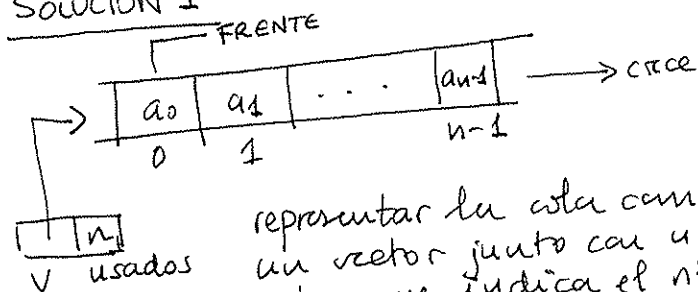


# LECCIÓN 10:

## COLAS usando vectores

### SOLUCIÓN 1



representar la cola como un vector junto con un entero que indica el n° de posiciones ocupadas.

### - COSTE de las OPERACIONES

- Inserción: si no tenemos que ampliar memoria es una operación elemental

$v[\text{usado}] = \text{nuevo\_elemento};$   
 $\text{usado}++;$

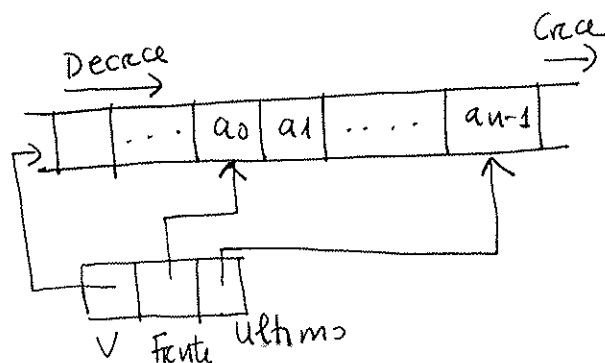
- Borrado: nos ocupa  $O(n)$  ya que debemos retroceder todos los elementos  $a_1, \dots, a_{n-1}$  una posición.

for ( $\text{int } i=1; i < \text{usado}; i++$ )  
 $v[i-1] = v[i];$

### CUESTIÓN: Usando vectores

¿Podemos implementar el borrado en  $O(1)$ ?

Primera Propuesta: Usando un vector junto con dos enteros que indican la posición del frente y el último.



### COSTE de las OPERACIONES

- Inserción: Se añade al final y se modifica la posición de último  $O(1)$

$v[\text{ultimo}+1] = \text{nuevo};$   
 $\text{ultimo}++;$

- Borrado: se incrementa en uno la posición del frente.  $O(1)$

$\text{frente}++;$

### Problema

Esta solución puede ser poco útil ya que inserciones y borrados de elementos provoca que los índices vayan avanzando y llegando a un momento que no tenga más espacio en el vector  $v$ .

SOLUCIÓN — Cuando no tengamos más capacidad situamos las próximas vacías al principio — VECTORES CIRCULARES

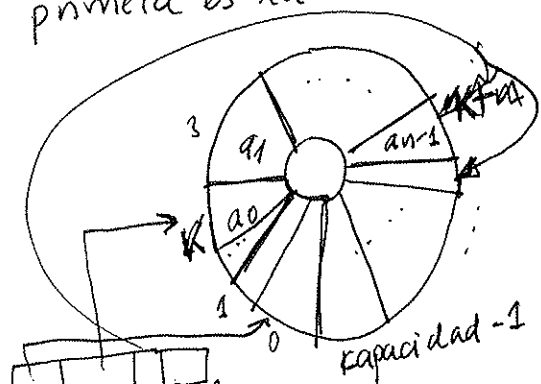
2

## LECCIÓN 10: continuación

### COLAS usando Vectores Circulares

#### VECTOR CIRCULAR

La posición siguiente a la última es la primera y la anterior a la primera es la última.



- El vector puede almacenar capacidad elementos

#### - OPERACIONES

- Insertar: Insertamos en la posición siguiente al último

$v[(ultimo+1) \% capacidad] = nuevo$

$ultimo = (ultimo+1) \% capacidad$   
 $n++$

Si no hubiere espacio tenemos que hacer un ~~resize~~antes de todo.

- Borrado: frente lo adelantamos en uno.

