Año 2025

Operaciones para manipular conjuntos

Ya hemos establecido que los conjuntos que almacenaremos en la memoria de una computadora vienen dados por extensión, son finitos y en general cambiarán su contenido al transcurrir el tiempo.

A estos conjuntos cuyo contenido variará en el tiempo, los denominamos dinámicos. Por lo tanto, para reflejar ese dinamismo, vamos a necesitar contar con operaciones que permitan:

- incorporar o memorizar un nuevo elemento en el conjunto: *ALTA*.
- eliminar u olvidar un elemento que estaba en el conjunto: *BAJA*.

En general, los conjuntos con los que vamos a trabajar tienen elementos no tan simples como sólo un número o un nombre sino más bien alguna combinación de ellos, porque provienen de la realidad.

Entonces, para cada elemento querremos registrar cierta información sobre valores que lo caracterizan. A las características $D_1, \ldots, D_k$ que se desean registrar para cada elemento del conjunto se las denomina *atributos* o *dominios*.

Ejemplo:

Si deseamos mantener la información sobre los empleados de una empresa, claramente es un conjunto cuyo contenido variará en el tiempo. Por ejemplo, es posible que nos interese guardar para cada empleado datos como:

- Nombre y Apellido,
- Nro. Documento,
- Nro. Legajo,
- Categoría.

□

En este caso, los elementos del conjunto deberían estar compuestos de un valor para cada uno de los atributos o dominios que nos interesa registrar. Así, cada elemento se podría ver como una nupla, es decir como un elemento del producto cartesiano de todos los atributos, y entonces nuestro conjunto sería realmente una relación como:

$$R \subseteq D_1 \times D_2 \times \ldots \times D_k$$

Ejemplo:

Podemos ver nuestro conjunto de datos de los empleados como una relación:

$$Empleados \subseteq \text{Nombre y Apellido} \times \text{Nro. Documento} \times \text{Nro. Legajo} \times \text{Categoría}$$

□

Como ahora los elementos del conjunto no son simples, no será lo usual preguntar si un elemento, una nupla, pertenece al conjunto porque deberíamos aportar los valores para cada uno de los atributos.

En general, en el ámbito de esta asignatura, supondremos que hay uno de los atributos que "identifica" a la nupla y preguntaremos si hay un elemento en el conjunto cuyo "identificador" coincida con el valor que buscamos. Decir que uno de los atributos "identifica" la nupla significa que no pueden existir dos nuplas que tengan el mismo valor para dicho atributo.

Es decir, que dado el valor del atributo aportado deseamos recuperar, si hay una nupla con ese valor de identificador, los valores que corresponden a los demás atributos; así, intentamos recuperar una nupla por algo que conozco de ella.

A esta operación la llamaremos *evocación asociativa* porque queremos "evocar" el elemento del conjunto, si lo hay, cuyo valor para el atributo aportado se "asocie" con el valor que buscamos.

Existen además otros tipos de evocaciones distintas, tales como las *evocaciones extremales* [1] y *evocaciones secuenciales* [2].

A los atributos o dominios de la relación los podemos clasificar en dos grupos:

- los que se aportan en la evocación y
- los que se reciben en la evocación.

Ejemplo:

En el ejemplo que estamos considerando:

Aportado: Nro. Legajo.

Recibidos: Nombre y Apellido, Nro. Documento y Categoría.

□

Para mostrar qué dominios o atributos se aportan y cuáles se reciben adoptamos la siguiente convención: *debajo de cada uno de los dominios que se aportan escribimos un "*****" y debajo de cada dominio o atributo que se recibe colocamos un "**?**"*.

