Tècnica de backtracking

Exercici backtracking

Assignació d'àrbitres a partits de futbol

Exercici 13 pàgina 55

Tècnica de backtracking

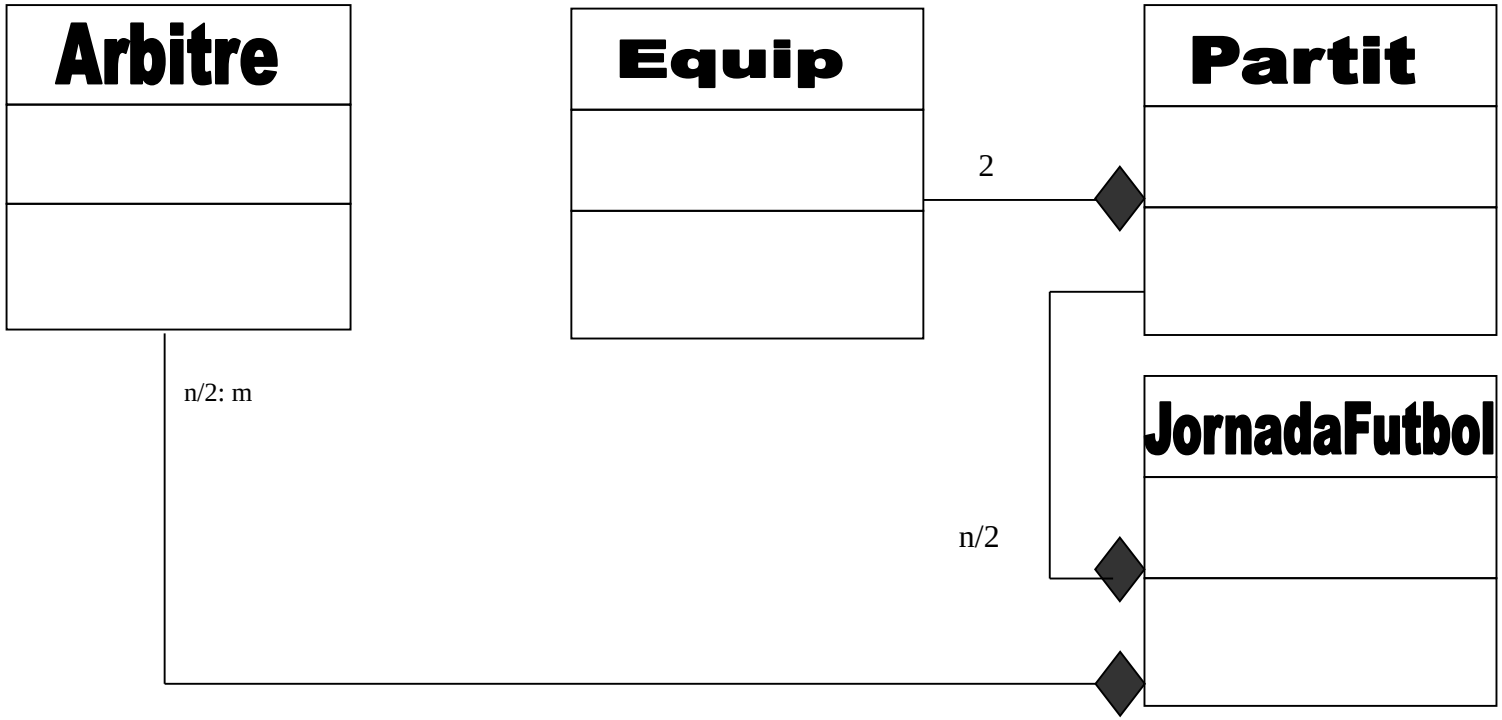
Assignació Arbitres

- **Enunciat**: Assignació d'arbitres als partits d'una jornada futbolística. Dades:
  - N equips (parell) → **N/2 partits**
  - **M àrbitres** → M>=N/2
  - Valoració equip a cada àrbitre: [0,10]
- **L'objectiu** és, per una jornada determinada, assignar un àrbitre **diferent** a cada partit, de manera que, la puntuació total dels àrbitres assignats sigui **màxima** tenint en compte les **preferències** dels equips. Cal aplicar la tècnica del backtracking per a trobar la **millor** assignació.
- Cal denotar que **no serà admissible** com a solució, assignar un àrbitre a un determinat partit si l'equip local i/o l'equip visitant són de la mateixa comarca que l'àrbitre. **Restricció del problema.**

Tècnica de backtracking

Assignació Arbitres

- Disseny de classes:



Tècnica de backtracking

Assignació Arbitres

```
public class Arbitre{
   private String nom;
   private String comarca;
   public Arbitre(String nom, String comarca){
       this.nom=nom;
       this.comarca=comarca;
   }
   public String getNom(){return nom;}
   public String getComarca(){return comarca;}
   public String toString(){
       return ("Nom Arbitre: "+ nom + "  Comarca " +
                                          comarca  +  “\n”);
   }
} //fi classe
```

