TecnoCampus Escola Superior Politècnica — Centre adscrit a la Universitat Pompeu Fabra Barcelona

Programació Avançada (UPF)
Programació 3 (UPC)

DATA:14 de Desembre del 2015
DURACIÓ:3h

Exercici 1. Tècnica del divideix i venç (1.5 punts)

Una empresa de fabricació de vestits disposa d'una barra on es troben penjats N vestits corresponents al mostrari d'aquesta propera temporada Primavera Estiu 2016. Se us demana d'escriure un procediment que usant **la tècnica del divideix i venç** trobi el model que té un cost de confecció més elevat i el que té el cost de confecció més baix. Els vestits estan tots en una taula, sense cap criteri d'ordenació, i cada objecte Vestit disposa d'un atribut amb el seu cost de confecció.

```
public class Vestit {
    private int identificacio;
    private String model;
    private float cost;

    public Vestit(int identificacio, String model) {
        this.identificacio = identificacio;
        this.model = model;
        this.cost = -1;
    }

    public float getCost() {
        return cost;
    }

    public String getModel() {
        return model;
    }

    public void setCost(float cost) {
        this.cost = cost;
    }
}
```

Prototipus del procediment a implementar amb Divideix i Venç:

```
public static String[] cercarMesMenys( Vestit vestits[]){
// tots els vestits del mostrari estan al vector, aquest està completament ple i
// tots els vestits tenen un cost superior a 0.
// a la posició d'índex 0 de l'objecte a retornar cal emmagatzemar el model del
// vestit més car i a la posició d'índex 1 el corresponent al model de vestit
//més barat
```

Exercici 2. Tècnica del backtracking (8.5 punts)

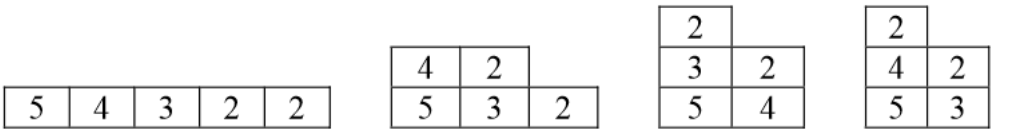
S'ha d'aplicar la tècnica del backtracking per resoldre el següent problema d'optimització:

S'han d'apilar N caixes sobre muntacàrregues. Cadascuna amb el seu propi pes i amb la seva etiqueta de fràgil o no segons procedeixi. Es tenen les següents restriccions:

1.- un muntacàrregues només pot suportar una única columna de caixes apilades.
2.- una caixa suporta tant pes a sobre ella com el seu propi pes.

Es poden usar un màxim de N muntacàrregues, i en aquest cas la solució seria posar una única caixa sobre cadascun d'ells, seria la pitjor solució ja que el lloguer d'aquesta màquina té un cost molt elevat i el que es vol és **minimitzar el número de muntacàrregues a llogar amb l'objetiu de minimitzar costos**.

Exemple:
Amb 5 caixes de pesos {5,4,3,2,2} podem usar fins a 5 muntacàrregues. Veiem algunes solucions:



5 muntacàrregues Solució 1 | 3 muntacàrregues Solució 2 | 2 muntacàrregues Solució 3 | 2 muntacàrregues NO Solució

Les propostes de solucions anteriors són totes correctes a excepció de la darrera, però l'òptima és la Solució 3, la que hauria de trobar el programa, la que usa el menor número possible de muntacàrregues i compleix les restriccions indicades.

