Multiestructuras

Objetivos

- Practicar los conceptos básicos de multiestructuras
- Practicar los conceptos básicos del Diseño de TADs

Multiestructuras

Ejercicio 1 Ranking

Se desea mantener una escala de equipos de fútbol en la que cada equipo esté situado en un único puesto. Los equipos nuevos se agregan en la base de la escala, es decir, en el puesto con numeración más alta. Un equipo puede retar a otro que esté en el puesto inmediato superior (el i al $i-1$, $i > 1$), y si le gana, cambia de puesto con él. Se sabe que hay un máximo de n equipos

Se puede representar la situación anterior mediante un TAD cuyo modelo fundamental sea una correspondencia de nombres de equipos (cadenas de char) con puestos (enteros 1, 2, ...).

Defina una una representación que contemple las siguientes operaciones:

- agregar(nombre): agrega el equipo nombrado como últimos en la escala (al puesto de numeración más alta).
- rivalDirecto(nombre): es una función que devuelve el nombre del equipo del puesto $i-1$ si el equipo nombrado está en el puesto i, $i > 1$.
- intercambia(i): intercambia los nombres de los equipos que estén en los puesto i e $i-1$, $i > 1$.

Los tiempos de acceso por nombre deben ser O(1) caso promedio y por posición en el ranking O(1) peor caso.

Ejercicio 2 Carreras 100 metros

Se desea contar con una estructura de datos para mantener información de atletas que participan en la competencia final de los 100 metros llanos en los juegos olímpicos. Los datos que interesan sobre cada competidor son la posición (1 a 8) y el año en que compitió. Cada competidor está identificado por un código (entero). Si un atleta participó más de una vez, aparece con la mejor posición que obtuvo. Pensar en una representación del problema, para que las siguientes consultas a la estrucutra se realicen en los órdenes solicitados:

- Obtener una lista de los competidores que salieron alguna vez en la primera posición. O(1) peor caso.
- Obtener el año en que un competidor obtuvo su mejor posición. O(log n) en caso promedio.
- Imprimir los competidores ordenados por código. O(n) peor caso.
- Obtener la lista de competidores (para imprimir) que alguna vez estuvieron en el puesto k (k entre 1 y 8). O(1) peor caso.
- Obtener cual es el puesto más alto obtenido por un competidor. O(log n) en caso promedio.
- Poder realizar operaciones de inserción, supresión y búsqueda de competidores en una posición. O(log n) en caso promedio.

Se pide:

(a) Presente una representación para resolver el ejercicio anterior.

(b) Escriba un procedimiento con el encabezado *void borrarCompetidor (Tabla100m &tabla, int codigo)* que borre de *tabla* al competidor con código *codigo* e indique el orden de ejecución para el caso promedio del procedimiento.

Ejercicio 3 Empleados

Los empleados de cierta compañía se representan en la base de datos de la compañía por su nombre (que se supone único), número de empleado y número de seguridad social. Se sabe que la cantidad de empleados puede estimarse en un valor K.

Se pide:

(a) Presente una estructura de datos que permita, dada la representación de un empleado, encontrar las otras dos representaciones del mismo individuo de la forma más rápida en promedio y evitando redundancia de información.

(b) Dada su representación, ¿qué tan rápida, en promedio, puede lograrse que sea cada una de estas operaciones? Justifique.

(c) Defina en C* la estructura del inciso anterior.

(d) Desarrolle en C* la operación que dado el nombre de un empleado imprime su número de empleado y número de seguridad social, si el empleado existe. Para definir el procedimiento puede suponer definidas las funciones de hash sobre cadenas de caracteres que considere necesarias.

(e) Considerando la estructura por usted definida, ¿pueden imprimirse los empleados de la compañía ordenados por su nombre de forma ordenada en O(n)? Justifique.

Ejercicio 4 Alumnos

Una institución desea mantener información de los alumnos de una carrera universitaria. Para cada alumno interesa su código de alumno de tipo alfanumérico de 10 caracteres (que se supone único), nombre completo (que se supone único), la colección de materias que aprobó, junto con las notas de aprobación obtenidas, y el conjunto de los compañeros con los cuales realizó algún proyecto a lo largo de su carrera. Cada materia está identificada por un código, que se supone único. Estos códigos son números posibles entre 0 y un entero positivo L. Cada alumno puede tener a lo sumo K compañeros de proyectos diferentes a lo largo de toda su carrera. La relación “compañero de proyecto”, entre dos alumnos, es simétrica (necesariamente). Se sabe que el número de alumnos está acotado por un valor M y se pude asumir que L =O(log M).

