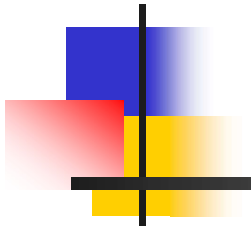


Programació Avançada

Tècniques de disseny d'algorismes

Tècnica Voraç

Filosofia de funcionament.
Aplicació.
Avantatges e Inconvenients.
Esquema/es.
Exemples.

Tècnica: Voraç (Greedy)

- També anomenada tècnica Greedy.
- Els algorismes que implementen aquesta tècnica s'anomenen voraços.
- És un esquema senzill i alhora força utilitzat.
- **S'usa bàsicament en problemes d'optimització,** és a dir, es vol resoldre un problema de forma òptima. Habitualment sota algun **criteri de maximització o minimització**.

Per exemple:

1.- Distribuir persones en una taula de comensals en funció d'afinitats.

2.- Quines cançons emmagatzemar en un CD per emplenar-lo al màxim.

Tècnica: Greedy (Voraç)

Problemes d'optimització

- En aquesta mena de problema existeix un conjunt de candidats a formar part de la solució.
- La tècnica s'ha d'ocupar de seleccionar aquells que formen la solució òptima al problema, habitualment sota algun criteri de maximització o minimització.
- L'estratègia és bàsicament iterativa.
- Aquesta tècnica funciona per etapes i en cadascuna s'escull el candidat que **sembla** més adequat independentment de les conseqüències futures.

Tècnica: Greedy (Voraç)

Filosofia:

Inicialment $C = \{c_1, c_2, \ldots c_n\}$

$S = \emptyset$

⇓ Funció Selecció

triem candidat "millor"

$C = C - \{c_i\}$

$S = S \cup \{c_i\}$

↑ si l'afegim serà definitiu

iterar fins a tenir la solució ó no hi ha candidats

Tècnica: Greedy (Voraç)

Filosofia:

→ **Pas 1:** inicialment el conjunt de candidats a formar la solució és buit. I tenim tots els candidats per escollir.

→ **Pas 2**: s'usa una **funció de selecció** basada en algun criteri d'optimització per escollir el "millor candidat". Aquest es tret del conjunt de candidats.

→ **Pas 3**: Cal esbrinar si el candidat passa o no a formar part dels conjunt solució. Si amb el nou candidat, el conjunt solució pot arribar a ser solució, és a dir, si el conjunt és **completable el candidat passarà a formar part de la solució per sempre. En cas contrari es descarta per sempre.**

→ **Pas 4**: després de cada incorporació cal esbrinar si ja s'ha arribat a la solució. Acabem en cas afirmatiu, sinó tornem al pas 2.

Tècnica: Greedy (Voraç)

Consideracions

- **Aplicació**: problemes d'optimització.
- Aquesta tècnica només troba **UNA** solució al problema.
- **Conjunt de candidats** a formar part de la solució: cal estriar els que donen lloc a la millor.
- **Filosofia de funcionament**: Procés **iteratiu**. En cada iteració cal estriar el millor candidat a formar part de la solució mitjançant l'ajut d'una **funció de selecció**.
- Quan un candidat s'ha seleccionat com a part de la solució **no** se'l pot treure.

Tècnica: Greedy (Voraç)

Esquema

```
funció voraç (conjunt_candidats C ) retorna
conjunt_solució S
      S=∅
      mentre (!solucio(S) && C!= ∅)
            X=seleccio_candidat_optim
            C=C-{X}
            si  factible(S,X)  llavors  S=SU{X}
            fsi
      fimentre
      si solucio(S) retorna S
      sino no hi ha solucio fsi
fifunció
```

Tècnica: Greedy (Voraç)

Consideracions

- **Avantatges**:
  - Fàcils d'implementar
  - Rapidesa – eficiència en temps
- **Inconvenients**
  - Determinació de la funció de selecció
  - Una funció de selecció que selecciona el candidat **òptim en cada iteració no necessàriament troba la millor**
  - No sempre troba la millor però si troba solucions bones

Tècnica: Greedy (Voraç)

Exemple 1

Canvi

- ***Objectiu***: retornar una quantitat de diners amb el **menor** nombre possible de monedes.
- ***Paràmetres:***

  import a retornar

  valor i quantitat (pot ser infinit) de les monedes disponibles

- ***Candidats***: cadascuna de les monedes dels diferents tipus que es disposa.
- ***Solució***: conjunt de monedes amb import igual a l'import a retornar.
- ***Solució factible-completable***: la suma dels valors de les monedes escollides com a solució no ha de superar l'import a retornar.
- ***Selecció de moneda*** (de candidat): la de **major import** possible entre les candidates. **Molt important!**
- ***Objectiu***: minimitzar el nombre de monedes a retornar.

Anàlisi

Tècnica: Greedy (Voraç)

Canvi

Infinites monedes!

```
public class Aplicacio{
   public static void main(String args[]){
        double []c={2,1,0.5,0.2,0.1,0.05,0.02,0.01};
        //monedes d'euro
        double [] Solucio=new double[100];
        int dimS;
        double Import=Keyboard.readDouble();
        dimS=dona_canvi(Import, c,Solucio);
        if (dimS!=0)
            for(int i=0; i<dimS; i++)
                System.out.println("moneda:  "+Solucio[i]);
            else
                System.out.println("No existeix solucio");
   }
```

Suposem és suficient!

Crida al procediment voraç

Sempre hi ha solució?

