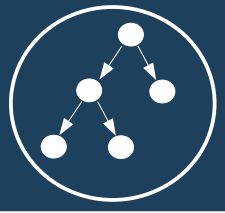


Data Structures and Algorithms

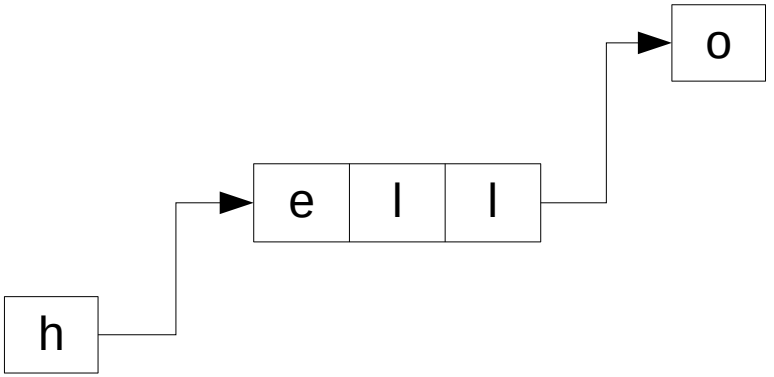
Структуры данных.
Очередь на основе двусвязного списка

Очередь

Очередь — это абстрактный тип данных, представляющий собой список элементов, организованных по принципу FIFO (англ. first in — first out, «первым пришёл — первым вышел»). Очередь - динамическая структура данных.

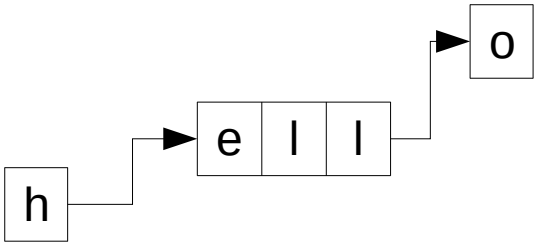
Поддерживаемые операции:

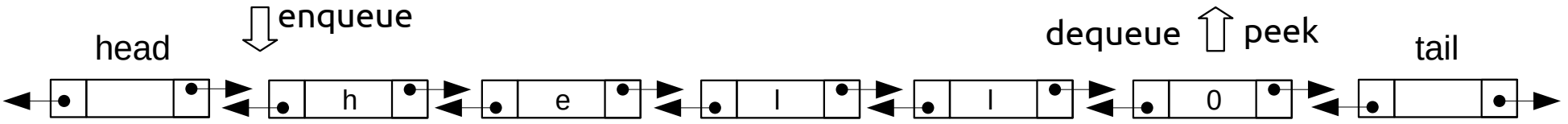
- Добавление элемента в конец очереди (enqueue)
- Удаление элемента из головы очереди (dequeue)
- Получение головного элемента без удаления (peek)
- Получение размера очереди



Реализация очереди на основе двусвязного списка

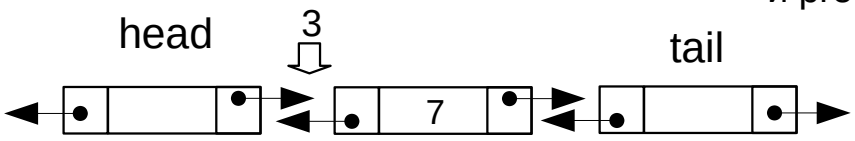
Пожалуй одной из самых простых реализаций очереди является использование двусвязного списка. У него операции добавления элемента в начало списка и удаления элемента с конца списка обладают константной сложностью, что делает его использование довольно оптимальным. К недостаткам стоит отнести повышенный расход памяти на хранение ссылок на элементы списка.



