education

Списки

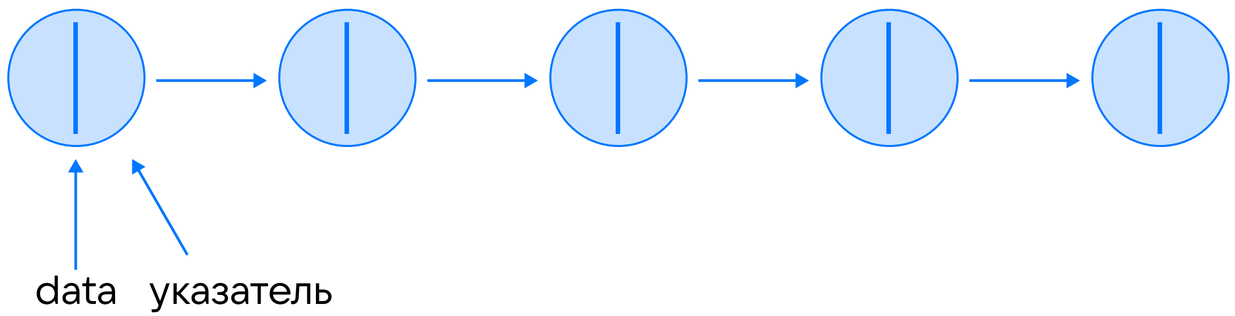
Алгоритмы и структуры данных
Илья Почуев

Что будет на занятии

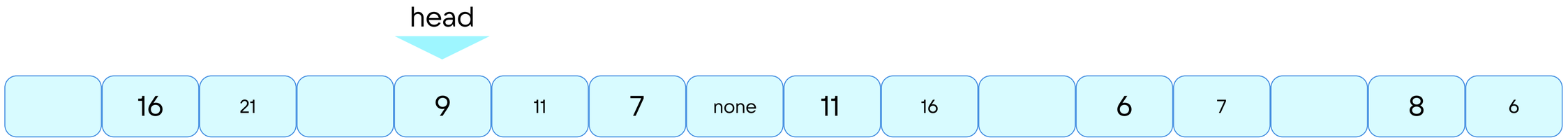
- Что такое список
- Односвязный список, представление в памяти
- Операции над списком и их сложность
- Двусвязный список
- Разворот списка
- Проверка списка на цикличность
- Задача на поиск середины списка

Виды связных списков

- Однонаправленный (односвязный)
- Двунаправленный (двусвязный)

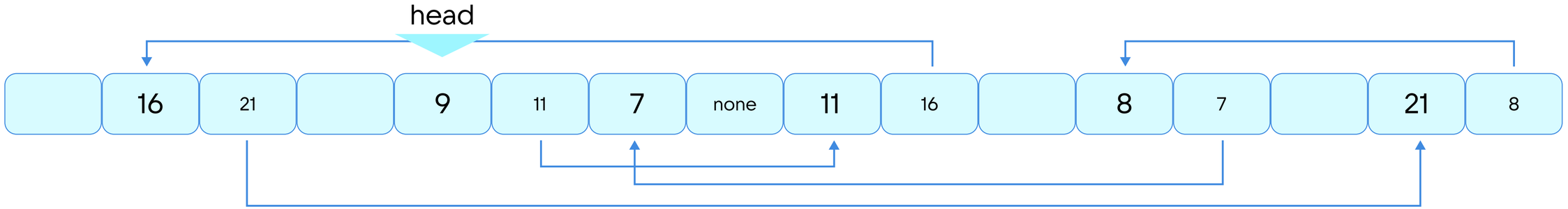


Структура однонаправленного списка



- **Узел –** основная часть списка, обычно определяющаяся классом или структурой.
- **head –** голова списка, его первый элемент, а точнее первый узел
- Список устроен таким образом, что каждый узел знает, где хранится последующий: head знает, где хранится второй, второй – где хранится третий, и так далее.
- Последний элемент вместо указателя на следующий хранит в себе null (none, nil – в зависимости от языка).

Важно понимать: в списке нет произвольного доступа по индексу к узлам, как в массивах, а это означает, что, чтобы найти элемент, надо пройти по всему списку.



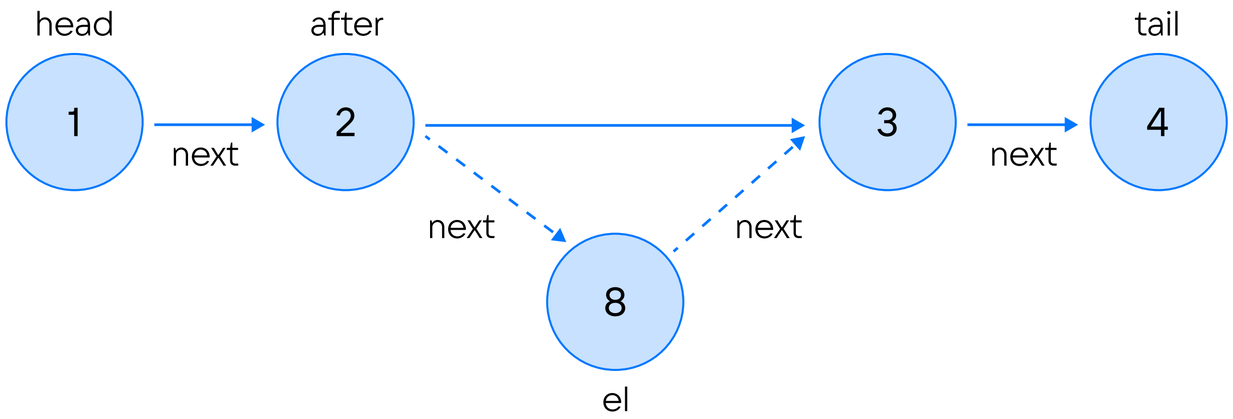
Преимущества списка перед массивом

- Нет необходимости располагать элементы последовательно
- Список аллоцирует память ровно столько, сколько элементов в себе содержит, плюс указатели на следующие элементы

Расположение в памяти
В отличие от массива нет необходимости хранить данные последовательно

Ограничения списка

- Каждый узел хранит в себе, помимо собственных данных, ссылку на следующий элемент
- Для вставки в любую точку списка необходимо изменить ссылки у рядом стоящих элементов



