

Trabajo Práctico Especial: Análisis y diseño de algoritmos

Sistema de ranking de empresas



30 de junio de 2021

Alumno: Avalos Enzo

E-mail: [avalos.enzo.g@gmail.com](mailto:avalos.enzo.g@gmail.com)

Grupo: 54

Institución: Universidad del Centro de la Provincia de Buenos Aires

Carrera: Ingeniería de Sistemas

Especificación formal en Nereus de los TDA identificados:

* **TDA Empresa**

CLASS Empresa

BASIC CONSTRUCTORS crearEmpresa

IMPORTS Natural, Cadena

EFFECTIVE

TYPE Empresa

OPERATIONS

crearEmpresa: Cadena \* Cadena \* Natural(emp) \* Cadena -> Empresa

pre: (emp > 0);

\_=\_: Empresa \* Empresa -> Empresa;

obtenerRazonSocial: Empresa -> Cadena;

obtenerPaisOrigen: Empresa -> Cadena;

obtenerRubro: Empresa -> Cadena;

obtenerEmpleados: Empresa -> Natural;

* **TDA Lista**

CLASS Lista [Elemento]

BASIC CONSTRUCTORS inicLista, agregarElemento

IMPORTS Natural

EFFECTIVE

TYPE Lista

OPERATIONS

inicLista: -> Lista;

agregarElemento: Lista \* Elemento \* Natural(p) -> Lista

pre: (p > 0);

listaVacia: Lista -> Boolean;

eliminarLista: Lista(l) -> Lista

pre: not listaVacia(l);

longLista: Lista -> Natural;

recuperarElemento: Lista(l) \* Natural(p) -> Elemento

pre: not listaVacia(l) and (p>0) and (p <= longLista(l));

* **TDA Ranking**

CLASS Ranking

BASIC CONSTRUCTORS inicRanking, agregarElemento

IMPORTS Empresa, Lista[Empresa], Cadena, Natural

EFFECTIVE

TYPE Ranking

OPERATIONS

inicRanking: -> Ranking;

agregarElemento: Ranking \* Empresa -> Ranking;

cantidadElementos: Ranking -> Natural;

vacio: Ranking -> Boolean;

busquedaPorNombre: Ranking(r) \* Empresa -> Natural

pre: not vacio(r);

busquedaPorPosicion: Ranking(r) \* Natural(p) -> Empresa

pre: (not vacio(r)) and (p > 0) and (p <= cantidadElementos(r));

busquedaPorRango: Ranking \* Natural(i) \* Natural(m) -> Lista

pre: (i >= 0) and (m > 0) and (i < m);

Estructuras de datos Implementadas sobre los TDA:

Cada clase presentada a continuación fue implementada en un encabezado propio, y separando su declaración de su implementación, en otro archivo fuente.

* **TDA Empresa**: implementado mediante una clase cuyas instancias representan una empresa en sí misma.
* **TDA Lista**: para implementar la estructura de lista, se utilizó una clase genérica cuya instancia representa una lista, con métodos propios al funcionamiento de esta estructura. Almacena en sus nodos las posiciones en el ranking de las empresas que cumplan con las condiciones buscadas.
* **TDA Ranking**: el TDA ranking es una abstracción del método usado para la resolución del problema planteado, la búsqueda de empresas con ciertas restricciones. Por esto, no hay una estructura especifica al ranking en la implementación. Sin embargo, para poder realizar su tarea de forma correcta y eficiente, debe tener conocimiento de otras estructuras tanto de almacenamiento como de organización de datos, las cuales son:
* **Arreglo**: se implementó una clase arreglo genérica, cuya instancia contiene una referencia a un bloque continuo de memoria dinámico, que contiene instancias de la clase empresa.
* **Árbol binario de búsqueda**: la estructura ABB se implementó mediante una clase, donde sus instancias representan arboles con una estructura interna de nodos, y estos contienen una referencia a un objeto de empresa ubicado en el arreglo de datos, así como su posición correspondiente en el ranking. En la forma que fue implementada, la estructura permite tener distintas instancias de árboles con diferentes criterios de ordenamiento para sus nodos, en este caso particular, ordenado por razón social o por posición en el ranking.

Descripción y complejidad temporal:

Se describen todas las operaciones asociadas a los TDA propuestos antes, pero teniendo en cuenta la forma en la que fueron implementados.

**Empresa:**

* crearEmpresa: corresponde al constructor de la clase Empresa, con ciertos argumentos, su complejidad temporal es constante O(1).
* obtenerRazonSocial: devuelve la razón social de la empresa, complejidad temporal constante O(1).
* obtenerPaisOrigen: devuelve el país de origen de la empresa, complejidad temporal constante O(1).
* obtenerRubro: devuelve el rubro de la empresa, complejidad temporal constante O(1).
* obtenerEmpleados: devuelve el número de empleados de la empresa, complejidad temporal constante O(1).

