

Обход всего дерева

```
void wrtree(ttree *p)
{
    if (p==NULL) return;
    wrtree(p->left);
    cout << p->inf << " ";
    wrtree(p->right);
}
```

Поиск элемента с заданным ключом

```
void poisktree(ttrees *p,int key,  
                bool &b, int &inf)  
{  
if ((p != NULL) && (b != true))  
{  
    if (p->inf !=key)  
    {  
poisktree(p->left,  key, b, inf);  
poisktree(p->right, key, b, inf);  
    }  
}
```

```
else
{
    b=true;
    inf=p->inf;
}
}
return;
}
```

Поиск элемента с максимальным ключем

```
int poiskmaxtree(ttree *p)
{
while (p->righth != NULL) p = p->righth;
return p->inf;
}
```

Поиск элемента с минимальным ключом

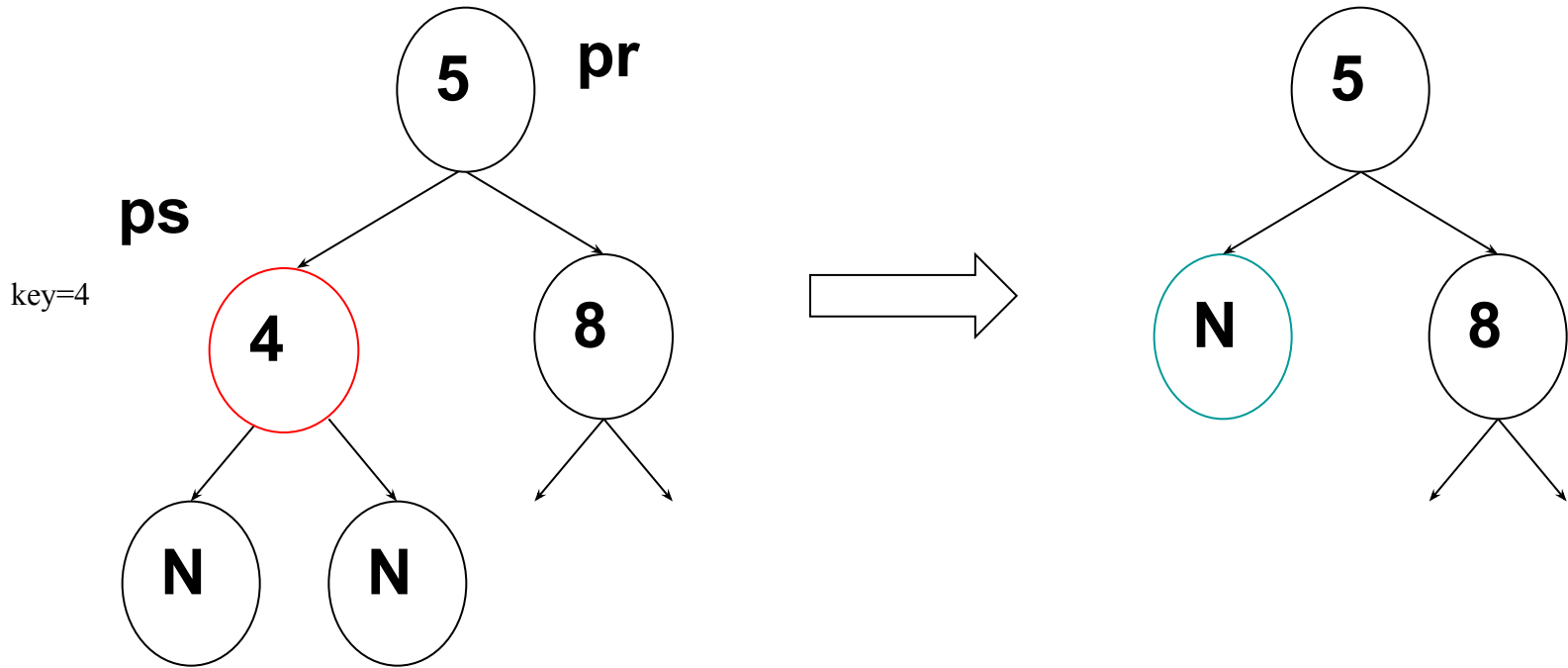
```
int poiskmintree(ttree *p)
{
while (p->left != NULL) p = p-> left;
return p->inf;
}
```


