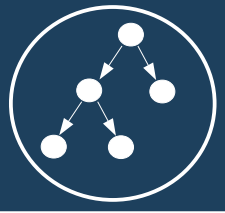


Data Structures and Algorithms

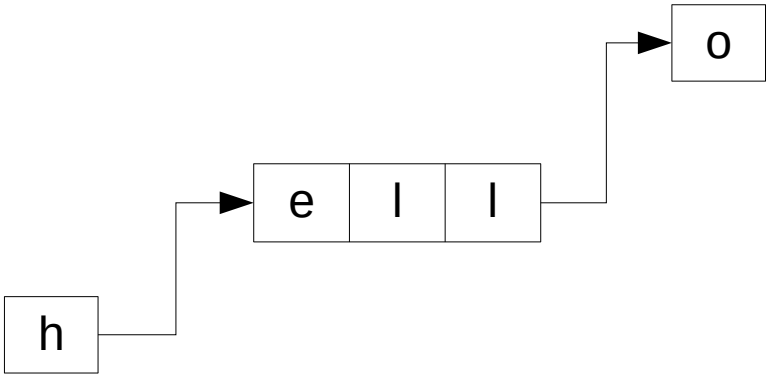
Структуры данных.
Очередь на основе массива

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.

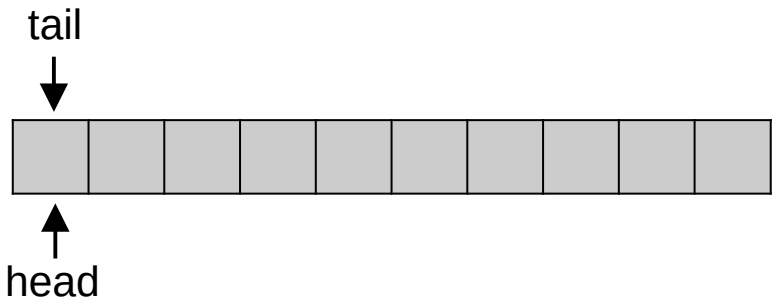
Поддерживаемые операции:

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

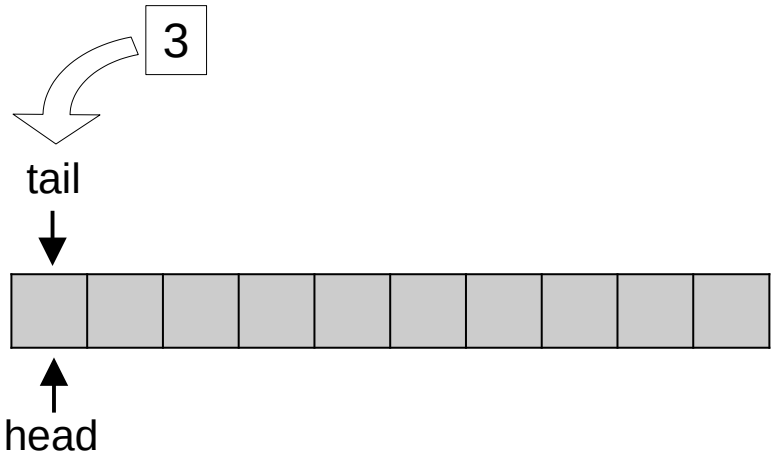


Реализация очереди на основе массива

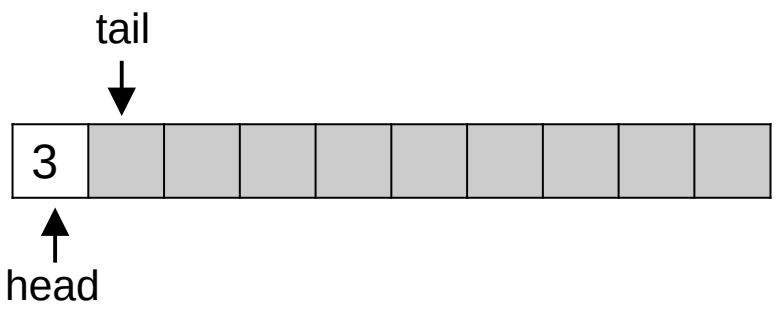
Для реализации очереди можно использовать одномерные массивы. Такой подход позволит значительно снизить затраты памяти на хранение по сравнению с очередью на основе двусвязного списка (не нужно хранить множество ссылок на соседние элементы). Идея и реализация также довольно простые. С помощью двух переменных хранят индексы указывающие на начало и конец очереди. Важной особенностью подобных индексов является их цикличность. Это означает что дойдя до конца последовательности они переходят в ее начало.



