

CPP

02_01

ЛР

1. ЛР Деревья
2. Курс С++
3. Курс Java
4. ЛР Qt
5. ЛР QML

Рубежки

- Деревья
- Qt & QML

Экзамен

- Python

Курсы Stepik

- <https://stepik.org/course/7/syllabus>
- <https://stepik.org/course/187/syllabus>

OpenEdu

- <https://openedu.ru/course/ITMOUniversity/PWADEV/>

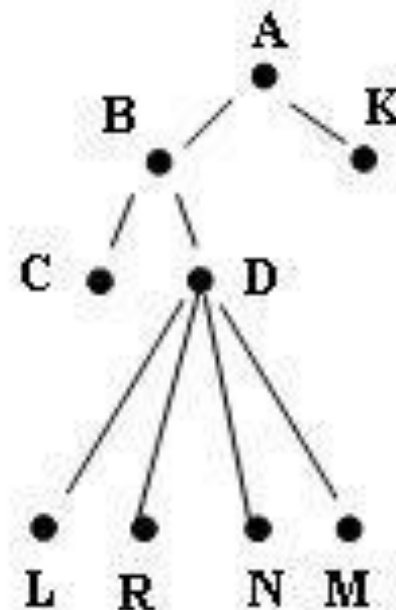
Деревья

Определение 1

- **дерево** как конечное множество T , состоящее из одного или более элементов (называемых вершинами или узлами), таких, что
- имеется одна специально выделенная вершина, называемая **корнем дерева**;
- остальные вершины (исключая корень) содержатся в m попарно непересекающихся множествах T_1, T_2, \dots, T_m , каждое из которых, в свою очередь, является деревом.
-
- Деревья T_1, T_2, \dots, T_m называются **поддеревьями** данного дерева.
- **Упорядоченным** деревом мы будем называть такое дерево, в котором важен порядок следования поддеревьев T_1, T_2, \dots, T_m .
-

- **Дуга** - это ориентированная связь между двумя вершинами дерева, поэтому, например, корень можно определить как такую вершину дерева, в который не входит ни одной дуги, поэтому часто говорят, что корень - это "исходная" вершина дерева, через которую доступны остальные его вершины.
- **Ребро** - это неориентированная связь между двумя вершинами дерева. Ясно, что ребро можно превратить в дугу, если задать на нем ориентацию (направление), а любое дерево можно превратить в ориентированное дерево, если задать ориентацию ребер.
- Количество поддеревьев некоторой вершины называется степенью этой вершины. Деревья, имеющие степень больше 2, называются **сильно ветвящимися деревьями**.
- Вершина с нулевой степенью называется **листом**, иначе - она называется внутренней вершиной (внутренним узлом).
- Число листьев дерева называется **весом дерева**.
- Символы A, B, C, ..., которые служат для обозначения вершин, называются **метками вершин**.

- **A, B, C, D, K, L, M, N, R** - метки вершин,
вершина **A** - корень,
вершины **C, L, R, M, N, K** - листья, вес
дерева равен **6** (количество листьев - 6),
вершина **B** имеет степень 2,
вершина **D** имеет степень 4



Определение 2

- Вершина Y , которая находится непосредственно под узлом X , называется (непосредственным) **потомком** (сыном) X , вершина X в данном случае называется (непосредственным) **предком** (отцом) Y .
- В этом случае, если вершина X находится на уровне i , то говорят, что вершина Y находится на уровне $i+1$. Мы будем считать, что **корень дерева** расположен на уровне 0. Максимальный уровень какой-либо вершины дерева называется его глубиной или высотой.
- Максимальная степень всех вершин дерева называется **степенью дерева**.

Следствия

- если вершина не имеет потомков, то она является листом;
- степень внутренней вершины можно определить как число ее (непосредственных) потомков.

