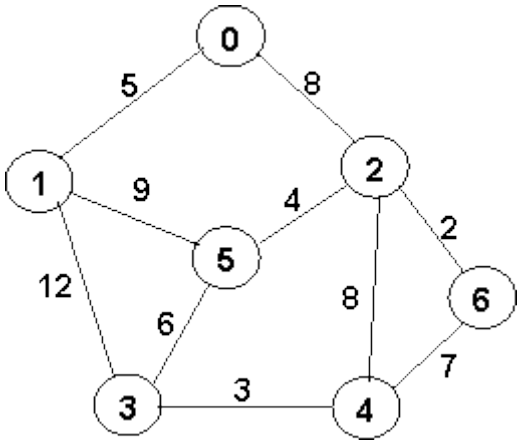
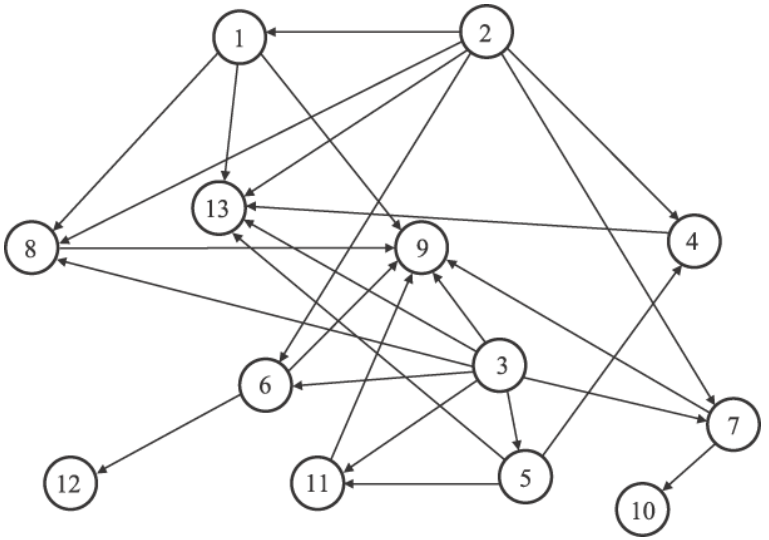
Графы

