Computación y Estructuras Discretas I

Andrés A. Aristizábal P.
aaaristizabal@icesi.edu.co

Departamento de Computación y Sistemas Inteligentes

UNIVERSIDAD ICESI

2024-2

Agenda

Introduction

What is an array?

Introduction

What is an array?

- A random access data structure.
- One can access any element in constant time.
- A typical random access example is a book.
- Random access is a fundamental matter for several algorithms such as the binary search.

Introduction

What is a linked list?

Introduction

What is a linked list?

- A sequential access data structure.
- One can only access an element in a particular order.
- A typical sequential access example is a paper roll or a tape.

Introduction

Are they any other types of data structures?

Introduction

Are they any other types of data structures?

- A subcase of sequential data structures.
- Limited access data structures.

Introduction

Are they any other types of data structures?

- A subcase of sequential data structures.
- Limited access data structures.

Which ones are those?

Introduction

Are they any other types of data structures?

- A subcase of sequential data structures.
- Limited access data structures.

Which ones are those?

- Stacks
- Queues

Agenda

Stacks

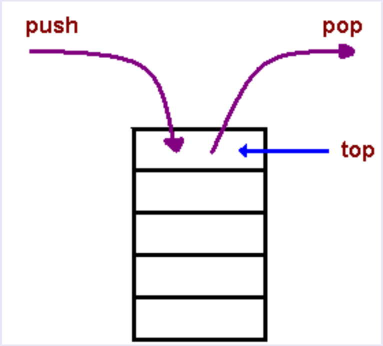
Intuitively, what is a stack?

Stacks

Intuitively, what is a stack?

- It is an object container where one can insert and extract elements according to the LIFO principle.
- It has two basic operations:
    - Push (inserting an element)
    - Pop (extracting an element)
- It is considered a limited access data structure since one can only insert and extract from its top.
- It is recursive.

Stacks



Stacks

What would it be a structural definition for a stack?

Stacks

What would it be a structural definition for a stack?

- A stack can be empty.
- or can have two parts.
  - The top of the stack (a value)
  - The rest (a stack)

Stacks

What does this definition tell us?

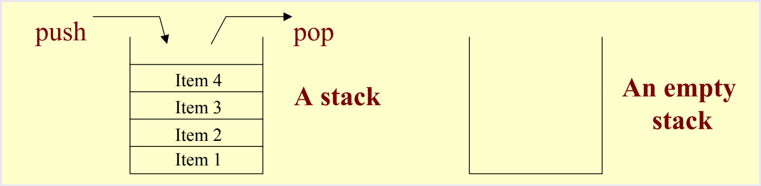
Stacks

What does this definition tell us?

- That it follows the normal pattern for a recursive definition.

  Base case: Empty is a valid value for the stack.
  Recursive case: A top and a stack are appropriate values for the stack.

Stacks



Stacks

Ejemplo

Let $S = (7, (29, (11, \emptyset)))$, *Is S a valid stack?*

Stacks

How would we define the Stack ADT?

Stacks

How would we define the Stack ADT?

Name?

Stacks

How would we define the Stack ADT?

Name?

Stack ADT

Stacks

How would we define the Stack ADT?

Name?

Stack ADT

Abstract object?

Stacks

How would we define the Stack ADT?

Name?

Stack ADT

Abstract object?

$Stack = \langle\langle e_1, e_2, e_3, ..., e_n\rangle, top\rangle$

Stacks

How would we define the Stack ADT?

Name?

Stack ADT

Abstract object?

$Stack = \langle\langle e_1, e_2, e_3, ..., e_n\rangle, top\rangle$

Invariant?

Stacks

How would we define the Stack ADT?

Name?

Stack ADT

Abstract object?

$Stack = \langle\langle e_1, e_2, e_3, ..., e_n\rangle, top\rangle$

Invariant?

$0 \leq n \wedge Size(Stack) = n \wedge top = e_n$

Stacks

Constructor operations?

Stacks

Constructor operations?

- `Stack` $- \longrightarrow$ *Stack*
  - Builds an empty stack
  - Preconditions: –
  - Postconditions: Stack $s = \emptyset$

Stacks

Modifiers?

Stacks

Modifiers?

- `push` $\mathit{Stack} \times \mathit{Element} \longrightarrow \mathit{Stack}$
  - Adds the new element e to stack s
  - Preconditions: Stack $s = \langle e_1, e_2, e_3, ..., e_n \rangle$ and element e or $s = \emptyset$ and element e
  - Postconditions: Stack $s = \langle e_1, e_2, e_3, ..., e_n, e \rangle$ or $s = \langle e \rangle$

Stacks

- `pop` *Stack* $\longrightarrow$ *Stack*
  - Extracts from the stack s, the most recently inserted element.
  - Preconditions: Stack $s \neq \emptyset$ i.e. $s = \langle e_1, e_2, e_3, ..., e_n \rangle$
  - Postconditions: Stack $s = \langle e_1, e_2, e_3, ..., e_{n-1} \rangle$

Stacks

- `top` *Stack* $\longrightarrow$ *Element*
  - Recovers the value of the element on the top of the stack.
  - Preconditions: Stack $s \neq \emptyset$ i.e. $s = \langle e_1, e_2, e_3, ..., e_n \rangle$
  - Postconditions: Element e_n

Stacks

- `isEmpty` *Stack* $\longrightarrow$ *boolean*
  - Determines if the stack s is empty or not.
  - Preconditions: Stack s
  - Postconditions: True if $s = \emptyset$, False if $s \neq \emptyset$

Stacks

Destructors?

