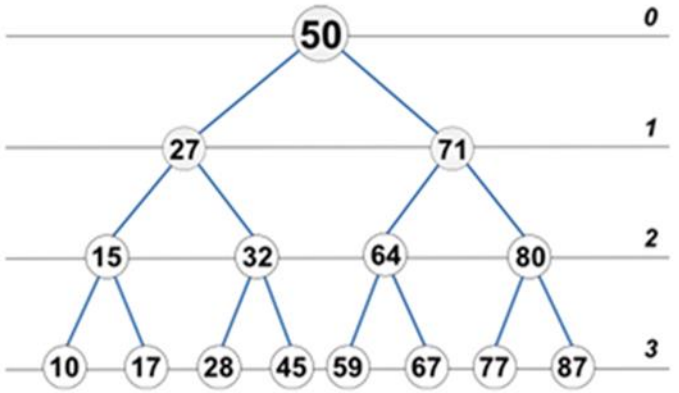
10 15 17 27 28 32 45 50 59 64 67 71 77 80 87

Затем классификация

```
void deleteNode(t_ptr &t){

  // 1. корень является листом
  // 2. у корня нет левого поддерева
  // 3. у корня нет правого поддерева
  // 4. корень имеет два поддерева

  t_ptr delNode;
  int repl;
  if ((t->lt == nullptr) && (t->rt == nullptr))
{// 1
    delete t;
    t = nullptr;
  }
 else if (t->lt == nullptr) {// 2
    delNode = t;
    t = t->rt;
    delete delNode;
  }
else if (t->rt == nullptr) {// 3
    delNode = t;
    t = t->lt;
    delete delNode;
  }
 else {// 4
    delLeftLeaf(t->rt, repl);
    t->inf = repl;
  }
}
```



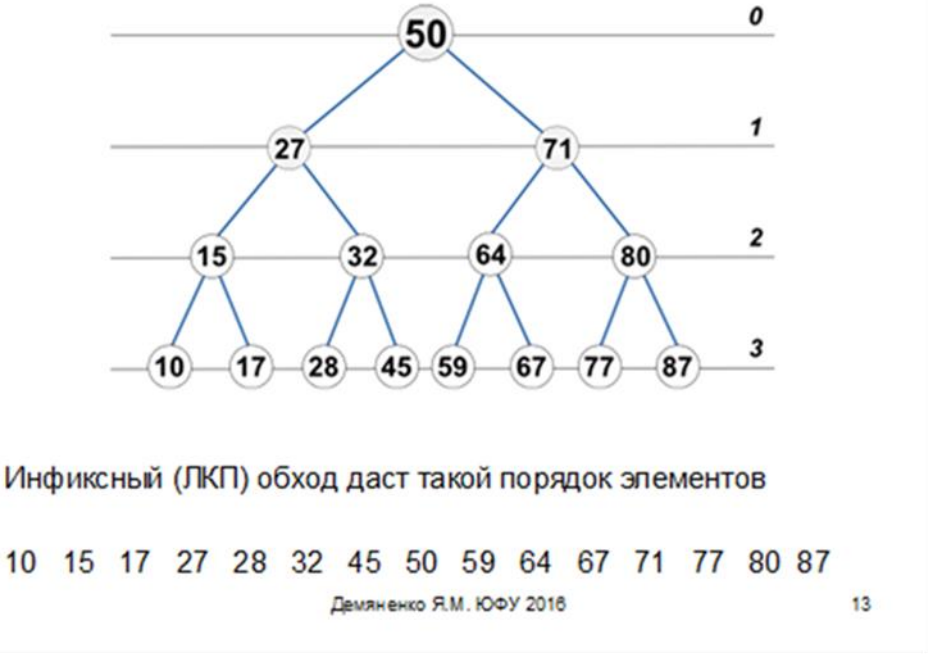
Инфиксный (ЛКП) обход даст такой порядок элементов

10 15 17 27 28 32 45 50 59 64 67 71 77 80 87

Демяненко Я.М. ЮФУ 2016 13

Если узел имеет оба поддерева, то для его удаления вызывается функция delLeftLeaf

```
void delLeftLeaf(t_ptr &t, int &repl) {
  if (!t->lt ) // у самого левого нет левых потомков
  {
    repl = t->inf;
    t_ptr delNode = t;
    t = t->rt;
    delete delNode;
  }
  else
    delLeftLeaf(t->lt, repl);
}
```



Функция ищет **самый левый узел в правом поддереве** удаляемого узла и запоминает его значение в параметре repl. Затем удаляет найденный самый левый узел.