Граф

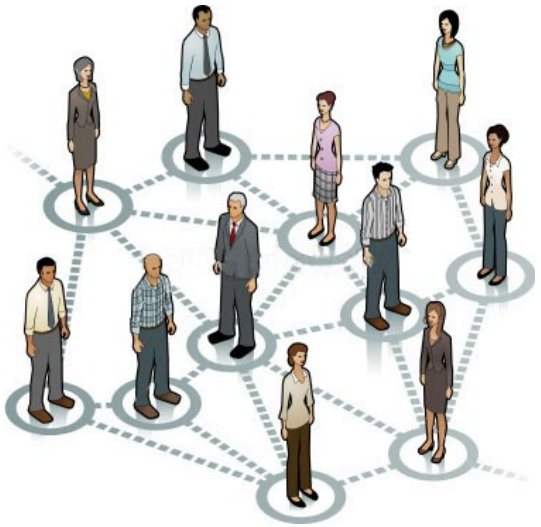


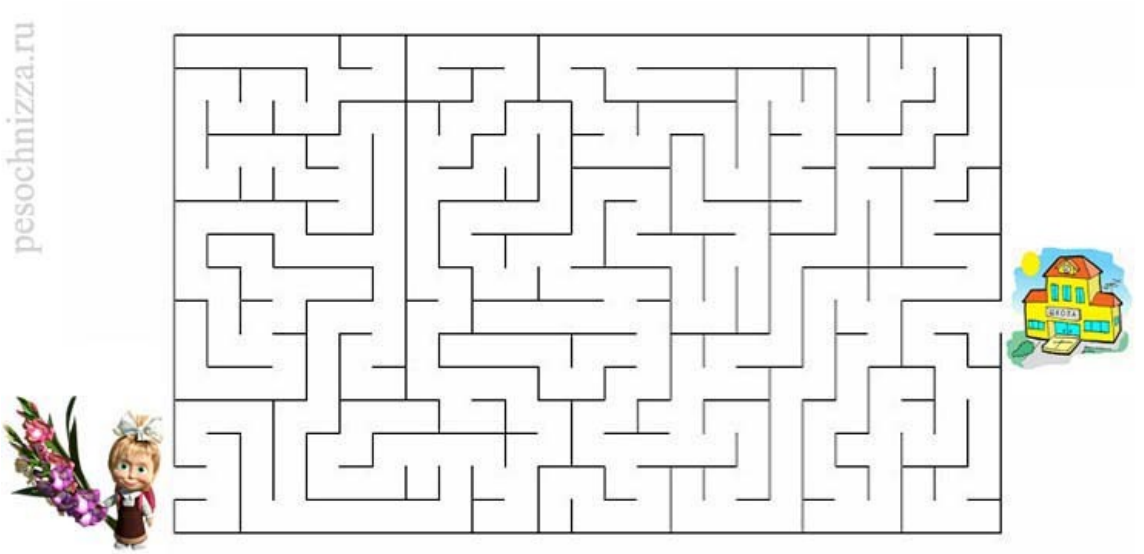
неориентированный

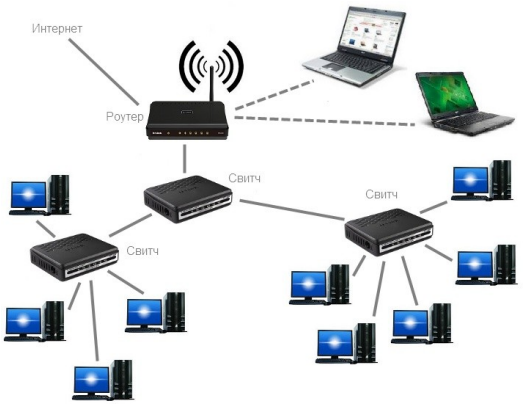


ориентированный

Граф





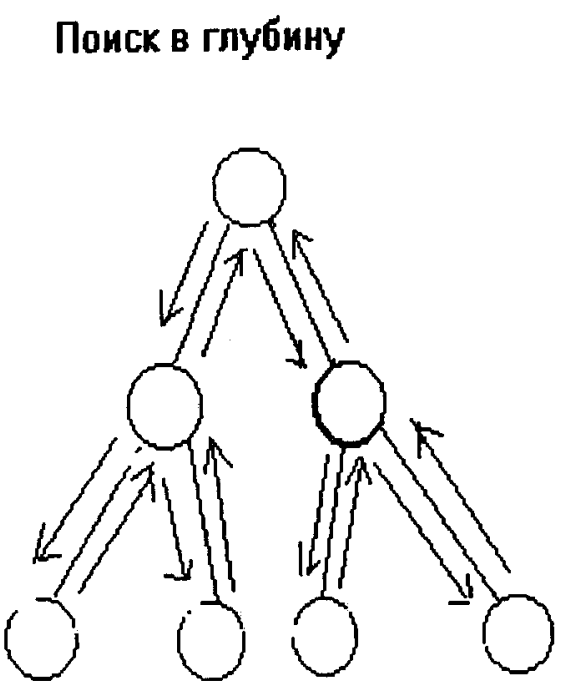


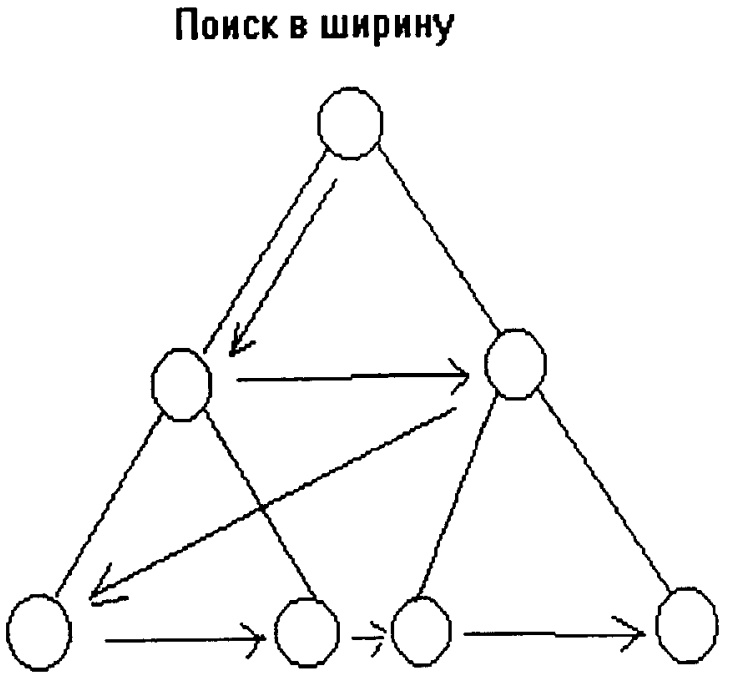
Способы представления графов

- Матрица смежности
- Матрица инцидентности
- Список ребер
- Список списков смежности

Способы обхода графов

- Поиск в глубину
- Поиск в ширину





Обход графа

- Начать обход связного графа можно с любой вершины
- Будем помечать вершины, которые мы посетили
- Переходим только в непросмотренные вершины
- Обход завершается, когда не останется непросмотренных вершин

Поиск в глубину

- Переходим в любую непросмотренную вершину, смежную с текущей
- Если все смежные вершины просмотрены, «возвращаемся на уровень выше».

