ESTRUCTURAS DE DATOS LINEALES

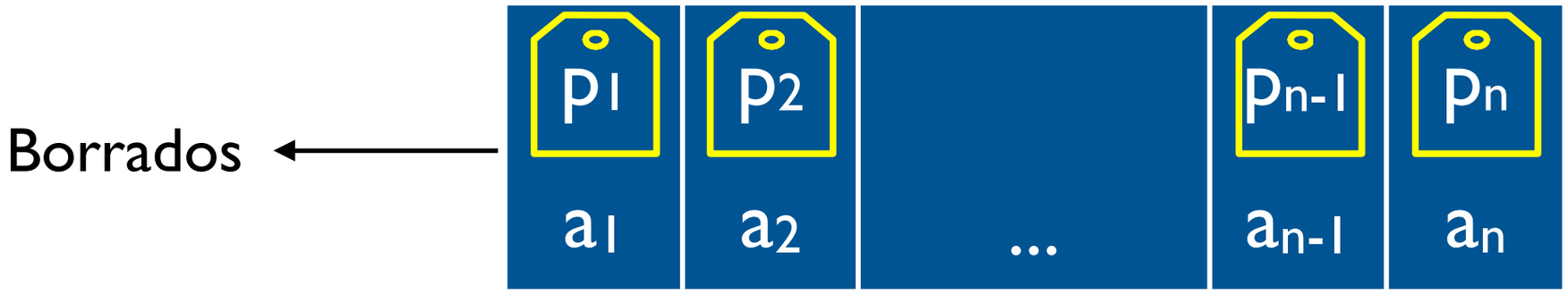
COLAS CON PRIORIDAD

Joaquín Fernández-Valdivia
Javier Abad
Dpto. de Ciencias de la Computación e Inteligencia Artificial
Universidad de Granada

DECSAI

Colas con prioridad

- Una cola con prioridad es una estructura de datos lineal diseñada para realizar accesos y borrados en uno de sus extremos (frente). Las inserciones se realizan en cualquier posición, de acuerdo a un valor de prioridad.



Colas con prioridad

1. **Especificación**:

- Contienen una secuencia de valores especialmente diseñadas para realizar accesos y borrados por el frente.
- A diferencia de las colas normales, las inserciones se realizan en cualquier punto de acuerdo a un criterio de prioridad.
- Cada dato que se guarda en la cola con prioridad debe componerse del dato y su prioridad.

Colas con prioridad

2. **Operación:**
   - Frente: devuelve el elemento en el frente
   - Prioridad: devuelve la prioridad del elemento en el frente
   - Quitar: elimina el elemento que está en el frente. Este es el más prioritario.
   - Vacía: indica si la cola está vacía
   - Poner: inserta un nuevo elemento de acuerdo a su prioridad
3. **Implementación**:
   - Al igual que con las colas simples podríamos plantearnos diferentes estructuras de datos: vector dinámico, vectores circulares, celdas enlazadas.

Colas con prioridad

Esquema de la interfaz

```
#ifndef __COLA_PRI_
#define __COLA_PRI__

class ColaPri{

private:
  ...      //La implementación que se elija

public:
  ColaPri();
  ColaPri(const ColaPri& c);
  ~ColaPri();
  ColaPri& operator=(const ColaPri& c);

  bool vacia() const;
  Tbase frente() const;
  Tprio prioridad_frente() const;
  void poner(Tbase e, Tprio prio);
  void quitar();
};

#endif /* ColaPri_hpp */
```

