GUÍA DE APRENDIZAJE PARA CONCURSANTES ICPC Y IOI: ESTRUCTURA DE DATOS PILA Y COLA (STACK, QUEUE)

---

1. Introducción

Dentro de las estructuras básicas que debe dominar un concursante están las estructuras de datos pila(*Stack*) y cola(*Queue*). A ellas les vamos a dedicar la siguiente guía de aprendizaje.

2. Conocimientos previos

2.1. Estructura de Datos

Una estructura de datos es una forma de organizar un conjunto de datos elementales con el objetivo de facilitar su manipulación. Un dato elemental es la mínima información que se tiene en un sistema. Una estructura de datos define la organización e interrelación de estos y un conjunto de operaciones que se pueden realizar sobre ellos. Cada estructura ofrece ventajas y desventajas en relación a la simplicidad y eficiencia para la realización de cada operación. De está forma, la elección de la estructura de datos apropiada para cada problema depende de factores como la frecuencia y el orden en que se realiza cada operación sobre los datos.

2.2. Estructuras Dinámicas Lineales

Las estructuras dinámicas son aquellas que el tamaño no está definido y puede cambiar en la ejecución del algoritmo. Estas estructuras suelen ser eficaces y efectivas en la solución de problemas complejos. Su tamaño puede reducir o incrementar en la ejecución del algoritmo.

Entre las estructuras dinámicas lineales están:

- Vector
- Pila
- Cola

3. Desarrollo

3.1. Cola

Una cola es una estructura de datos en la que el modo de acceso a sus elementos es de tipo FIFO (del inglés *First In First Out*, primero en entrar, primero en salir) que permite almacenar y recuperar datos.Así, los elementos sólo se pueden eliminar por el principio y sólo se pueden añadir por el final de la cola.

La estructura independientemente del lenguaje presenta un grupo de funciones para poder manipular los datos que en ella se almacena.

3.1.1. C++

Para la utilización de está estructura es necesario añadir en la cabecera del programa **#include <queue>**, gracias a esto ya podemos utilizar la estructura de otro modo no. La clase **queue** es genérica.

- **queue::empty():** Devuelve si la cola está vacía.
- **queue::size():** Devuelve el tamaño de la cola.
- **queue::swap():** Intercambia el contenido de dos colas, pero las colas deben ser del mismo tipo, aunque los tamaños pueden diferir.
- **queue::emplace():** Inserta un nuevo elemento en el contenedor de la cola, el nuevo elemento se agrega al final de la cola.
- **queue::front():** Devuelve una referencia al primer elemento de la cola.
- **queue::back():** Devuelve una referencia al último elemento de la cola.
- **queue::push(g):** Agrega el elemento 'gál final de la cola.
- **queue::pop():** Elimina el primer elemento de la cola.

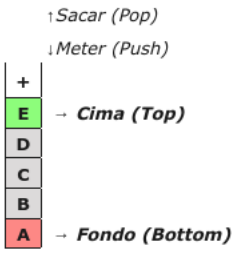
3.1.2. Java

Para la utilización de está estructura es necesario importar del paquete java dentro del subpaquete util la clase *Queue* **import java.util.Queue;** , gracias a esto ya podemos utilizar la estructura de otro modo no.

- **add boolean add(E e):** Agrega el elemento e a la cola al final (cola) de la cola sin violar las restricciones de capacidad. Devuelve verdadero si tiene éxito o IllegalStateException si la capacidad está agotada.
- **peek E peek():** Devuelve la cabeza (frente) de la cola sin eliminarla.
- **element E element():** Realiza la misma operación que el método peek(). Lanza NoSuchElementException cuando la cola está vacía.
- **remove E remove():** Elimina la cabeza de la cola y la devuelve. Lanza NoSuchElementException si la cola está vacía.
- **poll E poll():** Elimina la cabeza de la cola y la devuelve. Si la cola está vacía, devuelve nulo.
- **size int size():** Devuelve el tamaño o el número de elementos en la cola.
- **Offer boolean offer(E e):** Inserta el nuevo elemento e en la cola sin violar las restricciones de capacidad.

3.2. Pila

Una pila es una estructura de datos en la que el modo de acceso a sus elementos es de tipo LIFO (del inglés *Last In First Out*, último en entrar, primero en salir) que permite almacenar y recuperar datos. Para el manejo de los datos se cuenta con dos operaciones básicas: **apilar**, que coloca un objeto en la pila, y su operación inversa, **des-apilar** (retirar), que retira el último elemento apilado. La clase **stack** es genérica.



La estructura independientemente del lenguaje presenta un grupo de funciones para poder manipular los datos que en ella se almacena.

3.2.1. C++

Para la utilización de está estructura es necesario añadir en la cabecera del programa **#include <stack>**, gracias a esto ya podemos utilizar la estructura de otro modo no.