Поиск в глубину

1..N – пронумеруем вершины графа

mark[N] – массив, хранящий признак «просмотрена/непросмотрена»

```
search(v)
{
    for ( u : смежные с v )
    {
        if ( ! mark[u] )
            search(u);
    }
}
```

Поиск в ширину

- Будем использовать очередь для перехода по уровням
- Извлекаем из очереди вершину
- Добавляем в очередь непросмотренные смежные с ней вершины
- Помечаем их как просмотренные

Поиск в ширину

1..N – пронумеруем вершины графа

mark[N] – массив, хранящий признак «просмотрена/непросмотрена»

Q – очередь, используемая в алгоритме

Q – изначально пустая

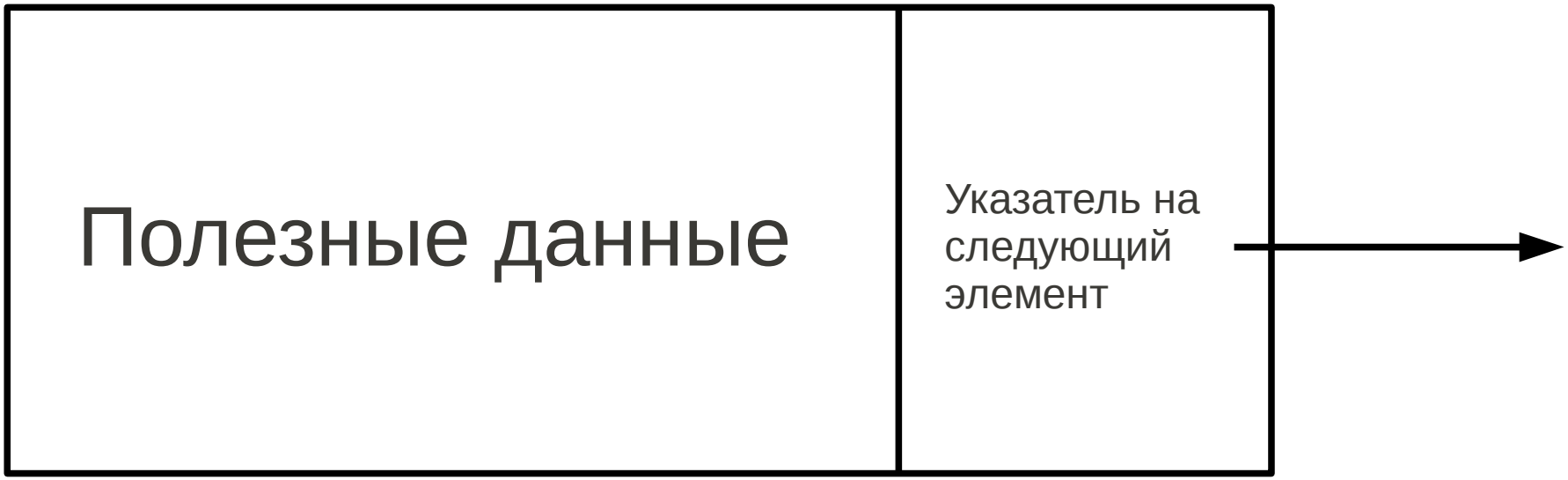
```
search(v_start)
{
    Q.push(v_start);

    while ( ! Q.empty() )
    {
        v = Q.pop();
        for (u : смежные с v)
        {
            if ( mark[u] )
                continue;

            Q.push(u);
            mark[u] = true;
        }
    }
}
```

