ESTRUCTURAS DE DATOS

TIPOS ABSTRACTOS DE DATOS LINEALES

Implementación del TAD Lista mediante arrays

Manuel Montenegro Montes
Departamento de Sistemas Informáticos y Computación
Facultad de Informática – Universidad Complutense de Madrid

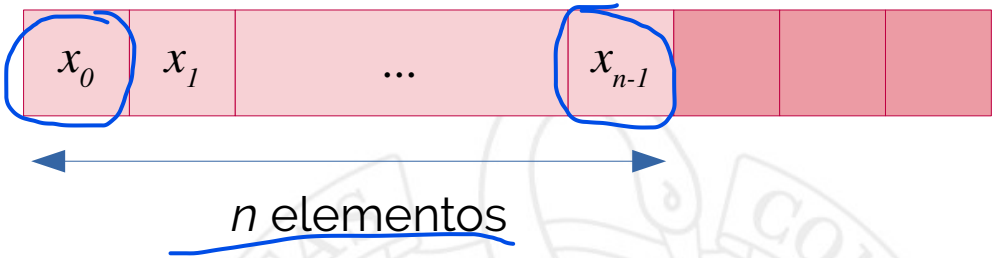
Recordatorio: operaciones del TAD Lista

Uno de los problemas de los arrays es que tienen una longitud limitada. Y una lista en principio podemos ir añadiendo elementos de uno en uno de manera indefinida

- **Constructoras:**
  - Crear una lista vacía: ***create_empty***() → *L:List*
- **Mutadoras:**
  - Añadir un elemento al principio de la lista: ***push_front***(*x: elem, L: List*).
  - Añadir un elemento al final de la lista: ***push_back***(*x: elem, L: List).*
  - Eliminar el elemento del principio de la lista: ***pop_front***(*L: List).*
  - Eliminar el elemento del final de la lista: ***pop_back***(*L: List).*
- **Observadoras:**
  - Obtener el tamaño de la lista: ***size***(*L: List*) → *tam: int.*
  - Comprobar si la lista es vacía ***empty***(*L: List*) → *b: bool.*
  - Acceder al primer elemento de la lista ***front***(*L: List*) → *e: elem.*
  - Acceder al último elemento de la lista ***back***(*L: List*) → *e: elem.*
  - Acceder a un elemento que ocupa una posición determinada ***at***(*idx: int, L: List*) → *e: elem.*

Implementación mediante arrays

$[\ x_0, x_1, \ldots, x_{n-1}\ ]$




```
class ListArray {
public:
    ...
private:
    int num_elems;
    std::string elems[MAX_CAPACITY];
};
```

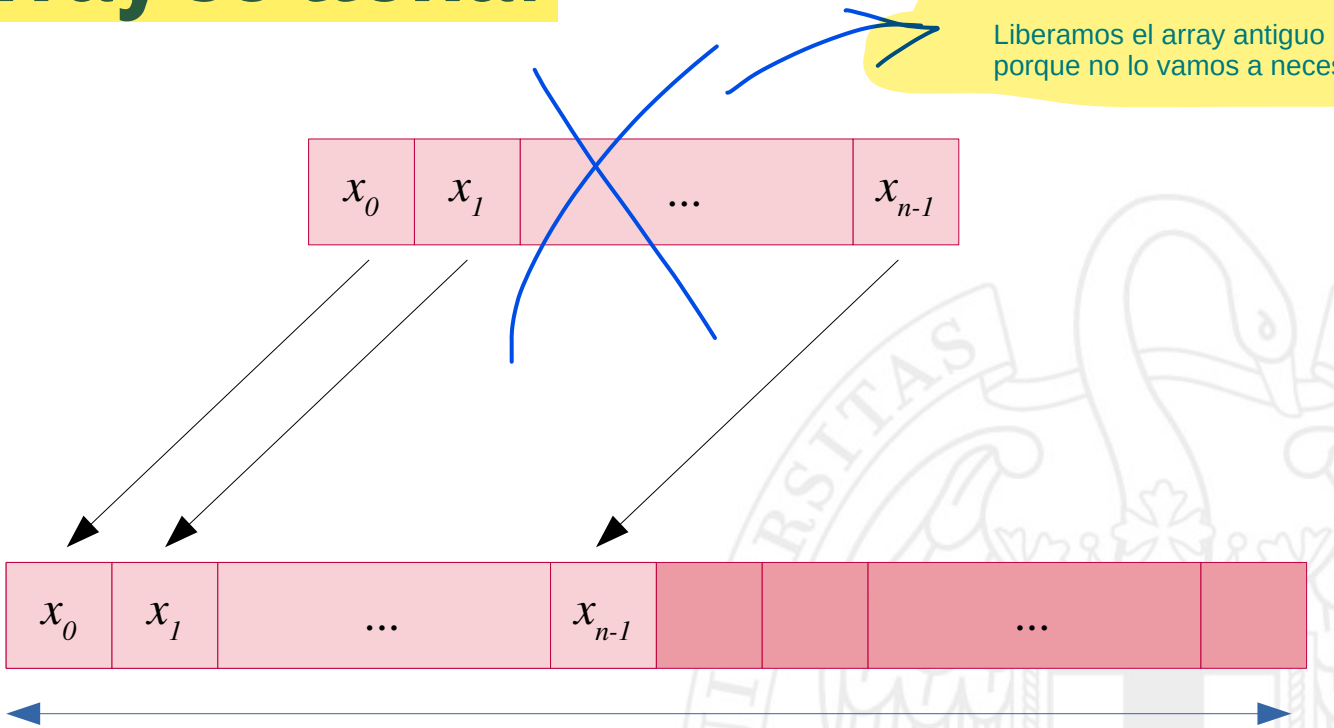
índice de la primera posición vacía.

