Mapping

Objetivos

- Trabajar con el TAD Mapping.
- Desarrollar y analizar diversas implementaciones para este TAD.
- Utilizar el TAD Tabla para la resolución de problemas simples y complejos.
- Introducir MultiConjuntos.

Ejercicio 1 Especificación TAD Tabla

Considere el TAD **Tabla no acotada** (también llamado Mapping o Asociación) con elementos del dominio **D** y elementos del rango **R**, conteniendo un conjunto mínimo de operaciones para:

- Crear una Tabla vacía.
- Determinar si una Tabla es vacía o no.
- Insertar parejas (d, r), con d en **D** y r en **R**, tal que se define $T(d) = r$, independientemente de que $T(d)$ estuviera ya definido.
- Determinar si $T(d)$ está definido.
- Supresiones, que borran $T(d)$ de la tabla, si está definido.
- Recuperar el valor $T(d)$, si está definido en la tabla.

(a) Desarrollar una especificación de dicho TAD.

(b) ¿Qué modificaciones deberían hacerse a la especificación anterior para obtener una especificación del TAD **Tabla acotada**?

Ejercicio 2 Implementación TAD Tabla con Hash

Se quiere implementar una tabla acotada, cuyo dominio son las cadenas de caracteres y su recorrido son los enteros, utilizando una tabla de dispersión abierta y en donde la resolución de colisiones usa *separate chaining* (resolución de colisiones usando listas).

(a) Analice el orden del tiempo de ejecución de las operaciones del TAD tabla implementado mediante una tabla de dispersión abierta. Compare el orden del tiempo de ejecución de cada operación con implementaciones alternativas.

(b) Analice diferentes opciones para la función de hash a utilizar, comentando pros y contras. A modo de ejemplo, para hacer corresponder el tipo *String* con el tipo *int* podría considerar la suma de la codificación ASCII de cada caracter de cada palabra. A su vez, pueden existir varias formas de combinar estos valores para computar un valor de hash. Se sugiere además, considerar si conviene o no utilizar todos los caracteres de cada palabra, si las palabras distribuyen bien las letras del abecedario, si esto depende del idioma, entre otras.

(c) Implemente asumiendo que se dispone del TAD Lista, el cuál debe especificar.

Ejercicio 3 Implementación TAD Tabla en $O(\log(n))$ (Examen Agosto 2020)

Considere el TAD Tabla no acotada de *unsigned int* (dominio) en *float* (codominio).

(a) Desarrolle una especificación de dicho TAD.

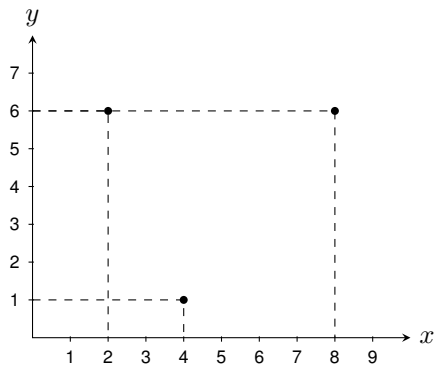
(b) Implemente el TAD Tabla de tal manera que el tiempo de ejecución de las operaciones de *insertar*, *recuperar* y *eliminar* sea $O(\log(n))$ en el caso promedio, siendo n la cantidad de correspondencias de la tabla.

(c) Realice las modificaciones necesarias a su implementación anterior de forma de poder agregar la operación *cantidad* que devuelva la cantidad de correspondencias en la tabla en $O(1)$ en el peor caso.

Ejercicio 4 Aplicación TAD Tabla (Similar a segundo parcial 2017)

Se debe implementar el TAD `Grafica` que modela mediante puntos una función parcial de enteros positivos en enteros positivos. Se considera un punto (x, y) como una asociación de una coordenada x cuyo valor asociado es la coordenada y. La cantidad máxima de asociaciones, N, que se puede mantener en un momento determinado está acotada por un parámetro definido en el constructor. Por ser una función, no puede haber dos puntos diferentes con la misma coordenada x. Se puede asumir que todos los puntos tienen la misma probabilidad de pertenecer a una gráfica. La especificación de `Grafica` se encuentra en el Listado 1.