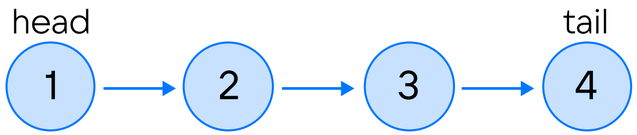
education

Абстрактное представление в коде

Структура узла в коде

- **Узел или Node —** основная часть списка, обычно определяющаяся классом или структурой
- **data —** полезная информация, которую несёт в себе узел
- **next —** указатель на следующий элемент

```
//структура каждого узла
Node {
   data int
   next Node
}
```



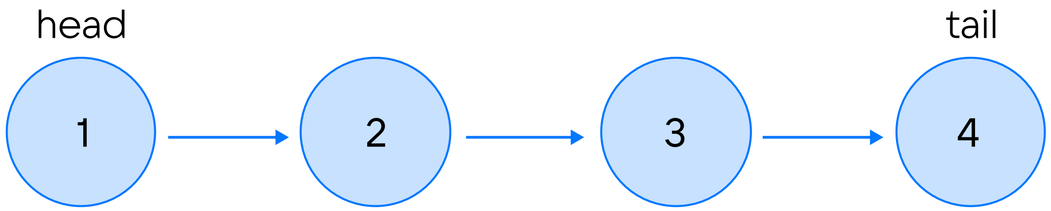
Структура списка в коде

Структуру или класс списка можно представить в виде head, size и иногда tail.

- **head** — голова списка
- **size** — размер списка
- **tail** — указатель на последний узел списка

```
//структура списка
LinkedList {
    head Node
    tail Node
    size int
}
```

В простейших однонаправленных списках на tail обычно не содержится отдельного указателя, но иногда он может быть полезен для оптимизации некоторых операций

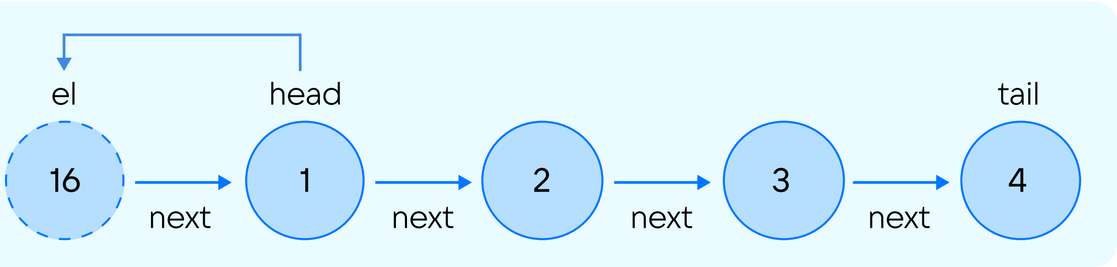


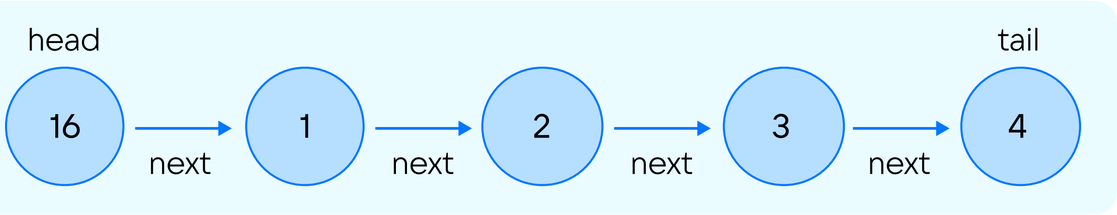
education

Добавление элемента в список

Вставка в начало списка

- Самая простая операция по добавлению элемента
- Для реализации необходимо только переопределить head
- Три действия за константное время приводят эту операцию к O(1)



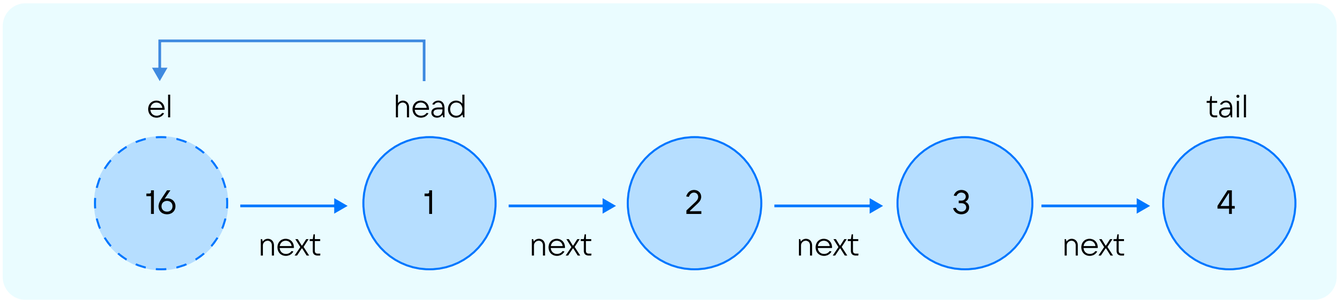


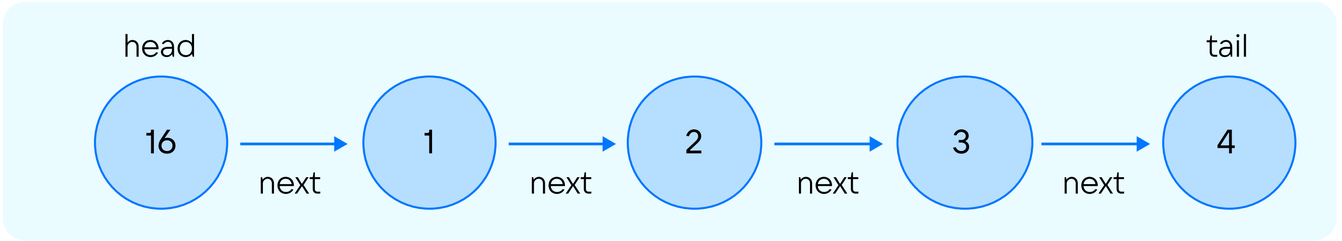
Вставка в начало списка

```
function addNewHead(linkedList, el) {
  node = {
      data: el,
      next: null
  } ;

  // если список был пуст
  if (head == null) {
     linkedList.tail = node;
  } else {
    //прежний head сдвигаем на
    //один узел назад
    //созданный узел в качестве
    //next ссылается на head
    node.next = head;
  }

  // устанавливаем новый узел в
  // качестве головы списка
  linkedList.head = node;
}
```