Добавление значения в конец очереди (tail + 1 ≠ head)



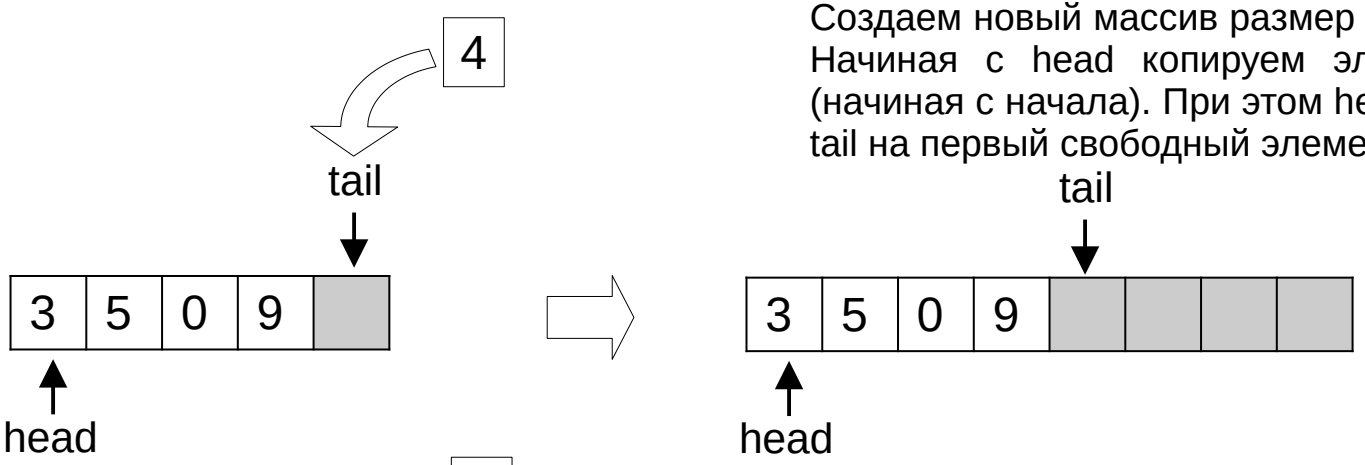
Устанавливаем элемент на индекс tail



Увеличиваем значение tail на единицу

Добавление значения в конец очереди (tail + 1 = head)

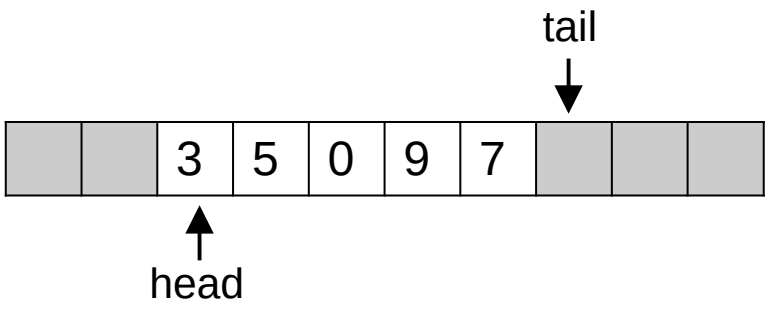
Создаем новый массив размер которого равен 4/3 + 1 от текущего. Начиная с head копируем элементы до tail в новый массив (начиная с начала). При этом head установить в начало массива, а tail на первый свободный элемент.



Ставим элемент на индекс tail. Увеличим значение tail на единицу.