- максимальное число вершин для дерева с высотой h и степенью d можно найти по формуле

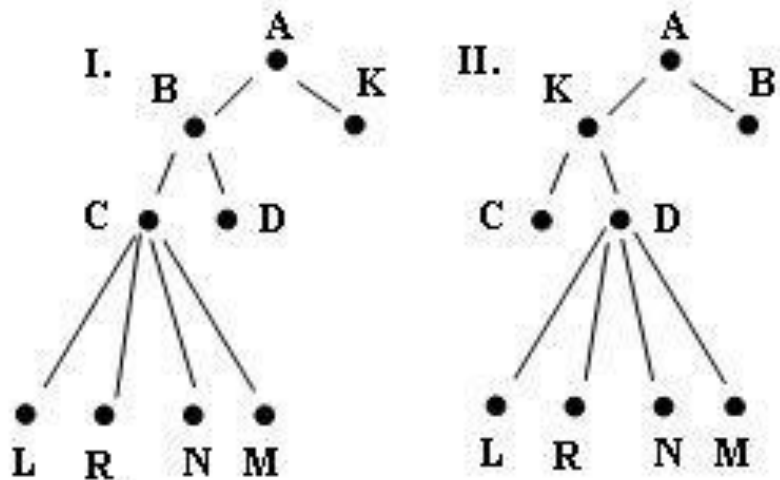
$$N(d, h) = 1 + d + d^2 + \dots + d^h = \sum_{i=0}^h d^i$$

$$N(2, h) = 1 + 2 + 2^2 + \dots + 2^h = \sum_{i=0}^h 2^i = 2^{h+1} - 1$$

Определение 3

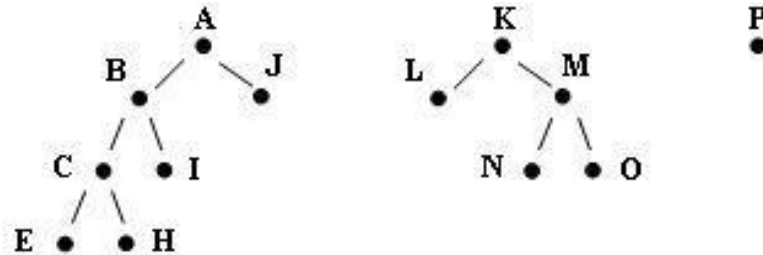
- Количество дуг, которые нужно пройти, чтобы продвинуться от корня к вершине X , **называется длиной пути к вершине X .**
- Вершина, расположенная на уровне i , имеет длину пути i .
- **Ветвью** будем называть путь от корня дерева к любому ее листу.
- **Длина пути дерева** определяется как сумма длин путей ко всем его вершинам. Она также называется длиной внутреннего пути дерева.

- Длина
 внутреннего пути =
 Длина
 внутреннего пути
 в левом
 поддереве +
 Длина
 внутреннего пути
 в правом
 поддереве +
 Количество узлов
 в дереве - 1.



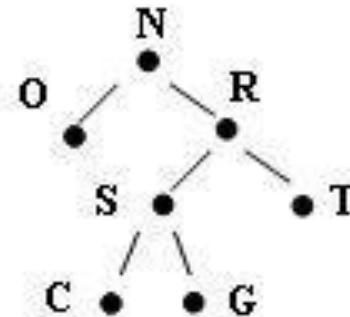
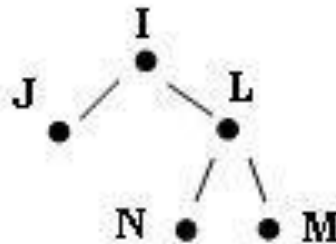
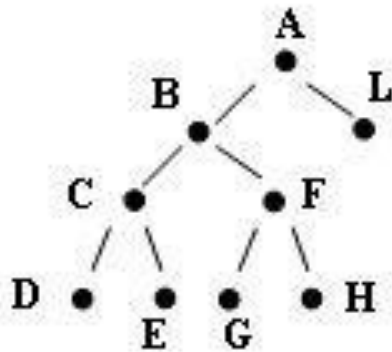
Определение 4

