# STACKS | QUEUES

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil


Centro de Informática UFPE

Universidade Federal de Pernambuco

# Agenda

# Stack ADT

Policy: LIFO = last-in, first-out

Operations:

- void clear(Stack s);
- void push(Stack s, E it);
- E pop(Stack s);
- E topValue(Stack s);
- int length(Stack s);

Implementations based on: arrays and linked lists

# Stack ADT implemented as a linked list

Composite type (Stack):

```
1  Link top;                    // reference to the first element
2  int size;                                  // number of elements
```

Differently from the List implementation as a linked list: no header node

**Algorithm:** Stack create_stack()

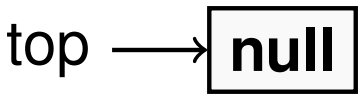
```
1  s.top ← NULL;
2  s.size ← 0;
3  return s;
```

# Stack ADT implemented as a linked list

**Algorithm:** void push(Stack s, E it)

1 $s.top \leftarrow create\_link(it, s.top)$;
2 $s.size$++;

`push(s,7)`, just after `create_stack()`



# Stack ADT implemented as a linked list

**Algorithm:** void push(Stack s, E it)

1 $s.top \leftarrow create\_link(it, s.top)$;
2 $s.size$++;

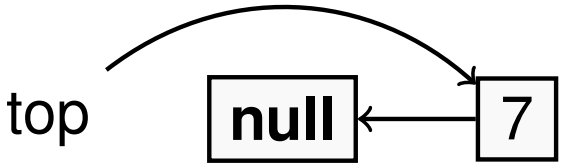
`push(s,7)`, just after `create_stack()`

# Stack ADT implemented as a linked list

**Algorithm:** void push(Stack s, E it)

1 $s.top \leftarrow create\_link(it, s.top)$;
2 $s.size$++;

`push(s,7)`, just after `create_stack()`

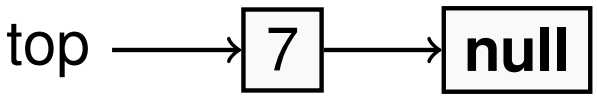


# Stack ADT implemented as a linked list

**Algorithm:** void push(Stack s, E it)

1 $s.top \leftarrow create\_link(it, s.top)$;
2 $s.size$++;

Better presented as follows

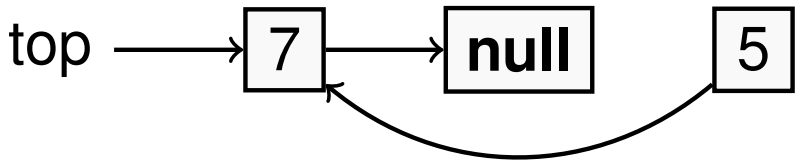


# Stack ADT implemented as a linked list

**Algorithm:** void push(Stack s, E it)

1 $s.top \leftarrow create\_link(it, s.top)$;
2 $s.size$++;

```
push(s,5)
```

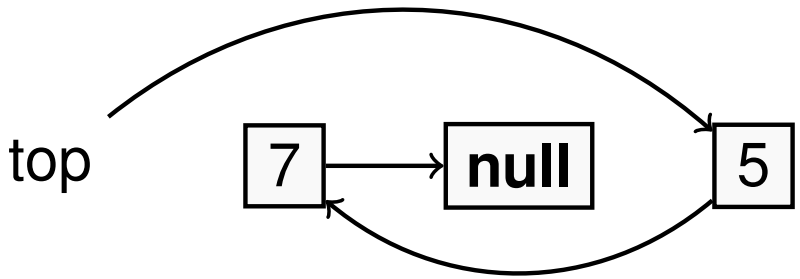


# Stack ADT implemented as a linked list

**Algorithm:** void push(Stack s, E it)

1 $s.top \leftarrow create\_link(it, s.top)$;
2 $s.size$++;

```
push(s,5)
```

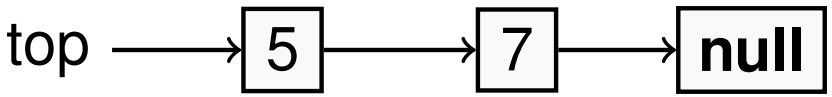


# Stack ADT implemented as a linked list

**Algorithm:** void push(Stack s, E it)

1 $s.top \leftarrow create\_link(it, s.top)$;
2 $s.size$++;