&

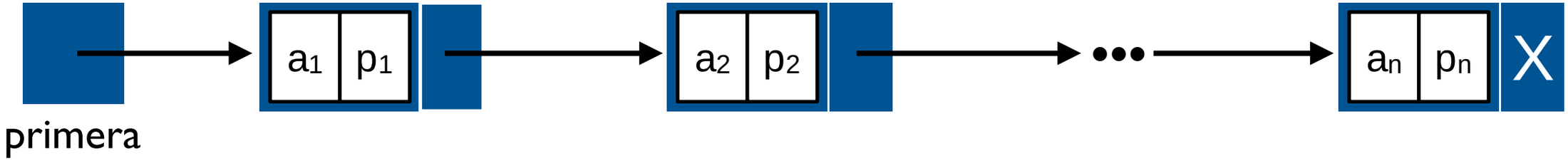
Colas con prioridad

Uso de una cola

```
#include <iostream>
#include "ColaPri.hpp"
using namespace std;
int main(){
  ColaPri c;
  int nota;
  string dni;

  cout<< "Escriba  una nota: ";
  cin >> nota;
  while(nota>=0  && nota<=10){
    cout << "Escribaun  dni: ";
    cin >> dni;
    c.poner(dni, nota);
    cout << "Escriba una nota: ";
    cin >> nota;
  }
  cout << "DNIs ordenados por nota:" << endl;
  while(!c.vacia()){
    cout << "DNI: " << c.frente() << " Nota: "
         << c.prioridad_frente() << endl;
    c.quitar();
  }
  return 0;
}
```

Colas con prioridad. Celdas enlazadas

Almacenamos la secuencia de parejas en celdas enlazadas



- Una cola vacía contiene un puntero nulo
- El frente de la cola está en la primera celda (muy eficiente)
- Si borramos el frente, eliminamos la primera celda
- En la inserción tenemos que buscar la posición según su prioridad

Colas con prioridad. Celdas enlazadas

- Estructura *info*

```
1  template <class Tprio, class T>
2  struct info {
3        Tprio prio;
4        T elemento;
5  };
```

- El campo *prio* será la prioridad del dato (que se almacena en el campo elemento).
- Ambos campos son templates, en este caso hemos usado dos templates para que el usuario tenga la libertad de definir la prioridad con un tipo diferente al tipo del dato.

Colas con prioridad. Celdas enlazadas

- Por ejemplo se podrían hacer las siguiente instanciaciones:

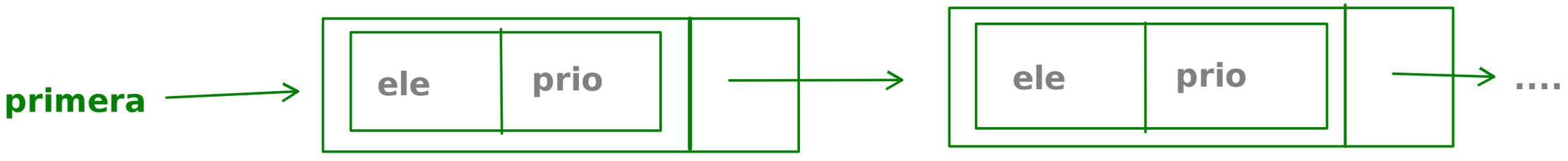
```
1  ...
2  info<int,int> a; //la prioridad int y el elemento int
3  info<char,int> b; //la prioridad char y el elemento int
4  ...
```

- El único detalle para poder usar la cola con prioridad instanciando la prioridad a un tipo concreto es que el usuario deberá ser consciente de que para establecer la prioridad el tipo al que se instancie *Tprio* debe tener definido el operador relacional menor (<).

ColaPri.h

```
#ifndef __COLA_PRI__
#define __COLA_PRI__

#include <string>
using namespace std;
typedef int Tprio;
typedef string Tbase;

struct Pareja{
  Tprio prioridad;
  Tbase elemento;
};

struct CeldaColaPri{
  Pareja dato;
  CeldaColaPri* sig;
};
```

```
class ColaPri{
private:
  CeldaColaPri* primera;
public:
  ColaPri();
  ColaPri(const ColaPri& c);
  ~ColaPri();
  ColaPri& operator=(const ColaPri& c);

  bool vacia() const;
  Tbase frente() const;
  Tprio prioridad_frente() const;
  void poner(Tbase e, Tprio prio);
  void quitar();
};

#endif /* ColaPri_hpp */
```



