Código: 75.001

Asignatura: Estructura de la Información

Curso: 2009/2010 – primer semestre

PEC 2

Ejercicio 1 [2 puntos]

Apartado 1.1) Describid el funcionamiento de un algoritmo que permita fusionar dos árboles AVL de tamaño N y M respectivamente. ¿Qué coste tiene esta operación de fusión?

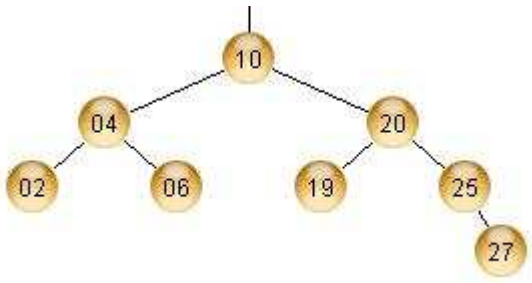
Apartado 1.2) Implementad en Java una extensión de la clase ArbolAVL de la biblioteca de clases para dotarla de un método que fusione el árbol actual con un árbol que recibís como parámetro. El método debe tener la siguiente firma:

```
public void fusionar(ArbolAVL arbol)
```

Apartado 1.3) Aplicad el algoritmo de fusión para fusionar los elementos del árbol X con el árbol Y y mostrad el árbol AVL resultado. Si la operación de fusión provoca desequilibrios comentad qué desequilibrio se produce y con qué rotación se soluciona.

Árbol X

Árbol Y



Ejercicio 2 [2 puntos]

Apartado 2.1) Implementad en Java un TAD que os permita gestionar los empleados de una empresa multinacional con las siguientes premisas:

- No podéis usar delegación.
- Usad las clases de la JCF.
- El número de empleados es muy grande y conocido.
- Se requiere un acceso rápido a los datos del empleado a partir de su dni.
- Para cada empleado guardaremos los siguientes datos:
  - Dni

    - Nombre
    - Apellidos
    - Teléfono
- Debéis ofrecer operaciones para:
    - Dar de alta un empleado
    - Consultar un empleado por dni
    - Dar de baja un empleado
    - Consultar si un dni corresponde a un empleado
    - Listar todos los empleados
    - Listar todos los dni de los empleados
    - Consultar el número d'empleados o Consultar el número de empleados con un nombre determinado
    - Añadir un conjunto d'empleados

Apartado 2.2) Implementad en Java un TAD que os permita gestionar los nombres de los empleados de una empresa multinacional con las siguientes premisas:

- No podéis usar herencia.
- Utilizad las clases de la biblioteca de clases de la asignatura.
- El número de empleados es grande pero desconocido.
- Se requiere una consulta eficiente.
- Debéis ofrecer operaciones para:
    - Dar de alta un nombre, si el nombre ya existe no hace nada
    - Consultar si un nombre existe
    - Eliminar un nombre
    - Listar todos los nombres
    - Consultar el número de nombres que empiecen por un carácter
    - Añadir un conjunto de nombres

Ejercicio 3 [2,5 puntos]

Apartado 3.1) Suponed que hemos diseñado una función de dispersión que tiene como claves los nombres y apellidos de los estudiantes matriculados en la UOC; suponed también que entre nombre y apellidos se pueden ocupar 60 caracteres. Se define la función de dispersión de la siguiente manera:

$$h(c_1...c_{60}) = (\sum k: 1 \leq k \leq 60 : ascii(c_k) * 2^k) \bmod 256$$

Comenta los posibles inconvenientes de la función de dispersión escogida y proponed una alternativa que los solucione.

Apartado 3.2) ¿Es cierto que en un TAD tabla con una función de dispersión mal diseñada se puede degradar el rendimiento de la operación de búsqueda por clave hasta llegar a ser lineal respecto del número de elementos de la tabla? Poned un ejemplo de función de dispersión con este comportamiento.

Apartado 3.3) Suponed que tenéis un TAD tabla y que no conseguís una función de dispersión que proporcione una distribución equiprobable de los elementos, cómo podríais mejorar la eficiencia de la búsqueda sin variar la función de dispersión?

Ejercicio 4 [3,5 puntos]

Debemos implementar un prototipo para realizar votaciones por Internet experimentando un sistema electoral con listas abiertas. on este sistema los candidatos pertenecen a algún partido pero los electores no votan a un partido, sino a un candidato concreto.

Una vez realizadas las votaciones se hace la división entera entre el número total de votantes y los E escaños, obteniendo, así, la *cuota* necesaria para obtener un escaño. Todos los candidatos que están por encima de la *cuota* obtienen escaño, y los que están por debajo quedan provisionalmente fuera.

Si el proceso se acaba aquí es muy probable que no se ocupen todos los E escaños, puesto que seguramente algunos de los candidatos pueden haber superado la *cuota* y un candidato solo puede ocupar un escaño, aunque le hayan votado para ocupar uno o dos. Con el fin de ocupar todos los E escaños, se debe redistribuir este excedente de *cuota*, pero ¿a quién? Nosotros nos interesamos por la siguiente solución:

En el momento de votar se pide a los electores que ordenen los candidatos de más a menos prioritario. De esta manera, el excedente de votos de un candidato X se puede redistribuir de acuerdo con la voluntad de los electores. Normalmente, pero, los electores sólo ordenan unos pocos candidatos (empíricamente se sabe que con 10 candidatos ya es suficiente por garantizar el proceso).