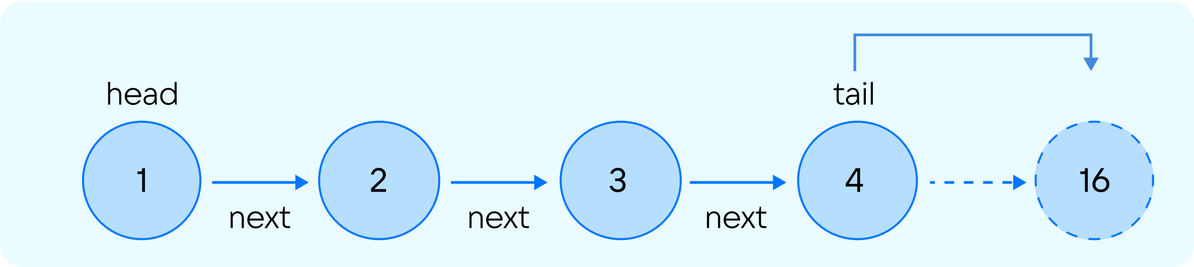


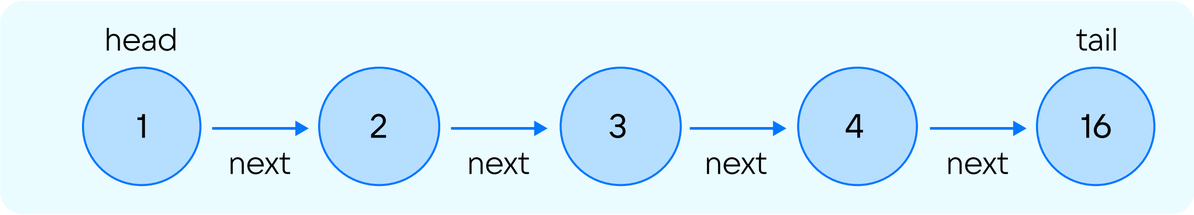


Вставка в конец списка

Различают два варианта вставки в конец списка:

- храним указатель на tail
- не храним указатель на tail





Вставка в конец списка с указателем на tail

- Создаём новый узел
- Если список ранее был пуст, то новый узел становится одновременно и head, и tail
- Записываем новый узел в качестве tail

```
function addNewTail(linkedList, el) {
    //создаем новый узел
    node = {
        data: el,
        next: null
    };
    // если список был пуст
     if (linkedList.tail == null) {
        linkedList.head = node;
     } else    {
       linkedList.tail.next = node;
     }
     // записываем новый узел в качестве tail
     linkedList.tail = node;
}
```

Вставка в конец списка без указателя на tail

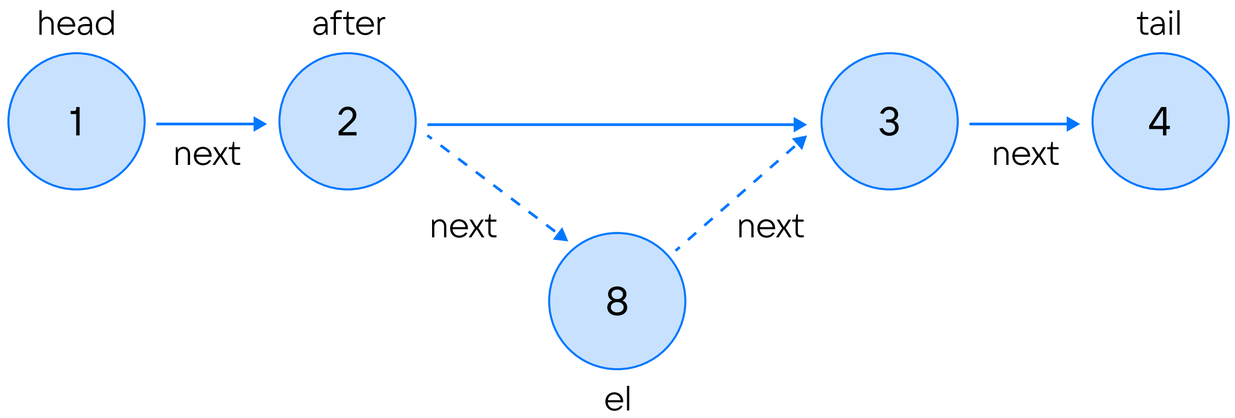
- Необходимо пройти в цикле по списку, чтобы найти последний элемент
- Это приводит к сложности O(n)
- Последний узел находится после цикла в currentNode
- В качестве поля next запишем новый узел

```
function addNewTail(linkedList, el) {
    node = {
        data: el,
        next: null
    };

    currentNode = linkedList.head;
    while (currentNode.next != null) {
        currentNode = currentNode.next;
    }
    currentNode.next = node;
}
```

Вставка в середину

Чтобы осуществить вставку в середину списка, необходимо вначале найти узел, после которого нужно добавить новый элемент



Вставка в середину

```
function insert(linkedList, after, el) {
  // находим after
  search = linkedList.head;
  while (search != null) {
      if (search.data == after) {
        break;
      }
      // переходим к следующему элементу
      search = search.next;
   }
   // если мы нашли элемент after
   if (search !== null) {
     node = {
        data: el,
        next: null
     };
     // если элемент, после которого нам надо произвести
вставку
    // является последним, то теперь новый элемент и есть
tail
    if (search == linkedList.tail) {
       linkedList.tail = node;
    }
    // узел, который был для search "следующим"
   // теперь является значением next для нового узла
   node.next = search.next;
   search.next = node;
  }
}
```

- Для этой операции необходимо знать узел, после которого совершим вставку
- Ищем элемент after
- Если элемент, после которого нам надо произвести вставку, является последним, то теперь новый элемент и есть tail
- Меняем необходимые указатели

Поиск элемента

В цикле выполняем итерации по списку, пока не найдём узел, у которого data равна исходному элементу

```
function search(head, el) {
    current = head;

    while (current != null) {
        if (current.data == el) {
            // нашли элемент, возвращаем его узел
            return current;
        }
        // переходим к следующему узлу
        current = current.next;
    }

    return null; // если элемент не найден
}
```

Сложность

- Вставка в начало O(1)
- Вставка в конец списка O(1) при наличии tail
- Вставка в конец списка O(n) при отсутствии tail
- Вставка в середину O(n)
- Поиск элемента O(n)
- Удаление из середины O(n)

Массив или список?

