Estructuras de Datos y Algoritmos

Práctica II - Curso 2015/16

Ropes

Objetivo de la práctica

El objetivo de ésta práctica es la de comparar la implementación habitual de las cadenas de caracteres (strings) mediante arrays con una representación basada en el concepto de **ropes,** de forma que podamos obtener una medida de la eficiencia de las operaciones de **concatenación** de strings y **acceso en secuencia** a los caracteres de un string.

Descripción técnica

Una *rope* representa una cadena de caracteres como un árbol binario donde cada nodo interno representa una concatenación de dos subcadenas (los hijos izquierdo y derecho) y los nodos externos (las hojas del árbol) representan literales de cadena.

Por lo tanto, la cadena total está dividida en una serie de trozos, almacenados en las hojas del árbol, y se puede reconstruir mediante la concatenación de esos trozos siguiendo un recorrido *inorden* del árbol.

Cada **nodo interno** representa una **concatenación** de dos subcadenas (la parte izquierda y la parte derecha), las cuales se representan también mediante *ropes*, y almacena la siguiente información:

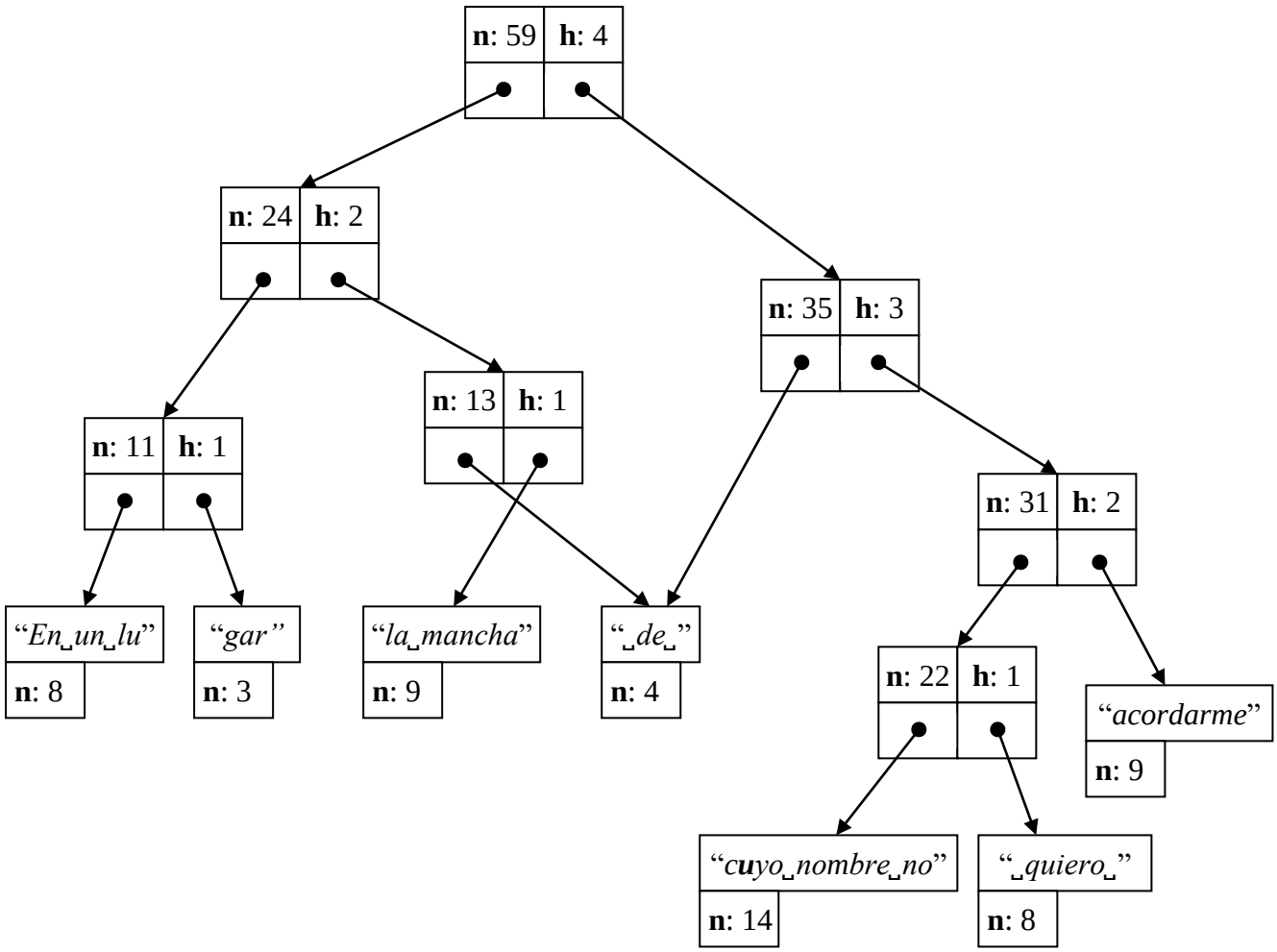
- Dos enlaces a los *ropes* que representan la subcadena **izquierda** y **derecha**.
- Un entero que almacena la **longitud** total, en caracteres, de la cadena representada.
- Un entero que almacena la **altura** del árbol del cuál es raíz ese nodo.

