Gestión de sentencias: Proyecto de programación

Daniel Franco Barranco
Iván Matellanes Pastoriza
Asier Mujika Aramendia

Estructuras de Datos y Algoritmos
Grado en Ingeniería Informática

Diciembre de 2012

Índice

Resumen

Aquí debe presentarse un resumen de todo el trabajo, que no debe superar la media página (más o menos). Debe explicarse en qué consiste el trabajo presentado, qué partes tiene y de qué trata cada

Introducción

En esta sección debe explicarse el escenario del trabajo. Se presentarán también los objetivos del mismo: enunciado del problema a resolver y el alcance de funcionalidad de la implementación que se ha desarrollado.

La idea principal del proyecto que se nos presenta en esta asignatura es la de gestionar sentencias simples, que tendrá la forma de ***sujeto propiedad objeto***. Para realizar este cometido, las sentencias han de agruparse en almacenes sobre los que se definen una serie de operaciones a implementar. A continuación se listan algunos ejemplos de estas operaciones:

- Obtener las sentencias de un almacén con un sujeto determinado.
- Obtener una colección ordenada de todas las sentencias del almacén.
- Dar una colección de entidades que son sujetos en varios almacenes distintos.
- Cargar sentencias a un almacén desde un fichero de texto.

Algunas de las propiedades pueden recibir un tratamiento especial, como es el caso de ***es*** y ***subClaseDe***, y en base a este se piden ciertas operaciones, entre las que se incluyen: las clases que son superclases de otra, los profesores que trabajan para un determinado departamento, etc.

Desde el primer momento, la intención de los tres componentes del grupo ha sido realizar la estructuración e implementación del proyecto de la forma más eficiente posible en base a las estructuras de datos que conocemos. De este modo, hemos pretendido escribir código limpio, correcto y eficaz para la versión inicial o versión 0, y

así evitar futuras restructuraciones o amplias modificaciones en el código. Las ideas planteadas se exponen más detalladamente en el siguiente apartado.
Éramos conscientes de que según avanzara el curso sería imprescindible realizar ajustes para hacer el programa más competitivo, pero nuestra intención era que los inevitables ajustes no supusieran replantear toda la estructura inicial.

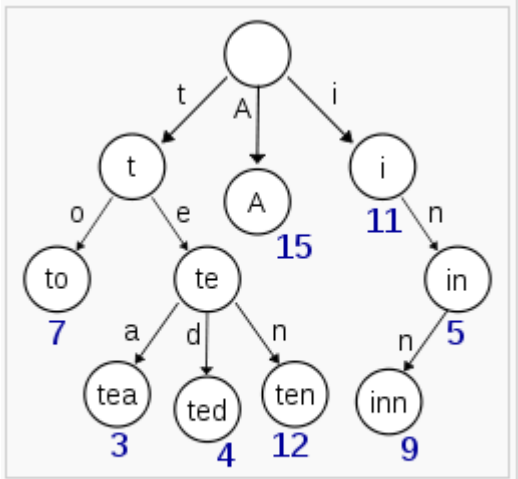
Versión 0

Puesto que se trata de dejar constancia de todo el trabajo realizado durante el curso, es interesante plasmar en distintas secciones los estados principales por los que ha ido pasando la aplicación desarrollada. Así, podemos redactar una sección, que corresponderá con una nueva versión, por cada organización fundamental de los datos a procesar (diseño de datos) y sus correspondientes implementaciones de métodos que hayamos alcanzado con esa versión.
Se explicarán brevemente las decisiones de diseño de las estructuras de datos seleccionadas para resolver los problemas que se aborden inicialmente y se enumerarán los métodos que se han implementado.
Para cada método se indicará lo que hace (especificación) y, si es relevante, se dará también una breve explicación de cómo lo hace. Cada método irá acompañado de su clasificación en complejidad algorítmica y de una justificación de esa clasificación.
Además, para los métodos más importantes, se realizarán pruebas de ejecución sobre datos de entrada de distintos tamaños para registrar el tiempo necesitado por esos métodos y se presentarán las tablas de resultados del modo más conveniente para su interpretación.
El código fuente de los programas implementados puede agruparse en un anexo que se presenta en una sección creada al efecto al final de la memoria.