“Right” branch of “if”	1–2			0.3 ns
L1 read	3–4			0.5 ns
L2 read		10–12		5–7 ns
“Wrong” branch of “if”		10–20		5–10 ns
L3 read		30–70		> 20 ns
Allocation + deallocation pair (small objects)			200–500	> 500 ns

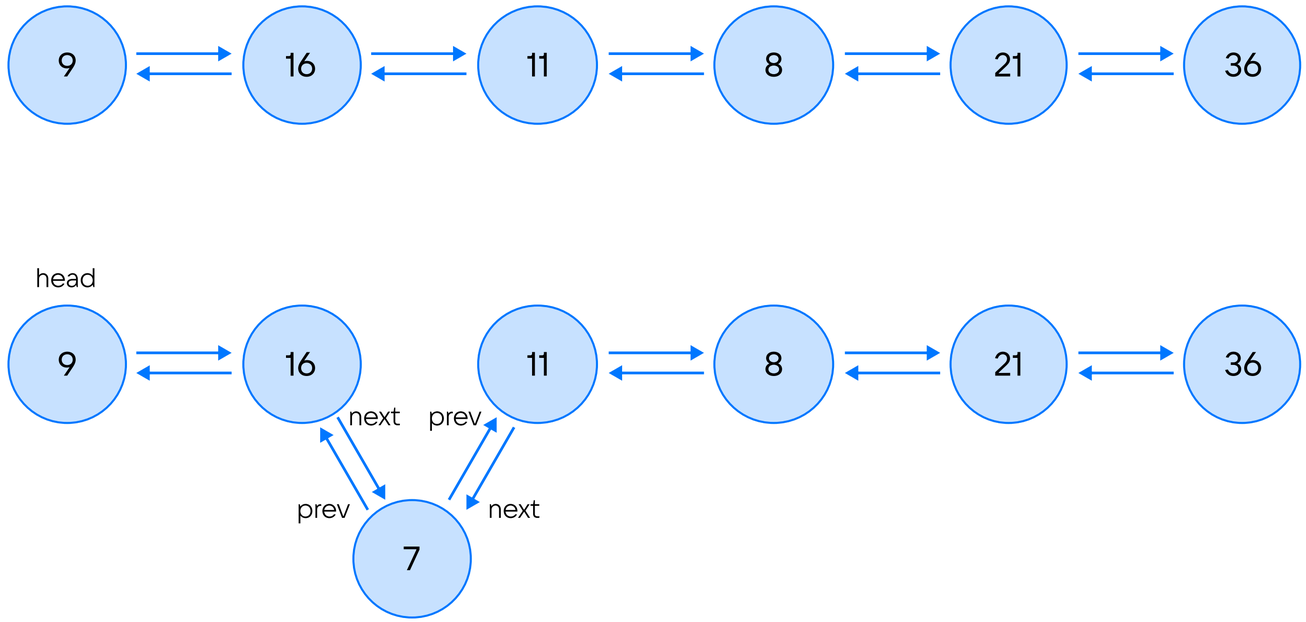
Массив, если он может поместиться в кеше, за счёт последовательного расположения в памяти будет читаться из кеша

education

Двусвязный список

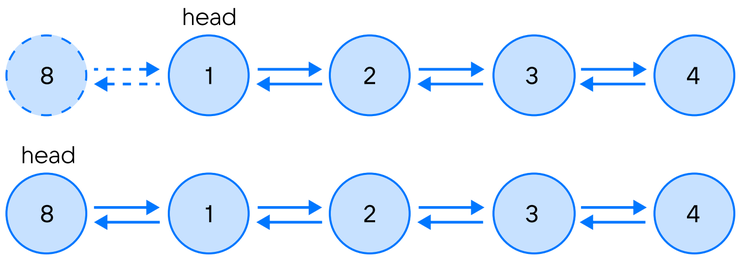
Двусвязный список

- Каждый узел, кроме первого и последнего, хранит указатели на следующий и на предыдущий узел
- Занимает больше памяти в сравнении с односвязным списком
- Можно производить вставку не только после, но и перед элементом
- При вставке или выборке необходимо обновлять два указателя: на следующий и на предыдущий узел



Вставка в начало

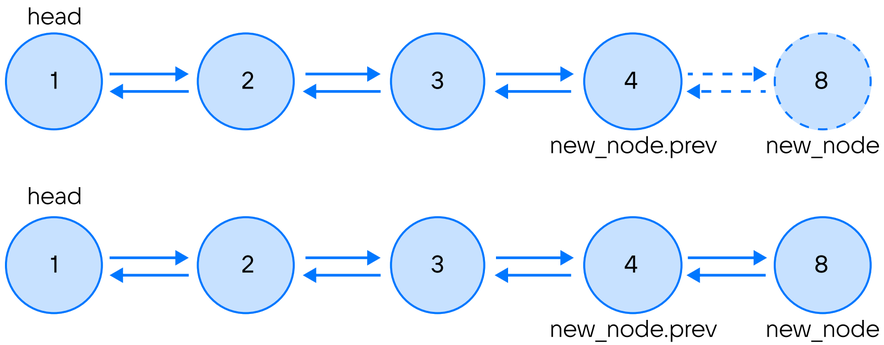
```
function append_front(linkedList, data) {
  // создаем новый узел и добавляем в него новое
значение data
  new_node = {
        data: data,
        next: null,
        prev: null
   };
   if (linkedList.head === null) {
      // если ранее список был пуст,
      // значит первый элемент и будет
      // являться головой (head)
      linkedList.head = new_node;
      return;
  }
  // если список не пуст, то устанавливаем head
  // в качестве параметра next для нового узла
  new_node.next = linkedList.head;
  //переписываем указатель prev
  linkedList.head.prev = new_node;
  // записываем в head новый узел
  linkedList.head = new_node;
}
```



- При вставке необходимо следить за указателем на предыдущий элемент
- append_front – создаём новый узел и добавляем в него новое значение data
- если ранее список был пуст, значит, первый элемент и будет являться головой (head)
- если список не пуст, то устанавливаем head в качестве параметра next для нового узла
- При этом старый head в качестве предыдущего теперь ссылается на новый узел
- Записываем в head новый узел