Se quieren satisfacer los siguientes requerimientos:

- *Nombre nombre (Estructura e, CodigoAlumno cod)* en el cual dada la estructura que representa los datos del problema y el código *cod* de un alumno, devuelve el nombre del alumno correspondiente. Si el código de alumno no existe la operación retorna “NN” como nombre. Esta operación se debe realizar en O(1) caso promedio.
- *CodigoAlumno codigo (Estructura e, Nombre nom)* en el cual dada la estructura que representa los datos del problema y el nombre *nom* de un alumno, devuelve el código de alumno correspondiente. Si el nombre del alumno no existe la operación retorna “-1” como código. Esta operación se debe realizar en O(1) caso promedio.
- *void listado (Estructura e)* en el cual dada la estructura que representa los datos del problema, imprime los nombres de los alumnos ordenados alfabéticamente. Esta operación se debe realizar en O(n) peor caso, siendo n la cantidad actual de alumnos.
- *void asignarCompañero (Estructura &e, Nombre nom, CodigoAlumno cod*) en el cual dada la estructura que representa los datos del problema, el nombre *nom* de un alumno y el código *cod* de otro alumno, los asigna como compañeros de proyecto, siempre que esto sea posible. La operación no tiene efecto si *cod* o *nom* no están registrados como alumnos en la estructura, o si se excede el máximo de compañeros permitidos para *nom* o para el alumno con código *cod*. Asumimos que ambos alumnos no son ya compañeros de proyecto. Esta operación se debe realizar en O(1) caso promedio.
- *void cargarMateria (Estructura &e, CodigoAlumno cod, CodigoMateria cm, Calificación cal)* en el cual dada la estructura que representa los datos del problema, el código *cod* de un alumno, el código *cm* de una materia y una calificación *cal*, agrega a la materia de código *cm*, y con calificación *cal*, a la colección de materias aprobadas por el alumno, siempre que esto esa posible. Si el alumno tiene ya la materia aprobada, actualiza la calificación con *cal*. Esta operación se debe realizar en O(1) caso promedio.
- *void agregarAlumno (Estructura &e, Nombre nom, CodigoAlumno cod)* en el cual dada la estructura que representa los datos del problema, el nombre *nom* de un alumno y su código de alumno *cod*,

agrega al alumno en la estructura, con ninguna materia cursada y un conjunto vacío de compañeros de proyectos. En caso de encontrar algún alumno con el mismo código *cod* o con el mismo nombre *nom* devuelve un mensaje de error sin modificar la estructura. Esta operación se debe realizar en O(log n) caso promedio, siendo n la cantidad actual de alumnos.

Se pide:

(a) Diseñar estructuras de datos apropiadas para implementar eficientemente dichas operaciones, describiendo como se obtienen las cotas de tiempo pedidas.

(b) Dar una declaración en C* de los tipos de datos descriptos en la parte anterior.

(c) Escribir en C* el procedimiento agregarAlumno. A los efectos de definir este procedimiento hay que utilizar solamente las estructuras definidas en la parte (a). Puede asumir definidas las funciones de hash sobre cadenas de caracteres que crea necesarias para la resolución del problema. Así como también que las cadenas de caracteres se pueden comparar con *==* y *!=* y asignar con *=*.

Diseño de TADs

Ejercicio 5 Cola de Prioridad

(Examen Agosto 2003)

(a) Realizar una especificación procedural del TAD Cola de Prioridad no acotada de elementos de un tipo genérico *T* donde las prioridades estén dadas por números naturales, dando un conjunto mínimo de operaciones necesarias. Se requiere que si dos elementos tienen igual prioridad en la cola de prioridad, se respeta la política *FIFO* (el primero en entrar es el primero en salir).

(b) Desarrollar una implementación del TAD anterior donde para la operación de inserción no se recorran los elementos ya existentes en la Cola de Prioridad. Para las demás operaciones del TAD no hay restricciones. Se debe dar una representación e implementar completamente las operaciones definidas en la parte anterior. *No se permite usar TADs auxiliares en este ejercicio.*

Ejercicio 6 TAD Pila

(Examen Diciembre 2003)

(a) Especifique un TAD *T* de elementos de un tipo genérico que permita almacenar a lo sumo *K* elementos donde se respeta la política *LIFO* (el último en entrar es el primero en salir). La operación de inserción no debe tener precondición, esto es: siempre se pueden agregar elementos pero sólo los últimos *K* (a lo sumo) se consideran.

(b) Desarrolle en una implementación del TAD *T* en la que las operaciones constructoras, selectoras y predicados se realicen sin recorrer la estructura.

Ejercicio 7 Matriz

(Examen Julio de 2012)

(a) Especifique en C*, con pre y postcondiciones un TAD Matriz de Naturales que contenga operaciones que permitan:

- **Crear** una matriz $n \times m$ de naturales, donde n es la cantidad de filas y m la cantidad de columnas. La matriz creada contiene un cero en cada entrada (fila, columna) definida. Se asume que n y m son mayores que 0.
- Dados una matriz M, un número de fila f y un número de columna c, **devolver** el natural ubicado en la entrada (f, c) de M.
- Dados una matriz M, un número de fila *f*, un número de columna *c* y un natural *elem*, **asignar** *elem* a la entrada (f,c) de M.
- **ImprimirFila** e **ImprimirColumna**, que dadas una matriz M y un número de fila o de columna imprimen todos los elementos de la fila o de la columna, respectivamente.

