

## 6.- TÈCNIQUES ESPECÍFIQUES DE DISSENY (ESQUEMES).

### Introducció a l'ús d'esquemes:

La idea d'esquema no resulta del tot desconeguda doncs ja fou vista a l'assignatura CP (Programació) quan al resoldre el problema; Donada una frase acabada amb un punt comptar quantes vegades conté la lletra A:

```

com:=0
iniciar_lectura
llegir(car)
fer car≠'.' → cas car='A' → com:=com+1
                [] car≠'A' → res
                fcas
                llegir(car)

ffer
escriu(com)

```

S'en treia el següent model per al recorregut de seqüències:

```

iniciar_tractament
obtenir_el_primer
fer ¬final → tractar_l'element
                obtenir_el_següent

ffer
acabar_tractament

```

De la solució del problema; Donada una frase acabada amb un punt cercar-ne la primera vocal, que podríem escriure:

```

iniciar_lectura
llegir(car)
fer (car≠'.') ∧ (¬vocal) → llegir(car) ffer
cas vocal → escriu(car)
[] ¬vocal → escriu('no hi ha vocals')
fcas

func vocal(c:car) ret (v:booleà) és
ret ((c='A') ∨ (c='E') ∨ (c='I') ∨ (c='O') ∨ (c='U'))
ffunc

```

S'en treia el següent model per a la recerca en seqüències:

```

obtenir_el_primer
fer (¬final) ∧ (¬trobat) → obtenir_el_següent ffer
cas trobat → tractament_d'èxit
[] ¬trobat → tractament_de_fracàs
fcas

```

Aquests dos esquemes són la base de la major part, per no dir de tots, els exercicis d'aquella assignatura. Per poder-los aplicar el primer que cal és reconèixer el problema com pertanyent al tipus (recorregut o recerca) i en segon lloc fer una aplicació correcta de l'esquema. Per tenir èxit en aquesta tasca caldrà tenir doncs molta cura en aquesta segona fase.

Aquesta part de l'assignatura introdueix nous esquemes aplicables a diversos tipus de problemes. En tots els casos caldrà justificar, en primer lloc, que l'esquema és aplicable al problema. En segon lloc caldrà definir amb precisió l'estructura de dades amb que es representen diferents elements del problema. I per últim caldrà identificar de forma completa els elements de l'esquema amb els del problema.

Així per comptar les As cal començar per raonar que la solució es pot trobar fent un recorregut total de la seqüència de caràcters que formen la frase. En segon lloc detallaríem les dades a usar: Un fitxer de caràcters (que conté un punt al final) i un natural per fer de comptador d'As. Per últim identificarem esquema amb problema:

Iniciar_tractament	com:=0
Obtenir_el_primer	iniciar_lectura; llegir(car)
final	car=''
tractar_l'element	<u>cas</u> car='A' → com:=com+1
	[] car≠'A' → res
	<u>fcas</u>
Obtenir_el_següent	llegir(car)
Acabar_tractament	escriu(com)

De cara a l'examen és molt important detallar aquestes tres fases, escriure l'algorisme que en resulta i intentar, encara que sigui informalment, justificar-ne la correctesa.

### Bibliografia:

- Data Structures and Algorithms/ A.V.Aho, J.E.Hopcroft i J.D.Ullman: Addison-Wesley, Reading (Mass.USA) 1983
- Algoritmos + Estructuras de datos = Programas/ N.Wirth: Castillo, Madrid 1980
- Fundamentals of Computer Algorithms/ Horowith i Sahni: Pitman
- Algorítmica: Concepción y Análisis/ G.Brassard i P.Bratley: Masson, Barcelona 1990

**Traducció a Eiffel** de les estructures de la notació algorítmica utilitzada: per a l'alternativa:

<u>cas</u> <condició> → <sentència>	<b>if</b> <condició> <b>then</b> <sentència>
[] ¬<condició> → res	
<u>fcas</u>	<b>end</b> --if
<u>cas</u> <condició> → <sentència a>	<b>if</b> <condició> <b>then</b> <sentència a>
[] ¬<condició> → <sentència b>	<b>else</b> <sentència b>
<u>fcas</u>	<b>end</b> --if
<u>cas</u> <condició a> → <sentència a>	<b>if</b> <condició a> <b>then</b> <sentència a>
[] <condició b> → <sentència b>	<b>elseif</b> <condició b> <b>then</b> <sentència b>
[] <condició c> → <sentència b>	<b>else</b> <sentència b>
<u>fcas</u>	<b>end</b> --if

En aquest darrer cas s'ha de complir: **precondició**  $\hat{U}$  ¬<condició a>  $\hat{U}$  ¬<condició b> **P** <condició c> perquè la sentència cas no avorti.