S'ha d'escriure un programa per **trobar la millor distribució de caixes en muntacàrregues**, és a dir la que minimitzi el nombre de muntacàrregues usats. A igualtat de muntacàrregues és millor aquella solució que té més caixes fràgils a dalt de cada pila (com més muntacàrregues tinguin una caixa fràgil a dalt de tot millor).

```
public class Caixa {
    private int identificacio;
    private float pes;
    private boolean fragil;
    public Caixa(int identificacio, float pes, boolean fragil) {
        this.identificacio = identificacio;
        this.pes = pes;
        this.fragil = fragil;
    }
    public float getPes() {
        return pes;
    }
    public int getIdentificacio() {
        return identificacio;
    }
    public boolean getFragil() {
        return fragil;
    }
    public String toString() {
        return "la caixa  identificacada amb: " + this.identificacio + "de "
                + this.pes;
    }
}
```

```
public class Muntacarrega {
        private int identificacio;
        private Caixa caixes[];
        private int quantes;      // Dimensió real de l'atribut previ
        public Muntacarrega(int identificacio, int quantes){
            this.identificacio=identificacio;
            this.quantes=0;
            this.caixes=new Caixa[quantes];
        }
        public int getIdentificacio(){return identificacio;}
        public int getQuantes(){return quantes;}
        public Caixa getDarrera(){  //retorna null si no té caixa
            if (quantes>0)return caixes[quantes-1];
            return null;
        }
        public void addCaixa(Caixa c){
            this.caixes[this.quantes]=c;
            quantes++;
        }
        public void remCaixa(Caixa c){
            quantes--;
        }
}
public class Repartiment {
    private Caixa caixes[];
    // Exercici 1 - Afegeix atributs
    public static void carregaCaixes(Caixa[] caixes) {
        // omplena el paràmetre caixes amb tots els objectes caixa a carregar
    }
    public Repartiment(int quantes) {
        caixes = new Caixa[quantes];
        carregaCaixes(this.caixes);
        // Completar Exercici 2
    }
    public static void main(String args[]) {
        // demanem número de caixes a distribuir
        Repartiment m = new Repartiment(Keyboard.readInt());

        m.backMillorSolucio(/*?????*/);
        // Exercici 3
        //Visualització a pantalla de la millor distribució
    }
    public void backMillorSolucio(/*????*/) {
        // Exercici 4
    }
    private boolean acceptable(/*???*/) {
        // Exercici 4
    }
    private boolean millorSolucio(/*???*/) {
        // Exercici 4
    }
    public String toString() {
        // Exercici 5
    }
}
```

Contesta i implementa:

→ 1.- (**1 punt**) Contesteu, sempre justificant la resposta:

- a.- Per què l'esquema de backtracking és aplicable per a resoldre aquest enunciat.
- b.- Determina quines decisions ha de prendre el programa. Quin és el criteri per determinar si un conjunt de decisions és o no completable. Quin és el criteri per determinar si un conjunt de decisions és o no solució. En cas de ser solució, com es sap si és millor. Dibuixa l'espai de cerca del problema, és a dir, l'arbre que recorrerà la tècnica del backtracking, especificant quina serà l'alçada i l'amplada, indicant si són valors exactes o valors màxims. Amb el teu plantejament cal usar la tècnica del marcatge?. Hi haurà sempre solució?
- c.- Indica els atributs que s'han d'**afegir** a la classe *Repartiment* per a fer la implementació que has descrit en l'apartat anterior, **l'objectiu és que el mètode que implementa el backtracking tingui el menor número possible de paràmetres**. Per cada atribut indica el seu ús.

→2.- (**0.25 punts**) Completa la implementació del mètode constructor, cal crear i inicialitzar els atributs que has afegit a la classe.

→3.- (**0.25 punts**) Completa el mètode *main*. Ha de cridar al mètode que implementa el backtracking i ha de mostrar per pantalla la millor solució trobada fent una crida al mètode print.

→4.- (**6 punts**) Implementa el mètode **public void** backMillorSolucio(???). Has de determinar el(s) paràmetre(s) necessaris, minimitzant-los.

Aquest mètode ha de cridar a un mètode privat per fer la comprovació de si "és acceptable" la decisió en l'aplicació de l'esquema de backtracking (**1 punt**). I altre mètode per determinar quan es troba una solució si aquesta és o no millor a la millor trobada fins aleshores (**1 punt**).