Cada **nodo externo** representa una **cadena literal**, y almacena la siguiente información:

- Un **string** que almacena la cadena de texto (en el formato nativo del lenguaje).
- Un entero que almacena la **longitud** de la cadena.

Para simplificar la longitud del código se permite el definir una única clase que represente ambas posibilidades (nodo interno o externo) definiendo todas las propiedades de ambos.

Por ejemplo, la cadena "*En un lugar de la mancha, de cuyo nombre no quiero acordarme*", podría representarse como:



Nota: En el diagrama **n** representa la longitud y **h** la altura. Existen muchas otras representaciones posibles de esta cadena, dependiendo de la división particular en trozos y la estructura del árbol que se elija.

En el ejemplo se puede apreciar que un nodo hoja, el que representa la subcadena " *de* ", está referenciado por dos nodos: Esto es posible porque los *ropes* representan cadenas de texto **inmutables**, y por lo tanto está garantizado que ninguna subcadena va a sufrir modificaciones (**Nota**: En realidad las *ropes* no son árboles, sino grafos acíclicos dirigidos, pero esta distinción no tiene relevancia práctica para ésta estructura).

El objetivo de ésta representación consiste en lograr que la operación de concatenación de cadenas tenga una eficiencia $O(1)$: Se puede apreciar que concatenar dos cadenas representadas por *ropes* consiste únicamente en crear un nuevo nodo raíz que apunte a las subcadenas. Esto se consigue a costa de complicar otras operaciones, como el acceso a caracteres individuales.

En los siguientes apartados se describe cómo se realizan las operaciones que son relevantes para el algoritmo que se va a medir.

1. Creación de un *rope* a partir de un string:

Dado un string, para crear su *rope* equivalente basta con crear un nodo externo (hoja) que almacene el string original y su longitud. El árbol resultante tienen un único nodo (la altura de un nodo hoja es por definición 0 y por ello no es necesario almacenarla).

2. Concatenación de dos *ropes*:

Se crea un nodo interno cuyos enlaces apunten a los dos *ropes* que se concatenan (no es necesario *clonar* los *ropes* originales ya que son inmutables) y cuya longitud es la suma de las longitudes de ambos *ropes* y su altura uno más que el máximo de la altura de los *ropes* que se concatenan. Este nodo pasa a ser el nodo raíz del *rope* resultante.

3. Acceso al carácter *i*-ésimo de un *rope*:

Para esta operación el algoritmo consiste en descender por el árbol desde la raíz hasta el nodo hoja que contiene el carácter *i*-ésimo, y se puede implementar de forma recursiva muy fácilmente:

a) Si el nodo es externo, basta con acceder al carácter *i*-ésimo del string almacenado con la correspondiente operación nativa del tipo string.