Ejemplo:

Para describir cuál es la evocación asociativa sobre la relación *Empleados* escribiríamos:

$$\begin{array}{lcccccccc} \textit{Empleados} & \subseteq & \text{Nombre y Apellido} & \times & \text{Nro. Documento} & \times & \text{Nro. Legajo} & \times & \text{Categoría} \\ & & \mathbf{?} & & \mathbf{?} & & \mathbf{*} & & \mathbf{?} \end{array}$$

□

A fin de simplificar nuestra notación, podemos agrupar todo lo aportado bajo el nombre de X y todo lo recibido bajo el nombre de Y y así podemos ver a R como una relación definida entre X e Y, y a la evocación asociativa, con asociante X y asociado Y, como:

$$\begin{array}{lccc} R & \subseteq & X & \times & Y \\ & & \mathbf{*} & & \mathbf{?} \end{array}$$

Ahora X e Y son dominios "conceptuales", aunque cada uno de ellos pueda constar de uno o más dominios o atributos reales. Entonces, consideraremos que nuestro conjunto contendrá ahora pares de elementos o nuplas de la forma: (*asociante*, *asociado*), o simplemente (x, y).

[1] En una evocación extremal recuperamos un elemento del conjunto que se encuentra en un extremo, de acuerdo a un determinado orden de los elementos del conjunto.

[2] En una evocación secuencial se devuelven total o parcialmente las nuplas ordenadas por algún atributo.

En el ámbito de nuestra materia trabajaremos bajo el supuesto que se cumple que $X \rightarrow Y$, que será equivalente a decir que no pueden existir dos elementos o nuplas con igual valor para X.

$X \rightarrow Y$ se lee "X ***determina funcionalmente a*** Y", o "Y ***depende funcionalmente de*** X". Formalmente, $X \rightarrow Y$ denota la existencia de una ***dependencia funcional*** entre los dominios X e Y [3].

Ejemplo:

Si el conjunto a almacenar considera para un negocio registrar en cada fecha el monto total facturado en el día, la relación *Facturación* la describiríamos como:

$$\textit{Facturación} \subseteq \text{Día} \times \text{Mes} \times \text{Año} \times \text{Monto Total}$$

Si en este caso estamos interesados en la siguiente evocación asociativa: *dada una fecha en particular, recuperar el monto total facturado*. La representación sería:

$$\begin{array}{lcccccccc} \textit{Facturación} & \subseteq & \text{Día} & \times & \text{Mes} & \times & \text{Año} & \times & \text{Monto Total} \\ & & * & & * & & * & & ? \end{array}$$

en este caso se debe cumplir la siguiente dependencia funcional: $\underbrace{\{\text{Día, Mes, Año}\}}_{X} \rightarrow \underbrace{\text{Monto Total}}_{Y}$.

□

Hasta ahora habíamos visto distintas maneras de alojar los elementos de un conjunto en un conjunto de celdas de memoria. Ahora nuestros conjuntos son relaciones, pero a pesar de ello todo lo que vimos se sigue aplicando si consideramos conceptualmente sólo el valor del dominio X y vemos al valor del dominio Y como un campo adicional.

Para analizar en mayor detalle las operaciones que vamos a necesitar para trabajar sobre relaciones que son dinámicas, debemos considerar lo siguiente:

- Es de buena práctica comprobar antes de agregar un elemento si éste ya no estaba (para preservar la definición de conjunto).
- Para eliminar un elemento del conjunto, primero debemos ubicar dónde se encuentra en la estructura.
- Para resolver la pertenencia, también debíamos ubicar dónde se encontraba el elemento buscado en la estructura para responder con éxito.
- Para resolver la evocación asociativa debemos encontrar el elemento cuyo valor de X sea el buscado y devolvemos el valor de Y asociado.

Como se puede observar, todas estas operaciones o rutinas necesitarían poder buscar un elemento en la estructura, es decir localizarlo. Aparece entonces la *localización* como una rutina útil a todas las operaciones sobre conjuntos. Por lo tanto, *altas*, *bajas*, *pertenencia* y *evocación asociativa* invocarán a la *localización*.

[3] En la materia de *Organización de Archivos y Base de Datos I* estudiarán los aspectos teóricos de las dependencias funcionales en el contexto de las bases de datos relacionales.

El encabezado de la rutina que realice la localización de un elemento en la estructura básicamente será:

Localizar (**in** x, **out** pos, **out** *éxito*)

Convención: En un pseudo-código de una rutina, cuando declaramos los parámetros, utilizaremos la palabra **in** para indicar que un parámetro es *de entrada* y la palabra **out** para indicar que es *de salida*. Un parámetro de entrada es aquél en que la rutina espera recibir datos desde el exterior y uno de salida es aquél en el que la rutina va a dejar datos que serán útiles para quien la invoca.

