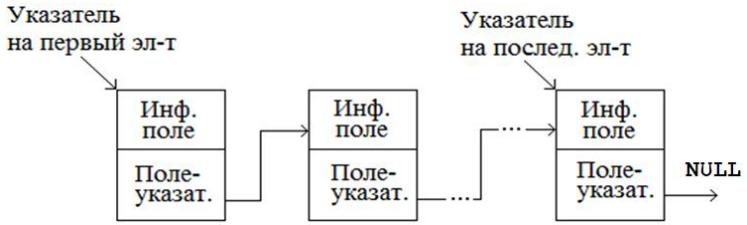
Лекция 5

Динамические структуры данных. Деревья.

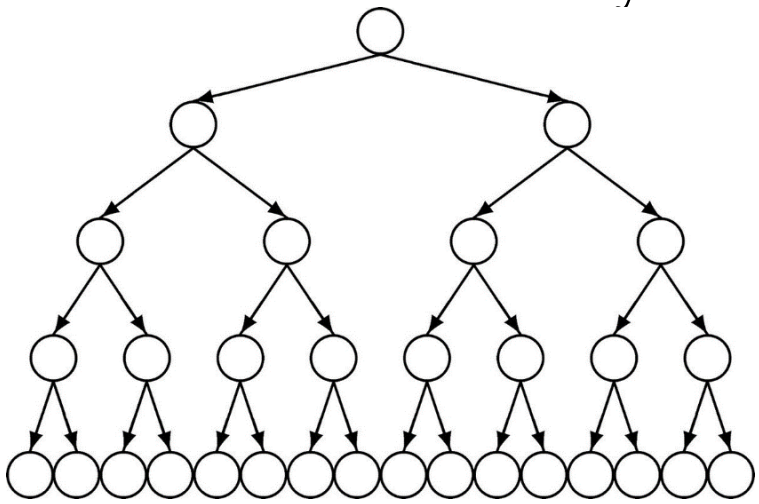
Основные понятия

Списки, стеки, очереди относятся к ***линейным структурам данных***, где связь между элементами выражается в терминах **«предыдущий-следующий».**



Древовидные структуры данных – это ***нелинейные структуры***, где связь между элементами выражается в терминах **«потомок-предок»,** т.е. такие структуры предназначены для отображения связей иерархической подчиненности элементов данных.

Их применение позволяет значительно увеличить скорость ***поиска данных***.



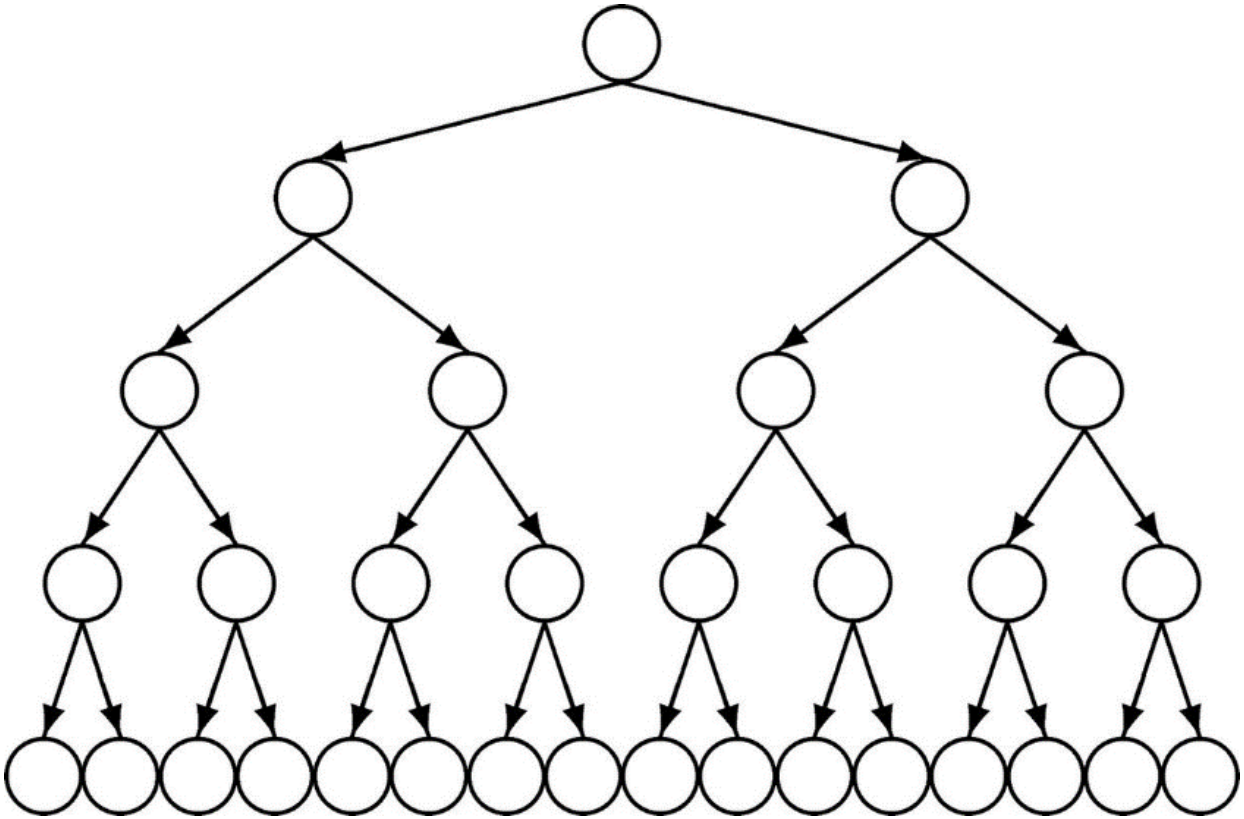
Основные понятия

В программировании часто используются **бинарные деревья.**

Дерево состоит из вершин (узлов) и связей.

Узел бинарного дерева содержит информационное поле и два указателя – на левое и правое поддерево.

Связи между узлами дерева называются его **ветвями**.



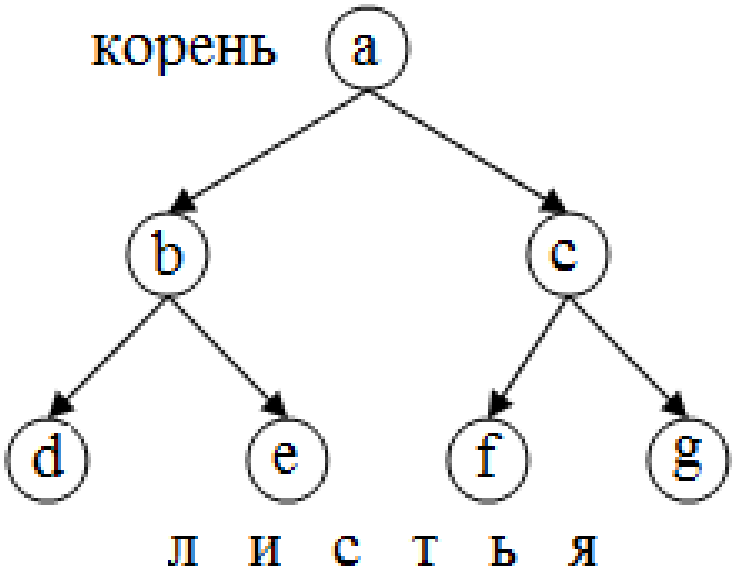
Основные понятия

Вершина (узел) дерева, которая не имеет вышестоящей вершины (предка), называется **корнем.** Считается, что корень дерева расположен на первом уровне.

Максимальный уровень дерева называется его **глубиной.** Количество непосредственных потомков узла называется **степенью узла.** Максимальная степень узла дерева называется **степенью дерева.**

Вершина дерева, которая не имеет дочерних вершин, называется **листом.**

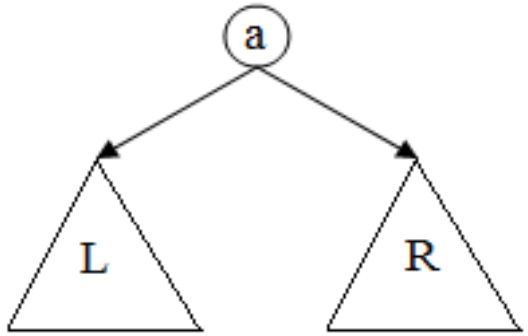
Длина пути до узла – это число ветвей, которое нужно пройти от корня к этому узлу (длина пути к корню равна нулю, узел на уровне `i` имеет длину пути равную `i`).



Основные понятия

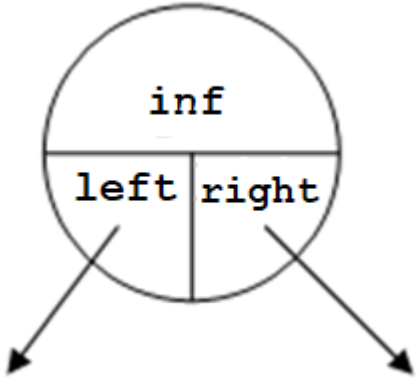
Степень бинарного дерева равна двум, т.е. каждый узел содержит два указателя: на *левое бинарное поддерево* (**ПД**) и *правое бинарное поддерево*.