ColaPri.cpp

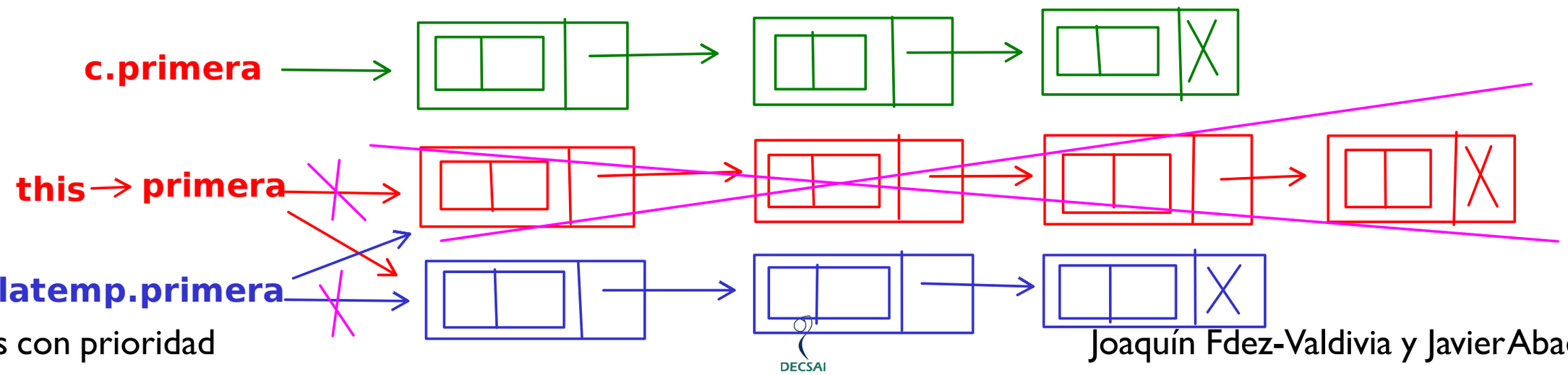
```
#include <cassert>
#include <utility>
#include "ColaPri.hpp"

ColaPri::ColaPri(){
  primera = 0;
}

ColaPri::ColaPri(const ColaPri& c){
  if(c.primera==0) //Si está vacía
    primera = 0;
  else{              //Si no está vacía
    primera = new CeldaColaPri;        //Crea la primera celda
    primera->dato = c.primera->dato; //Copia el dato
    CeldaColaPri* src = c.primera;   //Inicializa punteros
    CeldaColaPri* dest= primera;
    while(src->sig!=0){                //Mientras queden celdas
      dest->sig = new CeldaColaPri;  //Crea nueva celda
      src =src->sig;                   //Avanza punteros
      dest= dest->sig;
      dest->dato = src->dato;          //Copia el dato
    }
    dest->sig = 0;                     //Ajusta el puntero de la última
  }
}
```

ColaPri.cpp

```
ColaPri::~ColaPri(){
  CeldaColaPri* aux;
  while(primera != 0){          //Mientras queden celdas
    aux = primera;              //Referencia a la primera celda
    primera=primera->sig;       //Avanza primera
    delete aux;                 //Borra la celda
  }
}
```

```
ColaPri& ColaPri::operator=(const ColaPri &c){
  ColaPri colatemp(c);     //Usamos el constructor de copia
  swap(this->primera, colatemp.primera);
  return *this;
  //El destructor libera el contenido de *this
}
```

```
bool ColaPri::vacia() const{
  return (primera==0);
}
```



ColaPri.cpp

```
Tbase ColaPri::frente()const{
   assert(primera!=0);
   return (primera->dato.elemento);
}

Tprio ColaPri::prioridad_frente() const{
   assert(primera!=0);
   return(primera->dato.prioridad);
}

void ColaPri::quitar(){
   assert(primera!=0);
   CeldaColaPri* aux = primera;
   primera = primera->sig;
   delete aux;
}
```

