Codi: 05.001

Assignatura: Estructura de la Informació

Curs: 2010/2011 – segon semestre

PAC 1

Exercici 1 [2 punts]

Una cooperativa de taxis decideix fer una aplicació per gestionar l'assignació dels serveis als seus taxistes. Es desitja fer les assignacions en el mateix ordre que s'han generat. Es a dir, si en el moment que es demanen serveis no hi ha cap taxi disponible, en el moment que un taxista quedi lliure agafarà el primer servei demanat. El mateix criteri es desitja aplicar als taxistes. En cas que hi hagi taxistes lliures, en el moment que es demani un servei, s'assignarà al taxista que faci més temps que hagi lliurat.

El TAD TaxiManagement haurà de tenir la següent funcionalitat:

- Obtenir el taxi lliure amb major temps d'espera
- Donar d'alta un servei demanat per una client
- Assignar el servei més antic al taxi lliure amb més temps d'espera.
- Donar per conclòs un serveis.

Contesta i raona:

Apartat 1.1) Dóna la signatura del TAD TaxiManagement. Es a dir, indica el nom que donaries a les operacions encarregades de cada funcionalitat requerida, indica també, quins caldria que fossin els paràmetres d'entrada i quina la sortida en cas que es necessités. Suposa que existeixen els TADs Taxi per gestionar les dades d'un taxi i Service per gestionar les dades d'un servei de taxi.

```
TAD TaxiManagement signatura{
        Taxi getTaxi();
        void addService(Servei service);
        void assignServiceToTaxi();
        void endService(Service);
}
```

La operació *assignServiceToTaxi()* no rep paràmetre per assegurar la coherència de l'enunciat. Si passéssim el taxi o el servei, podria no coincidir amb el taxi ni el servei que més temps d'espera porten. En aquesta signatura també suposem que Service coneix el taxi que se li ha assignat per tal d'assegurar que el taxi que caldrà deixar lliure correspongui al del servei realitzat.

Apartat 1.2) El TAD específic que se us demana de dissenyar estarà compost per TADS genèrics de la biblioteca de l'assignatura. Indica quins i quants caldran per poder dur a terme la gestió.

Necessitarem dues Cues, una de taxis i una altre de serveis que caldrà implementar com:

Cua<Taxi> taxiQueue;
Cua<Service> serviceQueue;

Per mantenir els serveis en curs podríem usar un llista:

Llista<Service> servicesInProgress;

Apartat 1.3) Indica quines operacions dels TADs de biblioteca de l'assignatura que heu escollit caldrà cridar per resoldre les operacions del TAD TaxiManagement. Es a dir, suposant que se us demanés la funcionalitat d'afegir el numero de telèfon des d'on s'ha trucat i el TAD escollit fos una llista, una possible resposta seria: ***afegirAlfinal(telefon).*** Si la funcionalitat fos eliminar el número de telefon, una possible resposta seria: **obtenir un recorregut amb la operació *posicions()*. Iterar per les diferents posicions fins a trobar la posició on es troba el telefon cercat. Amb la posició trobada cridar *esborrar(posicio)*.**

Taxi getTaxi():

- consultar el primer elements de la cua de taxis.

void addService(Servei service):

- encuar el servei a la cua de serveis.

void assignServiceToTaxi():

- Desencuar el primer taxis de la cua de taxis
- Desencuar el primer servei de la cua de serveis
- Associar el taxis i el servei extrets.
- Afegir el servei associat a la llista de serveis en curs.

void endService(Service):

- Eliminar el servei de la llista de serveis en curs
- Obtenir el taxi associat al servei,
- Encuar el taxi a la cua de taxis
- Eliminar el servei del sistema

Apartat 1.4) Fes l'especificació contractual del TAD. Utilitza de model l'especificació de l'apartat 1.2.3 dels materials. Es valorarà especialment la concisió (absència d'elements redundants o innecessaris), precisió (definició correcta del resultat de les operacions), completesa (consideració de tots els casos possibles en què es pot executar cada operació) i manca d'ambigüitats (coneixement exacte de com es comporta cada operació en tots els casos possibles) de la solució. És important respondre aquest apartat usant una descripció condicional i no procedimental. L'experiència ens demostra que no sempre resulta fàcil distingir entre les dues descripcions, és per això que fem especial èmfasi insistint que poseu

molta atenció a les vostres definicions. A títol d'exemple indicarem que la descripció condicional (la correcta a utilitzar en el contracte) d'omplir un got buit amb aigua seria:

> @ pre el got es troba buit.
> @ post el got és ple d'aigua.

