AVL, Tablas de dispersión, Conjunto

Objetivos

- Implementar árboles *AVL* y tablas de dispersión.
- Trabajar con el TADs Conjunto y variantes.
- Desarrollar y analizar diversas implementaciones para estos TADs.
- Usarlos para la resolución de problemas simples y complejos.

Ejercicio 1 Tabla de dispersión

Implemente una tabla de dispersión que puede contener hasta n enteros resolviendo los conflictos mediante listas simplemente enlazadas. Se debe poder insertar, remover y buscar elementos. El tiempo de ejecución de las inserciones debe ser $O(1)$ en peor caso.

Ejercicio 2 Especificación TAD Conjunto

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

- Crear un Conjunto vacío.
- Determinar si un Conjunto es vacío o no.
- Incluir y excluir elementos en el Conjunto.
- Saber si un elemento pertenece a un Conjunto.
- Construir 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?

Ejercicio 3 Dominio acotado

Sabiendo que los elementos que puede contener un Conjunto están en el rango [1,n], desarrolle una implementación completa del TAD del ejercicio 2 parte (b) (TAD **Conjunto acotado** de naturales) usando una estructura de memoria estática, de tal manera que el tiempo de ejecución de las operaciones de unión, intersección y diferencia sea $O(n)$ en el peor caso.

Ejercicio 4 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 para las operaciones de inserción y eliminación de pedidos el tiempo de ejecución sea $O(1)$ 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 5 Suma de pares

Dado un entero k y una lista de enteros S, se quiere saber si existe un par de elementos de S cuya suma sea k, o sea si existen x, y en S tal que $x + y = k$. Este problema es una instancia particular del problema más general conocido como *Subset Sum Problem*, donde no hay restricción respecto a la cantidad de sumandos que sumen k. A diferencia del *Subset Sum Problem*, que es de clase *NP-Hard* [1] y admite una variedad de aproximaciones, el problema de la suma de pares puede resolverse en tiempo polinomial.

La solución debe permitir resolver el problema en $O(n)$ promedio, tanto en tiempo como en espacio auxiliar, donde n es la longitud de S.

Se pide:

1. Especifique de manera mínima un `TAD` con el que se resolverá el problema.
2. Resueva el problema aplicando las operaciones de ese `TAD`.
3. Justifique por qué se cumplen los órdenes $O(n)$ para lo cual indique que versión de implementación del `TAD` debe usarse.

Se puede asumir dado el `TAD Lista`, con operaciones `longitud`, `crearLista`, `esVacia`, `insertar`, `primero` y `resto`, y que todas tienen tiempo de ejecución $O(1)$.

Ejercicio 6 AVL

Implemente las operaciones insertar, borrar y buscar en los árboles `AVL`. El tiempo de ejecución de todas ellas debe ser $O(\log n)$, donde n es la cantidad de nodos del árbol.

Dibuje un diagrama para cada uno de los casos.

Ejercicio 7 Comparación de implementaciones

Proponga alternativas para la implementación completa del TAD del Ejercicio 2 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. El orden debe ser ajustado, es decir, debe ser O y también Ω.

Ejercicio 8 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 9 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:

[1] https://es.wikipedia.org/wiki/NP-hard

```
/* 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 .

Ejercicio 10 Intersección de listas

Dadas dos listas de enteros A y B, cada una de las cuales sin elementos repetidos, se quiere calcular de forma eficiente otra lista con los elementos que están en ambas, o sea, la intersección intersección de A y B.

La solución debe permitir resolver el problema en $O(n)$ promedio, tanto en tiempo como en espacio auxiliar, donde n es la suma de las longitudes de las listas.

Se pide:

1. Especifique de manera mínima un TAD con el que se resolverá el problema.
2. Resueva el problema aplicando las operaciones de ese TAD.
3. Justifique por qué se cumplen los órdenes $O(n)$ para lo cual indique que versión de implementación del TAD debe usarse.

Se puede asumir dado el `TAD Lista`, con operaciones `longitud`, `crearLista`, `esVacia`, `insertar`, `primero` y `resto`, y que todas tienen tiempo de ejecución $O(1)$.

Observación
Las listas no necesariamente están ordenadas.