Tècnica: Greedy (Voraç)

Canvi

```
private static int dona_canvi(double Import, double
[]Candidats, double []Solucio){
   int dimS=0;
   double suma=0.0; //acumulat solució
   // Els candidats venen ordenats de major a menor valor
   int quina; //índex del següent candidat a considerar
   boolean trobat=false; quina=0;
   while (quina<Candidats.length && !trobat){
       if (suma+Candidats[quina]<=Import){
              Solucio[dimS]=Candidats[quina];
              suma+=Solucio[dimS];
              dimS++; //Acumulem
              trobat=(suma==Import);
       } //fi if
```

Funció de selecció

Amb una condició hi ha suficient

Tècnica: Greedy (Voraç)

Canvi

```
            else //oblidem aquest candidat
                    quina++;
        } //fi while
        if (!trobat) /*No hi ha solució */
              dimS=0;
        return dimS;
} //fi procediment
```

En aquest cas SEMPRE hi ha solució

Tècnica: Greedy (Voraç)

Canvi

```
private static int dona_canvi(double Import, double
[]Candidats, double []Solucio){
   int dimS=0;
   double suma=0.0; //acumulat solució
   // Els candidats venen ordenats de major a menor valor
   int quina; //índex del següent candidat a considerar
   boolean trobat=false; quina=0;
   while (!trobat){ //en aquest exercicis tenim candidats infinits
        if (suma+Candidats[quina]<=Import){
                Solucio[dimS]=Candidats[quina];
                suma+=Solucio[dimS];
                dimS++; //Acumulem
                trobat=(suma==Import);
        } //fi if
        else    quina++;
   } //fi while
   return dimS;
} //fi procediment
```

Funció de selecció

Tècnica: Greedy (Voraç)

- Exercici: Modifiqueu l'enunciat, ara suposant que **NO es disposa d'un nombre infinit** de monedes de cada tipus, serà l'usuari de l'aplicació qui indicarà quantes monedes es tenen de cada tipus, donant per vàlid el valor de zero.

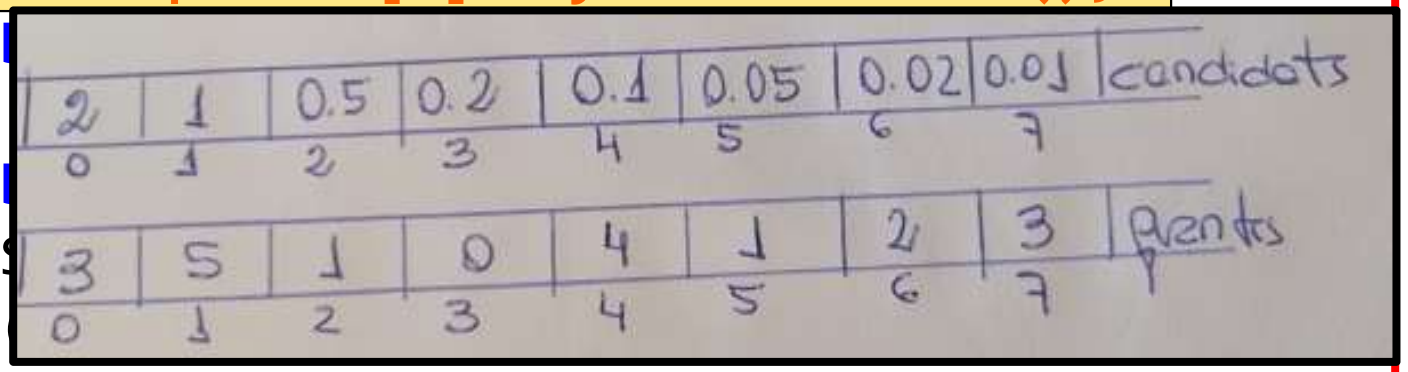
Canvi

Tècnica: Greedy (Voraç)

Solució 1

Canvi

```
public class Aplicacio{
   public static void main(String args[]){
      double []c={2,1,0.5,0.2,0.1,0.05,0.02,0.01};
      int []quantes= new int[8];
      for(int i=0; i<8;i++)
            quantes[i]=Keyboard.readInt();
      doul
      int
      doul
      dimS
      if
         for(int i=0; i<dimS; i++)
            System.out.println("moneda: "+Solucio[i]);
         else System.out.println("No existeix o no trobat");
}
```



Sempre hi ha solució?

Tècnica: Greedy (Voraç)

Solució 1

Canvi

```
public class Aplicacio{
   public static void main(String args[]){
     double []c={2,1,0.5,0.2,0.1,0.05,0.02,0.01};
     int []quantes= new int[8];
     for(int i=0; i<8;i++)
            quantes[i]=Keyboard.readInt();
     double [] Solucio=new double[100];
     int dimS;
     double Import=Keyboard.readDouble();
     dimS=dona_canvi(Import, c, Solucio, quantes);
     if (dimS!=0)
        for(int i=0; i<dimS; i++)
           System.out.println("moneda: "+Solucio[i]);
        else System.out.println("No existeix o no trobat");
   }
```

No!!! Fins i tot pot haver i aquest programa voraç no trobar-la!!!

Tècnica: Greedy (Voraç)

Canvi

```
private static int dona_canvi(double Import,
double[]Candidats,double []Solucio, int []quantes){
   int dimS=0; double suma=0.0;
   // Els candidats venen ordenats de major a menor valor
   int quina; //índex del següent candidat a considerar
   boolean trobat=false; quina=0;
   while (quina<Candidats.length && !trobat){
       if(quantes[quina]>0 && suma+Candidats[quina]<=Import){
              Solucio[dimS]=Candidats[quina];
              dimS++;suma+=Candidats[quina];
              trobat=suma==Import;
              quantes[quina]--;
       }
       else // oblidem aquest candidat
              quina++;
    } //fi while
   if (!trobat) return 0; return dimS; } //fi funció
```

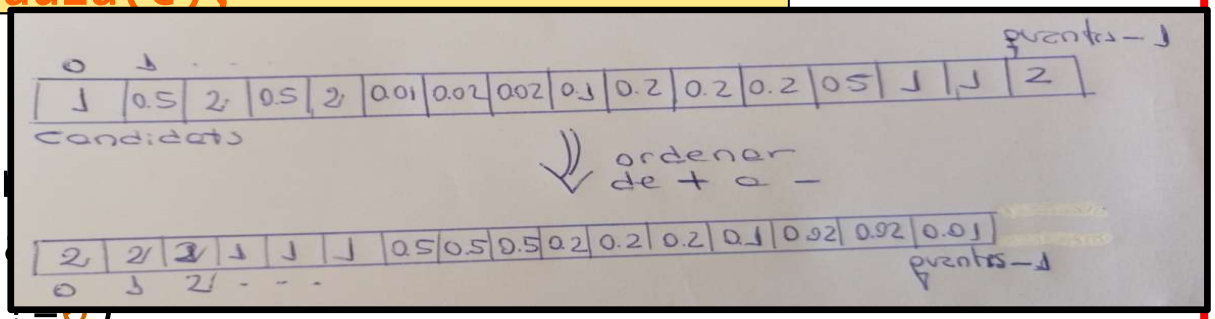
Funció de selecció

Tècnica: Greedy (Voraç)