**Lista:**

* agregarElemento: crea una nueva instancia de lista y la inserta en la última posición de la misma, debiendo recorrer la lista en su totalidad, por lo que su complejidad temporal es lineal y depende de la longitud “n” de la lista, es decir, O(n).
* listaVacia: verifica si la lista contiene elementos, para realizarlo solo se debe verificar si la referencia a la lista es nula, por lo que su complejidad temporal es constante O(1).
* eliminarLista: elimina todos los “n” nodos de la lista, por lo que su complejidad será de O(n).
* recuperarElemento: esta implementado como un cursor que apunta secuencialmente a los nodos de la lista, y mediante él se recuperan los datos almacenados en estos. Para recuperar todos los elementos de la lista, se deben recorrer los “n” nodos de esta, por lo que su complejidad será O(n).

**Ranking:**

* agregarElemento: se agrega un elemento del tipo empresa a un arreglo mediante un acceso indexado por lo que su complejidad temporal es constante O(1).
* cantidadElementos: esta función devuelve la cantidad de elementos almacenados en el arreglo, el cual se conoce por dato ya que corresponde a su tamaño, su complejidad temporal es constante O(1).
* Vacío: determina si el arreglo de datos asociados está vacío, esta acción requiere la invocación de cantidadElementos que tiene complejidad constante, por lo que la complejidad temporal de Vacío también es constante O(1).
* busquedaPorNombre: consiste en verificar si una cierta empresa existe dentro del ranking, y en dicho caso retornar su posición. Se implemento como una búsqueda sobre un árbol binario ordenado por razón social de la empresa. Complejidad temporal teórica O(n), donde n es la cantidad de empresas del ranking.
* busquedaPorPosicion: consiste en devolver la empresa que se encuentra en una posición determinada del ranking. Se implemento como una búsqueda sobre un árbol binario balanceado, ordenado por la posición en el ranking de cada empresa. Complejidad temporal teórica O(log n), donde n es la cantidad de empresas del ranking.
* busquedaPorRango: consiste en elaborar una lista de empresas con una cantidad de empleados dentro de un rango especificado. Esta operación se realiza mediante una función que cuenta con un bucle for que actúa sobre todas las “n” empresas en el arreglo de datos, insertando en una lista las empresas que cumplan las condiciones. Esta función de inserción debe recorrer los “n” nodos de la lista para insertar un nuevo elemento, que en el caso que todas las empresas cumplan con las condiciones, es igual al total de las empresas. Debido a esto la complejidad teórica total de la operación será O(n2).

Combinación de los TDA:

Las empresas por individual se implementaron como instancias de una clase Empresa que contiene todos los datos asociados a la misma. Estas instancias correspondientes a las empresas fueron almacenadas en una clase Arreglo de forma secuencial, la cual quedo con las empresas ordenadas por su posición en el ranking de forma ascendente, usando las posiciones del arreglo como indicadores de la posición en el ranking de la empresa almacenada. Luego, de forma paralela se usaron 2 árboles binarios implementados mediante clases, para almacenar referencias a las distintas posiciones del arreglo que contiene a las empresas, cada uno con un criterio de orden distinto para una mejora en la eficiencia de las búsquedas. Por último, se implementó una estructura de lista en forma de clase la cual almacena en sus nodos las posiciones de las empresas en el ranking, usada como colección de retorno para el servicio de búsqueda por rango de empleados.

Servicios requeridos y sus características:

**1\ Obtener la posición en el ranking para una empresa en particular**

- TDAs usados para su implementación:

Este servicio se implementó como una búsqueda sobre una instancia de la clase árbol binario, que se encuentra ordenado por razón social de la empresa, el cual contiene en sus nodos, referencias a objetos Empresa, almacenados en un bloque contiguo de memoria, atributo de la clase Arreglo. En caso de que la empresa buscada exista en el ranking, se retorna la posición de la empresa en el mismo, caso contrario se retorna un valor destacado que indica que la empresa no existe.

- Complejidad temporal:

Su complejidad temporal teórica será de O(n) ya que en caso de que los datos se encontraran ordenados al insertarlos en el árbol, este se asemejaría a una lista, y para verificar si una empresa se encuentra en el habría que recórrelo por completo. Aun así este caso es muy poco probable, teniendo generalmente un árbol más balanceado, por lo que en la práctica su complejidad temporal podrá ser O(log n) para ciertos casos.