Tècnica de backtracking

Assignació Arbitres

```
public class Equip{
   private String nom;
   private String comarca;
   public Equip(String nom, String comarca){
        this.nom=nom;
        this.comarca=comarca;
   }
   public String getNom(){return nom;};
   public String getComarca(){return comarca;}
   public String toString(){
        return ("Nom: "+ nom + “
                        Comarca " +  comarca + “\n”);
    }
}
```

Tècnica de backtracking

Assignació Arbitres

```
public class Partit{
   private Equip Local;
   private Equip Visitant;
   private int golsLocal;private int golsVisitant;
   public Partit(Equip local, Equip visitant){
        Local=local; Visitant=visitant;
        golsLocal=0; golsVisitant=0;
   }
   public Equip getLocal(){return Local;}
   public Equip getVisitant (){return Visitant;}
   public void setGolsLocal(int valor){golsLocal=valor;}
   public void setGolsVisitant(int valor){
        golsVisitant=valor;}
   public String toString(){
        return ("Local "+ Local.toString() + "  Visitant " +
                                  Visitant.toString() +"\n"); }
}
```

Tècnica de backtracking

Assignació Arbitres

```
public class JornadaFutbol{
    private int numJornada;
    private Arbitre []Arbitres;
    private int numArbitres; //dim real de la taula prèvia
    private Partit []Partits;
    private int numPartits; //dim real de la taula prèvia
    private    int    [][]preferencies;    //files→equips    -
  columnes→àrbitres
    private class Parella{ //classe privada
          Arbitre a; Partit p;
          public Parella(Arbitre a, Partit p)
           {this.a=a; this.p=p;}
    }
    private Parella MillorSolucio[]; private int costMillor;
    private void omplenaPreferencies(){
    /*sentències Random per omplenar la taula preferencies*/}
    private void omplenaPartits(){
    /*omplena la taula Partits amb els partits de la jornada*/ }
```

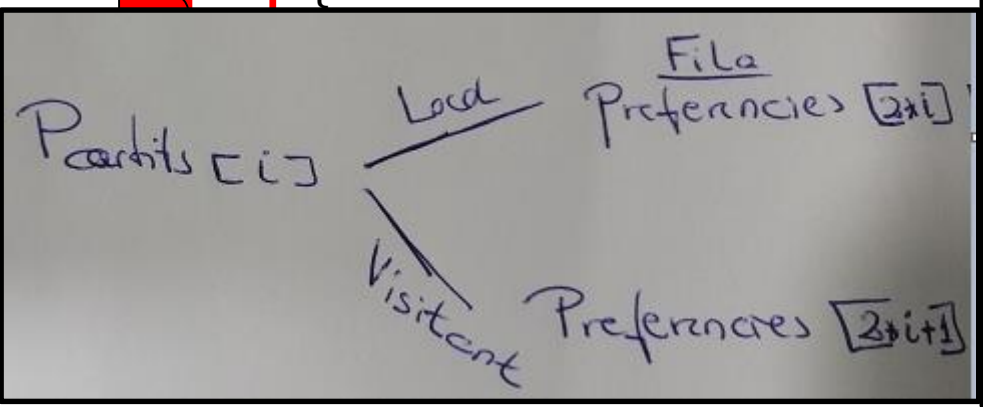
Enunciat indica COM s'emmagatzem les dades en les preferències