L'estructura iterativa seria:

inicialització	<b>from</b> inicialització
<u>fer</u> <condició> → <sentències>	<b>until</b> ¬<condició>
	<b>loop</b> <sentències>
<u>ffer</u>	<b>end</b> --loop

Tant acció com funció i var hauran de traduir-se **feature** (el cas de var haurà de traduir-se per **local** a les accions i funcions).

## 6.3.- MÈTODE D'ASSAIG I ERROR (BACKTRACKING).

- Tipus de problemes on és aplicable:
  - Mètode a aplicar a problemes d'heurística.
  - Destinat a obtenir solucions eficients a problemes complexos (complexitat elevada).
  - Quan la solució té forma de tupla de valors  $(x_1, \dots, x_n)$  ( $\forall i: x_i \in S_i$ ), o es pot expressar en aquesta forma.
  - Quan es disposa d'algun criteri que permet determinar quan una tupla és solució.
  
- La solució "força bruta" (sense el mètode)  $\Rightarrow$  Generar totes les solucions i destriar les que compleixen el criteri en qüestió  $\Rightarrow$  Cost exponencial en  $n$  ( $\prod i:1 \leq i \leq n: S_i$ ).

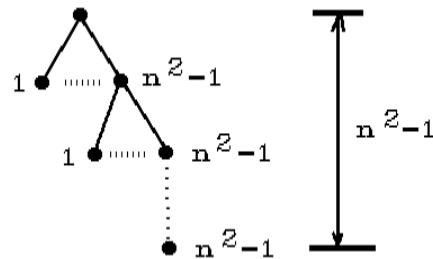
### El salt del cavall:

Donat un tauler de  $n \times n$  ( $n^2$  quadres,  $n \geq 3$ ) i un cavall, que es mou segons les regles dels escacs, situat al quadre  $x_0, y_0$ , ha de recórrer els  $n^2 - 1$  quadres restants, visitant-los una sola vegada amb  $n^2 - 1$  moviments.

{ tupla = combinacions ordenades de  $n^2 - 1$  quadres }

### Solució "força bruta":

Construir totes les tuples possibles amb els  $n^2 - 1$  quadres que cal recórrer, esbrinar si es poden recórrer amb els salts del cavall, determinant els que són solució. Complexitat =  $(n^2 - 1)^{n^2 - 1}$



### Solució "assaig i error":

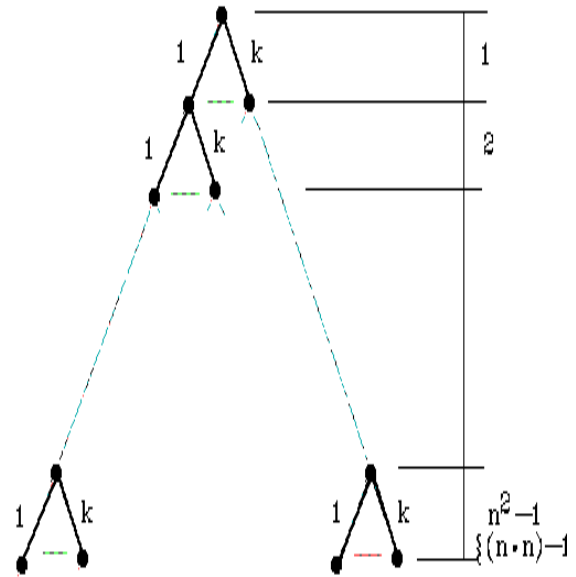
Des d'un quadre  $x_i, y_i$  pot accedir a un màxim de 8 quadres, depenent dels límits del tauler.

	8		1	
7				2
		♞		
6				3
	5		4	

- El mètode consisteix a anar construint la tupla solució a passos per un sistema "d'assaig i error" = Estructurar les possibles tuples en forma d'arbre; Les solucions s'obtenen per recorregut parcial d'aquest arbre.

Aquest arbre és bàsic tant per determinar la complexitat de l'algorisme (nombre de nusos) com per fer la identificació amb l'esquema. D'aquest arbre s'en diu l'**arbre de recerca**.

- L'eficiència millora en funció del mètode que ens permeti determinar a partir de quin moment una tupla no pot ser solució (quin nus pertany a l'arbre).



