TDA VECTOR DINAMICO

```
/**
 * @file vector_dinamico.h
 * @brief fichero cabecera del TDA vector dinamico
 *
 * se crea un vector con capacidad de crecer y decrecer
 *
 */
#ifndef _vectorDinamico_h
#define _vectorDinamico_h

/**
 * @brief TDA Vector_Dinamico
 *
 * Una instancia de v del tipo de dato abstracto
 * de vector_Dinamico sobre el tipo de float es un
 * array 1-D de un determinado tamaño de n, que
 * puede crecer y decrecer a peticion del usuario.
 * Lo podemos representar como:
 *
 *          {v[0], v[1], ..., v[n-1]}
 *
 * donde
 * v[i] es el valor almacenado en la posicion i del vector
 *
 * La eficiencia en espacio es de O(n).
 *
 * Un ejemplo de uso puede verse en:
 * @include ejemplo_vector_dinamico.cpp
 *
 */
```

```
class Vector_Dinamico {
  private:
    /**
     * @page rep Vector_Dinamico Rep del TDA Vector_Dinamico
     *
     * @section inv Vector_Dinamico Invariante de Representacion.
     *
     * Un objeto válido @e r del TDA Vector_Dinamico debe cumplir
     * - @c r.nelementos >= 0
     * - @c r.datos apunta a una zona de memoria con
     *   capacidad para albergar @c nelementos valores
     *   de tipo @c float
     *
     * @section fa Vector_Dinamico Función de abstracción
     *
     * Un objeto válido @e rep del TDA Vector_Dinamico
     * representa al vector de tamaño @e n
     *
     *   {v.datos[0], v.datos[1], ...., v.datos[v.nelementos-1]}
     *
     */
    float * datos; /**< Apunta a los elementos del vector */
    int nelementos; /**< Indica el número de elementos en
                         @c datos */
```

```
public:
    //--------- Constructores ---------

/**
 * @brief  Constructor por defecto
 * @param  n indica el número de componentes inicial
 *           reservados para el vector
 * @note   Este constructor tambien corresponde al de por defecto
 */
Vector_Dinamico (int n=0);
Vector_ Dinamico (const vector_Dinamico & original);

    //--------- Destructor ---------

~vector_Dinamico();

    //---------- Otras funciones ------

/**
 * @brief Numero de componentes del vector
 * @return Devuelve el número de componentes que puede
 *         almacenar en cada instante el vector
 * @see   resize( )
 */
int size() const;

/**
 * @brief  Acceso a un elemento
 * @param  i es la posicion del vector donde está
 *         el elemento.  0<=i< size()
 * @return Devuelve la referencia al elemento. Por
 *         tanto puede usarse para almacenar un
```

```
     *          valor en esa posicion.
     */
    float & operator [] (int i);

    /**
     * @brief Acceso a un elemento de un vector constante
     * @param i es la posicion del vector donde está
     *          el elemento. 0 <= i < size()
     * @return Devuelve la referencia al elemento. Se supone
     *          que el vector no se puede modificar y por
     *          tanto es acceso solo de lectura.
     */
    const float & operator [] (int i) const;

    /**
     * @brief Redimension del vector
     * @param n es el nuevo tamaño del vector. n>=0
     * @post Los valores almacenados antes de la
     *          redimension no se pierden (excepto los que
     *          se salen del nuevo rango de índices)
     */
    void resize (int n);

    /**
     * @brief Operador de asignacion
     */
    Vector_Dinamico & operator = (const Vector_Dinamico & original);
};
#endif /* _vectorDinamico_h */
```

Ejemplo: Clase vector dinámico (1/2).