→5.- (**0.5 punts**) Redefineix el mètode toString() per mostrar la millor solució trobada. Ha de mostrar a pantalla quants muntacàrregues s'han de llogar i per a cadascuna d'ells mostrar la pila de caixes qui s'han d'apilar. La visualització d'aquestes s'ha de fer de baix a dalt, és a dir primer mostrar la caixa que estarà a sota i en darrer lloc la de dalt de tot. Per a cada muntacàrregues mostra el pes que portarà.

Exemple de sortida:

```
La millor solució usa : 2 montacàrregues
La distribució és la següent:

El muntacarregues amb identificació 0 porta de baix a dalt:
============================================================
   la caixa  identificacada amb: 0 de 5.0 kilos
   la caixa  identificacada amb: 2 de 3.0 kilos
   la caixa  identificacada amb: 3 de 2.0 kilos
amb un pes total de: 10.0

El muntacarregues amb identificació 1 porta de baix a dalt:
============================================================
   la caixa  identificacada amb: 1 de 4.0 kilos
   la caixa  identificacada amb: 4 de 2.0 kilos
amb un pes total de: 6.0
```

→6.- (**0.5 punts**) Els problemes d'optimització poden esser resolts aplicant la tècnica voraç. Fes l'anàlisi d'aquest mateix exercici usant aquesta tècnica. Indica:

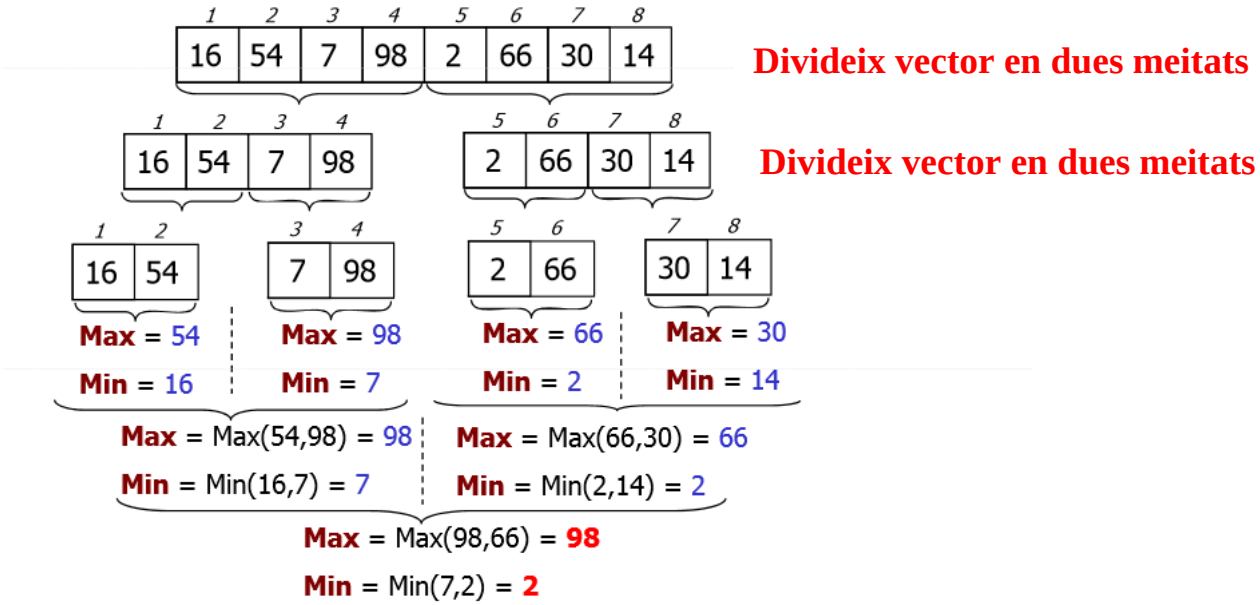
- Quins seran els candidats amb el teu plantajament voraç?
- Quina serà la teva funció de selecció?
- Amb aquesta funció trobaria la teva implementació la millor solució al problema? Justifica la resposta.