array de string con finitas posiciones.

Esta es la primera forma que se nos ocurriría a nosotros o a cualquier persona normal, UTILIZANDO UN ARRAY.

¿Y si el array se llena?

Liberamos el array antiguo una vez redimensionado porque no lo vamos a necesitar.



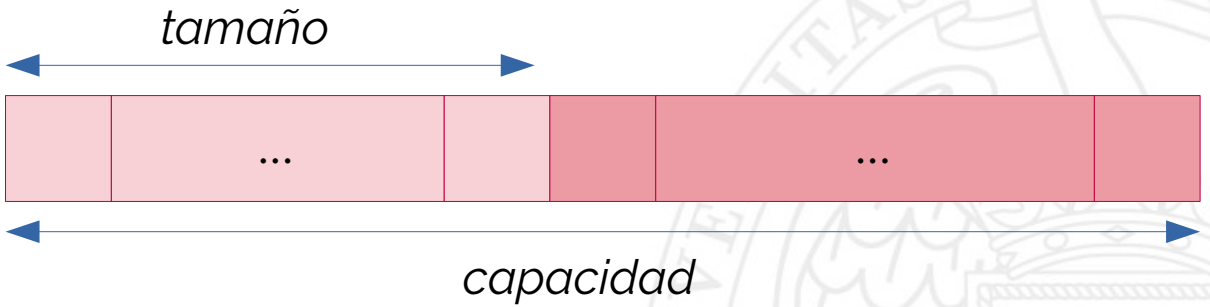
Nueva capacidad: $2 * n$

En este caso al doble de tamaño.

Se redimensiona el array con el doble de tamaño.

Tamaño vs. capacidad

- **Capacidad de una lista:** Tamaño del array que contiene los elementos.
- **Tamaño de una lista:** Número de posiciones ocupadas por elementos.
  - Siempre se cumple *tamaño ≤ capacidad*.



Es decir, el tamaño del array es el número de posiciones ocupadas por elementos, y la capacidad es el número de posiciones TOTALES del array, estén o no ocupadas.

Implementación con tamaño y capacidad

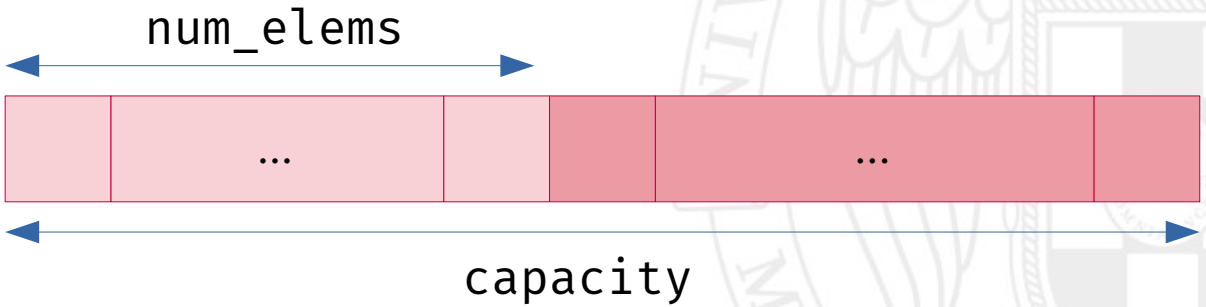
```
class ListArray {
public:
    ...
private:
    int num_elems;
    int capacity;
    std::string *elems;
};
```

el numero de posiciones ocupadas (o hasta que posición está ocupado el array.)