En canvi una descripció procedimental (incorrecta per utilitzar en el contracte) tindria una forma semblant a:

> @ pre el got hauria de trobar-se buit, si no es trobés buit s'hauria de buidar.
> @ post s'acosta el got a l'aixeta i s'hi tira aigua fins que estigui ple.

Cal també tenir en compte que un contracte hauria de disposar d'invariant sempre que aquesta fos necessària per descriure el TAD.

> @pre true
> @post $return el taxi que fa més temps que es troba esperant un servei.
> **Taxi getTaxi():**

> @pre true
> @post service s'ha afegit al TAD com a servei en espera
> **void addService(Servei service):**

> @pre true
> @post El taxi i el servei amb més temps d'espera es troben associats i el servei en curs.
> **void assignServiceToTaxi():**

> @pre el servei passat per paràmetre es troba en curs
> @post El taxi associat a service es troba en espera i service no és un servei del sistema.
> **void endService(Service service):**

Exercici 2 [2 punts]

Apartat 2.1) Escriu un algoritme que permeti comptabilitzar quantes vegades es repeteix un vector de caràcters en el contingut d'un segon vector. Exemple: Suposeu el vector v1={u,n, ,v,e,c,t,o,r, ,d,e, ,c,a,r,a,c,t,e,r,s, ,d,e, ,q,u,a,l,s,e,v,o,l, ,t,i,p,u,s} i el vector v2={d,e}, Si es demana quantes vegades es repeteix v2 a v1 caldrà respondre 2 vegades.

```
1.  public int countSubstrings(char[] v1, char[] v2){
2.      int pos1;
3.      int res=0;
4.      int pos2 =0;
5.      for(pos1=0; pos1<v1.length; pos1++){
6.          pos2=0;
7.          while(pos1+pos2<v1.length && pos2<v2.length && v1[pos1+pos2]==v2[pos2]){
8.              pos2++;
9.          }
10.         if(pos2==v2.length){
11.             res++;
12.         }
13.     }
14.     return res;
15.  }
```

Apartat 2.2) Fes un estudi del cost de l'algoritme.

	codi	cost		
1.	public int countSubstrings(...			
2.	int pos1;			
3.	int res=0;	1		
4.	int pos2 =0;	1		
5.	for(pos1=0; pos1<v1.length; ...	1; 1; 1		
6.	pos2=0;	1		
7.	while(pos1+pos2<v1.length...	7		
8.	pos2++;	1		
9.	}		M (7+1) +7	
10.	if(pos2==v2.length){	1		
11.	res++;	1		
12.	}		2	
13.	}			1+N (1+1+1+ M(8)+7+2)+1
14.	return res;	1		
15.	}			1+1+1 + N (1+1+1+ M(8)+7+2)+1+1

T(N) = 5+N(12+8M)=4+12N+8NM. Per tant, direm que l'algoritme té una complexitat asimptòtica de O(NM).

Exercici 3 [2 punts]

Indica quin arbre té un recorregut en preordre: {2,5,1,7,3,9,4,12} i en inordre: {5,7,1,3,2,9,12,4}. Raona la resposta explicant com arribes a la conclusió.

1) l'arrel és 2 perquè és el primer element del preodre

preordre: **2**,5,1,7,3,9,4,12

2) 5,7,1,3 es troben a l'esquerra de l'arrel i 9,12,4 a la dreta, d'acord amb l'inordre

inordre: 5,7,1,3,**2**,9,12,4

3) l'arrel del subarbre esquerre és el 5

preordre: 2,**5**,1,7,3,9,4,12

4) El 5 no té fill esquerre però si fill dret

inordre: **5**,7,1,3,2,9,12,4

5) L'arbre dret del node de valor 5 té per arrel l'1 amb el 7 com a fill esquerra i el 3 com a dret.

Preordre: 2,5,**1**,7,3,9,4,12

inordre: 5,7,**1**,3,2,9,12,4

6) el 9 és l'arrel des subarbre dret

preordre: 2,5,1,7,3,**9**,4,12

7) el 9 no té fill esquerra però si dret

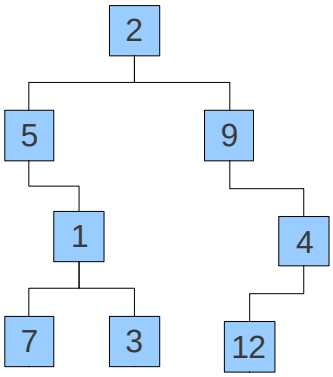
inordre: 5,7,1,3,2,**9**,12,4

8) El 12 és fill esquerra del 4

preordre: 2,5,1,7,3,9,**4**,12

inordre: 5,7,1,3,2,9,12,**4**

L'arbre quedarà com:



Exercici 4 [2 punts]

Apartat 4.1) Expliqueu amb les vostre paraules les diferències entre els TADs *Cua* i *Llista.* En quins casos faries servir una cua i en quins una llista?