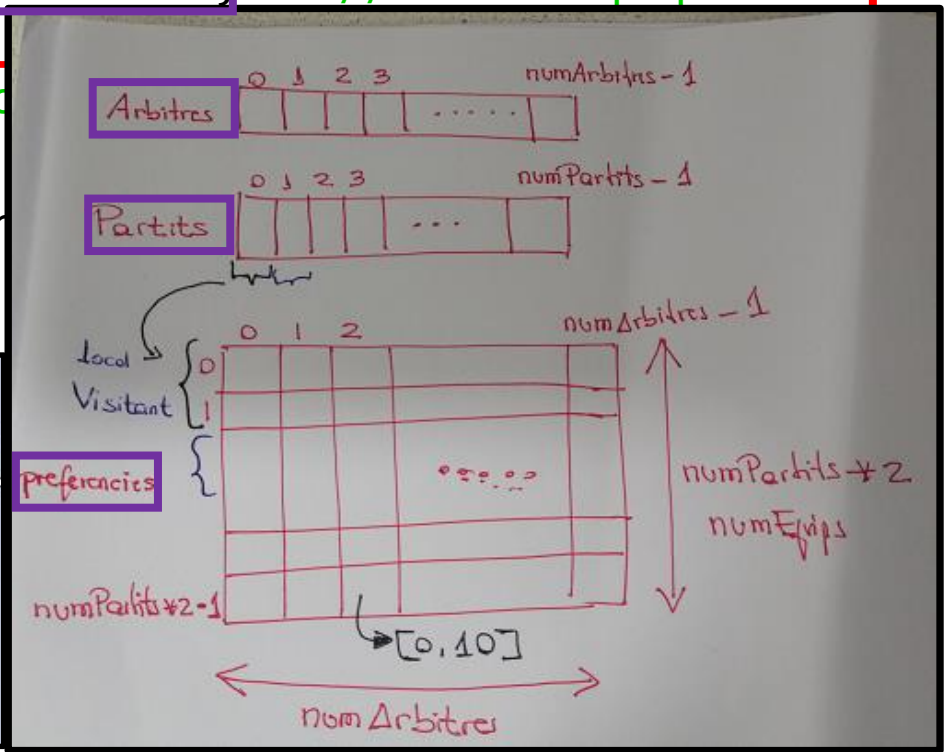
Entrada de dades

Tècnica de backtracking

Solució Arbitres

```
public class JornadaFutbol{
    private int numJornada;
    private Arbitre []Arbitres;
    private int numArbitres; //dim real de la taula prèvia
    private Partit []Partits;
    private int numPartits; //dim real de la taula prèvia
    private    int    [][]preferencies;    //files→equips  -
columnes→àrbitres
    private class Parella{ //c
        Arbitre a; Partit p;
        public Parella(Arbitr
        {this.a=a; this.p=p;
    }
```





Tècnica de backtracking

Assignació Arbitres

Entrada de dades

```
private void omplenaArbitres(){
/*omplena la taula Arbitres amb els arbitres
   disponibles*/
}
public JornadaFutbol(int numA, int numP) throws
IllegalArgumentException{
   if (numA<=0 || numP<=0 || numA<numP)
                throw new Exception("Incorrecte");
   numArbitres=numA; numPartits=numP;
   preferencies= new int[numP*2][numA];
   omplenarPreferencies();
   Partits = new Partit[numP];omplenaPartits();
   Arbitres=new Arbitre[numA];omplenaArbitres();
   MillorSolucio = new Parella[numP];
   for (int i=0; i<numP; i++)MillorSolucio[i]=null;
   costMillor= -1;
} // fi metode
```