Solució 2

Canvi

```
public class Aplicacio{
   public static void main(String args[]){
        int quantes= Keyboard.readInt();
        double []c=new double[quantes];
        for(int i=0; i<quantes;i++)
              c[i]=Keyboard.readInt();
        OrdenarTaula(c);
        double [
        int dimS
        double I
        dimS=don
        if (dimS!=0)
         for(int i=0; i<dimS; i++)
              System.out.println("moneda:  "+Solucio[i]);
        else
          System.out.println("No existeix o no trobat");
   }
```



Tècnica: Greedy (Voraç)

Solució 2

Canvi

```
public class Aplicacio{
   public static void main(String args[]){
        int quantes= Keyboard.readInt();
        double []c=new double[quantes];
        for(int i=0; i<quantes;i++)
              c[i]=Keyboard.readInt();
        OrdenarTaula(c);
        double [] Solucio=new double[100];
        int dimS;
        double Import=Keyboard.readDouble();
        dimS=dona_canvi(Import, c, Solucio);
        if (dimS!=0)
         for(int i=0; i<dimS; i++)
              System.out.println("moneda:  "+Solucio[i]);
        else
          System.out.println("No existeix o no trobat");
   }
```

Tècnica: Greedy (Voraç)

Canvi

```
private static int dona_canvi(double Import, double
[]Candidats,double []Solucio){
   int dimS=0; double suma=0.0;                    Funció de selecció
   // Els candidats venen ordenats de major a menor valor
   int quina; //índex del següent candidat a considerar
   boolean trobat=false; quina=0;
   while (quina<Candidats.length && !trobat){
       if (suma+Candidats[quina]<=Import){
              Solucio[dimS]=Candidats[quina];
              dimS++;suma+=Candidats[quina];
              trobat=suma==Import;
       }
       // sempre passem al següent candidat
       quina++;
    } //fi while
   if (!trobat) return 0; return dimS; } //fi funció
```

Tècnica: Greedy (Voraç)

Exemple 2

fitxers

- **Enunciat**:

  Es tenen N fitxers enumerats del 0 al N-1, cadascun d'ells d'una mida M_i megabytes. **Es volen emmagatzemar en un disc d'una capacitat màxima Max megabytes**, però no necessàriament tots els fitxers caben en el disc. Cal escriure un programa que a partir de les dades entrades per l'usuari:

  → capacitat màxima del disc, Max

  → nombre de fitxers, i

  → mida de cada fitxers,

  determini **quins** fitxers s'han d'emmagatzemar al disc si es vol emmagatzemar el **màxim nombre possible d'ells**.

  Cal trobar la solució aplicant la tècnica voraç.

Tècnica: Greedy (Voraç)

fitxers

1.- Quins són els candidats a formar part de la solució?

Els candidats a formar part de la solució són **tots** els fitxers.

2.- Quina funció de selecció cal utilitzar per trobar la millor solució.

En aquest cas per obtenir la millor solució la funció de selecció ideal és la de seleccionar els fitxers de menor mida a major. Per tant proporcionarem els candidats ordenats de menor a major mida.

3.- La tècnica voraç trobarà la solució òptima?

Si, en aquest problema, la solució trobada serà l'òptima.

Tècnica: Greedy (Voraç)

fitxers

```
public static void main(String args[]){
    double candidats[];  int N;
    do{    System.out.println("Quants fitxers tens ?");
           N=Keyboard.readInt();
     }while (N<=0);
     candidats= new double[N];                    Entrada de dades
     for (int i=0; i<N; i++){
            System.out.println("Entra la mida del fitxer "+ i);
            candidats[i]=Keyboard.readDouble(); }
     double CapMax=Keyboard.readDouble();
     ordenarCandidats(candidats); Funció de selecció
     double []Solucio=new double[candidats.length];
     for (int i=0; i<Solucio.length; i++)Solucio[i]=0.0;
     double ocupat=quinsFitxers(CapMax, candidats, Solucio);
     System.out.println("La capacitat ocupada és de: " + ocupat);
     for(int i=0; i<Solucio.length && Solucio[i]!=0.0; i++)
            System.out.println("el fitxer de mida "+ Solucio[i] +
           "el possem ");
                                                  Resultats
}
public static void ordenarCandidats(double []taula){  /*bombolla*/ }
```

Tècnica: Greedy (Voraç)

fitxers

```
public static double quinsFitxers(double CapacitatDisc,
double[]Candidats, double []Solucio){
      // els candidats venen ordenats de menor a major
       int quin; //índex del següent candidat a considerar
       boolean trobat=false; quin=0; double CapacitatOcupada=0.;
       while (quin<Candidats.length && !trobat){
        if (CapacitatOcupada+Candidats[quin]<=CapacitatDisc){
              Solucio[quin]=Candidats[quin];
              CapacitatOcupada+=Candidats[quin];
              trobat=(CapacitatOcupada==CapacitatDisc);
             // passem al següent candidat
             quin++;
         }
         else trobat =true; //són massa grans
       } //fi while
       return CapacitatOcupada;
} //fi procediment
```

Funció de selecció

Important!

Tècnica: Greedy (Voraç)

fitxers

- **<u>Modificació Enunciat</u>**:

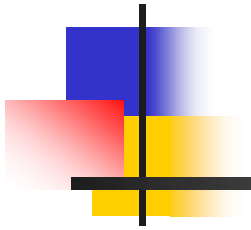
  Amb les mateixes de dades de l'enunciat previ ara és vol:

  Determinar quins fitxers s'han d'emmagatzemar al disc si es vol **maximitzar l'espai usat del disc**.