Т. о., бинарное (под)дерево состоит из корня, левого бинарного поддерева и правого бинарного поддерева *(рекурсивное определение).*



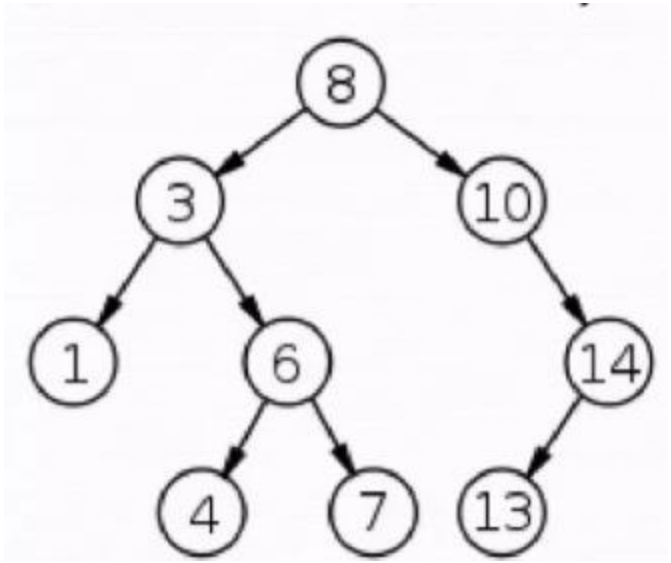
Пример описания узла бинарного дерева:

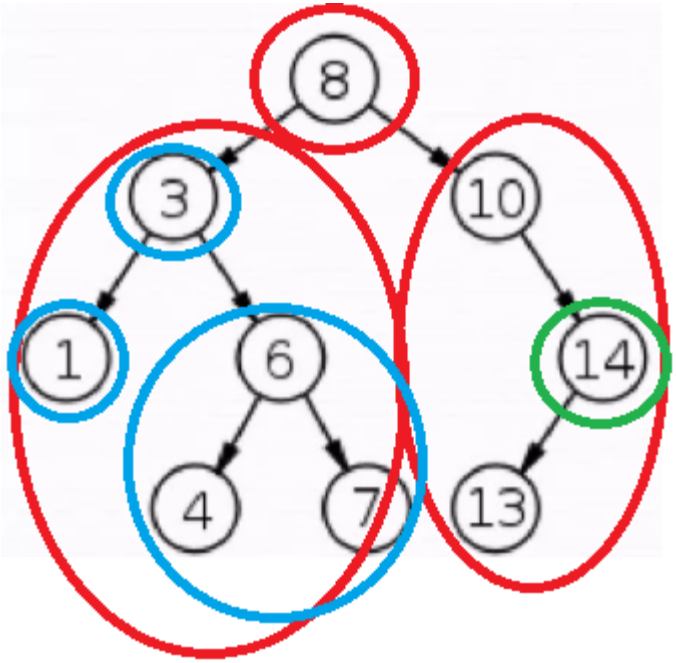
```
struct tree { /* узел дерева */
  <тип инф.поля> inf; // инф. поле
  struct tree *left; // левое поддерево
  struct tree *right; //правое поддерево
};
```



Основные понятия

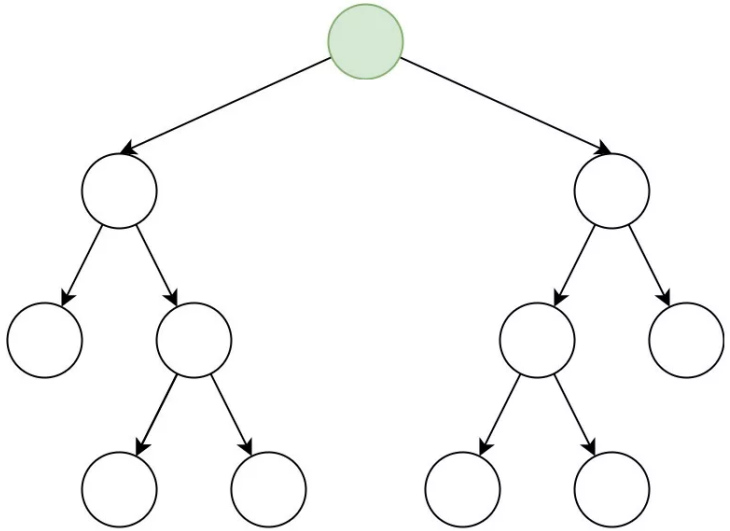
```
struct tree { /* узел дерева */
  <тип инф.поля> inf; // инф. поле
  struct tree *left; // левое поддерево
  struct tree *right; //правое поддерево
};
```



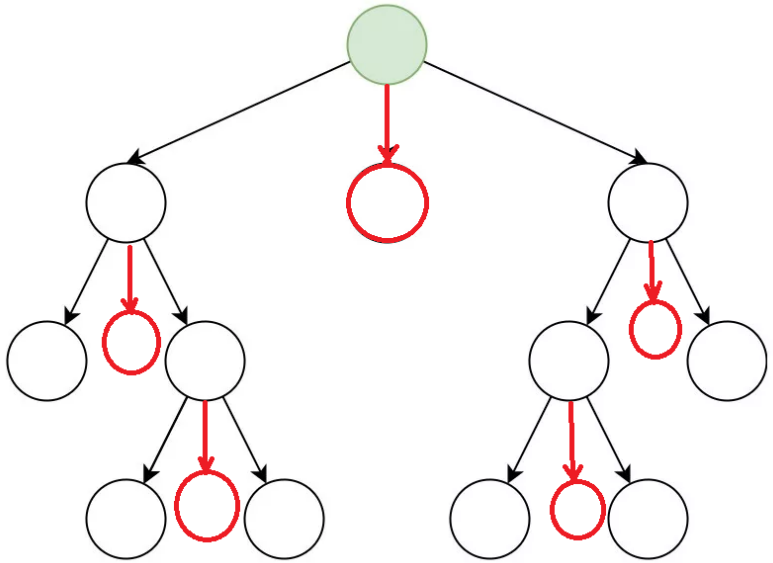


Пример бинарного и небинарного дерева

```
struct tree {
     <тип инф.поля> inf;
     struct tree *left;
     struct tree *right;
};
```

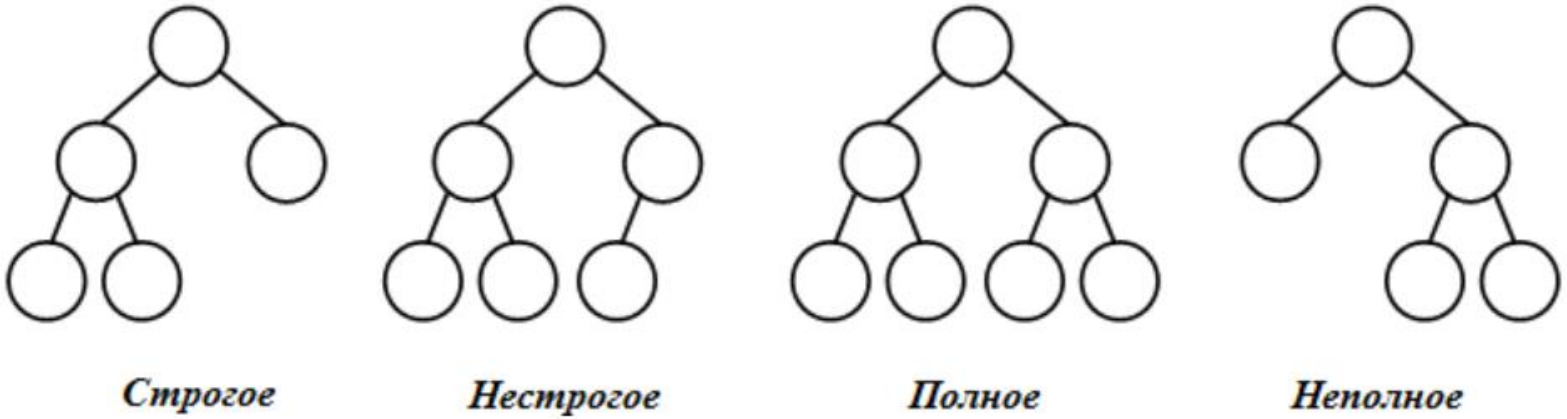


```
struct my_tree {
   <тип инф.поля> inf;
   struct my_tree *left;
   struct my_tree *right;
   struct my_tree *middle;
};
```



Виды деревьев

По степени вершин бинарные деревья делятся на: ***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).

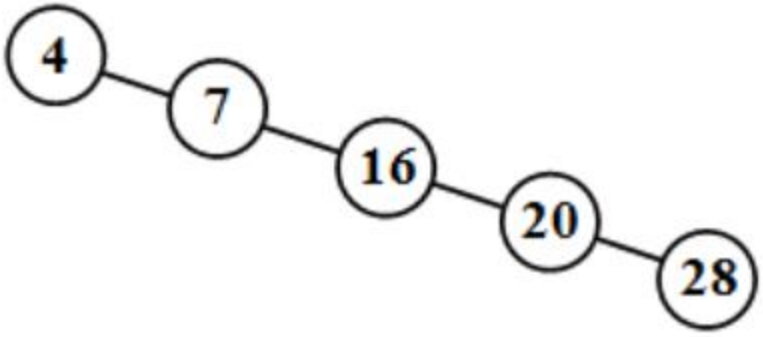


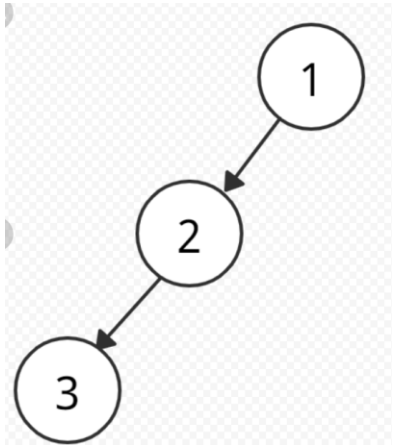
В общем случае у бинарного дерева на k-м уровне может быть до 2^{k-1} вершин. Бинарное дерево называется ***полным***, если оно содержит только полностью заполненные уровни. В противном случае оно является ***неполным***.

Виды деревьев

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.

