Codi: 05.001

Assignatura: Estructura de la Informació

Curs: 2010/2011 – primer semestre

PAC 2

Exercici 1 [2 punts]

Apartat 1.1) Descriviu com funcionaria un algorisme que us permeti fusionar dos arbres AVL de mida N i M respectivament. Quin cost tindria aquesta operació de fusió?

L'algorisme consisteix en un bucle que, mitjançant un iterador preordre, per exemple, pugui recórrer l'arbre de menor mida i va inserint els elements en l'altre arbre. Si l'arbre més petit té N elements, el bucle donarà N voltes. A cada volta executa unes operacions constants (llegir un element del primer AVL i assignar-lo al segon) i altres de cost logarítmic (buscar la posició que ha d'ocupar l'element i reestructurar l'arbre resultant). En el pitjor dels casos, el cost asimptòtic serà: **O(N · log (N+M))**

Apartat 1.2) Implementeu amb Java una extensió de la classe ArbreAVL de la biblioteca de classes per dotar-la d'un mètode que fusioni l'arbre actual amb un arbre que rebeu com a paràmetre. El mètode que cal que implementeu ha de tenir la següent signatura:

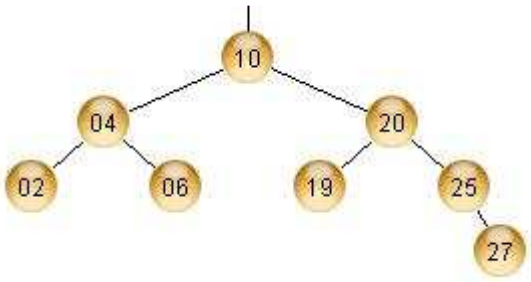
public void fusionar(ArbreAVL arbre)

```
public void fusionar(ArbreAVL<E> arbre)
{
        Recorregut<E> r = arbre.recorregutPreordre();
        while(r.hiHaSeguent())
        {
                this.afegir(r.seguent().getElem());
        }
}
```

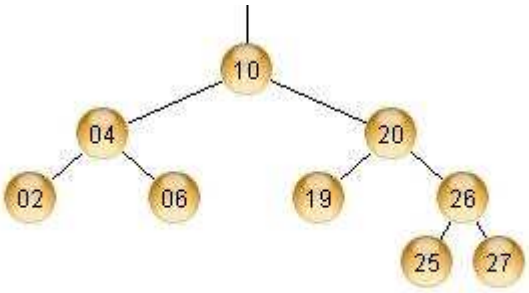
Apartat 1.3) Apliqueu l'algorisme de fusió per fusionar els elements de l'arbre X amb l'arbre Y i mostreu l'arbre AVL resultant. Si la operació de fusió provoca desequilibris, comenteu quin desequilibri es produeix i amb quina rotació es soluciona.

Arbre X

ArbreY



Aplicant l'algorisme de fusió sols ens cal inserir l'únic node de l'arbre X a l'arbre Y produeix un desequilibri DE obtenint el següent AVL resultat:



Exercici 2 [2 punts]

Apartat 2.1) Implementeu amb Java un TAD que us permeti gestionar els empleats d'una empresa multinacional amb les següents premisses:

- No podeu utilitzar delegació.
- Utilitzeu les classes de la JCF.
- El nombre d'empleats és molt gran i conegut.
- Cal un accés ràpid a les dades de l'empleat a partir del seu dni.
- Per cada empleat guardarem les següents dades:
  - Dni
  - Nom
  - Cognoms
  - Telèfon
- Cal que oferiu operacions per:
  - Donar d'alta un empleat
  - Consultar l'empleat per dni
  - Donar de baixa un empleat
  - Consultar si un dni correspon a un empleat
  - Llistar tots els empleats
  - Llistar tots els dni dels empleats
  - Consultar el nombre d'empleats
  - Consultar el nombre d'empleats amb un nom determinat
  - Afegir un conjunt d'empleats

Amb totes les dades que comenta l'enunciat sembla adient definir el TAD com una classe que estengui del TAD HashMap de la JCF ja que:

- Triem un TAD Taula perquè totes les operacions excepte la consulta del nombre d'empleats amb un nom determinat els les ofereix la inteface Map de la JCF
- Triem un HashMap (implementació de taula de dispersió) perquè ens demanen accés ràpid a les dades de l'empleat per clau i el nombre d'empleats es molt gran però conegut.