Stacks

Destructors?

- `~Stack` *Stack* $\longrightarrow$ –
  - Destroys stack *s* freeing memory.
  - Preconditions: Stack *s*
  - Postconditions: –

Stacks

Ejemplo

```
printStack(Stack s) {
  while (!s.isEmpty()) {
    elem = s.top())
    s.pop()
    print elem
  }
}
```

Stacks

Which are the axioms that the access operations must guarantee in a Stack ADT?

Stacks

Which are the axioms that the access operations must guarantee in a Stack ADT?

Let *s* be a stack and *e* an element.

1. (s.Stack()).isEmpty() = true
2. (s.push(e)).isEmpty() = false
3. (s.Stack( )).top() = error
4. (s.push(e)).top() = e
5. (s.Stack()).pop() = error
6. (s.push(e)).pop() = e

Stacks

Which are the similarities between a stack and a list?

Stacks

Which are the similarities between a stack and a list?

- top() is equivalent to obtaining the first element in a list.
- pop() is equivalent to deleting the first element in the list.
- push(e) is equivalent to adding an element at the beginning of the list.

Stacks

Which are the main operations in the Stack class implemented in Java?

Stacks

Which are the main operations in the Stack class implemented in Java?

- `boolean empty()` (`isEmpty()`)
- `E peek()` (`top()`)
- `E pop()` (`pop()`)
- `E push(E item)` (`push(E item)`)

Stacks

How would we implement the Stack ADT?

Stacks

How would we implement the Stack ADT?

What would be the interface?

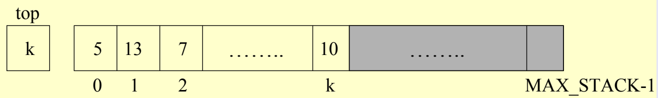
```
public interface StackInterface{
    public boolean isEmpty();
    //Pre: none
    //Post: Returns true if the stack is empty, otherwise returns false.
    public void push(Object item) throws StackException;
    //Pre:item is the new item to be added
    //Post: If insertion is successful, item is on the top of the stack.
    //Post:Throw StackException if the insertion is not successful
    public Object top( ) throws StackException;
    //Pre: none
    //Post: If stack is not empty, the item on the top is returned. The stack is left unchanged
    //Post:  Throws StackException if the stack is empty.
    public void pop( ) throws StackException;
    //Pre: none
    //Post: If stack is not empty, the item on the top is removed from the stack.
    //Post:  Throws StackException if the stack is empty.
```

Stacks

An implementation based on arrays?

Stacks

An implementation based on arrays?



Data structure

```
class StackArrayBased{
     final int MAX_STACK = 50;
     private Object items[ ];
     private int top;
...};
```

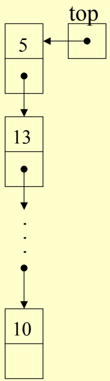
```
public class StackArrayBased
                           implements StackInterface{
     final int MAX_STACK = 50;
     private Object items[ ];
     private int top;

     public StackArrayBased( ){
            items = new Object[MAX_STACK];
            top = –1;
     }// end default constructor;

     public boolean isEmpty( ){
            return top < 0; }// end isEmpty
......}
```

Stacks

A dynamic implementation?

Stacks

A dynamic implementation?



Data structure

```
class StackReferenceBased{
      private Node top;
…}
```

```
class Node{
        Object item;
        Node next;
…}
```

```
public class StackReferenceBased
                            implements StackInterface{

      private Node top;

      public StackReferenceBased( ){
              top = null;
      }// end default constructor;

      public boolean isEmpty( ){
              return top == null; }// end isEmpty
……}
```

Implementation

Stacks

Which are some of the applications of stacks in computer science?

Stacks

Which are some of the applications of stacks in computer science?

- Reverse a word.
- The undo mechanism on a text editor
- Backtracking

Agenda

Queues

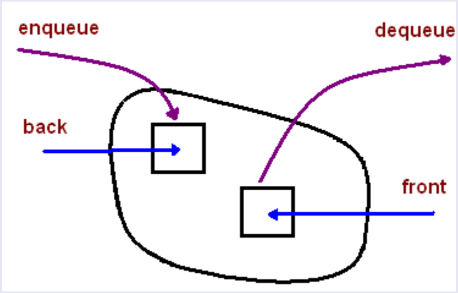
Intuitively, what is a queue?