vector_dinamico.h

```
#ifndef _vectorDinamico_h
#define _vectorDinamico_h

class Vector_Dinamico {
  private:
    float * datos;
    int nelementos;
  public:
    // ————————  Constructores  ————————
    Vector_Dinamico(int n[redacted]);
    Vector_Dinamico(const Vector_Dinamico& original);
    // ————————  Destructor  ————————
    ~Vector_Dinamico();
    // ————————  Otras funciones  ————————
    int size() const;
    float& operator[] (int i);
    const float& operator[] (int i) const;
    void resize(int n);
    Vector_Dinamico& operator=
                    (const Vector_Dinamico& original);
};

#endif                                  /* _vectorDinamico_h */
```

vector_dinamico.cpp

```
#include <cassert>
#include <vector_dinamico.h>

Vector_Dinamico::Vector_Dinamico(int n[redacted])
{
  assert(n>=0);
  if (n>0)
    datos= new float[n];
  nelementos= n;
}
/* ———————————————————————————————— */

Vector_Dinamico::Vector_Dinamico
            (const Vector_Dinamico& original)
{
  nelementos= original.nelementos;
  if (nelementos>0) {
    datos= new float[nelementos];
    for (int i=0; i<nelementos;++i)
      datos[i]= original.datos[i];
  }
  else datos=0;
}
```

Ejemplo: Clase vector dinámico (2/2).

vector_dinamico.cpp

```
Vector_Dinamico::~Vector_Dinamico()
{ if (nelementos>0) delete[] datos; }

int Vector_Dinamico::size() const
{ return nelementos; }

float& Vector_Dinamico::operator[] (int i)
{
  assert (0<=i && i<nelementos); return datos[i];
}

const float& Vector_Dinamico::operator[] (int i) const
{
  assert (0<=i && i<nelementos); return datos[i];
}

Vector_Dinamico& Vector_Dinamico::operator=
              (const Vector_Dinamico& original)
{
  if (this! = &original) {
    if (nelementos>0) delete[] datos;
    nelementos= original.nelementos;
    datos= new float[nelementos];
    for (int i=0; i<nelementos;++i)
      datos[i]= original.datos[i];
  }
  return *this;
}
```

vector_dinamico.cpp

```
void Vector_Dinamico::resize(int n)
{
  assert (n>=0);
  if (n! =nelementos) {
    if (n! =0) {
      float * nuevos_datos;
      nuevos_datos= new float[n];
      if (nelementos>0) {
        int minimo;
        minimo= nelementos<n?nelementos:n;
        for (int i= 0; i<minimo;++i)
          nuevos_datos[i]= datos[i];
        delete[] datos;
      }
      nelementos= n;
      datos= nuevos_datos;
    }
    else {
      if (nelementos>0)
        delete[] datos;
      datos= 0;
      nelementos= 0;
    }
  }
}
```

```
#include <iostream>                    /*Ejemplo de uso*/
#include <vector_dinamico.h>
using namespace std;
void cargar_indices (vector_Dinamico & V)
    {
     for (int i=0; i<V.size(); ++i)
          V[i]=i;
    }
float maximo (const Vector_Dinamico & V)
    {
     float max;
     if (V.size()==0) {
          cerr << "Uppsss máximo de ???? asignamos cero" <<endl;
          max=0.0;
     }
     else {
          max=V[0];
          for (int i=1; i<V.size(); ++i)
              if (max<V[i])
                  max=V[i];
          }
     return max;
    }
int main()
   {
    vector_Dinamico vec;
```

```
cargar_indices (vec);
cout << " Maximo de " << vec.size() << "elementos: "
     << maximo (vec ) << endl;
vec.resize (10);
cargar_indices (vec);
cout << " Maximo de" << vec.size() << " elementos:"
     << maximo (vec) << endl;
return 0;
}
```