Ahora veremos un pseudo-código genérico, es decir independiente de la estructura que usemos para almacenar la relación, para cada una de las operaciones. Dado que no detallaremos ninguna estructura particular habrá, en cada caso, parte del código donde sólo indicaremos la acción conceptual y no sus detalles.

ALTAS:

La rutina conceptualmente debería ser:

```
Alta ( in x, in y, out éxito)
    ⋮
    Localizar (x, pos, éxito')
    if éxito' then /* existe una nupla con ese x */
        éxito ← false
        if (elemento en pos).y = y then /* nupla repetida */
            ⋮
        else /* x repetido */
            ⋮
    else /* no hay una nupla con ese x */
        if hay espacio then
            ⋮
            modificación en la estructura para reflejar la incorporación de (x, y)
            ⋮
            éxito ← true
        else /* no hay espacio */
            éxito ← false
```

Un alta exitosa sería aquella en que la localización fracasó y que disponíamos de espacio para alojar al nuevo elemento de la relación. En caso contrario, podemos distinguir que el alta puede fracasar por dos motivos:

- por falta de espacio, o
- porque la nupla ya existía o se dejaba de cumplir la dependencia funcional.

Cabe destacar que no basta en esos casos con saber que el alta fracasó, sino que es necesario distinguir por cuál de estos dos motivos fue. Esto se debe a que los motivos son conceptualmente distintos y saber a cuál de ellos se debió el fracaso determinará qué hacer a continuación:

- pedir más espacio para la estructura (si existe más disponible) o
- no hacer nada porque es un error conceptual intentar memorizar en la relación algo que ya existía, o intentar violar la dependencia funcional.

Notar entonces que el parámetro *éxito* no puede ser booleano porque hay que distinguir tres valores:

- alta exitosa,
- alta que fracasa por falta de espacio y
- alta que fracasa por error de intentar memorizar algo ya existente o intentar violar la dependencia funcional.

La pregunta de dónde poner un nuevo elemento se resuelve pensando en el algoritmo de localización que se usa. Un nuevo elemento se debe poner en la secuencia de lugares que este algoritmo examinará no bien pase por uno libre o por la posición en donde se esperaría encontrarlo. Esta posición es aquella en la que fracasa la localización, es decir el punto en que se pudo inferir que ya no se encontrará una nupla con ese valor de x. La localización debería devolver esa posición en el parámetro *pos*.

BAJAS:

La rutina conceptualmente debería ser:

```
Baja ( in x, in y, out éxito)
    ⋮
    Localizar (x, pos, éxito')
    if éxito' then /* existe una nupla con ese x */
        if (elemento en pos).y = y then /* es la nupla que buscamos */
            ⋮
            modificación en la estructura para reflejar el olvido de (x, y)
            ⋮
            éxito ← true
        else /* no es la nupla que queremos eliminar */
            ⋮
            éxito ← false
    else /* no hay una nupla con ese x */
        ⋮
        éxito ← false
```

En este caso para que la baja sea exitosa debe serlo también la localización; pero dado que la localización sólo busca una nupla que coincida con el valor de x provisto, debemos además controlar que el valor de y también coincida mirando la parte Y de la nupla alojada en la posición pos devuelta por la localización.

En algunas situaciones, en que el usuario quiere dar de baja la nupla (x, y), la solución anterior

podría no ser la deseada porque es engorroso para él aportar ambas componentes x e y completas [4]. Entonces, una opción sería dar como entrada a la rutina de *Baja* sólo el x y, luego de localizar con éxito la nupla que coincide en su parte X con x, pedir la confirmación del usuario; es decir, se le muestra al usuario la parte Y correspondiente y se la consulta si coincide con el y de la nupla que se desea eliminar; si el usuario confirma el y se da de baja la nupla, en otro caso la baja fracasa.

