Estructuras de Datos y Algoritmos
Grado en Ingeniería Informática
Universidad de Zaragoza – curso 2023-2024

Sesión de problemas 7
Tablas multidimensionales

Objetivos

- Diseño de estructuras e implementación (en pseudocódigo) de diversas operaciones con tablas multidimensionales.

1. Agencia de turismo "Bon Voyage"

Se quiere informatizar la agencia de turismo "*Bon Voyage*", que organiza viajes a ciudades a un precio determinado, que depende de la compañía de transportes elegida. Las compañías se identifican por su nombre. Concretamente, las operaciones son:

```
crear: -> agencia
{Devuelve una agencia sin compañías, viajes ni ciudades}
ofrece: agencia a, cadena ciudad, entero precio, cía comp -> agencia
{Devuelve la agencia resultante de incrementar la oferta de la agencia a en que se
ofrece viajes a una ciudad, a un precio dado y trabajando con la compañía dada}
ofertasCiudad: agencia a, cadena ciudad -> listaDePares_NombreCía_Precio
{Devuelve una lista con todos los pares <nombreDeCompañía,precio> que representan
viajes que la agencia a organiza a la ciudad dada}
másBaratos: agencia a, entero precio -> listaDePares_Ciudad_NombreCía
{Devuelve una lista con todos los pares <ciudad,nombreDeCompañía> que representan
viajes que la agencia a organiza más baratos que el precio dado}
```

Se supone que ya están especificados los tipos `entero`, `cadena`, `cía` (con, al menos, la operación `nombre` que da el nombre de la compañía), `par` (genérico; con, al menos, la operación `crearPar` que crea un par a partir de sus dos elementos) y `listaDePares` (genérico; con, al menos, las operaciones `crearVacía` y `añadirDch`).

(a) Se sabe que el tipo `cía` contiene mucha información (aparte del nombre de la compañía). Se sabe además que cada agencia trabaja con muchísimas ciudades (decenas de miles... ¡y cada vez más!) y con pocas compañías de transporte (no más de un centenar). **Propón una representación del tipo `agencia` (basta con explicar la representación con palabras y la ayuda de un dibujo) de manera que**:

(1) para un valor constante del número de compañías, el coste asintótico de `ofertasCiudad` sea logarítmico en el número de ciudades, que

(2) no se malgaste espacio inútilmente (en particular, ni los nombres de las ciudades ni de las compañías ni sus ofertas pueden estar duplicados), y que

(3) aunque no aparece en la especificación, sea posible implementar fácilmente una operación que genere la lista de ciudades a las que viaja una compañía.

(b) Dadas la especificación y la representación anteriores, **implementa la operación `ofertasCiudad` (con el coste mencionado)**, que tiene la siguiente signatura:

```
procedimiento ofertasCiudad(ent a: agencia; ent nombreCiu: cadena;
                                        sal l: listaNomCíaPrecio)
```

Define previamente todos aquellos tipos imprescindibles para escribir el procedimiento.

2. Recetario de cocina

Se trata de desarrollar un recetario de cocina con las siguientes operaciones:

```
crear: -> recetario
meter: recetario r, receta unaReceta -> recetario
cambiar: recetario r, receta unaReceta -> recetario
recBarata: recetario r, entero precio, fecha f -> receta
recBuena: recetario r, cadena ingrediente -> receta
```

La información asociada a una receta es: *nombre*, *precio*, *ingrediente principal*, *otros ingredientes*, *explicación de cómo se hace* y *última fecha en que fue utilizada*.

La operación `crear` tiene el significado obvio; `meter` añade una nueva receta (no se permite que haya dos recetas con el mismo nombre); `cambiar` debe buscar una receta del mismo nombre que la dada y actualizarla (es decir, sustituir el resto de la información por la indicada); `recBarata` debe devolver una receta cuyo precio sea inferior al número natural indicado y que no haya sido utilizada desde al menos una semana antes de la fecha (si no hay tal receta en el recetario, debe devolver un huevo frito); `recBuena` debe devolver una receta cuyo ingrediente principal sea la cadena dada (si no hay ninguna, devuelve un huevo frito).

a) **Especifica los TADs `receta` y `recetario`**. Pueden utilizarse, además de `bool` y `entero`, los TAD `cadena de caracteres` (dotado de una operación de igualdad `=` entre cadenas) y `fecha` (dotado de una operación `dif: fecha f1, fecha f2 -> entero` que devuelve el número de días transcurridos entre el primer argumento y el segundo).

b) **Desarrolla un módulo que implemente el TAD `recetario` con las operaciones antes especificadas mediante una tabla dispersa con el nombre de la receta como clave de acceso utilizando un vector de 997 componentes**, con las siguientes características:

- la función de dispersión (o localización) se implementará mediante el método de la división;