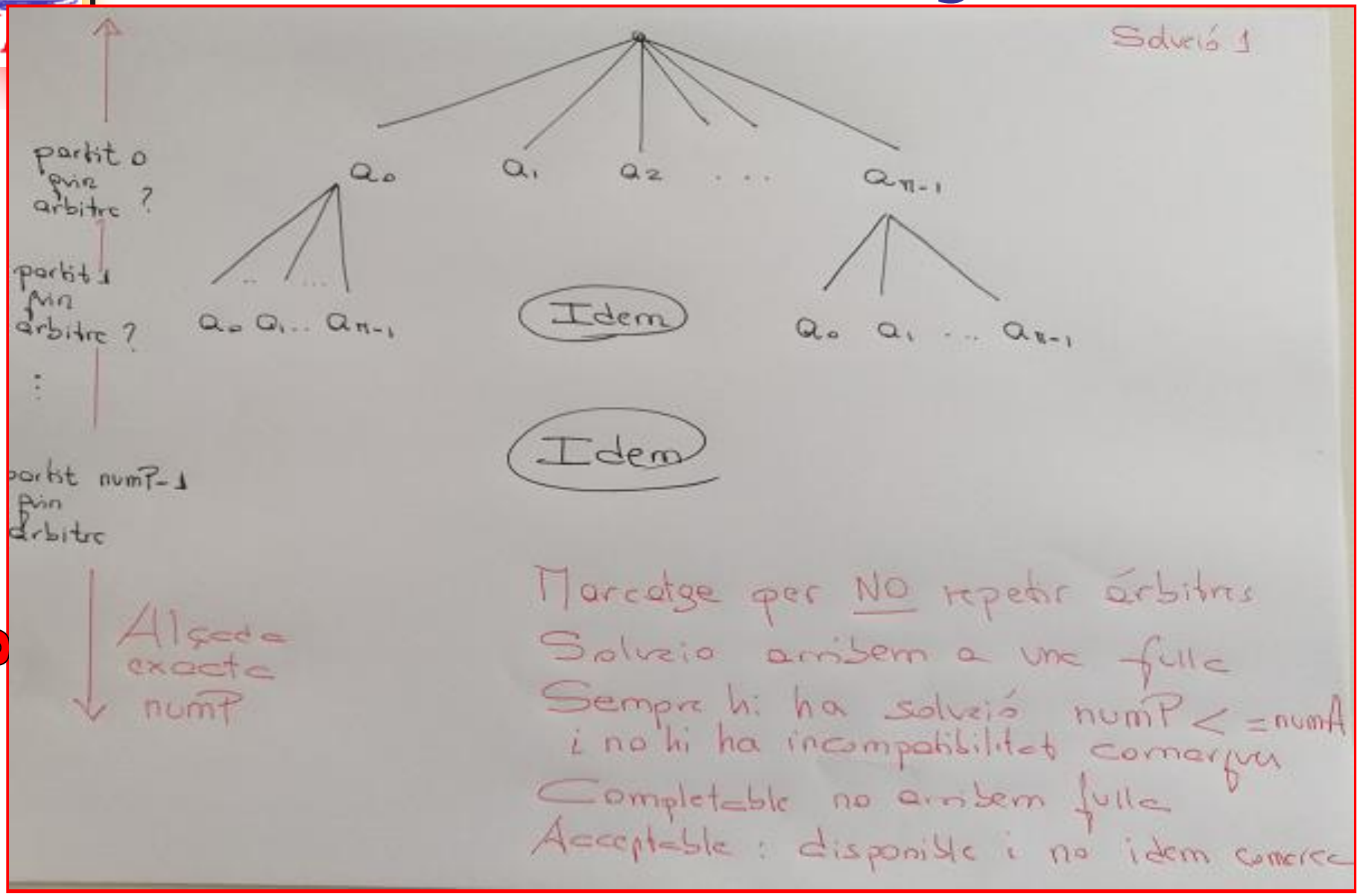
Constructor

Tècnica de backtracking

```
   public static void main(String args[]){ /*sentències*/ }
   public ????? assignarArbitres(??????){
              /*sentències backtracking*/
   }
   private boolean esSolucio(????){ /*sentències*/ }
   private boolean millorSolucio(????){ /*sentències*/ }
   public String toString(){ /* sentències*/ }
} // fi classe
```