Definim primer la classe Employee:

```
public class Employee {
	private String dni;
	private  String name;
	private  String surname;
	private  String phone;

	public Employee(String dni, String name, String surname, String phone)
	{
		this.dni = dni;
		this.name = name;
		this.surname = surname;
		this.phone = phone;
	}
```

```
		/* Getters and setters */
		public String getDni() {
			return dni;
		}
		public String getName() {
			return name;
		}
		public String getSurname() {
			return surname;
		}
		public String getPhone() {
			return phone;
		}
		public void setDni(String dni) {
			this.dni = dni;
		}
		public void setName(String name) {
			this.name = name;
		}
		public void setPhone(String phone) {
			this.phone = phone;
		}
		public void setSurname(String surname) {
			this.surname = surname;
		}
	}
```

i el TAD:

```
	import java.util.HashMap;
	public class MyTAD<TKey extends String, TValue extends Employee> extends HashMap<TKey,
	TValue> {
		public int count(String name)
		{
			int ocurrences = 0;
			for(Employee employee : this.values())
			{
				if(employee.getName().equals(name))
				{
					ocurrences++;
				}
			}
			return ocurrences;
		}
}
```

Apartat 2.2) Implementeu amb Java un TAD que us permeti gestionar els noms dels empleats d'una empresa multinacional amb les següents premisses:

- No podeu utilitzar herència.
- Utilitzeu les classes de la biblioteca de classes de l'assignatura.
- El nombre d'empleats és molt gran però desconegut.
- Cal una consulta eficient.
- Cal que oferiu operacions per:
  - Donar d'alta un nom, si el nom ja existeix no fa res
  - Consultar si un nom existeix
  - Eliminar un nom
  - Llistar tots els noms
  - Consultar el nombre de noms que comencin per 'A'
  - Afegir un conjunt de noms

Amb totes les dades que comenta l'enunciat sembla adient definir el TAD com una classe que utilitzi les operacions del TAD ConjuntAVLImpl de la biblioteca de classes de l'assignatura ja que:

- Triem un TAD Conjunt perquè totes les operacions demanades excepte les dues darreres les ofereixen els conjunts i les claus i els valors que cal emmagatzemar son els mateixos.
- Escollim un implementació basada en AVL perquè cal una consulta eficient però com que no sabem el nombre d'elements no es recomanable utilitzar una implementació basada en una Taula

```
import uoc.ei.tads.*;

public class MyTADConjunt<T extends String> {

	private ConjuntAVLImpl<T> cjt = new ConjuntAVLImpl<T>();

	public boolean estaBuit() {
		return cjt.estaBuit();
	}

	public void afegir(T elem) {
		cjt.afegir(elem);
	}

	public boolean hiEs(T elem) {
		return cjt.hiEs(elem);
	}

	public String esborrar(T elem) {
		return cjt.esborrar(elem);
	}

	public Iterador<T> elements() {
		return cjt.elements();
	}

	public int countFirstLetter(char letter)
	{
		int ocurrences = 0;
		for(Iterador<T> it = cjt.elements(); it.hiHaSeguent();)
		{
			if(it.seguent().charAt(0) == letter)
			{
				ocurrences++;
			}
		}
		return ocurrences;
	}

	public void afegirRang(Iterador<T> it)
	{
		while(it.hiHaSeguent())
		{
			this.afegir(it.seguent());
		}
	}

}
```

Exercici 3 [2,5 punts]

Apartat 3.1) Suposem que hem dissenyat una funció de dispersió que té com a claus els noms i cognoms dels estudiants matriculats a la UOC; suposem també que entre nom i cognoms es poden ocupar 60 caràcters. Es defineix la funció de dispersió de la següent manera:

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

Comenta els possibles inconvenients de la funció de dispersió escollida i proposa una alternativa que els solucioni.

Els inconvenients que es detecten a la funció son els següents:

- Sempre dona parell i així estem desaprofitant la meitat de la taula.
- En qualsevol ordinador produirà sobreeiximent.
- El valor de r es 256, la UOC tindrà força més alumnes que 256.
- Per tot l'anterior, h depèn de l'aspecte de la cadena d'entrada.
- Els valors generats no seran equiprobables, ja que la funció ascii te en conte els 127 caràcters i no tots apareixen als nom i cognoms

Hi ha moltes solucions possibles, una funció de dispersió que soluciona els problemes esmentats podria ser la fórmula que teniu als apunts del mòdul agafant b=26 i r el nombre primer més proper al valor suposat d'alumnes de la UOC:

$$h(c_1...c_{60}) = (\sum k: 1 \leq k \leq 60 : conv(c_k) * b^{k-1}) \bmod r$$

Apartat 3.2) És cert que un TAD taula amb una funció de dispersió mal dissenyada pot degradar el rendiment de l'operació de cerca fins a fer-la lineal respecte del nombre d'elements de la taula? Posa un exemple de funció de dispersió amb aquest comportament.