A més d'aquestes possibilitats cal suprimir aquelles en que el quadre ja ha estat visitat, amb el que quedarà l'arbre de recerca: Amb  $1 \leq k \leq n-1$   $k^{n^2-1} < 8^{n^2-1} = 2^{3 * (n^2-1)}$  nusos possibles

- Requisites:
  - Solució en forma de tupla de dades homogènies.
  - Existència d'un criteri de determinació tuples solució.
  - Existència d'un criteri per a eliminar tuples quan es preveu que no podran ser solució, abans de completar-les ( $\neg$ completable).
- En aquests problemes la definició de l'estructura de les dades ha d'incloure una definició precisa de l'arbre de recerca i de la representació de la tupla solució, així com de les decisions que es prenen.
- Primer esquema: Genera totes les solucions (cas d'haver-n'hi més d'una), suposant que s'arranca en condicions d'obtenir el primer i que l'últim du la informació de que ho és:

```

a) Recursiu: És una immersió amb la crida inicial assaja(1);
    acció assaja(k:nat) és [1]
        prepara_recorregut { queda en condicions d'obtenir el primer }
        fer  $\exists$ _succ  $\rightarrow$  següent_candidat
            cas solució  $\rightarrow$  tracta_solució
            []  $\neg$ solució  $\rightarrow$  res
            fcas
            cas completable  $\rightarrow$  assaja(k+1)
            []  $\neg$ completable  $\rightarrow$  res
            fcas
        ffer
    facció
    
```

b) Iteratiu:  $k:=1$  [2]  
prepara\_recorregut  
fer  $k>0 \rightarrow$  cas  $\exists\_succ \rightarrow$  següent\_candidat  
cas solució  $\rightarrow$  tracta\_solució  
 $[\ ] \neg solució \rightarrow$  res  
fcas  
cas completable  $\rightarrow$   $k:=k+1$   
prepara\_recorregut  
 $[\ ] \neg completable \rightarrow$  res  
fcas  
 $[\ ] \neg \exists\_succ \rightarrow$   $k:=k-1$   
fcas  
ffer

L'arbre de recerca és bàsic en dos aspectes ja que d'una banda dóna idea de la màxima complexitat de l'algorisme, el cost serà funció del nombre de nusos de l'arbre que es visitin. Per l'altre banda és la base de la identificació, ja que l'algorisme no fa altre cosa que recorre aquest arbre.

**El problema de les vuit reines:**

Col·locar vuit reines en un tauler d'escacs de manera que no es facin escac entre elles. Solució:

Fases:

- Justificació, estructura de dades i arbre de recerca
- Identificació i escriptura de l'algorisme
- Raonament de correctesa

Compliment dels requisits:

- a) Solució expressable en forma de tupla de dades homogènies, ja que a la solució cada reina estarà en una línia diferent  $\Rightarrow$  poden representar-se amb un vector de vuit elements on el subíndex és el número de la fila que ocupa i el valor la columna.
- b) Es pot reconèixer una solució ja que no hi pot haver dues reines a la mateixa columna ni repetir les diagonals (la representació de la solució impedeix que hi hagi dues reines a la mateixa fila).
- c) Es poden detectar les tuples no completables (no podem col·locar la reina següent).

Si només es consideren les columnes que no ocupen les reines anteriors llavors a la segona reina li queden 7 llocs, a la tercera 6, a la quarta 5, etc. ... i a la vuitena i darrera només 1.

Si es considera a més que també es poden suprimir les diagonals llavors la 2ª reina té 5 llocs, la 3ª ...

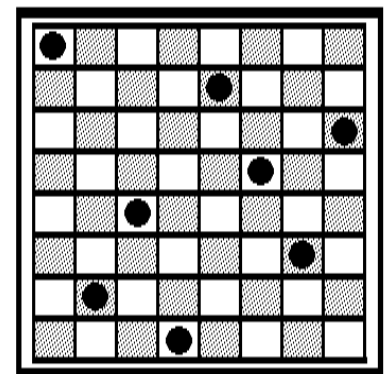
D'aquí surten dues interpretacions de l'esquema:

- a) La iteració principal va de posició en posició.
- b) La iteració principal va de posició lliure en posició lliure.