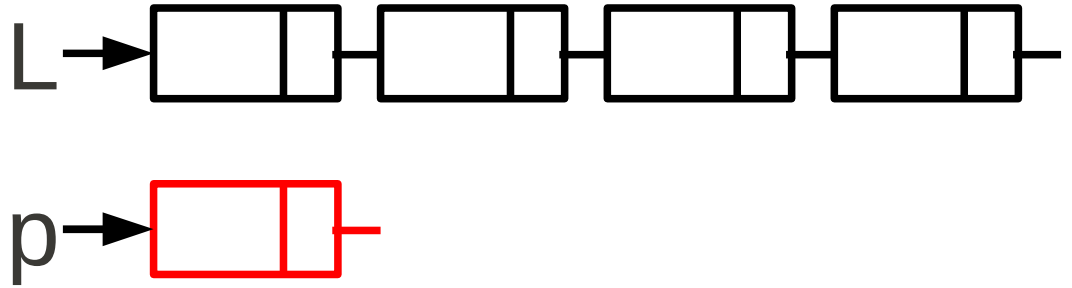
Связный список

Элемент списка

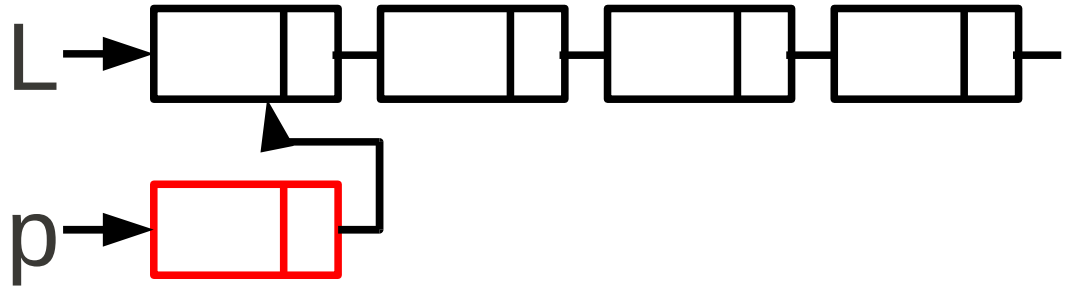


Вставка в начало списка

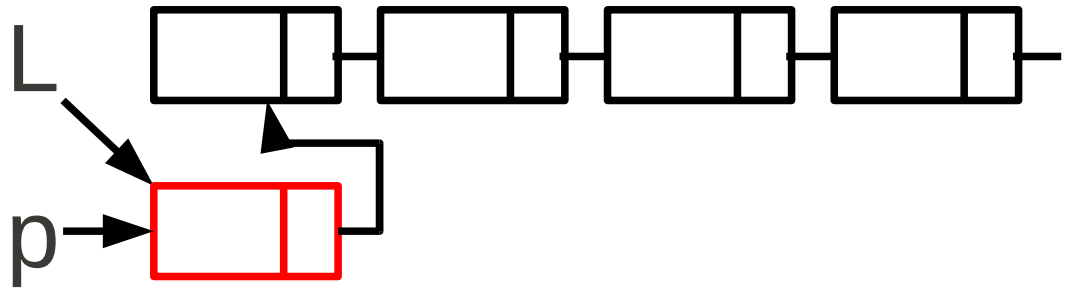
I.



II.

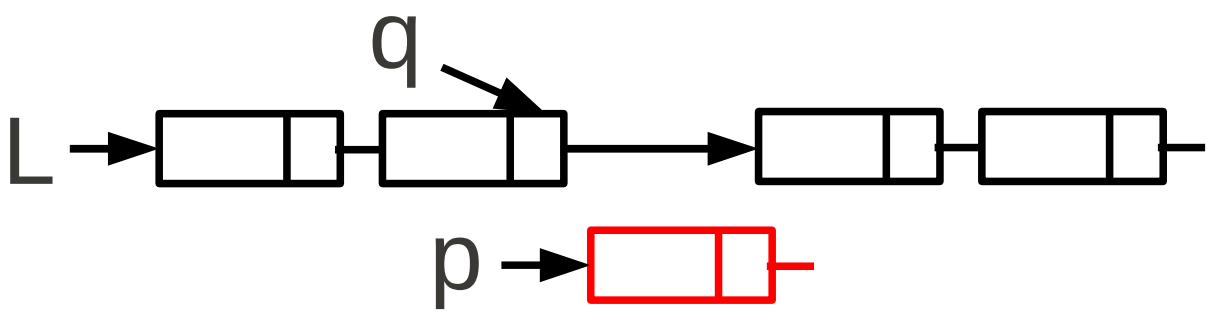


III.