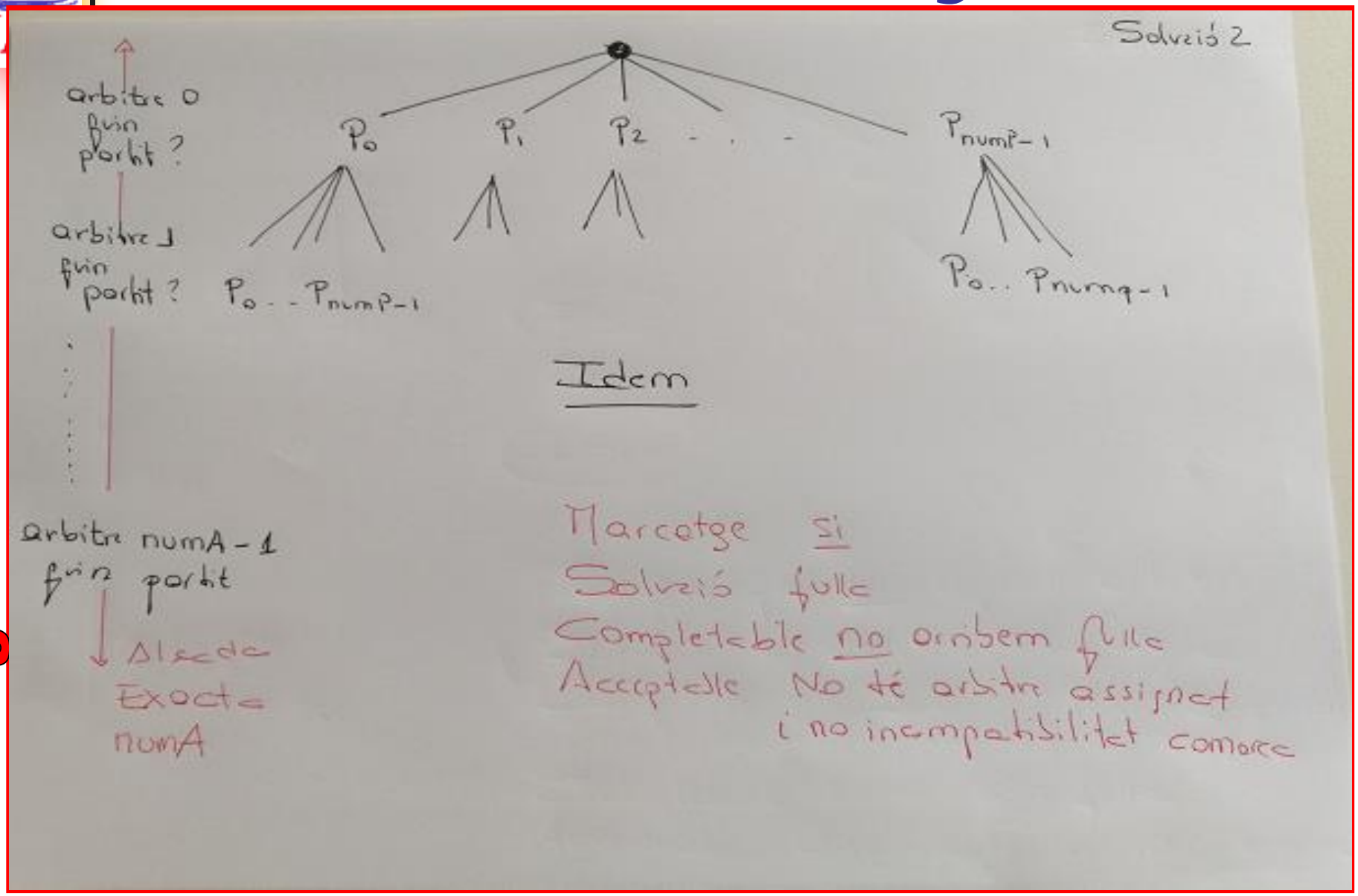
Assignació Arbitres

Tècnica de backtracking - Anàlisi



Assignació Arbitres

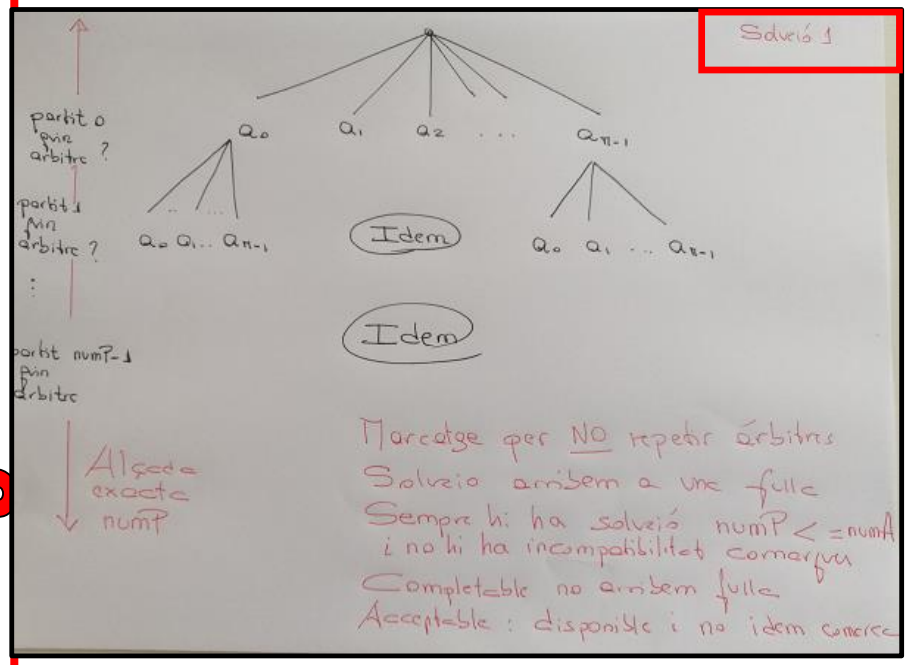
Tècnica de backtracking - Anàlisi

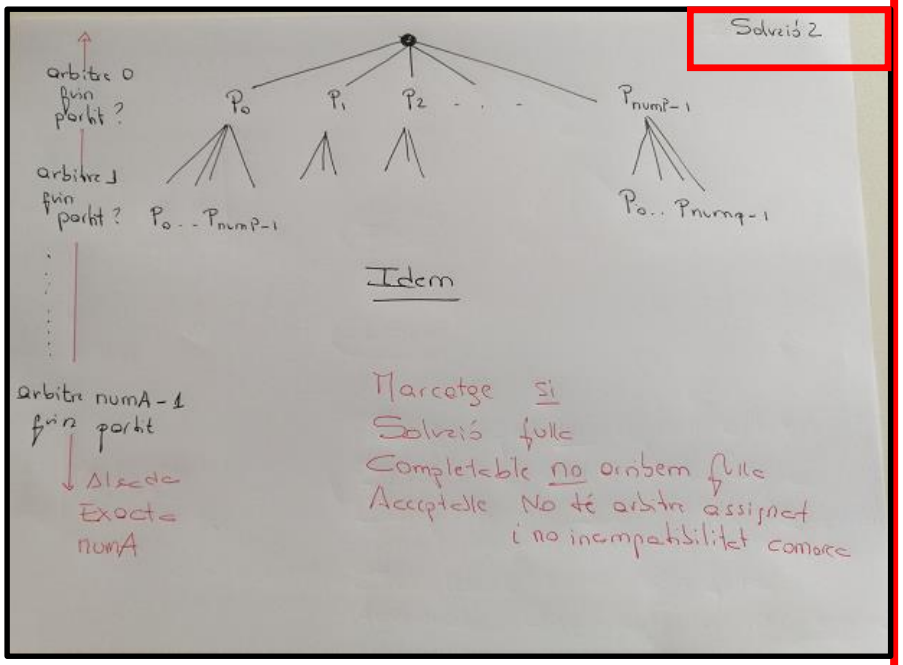


Tècnica de backtracking - Anàlisi