La idea básica con la que comenzamos a abordar este proyecto se detalla a continuación.

En primer lugar, para contener las relaciones entre sentencias, en lugar de utilizar *arrays* simples o listas enlazadas (que hubiera sido la idea más simple a primera vista), decidimos representar todas las sentencias utilizando un grafo dirigido ponderado en el que los nodos serían los sujetos y objetos, y el peso de las aristas estaría asociado a las propiedades de las sentencias. Se le asigna a cada entidad (sujeto, objeto y propiedad) un valor numérico para posteriormente administrar todo con una matriz de adyacencia.

1.1 Representación gráfica del grafo.



Un trie de las claves "A", "to", "tea", "ted", "ten", "i", "in", and "inn".

Para relacionar el *string* de una entidad con su valor numérico correspondiente utilizamos un árbol de prefijos (trie).

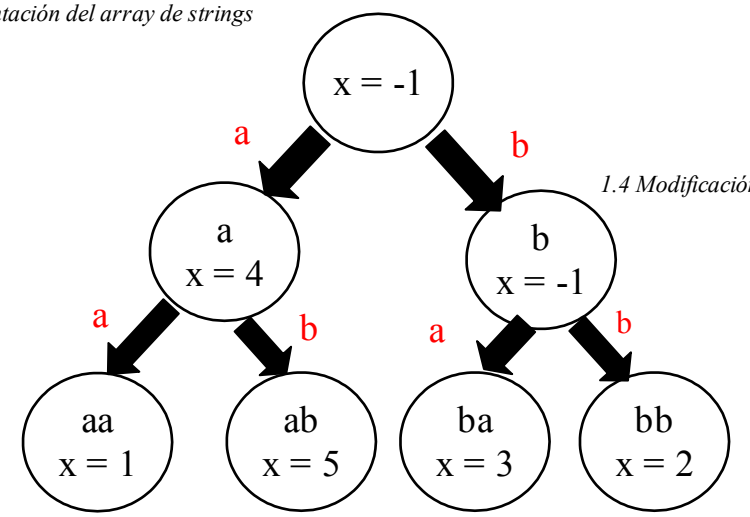
La idea es esta. Como en el ejemplo se puede observar para construir una palabra recorreremos el árbol hasta completar la palabra deseada. En nuestro caso, haremos el mismo árbol, guardando sus características pero con un pequeño cambio para relacionar las palabras con los números. Se añadirá a cada nodo aparte de la concatenación de caracteres que le corresponde, un valor numérico. Si la palabra estuviese en el array ese valor (variable X) tomaría el indice de la posición en la que se encuentra esa palabra, por el contrario tendrá el valor "-1" por defecto.

Ejemplo:

Las palabra que están en el array son estas y se representarían así:

1	2	3	4	5	
aa	bb	ba	a	ab	….

1.3 Representación del array de strings



1.4 Modificación de árbol Trie.

En el sentido contrario, si queremos conocer a qué *string* pertenece un índice dado, se accede a un *array* de *strings* en la posición indicada que contiene el *string* correspondiente. Posteriormente, una vez obtenido el valor numérico que se le asigna a cada *string* decidimos en un principio administrar estos datos con una matriz de adyacencia de esta manera (tomando en cuenta los valores del ejemplo 1.1) :

EJEMPLO EXPLICATIVO:

Estudiante2013 = 1
Matriculado = 3
En upv-ehu = 2

Sujetos / Objetos	...	En upv-ehu	...
Estudiante2013	-1	3	-1
...	-1	-1	-1
...	-1	-1	-1