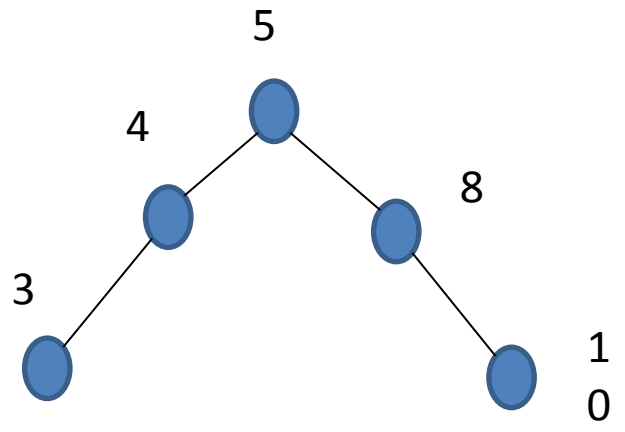
- **Лес** - это множество деревьев (обычно упорядоченное), состоящее из некоторого (быть может, равного нулю) числа непересекающихся деревьев. Часто для леса, состоящего из n деревьев пользуются термином "дерево с n -кратным корнем".



Определение 5

- **бинарное дерево** конечное множество элементов (называемых вершинами или узлами), которое:
- либо пусто,
- либо состоит из корня (некоторая выделенная нами вершина), связанного с двумя различными бинарными деревьями, называемыми левым и правым поддеревом корня



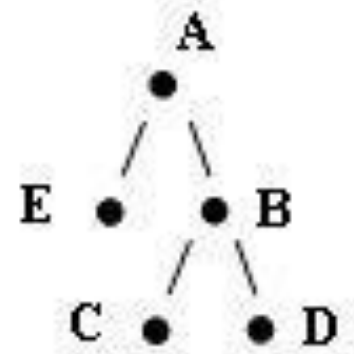
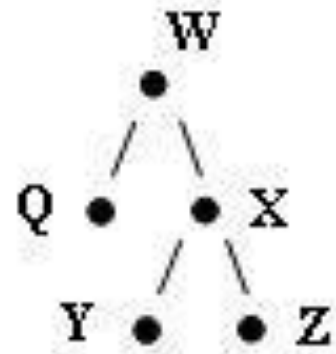
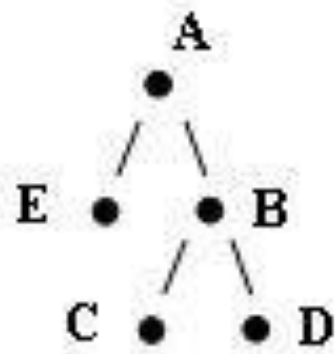
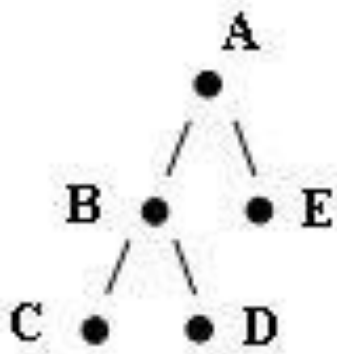


Определение 6

- два бинарных дерева T и T' **подобны**, если они имеют одинаковую структуру; это означает, что подобные деревья либо оба пусты, либо оба непусты и их левые и правые поддеревья соответственно подобны.
- Попросту говоря, подобие означает, что графические изображения деревьев T и T' имеют одинаковую "конфигурацию".

- бинарные деревья T и T' **эквивалентны**, если они подобны и если, кроме того, соответствующие вершины содержат одинаковую информацию.
- Если $\text{Info}(u)$ обозначает информацию, содержащуюся в вершине u , то формально деревья эквивалентны тогда и только тогда, когда они:
 - либо оба пусты,
 - либо же оба непусты, $\text{Info}(\text{Корень}(T)) = \text{Info}(\text{Корень}(T'))$ и их левые и правые поддеревья соответственно эквивалентны.