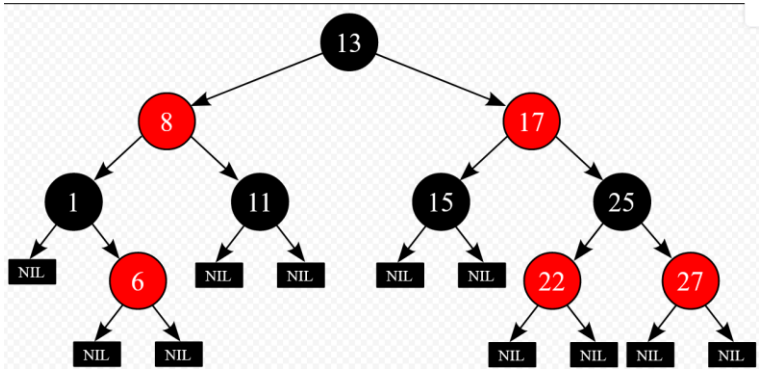
Бинарное дерево может выродиться в список (***вырожденное дерево***):



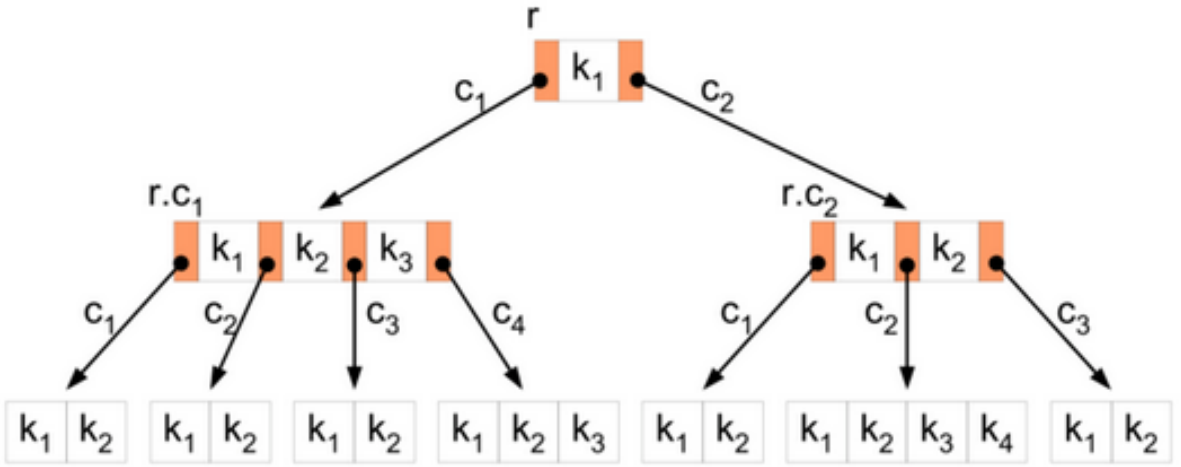


Виды деревьев

Красно-черное дерево один из видов самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты (глубины) дерева от числа узлов и позволяющее быстро выполнять основные операции дерева поиска: добавление, удаление и поиск узла.

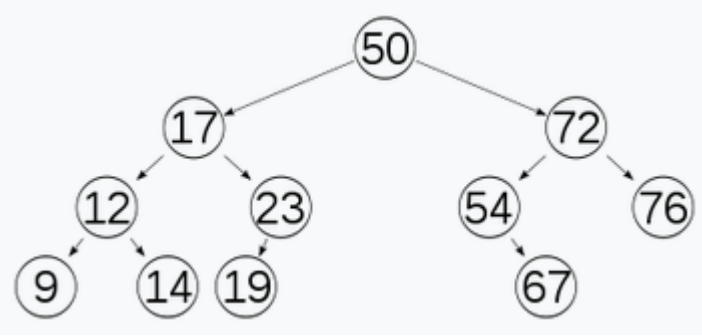


B-дерево – структура данных, дерево поиска, сбалансированное, сильно ветвистое дерево. Часто используется для хранения данных во внешней памяти.

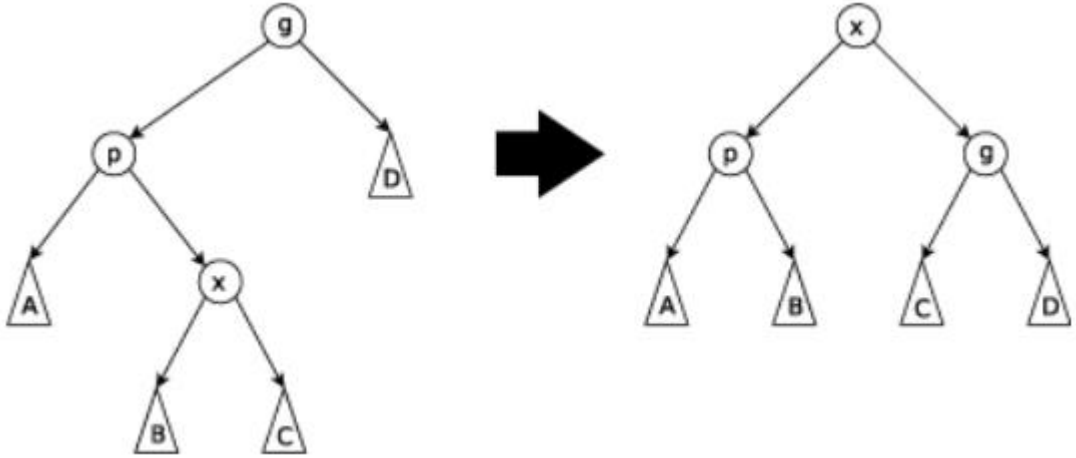


Виды деревьев

АВЛ-дерево – сбалансированное по высоте двоичное дерево поиска: для каждой его вершины высота её двух поддеревьев различается не более чем на 1.

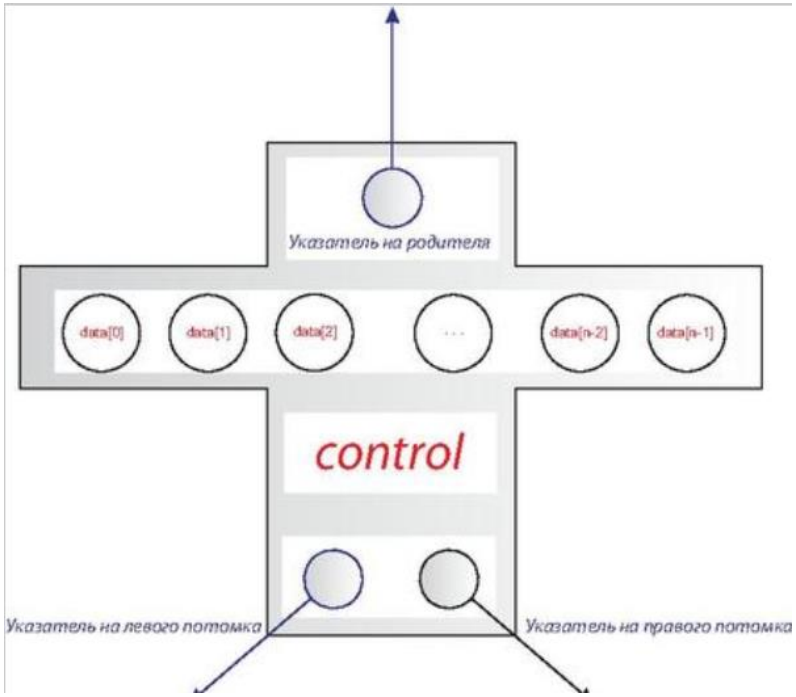


Расширяющееся (`splay tree`) или **косое дерево** является двоичным деревом поиска, в котором поддерживается свойство сбалансированности. Это дерево принадлежит классу «саморегулирующихся деревьев» и реализуется без использования каких-либо дополнительных полей в узлах дерева за счет «расширяющихся операций» (`splay operation`), частью которых являются вращения, которые выполняются при каждом обращении к дереву.



Виды деревьев

Т-дерево (**`T-tree`**) – сбалансированное бинарное дерево, оптимизированное для случаев, когда востребованные (горячие) данные полностью хранятся в оперативной памяти.



Танцующее дерево (**`Dancing tree`**) – древовидная структура хранения данных, которая похожа на B-дерево. По сравнению со сбалансирован-ными бинарными деревьями, которые пытаются сохранить свои узлы сбалансированными постоянно, танцующие деревья сохраняют баланс между узлами только при записи данных на диск: либо из-за ограничений памяти, или потому, что транзакция завершена

Алгоритмы работы с бинарными деревьями

Операции над бинарными деревьями:

- создание дерева;
- добавление узла в дерево;
- удаление узла из дерева;
- обход дерева и поиск узла в дереве.

Наиболее распространенным условием организации бинарных деревьев является ***упорядоченность.*** Каждый элемент в упорядоченном дереве имеет на своей левой ветви элементы с меньшими, чем у него, значениями, а на правой ветви элементы с большими значениями.

Для построения упорядоченного бинарного дерева из последовательности данных необходимо:

- выбрать ключевое поле, по которому будет строиться дерево;
- первый элемент последовательности данных сделать корнем дерева;
- из последовательности выбрать очередной элемент и добавить его в дерево по алгоритму, приведенному ниже.

Алгоритм добавления узла в бинарное дерево

Рассмотрим ***общий алгоритм добавления узла в упорядоченное бинарное дерево***. Доступ к дереву выполняется через указатель на его корень.

Алгоритм **добавления** формулируется рекурсивно с помощью правил:

1) если указатель на корень дерева (поддерева) равен **NULL**, то создать (под) дерево с одним узлом;

2) если добавляемый узел меньше корня, то **добавить** его в левое поддерево;

3) иначе **добавить** в правое поддерево.

В этом случае новые элементы всегда присоединяются к листьям дерева.

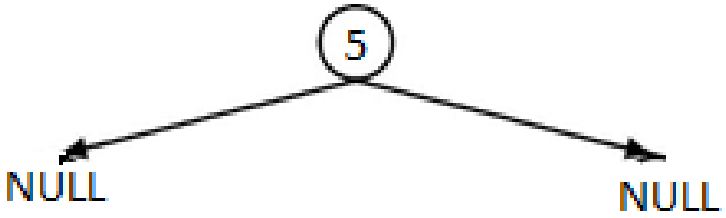
Рассмотрим на примере:

Алгоритм добавления узла в бинарное дерево

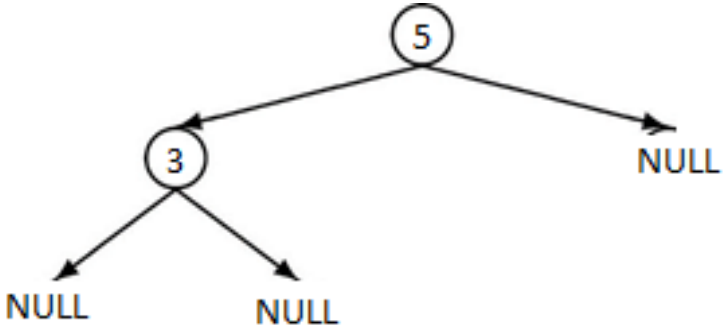
Например: пусть есть данные о детях Артека из отряда №10

5	Петя	Севастополь	12
3	Саша	Москва	11
8	Маша	Омск	12
2	Катя	Тверь	13
4	Наташа	Новосибирск	11
7	Паша	Новгород	12

Ключевым полем выберем порядковый номер.
Первая строка таблицы – корень дерева:

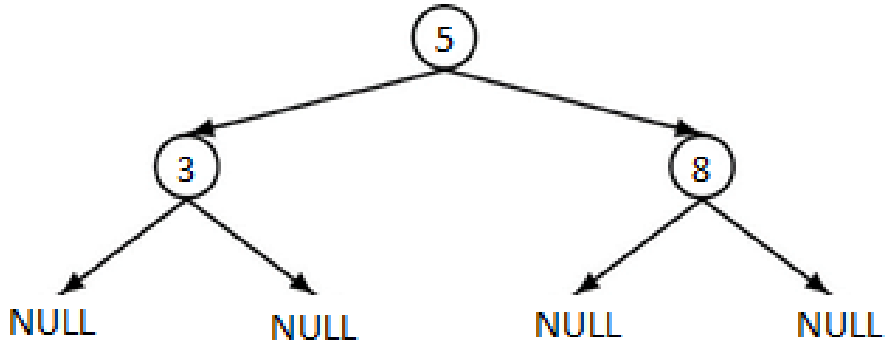


Вторая строка – порядковый номер равен 3, сравниваем с корнем (3 < 5) и уходим в левое поддерево:



Алгоритм добавления узла в бинарное дерево

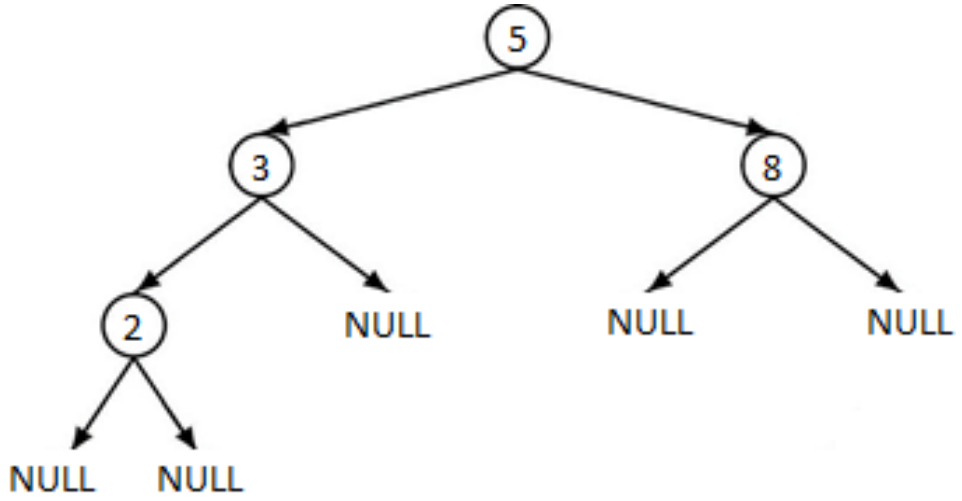
Следующий элемент – номер 8, сравниваем с корнем (8 > 5) и уходим в правое поддерево:



5	Петя	Севастополь	12
3	Саша	Москва	11
8	Маша	Омск	12
2	Катя	Тверь	13
4	Наташа	Новосибирск	11
7	Паша	Новгород	12

Алгоритм добавления узла в бинарное дерево

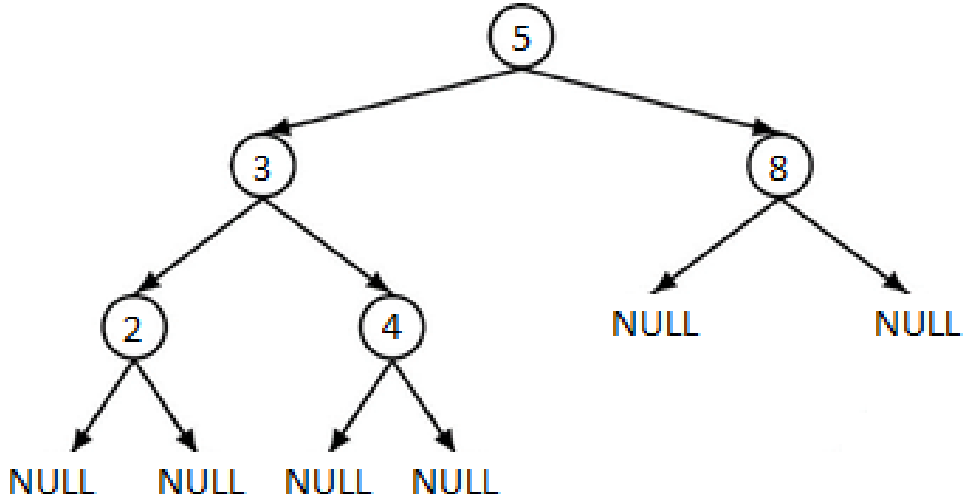
Следующий элемент – номер 2, сравниваем с корнем (2 < 5) и уходим в левое поддерево, сравниваем с корнем левого поддерева (2 < 3) и уходим в левое поддерево:



5	Петя	Севастополь	12
3	Саша	Москва	11
8	Маша	Омск	12
2	Катя	Тверь	13
4	Наташа	Новосибирск	11
7	Паша	Новгород	12

Алгоритм добавления узла в бинарное дерево

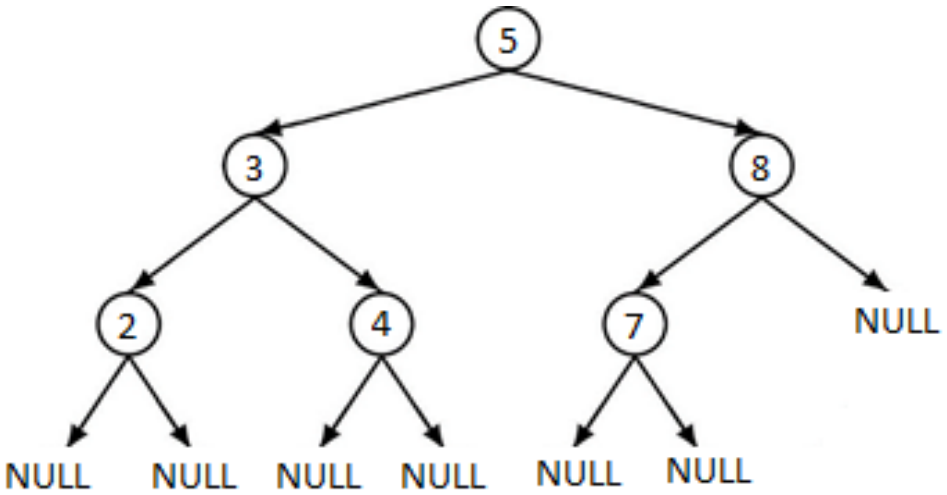
Следующий элемент – номер 4, сравниваем с корнем (4 < 5) и уходим в левое поддерево, сравниваем (4 > 3) и уходим в правое поддерево:



5	Петя	Севастополь	12
3	Саша	Москва	11
8	Маша	Омск	12
2	Катя	Тверь	13
4	Наташа	Новосибирск	11
7	Паша	Новгород	12

Следующий элемент – номер 7, сравниваем с корнем (7 > 5), уходим в правое поддерево, сравниваем (7 < 8) и уходим в левое поддерево:

Алгоритм добавления узла в бинарное дерево



5	Петя	Севастополь	12
3	Саша	Москва	11
8	Маша	Омск	12
2	Катя	Тверь	13
4	Наташа	Новосибирск	11
7	Паша	Новгород	12

5	Петя	Севастополь	12		
3	Саша	Москва	11		
8	Маша	Омск	12		NULL
2	Катя	Тверь	13	NULL	NULL
4	Наташа	Новосибирск	11	NULL	NULL
7	Паша	Новгород	12	NULL	NULL

Функция добавления узла в бинарное дерево

Рассмотрим реализацию функции, выполняющей добавление узла в дерево. Процесс добавления узла в дерево является рекурсивным: если добавленный узел меньше, чем корневой элемент, то добавляем элемент в левое поддерево, иначе добавляем его в правое поддерево. Добавление узла выполняется в этом случае на уровне листьев дерева.

Пусть информационное поле – это сведения о грузополучателе:

```
const int d_f=20,                    //длина строки с Ф.И.О.
          d_a=80;                    //длина строки с адресом
//-------------------структуры-----------------------------
struct gruz_p {          // описание структуры о грузополучателе
    char fio[d_f],
         address[d_a];
};
struct tree {                  // описание структуры дерево
    gruz_p data;
    tree* left;
    tree* right;
};
```

Функция добавления узла в бинарное дерево

```
tree* addtree(tree *top, const gruz_p& newtree){
    if (!top) {                    //если находимся на уровне листа,
        top=(tree*)malloc(sizeof(tree));//то выделить память
        if (!top) {
                printf("Не хватает памяти");
                return NULL;       //выход если память не выделена
          }
          top->data=newtree;           //запись данных в узел
          top->left=NULL;              //обнуление указателей
          top->right=NULL;
    } else          // иначе сравниваем значение в узле с добавляемым и
        if (strcmp(top->data.fio, newtree.fio)>0)
                              // добавляем в левое поддерево
           top->left=addtree(top->left,newtree);
                              // или правое поддерево
          else
          top->right=addtree(top->right,newtree);
    return top;               //возвращаем указатель на корень дерева
}
```