esto es la capacidad del array

Array reservado dinámicamente

puntero al array



Invariante y modelo de representación

```
class ListArray {
public:
 ...
private:
   int num_elems;
   int capacity;
   std::string *elems;
};
```

- Invariante de la representación:

  $I(x) = 0 \leq x.num\_elems \leq x.capacity$

  Menor o igual que la capacidad.

- Función de abstracción:

  $f(x) = [\ x.elems[0],\ x.elems[1],\ \ldots,\ x.elems[x.num\_elems - 1]\ ]$

Creación y destrucción de una lista

```
const int DEFAULT_CAPACITY = 10;

class ListArray {

public:
  ListArray(int initial_capacity)
      : num_elems(0), capacity(initial_capacity), elems(new std::string[capacity]) { }

  ListArray()
      : ListArray(DEFAULT_CAPACITY) { }

  ~ListArray() { delete[] elems; }

  …

private:
  int num_elems;
  int capacity;
  std::string *elems;
};
```

lista de inicialización se llamaba a esto.

inicializa el tamaño inicial a 0

capacidad

Constructor por defecto que asigna un valor por defecto a la capacidad.

destructor que libera el array elems que se ha creado.

Creación y destrucción de una lista

```
const int DEFAULT_CAPACITY = 10;

class ListArray {

public:
  ListArray(int initial_capacity = DEFAULT_CAPACITY)
     : num_elems(0), capacity(initial_capacity), elems(new std::string[capacity]) { }

  ListArray()
     : ListArray(DEFAULT_CAPACITY) { }


  ~ListArray() { delete[] elems; }

   …

private:
  int num_elems;
  int capacity;
  std::string *elems;
};
```

Valores por defecto para parámetros

Establecer por defecto un valor de la capacidad, en este caso es 10, de manera que no tenemos que utilizar el segundo constructor.

Añadir elementos al final de una lista

```
class ListArray {
public:
   void push_back(const std::string &elem);
   …
private:
  int num_elems;
  int capacity;
  std::string *elems;
};
```

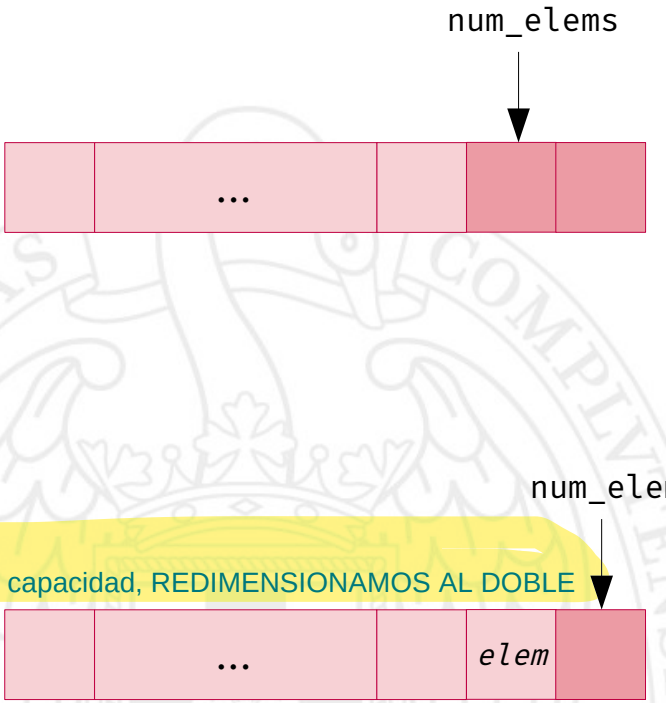
push_back = poner elemento al final de la lista.

```
void ListArray::push_back(const std::string &elem) {
  if (num_elems == capacity) {
    resize_array(capacity * 2);
  }

  elems[num_elems] = elem;
  num_elems++;
}
```

Si llenamos el array hasta el final de su capacidad, REDIMENSIONAMOS AL DOBLE

Primera posición vacía del array

coste lineal en el peor de los casos.