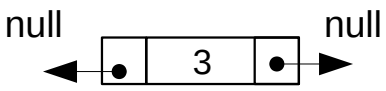


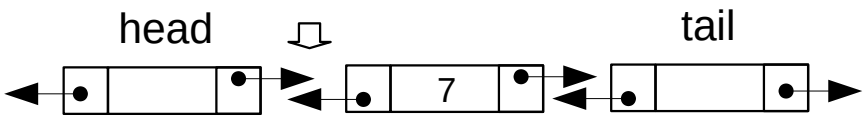
Добавление значения в конец очереди

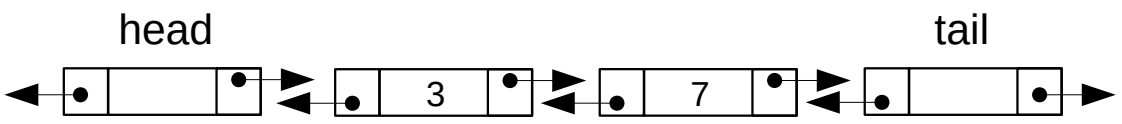
Создать новый узел который хранит добавляемое значение и next = null и prev = null



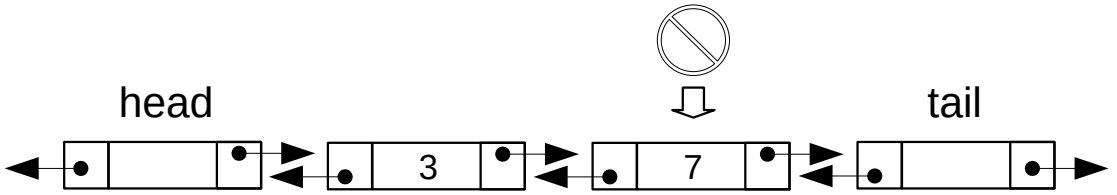
1) Установить значение next добавляемого узла равное head.next.
2) Установить prev добавляемого узла равное head.
3) Установить head.next.prev на добавляемый узел.
4) Установить head.next на добавляемый узел.



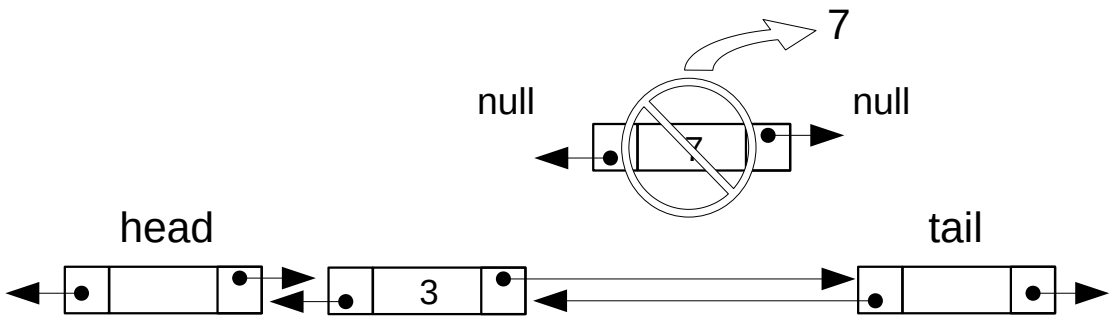




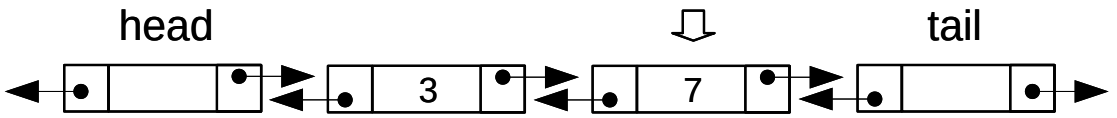
Получение значения с удалением с головы очереди