Assignació Arbitres

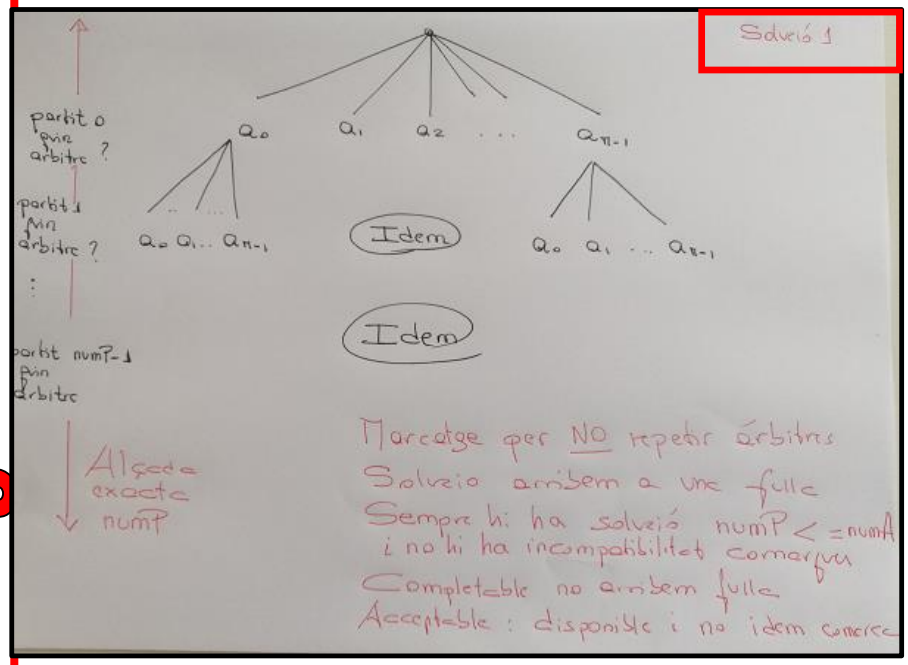
- Quin dels dos plantejaments us sembla millor?

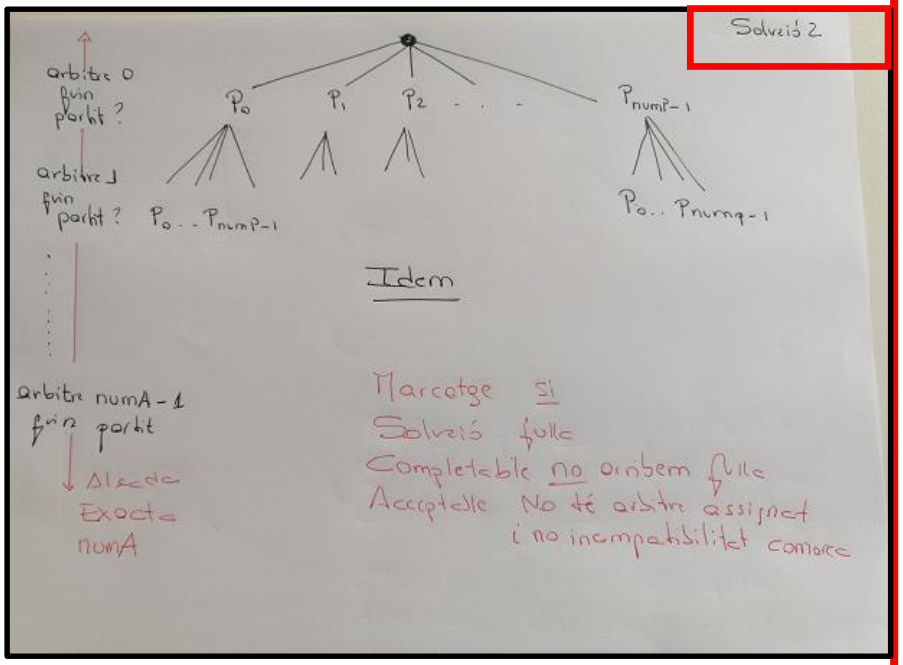




Tècnica de backtracking - Anàlisi

- Quin dels dos plantejaments us sembla millor?

