Algoritmos y Estructuras de Datos

Guía Práctica 7
Diseño con estructuras avanzadas
Segundo Cuatrimestre 2024

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Diseño con Árboles Binarios

Ejercicio 1. Implementamos un **Árbol Binario** (AB) con

```
 Nodo<T>es Struct<dato: T, izq: Nodo, der: Nodo >

Modulo ArbolBinario<T> implementa Árbol Binario<T>{
      var raíz: Nodo<T> // "puntero" a la raíz del árbol

          ...
}
```

1. Escriba en castellano el invariante de representación para este módulo
2. Escriba en lógica el invariante de representación para este módulo usando predicados recursivos.
3. Escriba los algoritmos para los siguientes procs y, de ser posible, calcule su complejidad
   a) `altura(in ab: ArbolBinario<T>): int` // *devuelve la distancia entre la raíz y la hoja más lejana*
   b) `cantidadHojas(in ab: ArbolBinario<T>): bool`
   c) `está(in ab: ArbolBinario<T>, int t: T): bool` // *devuelve true si el elemento está en el árbol*
   d) `cantidadApariciones(in ab: ArbolBinario<T>, int t: T): int`

Ejercicio 2. Un **Árbol Binario de Búsqueda** (ABB) es un árbol binario que cumple que para cualquier nodo N, todos los elementos del árbol a la izquierda son menores o iguales al valor del nodo y todos los elementos del árbol a la derecha son mayores al valor del nodo, es decir

`pred esABB` (a: ArbolBinario$\langle T \rangle$) {
$a = Nil \vee \big($
$(\forall e : T)(e \in elems(a.Izq) \rightarrow e \leq a.dato) \wedge (\forall e : T)(e \in elems(a.Der) \rightarrow e > a.dato) \wedge$
$esABB(d.Izq) \wedge esABB(d.Der)$
$\big)$
}
`aux elems` (a: ArbolBinario$\langle T \rangle$) : conj$\langle T \rangle${
IfThenElseFi$(a = Nil, \emptyset, \{a.dato\} \cup elems(a.Izq) \cup elems(a.Der))$
}

- Implemente los algoritmos para los siguientes procs y calcule su complejidad en mejor y peor caso
  1. `está(in ab: ABB<T>, int t: T): bool` // *devuelve true si el elemento está en el árbol*
  2. `cantidadApariciones(in ab: ABB<T>, int t: T): int`
  3. `insertar(inout ab: ABB<T>, int t: T)`
  4. `eliminar(inout ab: ABB<T>, int t: T)`
  5. `inOrder(in ab: ABB<T>) : Array<T>` // *devuelve todos los elementos del árbol en una secuencia ordenada*
- Asumiendo que el árbol está balanceado, recalcule, si es necesario, las complejidades en peor caso de los algoritmos del ítem anterior
- ¿Qué pasa en un ABB cuando se insertan valores repetidos? Proponga una modificación del módulo que resuelva este problema

Ejercicio 3. Implementar los siguientes TADs sobre ABB. Calcule las complejidades de los procs en mejor y peor caso

1. Conjunto$\langle T \rangle$
2. Diccionario$\langle K, V \rangle$
3. ColaDePrioridad$\langle T \rangle$

Recalcule, si es necesario, las complejidades en peor caso de los algoritmos de los TADs considerando que se implementan sobre AVL en vez de ABB.

Ejercicio 4. Considerar la siguiente extensión sobre el TAD Conjunto$\langle \mathbb{Z} \rangle$

```
TAD Conjunto⟨ℤ⟩ {
obs elems: conj⟨ℤ⟩
proc cantElementosEnRango(in c : Conjunto⟨ℤ⟩, in desde : ℤ, in hasta : ℤ) : ℤ
   requiere { desde ≤ hasta }
   asegura { (∃ c': conj⟨ℤ⟩ )((∀ e: ℤ) ( e ∈ c' ⟺ e ∈ c ∧ desde ≤ e < hasta) ∧ res = |c'| ) } }
```