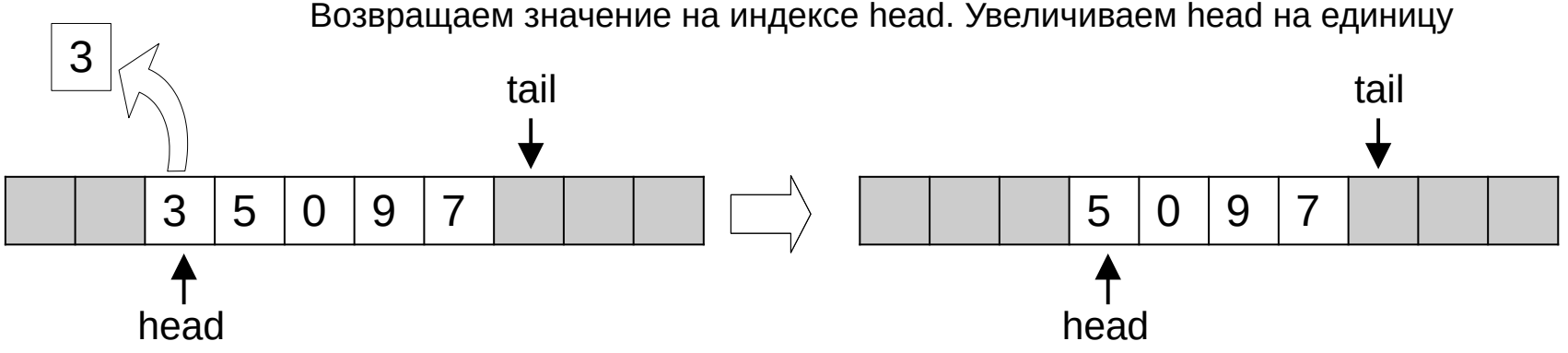
Освобождаем память занимаемую старым массивом

Получение значения с удалением с головы очереди

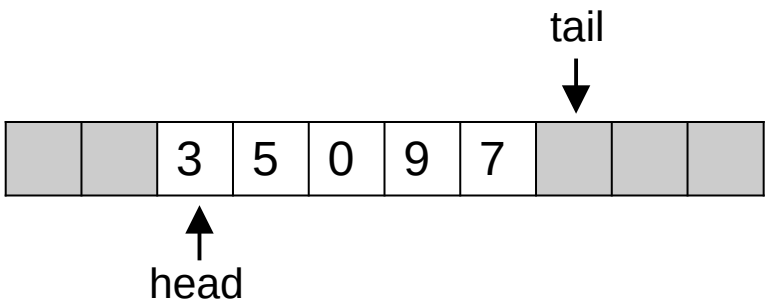


Если значение head=tail очередь пуста. Заканчиваем.

Возвращаем значение на индексе head. Увеличиваем head на единицу

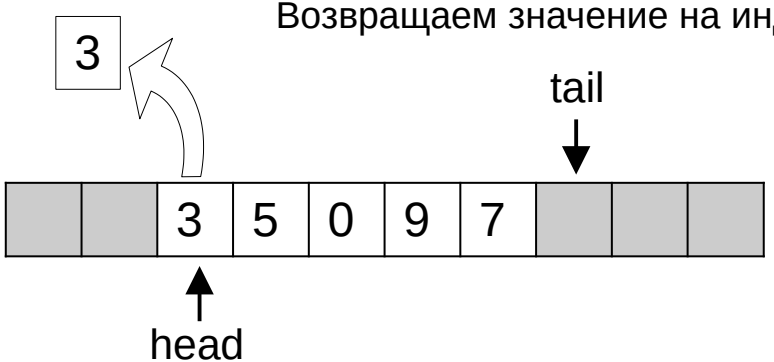


Получение значения без удаления с головы очереди



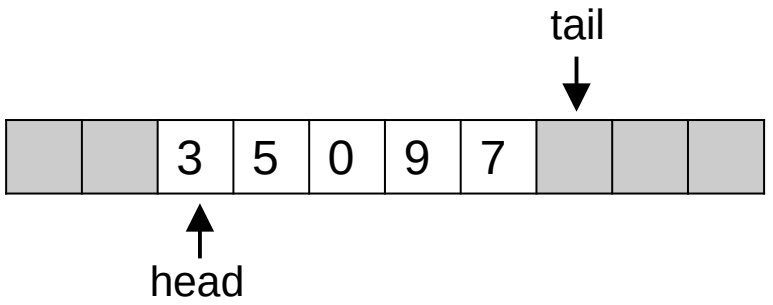
Если значение head=tail очередь пуста. Заканчиваем.