- Первые два из них не подобны;
второе, третье и четвертое деревья



Бинарные деревья поиска

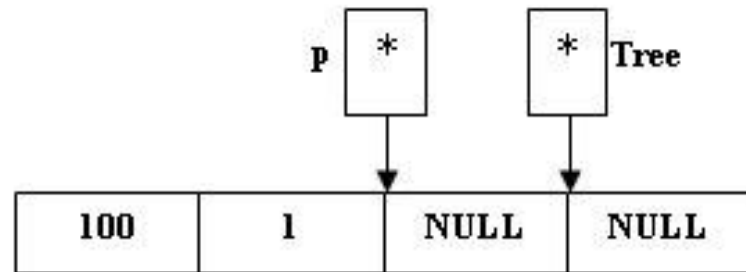
- Каждая вершина бинарного дерева является структурой, состоящей из четырех полей:
- информационное поле (ключ вершины),
- служебное поле (их может быть несколько!),
- указатель на левое поддерево,
- указатель на правое поддерево.

- struct node
- {
- int Key; // Ключ вершины.
- int Count; // Счетчик количества вершин с одинаковыми ключами.
- node *Left; // Указатель на "левого" сына.
- node *Right; // Указатель на "правого" сына.
- };

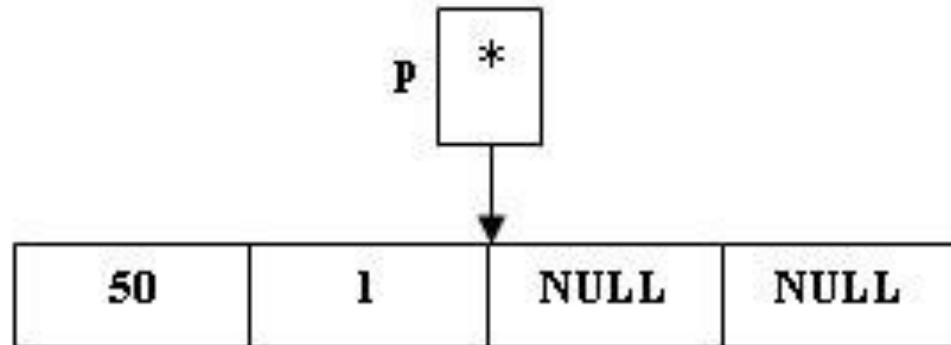
Построение бинарного дерева поиска

- **Tree** - указатель на корень дерева
- **p** - вспомогательный указатель на вершину дерева

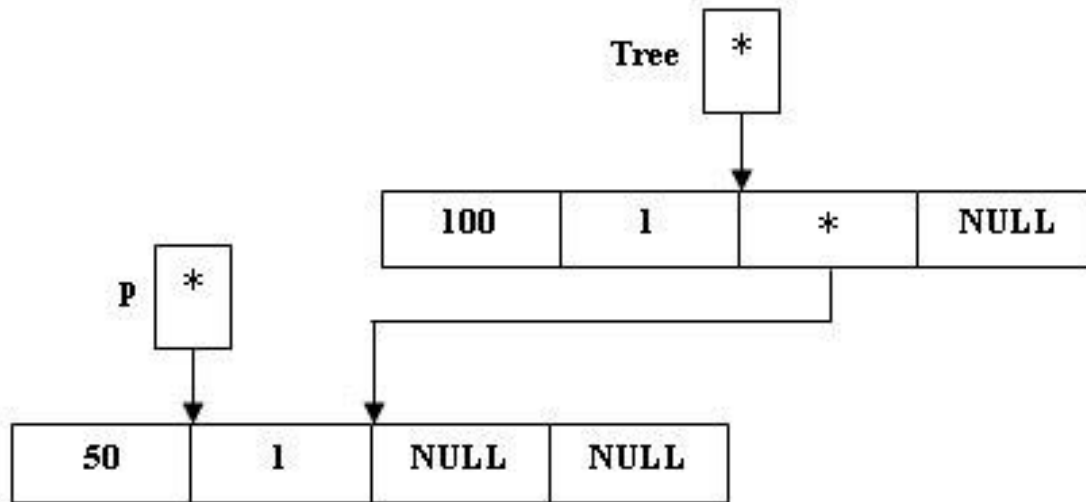
- `Tree = NULL; //Построение пустого дерева`
- `p = new(node);`
- `(*p).Key = 100;`
- `(*p).Count = 1;`
- `(*p).Left = NULL;`
- `(*p).Right = NULL;`
- `Tree = p;`