- las posibles colisiones se resolverán mediante recolocación en el mismo vector, utilizando una función de recolocación cuadrática.

c) Supón que se ha especificado e implementado, en forma análoga al recetario, una `bodega` o colección de vinos (mediante una tabla dispersa con el nombre del vino como clave de acceso). **Propón una estructura de datos, definiendo todos los tipos necesarios, que permita implementar las operaciones siguientes**:

```
platos?: recetario r, bodega b, vino v -> listaDeRecetas
caldos?: recetario r, bodega b, receta unaReceta -> listaDeVinos
```

que proporcionan la lista de recetas que van bien con el vino dado y la lista de vinos que van bien con la receta dada, respectivamente. **La estructura de datos debe permitir que la implementación de ambas operaciones sea muy eficiente** (de coste lineal en el tamaño de la lista resultante).

3. TAD `clan`

Se ha definido un TAD `clan` que permite representar relaciones (simétricas) de amistad y enemistad entre personas con las operaciones:

- `cvacío`: crea un clan vacío.
- `añadeAmigos`: dado el clan `c` y dos personas `p1` y `p2`, añade a `c` la información de que `p1` y `p2` son amigos; lógicamente, `p1` y `p2` se incorporan implícitamente al clan si no lo estaban ya; si `p1` y `p2` ya eran amigos en `c`, entonces la operación no produce ningún efecto; si `p1` y `p2` eran enemigos en `c`, entonces la operación también nos añade la información de que `p1` y `p2` han dejado de ser enemigos.
- `añadeEnemigos`: dado el clan `c` y dos personas `p1` y `p2`, añade a `c` la información de que `p1` y `p2` son enemigos; lógicamente, `p1` y `p2` se incorporan implícitamente al clan si no lo estaban ya; análogamente a la operación anterior, si `p1` y `p2` ya eran enemigos en `c`, entonces la operación no produce ningún efecto; y si `p1` y `p2` eran amigos en `c`, entonces la operación también nos añade la información de que `p1` y `p2` han dejado de ser amigos.
- `sonAmigos?`: dado el clan `c` y dos personas `p1` y `p2`, dice si `p1` y `p2` son amigos en `c`.
- `posiblesAliados?`: dado el clan `c` y dos personas `p1` y `p2`, dice si `p1` y `p2` tienen algún enemigo en común.

Se debe proponer una representación (estructura de datos) para el TAD `clan` para la que la eficiencia en tiempo de las operaciones sea lo mejor posible (se puede suponer que las personas se identifican por un dato de tipo cadena). Explica la estructura propuesta (incluyendo, en particular, un esquema o dibujo comentado) y detalla las definiciones de tipos de datos necesarias. Justifica además las decisiones tomadas y razona cuál es el coste de las operaciones en el caso peor.

4. Cartelera de cine

Se desea gestionar la información de la cartelera de cine, lo que básicamente incluye la información sobre en qué *cines* y *salas* se proyecta cada *película*. Sabemos que tendremos que gestionar la información de un número indeterminado, y bastante grande, de películas que están proyectándose actualmente en alguna sala de los numerosos cines que se incluyen en nuestra cartelera. Cada cine se identifica con su nombre, está situado en una localidad, y cuenta con una o más salas de proyección. Cada cine identifica sus salas mediante un número, y cada una de ellas cuenta con una cierta capacidad máxima (número de butacas). De cada película en cartelera contamos con bastante información, incluyendo: su *título*, *director*, *actores*, *año*, *fecha de estreno*, *género*, *descripción* y *duración* (en minutos). Una película puede estar proyectándose en varios cines, o incluso en varias salas del mismo cine. Un cine puede tener varias películas en cartelera, tanto si cuenta con una única sala como con varias. Para cada proyección de una película, que se realizará en un determinado cine y sala, y en una determinada fecha y hora, nos interesará también registrar el número de entradas vendidas, para posteriormente calcular la taquilla realizada por los diferentes cines y películas.

Se pide:

1.- **Diseña una estructura de datos que permita almacenar toda la información que se necesita para la gestión de la cartelera de cine**, propón una representación de los tipos de datos implicados (definiendo los tipos de datos y explicando la representación de toda la estructura con palabras, y además con un dibujo), de forma que no malgaste espacio en memoria, y que se optimicen (mínimo coste) las operaciones habituales de mantenimiento y consulta.

2.- **Implementa la siguiente operación, indicando el coste asintótico**:

- *Registrar la venta de una entrada para la proyección de una película, en un determinado cine, sala, fecha y hora, si es que quedan butacas disponibles para ella.*

3.- **Describe detalladamente cómo se utilizarían las estructuras de datos diseñadas para la implementación de la siguiente operación**, razonando el coste asintótico esperado:

- *Calcular el total de entradas vendidas para una película dada, en un periodo de fechas concreto.*