Вставка в конец

```
function append_back(linkedList, data) {
    // создаем новый узел и добавляем в него новое
значение data
    new_node = {
        data: data,
        next: null,
        prev: null
    };
    if (linkedList.head === null) {
        linkedList.head = new_node;
        return;
    }
    // пройдемся по списку до конца, начиная с головы
    cur_node = linkedList.head;
    while (cur_node.next !== null) {
        cur_node = cur_node.next;
    }
    // элементу, который был последним,
    // в поле next записываем новый
    cur_node.next = new_node;
    // в новый элемент, в поле prev записываем
    // узел, который до вставки был последним
    new_node.prev = cur_node;
}
```



append_back – повторяем первые два пункта из append_front

Идём по списку до конца, начиная с головы

Теперь в поле next у последнего элемента находится ссылка на новый

В новый элемент, в поле prev, записываем узел, который до вставки был последним

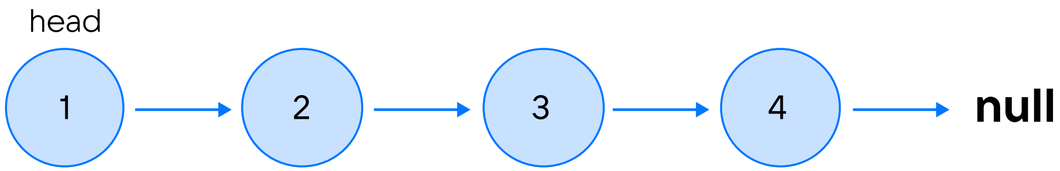
education

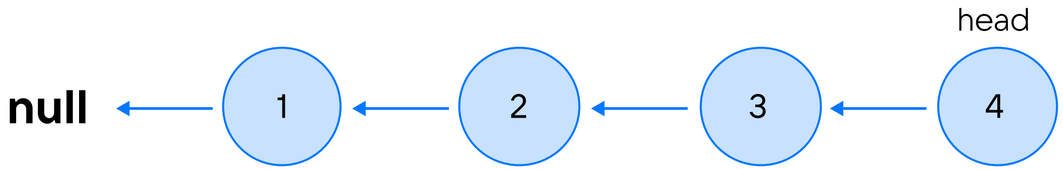
Задачи
на связный список

Развернуть односвязный список

Необходимо написать функцию, которая принимает на вход односвязный список и разворачивает его

Reverse Linked List



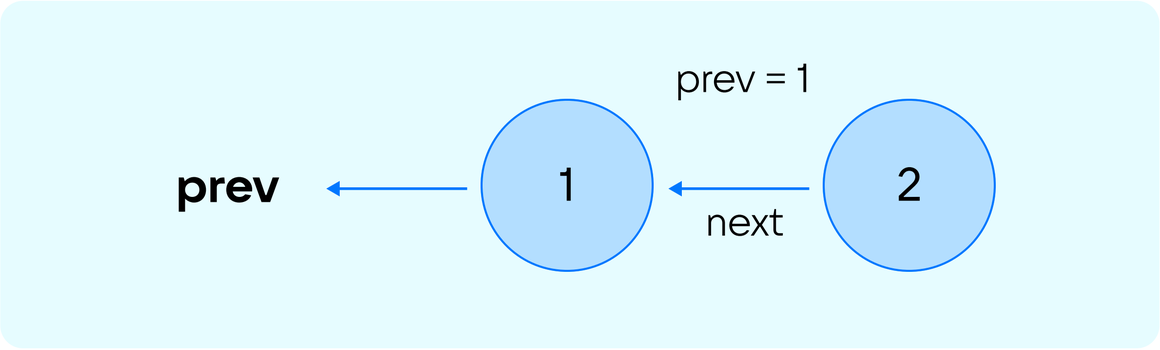


Reverse Linked List

- На первой итерации узел со значением “единица” должен ссылаться на null
- Для этого заведём переменную **prev** и запишем туда null
- При этом узел prev должен стать next для следующего узла (второго). То есть на следующей итерации двойка должна будет ссылаться на единицу
- Для этого в конце первой итерации двигаем prev на узел 1

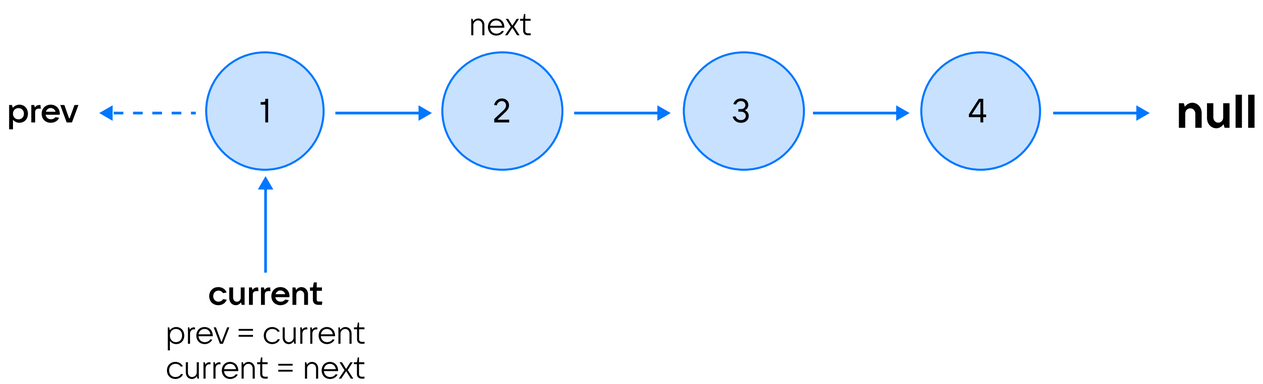
Reverse Linked List

- На второй итерации узел со значением два должен ссылаться в качестве next на 1
- При этом узел 2 должен стать next для следующего узла после двойки
- В конце второй итерации, так же как и в первой, двигаем prev, чтобы он стал next для узла 3



Reverse Linked List

Проходим последовательно в цикле по всему списку.
На каждой итерации у узлов меняем указатели