  Quina funció de selecció cal utilitzar per trobar la millor solució?

Es pot usar una funció de selecció que escolleixi els candidats de major a menor mida.

Es òptima?



Programació Avançada

Tècnica Voraç

Exercici 35 pàgina 97

Tècnica: Voraç

Enunciat

Una caixera d'un supermercat té una cua de carros de compra per cobrar, per a cada compra es coneixen les següents dades:

```
public class CarroCompra {
        private int numProductes; // productes comprats
        private long clientDNI;
        public CarroCompra(int numP, long dni){
                numProductes=numP; clientDNI=dni;}
        public int getNumProductes(){ return numProductes;}
        public long  getClientDNI(){ return clientDNI;}
} // fi classe
```

La **suma** dels temps d'espera de tots els clients que té a la cua **depèn de l'ordre en que es realitzin els cobraments**, per exemple en la següent cua de carros de compra:

Carro 1: 50 productes
Carro 2: 100 productes
Carro 3: 30 productes

Carro 1 → 50
Carro 2 → 50+100=150
Carro 3 → 150 + 30 = 180
Total=50+150+180=380

Unitat de temps per producte

Tècnica: Voraç

Enunciat

Cal escriure un algorisme **voraç** que trobi l'ordre en que s'han d'atendre els clients de la cua per a **minimitzar** el temps d'espera total. L'algorisme ha de calcular aquest valor mínim.
Abans d'escriure l'algorisme cal que contesteu les següents qüestions:

- → 1.- En aquest problema qui són els candidats?. Justifiqueu la resposta.
- → 2.- La funció de selecció ha d'escollir el millor candidat en cada iteració, quin criteri seguirà la vostra funció per fer la tria del candidat ?.
- → 3.- La vostra funció de selecció trobarà sempre la millor solució al problema?. Argumenteu la resposta, en cas negatiu cal trobar un contraexemple que ho il·lustri.

Cal usar i completar les següents classes (s'ha de seguir l'estructura marcada i s'han de implementar tots els mètodes). Podeu afegir atributs, mètodes implementats, variables locals, ... necessaris per la vostra proposta de solució:

Tècnica: Voraç

Enunciat

```
public class Solucio{
   private Cua carros; //emmagatzema els objectes CarroCompra
   public Solucio(){ //constructor
   // sentències per la creació i emplenat de la cua de carros a
   // cobrar. Cal demanar a l'usuari les dades
   }
   public static void main(String ars[]){
   //creació objectes i variables
   // crida procediment voraç
       System.out.println(?????);  //temps total d'espera
       System.out.println(?????);  //solució a pantalla, millor
                                   // ordre de cobrament
   }
   public int voraç(?????){
   // sentències esquema voraç
        return ?????? //temps total d'espera
   }
} // fi classe
```

Tècnica: Voraç

Enunciat

```
public class Solucio{
   private Cua carros; //emmagatzema els objectes CarroCompra
   public Solucio(){ //constructor
   // sentències per la creació i emplenat de la cua de carros a
   // cobrar. Cal demanar a l'usuari les dades
   }
   public static void main(String ars[]){
   //creació objectes i variables
   // crida procediment voraç
       System.out.println(?????);  //temps total d'espera
       System.out.println(?????);  //solució a pantalla, millor
                                   // ordre de cobrament
   }
   public int voraç(??
   // sentències esquer
        return ?????? /
   }
} // fi classe
```

```
public interface Cua{
   void encuar (Object o) throws Exception;
   Object desencuar () throws Exception;
   boolean cuaBuida();
   boolean cuaPlena();
   Object  consultaCua () throws Exception;
   void buidar();
}
```

Cua.java

Tècnica: Voraç

Anàlisi

- **Candidats**: els carros de la compra que estan a la cua de la caixa.
- **Funció Selecció**: de menys nombre de productes a més.
- Dona **sempre** la solució **òptima** al problema.
- **Atributs:** **Afegits!**

```
private Cua carros; //emmagatzema els
    //carros que estan a la cua de la caixa
private Cua solucio; //Afegit, per
    // emmagatzemar la solució!!
```

Tècnica: Voraç

Solució

```
public Solucio(){ //constructor
   carros=new CuaEnll();
   int quants=Keyboard.readInt();
   for (int i=0; i<quants; i++)
   try{ carros.encuar(new
                      CarroCompra(Keyboard.readInt(),null));}
   catch(Exception e){}
   }
   solucio=new CuaEnll();
}
public static void main(String ars[]){
   Solucio s= new Solucio();
   System.out.println(s.voraç()); // temps total d'espera
   System.out.println(s); // solució a pantalla, millor ordre
   treballs
}
public String toString(){
   return solucio.toString();
}
```

constructor

Classe implementadora de la Cua

Classe implementadora de la Cua

main

toString

Suposem que la classe Cua té redefinit aquest mètode

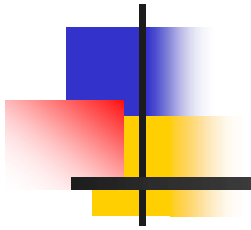
Tècnica: Voraç voraç

Solució

```
public int voraç(){
   int temps=0; int total=0;
   carros.ordenaMenorMajor(); //suposem ok!!   -
                              //ordena per numProductes
   while (!carros.cuaBuida()){ //sempre hi ha solució
      temps+=((CarroCompra)carros.consultaCua()).
                                   getNumProductes();
      total+=temps;  //Compte!, suma de temps
                     //d'espera!!!
      try{solucio.encuar(carros.desencuar());}
      catch(Exception e){}
   }
   return total;//temps total
}
```

Funció de selecció

Observeu que la iteració només té una condició. Tots els candidats han de formar part de la solució



