Práctica 3: Implementación del T.D.A. Vector Disperso

Dpto. Ciencias de la Computación e Inteligencia Artificial
E.T.S. de Ingenierías Informática y de Telecomunicación
Universidad de Granada

Estructuras de Datos

Grado en Ingeniería Informática

Indice de contenido

Índice de contenido

1. Objetivos

Los objetivos de este guión de prácticas son los siguientes:

1. Comprender la importancia de la abstracción de datos.
2. Familiarizarse con T.D.A.'s contenedores implementados en la STL http://www.cplusplus.com/reference/stl/

2. Introducción: T.D.A. Vector Disperso

Un vector disperso es un vector en el que la mayoría de los elementos tienen el mismo valor (conocido como valor por defecto). Esta característica representa una oportunidad a explotar para reducir el espacio de almacenamiento (evitar almacenar en memoria numerosas repeticiones de un mismo elemento) y la eficiencia de los algoritmos (para no considerar todos los elementos, sino sólo los distintos del valor por defecto).

Un vector se dice que es disperso cuando en él sólo se almacenan los elemenos que son distintos del valor por defecto (o valor nulo).

Ejemplo: Supongamos que tenemos el conjunto de palabras o lemas del castellano (fichero datos/lema.txt) almacenados en un vector (Lemas). Los valores de Lemas están ordenados alfabéticamente. Sea V un vector de enteros donde V[i] representa el numero de ocurrencias del i-ésimo lema (Lemas[i]) en un párrafo de un texto (por ejemplo, el datos/Quijote.txt). Este vector tendrá una dimensión de 85918 (número total de lemas distintos) y se podría declarar como

```
vector<int> V(85918,0):
```

Obviamente el número de palabras en un párrafo es mucho menor que el numero de posibles lemas. Por ejemplo, el primer párrafo de esta sección tiene un total de 65 palabras distintas, y por tanto, sólo 65 posiciones del vector V tomarán un valor distinto de 0. Almacenar toda la información en el vector puede ser muy costoso en términos de:

- espacio: pues almacenaríamos 84999 ceros.
- tiempo: para contar el numero total de palabras del párrafo, tendríamos que iterar por las 85918 posiciones para identificar aquellas con valor distinto de cero.

Una solución consiste en utilizar un tipo de dato abstracto que nos permita trabajar con este tipo de información de forma eficiente. En cualquier caso, debemos de dotar al vector disperso de un interfaz similar a la que ofrece la clase vector. En concreto en esta práctica proponemos que el vector disperso esté dotado de los métodos que se indican en la sección 2.1.

Así, por ejemplo el vector de nuestro ejemplo se podría declarar como

```
vectorD<int> V(85918,0);
```

y podríamos indicar que el lema 1546 ha ocurrido 12 veces en el párrafo o sumar todas las ocurrencias mediante

```
V.set(1546, 12);
...
for (int i=0; i<V.size(); i++)
    suma+=V[i];
```

Sin embargo, este vector NO almacenaría todos los posibles valores, sólo los valores no nulos. Para ello, es suficiente con considerar para cada valor no nulo la posición en la que se encuentra y el valor asociado. En la sección 3 propondremos una posible representación para el vector disperso, basada en la clase list de la STL.

2.1. Especificación

Ver la especificación en el fichero vectorD.h (adjunto al material de la práctica).

3. Representación

Se propone la siguiente representación utlizando la clase list de la STL:

```
list< pair<int,T> > vd; // vector que aloja los valores no nulos
int n_ele; // numero de elementos totales
T v_nulo; //valor nulo del vector
```

La función de abstracción y el invariante de la representación asociadas a esta representación son:

3.1. Función de Abstracción

```
/* FA: rep -- > vector
  FA(V) dado un vectorD V (vd,n_ele,v_nulo) con
  vd=[ (a,v1), (b,v2), ..., (n,vn) ]
  n_ele = M
  v_nulo = t
  ---------->  vector M, tal que M[pos]

  pos:       0 1 2 … a-1  a ...x...b...n-1  n  n+1...M-1

  M[pos]:    t t t … t    v1...t...v2...t   vn  t ... t
  */
```

3.2. Invariante de la Representación

```
/* IR : rep ----> bool
  IR(V) dado un vectorD V (vd,n_ele,v_nulo) se satisface que:

   0 <= vd.size() < n_ele;
   vd[i].second != v_nulo, para todo i = 0, ..., vd.size()-1;
   vd[i].first >=0, para todo i = 0, ..., vd.size()-1;
   vd[i].first < vd[j].first si i<j

  */
```

