Estructuras de Datos y de la Información

Ingeniería Técnica en Informática de Sistemas. Curso 2007/2008

Ejercicios del tema 5

Arboles

1. Añade las siguientes operaciones a la clase `TArbinDinamico<TElem>`, analizando su complejidad.

- `talla: Arbin[Elem]` → `Nat`

  Calcula la talla de un árbol binario, definida como el número de nodos de la rama más larga.

- `numNodos: Arbin[Elem]` → `Nat`

  Calcula el número de nodos de un árbol binario.

- `numHojas: Arbin[Elem]` → `Nat`

  Calcula el número de hojas de un árbol binario.

- `espejo: Arbin[Elem]` → `Arbin[Elem]`

  Construye la imagen especular de un árbol binario.

- `frontera: Arbin[Elem]` → `Sec[Elem]`

  Obtiene la secuencia formada por los elementos almacenados en las hojas de un árbol binario, tomados de izquierda a derecha.

- `==: (Arbin[Elem], Arbin[Elem])` → `Bool`

  Determina si dos árboles binarios son iguales.

2. Un *árbol de codificación* es un árbol binario *a* que almacena en cada una de sus hojas un carácter diferente. La información almacenada en los nodos internos se considera irrelevante. Si un cierto carácter *c* se encuentra almacenado en la hoja de posición α, se considera que α es el *código* asignado a *c* por el árbol de codificación *a*. Más en general, el código de cualquier cadena de caracteres dada se puede construir concatenando los códigos de los caracteres que la forman, respetando su orden de aparición.

(a) Dibuja el árbol de codificación correspondiente al código siguiente:

Carácter	Código
`'A'`	`1.1`
`'T'`	`1.2`
`'G'`	`2.1.1`
`'R'`	`2.1.2`
`'E'`	`2.2`

(b) Construye el resultado de codificar la cadena de caracteres "RETA" utilizando el código representado por el árbol de codificación anterior.

(c) Descifra 1.2.1.1.2.1.2.1.2.1.1 usando el código que estamos utilizando en estos ejemplos, construyendo la cadena de caracteres correspondiente.

(d) Desarrolla un módulo que implemente la clase *TArbCod* que representa a los árboles de codificación descritos en el ejercicio anterior, equipada con las siguientes operaciones

- `Nuevo:` → `ArbCod`

  Genera un árbol de codificación vacío.

– `Inserta: (ArbCod, Car, Sec[Nat]) – → ArbCod`

  Dado un árbol de codificación, un carácter y un código, representado como una secuencia de [1,2], añade el carácter al árbol en el lugar indicado por la secuencia. La operación no está definida si el carácter ya formaba parte del árbol.

– `codifica: (ArbCod, Cadena) – → Sec[Nat]`

  Construye el código de una cadena. La operación no está definida si la cadena contiene algún carácter que no se encuentra codificado en el árbol.

– `decodifica: (ArbCod, Sec[Nat]) – → Cadena`

  Construye una cadena a partir de su código. La operación no está definida si no es posible decodificar toda la entrada.

3. Modifica las implementaciones desarrolladas en clase para los árboles de búsqueda, de manera que las operaciones *busca*, *inserta* y *borra* sean iterativas en lugar de recursivas.

4. Añade las siguientes operaciones a la clase `TArbus<TClave, TValor>`, analizando su complejidad.

– `consultaK: (Arbus[Clave,Valor], Nat) → Clave`

  Obtiene la k-ésima clave de un árbol de búsqueda, considerando que en un árbol con n elementos k=0 corresponde a la menor clave y k=n–1 a la mayor.

– `recorreRango: (Arbus[Clave,Valor], Clave, Clave) → Sec[Valor]`

  Dadas dos claves *a* y *b* devuelve una secuencia con los valores asociados a las claves que están en el intervalo [*a* .. *b*].

En los dos siguientes ejercicios debes suponer que la clase *TArbus<TClave,TValor>* está equipada con la operación:

– `recorreClaveValor: Arbus[Clave,Valor] → Sec[Pareja[Clave,Valor]]`

  Obtiene una secuencia, ordenada por *clave*, con todas las parejas de *clave*, *valor* almacenadas en el árbol de búsqueda.

5. El problema de las *concordancias* consiste en lo siguiente: Dado un texto, se trata de contar el número de veces que aparece en él cada palabra, y producir un listado ordenado alfabéticamente por palabras, donde cada palabra aparece acompañada del número de veces que ha aparecido en el texto. Suponemos que el texto a analizar viene dado como secuencia de tipo *Sec[string]*, donde cada elemento de la secuencia es una palabra. Se pide construir un algoritmo que resuelva el problema con ayuda de un árbol de búsqueda de tipo *Arbus[string, int]*, y analizar su complejidad. El listado pedido se dará como secuencia de parejas, de tipo *Sec[Pareja[string, int]]*.

6. Dado un texto organizado por líneas, el problema de las *referencias cruzadas* pide producir un listado ordenado alfabéticamente por palabras, donde cada palabra del texto vaya acompañada de una *lista de referencias*, que contendrá los números de todas las líneas del texto en las que aparece la palabra en cuestión (con posibles repeticiones si la palabra aparece varias veces en una misma línea). Suponiendo que el texto a analizar venga dado como secuencia de tipo *Sec[Sec[string]]* –secuencia de líneas representadas como secuencias de palabras–, construye un algoritmo que resuelve el problema con ayuda de un árbol de búsqueda de tipo *Arbus[string, Sec[int]]*, y analiza su complejidad. El listado pedido se dará como secuencia de parejas, de tipo *Sec[Pareja[string, Sec[int]]]*.

7. Añade a la clase `TArbinDinamico<TElem>` una operación que determine si un árbol binario es un montículo. Analiza su complejidad.

8. Añade a la clase `TArbinDinamico<TElem>` una operación *maxNivel* que obtenga el máximo número de nodos de un nivel del árbol, esto es, el número de nodos del "nivel más ancho". Analiza su complejidad.

9. Se trata de modificar la implementación de la clase `TArbus<TClave,TValor>` para permitir que en un árbol de búsqueda se puedan almacenar distintos valores con la misma clave, lo cual implica cambios en el comportamiento de algunas operaciones de los árboles:

- Al insertar un par <*clave*, *valor*>, si la clave ya se encontrase en el árbol, en lugar de sustituir el valor antiguo por el nuevo, se asociará el valor adicional con la clave.
- La operación de consulta, en lugar de devolver el valor asociado con una clave dada, devolverá una secuencia con los valores asociados con dicha clave. La secuencia estará ordenada según el orden en el que se insertaron los valores en el árbol.
- La operación de borrado deberá eliminar todos los valores asociados con la clave dada.

Implementa en C++:

- Los cambios necesarios en la estructura de datos para permitir la implementación eficiente de las operaciones de este nuevo TAD. En la estructura de datos sólo puedes utilizar tipos predefinidos de C++; está prohibido el uso de otros TADs.
- Las operaciones de inserción y consulta, indicando razonadamente la complejidad temporal de las implementaciones obtenidas.

10. En la clase `TArbinDinamico<TElem>` añade una constructora con el siguiente perfil

```
template <class TElem>
TArbinDinamico::TArbinDinamico (TElem v[], int num );
```

donde *num* es el número de elementos del array *v*. La constructora debe generar un árbol semicompleto con los elementos del array. Indica razonadamente la complejidad temporal de la implementación obtenida.