Si que és cert, la funció de dispersió utilitzada en un TAD taula és clau per aconseguir un bon rendiment per les operacions de cerca, una mala funció de dispersió pot degradar completament el rendiment de les cerques.

Anant a l'extrem una funció de dispersió que col·loqués les entrades sempre a una posició fixa de la taula produiria un rendiment lineal a les operacions de cerca perquè s'hauria de recórrer tota la llista de sinònims de l'element i aquesta contindria tot els elements de la taula.

Apartat 3.3) Suposeu que teniu un TAD taula i que no aconseguiu una funció de dispersió que doni una distribució equiprobable dels elements, com podríeu millorar l'eficiència de la cerca sense variar la funció de dispersió?

Una possible manera de millorar l'eficiència de les operacions de cerca seria utilitzar un arbre AVL per les llistes de sinònims, amb aquesta solució passaríeu d'una eficiència lineal a logarítmica respecte al nombre de sinònims a les cerques que provoquin col·lisió. Tingueu en compte però que amb aquesta implementació també estaríeu penalitzant les insercions a la taula.

Exercici 4 [3,5 punts]

Hem d'implementar un prototip per realitzar votacions per Internet experimentant un sistema electoral amb llistes obertes. En aquest sistema els candidats pertanyen a algun partit però els electors no voten a un partit, sinó a un candidat concret.

Un cop realitzades les votacions es fa la divisió entera entre el nombre total de votants i els E escons, obtenint així la *quota* necessària per obtenir un escó. Tots els candidats que estan per sobre de la *quota* obtenen escó, i els que estan per sota queden provisionalment fora.

Si el procés s'acaba aquí es molt provable que no s'ocupin tots els E escons, ja que segurament alguns dels candidats poden haver superat la *quota* i un candidat només pot ocupar un escó, encara que l'hagin votat suficientment per ocupar-ne dos o més. Per tal d'ocupar tots els E escons, cal redistribuir aquest excedent de *quota*, però ¿a qui? Nosaltres ens interessem per la següent solució:

En el moment de votar, es demana als electors que ordenin els candidats de més a menys prioritari. D'aquesta manera, l'excedent de vots d'un candidat X es pot redistribuir d'acord amb la voluntat dels electors. Normalment, però, els electors només ordenen uns pocs candidats (empíricament se sap que amb 10 candidats ja és suficient per garantir el procés).

El procediment de redistribució d'excedent és el següent: Per cada candidat C amb excedent, es divideix l'excedent entre els seus votants NV, obtenint l'excedent proporcional (un valor decimal). És a dir, cada votant del candidat C "gasta" una part pel candidat C i, la resta, la dedica al següent candidat que hagi triat. Cada cop que es redistribueix l'excedent d'un candidat, aquest passa al grup de candidats seleccionats i ja no intervindrà més en el procés de redistribució. Aquest procediment es va repetint fins que desapareix l'excedent de tots els candidats.

Per exemple, si suposem que tenim 100 vots, 4 escons, 5 candidats (C1, C2, C3, C4 i C5) i que a les votacions C1 obté 50 vots, C2 n'obté 35 i C3 n'obté 15 els altres dos no obtenen cap vot. Amb aquestes dades tenim:

- Quota: 25 (100 vots / 4 escons)
- C1 té un excedent de quota de 25 (50-25) i un excedent proporcional de (25/50) = 0,5. El 50% del vot dels electors del candidat C1 s'acumula a la seva segona opció.
- C2 té un excedent de quota de 10 (35-25) i un excedent proporcional de (10/35) = 0,28. El 28% del vot dels electors del candidat C2 s'acumula a la seva segona opció.
- C3, C4 i C5 no tenen excedent de quota.

El procediment de redistribució d'excedents es faria de la manera següent:

- Triar un dels candidats amb excedent (suposem Cx).
- Recórrer tots els vots del candidat i, per cada un dels vots
    - Triar el primer candidat del vot (recordeu que en el moment de votar es demana als electors que ordenin de major a menor els candidats als que donen el seu vot) que no hagi entrar encara en el procediment de redistribució (suposem que és Cy)
    - Afegir el vot a Cy (actualitzant la fracció del vot amb l'excedent proporcional de Cx)

    - Actualitzar el nombre de vots de Cy (suma de les fraccions de tots els vots de Cy)
    - Comprovar si Cy supera la quota
- Actualitzar el nombre de vots de Cx (ara seran igual a la quota perquè hem redistribuït el seu excedent)
- Marcar que el candidat Cx ja ha entrat al procediment de redistribució.