Ejemplo:



En la gráfica representada en el diagrama hay 3 coordenadas para las cuales se mantiene una asociación. El valor asociado a 2 es 6, el valor asociado a 4 es 1 y el valor asociado a 8 es 6.
Si $N = 3$ no se podrían agregar nuevas asociaciones, aunque éstas pueden eliminarse y también actualizarse (ver las operaciones en la especificación del TAD `Grafica`).

Listado 1: Especificación de `Grafica`

```
/* Devuelve una gráfica vacia que puede mantener hasta 'N'
   asociaciones.
   Precondición: N > 0. */
Grafica CrearGrafica(int N);

/* Si en 'g' hay menos de 'N' asociaciones, asigna 'y' como valor
   asociado a 'x';
   en caso contrario la operación no tiene efecto.
   Precondición: x > 0, y > 0, Valor(x, g) == -1. */
void Asociar(int x, int y, Grafica &g);


/* Si en 'g' existe asociación para 'x', devuelve el valor
   asociado a 'x';
   en otro caso devuelve -1.
   Precondición: x > 0. */
int Valor(int x, Grafica g);

/* Elimina de 'g' la asociación para 'x', si existe tal asociación
   ;
   en caso contrario la operación no tiene efecto.
   Precondición: x > 0. */
void Desasociar(int x, Grafica &g);
```

(a) Implemente el TAD `Grafica` usando dispersión abierta con `h(x) = x % N` como función *hash*. Los tiempos de ejecución de la operación `Asociar` deben ser $O(1)$ en el peor caso, mientras que los de `Valor` y `Desasociar` deben ser $O(1)$ en el caso promedio.
Utilice la definición de $Punto$ y la especificación de $Lista$ de puntos (que se asume implementada) del Listado 2.

Listado 2: Definición de `Punto` y especificación de `Lista` de puntos

```
struct Punto { int x, y; };

/* Devuelve una lista vacia. */
Lista CrearLista();

/* Agrega 'pt' al inicio de 'l'. */
void Cons(Punto pt, Lista &l);

/* Devuelve el primer elemento de 'l'.
   Precondición:
    ! EsVaciaLista(l). */
Punto Primero(Lista l);

/* Devuelve 'l' sin el primer elemento.
   Precondición:
    ! EsVaciaLista(l). */
Lista Resto(Lista l);

/* Devuelve true si y solo si 'l' es vacia. */
bool EsVaciaLista(Lista l);

/* Devuelve true si y solo si en 'l' hay un punto cuya primera coordenada es 'x'. */
bool ExisteX(int x, Lista l);

/* Remueve el primer elemento de 'l' cuya primera coordenada es 'x'.
   Precondición: ExisteX(x,l). */
void RemoverX(int x, Lista &l);
```

(b) Justifique brevemente por qué su implementación de `Grafica` cumple con los tiempos de ejecucion requeridos.

El tiempo de ejecución en el peor caso de las operaciones `CrearLista`, `Cons`, `Primero`, `Resto` y `EsVacialista` es $O(1)$; el de `ExisteX` y `RemoverX` es $O(n)$, siendo n la cantidad de elementos de la lista.

Ejercicio 5 Autenticación de credenciales

La firma **LogIn** trabaja con clientes muy exigentes y necesita implementar un sistema de verificación de credenciales que sea muy eficiente. A tales efectos ha facilitado el listado de usuarios/contraseñas para que se implemente el sistema de acreditación. A partir de un vector de parejas $\{usr, pass\}$, se pide:

Implementar una aplicación que:

- Solicite las credenciales al usuario a través de la consola.
- Verifique que el usuario está registrado y que su clave es correcta eficientemente (O(1) en promedio).
- Advierta al usuario en caso de éxito o fracaso del proceso de autenticación.