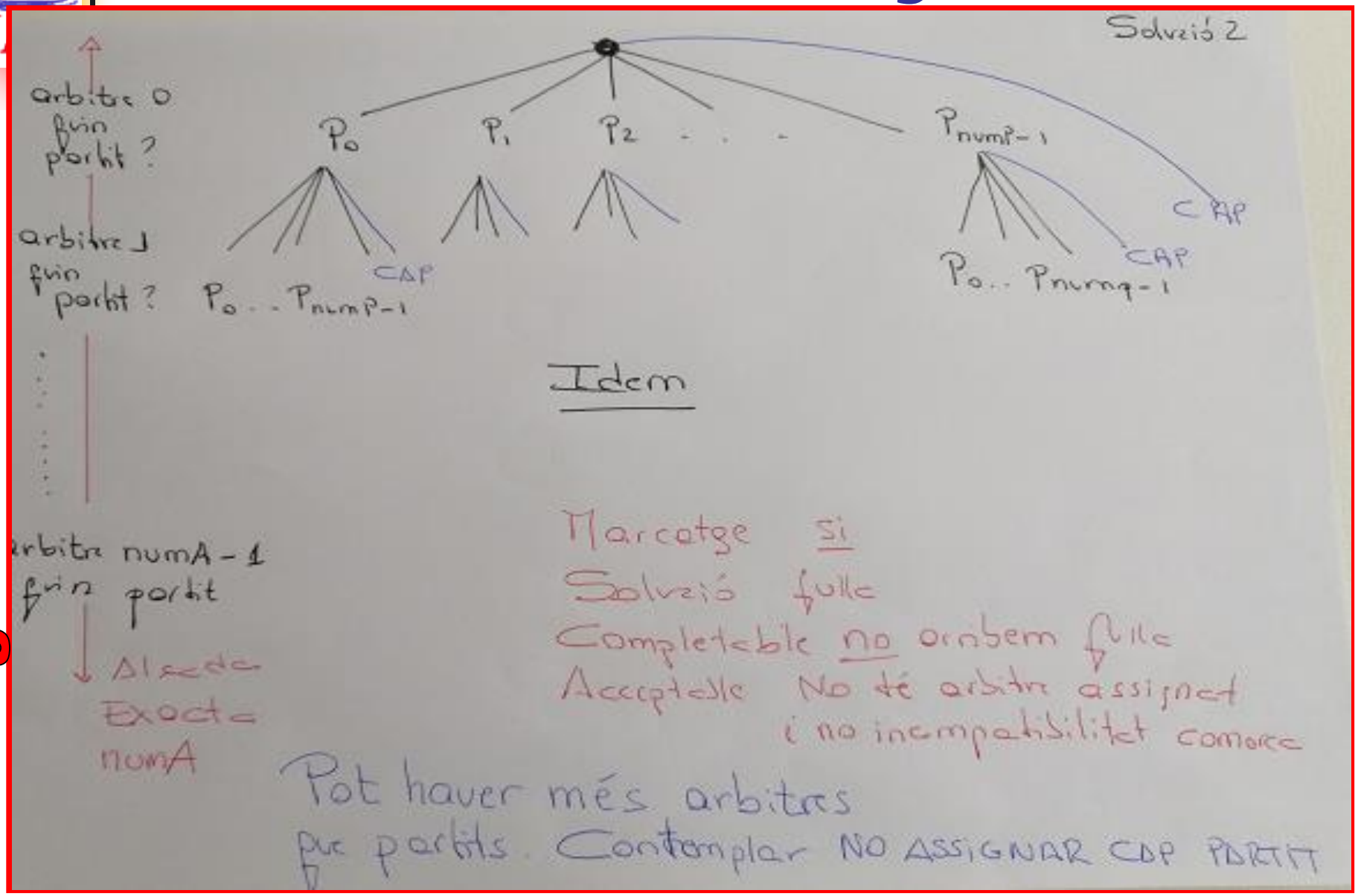




Solució 1

Assignació Arbitres

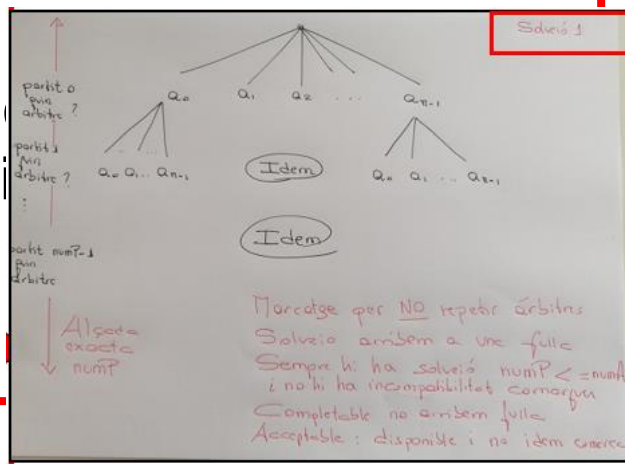
Tècnica de backtracking – Solució 2



Tècnica de backtracking – Solució 1

Assignació Arbitres

- **Backtracking?:** **és** un problema d'optimització, s'ha de trobar la millor distribució d'arbitre/partit maximitzant les preferències, cal aplicar l'esquema de trobar la millor solució.
- **Decisió:** a cada partit quin àrbitre li assigno?.
- **Acceptable**: Si l'àrbitre està lliure i no és de la mateixa comarca que cap dels dos equips que s'enfronten.
- **Arbre:** **L'Amplada** nombre d'àrbitre. **L'Alçada** de l'arbre és exacta, número de partits.
- **Solució** quant tots els partits tinguin àrbitre assignat, és a dir quan arribem a una fulla de l'arbre, i **completable** mentre no hi arribem.
- **Sempre trobarem solució**, si el nombre
  al nombre de partits i no hi ha incompatibili
- **Usarem marcatge**, un àrbitre només pot
  cada jornada futbolística.