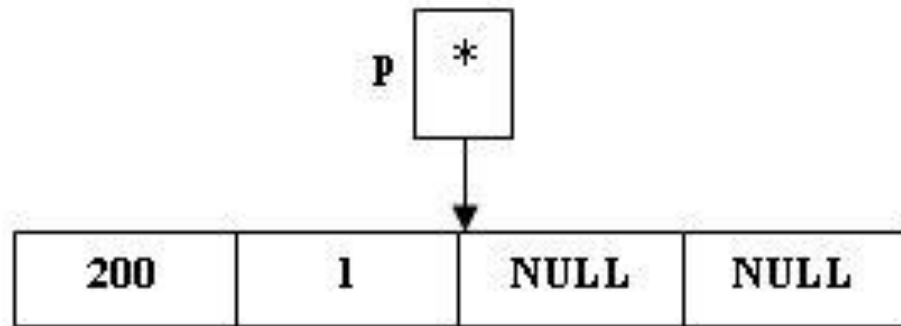
- `p = new(node);`
- `(*p).Key = 50;`
- `(*p).Count = 1;`
- `(*p).Left = NULL;`
- `(*p).Right = NULL;`



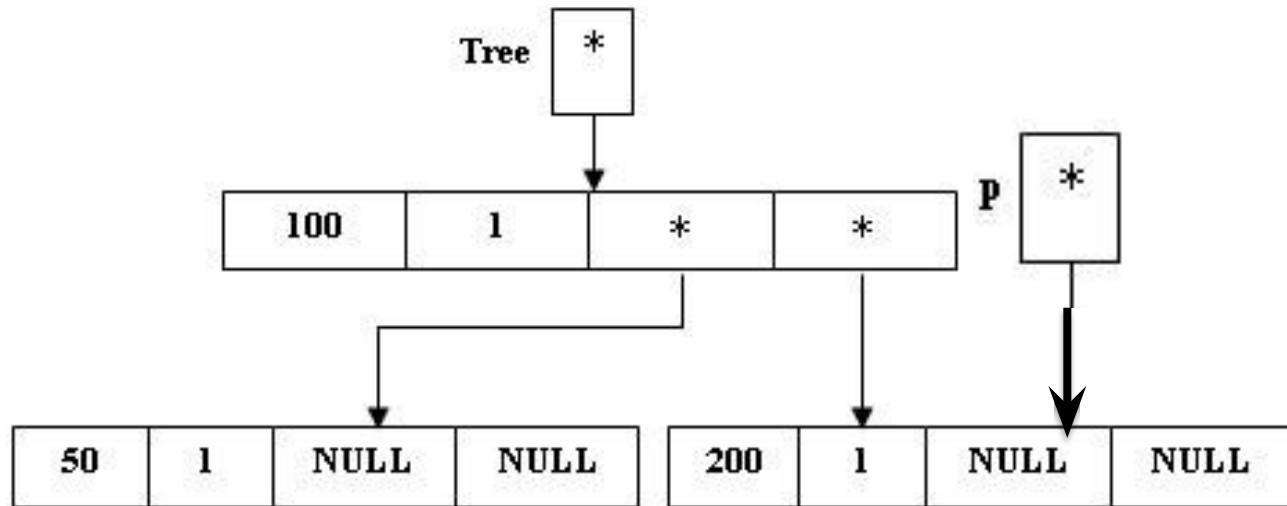
- `(*Tree).Left = p;`



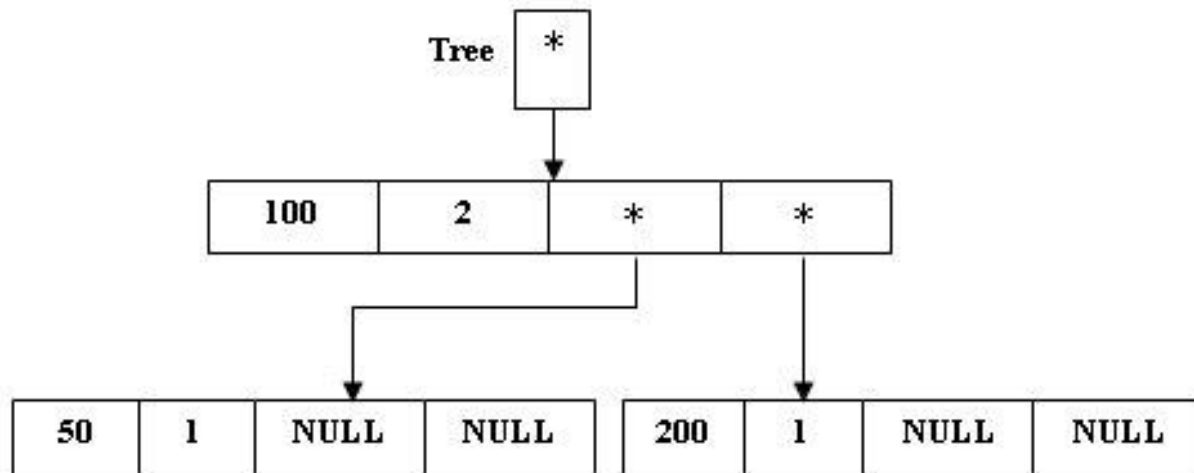
- `p = new(node);`
- `(*p).Key = 200;`
- `(*p).Count = 1;`
- `(*p).Left = NULL; (*p).Right = NULL;`



- `(*Tree).Right = p;`



- $(*Tree).Count = (*Tree).Count + 1;$



- `void BuildTree (node **Tree)`
- `// Построение бинарного дерева.`
- `// *Tree - указатель на корень дерева.`
- `{`
- `int el;`

- `*Tree = NULL; // Построено пустое бинарное дерево.`
- `cout<<"Вводите ключи вершин дерева...\n";`
- `cin>>el;`
- `while (el!=0)`
- `{ Search (el,Tree); cin>>el;}`
- `}`

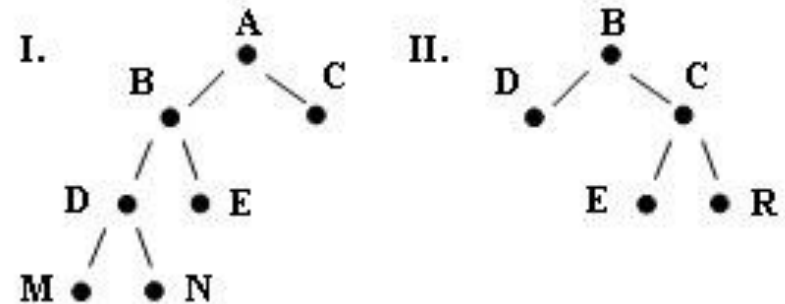
- void Search (int x, node **p)
- // Поиск вершины с ключом x в дереве со вставкой
- // (рекурсивный алгоритм).
- // *p - указатель на корень дерева.
- {
- if (*p==NULL)
- {
- // Вершины с ключом x в дереве нет; включить ее.
- *p = new(node);
- (**p).Key = x;
- (**p).Count = 1;
- (**p).Left = (**p).Right = NULL;
- }
- else
- //Поиск места включения вершины.
- if (x<(**p).Key)
- //Включение в левое поддерево.
- Search (x,&((**p).Left));
- else if (x>(**p).Key)
- //Включение в правое поддерево.
- Search (x,&((**p).Right));
- else (**p).Count = (**p).Count + 1;
- }
- }

Анализ алгоритма поиска с включениями

- **Теорема Хопкрофта-Ульмана**
- Среднее число сравнений, необходимых для вставки n случайных элементов в дерево поиска, пустое вначале, равно $O(n \log_2 n)$ для $n \geq 1$.