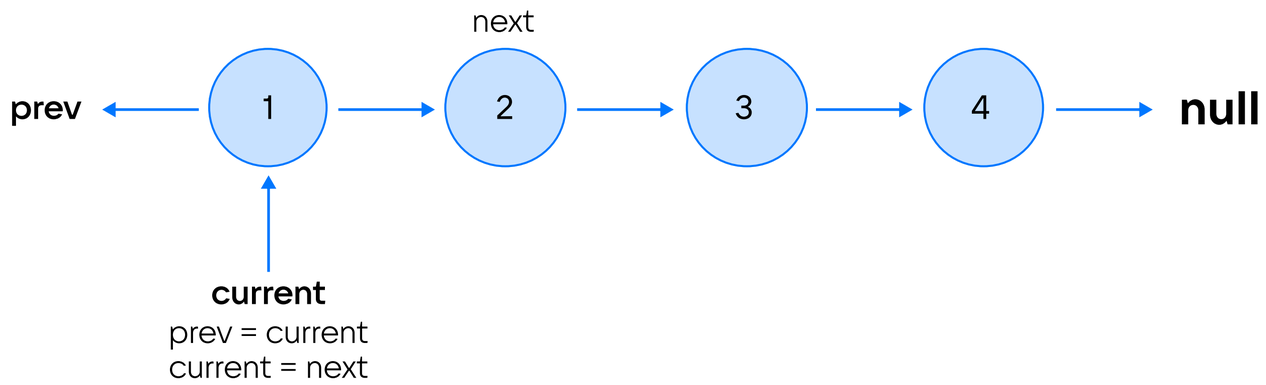

```
function reverseLinkedList(head) {
    prev = null
    current = head

    while (current != null) {
        ...
    }

    return head
}
```

Reverse Linked List

- На первой итерации current указывает на первый узел
- Первый узел в качестве параметра next получает значение из переменной prev, то есть null




```
function reverseLinkedList(head) {
        prev = null
        current = head

        while (current != null) {
            //на первой итерации next=2
            next = current.next
            //первый узел в качестве
            //параметра next получает null
            current.next = prev
            ...
        }
}
```

Reverse Linked List

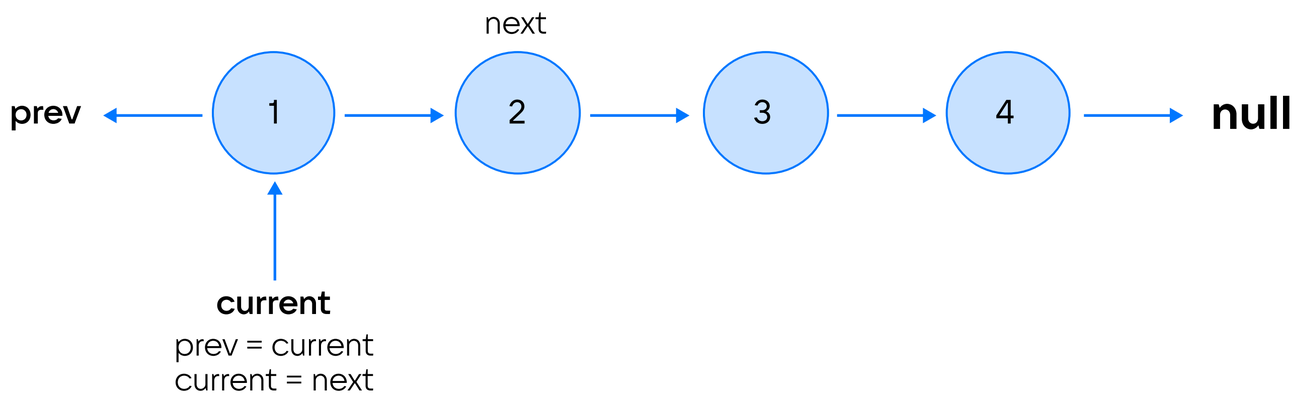
```
function reverseLinkedList(head) {
    prev = null
    current = head

    while (current != null) {
        next = current.next
        current.next = prev
        //двигаем значение prev на единицу
        //чтобы на следующей итерации узел 2
        //ссылался на узел 1 в качестве next
        prev = current
        //двигаем current с узла 1 на узел 2
        current = next
    }

    head = prev
    return head
}
```

Двигаем значение prev на первый узел, а current в конце первой итерации уже будет находиться на узле 2.

Повторим процедуру

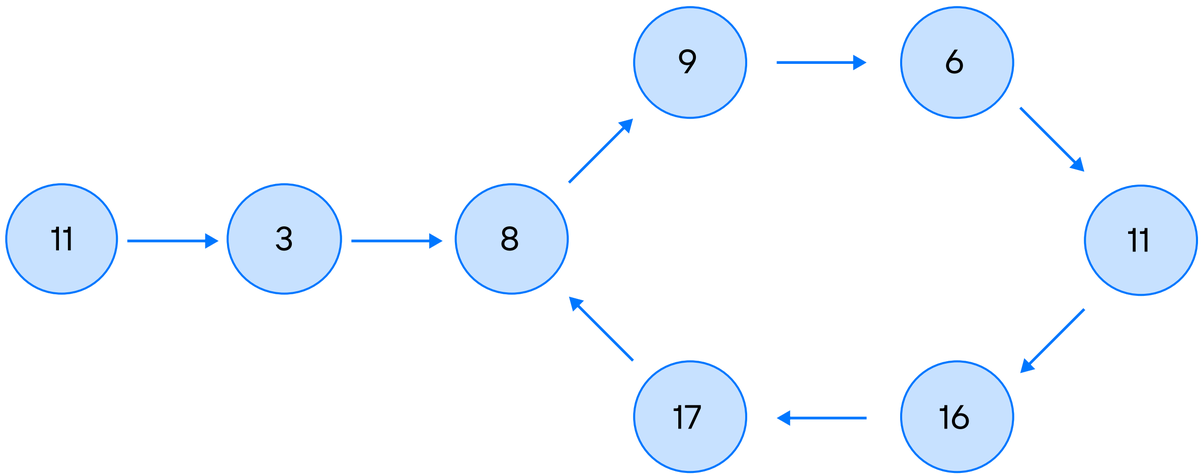


Проверить, является ли список циклическим

Дан односвязный список. Необходимо проверить, является ли этот список циклическим