Виды обхода бинарного дерева

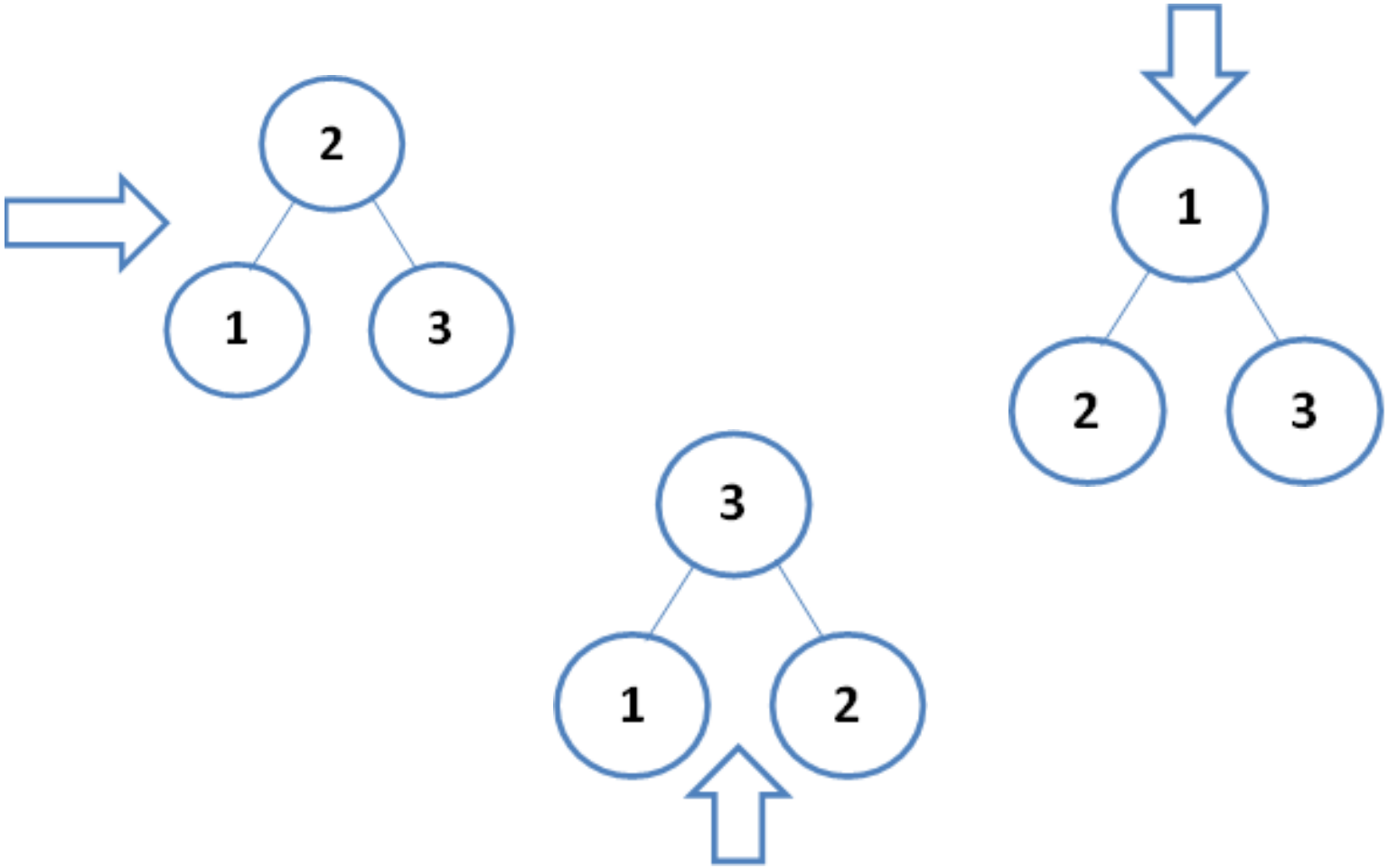
Для **обхода дерева** могут использоваться различные *стратегии*:
- обход ***сверху вниз*** (прямой);
- обход ***слева направо*** (симметричный);
- обход ***снизу вверх*** (обратный).

При выполнении обхода дерева применяют действия:
1) обработка (отображение значения) корня дерева;
2) обход левого поддерева;
3) обход правого поддерева.

Если перечисленные процедуры выполняются в порядке **1-2-3**, то выполняется **обход сверху вниз**; если в порядке **2-1-3**, то выполняется **обход слева направо**; если в порядке **2-3-1**, то выполняется **обход снизу вверх**.

Виды обхода бинарного дерева

Виды обхода бинарного дерева:



Виды обхода бинарного дерева

Пример **функции обхода** двоичного дерева слева направо.

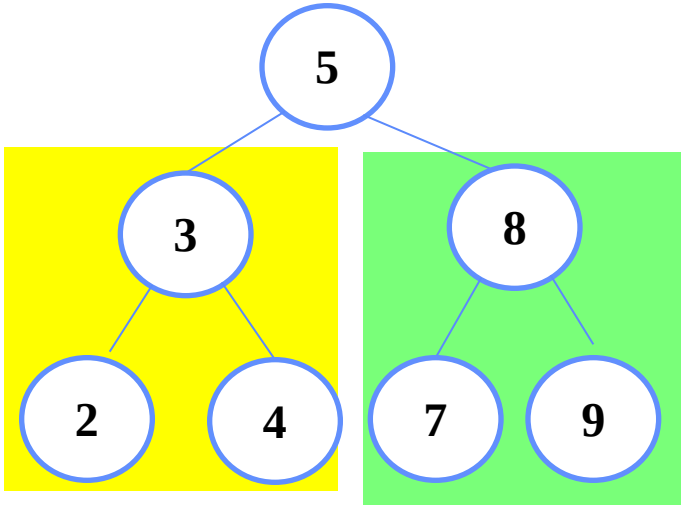
top – указатель на корень дерева.

```
void prosmotr(tree *top) {      //просмотр сверху вниз
  if (top) {
     printf("FIO = %s\t", top->data.fio);//вывод значения корня
     printf("address = %s\n", top->data.address);
     prosmotr(top->left);       //обход левого поддерева
     prosmotr(top->right);      //обход правого поддерева
    }
 }

void prosmotr(tree *top) {      //просмотр слева направо
 if (top) {
     prosmotr(top->left);
     printf("FIO = %s\t", top->data.fio);
     printf("address = %s\n", top->data.address);
     prosmotr(top->right);
    }
 }
```

Виды обхода бинарного дерева

5 3 8 4 7 9 2



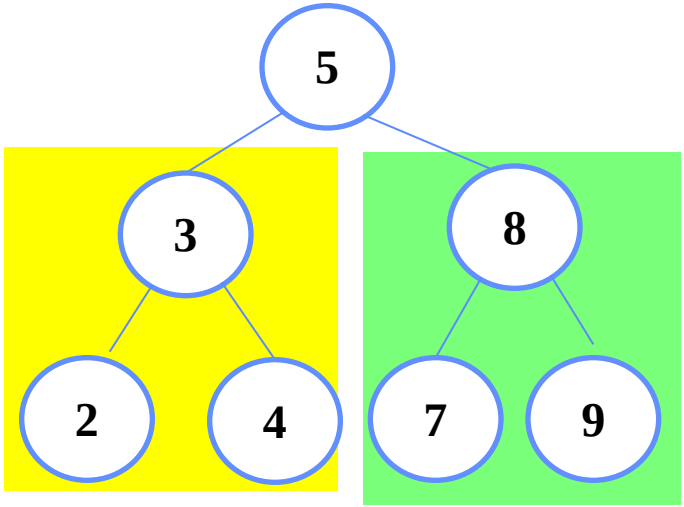
При обходе сверху вниз получим:

При обходе слева направо получим:

При обходе снизу вверх получим:

Виды обхода бинарного дерева

5 3 8 4 7 9 2



При обходе сверху вниз получим: **5 3 2 4 8 7 9**

При обходе слева направо получим: **2 3 4 5 7 8 9**

При обходе снизу вверх получим: **2 4 3 7 9 8 5**

Организация бинарного дерева

Определим функцию организации дерева:

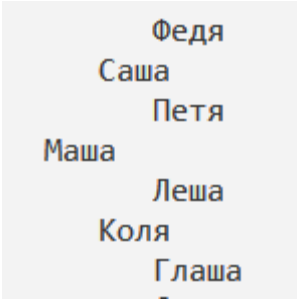
```
//--------------- ввод данных ---------------------
gruz_p vvod() {
     gruz_p p;
     printf("\nВведите Ф.И.О.");
          gets(p.fio);
     printf("Введите адрес");
          gets(p.address);
  return p;
}
//--------------- организация дерева ----------------
tree* org_tree() {
     tree* top=NULL;
     while (1) {
        top=addtree(top,vvod()); //ввод и  добавление элемента
        printf("хотите продолжить (выход - 0 )");
        if (getch()=='0')break;
     }
     return top;
}
```

Отображение структуры бинарного дерева

Для отображения структуры дерева опишем функцию, выводящую только значения ключевых полей. Дерево будет отображаться повернутым на 90 градусов, корень будет слева.

```
void otobr(tree *top, int otstup) {
     if (top) {
          otstup+=3;              //отступ от края экрана
          otobr(top->right,otstup); //обход правого поддерева
          //вывод значения фамилии, соответствующего узла
          for(int i=0; i<otstup; i++) printf(" ");
          printf("%s\n",top->data.fio);
          otobr(top->left,otstup); //обход левого поддерева
     }
}
```

Маша	Севастополь
Коля	Симферополь
Саша	Севастополь
Леша	Новгород
Глаша	Пермь
Петя	Ленинград
Федя	Москва




```
FIO = Маша      address = Севастополь
FIO = Коля      address = Симферополь
FIO = Глаша     address = Пермь
FIO = Леша      address = Новгород
FIO = Саша      address = Севастополь
FIO = Петя      address = Ленинград
FIO = Федя      address = Москва
```

Поиск узла в упорядоченном бинарном дереве

Упорядоченные бинарные деревья обеспечивают ***быстрый поиск*** записи по ее ключу.

Пример функции поиска узла с заданным значением **key** в бинарном дереве. Если дерево содержит узел со значением **key**, то функция возвращает указатель на найденный узел, иначе – **NULL**.

```
tree* find(char key[d_f], tree* top){
    tree* curr = top;
    while(curr && (strcmp(curr->data.fio, key)!=0)){
        if (strcmp(curr->data.fio, key)>0)
            curr = curr->left;
        else
            curr = curr->right;
    }
    return curr;
}
```

Вызов: `tree* temp=find("Федор", top);`

```
if (temp) printf("address= %s ", temp->data.address);
else printf("нет такого");
```

Формирование выровненного дерева

Эффективность поиска в бинарном дереве в значительной степени зависит от его симметричности.

Под **симметричным** понимается дерево, которое состоит из **n** уровней, причем **n-1** уровень занят полностью. Если листья дерева располагаются только на двух соседних уровнях **n-1** и **n**, а **n-1 = trunc($\log_2$ M),** то такое дерево называется **выровненным** (**M**-количество вершин дерева).

Формирование упорядоченного выровненного дерева можно выполнить с помощью следующего алгоритма.

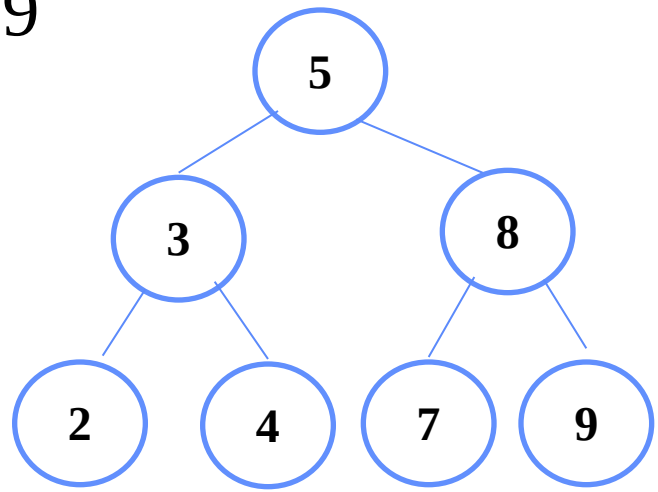
1. Исходные значения, помещаемые в дерево, должны иметь вид ***упорядоченного массива***.