Левосторонний обход бинарного дерева поиска

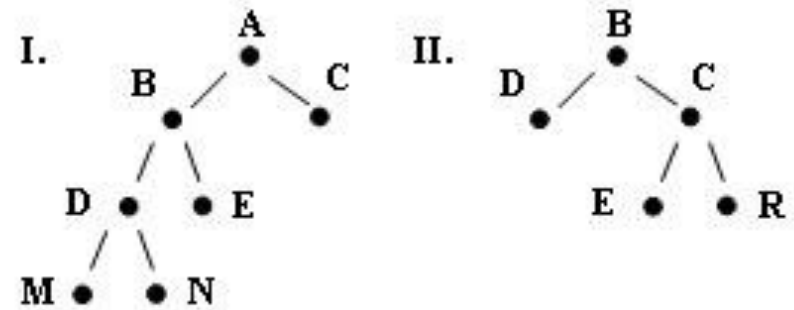
- A B D M N E C
- B D C E R
- посетите корень дерева
- обойдите левое поддерево;
- обойдите правое поддерево.



- `void ObhodLeft (node **w)`
- `// Левосторонний обход дерева.`
- `// *w - указатель на корень дерева.`
- `{`
- `if (*w!=NULL)`
- `{ cout<<(**w).Key<<" ";`
- `ObhodLeft (&(**w).Left);`
- `ObhodLeft (&(**w).Right); }`
- `}`

Концевой обход бинарного дерева поиска

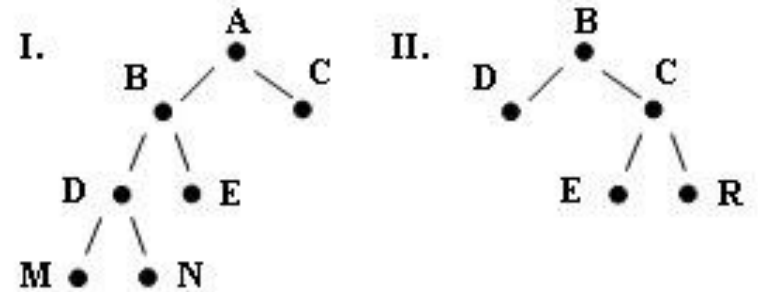
- обойдите левое поддерево;
- обойдите правое поддерево;
- посетите корень дерева.
- M N D E B C A
- D E R C B



- `void ObhodEnd (node **w)`
- `// Концевой обход дерева.`
- `// *w - указатель на корень дерева.`
- `{`
- `if (*w!=NULL)`
- `{ ObhodEnd (&((*w).Left));`
- `ObhodEnd (&((*w).Right));`
- `cout<<((*w).Key<<" "};`
- `}`

Обратный обход бинарного дерева поиска

- обойдите левое поддерево;
 - посетите корень дерева;
 - обойдите правое поддерево.
- M D N B E A C
- D B E C R



- `void ObhodBack (node **w)`
- `// Обратный обход бинарного дерева.`
- `// *w - указатель на корень дерева.`
- `{`
- `if (*w!=NULL)`
- `{ ObhodBack (&((*w).Left));`
- `cout<<((*w).Key<<" ";`
- `ObhodBack (&((*w).Right)); }`
- `}`