La Cua és una TAD molt més dinàmic que la llista. El primer es adequat per algoritmes que necessitin tractar les dades del TAD només un cop ja que en desencuar extrèiem també la dada del TAD. La llista en cavi manté un cert ordre i permet recorre els seus elements múltiples vegades sense canviar la composició. Per altra banda, l'ordre dels elements de la cua serà sempre el mateix que l'ordre d'arribada, en canvi la llista permet inserir elements en qualsevol posició. En certa forma podem dir que la cua és una llista específica per mantenir els seus elements en l'ordre amb el que arribin i el tractament d'aquests implica la seva supressió del contenidor.

Apartat 4.2) Indica si l'afirmació: *"El cost asimptòtic descriu a la perfecció el rendiment d'una operació"* és certa o falsa raonant la teva resposta.

El cost asimptòtic és una aproximació del comportament del cost d'un algoritme a mida que aquest hagi de tractar un nombre creixent de dades. Per tant la resposta seria falsa.

Apartat 4.3) Es pot cercar un element d'una llista ordenada amb repeticions amb un cost logarítmic? I si la llista no permet repeticions?

El cost logarítmic en cercar dins una llista lineal, es podria podria aconseguir fent cerca dicotòmica. Malauradament, aquest algorisme es posiciona sobre qualsevol de les dades repetides de valor coincident a la cerca. Això implica que no es possible distingir entre valors iguals en temps logarítmic. Per tant direm que no és possible cercar logarítmicament en llistes ordenades que admetin valors repetits.

Si que seria possible en llistes ordenades sense repeticions, sempre i quan la implementació de la llista es basés en un vector. Les llistes encadenades no suporten la cerca dicotòmica perquè no tenim manera de seleccionar el valor que es troba a la posició intermèdia entre dues posicions donades. Les implementacions vectorials, en canvi, en treballar amb posicions numèriques si que permeten calcular posicions intermèdies.

Apartat 4.4) Expliqueu breument com es relacionen els conceptes de "recursivitat" i "recorreguts d'un arbre".

Diem que els fills de l'arrel d'un arbre tenen també totes las característiques d'arbre i per tant suportaran las mateixes operacions que l'arbre principal. Això té unes implicacions molt interessants ja que els fills es consideraran també arrels del subarbre corresponent i els fills dels fills (en cas que existeixin) seguiran sent arbre sobre els quals aplicar les mateixes operacions que a qualsevol arbre.

El recorregut d'un arbre és una operació que es pot resoldre recursivament, aplicant la característica explicada al paràgraf anterior tenint en compte que qualsevol recorregut s'obté combinant en diferent ordre el recorregut parcial del fill esquerre, el recorregut parcial del fill dret i el node arrel.

Exercici 5 [2 punts]

Apartat 5.1) Es demana que implementeu el TAD PilaUnica, el qual estén del TAD Pila de la biblioteca però assegura de complir sempre la invariant:

@invariant Tots els seus elements són diferents. Formalment:

$all(e1:pila, !$exists(e2:pila, e1=e2));

NOTA la implementació heu de fer-la usant herència. És a dir només cal que implementeu les operacions noves o les ja existents que caldria canviar.

```
public class PilaUnica<E> extends PilaEncadenadaImpl<E> {

   public void empilar(E elem) {
      boolean found = false;
      Iterador<E> it = this.elements();
      while(it.hiHaSeguent() && !found){
         found = it.seguent().equals(elem);
      }
      if(!it.hiHaSeguent() && !found){
         super.empilar(elem);
      }
   }
}
```

Apartat 5.2) Quin cost tindria l'operació d'empilar un element?

Tindria una cost lineal perquè cal recorre tot el contenidor per assegurar que l'element que s'empila sigui únic.

Apartat 5.3) Quina estructura usaries per millorar-ne el rendiment?

Es podria usar un vector ordenat que permetés cercar en temps logarítmic si l'element a empilar es troba ja dins el contenidor o no.