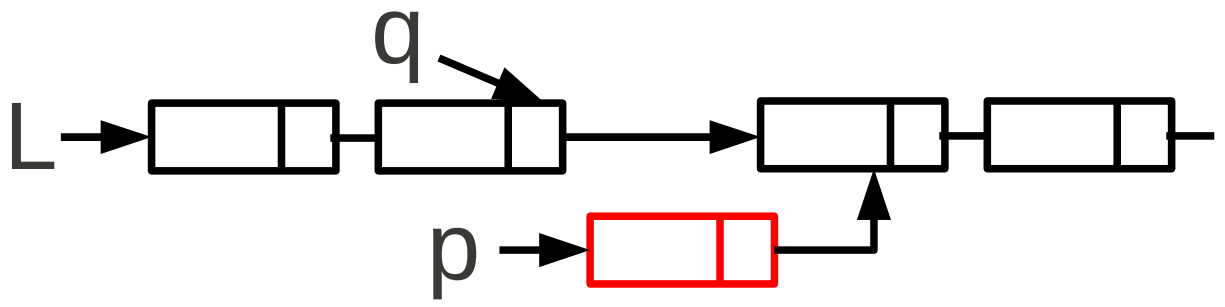


Вставка в середину и конец списка

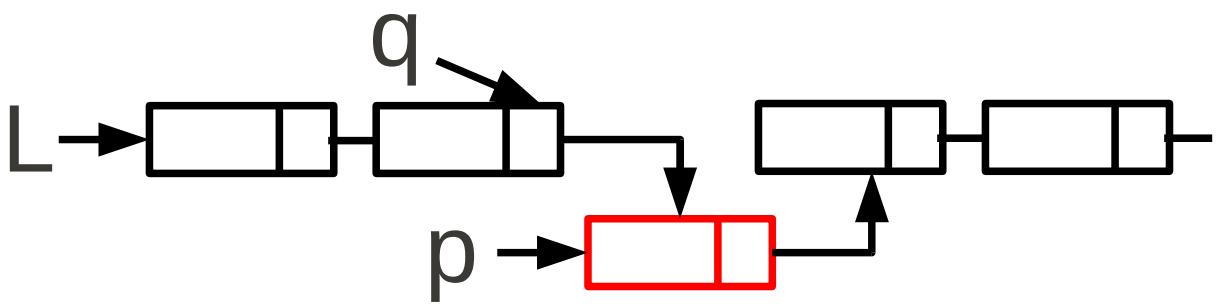
I.



II.

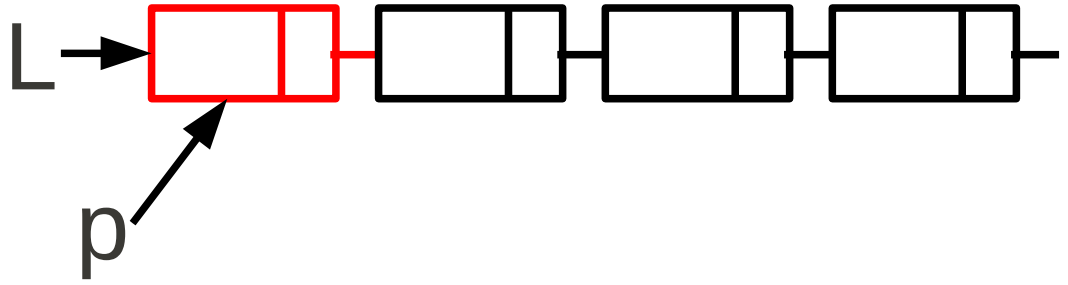


III.

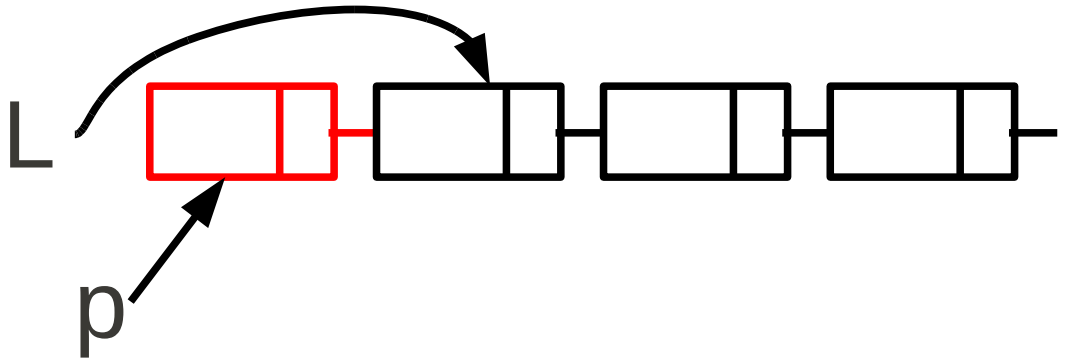


Удаление из начала списка

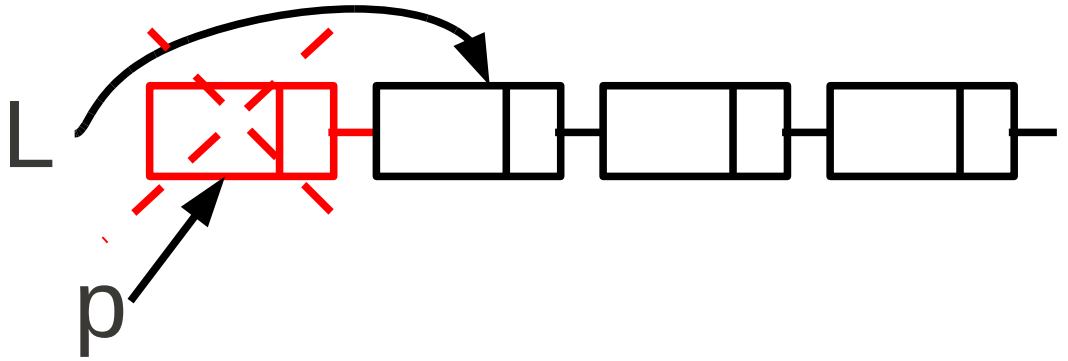
I.