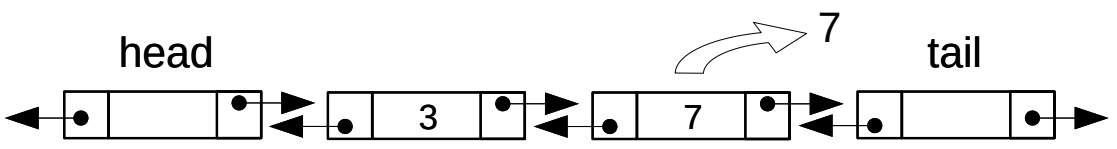
1) Устанавливаем значение tail.prev равным значению prev удаляемого узла
2) Устанавливаем значение next предыдущего узла равным значению tail
3) Устанавливаем значение next и prev удаляемого узла равными null
4) Сохраняем значение удаляемого узла
5) Освобождаем память занимаемую удаляемым узлом
6) Возвращаем сохраненное значение



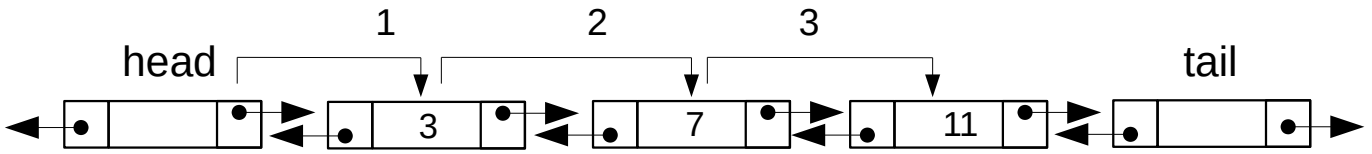
Получение значения без удаления с головы очереди



1) Перейти к узлу по ссылке tail.prev
2) Вернуть значение этого узла



Получение размера очереди



Для получения размера очереди стоит объявить переменную с начальным значением 0. Начиная с начала очереди выполнять переход по ссылке к следующему узлу. На каждом переходе увеличивать значение этой переменной на 1. Закончить на tail.

Python

Реализация на Python

Описание структуры узла и очереди

```
class Node:
    def __init__(self, data=None, next=None, prev = None):
        self.data = data
        self.next = next
        self.prev = prev

    def __str__(self):
        return str(self.date)


class Queue:

    def __init__(self):
        self.head = Node()
        self.tail = Node()
        self.head.next = self.tail
        self.tail.prev = self.head
        self.length = 0
```

Методы добавления в конец очереди

```
def enqueue(self,value):
        add_node = Node(value, self.head.next, self.head)
        self.head.next.prev = add_node
        self.head.next = add_node
        self.length += 1
```

Метод получения с удалением

```
def dequeue(self):
      if self.is_empty():
          return
      remove_node = self.tail.prev
      self.tail.prev = remove_node.prev
      remove_node.prev.next = remove_node.next
      return_value = remove_node.data
      remove_node.next = None
      remove_node.prev = None
      self.length -= 1
      return return_value
```

Методы получения без удаления

```
def peek(self):
        if self.is_empty():
            return
        remove_node = self.tail.prev
        return remove_node.data
```

Методы получения длины

```
def get_length(self):
    current_node = self.head.next
    length = 0
    while current_node != self.tail:
        length += 1
        current_node = current_node.next
    return length
```

Реализация на Java

Описание узла

```
private class Node {
	Object data;
	Node next;
	Node prev;

	public Node(Object data, Node next, Node prev) {
		super();
		this.data = data;
		this.next = next;
		this.prev = prev;
	}

	public Node() {
		super();
	}
	@Override
	public String toString() {
		return "Node [data=" + data + ", next=" + next + ", prev=" + prev + "]";
	}
}
```

Класс очередь

```
public class Queue {

    private final Node head;
    private final Node tail;
    private long length = 0;

    public Queue() {
        super();
        head = new Node();
        tail = new Node();
        head.next = tail;
        tail.prev = head;
    }
```

Метод добавления значения в голову очереди

```
public void enqueue(Object value) {
     Node addNode = new Node(value, head.next, head);
     head.next.prev = addNode;
     head.next = addNode;
     length += 1;
}
```

Метод получения значения с удалением

```
public Object dequeue() {
    if (isEmpty()) {
        return null;
    }
    Node removeNode = tail.prev;
    tail.prev = removeNode.prev;
    removeNode.prev.next = tail;
    Object value = removeNode.data;
    removeNode.next = null;
    removeNode.prev = null;
    length -= 1;
    return value;
}
```