Возвращаем значение на индексе head



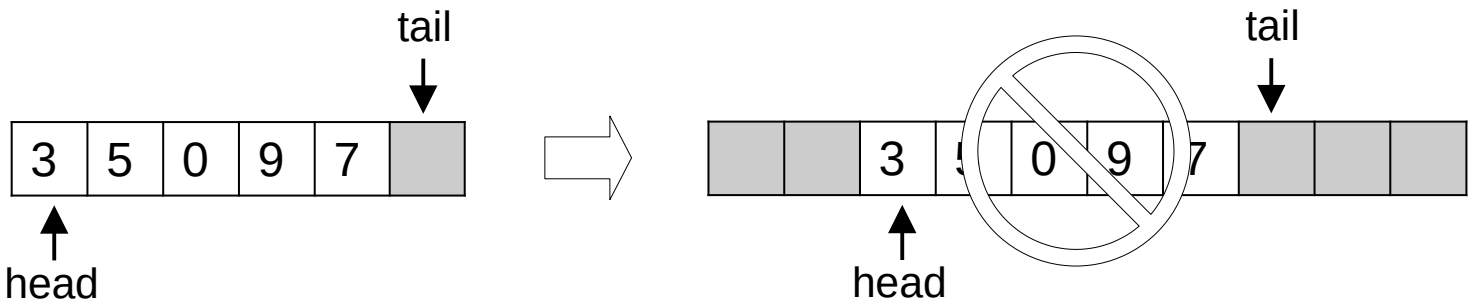
Уменьшение размера очереди

В большинстве случаев очередь на основе массива только увеличивает свой размер. Автоматического уменьшения не предусматривают. Для уменьшения размера используют функцию, вызов которой осуществляется по желанию разработчика.



Создается новый массив размер которого равен количеству элементов + 1. Начиная с head копируем элементы до tail в новый массив (начиная с начала). При этом head установить в начало массива, а tail на первый свободный элемент.

Освобождаем память занимаемую старым массивом



Получение размера очереди

Для получения размера очереди стоит объявить переменную с начальным значением 0. Начиная с начала очереди добавляем к значению head и этой переменной единицу, до тех пор пока head не станет равно tail. Вернуть значение переменной.

Реализация на Python

Отсутствие массивов в Python

Так как в Python отсутствует достаточная поддержка массивов, то реализация очереди на их основе не рассматривается.

Реализация на Java

Описание класса для реализации очереди

```
public class Queue {

     private final int DEFAULT_SIZE = 16;
     private Object[] data;
     private int head;
     private int tail;
     private int size;

     public Queue() {
          super();
          data = new Object[DEFAULT_SIZE];
          head = 0;
          tail = 0;
          size = 0;
     }
```

Метод добавления элемента в очередь

```
public void enqueue(Object value) {
	if ((tail + 1) % data.length == head) {
		increaseSize();
	}
	data[tail] = value;
	tail = (tail + 1) % data.length;
	size += 1;
}

private void increaseSize() {
	if (data.length >= Integer.MAX_VALUE - 10) {
		throw new IllegalArgumentException("can't increase the size");
	}
	Object[] newArray = new Object[Math.min(Integer.MAX_VALUE - 10, data.length * 4 / 3 + 1)];
	int addIndex = 0;
	for (;;) {
		if (head % data.length == tail) {
			break;
		}
		newArray[addIndex] = data[head % data.length];
		addIndex += 1;
		head = (head + 1) % data.length;
	}
	data = newArray;
	head = 0;
	tail = addIndex;
	size = tail;
}
```

Метод получения значения с удалением

```
public Object dequeue() {
    if (head == tail) {
        return null;
    }
    Object returnValue = data[head];
    data[head] = null;
    head = (head + 1) % data.length;
    size -= 1;
    return returnValue;
}
```

Метод для получения значения без удаления

```
public Object peek() {
    if (head == tail) {
        return null;
    }
    return data[head];
}
```

Метод для уменьшения размера

```
public void trimToSize() {
     Object[] newArray = new Object[size + 1];
     int addIndex = 0;
     for (;;) {
          if (head % data.length == tail) {
               break;
          }
          newArray[addIndex] = data[head % data.length];
          addIndex += 1;
          head = (head + 1) % data.length;
     }
     data = newArray;
     head = 0;
     tail = addIndex;
}
```