II.

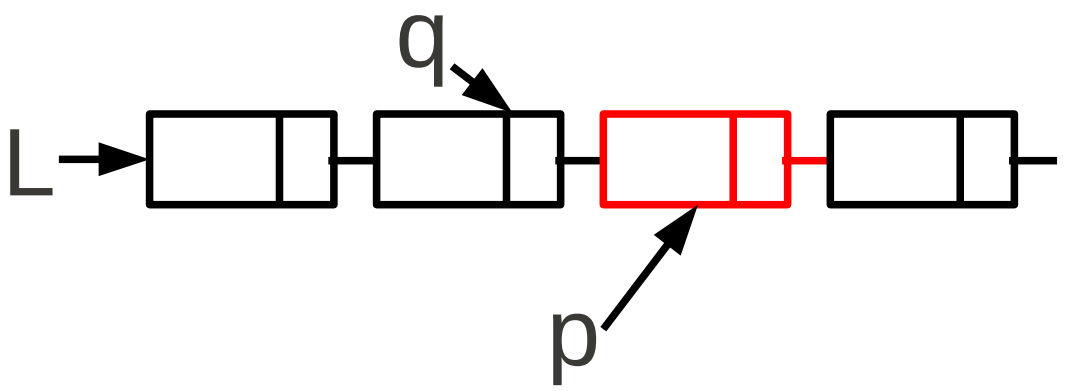


III.

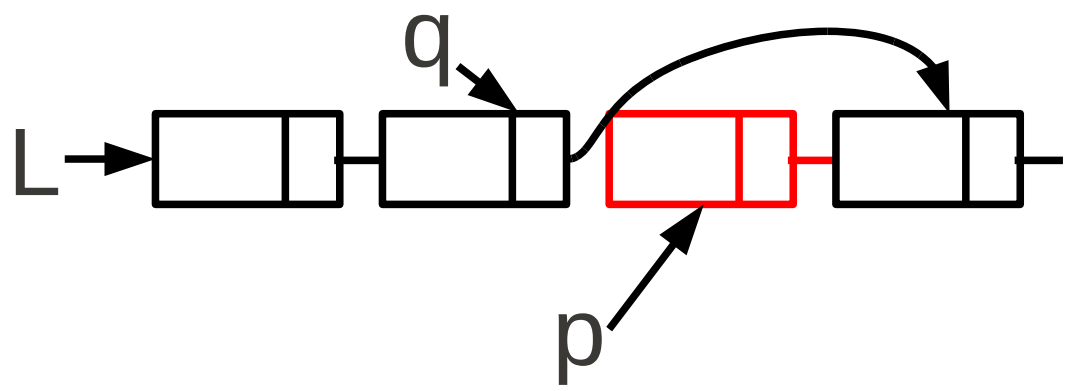


Удаление из середины списка

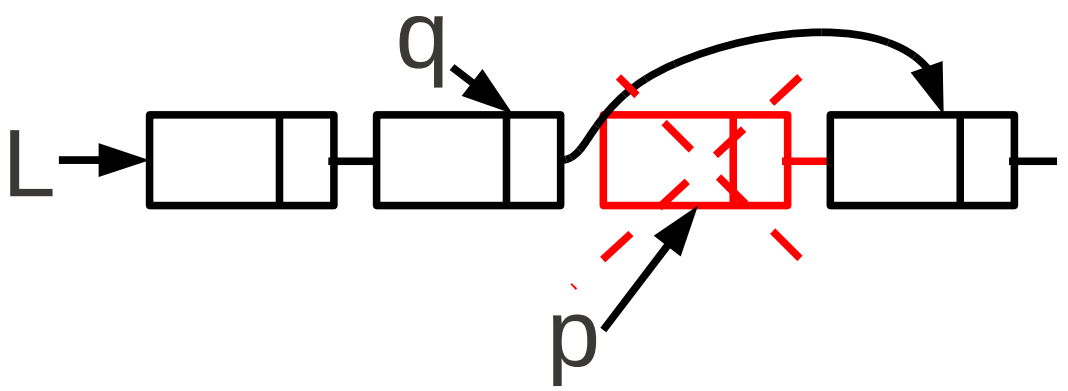
I.



II.