Ponemos el nuevo elemento en esa posición vacía y aumento el tamaño. Si llegamos a esa capacidad redimensionamos el array al doble del tamaño. En Java, TP2 acordarnos que se hacia utilizando Arrays.Copyof(array, array.size()*2)

Redimensionar el array

```
class ListArray {
public:
   …
private:
  int num_elems;
  int capacity;
  std::string *elems;

  void resize_array(int new_capacity);
};

void ListArray::resize_array(int new_capacity) {
  std::string *new_elems = new std::string[new_capacity];
  for (int i = 0; i < num_elems; i++) {
    new_elems[i] = elems[i];
  }

  delete[] elems;
  elems = new_elems;
  capacity = new_capacity;
}
```

puntero a un nuevo array de capacidad = new_cappacity

al nuevo array le asignamos todos los elementos del array antiguo y redimensionado al doble de tamaño, o al tamaño que sea, en este caso new_capacity.

destruimos el array antiguo y elems pasa a valer lo que el nuevo array que hemos creado

capacidad es la nueva capacidad.

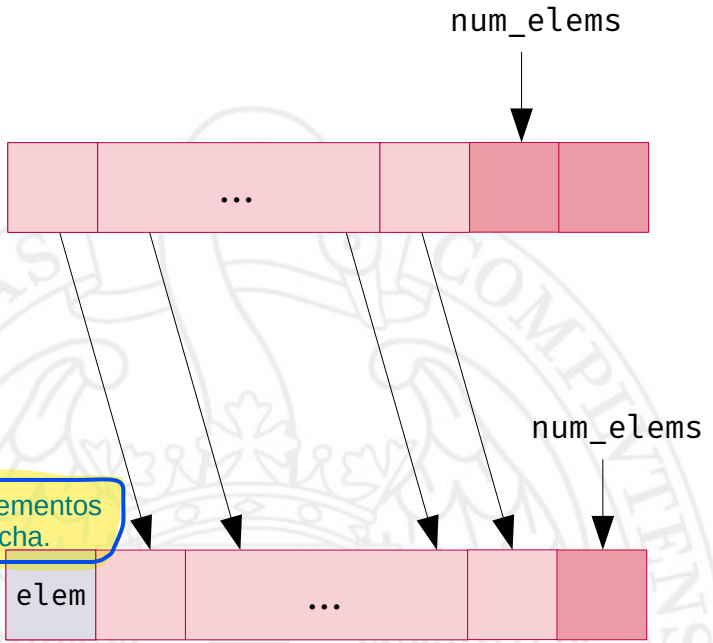
Añadir elementos al principio de una lista

```
class ListArray {
public:
  void push_front(const std::string &elem);
  …
};


void ListArray::push_front(const std::string &elem) {
  if (num_elems == capacity) {
    resize_array(capacity * 2);
  }

  for (int i = num_elems - 1; i ≥ 0; i--) {
    elems[i + 1] = elems[i];
  }
  elems[0] = elem;
  num_elems++;
}
```

desplazar todos los elementos una posición a la derecha.

asignamos el nuevo elemento a la primera posición

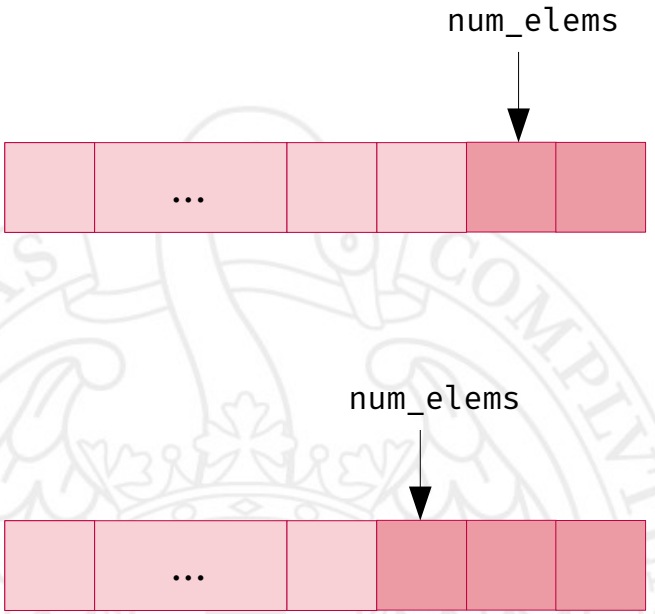
Coste también lineal



Eliminar elementos del final de una lista

```
class ListArray {
public:
  void pop_back();
  …
};


void ListArray::pop_back() {
  assert (num_elems > 0);
  num_elems--;
}
```