2. За корень дерева принимается средняя запись массива, которая разделяет массив примерно на две равные части.

3. Средние записи в обеих частях массива образуют вершины второго уровня, а массив оказывается разделенным на 4 части.

Формирование выровненного дерева

2 3 4 5 7 8 9



4. Средние записи каждой из четырех частей помещаются на третьем уровне бинарного дерева.

5. Процесс продолжается до тех пор, пока все записи массива не будут внесены в дерево.

Удаление узла из бинарного дерева

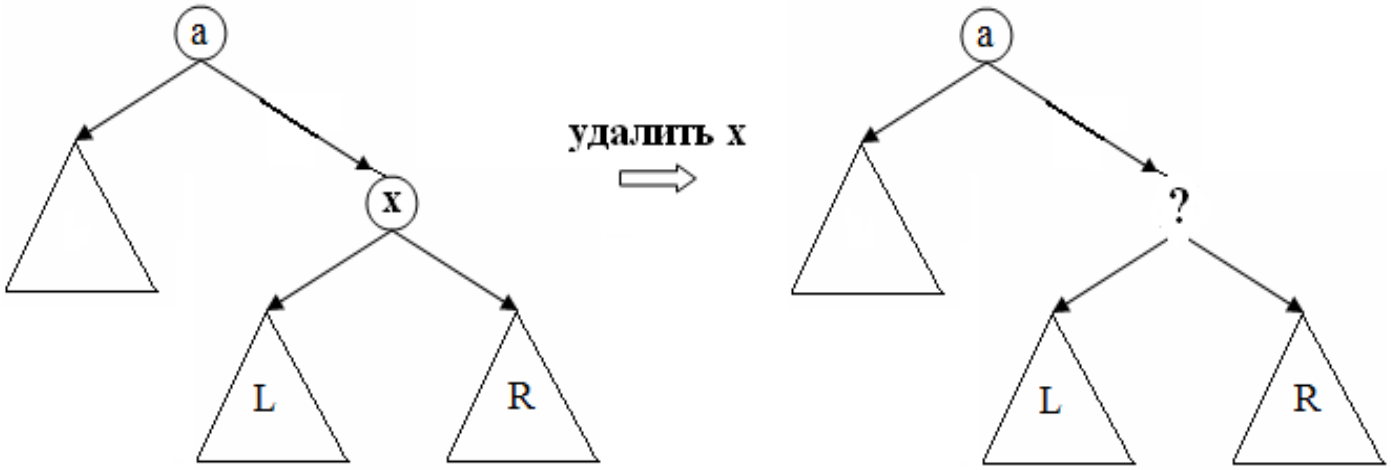
При удалении узла выделяют несколько случаев:

1. удаление узла с ключом, равным **node**, нет;
2. узел с ключом **node** имеет не более одного потомка;
3. узел с ключом **node** имеет 2-х потомков.

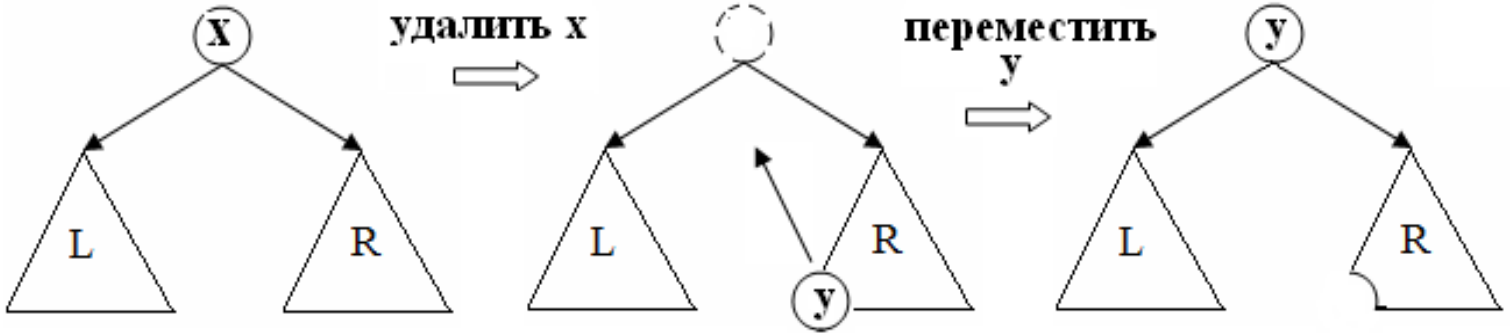
Последний случай самый сложный, чтобы сохранить упорядоченность дерева при удалении узла с двумя потомками, его заменяют на узел с самым близким к нему ключом. Это может быть самый левый узел его правого поддерева (1 способ) или самый правый узел левого поддерева (2 способ).

Рассмотрим реализацию 1 способа.

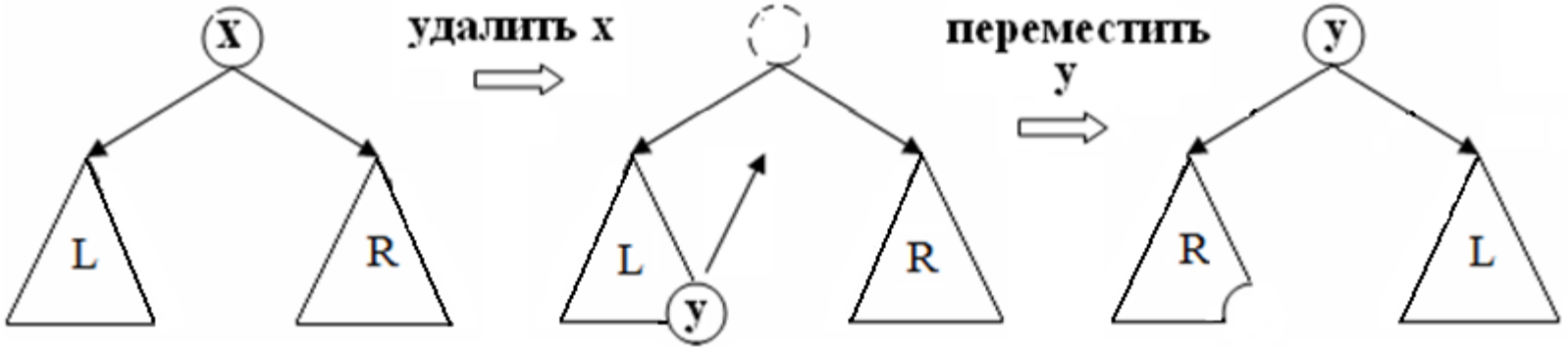
Удаление узла из бинарного дерева



I



II



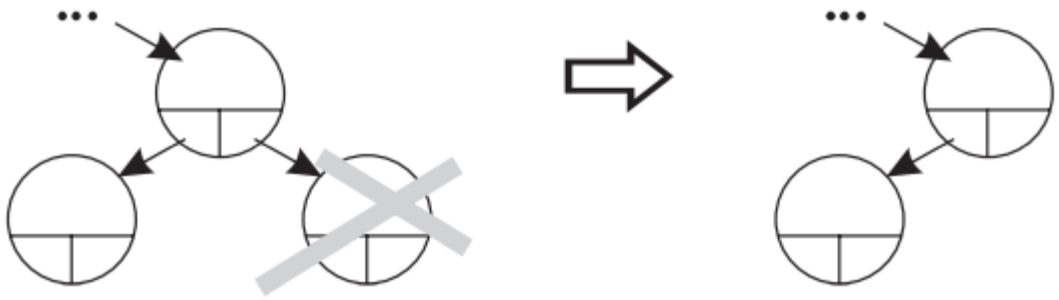
Удаление узла из бинарного дерева поиска

Процесс удаления можно разбить на этапы:

- найти узел, который будет поставлен на место удаляемого;
- реорганизовать дерево так, чтобы не нарушились его свойства;
- присоединить новый узел к узлу-предку удаляемого узла;
- освободить память из-под удаляемого узла.

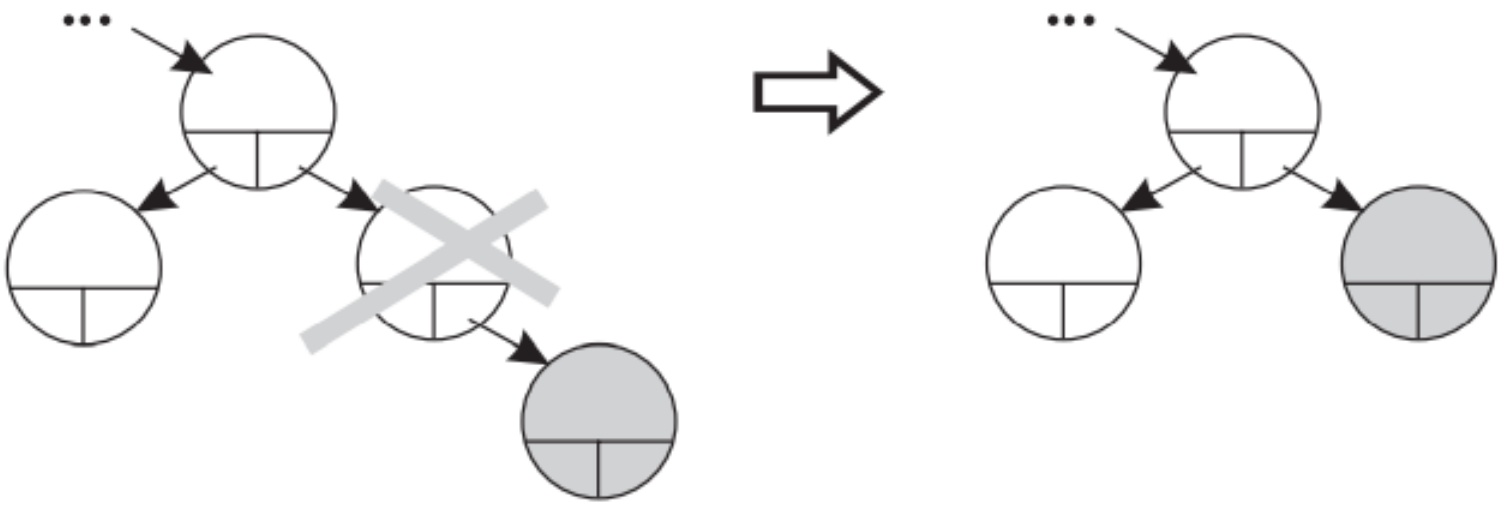
Удаление узла происходит по-разному в зависимости от его расположения в дереве.