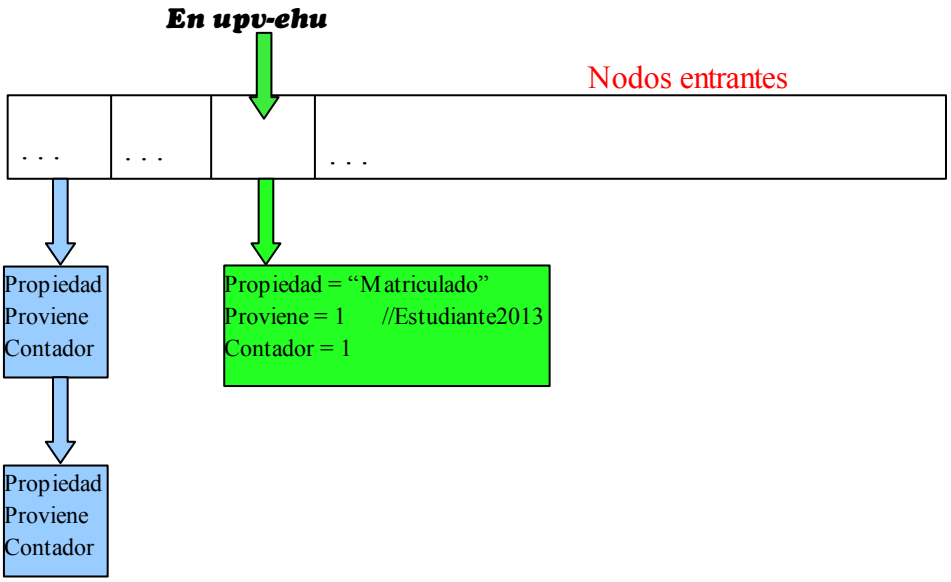
Como se puede observar en el ejemplo se pondría en cada casilla un valor '-1' en caso de no haber relación entre los sujetos y objetos. Y en el caso de que la hubiese se pondría el valor de la propiedad que les une.

1.5 Representación gráfica de la matriz de adyacencia

El caso de la organización de los objetos, sujetos y propiedades de la opción anterior no resulto ser como creíamos de efectiva al tener que hacer una matriz de adyacencia con demasiadas filas y columnas. Por lo tanto, para esta administración en la 3º reunión decidimos utilizar otra estructura. La idea es con dos arrays estructurar la información de las aristas (propiedades) . Un array para las aristas de entrada a un nodo y otro para las salidas. Cada posición del array corresponderá con los sujetos/objetos de la posición correspondiente en el array que teníamos (el array que relacionaba a cada string su valor numérico creado a partir del árbol trie). Cada posición tendrá un apuntador a otro apuntador con 3 variables para darnos información de una arista (propiedad) que entra/sale al nodo en el que estemos. Este se enlazará con otro apuntador con información de otra arista que entre/salga y así sucesivamente hasta que tengamos todas las propiedades relacionadas con ese sujeto/objeto. Es decir, los valores serán toda la información del nodo. Estos serán los valores de la variables del apuntador:

- Propiedad: nos dará la propiedad que entra/sale del nodo en el que estemos.
- Proviene: nos dará la información del nodo del cual proviene.
- Contador: nos dará la información de cuantas veces esta repetida la arista que entra/sale a este nodo.

ESQUEMA GRÁFICO (ejemplo 1.1):



1.6 Representación gráfica del array que administra los nodos.

Y esta estructura sería la misma pero para los nodos en los que les entran las aristas.

Estas dos estructuras de datos permiten realizar ambas operaciones asociativas en un tiempo mínimo, como se puede apreciar en el análisis de complejidad de los métodos:

INSERTAR ANÁLISIS AQUÍ

Mediante el uso de estos métodos, las primeras cuatro operaciones que figuran en la definición del proyecto se pueden implementar de una forma muy sencilla a la par que eficiente, como se demuestra en este análisis:

1) Colección de sentencias del almacén que tienen un sujeto determinado

```
public String[] sentenciasPorSujeto( String Sujeto ) throws IOException {
		String[] coleccionSentencias= new String[200000];
		int index = arbolSujetosObjetos.obtenerValor(Sujeto);
		Arista prov;
		ListaEnlazada.Iterador<Arista> it = new
ListaEnlazada.Iterador<Arista>();
		it.load(nodosSalientes.getElementByPosition(index));
		while (it.hasNext()){
			prov = it.next();
			for(int i=0;i<prov.repeticiones;i++){
				coleccionSentencias[i] =
listaSujetosObjetos.getElementByPosition(index) +" "+
listaPropiedades.getElementByPosition(prov.arista) +" "+
listaSujetosObjetos.getElementByPosition(prov.verticeObjetivo)+ "  .\n";

			}
		}
		return coleccionSentencias;
	}
```