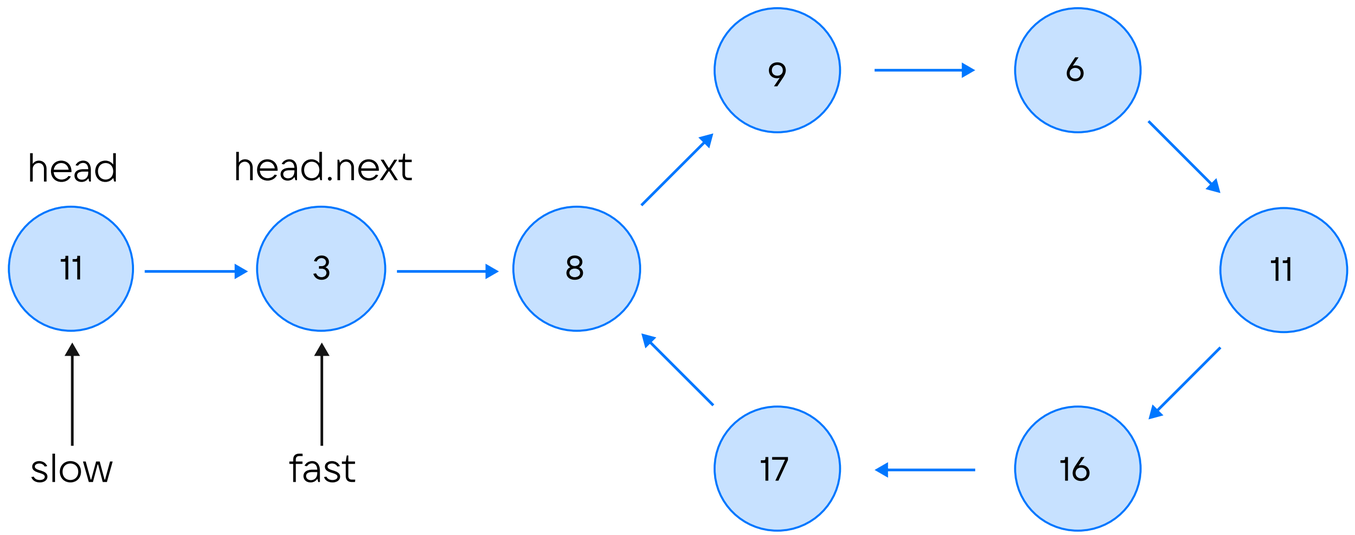
Циклическим (кольцевым) списком называется список, у которого последний узел ссылается на один из предыдущих узлов

Циклический список



Быстрый и медленный указатель

- Решаем задачу через два указателя
- Один указатель медленный, другой быстрый
- Медленный начинается с головы
- Быстрый — со второго элемента
- Медленный «шагает» узел за узлом
- Быстрый перескакивает через узел



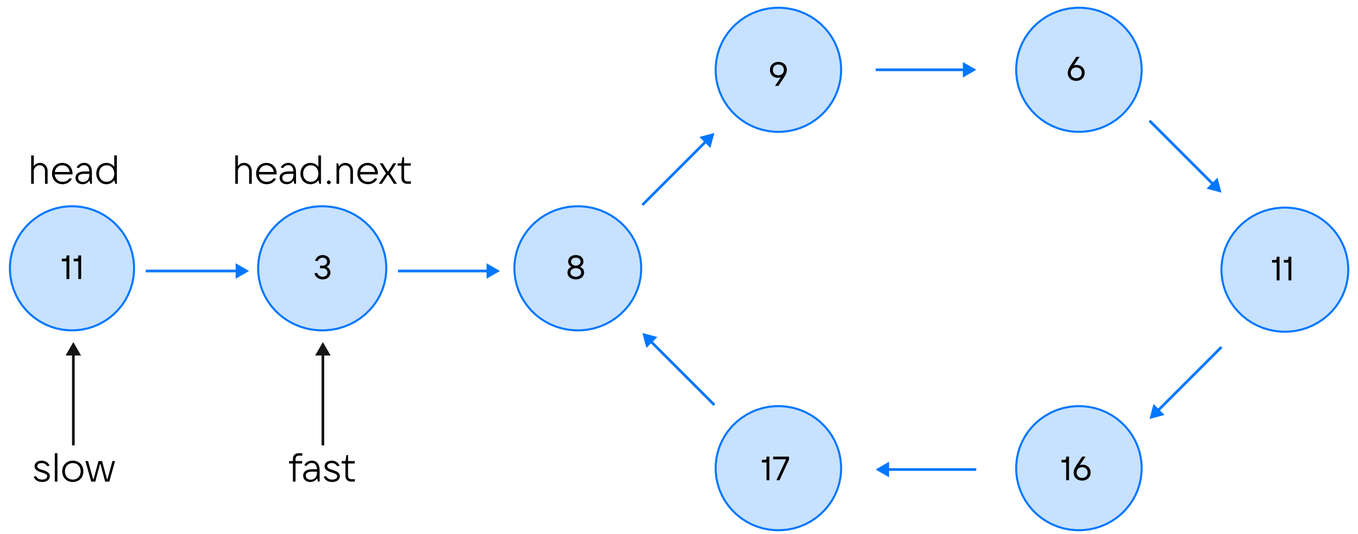
Является ли список циклическим

```
function hasCycle(head) {
    if head == null || head.next == null {
        return false
    }

    slow = head
    fast = head.next

    while slow != fast {
        if fast == null || fast.next == null {
            return false
        }
        slow = slow.next
        fast = fast.next.next
    }
    return true
}
```

Проходимся в цикле по нашему списку. Если быстрый указатель дошёл до null, значит, список не циклический. Если медленный указатель в какой-то момент стал равен быстрому, значит, мы нашли цикл