Un cop redistribuïts tots els excedents, començaria una segona etapa d'eliminació de candidats fins quedar-ne E (el nombre d'escons), però de moment no implementarem aquesta segona etapa.

Per la resolució de l'exercici fem les següents consideracions:

- El nombre de partits P és petit i conegut.
- El nombre de candidats C és relativament gran i conegut (de l'ordre de milers).
- El nombre de votants V és desconegut (depèn del grau de participació) i pot ser molt gran (de l'ordre de milions).
- El nombre d'escons E és petit i conegut.

Es vol dissenyar un TAD Votacions amb les operacions següents:

1) crear(). Crear l'estructura, inicialment buida.

2) afegirPartit(Partit). Afegeix un partit a l'estructura.

3) afegirCandidat(Candidat, Partit). Afegeix un candidat associat a un partit.

4) afegirVotant(Votant, cua <Candidat>). Registra un votant amb la seva llista de preferències i actualitza el total de vots del candidat en primera opció. Si el votant ja havia votat, dóna un missatge d'error. Se suposa que la llista de candidats és correcte.

5) finalitzarVotacions(). Tanca les votacions i prepara l'estructura pel procés de redistribució.

6) quedenExcedents(). Retorna un booleà indicant si queden candidats amb excedent > 0.

7) redistribuirExcedent(). Redistribueix l'excedent del candidat amb més excedent.

8) llistatCandidatsSeleccionats(). Retorna un iterador per recórrer els candidats seleccionats. No importa l'ordre. Inicialment aquesta llista és buida i es va ampliant a cada pas del procés de redistribució.

9) partitMésVotat(). Retorna el partit més votat.

Requisits d'eficiència:

- Les operacions 1, 2, 3 i 5 no cal que siguin especialment eficients.
- Les operacions 4, 6 i 7 han de ser el màxim d'eficients possibles.

Us demanem el següent:

Apartat 4.1) Feu un dibuix de l'estructura de dades resultant, que deixi clares les parts que la componen mitjançant les representacions gràfiques vistes a l'assignatura. Podeu posar una breu descripció (dues o tres línies) de cada component de l'estructura (per exemple, l'estratègia de representació: seqüencial, encadenada per vectors, encadenada indirecte, etc.). Ha de quedar clar quina és la informació continguda a cadascun d'aquests components.

Apartat 4.2) Estudieu l'eficiència de les operacions afegirVotant() i redistribuirExcedent(). Concretament, per a cada operació heu de descriure breument el seu comportament indicant els passos que la componen (amb frases com ara: "inserir en l'arbre AVL / esborrar de la taula de dispersió / consulta del piló / ordenar el vector..."), dient l'eficiència asimptòtica de cada pas i donant l'eficiència total de l'operació.

Utilitzarem una llista afitada per guardar els P partits que es presenten a les eleccions. Sabem que P és petit i conegut i, per tant, una llista afitada s'ajusta a les necessitats, ja que el temps de consulta d'un partit serà menyspreable (degut a la mida petita de P).

Podríem utilitzar una llista encadenada per guardar els candidats d'un partit, però a l'enunciat no hi ha cap mètode que requereixi aquesta informació i, per tant, descartem aquesta estructura. En canvi, per cada candidat si que guardarem una referència al partit al qual pertany (per poder comptabilitzar els vots de cada partit).

El nombre de candidats és elevat i conegut. Això ens permet utilitzar una taula de dispersió, ja que no necessitem cap recorregut ordenat de candidats. A més, el baix cost de les consultes ens permetran optimitzar el mètode afegir votant.