Programació Avançada

Exercici 29 pàgina 86

Festa Aniversari

Tècnica Voraç

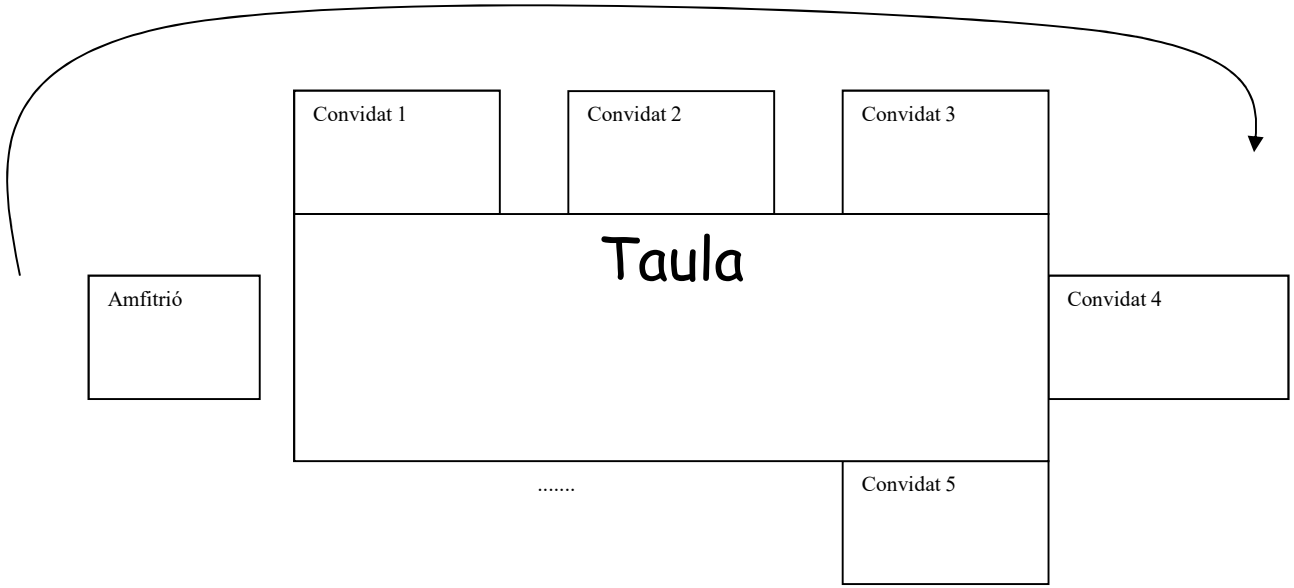
Tècnica Voraç: Aniversari

Enunciat

- Amb motiu del setantè aniversari de l'avi Pep, la família li vol organitzar un dinar sorpresa.
- S'ha de decidir la distribució dels convidats en la taula, per això es tindran en compte els graus d'afinitat entre els convidats, aquesta informació s'ha codificat numèricament amb un valor sencer dins l'interval [0,5], essent el valor de 5 el millor grau d'afinitat i 0 el pitjor.
- Es tracta d'escriure un **algorisme per trobar la millor distribució de convidats, amb suma total d'afinitats màxima**.
- Al cap de la taula s'hi assentarà l'amfitrió, l'algorisme ha de d'anar determinant a partir d'aquest i per a cadascun dels convidats que anem assentant a qui assentem al seu costat, féu aquesta distribució seguint el sentit de les agulles del rellotge.

Tècnica Voraç: Aniversari

Enunciat



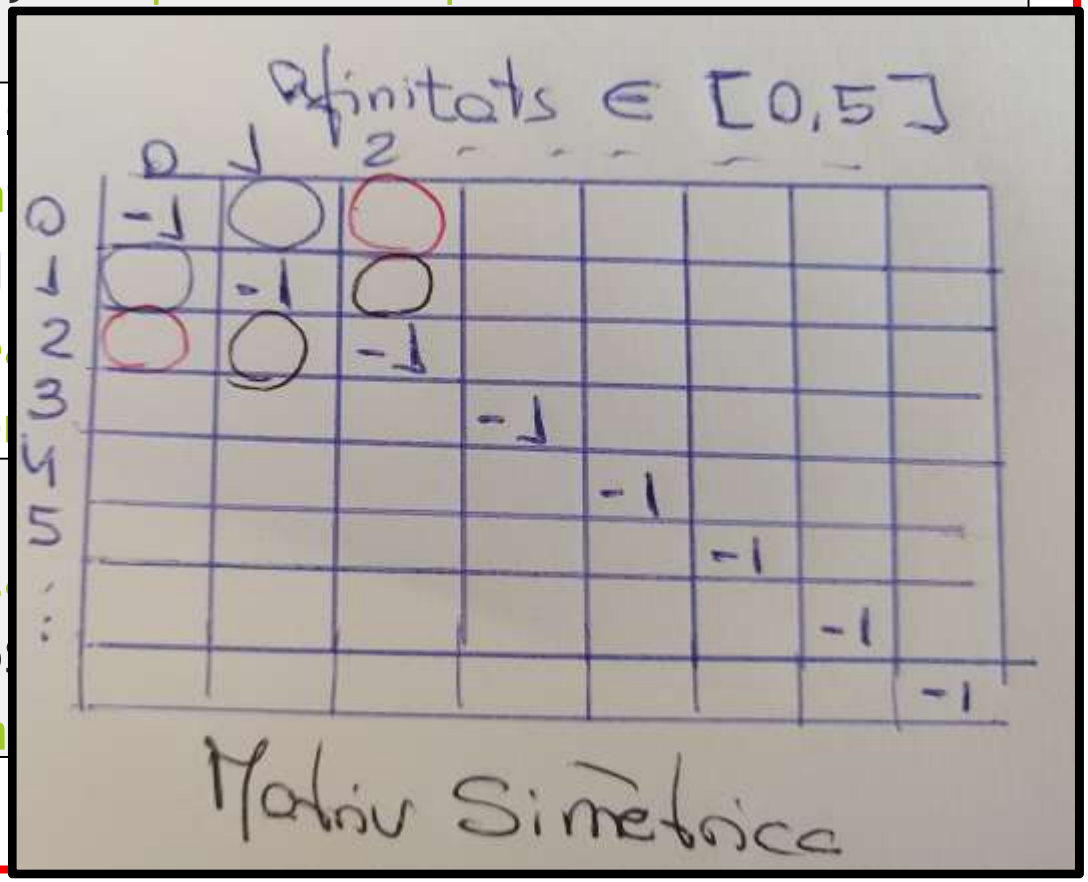
Dades: nombre de convidats i nom de cadascun. Afinitats

