Práctico 7

Objetivos

- Trabajar con los TADs Conjunto y Diccionario.
- Desarrollar y analizar diversas implementaciones para estos TADs.
- Usarlos para la resolución de problemas simples y complejos.

Ejercicio 1 Especificación TAD Conjunto

(a) Escriba un módulo de especificación para el TAD **Conjunto no acotado** de naturales, conteniendo un conjunto mínimo de constructores, selectores, predicados y destructores para:

- Crear un Conjunto vacío.
- Determinar si un Conjunto es vacío o no.
- Inserciones. Borrado.
- Saber si un elemento pertenece a un Conjunto.
- Unión, intersección y diferencia de Conjuntos.

(b) ¿Qué operación u operaciones se deben agregar/quitar para que la especificación sea del TAD **Conjunto acotado** de naturales?

(c) ¿Qué operación u operaciones se deben agregar/quitar para que la especificación sea del TAD **Diccionario** de naturales.

(d) Un MultiConjunto (o Bag) es un tipo abstracto de datos que especifica una colección de elementos en la que, a diferencia del Conjunto, pueden existir ocurrencias repetidas de los mismos. Por ejemplo, el MultiConjunto {1, 3, 3, 4} no es el mismo que el MultiConjunto {1, 3, 4}. El orden de los elementos es irrelevante, distinguiéndose así de la noción de lista.

¿Qué operación u operaciones se deben modificar para que la especificación sea del TAD **MultiConjunto no acotado** de naturales?

Ejercicio 2 Implementación TAD Conjunto

(a) Sabiendo que los elementos que puede contener un Conjunto están en el rango [1,n], desarrolle una implementación completa del TAD del ejercicio 1 parte (b) (TAD **Conjunto acotado** de naturales) usando una estructura de memoria estática, de tal manera que las operaciones de unión, intersección y diferencia tengan $O(n)$ de tiempo de ejecución en el peor caso, donde n es el tamaño máximo de los conjuntos.

(b) Proponga alternativas para la implementación completa del TAD del Ejercicio 1 parte (a) (TAD **Conjunto no acotado** de naturales) y parte (b) (TAD **Conjunto acotado** de naturales). Incluya entre las alternativas AVL y Hash. Presente en formato tabla el orden del tiempo de ejecución de todas las operaciones para cada implementación.

(c) ¿Qué modificaciones se deben hacer para que la implementación de la parte (a) se ajuste al TAD **MultiConjunto** del ejercicio 1 parte d? Considere que los elementos están en el rango [1,n], y que un MultiConjunto puede contener cualquier cantidad de cada uno de ellos.

(d) Implemente las operaciones insertar, pertenece y borrar del TAD **Diccionario** de naturales para implementaciones que cumplan con los siguientes requerimientos:

1. $O(1)$ de tiempo de ejecución en el peor caso de la función insertar.
2. $O(log_2(n))$ de tiempo de ejecución en el peor caso de las funciones insertar, pertenece y borrar.
3. $O(1)$ de tiempo de ejecución promedio para las funciones insertar, pertenece y borrar.

Ejercicio 3 Segundo parcial 2016

Se quiere especificar e implementar un TAD Pedidos que permita registrar una colección de pedidos y atender los pedidos respetando el orden de llegada. Un pedido tiene un identificador que es un número en el rango $[0 : K]$ y una descripción que es un string. Se pide:

(a) Especifique completamente, en C* el TAD Pedidos con operaciones que permitan:

- Crear una estructura de pedidos vacía, que pueda contener hasta $K + 1$ pedidos.
- Insertar un pedido con identificador $i \in (0 \leq i \leq K)$ y descripción d, en una colección de pedidos p, siempre que no exista otro pedido con el mismo identificador i en p; en caso contrario la operación no tendrá efecto.
- Eliminar el pedido más antiguo (el primero ingresado) de una colección de pedidos no vacía y devolver la descripción de dicho pedido.
- Consultar si una colección de pedidos es vacía.
- Consultar si hay un pedido con identificador $i \in (0 \leq i \leq K)$ en una colección de pedidos.

(b) Defina una representación del TAD anterior de tal manera que las operaciones de inserción y eliminación de pedidos tengan O(1) de tiempo de ejecución en el peor caso.

(c) Implemente las operaciones para crear e insertar de la parte a) accediendo directamente a la representación propuesta. Omita el código del resto de las operaciones del TAD (que puede asumir están implementadas).

Ejercicio 4 Segundo parcial 2014

Se desea modelar un TAD Urna que permita registrar votos a diferentes candidatos. Las operaciones con las que debe contar el TAD son las siguientes:

- *urnaVacia*, que crea una Urna vacía.
- *insertarVoto*, que recibe un candidato y una Urna, y agrega a la Urna un voto para ese candidato.
- *cantVotos*, que recibe un candidato y una Urna, y devuelve la cantidad de votos para ese candidato que están en la Urna.

(a) Escriba una especificación del TAD Urna no acotado descrito anteriormente. Asuma que los candidatos son de tipo *string*.

(b) Desarrolle una implementación del TAD Urna especificado en la parte (a), de forma que insertarVoto sea *O(1)* en peor caso.

(c) Indique para cada una de las tres operaciones implementadas en la parte (b) cuál es el orden de tiempo de ejecución en el peor caso. Justifique brevemente.

(d) Si sabe que hay n candidatos y que cada uno de ellos puede identificarse con un número en el rango $[1, n]$, explique qué implementación permitiría que las operaciones *insertarVoto* y *cantVotos* sean *O(1)* en peor caso. ¿Cuál sería el orden de tiempo de ejecución en ese caso de la operación urnaVacia?

Ejercicio 5 Examen Febrero 2015

Una coordenada es un tipo de datos de nombre *Coord* que contiene un par (x, y) de elementos de tipo *int*. El TAD **Coord** contiene las siguientes operaciones:

```
/* Crea una coordenada con el par (x, y) */
Coord crearCoord (int x, int y);


/* Retorna el primer elemento de una coordenada c. */
int coordX (Coord c);


/* Retorna el segundo elemento de una coordenada c. */
int coordY (Coord c);
```

Se cuenta además con el TAD **ConjCoord** con las operaciones tradicionales de conjunto mas las dos operaciones siguientes:

```
/* Retorna un conjunto de coordenadas con todas las coordenadas de ↩
   conj que contienen a x como primer elemento del par. */
ConjCoord subconjX (int x, ConjCoord conj);

/* Retorna un conjunto de coordenadas con todas las coordenadas de ↩
   conj que contienen a y como último elemento del par. */
ConjCoord subconjY (int y, ConjCoord conj);
```

Utilizando las operaciones del TAD **ConjCoord** implemente la operación:

```
/* Retorna todas las coordenadas de conj que se encuentran dentro del↩
    rectángulo formado por los puntos (c1.x, c1.y), (c2.x, c1.y), (c2↩
   .x, c2.y), (c1.x, c2.y). Si hay elementos en el borde del rectá↩
   ngulo también deben aparecer en el conjunto resultado.*/
ConjCoord coordenadasInternas(Coord c1, Coord c2, ConjCoord conj);
```

Por ejemplo:

Conj	C1	C2	**Resultado**
[(1,2), (3,5), (1,1)]	(0,0)	(3,5)	[(1,2), (3,5), (1,1)]
[(1,-2), (3,-5), (-1,-1)]	(0,0)	(3,5)	[]
[(1,-2), (3,-5), (-1,-1)]	(0,0)	(3,-2)	[(1,-2)]

NOTAS: La implementación NO deberá iterar explícitamente sobre todas las coordenadas del rectángulo determinado por las coordenadas c1 y c2 .