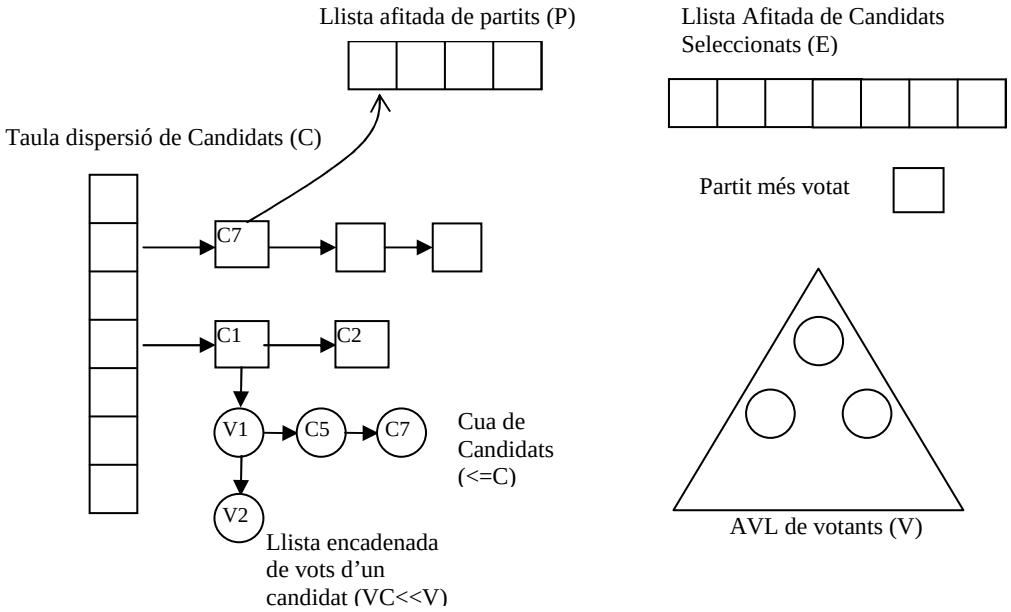
Per afegir una votació caldrà verificar que el votant no ho hagi fet amb anterioritat. A tal efecte utilitzarem un AVL per localitzar els votants ràpidament. No podem utilitzar una taula de dispersió ja que el nombre de votants és desconegut. Un cop verificat ja podrem afegir la votació al candidat. Per cada candidat X utilitzarem una llista encadenada amb totes les votacions. Utilitzem una llista encadenada ja que, no cal dir, desconeixem el nombre de votacions que tindrà cada candidat, i no necessitem cap operació especial. Aquesta llista la utilitzarem per redistribuir els excedents dels candidats. Sense aquesta llista hauríem de fer un recorregut de totes les votacions per localitzar les del candidat amb excedent, cosa que dispararia el mètode 7.

Cada votació consistirà d'una cua de candidats, d'acord amb l'elecció del votant. Cada cop que assignem una part del vot a algun candidat, aquest desapareix de la cua. Això ens permet alliberar espai que no necessitem per res més. A més, hem de saber quina fracció del vot hem assignat al candidat actual. Inicialment assignarem tot el vot al primer candidat, però a cada redistribució aquesta fracció s'anirà fent petita.

Durant el procés de redistribució d'excedents cal localitzar ràpidament els candidats amb excedent. Proposem la utilització d'una estructura afitada al nombre d'escons E per guardar els Candidats que han superat la quota. Aquesta estructura s'inicialitza durant el mètode finalitzarVotacions fent un recorregut per tots els candidats i afegint-hi els que hagin superat

la quota. A cada pas del procés de redistribució s'escull un dels candidats i es marca per evitar que se li apliqui una segona redistribució. Cada cop que un candidat supera la quota a resultes d'una redistribució, aquest candidat s'afegeix a l'estructura. Com que es tracta d'una estructura de mida petita E, no perjudicarem l'eficiència ja que les operacions tindran un cost aproximat O(1).

El següent dibuix mostra l'estructura que hem descrit:



Estudi d'eficiència de les operacions demanades:

4) afegirVotant(Votant, Cua<Candidat>)

- Buscar Votant a AVL de votants --> O(log V)
- Buscar el primer candidat a la taula de dispersió de candidats --> O(1)
- Incrementar (+1) el comptador de vots al primer candidat -->O(1)
- Crear un nou Vot amb la cua de candidats (eliminant el primer) i inicialitzant la fracció a 1. --> O(1)
- Afegir un vot a la llista encadenada de vots del candidat --> O(1)
- Incrementar el comptador de vots del partir del primer candidat, i actualitzar el partit més votat (si s'escau) -->O(1)
- **Total: O(logV) -> O(logV)**

7) redistribuirExcedent().

- Accedir a un candidat X amb excedent -> O(E)≈O(1)
- Calcular l'excedent proporcional -> O(1)

- Fer un recorregut per cada votació del candidat -> O(VC)
    - Localitzar el següent candidat Y comprovant que no hagi passat ja pel procés de redistribució--> O(E) ≈O(1)
    - Treure el vot de la llista de X i afegir-lo a la llista de vots de Y--> O(1)
    - Eliminar el candidat Y de la cua de preferències del vot -->O(1)
    - Actualitzar la fracció del vot utilitzant l'excedent proporcional d'X-->O(1)
    - Incrementar els vots de Y d'acord amb la nova fracció --> O(1)
    - Si el candidat ha superat la quota gràcies a aquesta fracció, l'afegim a la llista de candidats seleccionats --> O(1)
- Actualitzar els vots del candidat X (passen a ser el valor de la quota) --> O(1)
- Marcar el candidat X per no entrar més en el procés de redistribució --> O(1)

Total: O(VC+E) ≈O(VC)