Objectiu: distribuir-los maximitzant l'afinitat total (suma)

Tècnica Voraç: Aniversari

Estructura

```
public class Aniversari{
   private int afinitats[][]; //serà simètrica
   private String convidats[]; //noms convidats
   private int quants; // posicions plenes de la taula
   de convidats
   public Aniversari(
       /*creació maga
   public static void
   /*declaració i cre
   crida al mètode vo
   public ????? Voraç
   /*crida obligatòri
   public ???? Funcio
       /*determinar m
} // fi classe
```



Tècnica Voraç: Aniversari

Estructura

```
public class Aniversari{
   private int afinitats[][]; //serà simètrica
   private String convidats[]; //noms convidats
   private int quants; // posicions plenes de la taula
   de convidats
   public Aniversari(int quantsConvidats){
       /*creació magatzems+entrada dades*/ }
   public static void main(String args[]){
   /*declaració i creació objectes i variables
   crida al mètode voraç + visualització de resultats*/ }
   public ????? Voraç(?????){
   /*crida obligatòria a la funció de selecció*/ }
   public ???? FuncioSeleccio(?????){
       /*determinar millor candidat*/ }
} // fi classe
```

Tècnica Voraç: Aniversari

Anàlisi

Anàlisi

- 1.-**Candidats**: cadascun dels convidats, excepte l'amfitrió.
- 2.- **Funció selecció**: com es vol maximitzar el grau total d'afinitats començant per l'amfitrió buscarem el seu millor veí X a assentar a la seva esquerra (amb qui té major grau d'afinitat), del X el seu millor Y, i així successivament fins que tots els convidats estiguin assentats.
- 3.- **No** necessàriament trobarà la millor solució, però **sempre hi haurà solució.**

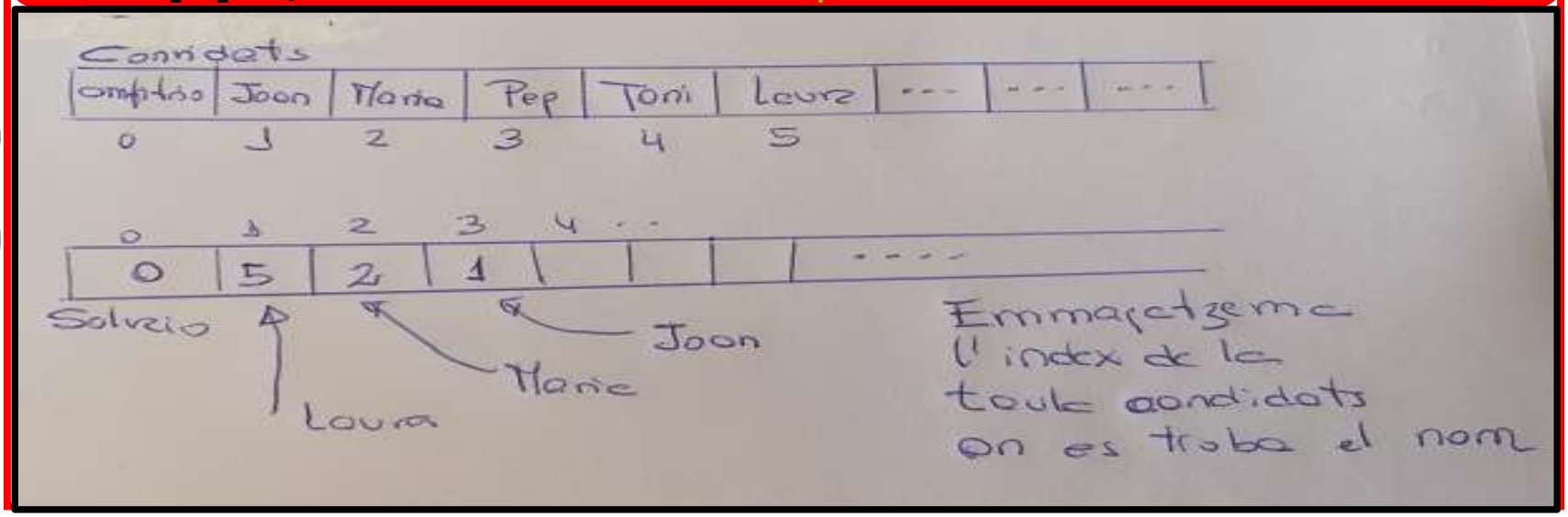
Aniversari - Constructor

Solució

```
public Aniversari( int quantsConvidats){
   quants=quantsConvidats;
   convidats=new String[quants];
   System.out.println("Especifica el nom de l'amfitrió");
   convidats[0]=Keyboard.readString();
   for (int i=1; i<quantsConvidats; i++)
        convidats[i]=Keyboard.readString();
   sol= new int [quants]; //índex convidat!!!!!
   sol[0]=0; //assentem amfitrió primer lloc
```

Entrada de dades

afegit atribut!!



Aniversari - Constructor

Solució

```
public Aniversari( int quantsConvidats){
    quants=quantsConvidats;                                   Entrada de dades
    convidats=new String[quants];
    System.out.println("Especifica el nom de l'amfitrió");
    convidats[0]=Keyboard.readString();
    for (int i=1; i<quantsConvidats; i++)
        convidats[i]=Keyboard.readString();
    sol= new int [quants]; //índex convidat!!!!!
    sol[0]=0; //assentem amfitrió primer lloc
    afinitats= new int [quants][quants];
    //a la diagonal no cal posar res !
    for (int i=0; i<quants; i++)
        for (int j=i; j<quants; j++){ //triangle superior
            if (i==j) afinitats[j][i]=-1; //sentinella
            else{ afinitats[i][j]=(int)(6*Math.random());
                  afinitats[j][i]=afinitats[i][j];}
    }
}
```

afegit atribut!!