El procedimiento de redistribución de excedente es el siguiente: Para cada candidato C con excedente, se divide el excedente entre sus votantes NV, obteniendo el excedente proporcional (un valor decimal). Es decir, cada votante del candidato C “gasta” una parte para el candidato C y, el resto, la dedica al siguiente candidato que haya elegido. Cada vez que se redistribuye el excedente de un candidato, este pasa al grupo de candidatos seleccionados y ya no intervendrá más en el proceso de redistribución.

Este procedimiento se va repitiendo hasta que desaparece el excedente de todos los candidatos. Por ejemplo, si suponemos que tenemos 100 votos, 4 escaños, 5 candidatos (C1, C2, C3, C4 y C5) y que a las votaciones C1 obtiene 50 votos, C2 obtiene 35 y C3 obtiene 15 los otras dos no obtienen ningún voto. Con estos datos tenemos:

- Cuota: 25 (100 votos /4 escaños)

- C1 tiene un excedente de cuota de 25 (50-25) y un excedente proporcional de (25/50) = 0,5. El 50% del voto de los electores del candidato C1 se acumula a su segunda opción.

- C2 tiene un excedente de cuota de 10 (35-25) y un excedente proporcional de (10/35) = 0,28. El 28% del voto de los electores del candidato C2 se acumula a su segunda opción.

- C3, C4 y C5 no tienen excedente de cuota.

El procedimiento de redistribución se haría de la siguiente forma:

- Elegir uno de los candidatos con excedente (suponemos Cx).
- Recorrer todos los votos del candidato y, para cada uno de los votos:
    - Elegir el primer candidato del voto (recordad que en el momento de la votación se pide a los electores que ordenen de mayor a menor loas candidatos a los que dan su voto) que no haya entrado todavía en el procedimiento de redistribución (suponemos que es Cy)
    - Añadir el voto a Cy (actualizando la fracción del voto con el excedente proporcional de Cx)
    - Actualizar el número de votos de Cy (suma de las fracciones de todos los votos de Cy)
    - Comprobar si Cy supera la cuota
- Actualizar el número de votos de Cx (ahora serán igual a la cuota porqué hemos redistribuido su excedente)
- Marcar que el candidato Cx ya ha entrado en el procedimiento de redistribución.

Una vez redistribuidos todos los excedentes, empezaría una segunda etapa de eliminación de candidatos hasta quedar E (el número de escaños), pero por el momento no implementaremos esta segunda etapa.

Para la resolución del ejercicio hacemos las siguientes consideraciones:

- El número de partidos P es pequeño y conocido.
- El número de candidatos C es relativamente grande y conocido (del orden de miles).
- El número de votantes V es desconocido (depende del grado de participación) y puede ser muy grande (del orden de millones).
- El número de escaños E es pequeño y conocido.

Se quiere diseñar un TAD Votaciones con las operaciones siguientes:

1) crear(). Crear la estructura, inicialmente vacía.
2) anadirPartido(Partido). Añade un partido a la estructura.
3) anadirCandidato(Candidato, Partido). Añade un candidato asociado a un partido.
4) anadirVotante(Votante, cola<Candidato>). Registra un votante con su lista de preferencias y actualiza el total de votos del candidato de primera opción. Si el votante ya había votado, da un mensaje d'error. Se supone que la lista de candidatos es correcto.
5) finalizarVotaciones(): Cierra las votaciones y prepara la estructura para el proceso de redistribución.

6) quedanExcedentes(). Devuelve un booleano indicando si quedan candidatos con excedente >0 .
7) redistribuirExcedente(): Redistribuye el excedente del candidato con más excedente.
8) listadoCandidatosSeleccionados(). Devuelve un iterador para recorrer los candidatos seleccionados. No importa el orden. Inicialmente esta lista está vacía y se va ampliando a cada paso del proceso de redistribución.
9) partidoMasVotado(): Partido devuelve el partido más votado.

Requisitos de eficiencia:

- Las operaciones 1, 2, 3 i 5 no deben ser especialmente eficientes.
- Las operaciones 4, 6 i 7 deben ser el máximo de eficientes posible.

Os pedimos lo siguiente:

Apartado 4.1) Realizad un dibujo de la estructura de datos resultante, que deje claras las partes que la componen mediante las representaciones gráficas vistas en la asignatura. Podéis poner una breve descripción (dos o tres líneas) de cada componente de la estructura (por ejemplo, la estrategia de representación: secuencial, encadenada por vectores, encadenada indirecta, etc.). Debe quedar claro qué es la información contenida en cada uno de estos componentes.

Apartado 4.2) Estudiad la eficiencia de las operaciones anadirVotante() y redistribuirExcedente(). Concretamente, para cada operación, debéis describir brevemente su comportamiento indicando los pasos que la componen (con frases como por ejemplo: “insertar en el árbol AVL / borrar de la tabla de dispersión / consulta del piló / ordenar el vector...”), comentando la eficiencia asintótica de cada paso y dando la eficiencia total de la operación.