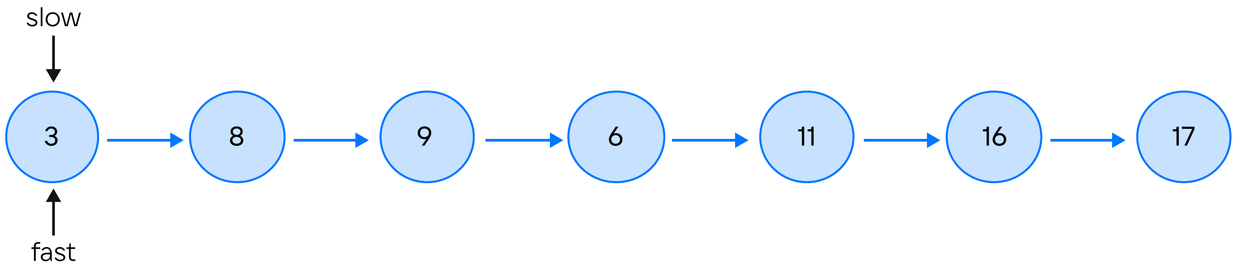


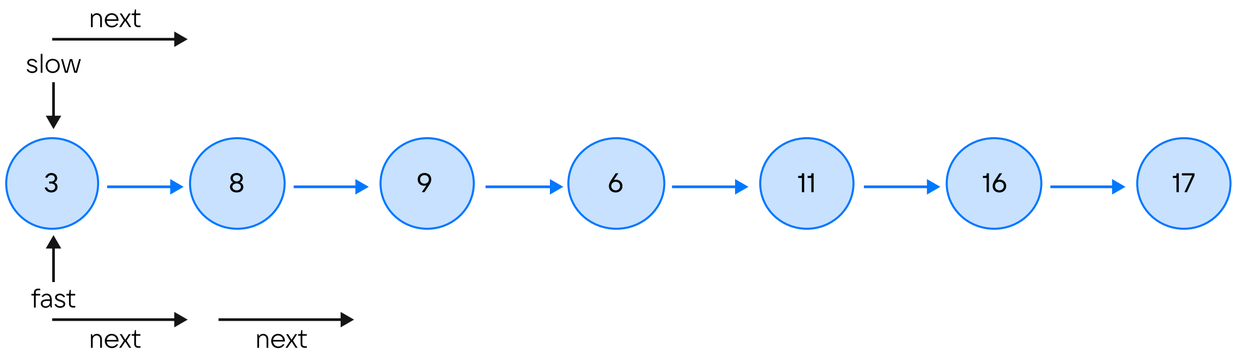
Найти середину списка

Дан связный список. Необходимо найти середину списка. Сделать это необходимо за O(n) без дополнительных аллокаций

Середина связного списка

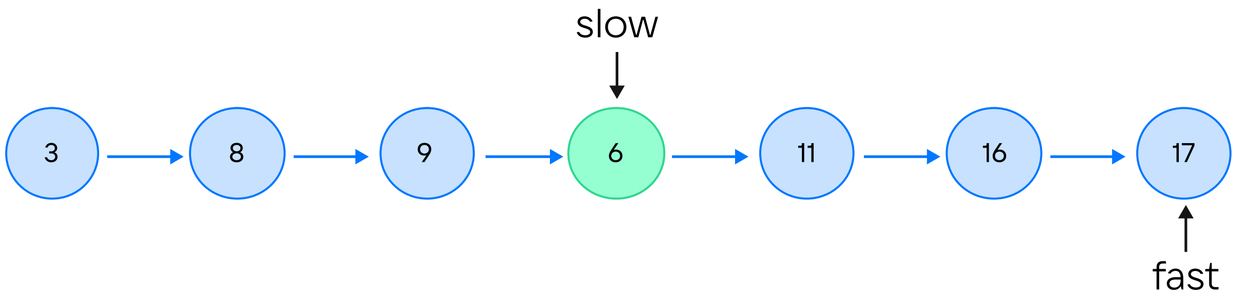
Применим два указателя. Один будет двигаться вперёд на next, а другой — на next.next





Середина связного списка

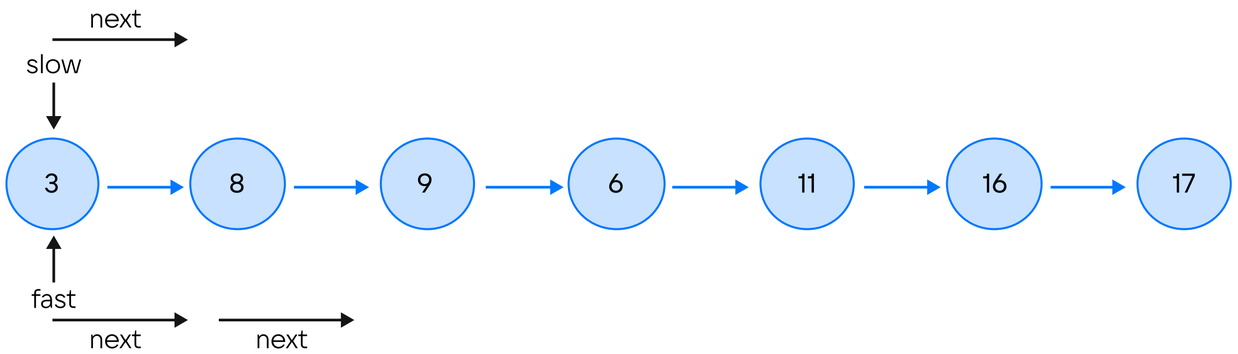
Когда быстрый указатель достигнет конца списка, указатель slow будет указывать на середину



Середина связного списка

Медленный и быстрый указатели стартуют с головы списка

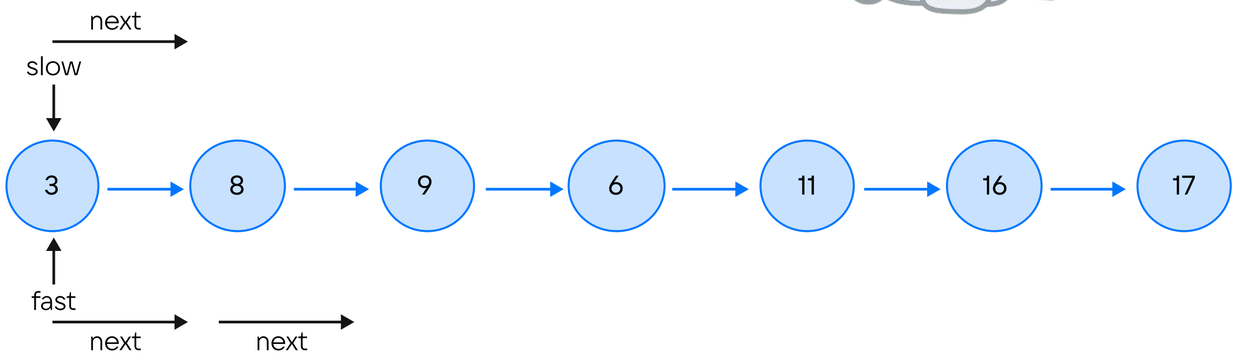
```
function middleNode(head) {
    slow = fast = head
    // . . .?
    return slow
}
```



Середина связного списка

Когда быстрый указатель достигнет конца списка, указатель **slow** будет указывать на середину

```
function middleNode(head) {
    slow = fast = head
    while fast != null and fast.next != null {
            slow = slow.next
            fast = fast.next.next
     }
     return slow
}
```



Резюме

- Элементы списка располагаются в памяти хаотично
- Каждый элемент знает, где хранится следующий
- Итерации по списку начинаются с его головы
- К списку также применим шаблон «два указателя»

education

Всем спасибо)