(b) Las matrices *dispersas* son aquellas donde la mayoría de sus elementos valen 0. Para almacenar este tipo de matrices se busca optimizar el uso de memoria NO reservando memoria para aquellos elementos de la matriz que valgan 0, buscando otra forma de representarlos. Implemente el TAD Matriz anterior, asumiendo que lo que se quiere almacenar son matrices dispersas. No implemente las operaciones, simplemente defina las estructuras de datos necesarias, explicando la representación diseñada haciendo hincapié en cómo representa los elementos que valen 0.

(c) Suponga ahora que se le solicita implementar las operaciones ImprimirColumna e ImprimirFila, definidas en la **parte a**, en $O(n+m)$ peor caso, siendo *n* la cantidad de filas y *m* la cantidad de columnas de la matriz. ¿Es posible hacer esto con la implementación que propuso en la **parte b**? Si es así, justifique y si no, provea una nueva representación que sí lo permita.

(d) Implemente las operaciones (definidas en la **parte a**) que permiten **crear** una matriz y **asignar** un valor natural a una entrada (f,c) de una matriz, sobre la representación del TAD Matriz resultante de la **parte c**.

Ejercicio 8 TAD ConcCarnaval

Considere el TAD *ConcCarnaval* que permite registrar los nombres de las agrupaciones de carnaval y para cada una su puntaje en el concurso. Es decir que este TAD es una función parcial entre strings del dominio (posibles nombres de las agrupaciones de carnaval) y enteros del codominio (puntaje actual de cada agrupación). Los puntajes están en el rango [0,N].

Se pide:

(a) Especifique completamente el TAD *ConcCarnaval* (con pre y post condiciones), que tenga al menos operaciones para:

- Crear un *ConcCarnaval* vacío.
- Asociar a *agrup* el valor *punt* en un *ConcCarnaval*; si ya tenía un valor asociado, lo redefine.
- Consultar si un *agrup* tiene asociado un puntaje en un *ConcCarnaval*.
- Devolver puntaje del codominio asociado a *agrup* en un *ConcCarnaval*.
- Eliminar la asociación de *agrup* en un *ConcCarnaval*.
- *ImprimirAgrup* imprime las agrupaciones en orden alfabético con sus respectivos puntos.
- *ListaPuntos* devuelve la lista de las agrupaciones que tienen p puntos.

(b) Proponga una implementación del TAD *ConcCarnaval*, defina el tipo para la representación elegida del TAD y desarrolle el código de las operaciones, de modo que las operaciones selectoras y la operación de inserción sean O(log n) de tiempo de ejecución en el caso promedio, siendo n la cantidad de agrupaciones. El orden del tiempo de ejecución en el peor caso de *ImprimirAgrup* debe ser O(n). El orden del tiempo de ejecución en el peor caso de *ListaPuntos* debe ser O(1). **Explique cómo la implementación elegida permite cumplir con los requerimientos**.

Ejercicio Complementario

Ejercicio 9 Gestión de memoria

(Examen Diciembre 2006)

Se quiere diseñar un TAD que permita realizar el manejo dinámico de la memoria. Considere la memoria como un espacio compuesto por una cantidad finita pero muy grande de bytes, los cuales pueden estar en dos estados: ocupados o libres. Un bloque es un espacio contiguo de bytes, ya sean libres u ocupados. El TAD solamente se encargará de considerar que bloques están siendo usados y cuales están libres, no se considerará que información esta siendo guardada en cada uno de los bloques ocupados. Las operaciones sobre la memoria que se han identificado como necesarias son las siguientes:

- **InicializarMemoria**: Inicializa la memoria, colocándola como libre. Es decir, existe un único bloque con toda la memoria, el cual está libre.

- **SolicitarMemoria(k)**: Se solicita un bloque libre de tamaño *k* bytes. En caso de existir solamente bloques libres de tamaño mayores a *k* bytes, se debe asignar el bloque de tamaño más pequeño disponible; el espacio sobrante del bloque debe permanecer libre. Se debe indicar si fue posible realizar o no la asignación de memoria, además de indicar la dirección de comienzo del bloque asignado, cuando corresponda.
- **LiberarMemoria(dir)**: Se solicita la liberación del bloque ocupado que comienza en la dirección dir. Se debe liberar el bloque y considerar si existen bloques contiguos libres con los cuales fusionarse para lograr un único bloque de tamaño más grande.

(a) Especifique un TAD Gestor de Memoria de acuerdo a lo descrito arriba.

(b) Proponga una implementación dinámica del TAD anterior. Defina en la estructura utilizada para representar el TAD.

(c) Implemente las operaciones del TAD. Respecto a las operaciones del TAD correspondientes a **SolicitarMemoria** y **LiberarMemoria**, desarrolle solamente el código de una de estas dos operaciones, a elección.