Nota: podeu afegir mètodes en qualsevol de les classes, que evidentment cal que implementeu. Només podeu afegir atributs a la classe Repartiment.

Solució

Exercici 1

suposem la següent taula de costos:




```
// Prototipus del mètode a implementar:
public static String[] cercarMesMenys(Vestit vestits[]) {
    Vestit resultat[]=cercarMesMenys(vestits, 0, vestits.length - 1);
    String[] r= {resultat[0].getModel(), resultat[1].getModel()};
    return r;
}
public static Vestit[] cercarMesMenys(Vestit vestits[], int ini, int fi) {
    Vestit[] resultat = new Vestit[2];
    if (ini == fi) { // Nomes un element
        resultat[0] = vestits[ini];
        resultat[1] = vestits[fi];
    } else {
        int mig = (ini + fi) / 2;
        Vestit[] r1 = cercarMesMenys(vestits, ini, mig);
        Vestit[] r2 = cercarMesMenys(vestits, mig + 1, fi);
        if (r1[0].getCost() > r2[0].getCost()) resultat[0] = r1[0];
        else resultat[0] = r2[0];
        if (r1[1].getCost() > r2[1].getCost()) resultat[1] = r2[1];
        else resultat[1] = r1[1];
    }
    return resultat;
}
```

Exercici 2

S'han de prendre decisions i tenir la possibilitat de desdir-te'n. La solució la podem expressar com una seqüència de decisions. En cada nivell ubiquem una caixa en un muntacàrregues, per tant la pregunta al nivell n-èssim serà: "la caixa n-èssima en quin

muntacàrregues la carrego". L'alçada de l'espai de cerca serà exacta i es correspondrà al número de caixes a carregar. L'amplada serà màxima i vindrà donada per el número màxim de muntacàrregues que es poden usar, que coincideix amb el número de caixes. Amb aquest plantejament no s'usarà marcatge doncs els muntacàrregues poden repetir-se. La millor solució és la que usa menys muntacàrregues i en cas d'igualtat la que té més caixes fràgils al damunt de la pila. Per ser solució s'ha d'arribar a una fulla de l'arbre i les assignacions han de complir les restriccions indicades en l'enunciat. Sempre hi haurà solució i serà la que usa tants muntacàrregues com caixes a carregar, una en cada màquina.

```
public class Repartiment {
    private Caixa caixes[];
    // Exercici 1 - Afegeix atributs
    private Muntacarrega[] sol;          // sol[i] muntacàrregues amb la seva taula de caixes
    private int qSol; // grues ocupades
    private Muntacarrega[] millor;
    private int qMillor;
    public static void carregaCaixes(Caixa[] caixes) {
        // omplena el paràmetre caixes amb tots els objectes caixa a carregar
    }
    public Repartiment(int quantes) {
        caixes = new Caixa[quantes];
        carregaCaixes(this.caixes);
        // Completar Exercici 2
        sol = new Muntacarrega[quantes];
        for (int i=0; i<sol.length; i++)
            sol[i]= new Muntacarrega(i,quantes);
        qSol = 0; // numero de muntacàrregues amb caixes
        millor = new Muntacarrega[quantes];
        for (int i=0; i<millor.length; i++)
            millor[i]= new Muntacarrega(i,quantes);
        qMillor = quantes + 1; //per provocar el canvi en la primera solució trobada
    }
 public static void main(String args[]) {
     // demanem numero de caixes a ubicar
     Repartiment m = new Repartiment(Keyboard.readInt());
     // Exercici 2
     m.backMillorSolucio(0);
     // Exercici 3
     System.out.println(m);
 }
```

```
public void backMillorSolucio(int k) {
    // Exercici 4 - k indica número de caixa a ubicar
    int i = 0;
    while (i < sol.length) {
        // acceptable caixa k al muntacarregues i
        if (acceptable(i, k)) {
            if (sol[i].getQuantes() == 0)
                qSol++;
            sol[i].addCaixa(caixes[k]);
            if (k == caixes.length-1) { // és solució
                if (millorSolucio()) {
                    for (int m = 0; m < sol.length; m++)
                        millor[m]=(Muntacarrega)(sol[m].clone());//s'ha de redefini el clone()
                    qMillor = qSol;
                }
            } else
                backMillorSolucio(k + 1);
            // si haviem ocupat per primer cop decrementem
            if (sol[i].getQuantes() == 1)qSol--;
            sol[i].remCaixa(caixes[k]);
        }
        i++;
    }
}
```