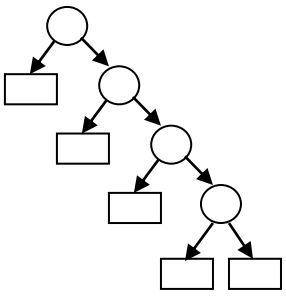
b) Si el nodo es interno, se mira la longitud de la cadena representada por el *rope* izquierdo, supongamos que es el valor *m*.

c) Si $i \leq m$, entonces es un carácter perteneciente al *rope* izquierdo: Buscamos recursivamente el carácter *i*-ésimo del *rope* izquierdo.

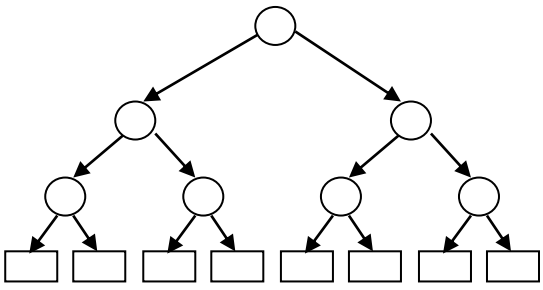
d) Si $i > m$, entonces es un carácter perteneciente al *rope* derecho: Buscamos recursivamente el carácter $(i - m)$-ésimo del *rope* derecho (**Nota**: estamos suponiendo que la indexación de los caracteres comienza por 1, si el lenguaje de programación utiliza otro esquema se deberá adaptar el algoritmo anterior a esa situación)

4. Balanceado:

La operación anterior tiene un peor caso de orden $O(h)$, siendo h la altura del árbol, ya que éste valor corresponde al máximo recorrido descendente. La peor estructura posible del árbol es la de un árbol lineal, donde la altura es proporcional al número de subcadenas (nodos externos) almacenadas. El mejor caso corresponde al de un árbol perfectamente equilibrado, donde la altura es proporcional al **logaritmo** del número de subcadenas.



Árbol lineal



Árbol perfectamente equilibrado

Para evitar que la operación de acceso al carácter *i*-ésimo se realice sobre un árbol mal equilibrado se define la siguiente operación de **balanceado** del árbol, la cual se utilizará por parte del programador cuando desee garantizar que se utiliza un *rope* equilibrado. Esta operación devuelve un nuevo *rope* basado en el anterior el cual se genera de la siguiente forma:

- Se crean una **pila** y una **lista**, inicialmente vacías.
- Se inserta el nodo raíz en la pila.

- Se entra en un bucle del que se sale cuando la pila esté vacía, en cada iteración se extrae un nodo de la pila. Si es un nodo externo (hoja), se añade al final de la lista. En caso contrario se inserta en la pila su hijo derecho y su hijo izquierdo (en ese orden).
- Tras el bucle, la lista contiene todos los nodos externos (hojas) del *rope* en el orden de concatenación.

Se pasa esta lista como argumento a una función, *arbolizar*, que devuelve un *rope* como resultado y que realiza lo siguiente:

- Si la lista contiene un único elemento devuelve ese elemento.
- Si contiene $n > 1$ elementos, divide la lista en dos sublistas, una conteniendo los primeros $n/2$ elementos y la otra los elementos restantes. Se llama recursivamente a *arbolizar* con cada una de las sublistas y se devuelve un nodo interno cuyos enlaces izquierdo y derecho apunten a los *ropes* resultantes de las llamadas recursivas con la primera y segunda sublista, respectivamente. **Nota**: No es necesario construir sublistas separadas, se aconseja usar la lista original junto con un par de índices que indiquen la zona de la lista sobre la que se actua.

Nota: No es necesario crear clases que representen las estructuras de datos pila y lista, pueden definirse utilizando clases o estructuras de datos ya predefinidas en el lenguaje o entorno de programación utilizado. En cualquier caso se debe tener cuidado en que la eficiencia de las operaciones utilizadas sobre ellas sea óptima.