TDA CONJUNTO DE REALES

```
/**
 * @file  conjunto_reales.h
 * @brief Fichero cabecera del TDA Conjunto_Reales
 *
 */
#ifndef _conjunto_reales_h
 #define _conjunto_reales_h
#include <vector_dinamico.h>
#include <cassert>

/**
 * @brief TDA Conjunto_Reales
 *
 * Una instancia de c del tipo de dato abstracto
 * c Conjunto_Reales es un conjunto de números de tipo fload.
 *
 * El número de elementos del conjunto se denomina cardinal
 * o tamaño del conjunto. Un conjunto de tamaño cero se
 * denomina vacío.
 * Lo podemos representar como
 *      {e1, e2, e3, ..., en-1, en}
 * donde n es el número de elementos del conjunto.
 * La eficiencia en espacio es de O(n)
 *
 */
```

```
class Conjunto_Reales {
  private:
    /**
     * @page repConjunto_Reales Rep del TDA Conjunto_Reales
     *
     * @section invConjunto_Reales Invariante de Representacion
     *
     * Un objeto válido de rep del TDA Conjunto_Reales debe cumplir
     * - @c rep.v.size() >= rep.nelementos
     * - @c rep.nelementos >=0
     * - \c rep.v[i] < rep.v[j] para todo \e i,j tal que
     *   \e \c 0 <= i < j < rep.nelementos
     * @section faConjuntoReales Función de abstracción
     *
     * Un objeto válido de rep del TDA Conjunto_Reales
     * representa al vector {rep.v[0], ...., rep.v[rep.nelementos]}
     *
     */
  Vector_Dinamico v; /**< Almacena los elementos del conjunto */
  int nelementos; /**< Número de posiciones de \c v usadas */

   /**
    * @brief Localizador de una posición en @c v
    *
    * @param val es el valor del elemento a localizar en la matriz.
    * @retval pos, posición donde se encuentra el @a valor
    *         (si está) ó la posición donde debería insertarse (si no está)
    * @return si el valor @a val está en el vector devuelve true
    * @note La eficiencia es logarítmica (usa búsqueda binaria)
    */
  bool posicion_elemento(int & pos, float val) const;
```

```
public:
    Conjunto_Reales() : nelementos(0) { }
    // Conjunto_Reales (const Conjunto_Reales & c);
    // ~Conjunto_Reales ();
    // Conjunto_Reales & operator = (const Conjunto_Reales & c);

/**
 * @brief Añadir un elemento
 * @param f valor a insertar en el conjunto
 * @return true si el número de elementos ha aumentado y
 *         false si el elemento ya estaba en el conjunto.
 *
 */
bool insertar (float f);

/**
 * @brief Eliminar un elemento
 * @param f valor a eliminar del conjunto
 * @return true si el número de elementos ha disminuido y
 *         false si el elemento no estuviera en el conjunto
 *
 */
bool borrar (float f);

  /**
   * @brief Consultar la existencia de un elemento
   * @param f valor a consultar en el conjunto
   * @return true si el elemento está en el conjunto, false
   *         en caso contrario
   *
   */
bool pertenece (float f) const { int pos;
    return posicion_elemento (pos, f); }
```

```
/**
 * @brief Valor del i-ésimo elemento
 *
 * @param i indica el elemento del conjunto que queremos obtener
 * @pre 0<=i<size()
 * @return devuelve el valor del i-ésimo elemento
 */
float elemento(int i) const {assert(0<=i && i<=v.size());
                             return v[i]; }

/**
 * @brief Conjunto vacio
 * @return true si el conjunto esta vacio (\c size()==0)
 */
bool vacio() const { return nelementos==0; }

/**
 * @brief Cardinal del conjunto
 * @return Devuelve el número de elementos del conjunto
 */
int size() const { return nelementos; }

};

#endif /* _conjunto_reales_h */
```

Ejemplo: Clase Conjunto_Reales (1/2).