Если узел является листом, то есть не имеет потомков, достаточно обнулить соответствующий указатель узла-предка:



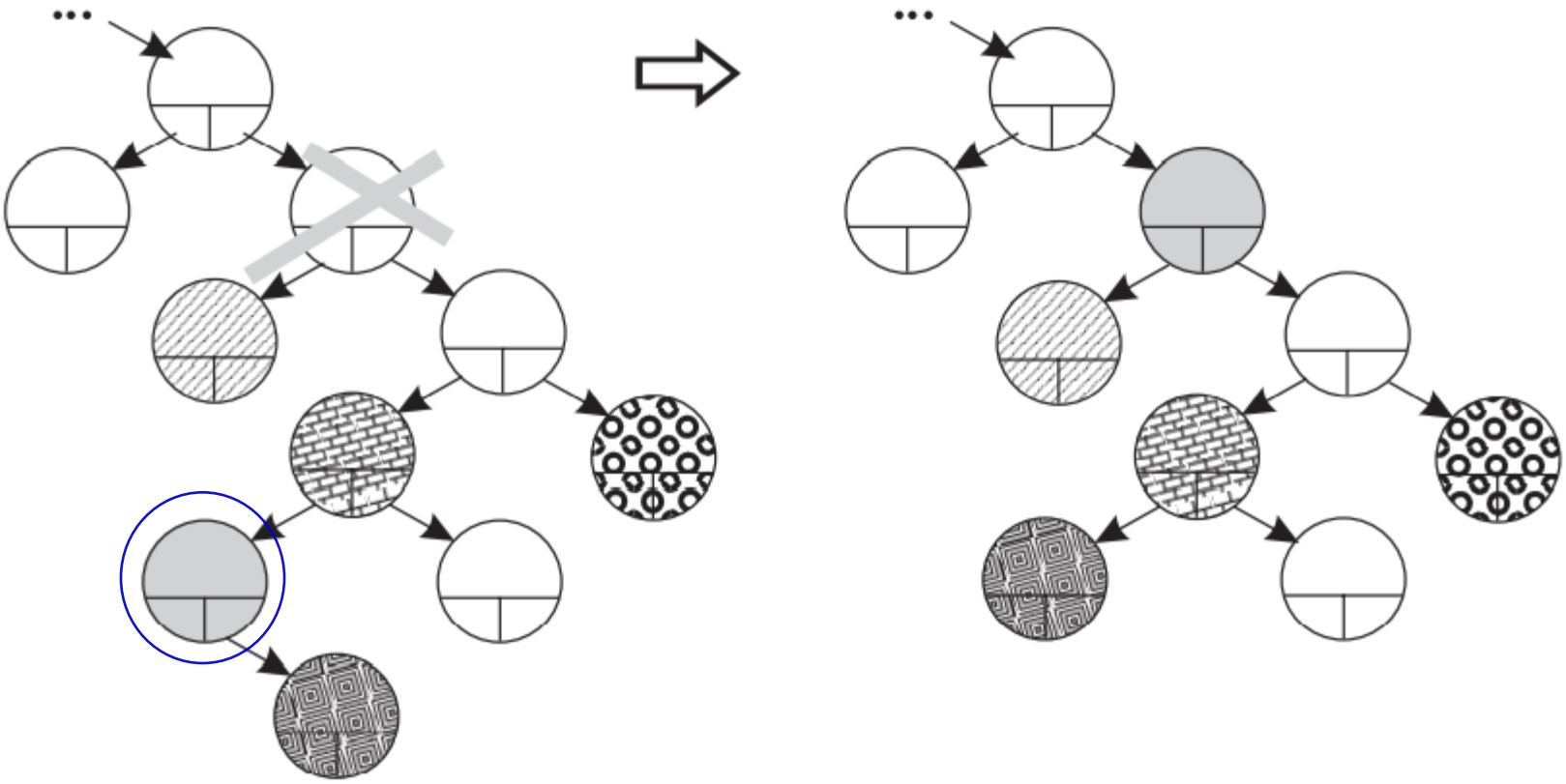
Удаление узла из бинарного дерева поиска

Если узел имеет только одного потомка, то этот потомок ставится на место удаляемого узла, а в остальном дерево не изменяется:

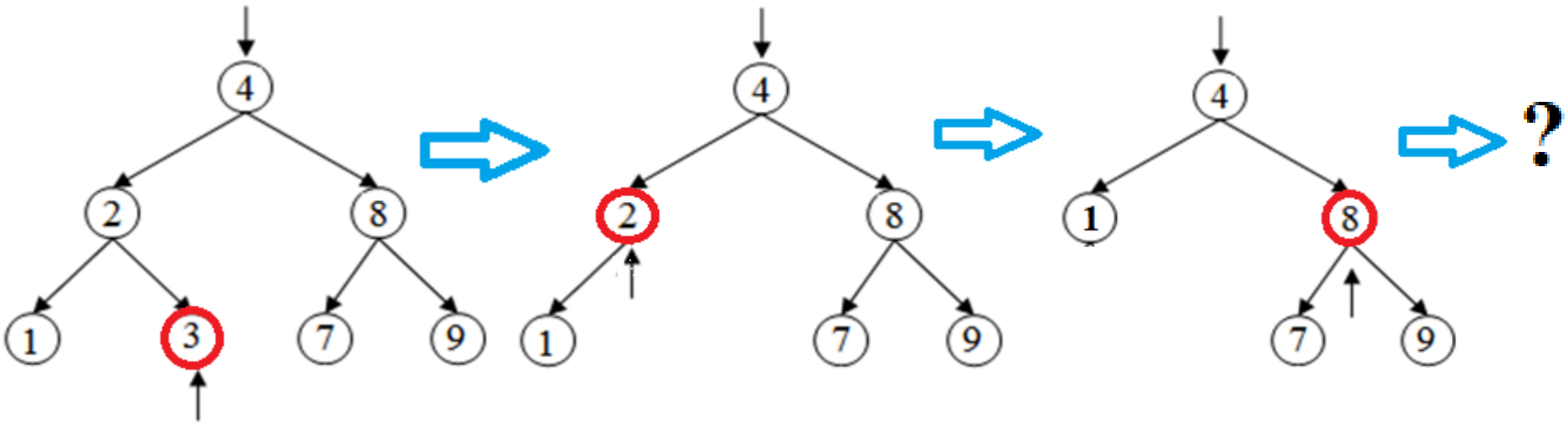


Удаление узла из бинарного дерева поиска

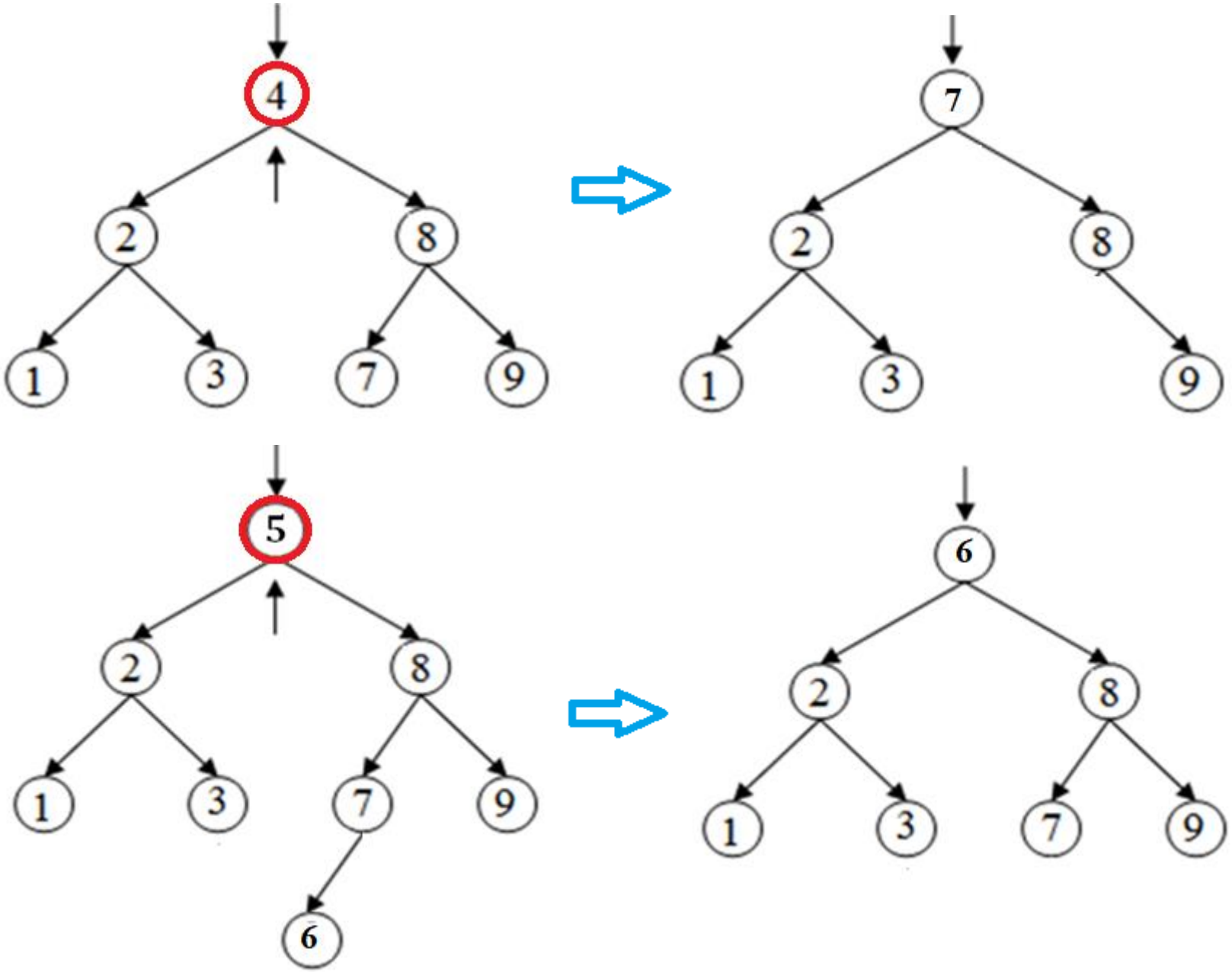
Если у узла есть оба потомка, на место удаляемого узла помещается самый левый лист его правого поддерева, это не нарушает свойств дерева поиска:



Примеры преобразования БД при удалении узлов



Примеры преобразования БД при удалении узлов



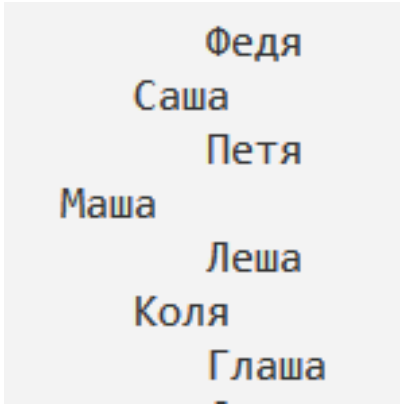
Удаление узла из бинарного дерева поиска

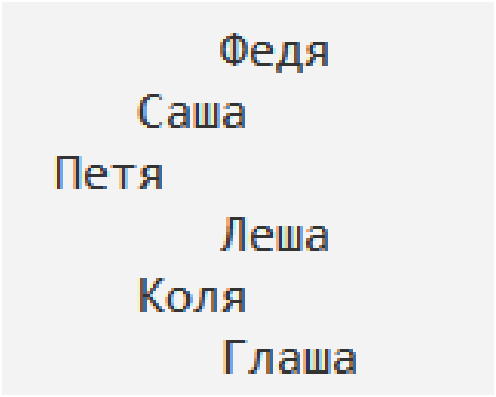
```
tree* DeleteNode(tree* node, char val[d_f]) {
   if(node == NULL) return node; // выход если пустой узел
   if(strcmp(val,node->data.fio)==0) { //найден удал. узел
      tree* tmp;                    // указатель
      if(node->right == NULL) tmp = node->left;
      else {                        // существует правое поддерево
        tree* ptr = node->right;
        if(ptr->left == NULL) { // у правого ПД отсутствует левое ПД
            ptr->left = node->left;
            tmp = ptr;
         } else { tree* pmin = ptr->left; // поиск самого левого
                  while(pmin->left != NULL) {// узла в правом ПД
                      ptr  = pmin;
                      pmin = ptr->left;
                  }  // найден самый левый узел правого ПП (pmin)
                  ptr->left   = pmin->right;
                  pmin->left  = node->left;
                  pmin->right = node->right;
                  tmp = pmin;
                }
      }
```

Удаление узла из бинарного дерева поиска

```
        delete node;
        return tmp;
    } else                          //поиск в левом или правом поддереве
      if(strcmp(val, node->data.fio)<0)
        node->left  = DeleteNode(node->left, val);
    else
        node->right = DeleteNode(node->right, val);
    return node;
}
```

Удалим корень «Маша» из рассмотренного выше примера и получим:





Удаление бинарного дерева

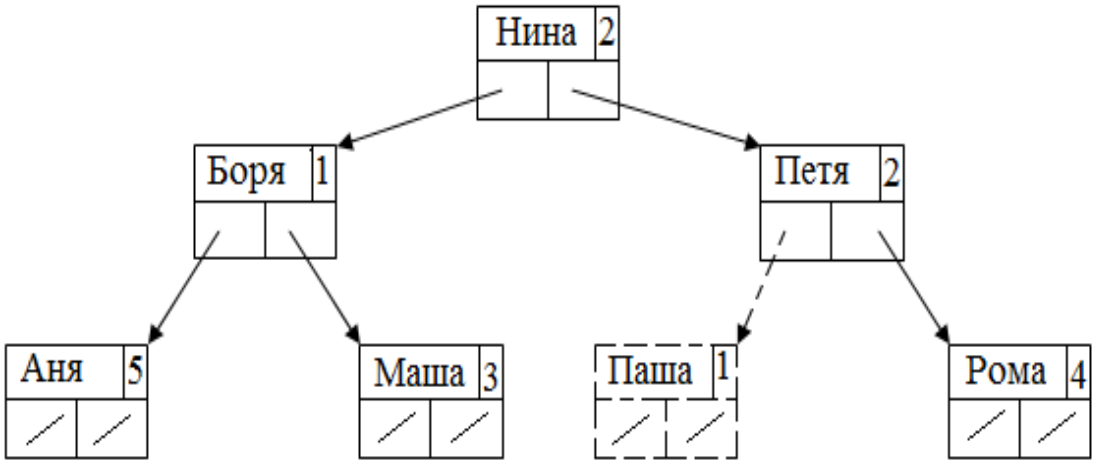
Последним действием при выходе из программы обязательно необходимо освободить память:

```
// --------------- Удаление дерева ----------------
void remove_tree(tree* top) {
   if (top!=NULL) {              // условие выхода из рекурсии
      remove_tree(top->left);     //  вызов для левого ПД
      remove_tree(top->right);    // вызов для правого ПД
      free (top);                 // освобождение памяти
   }
}
```

Поиск с включением узлов в бинарное дерево

Рассмотрим **задачу создания частотного словаря**: требуется подсчитать сколько раз встречается каждое слово в заданной последовательности слов.

Будем размещать слова в бинарном поисковом дереве. Эту задачу называют также **поиск с включением узлов в бинарное дерево**.



Поиск с включением предполагает выполнение двух операций:

1. Поиск заданного слова в дереве. Если слово встретилось, то увеличиваем счётчик на 1;

2. Если слово не найдено, то добавляем новый узел.

Поиск с включением узлов в бинарное дерево

```
#include <stdio.h>
#include <iostream>
#include <ctype.h>
#include <string.h>
#define MAXWORD 100
//-----------------------------------------
struct my_tree {              // узел дерева
     char word[MAXWORD];      // указатель на слово
     int count;               // число вхождений
     struct my_tree *left;  // левое поддерево
     struct my_tree *right; // правое поддерево
};
//-----------------------------------------
struct my_tree *addtree(struct my_tree *p, char *w);
void treeprint(struct my_tree *p);
void otobr(my_tree *p, int otstup);
struct my_tree *talloc(void);
```

Поиск с включением узлов в бинарное дерево

```
/* подсчет частоты встречаемости слов */
main() {
     struct my_tree *top=NULL;
     char word[MAXWORD];
     puts("\nВводите текст:");
     while (gets (word) != NULL)          // считываем слово
            top = addtree(top, word); //добавляем в дерево
     puts("\nЧастота встречаемости слов:");
     treeprint(top);  // печать дерева (обход слева направо)
     otobr(top,1);    // печать структуры дерева
     return 0;
}
```

Поиск с включением узлов в бинарное дерево

```
//------- функция поиска и добавления ------------
struct my_tree *addtree(struct my_tree *p, char *w){
 int cond;
 if (p == NULL) {        // слово встречается впервые
     p = talloc();       // создается новый узел
     strcpy(p->word,w);
     p->count = 1;
     p->left = p->right = NULL;
   } else
      if ((cond = strcmp(w, p->word)) == 0)
           p->count++;    // это слово уже встречалось
     else if (cond < 0)    // меньше корня
           p->left = addtree(p->left, w);
     else                          // больше корня
           p->right = addtree(p->right, w);
     return p;
 }
```

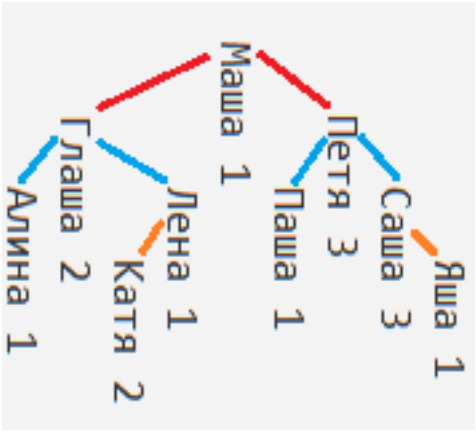
Поиск с включением узлов в бинарное дерево

```
//-------- печать дерева (обход слева направо)---------
void treeprint(struct my_tree *p) {
     if (p!=NULL) {
          treeprint(p->left);
          printf("%4d %s\n", p->count, p->word);
          treeprint(p->right);
     }
}
//--------выделение пямяти под новый узел ------------
struct my_tree *talloc(void) {
return (struct my_tree*)malloc(sizeof(struct my_tree));
}
// ----------------отображение структуры дерева-----
void otobr(my_tree *p, int otstup) {
if (p) {   otstup+=3;                    //отступ от края экрана
          otobr(p->right,otstup); //обход правого поддерева
          for(int i=0; i<otstup; i++) printf(" ");
          printf("%s %d\n",p->word,p->count);
          otobr(p->left,otstup);   //обход левого поддерева
     }
}
```

Поиск с включением узлов в бинарное дерево

Входные данные:

```
Вводите текст:
Маша
Петя
Саша
Яша
Глаша
Паша
Петя
Саша
Лена
Катя
Катя
Глаша
Алина
Петя
Саша
^Z
```



Результат:

```
Частота встречаемости слов:
   1 Алина
   2 Глаша
   2 Катя
   1 Лена
   1 Маша
   1 Паша
   3 Петя
   3 Саша
   1 Яша
            Яша 1
         Саша 3
      Петя 3
         Паша 1
   Маша 1
         Лена 1
            Катя 2
      Глаша 2
         Алина 1
```

ИТОГИ

Деревья являются одними из наиболее широко распространенных структур данных в программировании, которые представляют собой иерархические структуры в виде набора связанных узлов.

Каждое дерево обладает следующими свойствами: существует узел, в который не входит ни одной дуги (корень); в каждую вершину, кроме корня, входит одна дуга (ветвь).

С понятием дерева связаны такие понятия, как корень, ветвь, вершина, лист, предок, потомок, степень вершины и дерева, глубина дерева.

Дерево можно упорядочить по указанному ключу.

Просмотреть все вершины дерева можно с помощью различных способов обхода дерева.

ИТОГИ

В программировании при решении большого класса задач используются бинарные деревья.

Бинарные деревья по степени вершин делятся на строгие и нестрогие, по характеру заполнения узлов – на полные и неполные, по удалению вершин от корня – на сбалансированные и почти сбалансированные.

Основными операциями 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.

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