Trabajo que se debe realizar

Se debe crear un código que permita representar cadenas de caracteres como *ropes* de la forma indicada en el apartado anterior y la realización de las 4 operaciones descritas (creación a partir de string, concatenación, acceso al carácter *i*-ésimo y balanceado). Esta representación se someterá a una batería de test (proporcionados en la presentación) para comprobar su eficiencia (mediante la medida del tiempo empleado) respecto a la representación nativa de cadenas de caracteres en el lenguaje de programación utilizado.

Como ejemplo del tipo de test que se puede solicitar se muestra el siguiente código Java, en el que se generan 30.000 cadenas de caracteres las cuales se concatenan y sobre la cadena resultante se recorren todos sus caracteres calculando un valor (la *huella*), este proceso se realiza tanto con **strings** como con **ropes** (se supone que se ha definido la clase Rope con las operaciones adecuadas):

```
long tpo1,tpo2,tpo3,tpo4;
int huella;

System.out.println("PRUEBA CON STRING");
tpo1 = System.currentTimeMillis();
String cad = "";
for(int i = 1; i < 30000; i++) {
    cad += "Este es un ejemplo de lo que puede ser una cadena de texto, en
este caso estamos hablando de la linea numero "+i;
}
tpo2 = System.currentTimeMillis();
huella = 0;
```

```
for(int i = 0; i < cad.length(); i++) {
    huella = huella*23 + cad.charAt(i);
}
tpo3 = System.currentTimeMillis();
System.out.println("Concatenar: "+(tpo2-tpo1)+" mseg.");
System.out.println("Recorrer: "+(tpo3-tpo2)+" mseg.");
System.out.println("Huella = "+huella);
System.out.println();

System.out.println("PRUEBA CON ROPES");
tpo1 = System.currentTimeMillis();
Rope rop = new Rope("");
for(int i = 1; i < 30000; i++) {
    rop = rop.concat(new Rope("Este es un ejemplo de lo que puede ser una
cadena de texto, en este caso estamos hablando de la linea numero "+i));
}
tpo2 = System.currentTimeMillis();
rop = rop.balancear();
tpo3 = System.currentTimeMillis();
huella = 0;
for(int i = 0; i < rop.longitud(); i++) {
    huella = huella*23 + rop.charAt(i);
}
tpo4 = System.currentTimeMillis();
System.out.println("Concatenar: "+(tpo2-tpo1)+" mseg.");
System.out.println("Balancear: "+(tpo3-tpo2)+" mseg.");
System.out.println("Recorrer: "+(tpo4-tpo3)+" mseg.");
System.out.println("Huella = "+huella);
```

El resultado de la ejecución de éste código sería (se omite los tiempos del *rope* para mantener el suspense):

```
PRUEBA CON STRING
Concatenar: 61395 mseg.
Recorrer: 21 mseg.
Huella = -480628551

PRUEBA CON ROPES
Concatenar: ¿? mseg.
Balancear: ¿? mseg.
Recorrer: ¿? mseg.
Huella = -480628551
```

Presentación y Evaluación de la práctica

Se pueden utilizar los lenguajes Java, Python, C, Pascal, Haskell o R para la implementación de la aplicación. Si se desea utilizar otro lenguaje debe obtenerse primero la autorización del profesor.

Se recomienda que la práctica se realice en parejas de dos personas, aunque en casos especiales se puede permitir la realización individual o en grupos de 3 personas (en este último caso la nota obtenida sufre una minoración)

Para una correcta evaluación de la práctica el alumno deberá:

1. Presentar electrónicamente (por el Aula Virtual de la Escuela o por correo electrónico), antes de las 23:59 del 13 de diciembre de 2015, un fichero comprimido que contenga el código fuente de la aplicación utilizada para resolver el problema planteado.
2. Presentarse a la sesión de evaluación que le corresponda según su grupo de laboratorio en la semana del 14 al 16 de diciembre de 2015. En esta sesión deberá presentar los resultados de un análisis de eficiencia basado en un caso que se proporcionará en dicha sesión.

En el caso de realización por parejas (la situación habitual), tan sólo es necesario que uno cualquiera de ellos realice la presentación electrónica. En la evaluación, sin embargo, si es necesaria la presencia de ambos y la evaluación puede ser distinta para cada uno de ellos.