Метод для получения значения без удаления

```
public Object peek() {
    if (isEmpty()) {
        return null;
    }
    Node removeNode = tail.prev;
    return removeNode.data;
}
```

Метод для получения размера

```
public long getLength() {
     long length = 0;
     Node currentNode = head.next;
     for (; currentNode != tail;) {
          length += 1;
          currentNode = currentNode.next;
     }
     return length;
}
```

Реализация на Fortran

Описание узла и списка

```
type Node
    integer::data_value
    class(Node),pointer::next=>null()
    class(Node),pointer::prev=>null()
end type Node

type Queue
    class(Node),pointer::head => null()
    class(Node),pointer::tail => null()
    integer::length = 0
    contains
        procedure,pass::init
        procedure,pass::enqueue
        procedure,pass::dequeue
        procedure,pass::get_length
        procedure,pass::peek
        procedure,pass::show
        procedure,pass::clear
        procedure,pass::destroy


end type Queue
```

Методы добавления в голову очереди

```
subroutine enqueue(this, data_value)
    class(Queue)::this
    integer, intent(in)::data_value
    class (Node), pointer::new_node
    allocate(new_node)
    new_node%data_value = data_value
    new_node%next => this%head%next
    new_node%prev => this%head
    new_node%next%prev => new_node
    this%head%next => new_node
    this%length = this%length + 1
end subroutine enqueue
```

Метод получения с удалением

```
function dequeue(this, op_result)
    class(Queue)::this
    logical, intent(inout) :: op_result
    class (Node), pointer::delete_node
    integer::dequeue
    op_result = .false.
    if(associated(this%head%next, this%tail)) then
        return
    end if
    delete_node => this%tail%prev
    this%tail%prev => delete_node%prev
    delete_node%prev%next => this%tail
    dequeue = delete_node%data_value
    delete_node%next => null()
    delete_node%prev => null()
    deallocate(delete_node)
    this%length = this%length - 1
    op_result = .true.
end function dequeue
```

Метод получения без удаления

```
function peek(this, op_result)
    class(Queue)::this
    logical, intent(inout) :: op_result
    integer::peek
    op_result = .false.
    if(associated(this%head%next, this%tail)) then
        return
    end if
    peek = this%tail%prev%data_value
    op_result = .true.
end function peek
```

Метод для получения длины

```
function get_length(this)
    class(Queue)::this
    class(Node), pointer::current_node
    integer::get_length
    get_length = 0
    current_node => this%head%next
    do
        if(associated(current_node,this%tail)) then
            exit
        end if
        get_length = get_length + 1
        current_node => current_node%next
    end do
end function get_length
```

Метод для создания, очистки, и удаления

```
subroutine init(this)
    class(Queue)::this
    allocate(this%head)
    allocate(this%tail)
    this%head%next => this%tail
    this%tail%prev => this%head
end subroutine init

subroutine clear(this)
    class(Queue)::this
    integer::n
    logical::op_result
    do
        n = this%dequeue(op_result)
        if(.not. op_result) then
            exit
        end if
    end do
end subroutine clear

subroutine destroy(this)
    class(Queue)::this
    if( .not. associated(this%head)) then
        return
    end if
    if(this%length > 0) then
        call this%clear()
    end if
    this%head%prev => null()
    this%head%next => null()
    this%tail%prev => null()
    this%tail%next => null()
    deallocate(this%head)
    deallocate(this%tail)
end subroutine destroy
```

Список литературы

1) Томас Кормен, Чарльз Лейзерсон, Рональд Ривест, Клиффорд Штайн // Алгоритмы: построение и анализ 3-е издание. — М.: «Вильямс», 2013. — С. 1328. ISBN 978-5-8459-1794-2
2) Роберт Седжвик, Кевин Уэйн «Алгоритмы на java 4-е издание» Пер. с англ. - М. : ООО "И.Д. Вильямс", 2013. ISBN 978-5-8459-1781-2.