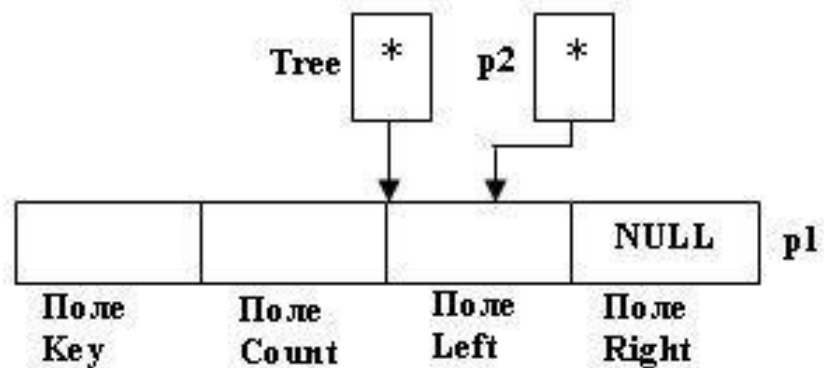
Вывод бинарного дерева поиска

- void Vyvod (node **w,int l)
- // Изображение дерева w на экране дисплея.
- // (рекурсивный алгоритм).
- // *w - указатель на корень дерева.
- {
- int i;

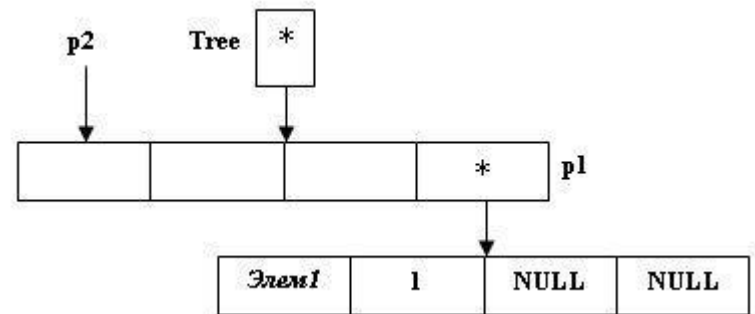
- if (*w!=NULL)
- { Vyvod (&(**w).Right),l+1);
- for (i=1; i<=l; i++) cout<<" ";
- cout<<(**w).Key<<endl;
- Vyvod (&(**w).Left),l+1); }
- }

Построение бинарного дерева (нерекурсивный алгоритм)

- `Tree = new(node);`
- `(*Tree).Right = NULL;`
- `p2 = Tree;`
- `p1 = (*p2).Right;`



- `p1 = new(node);`
- `(*p1).Key = Элем1;`
- `(*p1).Left =`
`(*p1).Right = NULL;`
- `(*p1).Count = 1;`



- void TreeSearch (node **Tree,int el)
- // Поиск вершины с информационным полем el в дереве
- // с последующим включением.
- // *Tree - указатель на корень дерева.
- {
- node *p1;
- node *p2; // Указатель p2 "опережает" указатель p1.
- int d; // Флаг для распознавания поддеревьев.
-
- p2 = *Tree; p1 = (*p2).Right;
- d = 1; // Флаг правого поддерева.
- while (p1!=NULL && d!=0)
- { p2 = p1;
- if (el<(*p1).Key) { p1 = (*p1).Left; d = -1; //Флаг левого поддерева. }
- else
- if (el>(*p1).Key) { p1 = (*p1).Right; d = 1; }
- else d = 0; }
- if (d==0) (*p1).Count = (*p1).Count + 1;
- else
- { p1 = new(node);
- (*p1).Key = el; (*p1).Left = (*p1).Right = NULL; (*p1).Count = 1;
- if (d<0) (*p2).Left = p1; else (*p2).Right = p1;}
- }

Изображение бинарного дерева (нерекурсивный алгоритм)

- struct no
- {
- no *sled; // Указатель на вершину.
- node *elem; // Информационное поле.
- int ch; // Уровень вершины.
- }



- Создание БД
- Поиск по БД
- Левосторонний обход БД
- Обратный обход БД
- Концевой обход БД