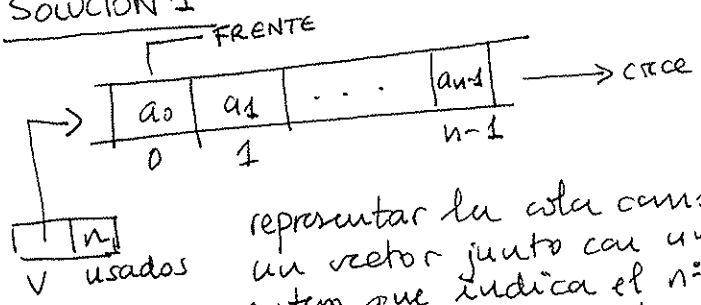
LECCIÓN 10:

COLAS usando vectores

SOLUCIÓN 1



representar la cola como un vector junto con un entero que indica el nº de posiciones ocupadas.

- COSTE de las OPERACIONES

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

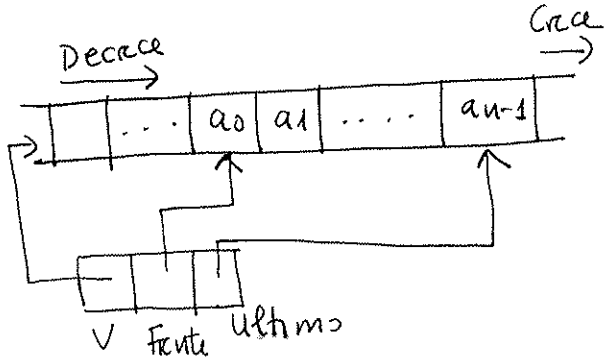
  ```
  v[usado] = nuevo_elemento;
  usado++;
  ```

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

  ```
  for (int i=1; i<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[ultimo+1] = nuevo;
ultimo++;
```

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

```
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.

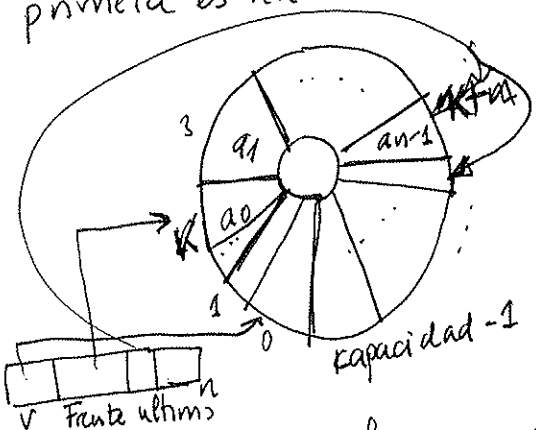
SOLUCIÓN — Cuando no tengamos más capacidad situamos las posiciones vacías al principio — VECTORES CIRCULARES

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 hubiese 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

```
#ifndef _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
```

LECCIÓN 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];
}
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
   =v;
   ultimo=(ultimo+1)%capacidad
   n++;
}
template <class T>
void Cola<T>::Quitar(){
   primero=(primero+1)%capacidad;
   n--;
   if(n<reservados*5/4)
      resize(reservados/2);
}
```