Better presented as follows

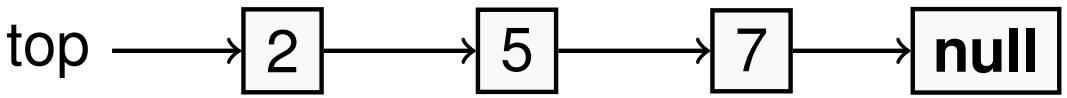


# Stack ADT implemented as a linked list

**Algorithm:** E pop(Stack s)

```
1 if s.top = NULL then error;
2 it ← s.top.element;
3 s.top ← s.top.next;
4 s.size--;
5 return it;
```

`pop(s)`, just after `push(s,2)`




Centro de Informática UFPE

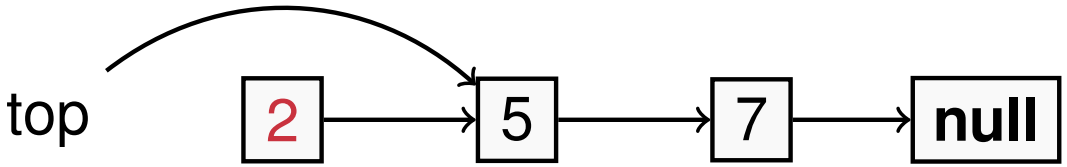
Universidade Federal de Pernambuco

# Stack ADT implemented as a linked list

**Algorithm:** E pop(Stack s)

```
1 if s.top = NULL then error;
2 it ← s.top.element;
3 s.top ← s.top.next;
4 s.size--;
5 return it;
```

`pop(s)`, just after `push(s,2)`

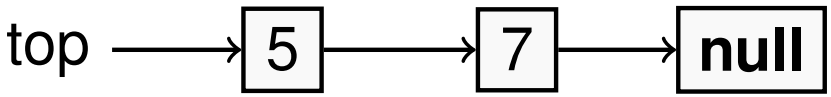


# Stack ADT implemented as a linked list

**Algorithm:** E pop(Stack s)

```
1 if s.top = NULL then error;
2 it ← s.top.element;
3 s.top ← s.top.next;
4 s.size--;
5 return it;
```

After memory deallocation (by the user or by the GC)




Centro de Informática UFPE

Universidade Federal de Pernambuco

# Agenda

# Queue ADT

Policy: FIFO = first-in, first-out

Operations:

- void clear(Queue q);
- void enqueue(Queue q, E it);
- E dequeue(Queue q);
- E frontValue(Queue q);
- int length(Queue q);

Implementations based on: arrays and linked lists

# Queue ADT implemented as a linked list

Composite Type (Queue):

```
1  Link front; Link rear; int size ;                    // queue size
```

This implementation does consider a header node

**Algorithm:** Queue create_queue()

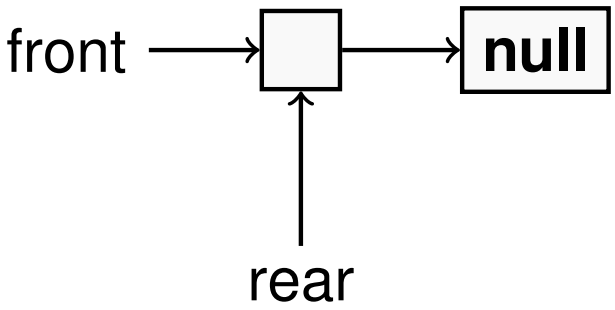
```
1  q.front ← q.rear ← create_link(NULL);                // header node
2  q.size ← 0;
3  return q;
```

# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1. $q.rear.next \leftarrow create\_link(it, NULL)$;
2. $q.rear \leftarrow q.rear.next$;
3. $q.size$++;

`enqueue(q,7)`, just after `create_queue()` (empty node: header)

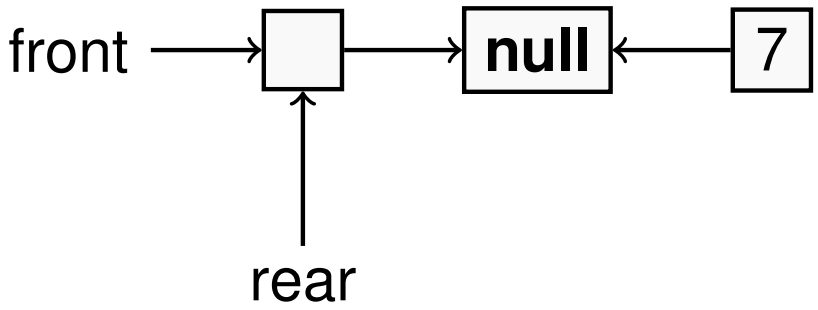


# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1 $q.rear.next \leftarrow create\_link(it, NULL)$;
2 $q.rear \leftarrow q.rear.next$;
3 $q.size$++;