Tècnica de backtracking

Assignació Arbitres

Amb el espai de cerca del teu plantejament quins atributs vols afegir a la classe? Es vol **minimitzar** el número de paràmetres del mètode del backtracking:

```
marcats=new boolean[numArbitres];
solucio=new Parella[numPartits];
for (int i=0; i<marcats.length; i++) marcats[i]=false;
for (int i=0; i<numPartits; i++) solucio[i]=null;
cost=0;
```

Afegim

```
private class Parella{ //classe privada
      Arbitre a; Partit p;
      public Parella(Arbitre a, Partit p)
       {this.a=a; this.p=p;}
}
private Parella MillorSolucio[]; private int costMillor;
```

enunciat

Tècnica de backtracking

Assignació Arbitres

- Implementeu el mètode *main*, ha d'iniciar el procés d'assignació d'àrbitres cridant al mètode que farà el backtracking assignarArbitres(?????). Un cop cridat al mètode, s'ha de visualitzar per pantalla la solució trobada. Si no hi ha solució també cal indicar-ho!

El main està ubicat dins de la classe ***JornadaFutbol***, en conseqüència podrà accedir a tot el contingut de la classe

Tècnica de backtracking

Assignació Arbitres

```
public static void main(String args[]){
    System.out.println("Quants arbitres hi ha ?");
    int m = Keyboard.readInt();
    System.out.println("Quants equips participen ?");
    int n = Keyboard.readInt();
    JornadaFutbol A = new JornadaFutbol(m, n/2);
    A.assignarArbitres(0);
    if (A.costMillor != -1) {
         System.out.println("L'assignació òptima ha resultat ser:" );
         for (int i=0; i<n/2; i++)
             System.out.println(A.MillorSolucio[i].p.toString()
                          +  A.MillorSolucio[i].a.toString());
        System.out.println("amb un cost de " + A.costMillor);
    }
    else System.out.println("no hi ha solució);
}
```

Tècnica de backtracking

Assignació Arbitres

- Implementeu el mètode esSolucio(???) que ha de retornar una expressió de *true* si la proposta de solució és solució al problema a resoldre i *false* en cas contrari.

```
public boolean esSolucio(int k){
        return (k==numPartits-1);
//assignat àrbitre a tots el partits!
}
```

- Implementeu el mètode millorSolucio(????) que ha de retornar un valor de true si la millor solució trobada fins aleshores és pitjor a una altra solució.

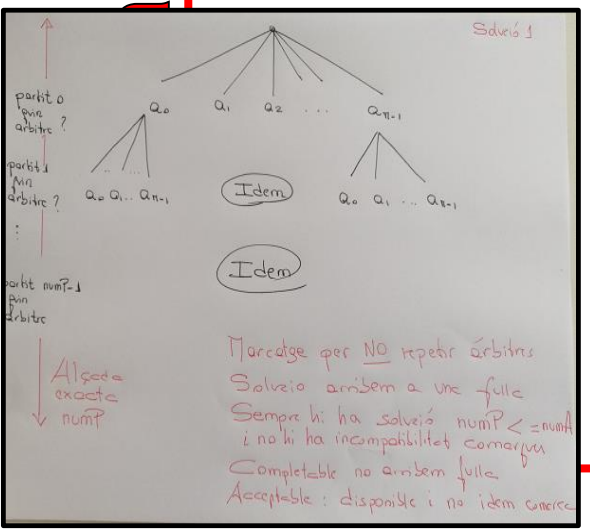
```
public boolean millorSolucio(){
        return (cost>costMillor);
}
```

Tècnica de backtracking

Assignació Arbitres

```
public void assignarArbitres(int k){
   for (int i=0; i<numArbitres; i++){
        if (!marcats[i] && acceptable(k,i)){
            solucio[k]=new Parella(Arbitres[i],Partits[k]);
            marcats[i]=true;
            cost=cost+preferencies[k*2][i]+
                                preferencies[k*2+1][i];
            if (esSolucio(k)){    //recordem com estan les
                  if (millorSolucio()){    //preferències
                        //trobada millor solució
                        costMillor=cost;
                        for (int j=0; j<=k; j++)
                              MillorSolucio[j]= solucio[j];
                  } // fi if millor
            } // fi es solucio
            else  assignarArbitres(k+1);
```

k→partits
i→àrbitres



Tècnica de backtracking

Assignació Arbitres

```
        // desfem valor
        marcats[i]=false;
        cost=cost-preferencies[k*2][i]-
                                preferencies[k*2+1][i];
        solucio[k]=null;
      } // acceptable
    } // for
} //procediment
private boolean acceptable(int k, int j){
          // no de la mateixa comarca
        return (Partits[k].getLocal().getComarca().
              equals(Arbitres[j].getComarca())==false &&
              Partits[k].getVisitant().getComarca().
              equals(Arbitres[j].getComarca()==false);
}
```

k→partit
j→àrbitre