Реализация на Fortran

Описание структуры очереди

```
type Array_Queue
    integer,allocatable::data_array(:)
    integer::tail, head, queue_size

    contains
        procedure,pass::init
        procedure,pass::enqueue
        procedure,pass::up_resize
        procedure,pass::dequeue
        procedure,pass::pop
        procedure,pass::trim_to_size
        procedure,pass::clear
        procedure,pass::destroy
        procedure,pass::show
end type Array_Queue
```

Процедура инициализации очереди

```
subroutine init(this)
    class(Array_Queue)::this
    if(.not. allocated(this%data_array)) then
        allocate(this%data_array(16))
        this%tail = 1
        this%head = 1
        this%queue_size = 0
    end if
end subroutine init
```

Процедура добавления элемента

```
subroutine enqueue(this, data_value)
    class(Array_Queue)::this
    integer, intent(in)::data_value
    integer:: current_size, next_index
    current_size = size(this%data_array)
    next_index = this%tail + 1
    if (next_index > current_size) then
        next_index = 1
    end if
    if (next_index == this%head) then
        call this%up_resize()
    end if
    this%data_array(this%tail) = data_value
    this%tail = this%tail + 1
    if (this%tail > size(this%data_array)) then
        this%tail = 1
    end if
    this%queue_size = this%queue_size + 1
end subroutine enqueue
```

Процедура увеличения размера

```
subroutine up_resize(this)
    class(Array_Queue)::this
    integer,allocatable::temp_array(:)
    integer::add_index, new_size, old_size
    old_size = size(this%data_array)
    new_size = old_size * 4/3 + 1
    allocate(temp_array(new_size))
    add_index = 1
    do
        if(this%head == this%tail) then
            exit
        end if
        temp_array(add_index) = this%data_array(this%head)
        add_index = add_index + 1
        this%head = this%head + 1
        if(this%head > size(this%data_array)) then
            this%head = 1
        end if
    end do
    this%data_array = temp_array
    this%head = 1
    this%tail = add_index
    this%queue_size = add_index - 1
    deallocate(temp_array)
end subroutine up_resize
```

Процедура получения элемента с удалением

```
subroutine dequeue(this, element, op_result)
    class(Array_Queue)::this
    integer, intent(inout)::element
    logical, intent(inout)::op_result
    if(this%head == this%tail) then
        op_result = .false.
        return
    end if
    element = this%data_array(this%head)
    op_result = .true.
    this%head = this%head + 1
    if(this%head > size(this%data_array)) then
        this%head = 1
    end if
    this%queue_size = this%queue_size - 1
end subroutine dequeue
```

Процедура для получения элемента без удаления

```
subroutine peek(this, element, op_result)
    class(Array_Queue)::this
    integer, intent(inout)::element
    logical, intent(inout)::op_result
    if(this%head == this%tail) then
        op_result = .false.
        return
    end if
    element = this%data_array(this%head)
    op_result = .true.
end subroutine peek
```

Процедура для уменьшения размера

```
subroutine trim_to_size(this)
    class(Array_Queue)::this
    integer::add_index
    integer, allocatable::new_array(:)
    allocate(new_array(this%queue_size + 1))
    add_index = 1
    do
        if(this%head == this%tail) then
            exit
        end if
        new_array(add_index) = this%data_array(this%head)
        add_index = add_index + 1
        this%head = this%head + 1
        if(this%head > size(this%data_array)) then
             this%head = 1
        end if
    end do
    this%data_array = new_array
    this%head = 1
    this%tail = add_index
    this%queue_size = add_index - 1
    deallocate(new_array)
end subroutine trim_to_size
```

Процедура для очистки и удаления очереди

```
subroutine clear(this)
    class(Array_Queue)::this
    if(allocated(this%data_array)) then
        deallocate(this%data_array)
        allocate(this%data_array(16))
    end if
    this%head = 1
    this%tail = 1
    this%queue_size = 0
end subroutine clear

subroutine destroy(this)
    class(Array_Queue)::this
    if(allocated(this%data_array)) then
        deallocate(this%data_array)
    end if
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.