ANÁLISIS DEL ALGORITMO:

2) Colección de sentencias distintas del almacén que tienen un sujeto determinado

```
	public String[] sentenciasDistintasPorSujeto( String Sujeto )
	throws IOException {
		String[] coleccionSentencias= new String[200000];
		int i=0;
		int index = arbolSujetosObjetos.obtenerValor(Sujeto);
		Arista prov;
		ListaEnlazada.Iterador<Arista> it = new
ListaEnlazada.Iterador<Arista>();
		it.load(nodosSalientes.getElementByPosition(index));
		while (it.hasNext()){
			prov= it.next();
```

```
                  coleccionSentencias[i] =
listaSujetosObjetos.getElementByPosition(index) +" "+
listaPropiedades.getElementByPosition(prov.arista) +" "+
listaSujetosObjetos.getElementByPosition(prov.verticeObjetivo)+ "
.\n";

                  i++;
            }
            return coleccionSentencias;
      }
```

ANÁLISIS DEL ALGORITMO:

3) Colección de propiedades distintas que aparecen en las sentencias del almacén

```
public String[] propiedadesDistintas() throws IOException {
            String[] coleccionSentencias= new String[200000];
            for(int i=0;i<listaPropiedades.size();i++){
                  coleccionSentencias[i] =
listaPropiedades.getElementByPosition(i)+"\n";
            }
            return coleccionSentencias;
      }
```

ANÁLISIS DEL ALGORITMO:

$$\sum_{i=0}^{n-1} 1 = n\text{-}1\text{+}1\text{-}0 = n \text{ ----> } \mathbf{OF(n)}$$

4) Colección de entidades distintas que son sujeto de alguna sentencia y también son objeto de alguna sentencia de ese almacén

```
public String[] entidadesSujetoObjeto() throws IOException {
            String[] coleccionSentencias= new String[200000];
            for(int i = 0; i<nodosSalientes.size();i++){
                  if(!nodosSalientes.getElementByPosition(i).isEmpty()
&& !nodosEntrantes.getElementByPosition(i).isEmpty()){
                        coleccionSentencias[i] =
listaSujetosObjetos.getElementByPosition(i)+"\n";

                  }
            }
            return coleccionSentencias;
      }

}
```

ANÁLISIS DEL ALGORITMO:

$$\sum_{i=0}^{n-1} 1 = n\text{-}1\text{+}1\text{-}0 = n \text{ ----> } \mathbf{OF(n)}$$

Versión 1

Según vayamos avanzando en el curso y vayamos conociendo nuevas formas de estructurar los datos y nuevos modos de tratar esas estructuras, descubriremos que la Versión 0, realizada como primera opción, es mejorable con la aplicación de esos nuevos conocimientos.
Aprovecharemos todo lo que se pueda de la versión anterior (y no habrá que presentarlo en esta nueva sección, bastará con indicarlo). Con cada nueva versión, se explicarán y se justificarán los cambios fundamentales en las estructuras de datos y los métodos correspondientes.
Se mantendrá el estilo de la presentación: descripción y justificación de la representación de los datos, enumeración y especificación de cada método, análisis de eficiencia y tablas de tiempos de ejecución de los más relevantes.

Versión 2

Es más que probable que convenga modificar las estructuras de datos más de una vez, ya que vamos a ir descubriendo estructuras interesantes durante todo el curso. Si se encuentra algo mejor, se valorará su incorporación a la aplicación. La finalidad del proyecto de programación es demostrar que se conocen las distintas estructuras de datos que iremos viendo en el curso y que se saben usar allá donde conviene para conseguir aplicaciones eficientes.
En cada sección, dedicada al efecto, se explicarán las razones de la correspondiente modificación.
El código fuente se presentará, agrupado, en la correspondiente sección de anexos.