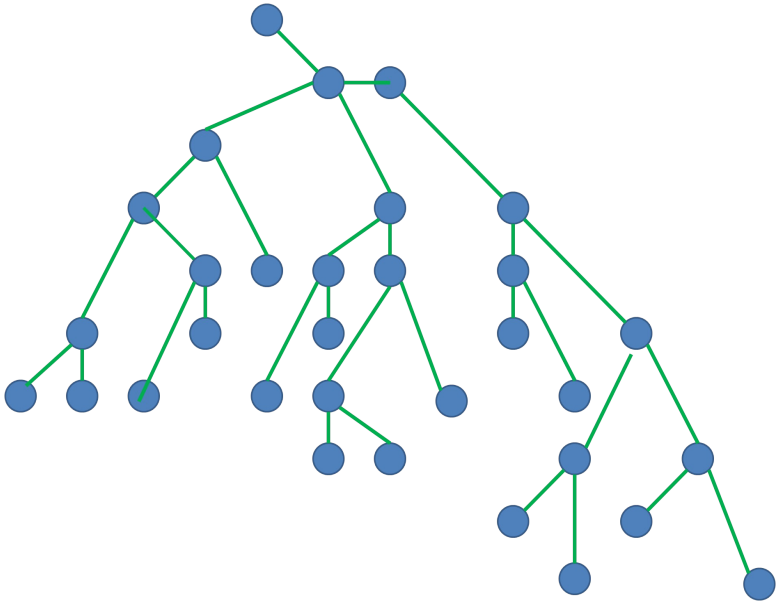


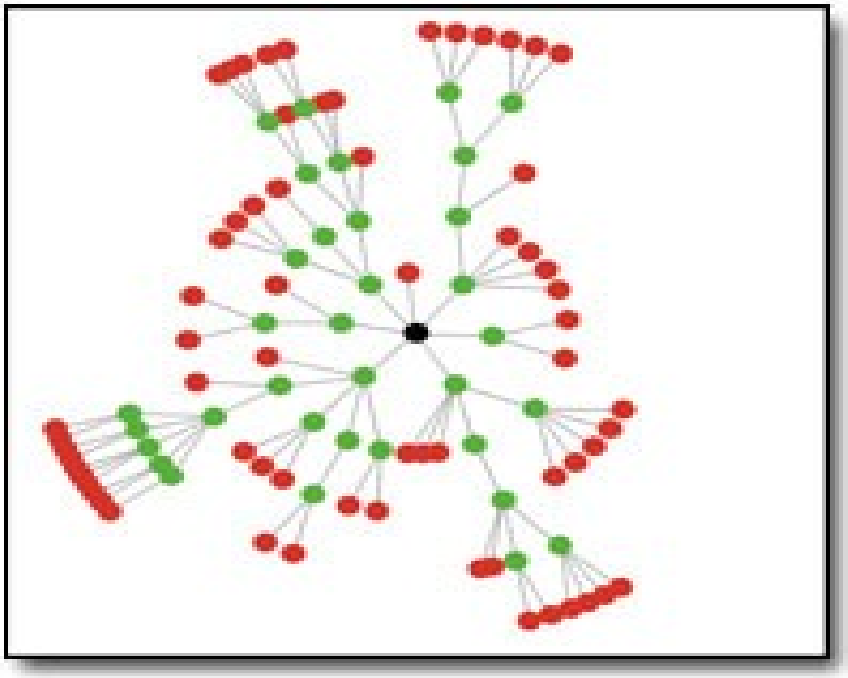
III.