Redefinim a la classe Muntacarrega → cal indicar que la classe implementa Cloneable

```
public Object clone() {
    Object o = null;
    try {
        o = super.clone();
        Muntacarrega m = (Muntacarrega) o;
        m.caixes = new Caixa[this.caixes.length];
        for (int i = 0; i < caixes.length; i++) {
            m.caixes[i] = this.caixes[i];
        }
    } catch (Exception j) {
    }
    return o;
}
private boolean acceptable(int i, int k) {
    // Exercici 4 i--> muntacarregues / k--> caixa
    float acc = 0;
    boolean trobat = false;
    for (int j = sol[i].getQuantes() - 1; j >= 0 && !trobat; j--) {
        if (acc + caixes[k].getPes() > sol[i].donaCaixa(j).getPes())
            trobat = true;
        else
            acc += sol[i].donaCaixa(j).getPes();
    }
    return !trobat;
}
private boolean millorSolucio() {
    return qSol < qMillor || (qSol == qMillor && fragils());

}
```

Afegim a la classe Muntacarrega un mètode per obtenir la caixa j-èssima

```
public Caixa donaCaixa(int j) {
    if (quantes >= j)
        return caixes[j];
    return null;

}
```

Continuem a la classe Repartiment

```
private boolean fragils() {
    int cont1 = 0, cont2 = 0;
    for (int i = 0; i < sol.length; i++) {
        if (sol[i].getQuantes() > 0 && sol[i].getDarrera().getFragil())
            cont1++;
        if (millor[i].getQuantes() > 0
                && millor[i].getDarrera().getFragil())
            cont2++;
    }
    return cont1 > cont2;
}
public String toString() {
    // Exercici 5
    String r;
    r="La millor solució usa :" + this.qMillor + "montacarregues";
    r=r+"\nLa distribució és la següent: ";
    for (int i=0; i<millor.length; i++)
        if (millor[i].getQuantes()>0) r=r+millor[i].toString()+"\n";
    return r;
}
```

Redefinim a la classe Muntacarrega el mètode toString

```
public String toString() {
    String r = "El muntacarregues amb identificació " + this.identificacio
            + " porta de baix a dalt: ";
    float kilos = 0;
    for (int i = 0; i < this.quantes; i++) {
        r += caixes[i].toString();
        kilos += caixes[i].getPes();
    }
    return r + "amb un pes total de: " + kilos;
}
```

La tècnica voraç també pot resoldre aquest problema.
Candidats: les caixes
I la manera de fer seria intentar emplenar al màxim un muntacàrregues i no iniciar la càrrega d'altre mentre pugui carregar-se als ja "iniciats".
En cada iteració s'ha d'estriar una caixa per apilar. Com és selecciona?. La funció de selecció seria agafar la caixa de més pes.
No trobaria la millor solució. Per exemple, en l'exemple del propi enunciat trobaria la solució 2 usant 3 muntacàrregues en lloc de trobar la solució 3 que és la òptima. Col·locaria la caixa de pes 4 a sobre la de 5 i a partir d'aquí ja no es pot fer enrere.