Aniversari: Mètode main

Solució

```
public static void main(String args[]){
   int quants=Keyboard.readInt();
   Aniversari a=new Aniversari(quants);
   int guany=a.Voraç();
   System.out.println("L'afinitat global és de:" +guany);
   System.out.println(a);
}

public String toString(){
   String r="Llista començant per l'amfitrió i seguint
   el sentit de les agulles del rellotge";
   for (int i=0; i<sol.length; i++)
        r+=convidats[sol[i]]+" ";
   return r;
}
```

Aniversari: Mètode voraç

Observeu que la iteració només té una condició

solució

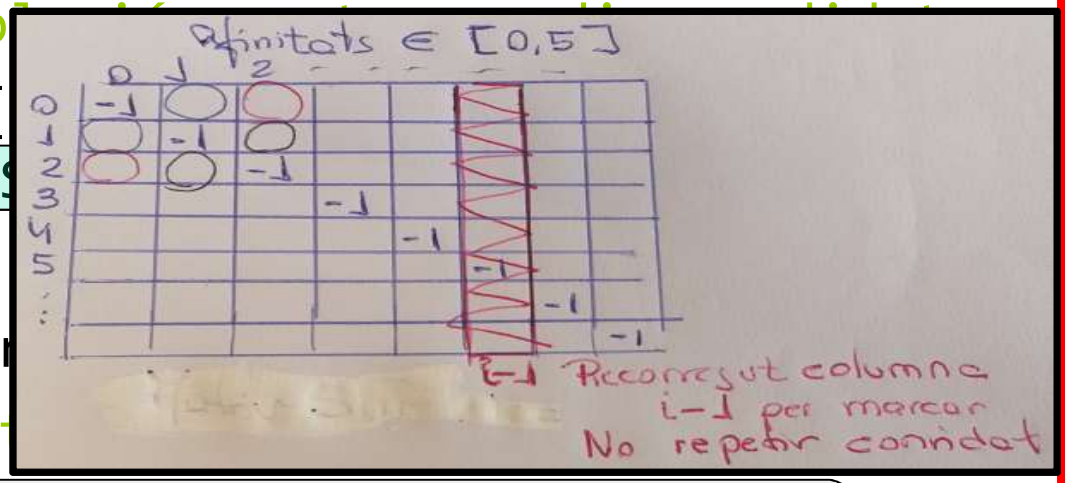
```
public int Voraç(){
   int afinitat=0;int i=1; //índex a emplenar
   //sempre hi ha solució- mentre quedin candidats
   while (quants>1){ //l'amfitrió no l'hem de col·locar
      sol[i]=FuncioSeleccio(sol[i-1]);
       quants--;
       afinitat+=afinitat[sol[i-1]][sol[i]];i++;
       //per no repetir candidat!!!! - Marcatge
       for (int u=0; u<afinitats.lenght; u++)
            afinitats[u][sol[i-1]]=-1;
   }
   return afinitat+=afinitats[0][sol[i-1]];???
}
```

Aniversari: Mètode voraç

Observeu que la iteració només té una condició

solució

```
public int Voraç(){
   int afinitat=0;int i=1; //índex a emplenar
   //sempre hi ha so
   while (quants>1){
       sol[i]=Funcio
        quants--;
        afinitat+=afi
        //per no repe
        for (int u=0; u<afinitats.lenght; u++)
              afinitats[u][sol[i-1]]=-1;
   }
   return afinitat+=afinitats[0][sol[i-1]];
}
```



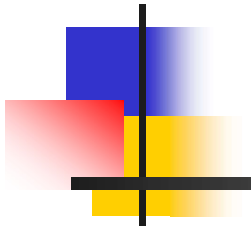
???

Aniversari: Funció Selecció

Solució

```
public int FuncioSeleccio(int veí){
   // Determinar el valor major de la fila veí
   int quin=-1; //l'amfitrió no el podem escollir
   for (int i=1; i<afinitats.lenght; i++){
       if (quin==-1 ||
          afinitats[veí][i]>afinitats[veí][quin])
             quin=i;
   }
   return quin;
}
```