Queues

Intuitively, what is a queue?

- It is an object container where you can insert and extract elements following the FIFO principle.
- It has two basic operations:
  - Enqueue (to insert an element)
  - Dequeue (to extract an element)
- It has limited access since one can only insert an element at the back of the queue and extract from its front.

Queues



Queues

Which are the main operations in the Queue interface in Java?

Queues

Which are the main operations in the Queue interface in Java?

- `boolean isEmpty()` (`isEmpty()`)
- `E peek()` (`front()`)
- `E poll()` (`dequeue()`)
- `boolean offer(E item)` (`enqueue(E item)`)

Queues

How would we define the Queue ADT?

Queues

How would we define the Queue ADT?

Name?

Queues

How would we define the Queue ADT?

Name?

Queue ADT

Queues

How would we define the Queue ADT?

Name?

Queue ADT

Abstract object?

Queues

How would we define the Queue ADT?

Name?

Queue ADT

Abstract object?

$Queue = \langle \langle e_1, e_2, e_3, ..., e_n \rangle, front, back \rangle$

Queues

How would we define the Queue ADT?

Name?

Queue ADT

Abstract object?

$Queue = \langle \langle e_1, e_2, e_3, ..., e_n \rangle, front, back \rangle$

Invariant?

Queues

How would we define the Queue ADT?

Name?

Queue ADT

Abstract object?

$Queue = \langle\langle e_1, e_2, e_3, ..., e_n\rangle, front, back\rangle$

Invariant?

$0 \leq n \wedge Size(Queue) = n \wedge front = e_1 \wedge back = e_n$

Queues

Constructor operations?

Queues

Constructor operations?

- `Queue` $- \longrightarrow$ *Queue*
  - Builds an empty queue
  - Preconditions: –
  - Postconditions: Queue $q = \emptyset$

Queues

Modifiers?

Queues

Modifiers?

- `enqueue` $\mathit{Queue} \times \mathit{Element} \longrightarrow \mathit{Queue}$
  - Inserts a new element e to the back of the queue q
  - Preconditions: Queue $q = \langle e_1, e_2, e_3, ..., e_n \rangle$ and element e or $q = \emptyset$ and element e
  - Postconditions: Queue $q = \langle e_1, e_2, e_3, ..., e_n, e \rangle$ or $q = \langle e \rangle$

Queues

- `dequeue` *Queue* $\longrightarrow$ *Element*
  - Extracts the element in Queue q's front
  - Preconditions: Queue $q \neq \emptyset$ i.e. $q = \langle e_1, e_2, e_3, ..., e_n \rangle$
  - Postconditions: Queue $q = \langle e_2, e_3, e_4, ..., e_{n-1} \rangle$ and Element e_1

Queues

- `front` *Queue* $\longrightarrow$ *Element*
  - Recovers the value of the element on the front of the queue.
  - Preconditions: Queue $q \neq \emptyset$ i.e. $q = \langle e_1, e_2, e_3, ..., e_n \rangle$
  - Postconditions: Element e_1

Queues

- `isEmpty`
    - Determines if the Queue q is empty or not.
    - Preconditions: Queue q
    - Postconditions: True if $q = \emptyset$, False if $q \neq \emptyset$

Queues

Destructors?

Queues

Destructors?

- `~Queue`
    - Destroys queue *q* freeing memory.
    - Preconditions: Queue *q*
    - Postconditions: –

Queues

Which are the axioms that the access operations must guarantee in a Queue ADT?

Queues

Which are the axioms that the access operations must guarantee in a Queue ADT?

Let *q* be a queue and *e* an element.

1. (q.Queue()).isEmpty() = true
2. (q.enqueue(e)).isEmpty() = false
3. (q.Queue()).front() = error
4. pre: q.isEmpty() = true
   (q.enqueue(e)).front() = e
5. (q.Queue()).dequeue() = error
6. pre: q.isEmpty()=false
   (q.enqueue(e)).dequeue()=(q.dequeue()).enqueue(e)

Queues

How would we implement the Queue ADT?

Queues

How would we implement the Queue ADT?

- Dinamically
- Statically

Queues

Which are some of the applications of queues in computer science?

Queues

Which are some of the applications of queues in computer science?

- Round robin scheduling
- Job scheduling
- Keyboard buffer
- Print queue

Agenda

Exercises

Ejercicio

Using stacks and/or queues (use the class Stack and the interface Queue).

1. *Implement an algorithm that reverses a string.*
2. *Implement an algorithm that reverses a stack (use a generic method)*
3. *Implement an algorithm that reverses a queue (use a generic method).*
4. *Implement an algorithm that verifies if an arithmetic expression has balanced parenthesis.*

Exercises

Ejercicio

Complete the Stack and Queue ADT and start their implementation using Generics