$frente = (frente+1) \% capacidad$   
 $n--$

- COLA Llena - capacidad = n

- COLA VACIA = n = 0

#### IMPLEMENTACIÓN

#include <COLA.H>

#define COLA\_H

template <class T>

class Cola {

private:

T \* datos;

int capacidad; // reservados

int n; // elementos almacenados

int primero, ultimo;

void resize(int tam);

void copiar(const Cola<T> &c);

public:

Cola(int tam=1);

Cola(const Cola<T> &c);

~Cola();

Cola<T> &operator=(const Cola<T> &c);

int size() const { return n; }

bool vacia() const { return n==0; }

bool llena() const { return n==capacidad; }

T Frente();

assert(n>0);

return datos[primero];

void Poner(const T &v);

void Quitar();

};

#include "cola.cpp";

#endif

3

LECCION 10: continuación

```
template <class T>
void Cola<T>::resize(int tam){
    T* aux = new T[tam];
    int minimo = (n < tam)? n : tam;
    for (int i=0; i<minimo; i++)
        aux[i] = datos[(i+primero)%capacidad];
```

```
    primero = 0;
    ultimo = minimo;
    n = minimo;
    capacidad = tam;
    delete [] datos;
    datos = aux;
}
```

template <class T>

```
void Cola<T>::Copiar(const Cola<T> &C){
    capacidad = C.capacidad;
    primero = C.primero;
    ultimo = C.ultimo;
    n = C.n;
    datos = new T[capacidad];
    for (int i=0; i<n; i++)
        datos[(i+primero)%capacidad] =
            C.datos[(i+primero)%capacidad];
}
```

template <class T>

```
Cola<T>::Cola(int tam){
    capacidad = tam;
    primero = 0; ultimo = 0;
    n = 0;
    datos = new T[capacidad];
}
```

3

```
template <class T>
Cola<T>::~Cola() {
    if (datos != 0)
        delete [] datos;
}
```

```
template <class T>
Cola<T> & Cola<T>::operator=
(const Cola<T> & C){
    if (this != & C){
        if (datos != 0)
            delete [] datos;
        Copiar(C);
    }
}
```

```
template <class T>
void Cola<T>::Poner(const T &v){
    if (n == capacidad)
        resize(2*reservados);

    datos[(ultimo+1)%capacidad]
        = v;
    ultimo = (ultimo+1)%capacidad;
    n++;
}
```

```
template <class T>
void Cola<T>::Quitar() {
    primero = (primero+1)%capacidad;
    n--;
    if (n < reservados/4)
        resize(reservados/2);
}
```

## Cola en la STL (Queue)

Librería  $\Rightarrow$  `#include <queue>`

clase  $\rightarrow$  `class queue`

Operaciones

`.empty`, `size`, `front`, `pop`, `push`, `swap`  
`emplace`, `swap`, `back`.

```
#include <queue>
```

```
using namespace std;
```

```
int main() {
```

```
    typedef pair<int, string> mipair;
```

```
    queue<mipair> miCola, otra;
```

```
    miCola.emplace(1, "Naranja");
```

```
    miCola.emplace(2, "Platano");
```

```
    cout << miCola.front() << " " <<  
          miCola.back() << " " <<  
          miCola.size();
```

```
    while (!miCola.empty()) {
```

```
        cout << miCola.front() << endl;
```

```
        miCola.pop();
```

```
    }
```

```
    mipair a(5, "Cinela");
```

```
    miCola.push(a);
```

```
    miCola.swap(otra);
```

```
}
```

EJERCICIO.- Dadas dos colas de pares de persona (dni) y año de nacimiento, crear una nueva cola cuyo año de nacimiento sea menor a uno dado. Los elementos que pasan deben estar en el mismo orden.

#include <queue>

using namespace std;

typedef pair<string, int> mipair;

queue<mipair> FiltrarYear(const queue<mipair> &q,  
int th-year){

queue<mipair> aux (q), c-out;

stack<mipair> p;

while (!aux.empty()){

if (aux.front().second > th-year){

p.push(aux.front());

aux.pop();

}

while (!p.empty()){

c-out.push(p.top());

p.pop();

}

return c-out;

}

..

..

..

..

..