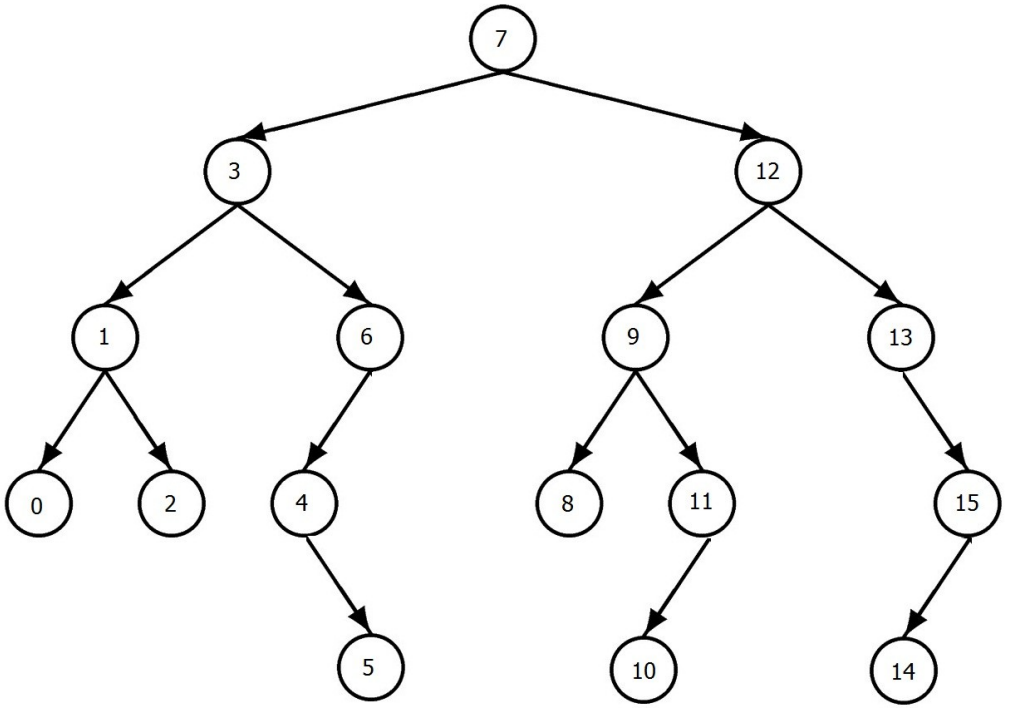
Дерево

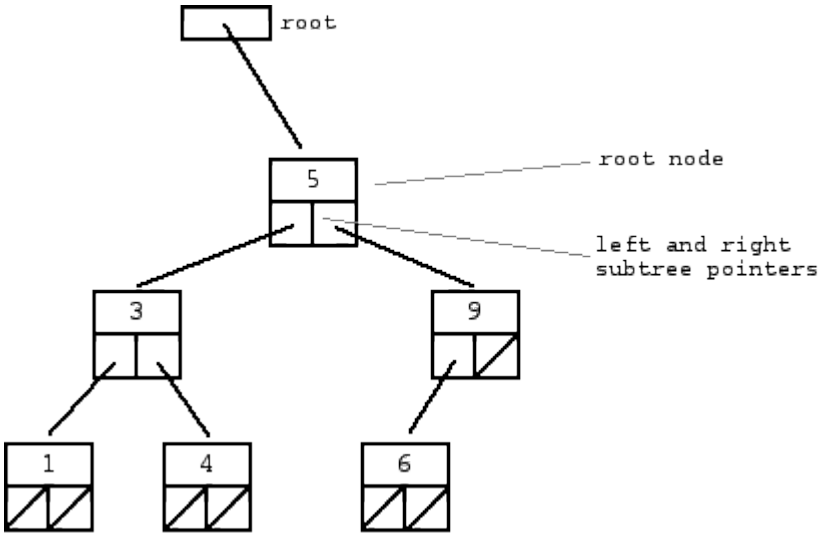
Дерево





Бинарное дерево поиска





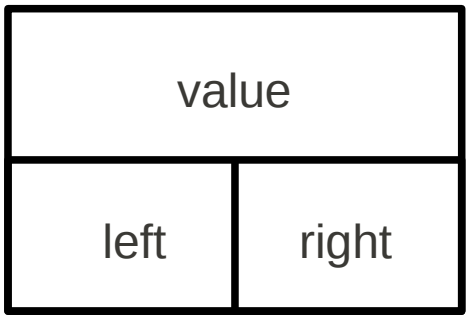
Поиск элемента в бинарном дереве

```
struct Node
{
    value;
    Node left;
    Node right;
};

search(root, element)
{
    if (root->value == element)
        return root;

    if (root->value > element)
        return search(root->right, element);

    if (root->value < element)
        return search(root->left, element);
}
```



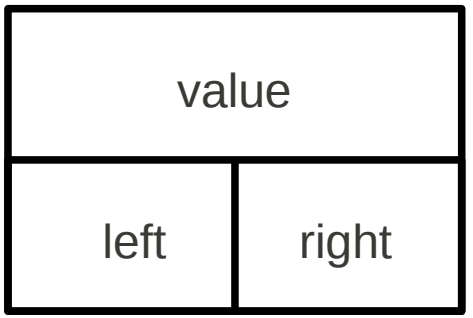
Вставка элемента в бинарное дерево

```
insert(root, element)
{
    if (root == null)
        root = Node(element)

    if (root->value == element)
        return;

    if (root->value > element)
        insert(root->right, element);

    if (root->value < element)
        insert(root->left, element);
}
```



Сложность процедуры поиска и вставки

- Для того, чтобы убедиться, что элемента нет в списке, нужно просмотреть весь список. (Говорят, что сложность алгоритма поиска O(N))
- Для того, чтобы убедиться, что элемента нет в дереве поиска, не обязательно просматривать все дерево.
- Если дерево сбалансировано, то сложность поиска будет порядка $O(\log_2(N))$