1. Escriba el algoritmo para el `cantElementosEnRango` si el TAD Conjunto$\langle \mathbb{Z} \rangle$ está implementado sobre ABB. ¿Qué complejidad tiene?
2. Modifique la estructura de representación del módulo para poder realizar la operación `cantElementosEnRango` en $O(ln(n))$ si el árbol se encuentra balanceado, manteniendo la complejidad del resto de las operaciones.
3. Escriba el invariante de representación actualizado. ¿Que sucede con la función de abstracción?
4. Escriba los algoritmos actualizados de `agregar` y `cantElementosEnRango`

Diseño con Heaps

Ejercicio 5. Escriba un algoritmo que verifique si un árbol binario cumple con la propiedad de max-heap

Ejercicio 6. Implemente el TAD ColaDePrioridad$\langle T \rangle$ utilizando heaps (implementados con arreglos).

1. Escriba en castellano y en lógica (mediante predicados recursivos o con alguna otra estrategia vista en clase) el invariante de representación.
2. Escriba los algoritmos para las operaciones `encolar`, `desencolarMax` y `cambiarPrioridad`. Justifique la complejidad de cada operación.

Ejercicio 7. ¿Cómo haría para implementar una ColaDePrioridad ordenada por dos criterios? Por ejemplo, se quiere tener una cola de personas donde el criterio de ordenamiento es por edad y, en caso de empate, por apellido? Describa todos los cambios necesarios a la implementación del ejercicio anterior.

Ejercicio 8. ¿Cómo utilizaría un heap para ordenar una secuencia de elementos? Escriba el algoritmo y calcule su complejidad.

Ejercicio 9. Un heap k-ario (*k-ary heap* o *d-ary heap*) es una generalización del heap tradicional en la cuál cada nodo tiene a lo sumo k hijos. La propiedad de heap se mantiene, es decir, el valor de cada nodo es mayor o igual que el valor de sus hijos.

1. Diseñe el módulo `HeapKArio`. Elija la estructura y escriba los algoritmos
2. Calcule la complejidad de las operaciones
3. ¿Qué diferencias le vé respecto del heap tradicional? ¿Qué ventajas y desventajas observa?

Diseño con Tries

Ejercicio 10. Implemente un Trie utilizando arreglos y listas enlazadas para los nodos.

- Describa en castellano el invariante de representación
- Escriba los algoritmos para las operaciones `buscar` y `agregar` y justifique la complejidad de cada operación.
- ¿Qué diferencias observa entre ambas implementaciones? ¿Qué ventajas y desventajas tiene cada una? En qué casos utilizaría cada una?

Ejercicio 11. Utilizando una estructura de Trie para almacenar palabras, escriba los siguientes algoritmos, justificando la complejidad de peor caso de cada uno:

1. `primeraPalabra`: Devuelve la primera palabra en orden lexicográfico.
2. `ultimaPalabra`: Devuelve la última palabra en orden lexicográfico.
3. `buscarIntervalo`: Dadas dos palabras p_1 y p_2, devolver todas las palabras que se encuentren entre p_1 y p_2 en orden lexicográfico, ordenadas.

Ejercicio 12. Dada una matriz binaria (de unos y ceros), identifique si existen filas repetidas. Debe realizar esto recorriendo la matriz sólo una vez.

Ejemplo:

```
Entrada:
[1  0  0  1  0]
[0  1  1  0  0]
[1  0  0  1  0]
[0  0  1  1  0]
[0  1  1  0  0]

Salida:
{2, 4} (la fila 2 es igual a la fila 0 y la fila 4 es igual a la fila 1)
```

Ejercicio 13. Tenemos un conjunto de palabras escritas en notación "camel case" (por ejemplo `BuscarPalabra` o `DevolverMinimoDelConjunto`). A partir de una secuencia de letras en mayúsculas, se quiere buscar todas las palabras del conjunto que tengan ese patrón.

Ejemplo:

```
Entrada: [BuscarPalabra, BuscarPalabraMasLarga, BuscarPalabraMasCorta, SacarTodo, BuscarTodo]

Si el patrónn es BPM, la salida debería ser [BuscarPalabraMasLarga, BuscarPalabraMasCorta]
Si el patrón es S, la salida debería ser [SacarTodo]
Si el patrón es B, la salida debería ser [BuscarPalabra, BuscarPalabraMasLarga,
    BuscarPalabraMasCorta, BuscarTodo]
```