Удаление всего дерева

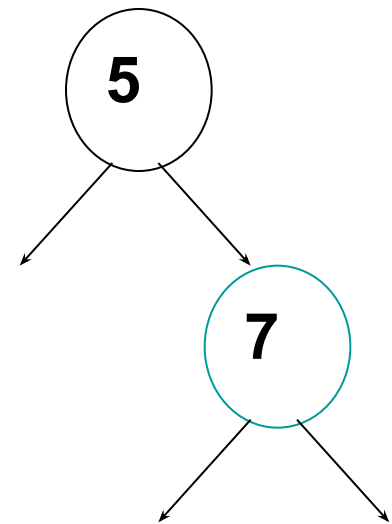
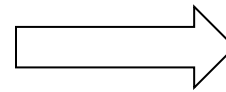
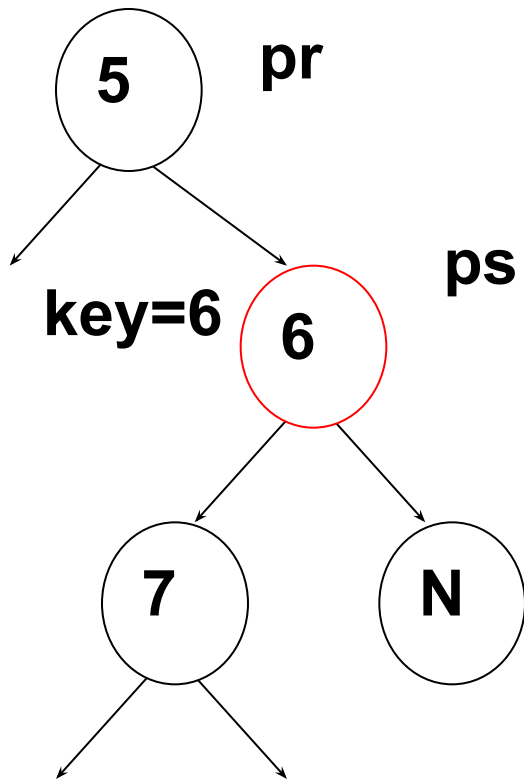
```
ttree *deltree(ttree *p)
{
    if (p==NULL) return NULL;
    deltree(p->left);
    deltree(p->right);
    delete(p);
return NULL;
}
```

Удаление элемента с заданным ключом

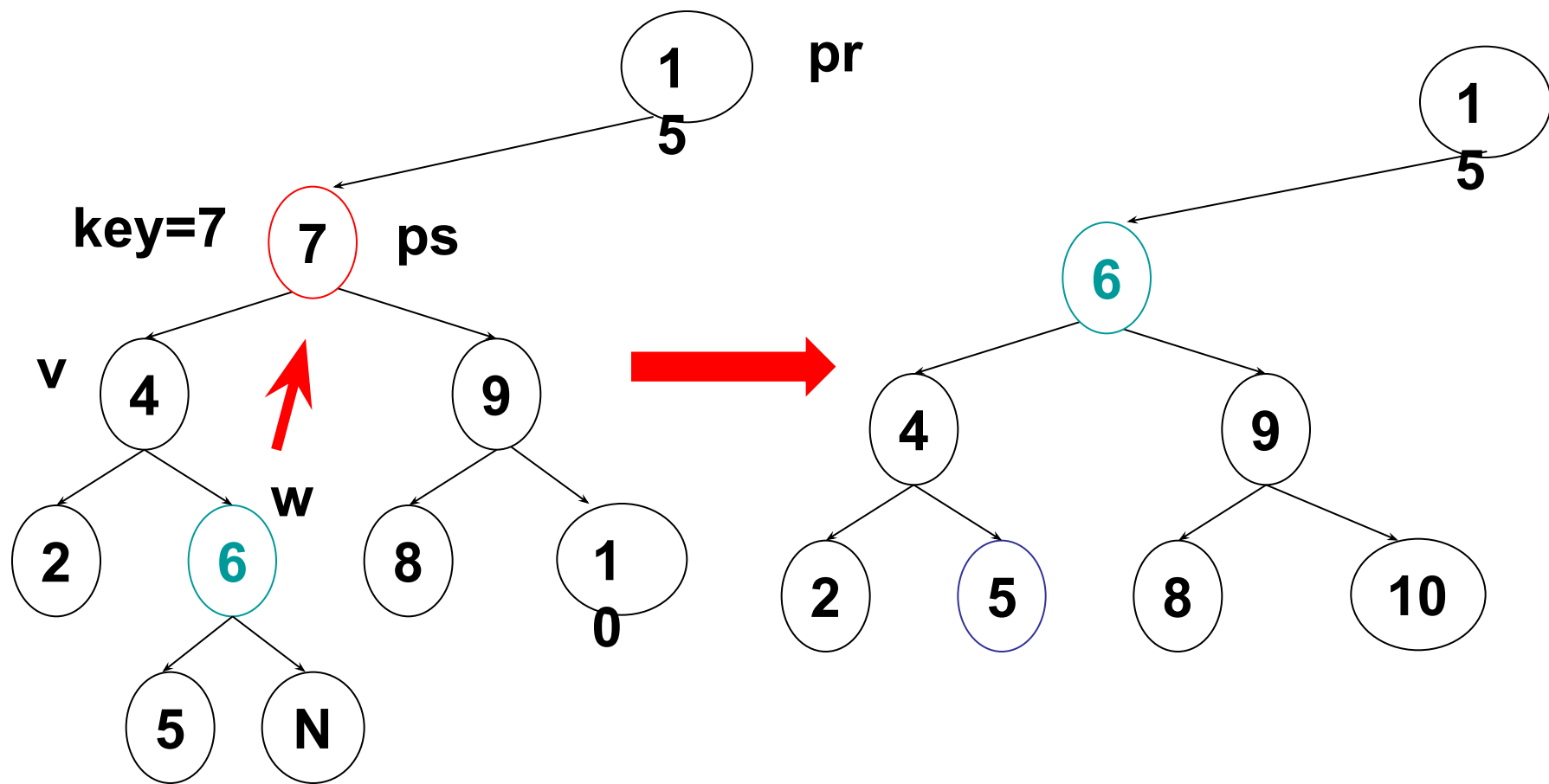
Удаление листа с ключом key:



Удаление узла имеющего одну дочь:



Удаление узла, имеющего двух дочерей



```
ttree *dellist(ttree *proot, int inf)
{
ttree *ps = proot, *pr = proot, *w, *v;
// Поиск удалемого узла
while ((ps != NULL) && (ps->inf != inf))
{
pr = ps;
if (inf < ps->inf) ps = ps->left;
    else ps = ps->right;
}
if (ps == NULL) return proot; // Если узел не найден
```

```
// Если узел не имеет дочерей
```

```
if ((ps->left == NULL) &&  
    (ps->right == NULL))
```

```
{
```

```
    if (ps == pr) // Если это был последний элемент  
    {  
        delete(ps);  
        return NULL;  
    }
```



```
if (pr->left == ps) // Если удаляемый узел слева
    pr->left = NULL;
else // Если удаляемый узел справа
    pr->right = NULL;
delete(ps);
return proot;
}
```

```
// Если узел имеет дочь только справа
```

```
if (ps->left == NULL)
```

```
{
```

```
    if (ps == pr) // Если удаляется корень
```

```
        {
```

```
            ps = ps->right;
```

```
            delete(pr);
```

```
            return ps;
```

```
        }
```

```
if (pr->left == ps) // Если удаляемый узел слева
    pr->left = ps->right;
else // Если удаляемый узел справа
    pr->right = ps->right;
delete(ps);
return proot;
}
```

```
// Если узел имеет дочь только слева
```

```
if (ps->right == NULL)
```

```
{
```

```
    if (ps == pr) // Если удаляется корень
```

```
        {
```

```
            ps = ps->left;
```

```
            delete(pr);
```

```
            return ps;
```

```
        }
```

```
if (pr->left == ps) // Если удаляемый узел слева
    pr->left = ps->left;
else // Если удаляемый узел справа
    pr->right = ps->left;
delete(ps);
return proot;
}
```

// Если узел имеет двух дочерей

w = ps->left;

if (w->right == NULL) // Если максимасальный
// следует за ps

w->right = ps->right;

```
else // Если максимасальный не следует за ps
{
    while (w->right != NULL)
    {
        v = w;
        w = w->right;
    }
    v->right = w->left;
    w->left = ps->left;
    w->right = ps->right;
}
```

```
if (ps == pr) // Если удаляется корень
    {
    delete(ps);
    return w;
    }
```



```
if (pr->left == ps) // Если удаляемый узел слева
    pr->left = w;
else // Если удаляемый узел справа
    pr->right = w;
delete(ps);
return proot;
}
```