Ejercicio 6 Consolidar tablas (Examen Agosto 2020)

Una empresa almacena los sueldos de sus empleados en tablas (de tipo *Tabla*, según la especificación habitual) donde el dominio de tipo *unsigned int* corresponde al número de empleado y el codominio de tipo *float* corresponde al salario del empleado.

La empresa quiere consolidar salarios dispersos en varias tablas, para esto se propone implementar una función *consolidar* que, dadas dos tablas *t1* y *t2* (de tipo *Tabla*) genere una nueva tabla (de tipo *Tabla*) que contenga los sueldos de los empleados que están en *t1* o en *t2*. Si un empleado está en las dos tablas, se tomará la suma de los salarios que tenga en ambas tablas, y si está solamente en una tabla, se tomará su sueldo en dicha tabla. Para la operación *consolidar* se considerarán solamente empleados cuyos número de empleado sea mayor o igual que *inf* y menor o igual que *sup* (asumimos *inf* <*sup*). **Implemente *consolidar* sin acceder a la representación del TAD *Tabla*.**

```
Tabla consolidar (Tabla t1, Tabla t2, unsigned int inf,
                  unsigned int sup)
```

Ejercicio 7 MultiConjuntos

(a) 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?

(b) Considerando la implementación realizada para resolver el **Ejercicio 3, Dominio acotado, del Práctico AVL, Tabla de dispersión, Conjunto**, ¿qué modificaciones deben hacerse para que dicha implementación se ajuste al TAD MultiConjunto de la parte anterior? Considere que los elementos están en el rango [1,n], y que un MultiConjunto puede contener cualquier cantidad de cada uno de ellos.

(c) Discuta cómo sería la implementación de un MultiConjunto utilizando un ABB. ¿Qué consideraciones deberíamos tener en cuenta? Si un elemento se encuentra varias veces en el MultiConjunto, ¿cuántos nodos tendríamos en el ABB para dicho nodo?

Ejercicio 8 Examen Diciembre 2020

Considere la siguiente especificación del `TAD MultiSet` no acotado de números enteros.

```
struct RepresentacionMultiset;
typedef RepresentacionMultiset * Multiset;
// POS: Devuelve el multiset vacío.
Multiset crear();

// POS: Agrega n ocurrencias del elemento x al multiset m. PRE: n>0.
void insertar (Multiset & m, int x, int n);

// POS: Devuelve la cantidad total de elementos del multiset m (0 si ↩
   m está vacío).
int cantidad (Multiset m);

// POS: Devuelve la cantidad de ocurrencias del elemento x del ↩
   multiset m (0 si x no está en m).
int ocurrencias (Multiset m, int x);

/* POS: Elimina a lo sumo n ocurrencia del elemento x del multiset m.↩
    Si ocurrencias(m, x)<=n entonces en m no quedarán ocurrencias del↩
    elemento x, y se liberará la memoria correspondiente. */
void eliminar (Multiset & m, int x, int n);

// POS: Retorna el mínimo elemento del multiset m (independientemente↩
    del número de ocurrencias). PRE: m no vacío.
int min (Multiset m);

// POS: Retorna el máximo elemento del multiset m (independientemente↩
    del número de ocurrencias). PRE: m no vacío.
int max (Multiset m);
```

Se pide:

(a) Implemente el `TAD Multiset` anterior de tal manera que el tiempo de ejecución de las operaciones `insertar`, `ocurrencias` y `eliminar` sean $O(\log(n))$ en el caso promedio, siendo n la cantidad de elementos diferentes del multiset, y el de las operaciones `crear`, `cantidad`, `min` y `max` sea $O(1)$ peor caso. Concretamente, defina en `C#` la representación del `TAD` y escriba los códigos de las operaciones `insertar`, `ocurrencias` y `min`. Omita el código del resto de las operaciones, que puede asumir están implementadas.

(b) Implemente el procedimiento `vaciar` que dado un multiset `m`, según la especificación dada previamente, elimine todos sus elementos, dejando a éste vacío. El procedimiento vaciar no debería acceder a la representación del `TAD` ni dejar memoria colgada (sin liberar).

```
void vaciar (Multiset & m)
```