conjunto_reales.h

```
#ifndef _conjunto_reales_h
#define _conjunto_reales_h

#include <vector_dinamico.h>
#include <cassert>

class Conjunto_Reales {
  private:
    Vector_Dinamico v;
    int nelementos;
    bool posicion_elemento(int& pos, float val) const;
  public:
    Conjunto_Reales(): nelementos(0) {}
    // Conjunto_Reales(const Conjunto_Reales& c);
    // ~Conjunto_Reales();
    // Conjunto_Reales& operator=
    //                  (const Conjunto_Reales& c);
    bool insertar(float f);
    bool borrar(float f);
    bool pertenece(float f) const
      { int pos; return posicion_elemento(pos,f); }
    float elemento(int i) const
      { assert(0<=i && i<=v.size()); return v[i]; }
    bool vacio() const { return nelementos==0; }
    int size() const { return nelementos; }
};

#endif                              /* _conjunto_reales_h */
```

conjunto_reales.cpp

```
#include <cassert>
#include <conjunto_reales.h>

bool Conjunto_Reales::posicion_elemento(int& pos, float val)
const
{
  int izq=0, der=nelementos-1,centro;

  while (der-izq>=0) {
    centro=(izq+der)/2;
    if (val<v[centro])
      der=centro-1;
    else if (val>v[centro])
           izq=centro+1;
         else {
           pos=centro;
           return true;
         }
  }
  pos= izq;
  return false;
}
```

Ejemplo: Clase Conjunto_Reales (2/2).

conjunto_reales.cpp

```
bool Conjunto_Reales::insertar(float f)
{
  int pos;
  if (posicion_elemento(pos,f))
    return false;
  else {
    if (v.size()==nelementos)
      if (v.size()==0)
        v.resize(1);
      else v.resize(2*v.size());
    for (int j=nelementos; j>pos; --j)
      v[j]=v[j-1];
    v[pos]= f;
    nelementos++;
    return true;
  }
}
```

conjunto_reales.cpp

```
bool Conjunto_Reales::borrar(float f)
{
  int pos;
  if (posicion_elemento(pos,f)) {
    nelementos--;
    for (int j=pos;j<nelementos;++j)
      v[j]=v[j+1];
    if (nelementos<v.size()/4)
      v.resize(v.size()/2);
    return true;
  }
  else return false;
}
```

T.DA VECTOR DISPERSO

```
/**
 * @file vector_disperso.h
 * @brief Fichero cabecera del TDA vector disperso
 *
 * Se crea un vector con múltiples elementos dispersos en un
 * rango amplio de su índice
 *
 */
#ifndef _vector_disperso_h
#define _vector_disperso_h

 /**
  *
  *
  * Una instancia de v del tipo de dato abstracto de Vector_Disperso
  * sobre el tipo de float es un array 1-D con índices enteros
  * positivos sin limitación de rango.
  *
  * Este tipo de dato se diseña especialmente para los problemas
  * en los que el vector almacena en la mayoría de sus posiciones
  * un valor predeterminado de d, mientras que modifica un
  * pequeño conjunto de ellas. Lo podemos representar como:
  *     {(i[0], v[0]), (i[1], v[1]), ..., (i[n-1], v[n-1]), (*, d)}
  *
  * La eficiencia en espacio es de O(n), donde de n es el
  * número de posiciones del vector que almacenan un valor
  * distinto de de d
  *
  *
  *
  */
```

```
Class Vector_Disperso {

private:
/** @page repVector_Disperso Rep del TDA Vector_Disperso
 *
 * @section invVector_Disperso Invariante de la representación
 *
 * Un objeto válido rep del TDA Vector_Disperso debe cumplir
 * - @c rep.nelementos >= 0
 * - @c rep.reservados >= 0
 * - @c rep.datos apunta a una zona de memoria con
 * capacidad para albergar @c reservados valores de tipo
 *       @c Elemento
 * - @c 0 <= rep.datos[i].indice < rep.datos[j].indice para
 *       todo @e i, j, tal que @c 0 <= i < j < rep.nelementos
 * @section faVector_Disperso Función de abstracción
 *
 * Un objeto válido rep del TDA Vector_Disperso representa
 * al vector:
 * { (rep.datos[0].indice, rep.datos[0].valor), ....,
 *   (rep.datos[rep.nelementos - 1].indice, rep.datos[rep.nelementos-1]
 *    .valor), (*, rep.valor_por_defecto) }
 *
 */
```

```
/**
 * @brief Posición en el vector
 * El tipo de Elemento almacena una pareja de valores que indica el
 * índice y el valor correspondiente de una posición en el vector
 *
 */
struct Elemento {
    int indice; /**< Posición en el vector */
    float valor; /**< Valor almacenado en la posición de índice del vector */
};
Elemento * datos; /**< Matriz de posiciones usadas en el vector */
int nelementos; /**< Número de elementos usados en la matriz de datos */
int reservados; /**< Número de posiciones reservadas en de datos */
float valor_por_defecto; /**< Valor en las posiciones fuera de de datos */
```

```
/**
```