PERTENENCIA:

La rutina conceptualmente debería ser:

```
Pertenencia ( in x, in y, out éxito)
   ⋮
   Localizar (x, pos, éxito')
   if éxito' then /* existe una nupla con ese x */
      if (elemento en pos).y = y then /* es la nupla que buscamos */
          éxito ← true
      else /* no es la nupla que buscamos */
          éxito ← false
   else /* no hay una nupla con ese x */
      éxito ← false
```

EVOCACION ASOCIATIVA:

La rutina conceptualmente debería ser:

```
Evocación ( in x, out y, out éxito)
   ⋮
   Localizar (x, pos, éxito')
   if éxito' then /* existe una nupla con ese x */
      y ← (elemento en pos).y
      éxito ← true
   else /* no hay una nupla con ese x */
      éxito ← false
```

Cabe aclarar que sólo podremos usar el valor de salida del parámetro y cuando la evocación haya sido exitosa, en otro caso y no contendrá nada válido. Así, quien invoque a la evocación asociativa deberá primero verificar que ésta haya sido exitosa antes de utilizar el valor de y devuelto. Además se puede observar que se podría invocar a la rutina *Localizar* con el mismo parámetro *éxito*, quedando el código de la siguiente manera:

[4] En nuestro ejemplo de los empleados de una empresa, para dar de baja un elemento no sólo se debería especificar el Nro. de Legajo, sino también Nombre y Apellido, Nro. Documento y Categoría. Además si se comete algún error al ingresar alguno de los datos la baja fracasará.

```
Evocación ( in x, out y, out éxito)
    ⋮
    Localizar (x, pos, éxito)
    if éxito then /* existe una nupla con ese x */
        y ← (elemento en pos).y
        /* si éxito es ''false'' no existe una nupla con ese x */
```

Modificaciones:

En algunos casos podría ser de interés permitir modificar alguna de las componentes de una nupla (x, y); es decir, cambiar el valor x por otro x' o el valor y por otro y'.

Como en algunas estructuras el lugar en que se aloja una nupla depende de su valor de X, si permitiéramos cambiar x por x' esta modificación podría implicar sacar la nupla de donde estaba y ubicarla en una nueva posición. Así, conceptualmente, podríamos ver a esta operación como la secuencia:

1. olvidarnos de la nupla (x, y), y luego
2. memorizar una nueva nupla (x', y).

Por lo tanto una modificación de este tipo se debería realizar como una baja de (x, y) seguida de un alta de (x', y).

Entonces consideraremos como una *modificación* solamente a aquella operación que nos permita cambiar el valor de Y asociado a un determinado valor de X.

MODIFICACIÓN:

La rutina conceptualmente debería ser:

```
Modificación ( in x, in y, out éxito)
    ⋮
    Localizar (x, pos, éxito')
    if éxito' then /* existe una nupla con ese x */
        (elemento en pos).y ← y
        éxito ← true
    else /* no hay una nupla con ese x */
        éxito ← false
```

Importante:

Es muy buena práctica de programación sólo permitir que las estructuras interactúen con el programa principal, o quien las invoque, a través de sus rutinas de *Alta*, *Baja*, *Pertenencia*, *Evocación Asociativa* y *Modificación*, o aquellas definidas para tal efecto y de los parámetros de las mismas.

La *Localización* es una rutina que sólo debe invocarse dentro de estas rutinas y nunca en el programa principal, esto se debe a que fuera del ámbito de las rutinas que manejan la estructura no se deben dar a conocer posiciones de la misma para evitar que se puedan producir modificaciones no deseadas de la estructura.

Es muy importante que las rutinas sólo se comuniquen con el programa principal, o quien las invoque, a través de sus parámetros. Por ejemplo, en una evocación asociativa por X no se debe imprimir el valor de Y dentro de la evocación, ya que la evocación no sabe qué deseará hacer quien la invoca con el valor de Y que se devuelve.

En las estructuras hay que saber determinar cuándo está vacía y en algunos casos cuándo está llena (en aquellas que tienen previamente limitado su espacio disponible).

Reconocimientos

El presente apunte se realizó tomando como base notas de clases de Estructuras de la Información y de Estructuras de Datos y Algoritmos del Profesor Hugo Ryckeboer.