`enqueue(q,7)`, just after `create_queue()` (empty node: header)

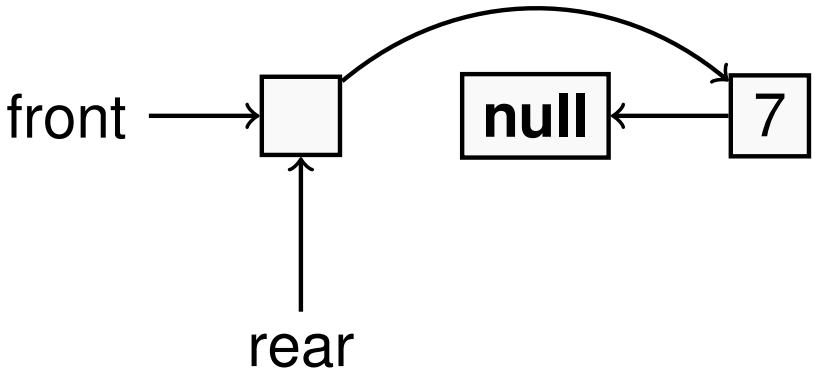


# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1 $q.rear.next \leftarrow create\_link(it, NULL)$;
2 $q.rear \leftarrow q.rear.next$;
3 $q.size$++;

`enqueue(q,7)`, just after `create_queue()` (empty node: header)




Centro de Informática UFPE

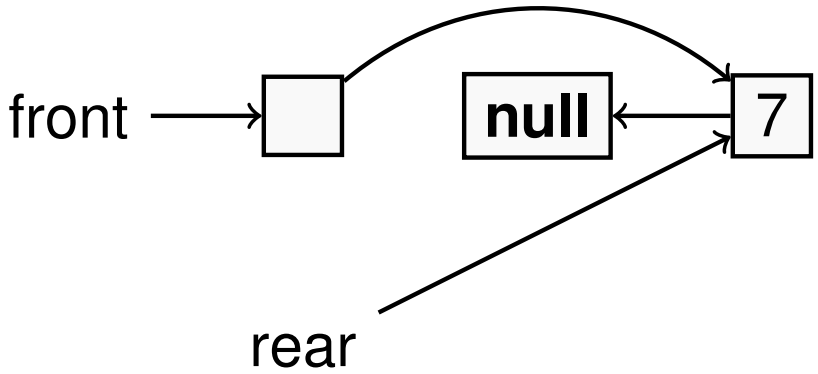
Universidade Federal de Pernambuco

# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1 $q.rear.next \leftarrow create\_link(it, NULL)$;
2 $q.rear \leftarrow q.rear.next$;
3 $q.size$++;

`enqueue(q,7)`, just after `create_queue()` (empty node: header)




Centro de Informática UFPE

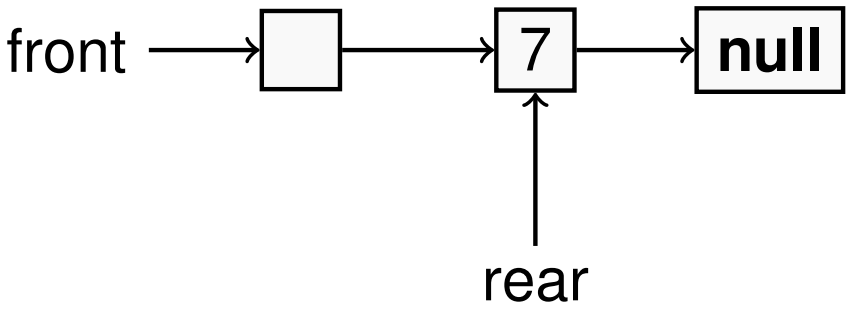
Universidade Federal de Pernambuco

# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1 $q.rear.next \leftarrow create\_link(it, NULL)$;
2 $q.rear \leftarrow q.rear.next$;
3 $q.size$++;

Better presented as follows:

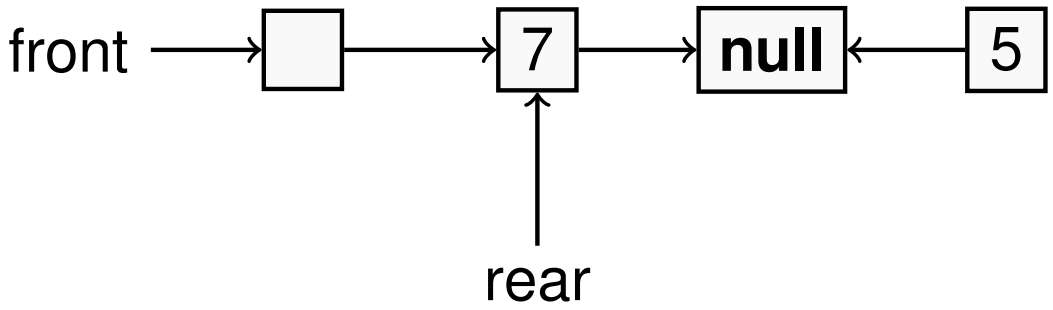


# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1 $q.rear.next \leftarrow create\_link(it, NULL)$;
2 $q.rear \leftarrow q.rear.next$;
3 $q.size$++;

```
enqueue(q,5)
```

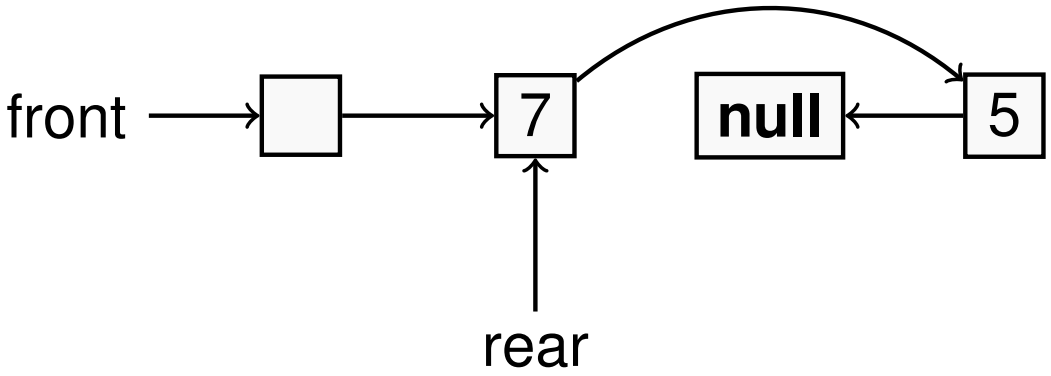


# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1. $q.rear.next \leftarrow create\_link(it, NULL)$;
2. $q.rear \leftarrow q.rear.next$;
3. $q.size$++;

```
enqueue(q,5)
```

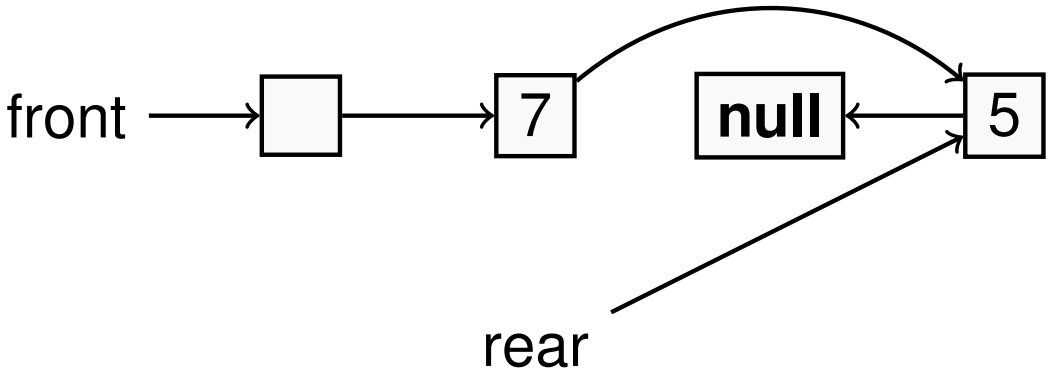


# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1 $q.rear.next \leftarrow create\_link(it, NULL)$;
2 $q.rear \leftarrow q.rear.next$;
3 $q.size$++;

```
enqueue(q,5)
```