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
20	0	0	0	0	0	0	0	0	50	0	0	0	0	-1	0	0	0	0	0	0	0	0	28	0

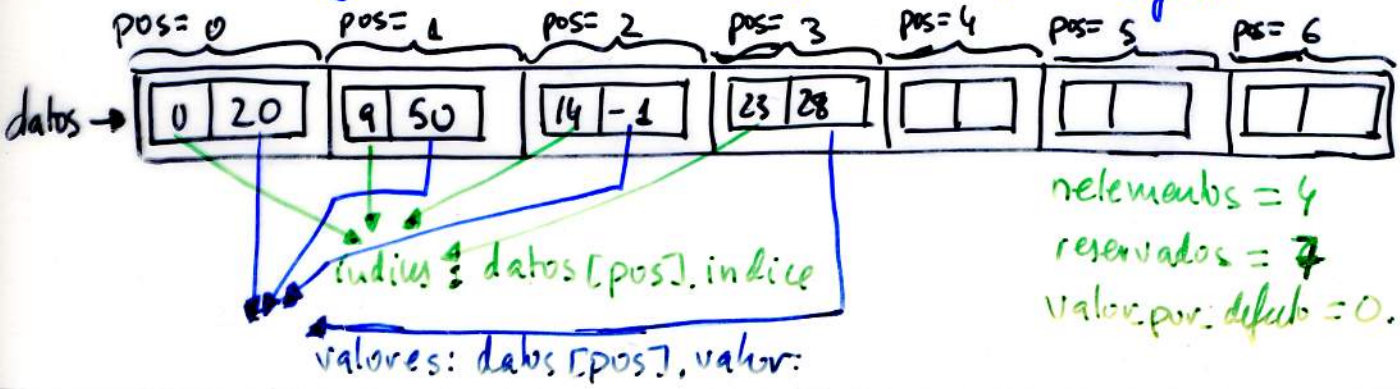
Vector disperso con 4 posiciones válidas:

indice i=0 con valor 20 || indice i=14 con valor -1
indice i=9 con valor 50 || indice i=23 con valor 28

Representación:

{(0, 20), (9, 50), (14, -1), (23, 28) (0)}

Elemento 1 | Elemento 2 | Elemento 3 | Elemento 4 | valor por defecto



Ejemplo: Clase Vector Disperso (1/3).

vector_disperso.h

```
#ifndef _vector_disperso_h
#define _vector_disperso_h

class Vector_Disperso {
  private:
    struct Elemento {
      int indice;
      float valor;
    };
    Elemento * datos; int nelementos; int reservados;
    float valor_por_defecto;
    bool posicion_indice(int& pos, int i) const;
    void resize(int n);
  public:
    Vector_Disperso(float def[illegible]);
    Vector_Disperso(const Vector_Disperso& orig);
    ~Vector_Disperso();
    Vector_Disperso& operator=
            (const Vector_Disperso& original);
    float get_default() const;
    float get(int i) const;
    void set(int i, float f);
    int num_no_default() const;
    void datos_posicion (int i, int &indice, float& valor) const;
};
#endif                                  /* _vector_disperso_h */
```