Actas de las reuniones

En esta sección se irán redactando, en orden cronológico y "en tiempo real", las actas de las reuniones del grupo de trabajo. La periodicidad y el número de reuniones será una decisión del grupo de trabajo; pero parece razonable que, al menos, haya una cada semana. Cada acta llevará la fecha de realización y duración de la reunión. Cada acta tendrá una redacción breve; me extrañaría que fuese necesario superar la media página.
En estas actas deben reflejarse las decisiones tomadas en cada reunión y deben reflejarse los hitos relevantes del proyecto. Por ejemplo, deberá reflejarse el reparto de tareas entre las personas del grupo decidido para un objetivo concreto y, cuando sea el caso, reflejará los objetivos que se han conseguido al terminar determinada tarea. De este modo se puede tener un seguimiento del proceso de desarrollo del proyecto. Para ello es primordial que esas actas se redacten en el momento correspondiente y no se demore nunca su redacción más allá del comienzo de la siguiente reunión. Las decisiones de reparto de las tareas pueden ser fundamentales para un eficiente desarrollo del proyecto. Por ejemplo, mientras alguien realiza la implementación de unos métodos, otra persona puede realizar los métodos de prueba de los mismos y una tercera redactar las secciones correspondientes para la memoria. Puede ser interesante que, después, cada cual revise algún aspecto crítico de la tarea realizada por otro miembro del grupo, para asegurar la corrección de la misma. Este modo de actuar puede ser mucho más eficiente que el intento de realizar todos todo y a la vez. El modo de trabajo se verá reflejado en las actas de las reuniones.

28/10/2012:

Este día nos hemos reunido por primera vez el grupo para acordar como vamos ha abordar el proyecto de EDA. Hemos discutimos las diferentes ideas que había entre los participantes del grupo y hemos decidimos coger las ideas principales y desarrollarlas. Tras varias deliberaciones hemos llegamos a una idea de la estructura que vamos a implementar para realizar el proyecto. Esta nueva estructura nos permitirá realizar las funciones deseadas menor tiempo y con más eficacia.

01/10/2012:

En esta primera reunión hemos decidido varios aspectos a tener en cuenta y nos hemos repartido la tarea en varias partes:

- Hemos terminado de debatir las estructuras que vamos a utilizar en el proyecto.
- A cada miembro del grupo se les ha asignado un tipo de tarea que realizar:
  - Asier Mujika Aramendia realizará la implementación de el árbol necesario para la creación del array.
  - Iván Matellanes Pastoriza realizará las funciones respecto al uso del fichero.
  - Daniel Franco Barranco probará casos de prueba para las funciones creadas anteriormente por Iván y realizará la parte de la memoria correspondiente a los primeros pasos del proyecto.

Conclusión del acta: Tras un día de reparto de los trabajos individuales se han completado con éxito todas las expectativas. En la siguiente reunión grupal se decidirán los nuevos trabajos a cada miembro del grupo.

03/10/2012:
En esta tercera reunión hemos decidido algunas nuevas ideas para modificar y mejorar el proyecto:

- La estructura para hacer los ejercicios que se nos proponen no fue del todo correcta decidida en la primera reunión. Por lo que hemos decidido una nueva manera de implementación como se refleja y se redacta detalladamente en la versión 0 de la memoria.

05/10/2012:
En esta cuarta reunión hemos decidido repartir el trabajo a cada miembro del grupo para la continuación de las tareas del proyecto.

- Ivan Matellanes Pastoriza: aclarará la programación del proyecto para una mejor visibilidad de las estructuras y una mejor complexión de estas.
- Daniel Franco Barranco: realizará algunos cambios en los programas para adaptar estos al enunciado del proyecto. También realizará el análisis del algoritmo de algunos programas realizados.
- Asier Mujika Aramendia: Realizará los pertinentes cambios a la memoria acorde al seguimiento del proyecto.

Anexos

Anexo versión 0
Código fuente de lo implementado para la versión 0.
Anexo versión 1
Código fuente de lo implementado para la versión 1.
Anexo versión 2
Código fuente de lo implementado para la versión 2.