Busquem quin és més afí a veí. Recorregut per la fila veí



Programació Avançada

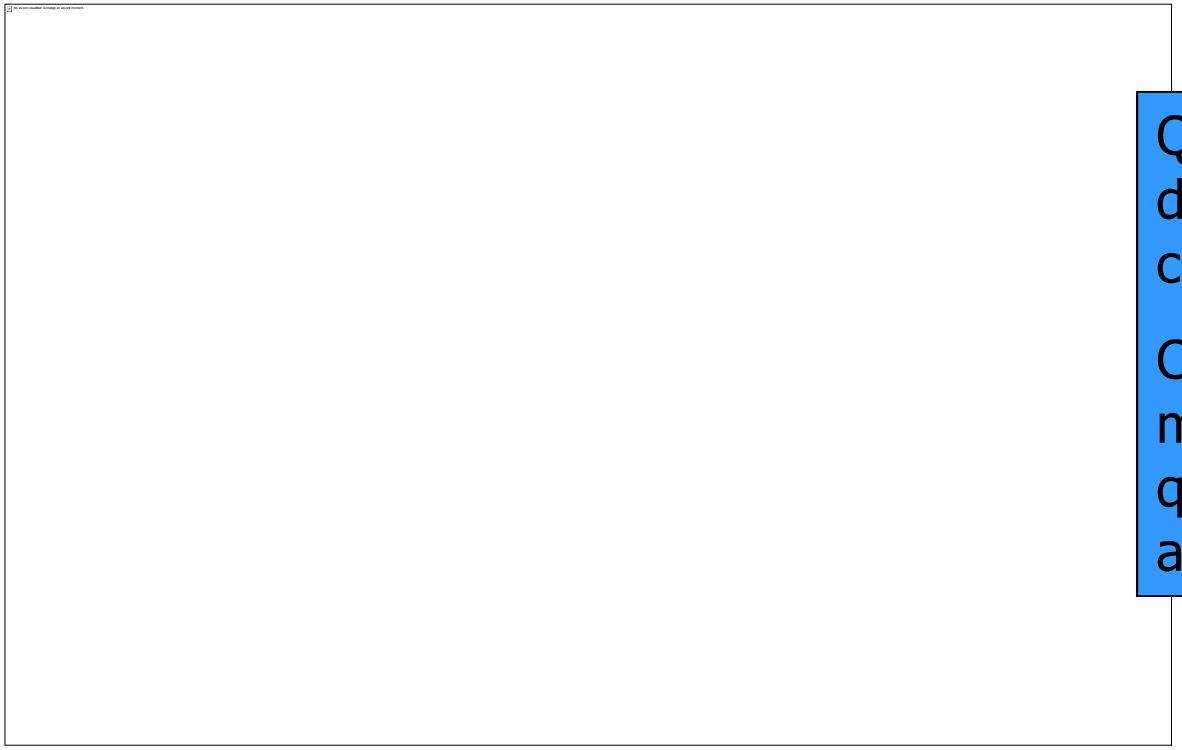
Exercici 18 pàgina 67

El rei coix

Tècnica Voraç

Tècnica Voraç: El rei coix

Enunciat



Quantitat d'euros en cada casella.

Cal maximitzar la quantitat final acumulada

- **Moviment 1**: la seva posició canviarà a la posició (x-1, y) **Amunt**
- **Moviment 2**: canviarà a la posició (x, y+1) **Dreta**
- **Moviment 3**: canviarà a la posició (x-1, y+1) **Diagonal amunt**

Tècnica Voraç: El rei coix

Enunciat

```
public class Taulell{
   private class Coordenada{
        private int x,y;
        public Coordenada(int x, int y){
               this.x=x; this.y=y;}
        public String toString(){
               return new String(x+””+y);}
        public int getX(){return x;}
        public int getY(){return y;}
   } // fi classe interna
   public static final int N=8; // taulell quadrat de 8x8
   private double [][]taulell; Atribut
   public Taulell(){ //constructor
   // sentències per la creació i omplenat del taulell
}
```

Tècnica Voraç: El rei coix

Enunciat

```
public static void main(String args[]){
  //creació objectes i variables
  int quantes = ??????;//crida procediment voraç
  for (int i=0; i<quantes; i++)
            System.out.println(?????);
  // solució a pantalla
}
```

Tècnica Voraç: El rei coix

Enunciat

```
public int voraç(?????){
   int x=7,y=0; //posició inicial del rei
   // emmagatzematge en el magatzem solució de la
   // casella de sortida
   // sentències esquema voraç obligatòriament s'ha de
   // cridar al mètode que aplica la funció de
   // selecció
   System.out.println(???????); //guany del rei amb la
   // solució trobada
   return ?????? //nombre de caselles que formen la
   // solució
}
```

Tècnica Voraç: El rei coix

Enunciat

```
private Coordenada FuncioSeleccio(int x, int y){
   // els paràmetres determinen la posició actual
   // del rei, determinar i retornar quin
   // moviment aplicar, coordenada on ubicar el
   // rei
   // Sentències
   }
}
```

Tècnica Voraç: El rei coix

Anàlisi

Anàlisi:

- El rei sempre podrà arribar a la casella destí. **Sempre** hi haurà solució! El nombre màxim de moviments a realitzar per arribar des de la sortida a la casella de destí són 14. Podem repetir caselles?
- **Candidats: els tres moviments permesos**. No sempre seran factibles els tres.
- **Funció Selecció:** s'escollirà aquell moviment que comporti a la casella que emmagatzemi més diners.
- No és una funció òptima.

Tècnica Voraç: El rei coix

Solució

```
public class Taulell{
   private class Coordenada{
      ........
   } // fi classe interna
   public static final int N=8;
   private double [][]taulell;
   public Taulell(){ //constructor
   // sentències per la creació i omplenat del taulell
       taulell=new float[N][N];
       for (int i=0; i<N; i++)
              for (int i=0; i<N; i++)
                 taulell[i][j]=Keyboard.readDouble();
   }
```

Tècnica Voraç: El rei coix

Solució

```
public static void main(String ars[]){
  //creació objectes i variables
  Taulell t=new Taulell();
  Coordenades []s = new Coordenades[15];
  // 14+1 per emmagatzemar també la casella de
  // sortida
  int quantes = t.voraç(s);
  for (int i=0; i<quantes; i++)
            System.out.println(s[i]);
  // solució a pantalla
}
```

Tècnica Voraç: El rei coix

Solució

```
public int voraç(Coordenades []s ){
   int x=7,y=0; //posició inicial del rei
   double guany=0.0;
   int quantes=1; s[0]=new Coordenada(7,0);
   while (x!=0 || y!=7){ //sempre hi ha solució
        Coordenada on=FuncioSeleccio(x,y);
        s[quantes]=on; quantes++;
        x=on.getX(); y=on.getY();
        guany+= taulell[x][y];
   }
   System.out.println(guany); //guany del rei
   return quantes;
}
```

Tècnica Voraç: El rei coix

Solució

```
private Coordenada FuncioSeleccio(int x, int y){
   if (x==0) return new Coordenada(x,y+1);
   if   (y==7)   return   new   Coordenada(x-1,y);
   // tenim els 3 moviments per escollir
   if (taulell[x-1][y+1]) > taulell[x][y+1] &&
            taulell[x-1][y+1] >taulell[x-1][y])
             return new Coordenada(x-1,y+1);
   if (taulell[x][y+1] > taulel[x-1][y])
            return new Coordenada(x,y+1);
   return new Coordenada(x-1,y);
}
} // fi classe
```

Tècnica Voraç: El rei coix

Millora!

Anàlisi

Anàlisi:

- El rei sempre podrà arribar a la casella destí. Sempre hi haurà solució!!!. El nombre màxim de moviments a realitzar per arribar des de la sortida a la casella de destí són 14.
- **Candidats:** els tres moviments permesos. No sempre seran factibles els tres.
- **Funció Selecció:** s'escollirà aquell moviment que comporti a la casella que emmagatzemi més diners. Només considerant la dreta o amunt ja que si hi ha més diners a la diagonal podríem accedir-hi posteriorment.
- No és una funció òptima.

Refer funció selecció