vector_disperso.cpp

```
bool Vector_Disperso::posicion_indice(int& pos, int i) const
{
  int izq=0, der=nelementos-1,centro;

  while (der-izq>=0) {
    centro=(izq+der)/2;
    if (i<datos[centro].indice)
      der=centro-1;
    else if (i>datos[centro].indice)
            izq=centro+1;
          else {
            pos=centro;
            return true;
          }
  }
  pos= izq;
  return false;
}
void Vector_Disperso::resize(int n)
{
  assert(n>=nelementos && n>0);
  Elemento * nuevos_datos= new Elemento[n];
  for (int j= 0; j<nelementos ;++j)
    nuevos_datos[j]= datos[j];
  delete[] datos;
  datos= nuevos_datos;
  reservados=n;
}
```

Ejemplo: Clase Vector Disperso (2/3).

vector_disperso.cpp

```
Vector_Disperso::Vector_Disperso(float defecto)
{
  datos=new Elemento[1];
  nelementos=0;
  reservados=1;
  valor_por_defecto= defecto;
}

Vector_Disperso::Vector_Disperso(const Vector_Disperso& orig)
{
  valor_por_defecto= orig.valor_por_defecto;
  reservados= nelementos= orig.nelementos;
  if (nelementos>0) {
    datos= new Elemento[nelementos];
    for (int i=0; i<nelementos;++i)
      datos[i]= orig.datos[i];
  }
  else datos=0;
}
```

vector_disperso.cpp

```
Vector_Disperso::Vector_Disperso()
{ datos=0;
  nelementos=reservados=0;
  valor_por_defecto= 0.0;
}

Vector_Disperso::~Vector_Disperso()
{
  if (reservados>0)
    delete[] datos;
}

Vector_Disperso& Vector_Disperso::operator=
                (const Vector_Disperso& orig)
{
  if (this! = &original) {
    if (reservados>0)
      delete[] datos;
    valor_por_defecto= orig.valor_por_defecto;
    reservados= nelementos= orig.nelementos;
    if (nelementos>0) {
      datos= new Elemento[nelementos];
      for (int i=0; i<nelementos;++i)
        datos[i]= orig.datos[i];
        }
    else datos=0;
  }
  return *this;
}
```

Ejemplo: Clase Vector Disperso (3/3).

vector_disperso.cpp

```
int Vector_Disperso::num_no_default() const
{
  return nelementos;
}
void Vector_Disperso::datos_posicion(int i, int& indice, float&
valor) const
{
  assert (0<=i && i<nelementos);
  indice= datos[i].indice;
  valor= datos[i].valor;
}

float Vector_Disperso::get_default() const
{
  return valor_por_defecto;
}

float Vector_Disperso::get(int i) const
{
  int pos;
  if (posicion_indice(pos,i))
    return datos[pos].valor;
  else return valor_por_defecto;
}
```

vector_disperso.cpp

```
void Vector_Disperso::set(int i, float f)
{
  int pos;

  if (posicion_indice(pos,i)){                    // Está en pos
    if (f! =valor_por_defecto)
      datos[pos].valor= f;
    else {                                        // hay que eliminarlo
      nelementos= nelementos-1;
      for (int j=pos;j<nelementos;++j)
        datos[j]=datos[j+1];
      if (nelementos<reservados/4)
        resize(reservados/2);
    }
  }
  else {              // No está en el vector (estaría en pos)
    if (f! =valor_por_defecto) {
      if (nelementos==reservados)
        resize(reservados*2);
      for (int j=nelementos; j>pos; --j)
        datos[j]= datos[j-1];
      datos[pos].indice= i;
      datos[pos].valor= f;
      ++nelementos;
    }
  }
}
```