# COLAS CON PRIORIDADES VARIABLES



**ALBERTO VERDEJO** 

# Colas de prioridad con prioridades variables

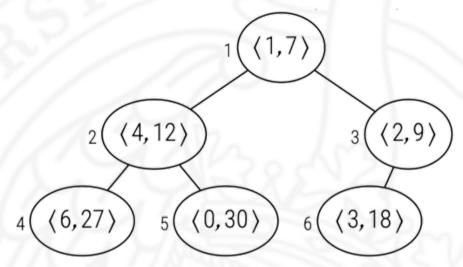
- ► En ocasiones queremos poder referirnos a elementos que ya están en la cola de prioridad para cambiarles su prioridad.
- Lo más sencillo es asociar un número distinto a cada elemento y utilizar ese *índice* para referirnos a él.
- Suponemos que el número de elementos a los que nos vamos a querer referir es fijo, N, y que los elementos están identificados con los índices de 0 a N-1.

#### IndexPQ

La clase IndexPQ<T> cuenta con las siguientes operaciones:

- crear una cola de prioridad vacía, IndexPQ(int N)
- ▶ insertar un elemento, void push(int e, T const& p)
- modificar la prioridad de un elemento, void update(int e, T const& p)
- consultar el elemento más prioritario, Par const& top() const
- eliminar el primer elemento, void pop()
- determinar si la cola de prioridad es vacía, bool empty() const
- consultar el número de elementos de la cola, int size() const

## Representación



posiciones



6



```
// T es el tipo de las prioridades
// Comparator dice cuándo un valor de tipo T es más prioritario que otro
template <typename T, typename Comparator = std::less<T>>
class IndexPQ {
public:
   // registro para las parejas < elem, prioridad >
   struct Par { int elem; T prioridad; };
private:
  // vector que contiene los datos (pares < elem, prio >)
   std::vector<Par> array;  // primer elemento en la posición 1
   // vector que contiene las posiciones en array de los elementos
   std::vector<int> posiciones; // un 0 indica que el elemento no está
   // Objeto función que sabe comparar prioridades.
   // antes(a,b) es cierto si a es más prioritario que b
   Comparator antes:
```



```
public:
  IndexPQ(int N, Comparator c = Comparator()) :
             array(1), posiciones(N, 0), antes(c) {}
  Par const& top() const {
     if (size() == 0)
         throw std::domain_error(
                  "No se puede consultar el primero de una cola vacia");
      else
         return array[1];
```



```
public:
  void push(int e, T const& p) {
      if (posiciones.at(e) != 0)
         throw std::invalid_argument(
                          "No se pueden insertar elementos repetidos.");
     else {
         array.push_back({e, p});
         posiciones[e] = size();
         flotar(size());
```



```
private:
void flotar(int i) {
  Par parmov = array[i];
  int hueco = i;
  while (hueco != 1 && antes(parmov.prioridad, array[hueco/2].prioridad)) {
      array[hueco] = array[hueco/2];
      posiciones[array[hueco].elem] = hueco;
      hueco /= 2;
  array[hueco] = parmov;
  posiciones[array[hueco].elem] = hueco;
```



```
public:
  void pop() {
      if (size() == 0)
          throw std::domain_error(
                  "No se puede eliminar el primero de una cola vacía.");
      else {
         posiciones[array[1].elem] = 0; // para indicar que no está
         if (size() > 1) {
            array[1] = array.back(); array.pop_back();
            posiciones[array[1].elem] = 1;
            hundir(1);
         } else
            array.pop_back();
```



```
private:
void hundir(int i) {
    Par parmov = array[i];
    int hueco = i;
    int hijo = 2*hueco; // hijo izquierdo, si existe
    while (hijo <= size()) {</pre>
       // cambiar al hijo derecho de i si existe y va antes que el izquierdo
       if (hijo < size() && antes(array[hijo + 1].prioridad, array[hijo].prioridad))</pre>
          ++hijo;
       // flotar el hijo si va antes que el elemento hundiéndose
       if (antes(array[hijo].prioridad, parmov.prioridad)) {
          array[hueco] = array[hijo];
          posiciones[array[hueco].elem] = hueco;
          hueco = hijo; hijo = 2*hueco;
       else break;
    array[hueco] = parmov; posiciones[array[hueco].elem] = hueco;
```



```
public:
  void update(int e, T const& p) {
      int i = posiciones.at(e);
      if (i == 0) // el elemento e se inserta por primera vez
        push(e, p);
      else {
         array[i].prioridad = p;
         if (i != 1 && antes(array[i].prioridad, array[i/2].prioridad))
            flotar(i);
         else // puede hacer falta hundir a e
            hundir(i);
```