- Estructura de datos elegida:

La estructura de datos elegida favorece el costo de la resolución, ya que, para la mayoría de los casos no es necesario recorrer la totalidad de los datos para llegar a esta. En este caso, la mejora no se observará en la complejidad teórica, que seguirá siendo la misma, sino que esta decisión de implementación impactará puramente en la práctica.

**2\ Obtener la información de la empresa que se encuentra en una posición determinada**

**-** TDAs usados para su implementación:

Este servicio se implementó como una búsqueda sobre una instancia de la clase árbol binario, idéntico al usado al servicio anterior, con la diferencia que este se encuentra ordenado por posición de las empresas en el ranking. Para esto se recibe una posición y si la misma es válida y corresponde con el tamaño del ranking, se devuelve una referencia a la empresa buscada, caso contrario se retorna un valor nulo.

- Complejidad temporal:

Este árbol se genera sobre un arreglo previamente ordenado de forma descendiente por posición en el ranking, insertando primeramente el valor en la mitad del arreglo, luego insertando los valores en la mitad de cada subarreglo resultante, y así sucesivamente. Por esto mismo se obtiene un árbol completamente balanceado y la complejidad temporal tanto teórica como practica será O(log n).

- Estructura de datos elegida:

La estructura de datos elegida favorece el costo de la resolución, ya que al partir de un arreglo ordenado por el mismo criterio que el árbol binario a elaborar, con una correcta implementación al momento de su creación, se obtiene un árbol balanceado y es posible disminuir significativamente la complejidad del servicio completo. Cabe destacar que, para la creación de un árbol con estas características, se requiere recorrer el arreglo de datos en su totalidad O(n), pero solamente al inicio del programa y no cada vez que se requiera hacer uso de este servicio, por lo que esto se ve justificado.

- Mejoras posibles:

Una forma de mejorar la complejidad del servicio seria usando un arreglo ordenado por posición en el ranking, de esta forma se podría implementar este servicio de búsqueda mediante un acceso indexado al arreglo, el cual tiene una complejidad temporal constante de O(1).

**3\ Listar las empresas, junto a su posición en el ranking, que cuentan con un número de empleados dentro de un rango especificado**

- TDAs usados para su implementación:

Para este servicio, se reciben dos parámetros, la cantidad mínima y máxima de empleados, y si son válidos, se realiza una búsqueda secuencial sobre una instancia de la clase Arreglo. Simultáneamente se elabora una lista enlazada, implementada mediante una clase Lista, donde se guardan las posiciones en el ranking de las empresas que cumplan con las restricciones indicadas.

- Complejidad temporal:

La función de inserción perteneciente a la lista, tiene una complejidad de O(n), con n correspondiente a la longitud de la lista, y al estar anidada dentro de un bucle for que recorre la totalidad del arreglo también en O(n), con n cantidad de elementos en el arreglo, la complejidad del servicio resulta O(n2). Se podría mejorar la complejidad práctica, para algunos casos, si se tuviera en cambio otra estructura de datos u otro criterio de orden sobre el arreglo de datos implementado, aunque esto no afectaría a la complejidad teórica. Por otra parte, la complejidad teórica de este servicio podría mejorarse si la inserción en la lista fuera de orden constante O(1), lo que se lograría si se insertara cada nuevo elemento al inicio de la lista, lo que llevaría a una complejidad teórica total del servicio de O(n), debido a que igualmente se debe recorrer los “n” elementos del arreglo para confeccionar la lista.

- Estructura de datos elegida:

La estructura del arreglo en la forma que es usada en este caso, no es la solución más eficiente para este servicio, ya que, aunque la complejidad teórica seguiría siendo igual, se podría mejorar el costo practico usando otra disposición del mismo.

- Mejoras posibles:

Una forma de mejorar el servicio seria elaborar otra estructura de datos, pero con un criterio de orden basado en la cantidad de empleados de las empresas. De esta forma, incluso si igualmente se usara un arreglo o una lista, al estar ordenado de forma ascendente en cantidad de empleados, cuando se encuentre una empresa con mayor cantidad de empleados que los buscados, se podría interrumpir la búsqueda, ya que se sabe que los nodos siguientes tampoco cumplirán con la consigna buscada. Otra estrategia seria usar un árbol binario con el mismo criterio, ya que nuevamente en ciertos casos se podría bajar la complejidad teórica, al cortar ciertas ramas que se sabe que no aportaran datos relevantes. Igualmente, si el rango especificado por el usuario abarcara todas las empresas, se debería recorrer cualquier estructura de datos implementada en su totalidad. Sin embargo, teniendo en cuenta que este servicio va a ser requerido varias veces durante un mismo día, sería una buena idea intentar bajar su complejidad, aunque exija una mayor cantidad de recursos al inicio, y solo sirva para ciertos casos.