Algoritmos y Estructuras de Datos

Guía Práctica 7.5
Diseño: Elección de estructuras (parte 2)

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

Elección de estructuras

En esta práctica nos concentraremos en la elección de estructuras y los algoritmos asociados. Cuando el enunciado diga "diseñe este módulo", salvo que indique lo contrario, sólo nos interesa: 1. la estructura, 2. las operaciones con sus parámetros y su complejidad (no los Requiere y Asegura) y 3. los algoritmos correspondientes.

Ejercicio 1. Confeccione una tabla comparativa de las complejidades de peor caso de las operaciones de pertenencia, inserción, borrado, búsqueda del mínimo y borrado del mínimo para conjuntos de naturales sobre las siguientes estructuras:

1. Lista enlazada
2. Lista enlazada ordenada
3. Árbol binarios de búsqueda
4. Árbol AVL

NOTA: Este ejercicio es conceptual, no hay que usar módulos básicos sino imaginar los algoritmos y analizar sus complejidaded en caso de que utilizáramos las estructuras mencionadas para implementar los conjuntos.

Ejercicio 2. Se desea diseñar un sistema para registrar las notas de los alumnos en una facultad. Al igual que en Exactas, los alumnos se identifican con un número de LU. A su vez, las materias tienen un nombre, y puede haber una cantidad no acotada de materias. En cada materia, las notas están entre 0 y 10, y se aprueban si la nota es mayor o igual a 7.

```
TAD Sistema {
      proc RegistrarMateria(inout s: Sistema, in m: materia)

      proc RegistrarNota(inout s: Sistema, in m: materia, in a: alumno, in n: nota)

      proc NotaDeAlumno(in s: Sistema, in a: alumno, m: materia): nota

      proc CantAlumnosConNota(in s: Sistema, in m: materia, n: nota): ℤ

      proc CantAlumnosAprobados(in s: Sistema, in m: materia): ℤ
}
```

Dados $m = cantmaterias$ y $n = cantalumnos$ se desea diseñar un módulo con los siguientes requerimientos de complejidad temporal:

- RegistrarMateria en O(log m)
- RegistrarNota en O(log n + log m)
- NotaDeAlumno en O(log n + log m)
- CantAlumnosConNota y CantAlumnosAprobados en O(log m)

Ejercicio 3. El TAD *Matriz infinita de booleanos* tiene las siguientes operaciones:

- *Crear*, que crea una matriz donde todos los valores son falsos.
- *Asignar*, que toma una matriz, dos naturales (fila y columna) y un booleano, y asigna a este último en esa coordenada. (Como la matriz es infinita, no hay restricciones sobre la magnitud de fila y columna.)
- *Ver*, que dadas una matriz, una fila y una columna devuelve el valor de esa coordenada. (Idem.)
- *Complementar*, que invierte todos los valores de la matriz.

Elija la estructura y escriba los algoritmos de modo que las operaciones *Crear*, *Ver* y *Complementar* tomen $O(1)$ tiempo en peor caso.

Ejercicio 4. Una matriz finita posee las siguientes operaciones:

- *Crear*, con la cantidad de filas y columnas que albergará la matriz.
- *Definir*, que permite definir el valor para una posición válida.
- *#Filas*, que retorna la cantidad de filas de la matriz.
- *#Columnas*, que retorna la cantidad de columnas de la matriz.
- *Obtener*, que devuelve el valor de una posición válida de la matriz (si nunca se definió la matriz en la posición solicitada devuelve cero).
- *SumarMatrices*, que permite sumar dos matrices de iguales dimensiones.

Dado n y m son la cantidad de elementos no nulos de A y B, respectivamente, diseñe un módulo (elegir una estructura y escribir los algoritmos) para el TAD *MatrizFinita* de modo tal que dadas dos matrices finitas A y B,

(a) *Definir* y *Obtener* aplicadas a A se realicen cada una en $\Theta(n)$ en peor caso, y

(b) *SumarMatrices* aplicada a A y B se realice en $\Theta(n + m)$ en peor caso,

Ejercicio 5. Considere el TAD *Diccionario con historia*, cuya especificación es la siguiente:

```
TAD DiccionarioConHistoria<K, V> {
   obs data: dict<K, V>
   obs cant: dict<K, int>

      proc nuevoDiccionario(): DiccionarioConHistoria<K, V>
         asegura { res.data == {}}
         asegura { res.cant == {}}

      proc esta(in d: DiccionarioConHistoria<K, V>, in k: K): Bool
         asegura { res ↔ k in d.data}

      proc definir(inout d: DiccionarioConHistoria<K, V>, in k: K, in v: V)
         asegura { d.data == setKey(old(d).data, k, v)}
         asegura { d.cant == setKey(old(d).cant, k, suma1(k, old(d).cant)}

      proc  obtener(in d: DiccionarioConHistoria<K, V>, in k: K): V
         requiere { k in d.data}
         asegura { v == d.data[k]}

      proc  borrar(inout d: DiccionarioConHistoria<K, V>, in k: K): V
         requiere { k in d.data}
         asegura { d.data == delKey(old(d).data, k)}
         asegura { d.cant == old(d).cant}

      proc  cantSignificados(in d: DiccionarioConHistoria<K, V>, in k: K): ℤ
         requiere { k in d.data}
         asegura { res == d.cant[k]}

      aux suma1(k:K, cant: dict<K, int>)
         { if (k in cant) then (cant[k] + 1) else 1 fi}
}
```

Dado n es la cantidad de claves que están definidas en el diccionario, se debe diseñar este TAD respetando los siguientes órdenes de ejecución en el peor caso:

- `esta` $O(n)$
- `obtener` $O(n)$
- `borrar` $O(n)$
- `nuevoDiccionario` $O(1)$
- `definir` $O(n)$
- `cantSignificados` $O(n)$

Ejercicio 6. Se desea diseñar un sistema de estadísticas para la cantidad de personas que ingresan a un banco. Al final del día, un empleado del banco ingresa en el sistema el total de ingresantes para ese día. Se desea saber, en cualquier intervalo de días, la cantidad total de personas que ingresaron al banco. La siguiente es una especificación del problema.

```
TAD IngresosAlBanco {
  obs totales: seq<ℤ>

    proc nuevoIngresos(): IngresosAlBanco
      asegura {totalDia == []}

    proc registrarNuevoDia(inout i: IngresosAlBanco, in cant: ℤ)
      requiere { cant ≥ 0}
      asegura {i.totales == old(i).totales + [cant]}

    proc cantDias(in i: IngresosAlBanco): ℤ
      asegura {res == |i.totales|}

    proc cantPersonas(in i: IngresosAlBanco, in desde: ℤ, in hasta: ℤ): ℤ
      requiere { 0 ≤ desde ≤ hasta ≤ |i.totales|}
      asegura {res = ∑_{j=desde}^{hasta} i.totales[j]}
}
```

1. Dar una estructura de representación que permita que la función *cantPersonas* tome $O(1)$.
2. Calcular cómo crece el tamaño de la estructura en función de la cantidad de días que pasaron.
3. Si el cálculo del punto anterior fue una función que no es $O(n)$, piense otra estructura que permita resolver el problema utilizando $O(n)$ memoria.
4. Agregue al diseño del punto anterior una operación *mediana* que devuelva el último (mayor) día d tal que cantPersonas$(i, 1, d)$ $\leq$ cantPersonas$(i, d + 1$,totDias$(i))$, restringiendo la operación a los casos donde dicho día existe.