4. Tareas a realizar

4.1. Tareas obligatorias

- **1**. Implementar la clase vector disperso (vectorD) siguiendo la representación propuesta en la sección 3 y completando los métodos especificados en el fichero vectorD.h (adjunto al material de la práctica).
- **2**. Implementar dos tipos de iteradores sobre el vector disperso: stored_iterator (iterador sobre elementos no nulos) e iterator (iterador sobre todos los elementos del vector), según se especifica en vectorD.h. A modo ilustrativo, los métodos de stored_iterator ya están implementados en dicho fichero. No olvide implementar también los métodos vectorD<T>::begin(), end(), sbegin() y send().

  Además de implementar estos iteradores, fíjese en la cabecera propuesta para operator*:

  stored_iterator: `const pair<int, T > & operator* ();`

  iterator: `const  T  & operator* ();`

  Ambos operadores devuelven referencias constantes (siguiendo la nomenclatura de C++ los tipos de iteradores deberían entonces llamarse const_stored_iterator y const_iterator, respectivamente). Reflexione sobre qué consecuencias tendría la sobrecarga de operator* siguiendo esta cabecera:

  stored_iterator: `pair<int, T > & operator* ();`

  iterator: `T  & operator* ();`

4.2. Tareas adicionales

- **3**. Adicionalmente, realizar una especificación e implementación completas de la clase vectorD utilizando una representación interna map<int,T>. Realizar un análisis comparativo de la eficiencia teórica y empírica para la implementación desarrollada en 1 y 3 para las operaciones de acceso, inserción y borrado de elementos de vectorD.

4.3. Material facilitado para la práctica

Como material imprescindible para la realización de la práctica:

- vectorD.h: especificación del TDA Vector Disperso.

A modo ilustrativo y relacionado con el ejemplo visto en la introducción, se entrega de forma adicional:

- datos/quijote.txt: fichero que contiene el texto completo de “El Quijote”
- datos/lemas.txt: fichero de texto con 85.918 palabras del castellano, por orden alfabético.
- src/frecuencias.cpp: fichero con distintos algoritmos para contar la frecuencia de las

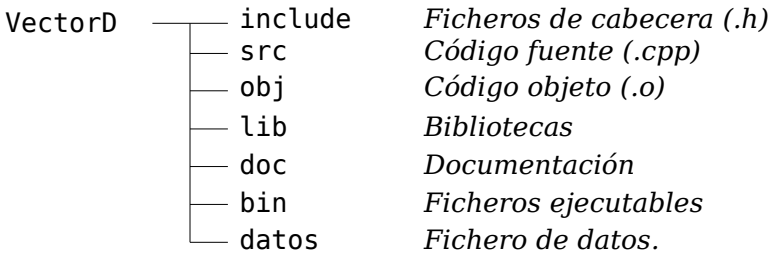
palabras en un texto. Ilustra el uso de clases contenedoras de la STL: vector y map. Incluye una parte de código para evaluar una alternativa basada en vectorD.

– src/ejemplo_vectorD.cpp: fichero que exhibe parte de la funcionalidad que debe programarse para vectorD.

4.4. Normas de la entrega

El alumno deberá empaquetar todos los archivos relacionados en el proyecto en un archivo con nombre "vectorD.*tgz*" y entregarlo antes de la fecha que se publicará en la página web de la asignatura. Tenga en cuenta que no se incluirán ficheros objeto, ni ejecutables. Es recomendable que haga una "limpieza" para eliminar los archivos temporales o que se puedan generar a partir de los fuentes.

El alumno debe incluir el archivo Makefile para realizar la compilación. Tenga en cuenta que los archivos deben estar distribuidos en directorios*:*



Para realizar la entrega, en primer lugar, realice la limpieza de archivos que no se incluirán en ella, y sitúese en la carpeta superior (en el mismo nivel de la carpeta "*vectorD*") para ejecutar:

```
prompt% tar zcv vectorD.tgz vectorD
```

tras lo cual, dispondrá de un nuevo archivo vectorD.tgz que contiene la carpeta vectorD así como todas las carpetas y archivos que cuelgan de ella.

5. Referencias

• [GAR06b] Garrido, A. Fdez-Valdivia, J. *"Abstracción y estructuras de datos en C++"*. Delta publicaciones, 2006.