Centro de Informática UFPE

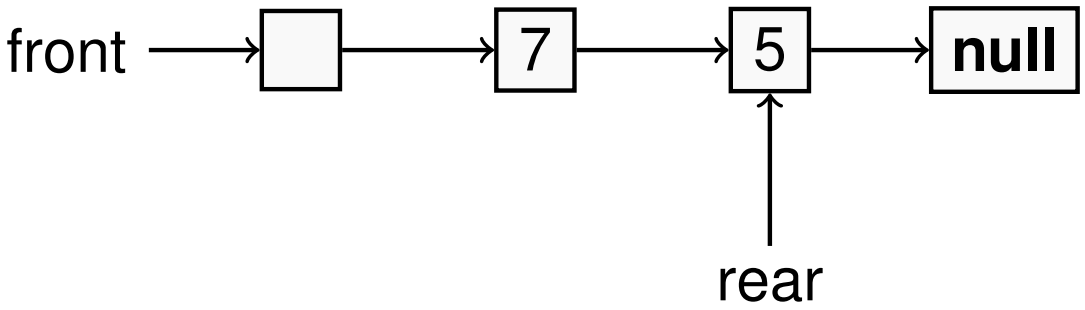
Universidade Federal de Pernambuco

# Queue ADT implemented as a linked list

**Algorithm:** void enqueue(Queue q, E it)

1 $q.rear.next \leftarrow create\_link(it, NULL)$;
2 $q.rear \leftarrow q.rear.next$;
3 $q.size$++;

Better presented as follows

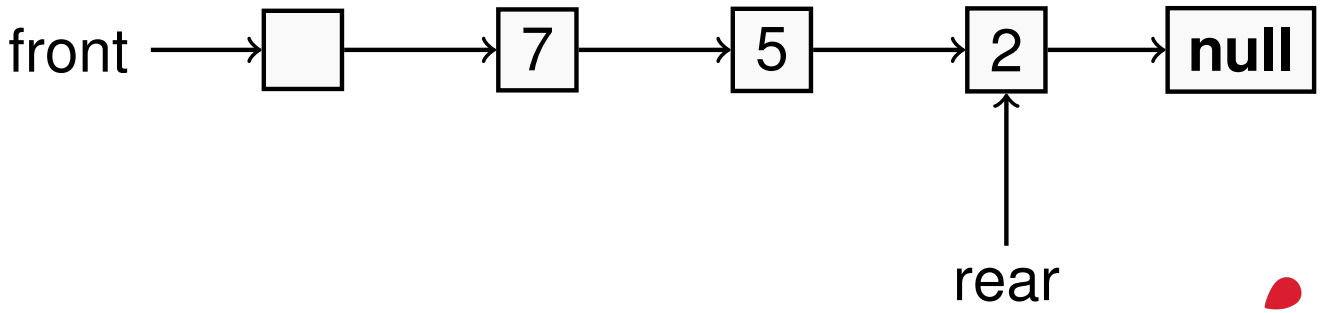


# Queue ADT implemented as a linked list

**Algorithm:** E dequeue(Queue q)

```
1 if q.size = 0 then error;
2 it ← q.front.next.element;
3 q.front.next ← q.front.next.next;
4 if q.front.next = NULL then q.rear ← q.front;
5 q.size--;
6 return it;
```