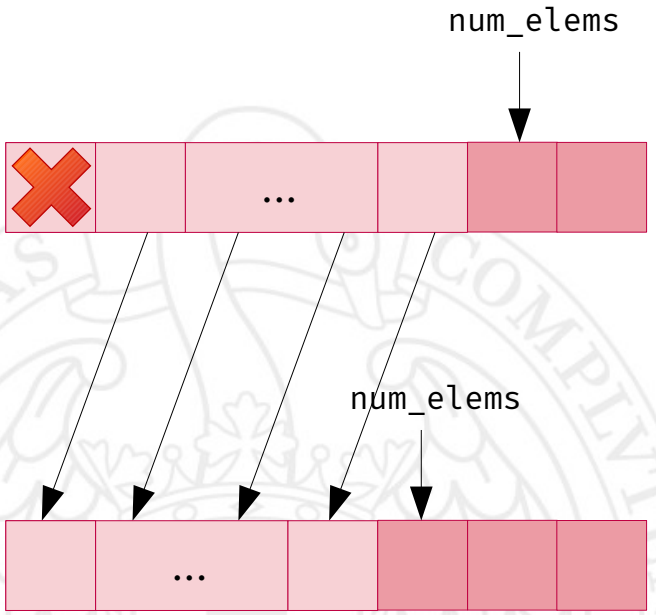
solo tenemos que decrementar num_elems



Eliminar elementos del principio de una lista

```
class ListArray {
public:
  void pop_front();
  …
};


void ListArray::pop_front() {
  assert (num_elems > 0);

  for (int i = 1; i < num_elems; i++) {
    elems[i - 1] = elems[i];
  }
  num_elems--;
}
```

coste lineal respecto al número de elementos del array.



Movemos todos los elementos a la izquierda para que el primer elemento quede fuera del array y decrementamos el número de elementos de nuestro array.

Funciones observadoras (1)

```
class ListArray {
public:

  int size() const {
    return num_elems;
  }

  bool empty() const {
    return num_elems == 0;
  }

  std::string front() const {
    assert (num_elems > 0);
    return elems[0];
  }
  ...
};
```

devuelve el número de elementos que tiene, es decir su TAMAÑO, NO LA CAPACIDAD.

COSTE CONSTANTE.

Devuelve el primer elemento de una lista.

devuelve falso en caso contrario.

Funciones observadoras (2)

```
class ListArray {
public:

  std::string back() const {
    assert (num_elems > 0);
    return elems[num_elems - 1];
  }

  std::string at(int index) const {
    assert (0 <= index && index < num_elems);
    return elems[index];
  }
  ...
};
```

Devuelve el último elemento del array

devuelve un elemento que ocupa una posición concreta (index) en el array.

Mostrar el contenido de una lista

```
class ListArray {
public:
  void display() const;
  …
};
```

```
void ListArray::display() const {
  std::cout << "[";
  if (num_elems > 0) {
    std::cout << elems[0];
    for (int i = 1; i < num_elems; i++) {
      std::cout << ", " << elems[i];
    }
  }
  std::cout << "]";
}
```

Muestra el contenido de la lista.

coste lineal con respecto al número de elementos de la lista.

Prueba de ejecución

```
int main() {
  ListArray l;
  l.push_back("David");
  l.push_back("Maria");
  l.push_back("Elvira");
  l.display(); std::cout << std::endl;

  std::cout << "Elemento 1: " << l.at(1) << std::endl;

  l.pop_front();
  l.display(); std::cout << std::endl;

  return 0;
}
```

Introduces 3 elementos

elemento en la posición 1

elimina el primer elemento

```
[David, Maria, Elvira]
Maria
[Maria, Elvira]
```