- **stack::top():** Devuelve el elemento que esta en el tope de la pila.
- **stack::empty():** Está función retorna verdad si la pila está vacía y retorna falso si es que por lo menos tiene un elemento como tope.
- **stack::size():** Está función retorna cuantos elementos tiene la pila, pero sin embargo no se puede acceder a ellas por lo que no es muy usual el uso de está función.
- **stack::push(g):** Agrega el elemento 'gál tope de la pila.
- **stack::pop():** Elimina el elemento que esta en el tope de la pila.

3.2.2. Java

Para la utilización de está estructura es necesario importar del paquete java dentro del subpaquete util la clase *Stack* **import java.util.Stack;** , gracias a esto ya podemos utilizar la estructura de otro modo no.

- **push:** Adiciona al tope de la pila el elemento pasado por parámetro
- **pop:** Elimina y devuelve el elemento que esta en el tope de la pila siempre que esta tenga elemento, en caso de estar vacia se lanza una excepción.
- **peek:** Devuelve el elemento sin eleiminarlo que esta en el tope de la pila siempre que esta tenga elemento, en caso de estar vacia se lanza una excepción.

- **empty:** Comprueba si la pila esta vacia. Devuelve verdadero si la pila esta vacia y falso en caso contrario.
- **search:** Busca un elemento de la pila y devuelve la posición con respecto al tope de la pila donde se encuentra la primera ocurrencia del elemento. En caso que el elemento fuera el tope de la pila el valor devuelto sería 1 y así sucesivamente se iría incrementando a medida que se alejara del tope. En caso que elemento no este el valor devuelto será -1.

4. Implementación

4.1. C++

4.1.1. Pila

```
#include <iostream>
#include <stack>
using namespace std ;
int main (){
   stack<int> stc;
   stc.push(100) ;
   stc.push (200) ;
   stc.push(300) ;
   cout<<stc.top()<<"\n";//resultado 300
   stc.pop();
   cout<<stc.top()<<"\n";//resultado 200
}
```

4.1.2. Cola

```
#include <iostream>
#include <queue>

using namespace std;

void showq(queue<int> gq){
   queue<int> g = gq;
   while (!g.empty()) {
      cout << '\t' << g.front();
      g.pop();
   }
   cout << '\n';
}

int main(){
   queue<int> gquiz;
   gquiz.push(10);
   gquiz.push(20);
```

```
    gquiz.push(30);
    cout << "The queue gquiz is : ";
    showq(gquiz);
    cout << "\ngquiz.size() : " << gquiz.size();
    cout << "\ngquiz.front() : " << gquiz.front();
    cout << "\ngquiz.back() : " << gquiz.back();
    cout << "\ngquiz.pop() : ";
    gquiz.pop();
    showq(gquiz);
    return 0;
}
```

4.2. Java

4.2.1. Pila

```
import java.util.*;
public class Main {
   public static void main(String[] args) {
      Stack<String> pila =new Stack<String> ();
      pila.push("Matanzas");
      String tope=pila.peek();
      int position=pila.search("Matanzas");
      String tope=pila.pop();
      boolean isEmpty=pila.empty();
   }
}
```

4.2.2. Cola

```
import java.util.*;

public class Main {
   public static void main(String[] args) {
      Queue<String> str_queue = new LinkedList<>();
      str_queue.add("one");
      str_queue.add("two");
      str_queue.add("three");
      str_queue.add("four");
      System.out.println("The Queue contents:" + str_queue);
   }
}
```

5. Complejidad

Todos los métodos de estas estructuras su complejidad es $O(1)$. Lo anterior no quita que cuando tengamos que despilar o descolar las estructuras este proceder su complejidad va ser igual $O(n)$ siendo n la cantidad de elementos que posea la estructura.

6. Aplicaciones

Las pilas suelen emplearse en los siguientes contextos:

- Evaluación de expresiones en notación postfija (notación polaca inversa).
- Reconocedores sintácticos de lenguajes independientes del contexto
- Implementación de recursividad. La idea es utilizar esta estructura para eliminar los entornos recursivos que pueden crear los métodos recursivos.
- En la simulación de procesos donde los elementos organizan y se toman acorde a lo establecido en la estructura.

Las colas suelen emplearse en los siguientes contextos:

- Implementación del algoritmo de Búsqueda del Primero a lo Ancho (BFS)
- En la simulación de procesos donde los elementos organizan y se toman acorde a lo establecido en la estructura.

7. Ejercicios propuestos

A continuación una lista de ejercicios que se resuelven utlizando estas estructuras:

- DMOJ - Transformación: de A hacia B
- DMOJ - Street Parade
- DMOJ - Lavando los Platos
- DMOJ - Juego del Viajero Veloz
- DMOJ - Cotillón Vacuno
- DMOJ - Parejas Balanceadas