`dequeue(q)`, just after `enqueue(q,2)`




Centro de Informática UFPE

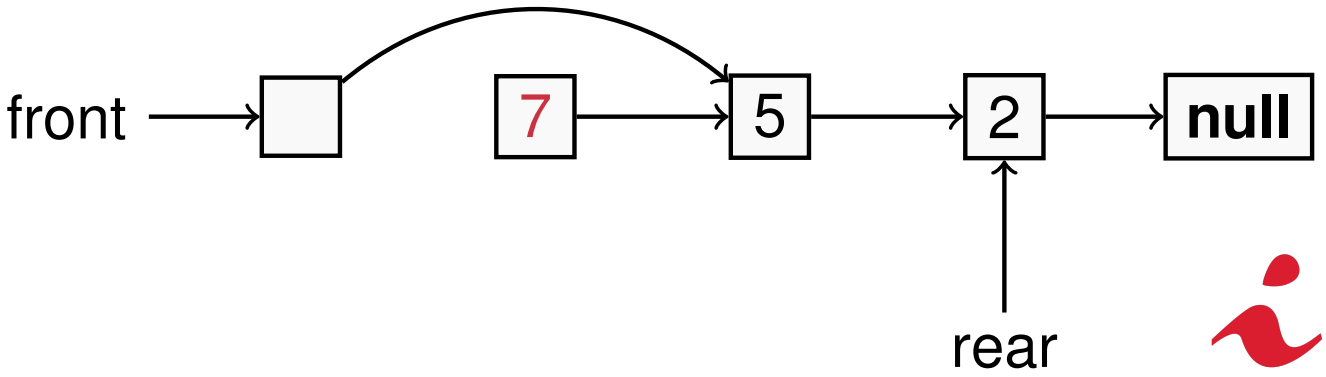
Universidade Federal de Pernambuco

# Queue ADT implemented as a linked list

**Algorithm:** E dequeue(Queue q)

```
1 if q.size = 0 then error;
2 it ← q.front.next.element;
3 q.front.next ← q.front.next.next;
4 if q.front.next = NULL then q.rear ← q.front;
5 q.size--;
6 return it;
```

`dequeue(q)`, just after `enqueue(q,2)`




Centro de Informática UFPE

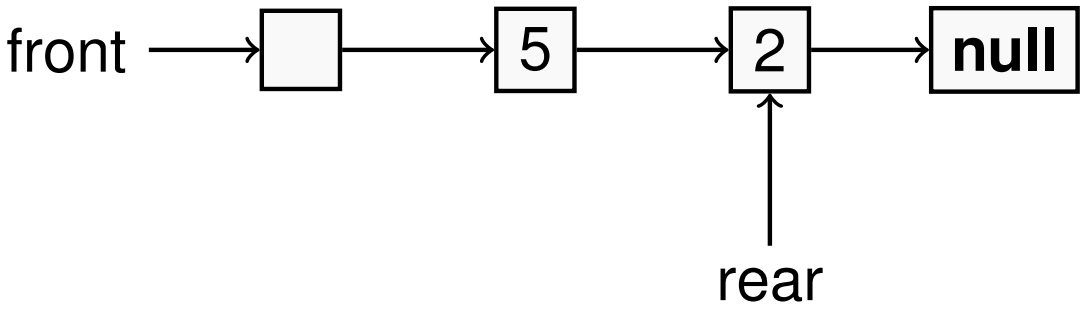
Universidade Federal de Pernambuco

# Queue ADT implemented as a linked list

**Algorithm:** E dequeue(Queue q)

```
1 if q.size = 0 then error;
2 it ← q.front.next.element;
3 q.front.next ← q.front.next.next;
4 if q.front.next = NULL then q.rear ← q.front;
5 q.size--;
6 return it;
```

After memory deallocation (by the user or by the GC)



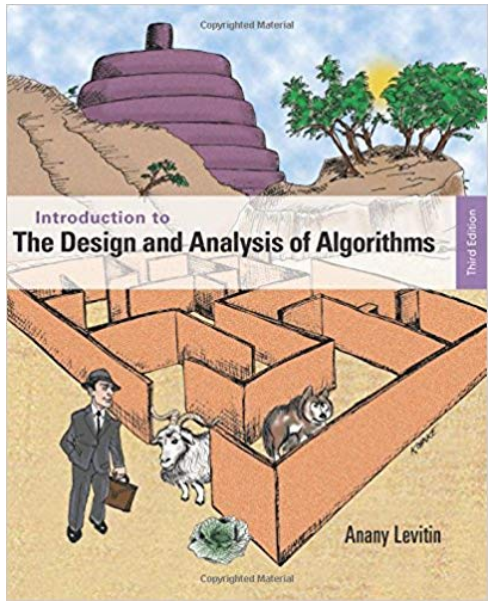

Centro de Informática UFPE

Universidade Federal de Pernambuco

# Agenda

# Bibliography


**Chapter 4 (pp. 117–131)**
**Clifford Shaffer.**
*Data Structures and Algorithm Analysis*.
Dover, 2013.




**Chapter 1 (pp. 25–28).**
**Anany Levitin.**
*Introduction to the Design and Analysis of Algorithms*.
3rd edition. Pearson. 2011.

Centro de Informática UFPE

Universidade Federal de Pernambuco

# STACKS | QUEUES

Gustavo Carvalho
(ghpc@cin.ufpe.br)

Universidade Federal de Pernambuco
Centro de Informática, 50740-560, Brazil


Centro de Informática UFPE

Universidade Federal de Pernambuco