ColaPri.cpp

```
void ColaPri::poner(Tbase e, Tprio prio){
  CeldaColaPri* aux = new CeldaColaPri;   //Creamos una nueva celda
  aux->dato.elemento=e;                   //Guardamos la información
  aux->dato.prioridad= prio;
  aux->sig = 0;
  if (primera==0)                         //Si la cola está vacía
     primera = aux;
  else if(primera->dato.prioridad<prio){  //Si no está vacía y tiene
                                          //prioridad máxima
     aux->sig=primera;                    //La insertamos la primera
     primera = aux;
  }
  else{                                   //Caso general
     CeldaColaPri* p = primera;
    while(p->sig!=0){                     //Avanza por las celdas
      if(p->sig->dato.prioridad<prio){
          aux->sig = p->sig;
          p->sig = aux;
          return;
       }
       else p = p->sig;
    }
    p->sig = aux;
  }
}
```

c.poner (z, 19); c.poner (r,8); c.poner (w,1)

Colas con prioridad

Uso de una cola
STL

```
#include <iostream>
#include <queue>
using namespace std;
int main(){
   priority_queue<Pareja> c;
   Pareja p;
   cout<< "Escriba  una nota: ";
   cin >> p.nota;
   while(p.nota >=0 && p.nota <=10){
      cout << "Escriba un dni: ";
      cin >> p.dni;
      c.push(p);
      cout << "Escriba una nota: ";
      cin >> p.nota;
   }
   cout << "DNIs ordenados por nota:" << endl;
   while(!c.empty()){
      p = c.top();
      cout << "DNI: " << c.top().dni << " Nota: "
      << c.top().nota << endl;
      c.pop();
   }
   return 0;
}
```

```
structPareja{
   int nota;
   string dni;
   bool operator<(const struct Pareja & otra) const{
      return (this->nota < otra.nota);
   }
};
```

Colas con prioridad

Uso de una cola
STL

```
#include <iostream>
#include <queue>
using namespace std;

int main(){
   priority_queue<pair<int, string>> c;
   pair<int, string> p;

   cout<< "Escriba  una nota: ";
   cin >> p.first;
   while(p.first >=0 && p.first <=10){
      cout << "Escriba undni:  ";
      cin >> p.second;
      c.push(p);
      cout<< "Escriba  una nota: ";
      cin >> p.first;
   }
   cout << "DNIs ordenados por nota:" << endl;
   while(!c.empty()){
      p = c.top();
      cout << "DNI: " << c.top().second << " Nota: "
      << c.top().first << endl;
      c.pop();
   }
   return 0;
}
```

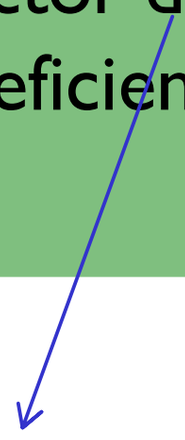
Ejercicio propuesto

- Desarrollar una clase cola con prioridad genérica usando templates
- Podríamos pensar en desarrollar la clase patrón usando dos parámetros, uno para el tipo base y otro para la prioridad
- Sin embargo, el enfoque más cómodo y versátil, tanto para el desarrollador como para el usuario de la clase, es seguir el enfoque de la STL: dejar en manos del usuario de la clase la definición del tipo base, al que sólo se le exige que tenga definido el operador <
- De esta forma, podemos usar la cola para almacenar valores de cualquier tipo simple (enteros, caracteres…), parejas valor-prioridad (como en el ejemplo anterior) o cualquier tipo/clase más complejo que tenga implementado el operador <

Eficiencia

- Como se puede observar la eficiencia de las funciones son constantes a excepción de *Poner* que realiza una búsqueda secuencial para establecer donde realizar la inserción.

- Usando una representación en la que tuviésemos acceso directo a cada elemento, como por ejemplo en un vector dinámico, podríamos realizar una búsqueda binaria, obteniendo una eficiencia en el peor de los casos de *Poner* de O(log2(n)).



Arboles parcialmente ordenados