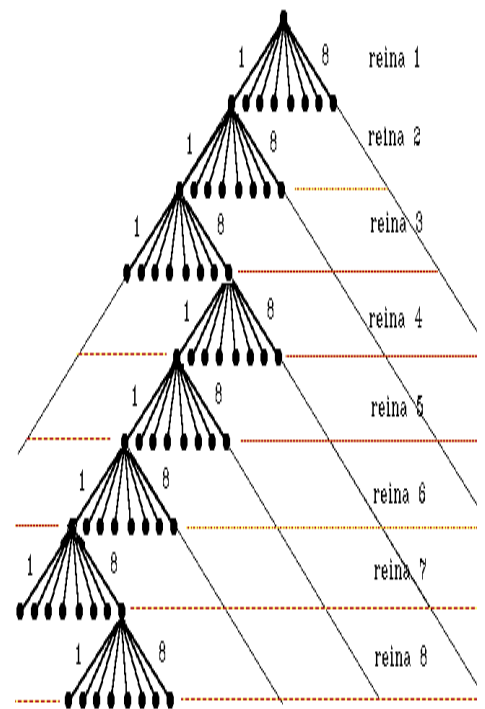
**Versió a) Identificació:**

<u>prepara_recorregut</u>	$r(k):=0$
$\exists\_succ$	$r(k)<8$
<u>següent_candidat</u>	$r(k):=r(k)+1$
<u>solució</u>	$(k=8) \wedge \text{lliure}(k,r)$
<u>tracta_solució</u>	<u>imprimir_solució</u>
<u>completable</u>	$(k<8) \wedge \text{lliure}(k,r)$

tipus tauler:taula(1÷8) de nat ftipus



1,5,8,6,3,7,2,4



acció reines(k:nat;var r:tauler) és  
 r(k):=0  
fer r(k)<8 → r(k):=r(k)+1  
     cas (k=8) ∧ lliure(k,r) → imprimir\_solució  
     [] (k<8) ∨ ¬lliure(k,r) → res  
     fcas  
     cas (k<8) ∧ lliure(k,r) → reines(k+1,r)  
     [] (k=8) ∨ ¬lliure(k,r) → res  
     fcas

ffer

facció

Equivalent a:

acció reines(k:nat;var r:tauler) és  
 r(k):=0  
fer r(k)<8 → r(k):=r(k)+1  
     cas lliure(k,r) → cas k=8 → imprimir\_solució  
     [] k<8 → reines(k+1,r)  
     fcas

[] ¬lliure(k,r) →

res

fcas

ffer

facció

Quan volen posar una nova reina (i) només hi ha reines posades (k) més amunt (i>k). Dues reines estan a la mateixa columna si:

$$r(i)=r(k)$$

Dues reines estan a la mateixa diagonal descendent si:

$$r(i)-r(k) = i-k$$

Dues reines estan a la mateixa diagonal ascendent si:

$$r(k)-r(i) = i-k$$

Poden doncs resumir que dues reines es maten en diagonal si:

$$(i-k) = \text{abs}(r(i)-r(k))$$

$$\{(1 \leq i \leq 8) \wedge (\forall j: 1 \leq j \leq i: 1 \leq r(j) \leq 8)\}$$

func lliure(i:nat;r:tauler) ret (ll:booleà) és

var k:nat fvar

k:=1

fer (r(k)≠r(i)) ∧ ((i-k)≠abs(r(i)-r(k))) → k:=k+1 ffer

ret i=k

ffunc

Raonament de correctesa: Només s'ha posat una reina si la posició no està amenaçada ⇒ si es posen vuit reines s'ha assolit l'objectiu.

Iterativa: k:=1

r(k):=0

fer k>0 → cas r(k)<8 → r(k):=r(k)+1  
     cas lliure(k,r) → cas k=8 → imprimir\_solució  
     [] k<8 → k:=k+1  
     r(k):=0

fcas

[] ¬lliure(k,r) → res

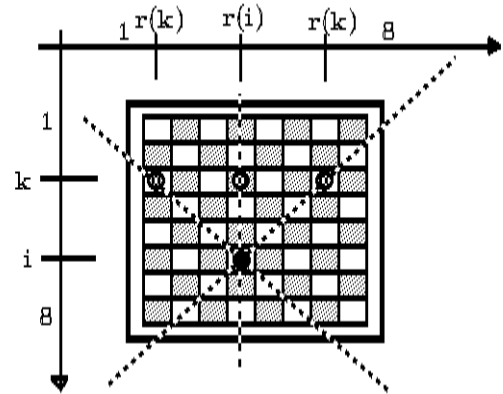
fcas

[] r(k)=8 → k:=k-1

fcas

ffer

Anàlisi quantitativa: 1625 nusos visitats d'un arbre de 69.281, que és el 2,34% de la solució de les permutacions 8!=40.320 i el 0,0096853 de la solució bruta 8<sup>8</sup>=16.777.216





**Versió b:** La iteració principal va de posició lliure en posició lliure. Recordeu la preconditionió de la funció lliure  $\{(1 \leq i \leq 8) \wedge \exists j: 1 \leq j \leq i: 1 \leq r(j) \leq 8\}$

acció reines\_3a(k:nat;var r:tauler) és [3]

```

r(k):=1
fer (r(k)≤8) ∧c ¬lliure(k,r) → r(k):=r(k)+1 ffer
fer r(k)≤8 → cas k=8 → impr_solució |
                [] k<8 → res | cas k=8 → impr_solució
                fcas | [] k<8 → reines_3a(k+1,r)
                cas k<8 → reines_3a(k+1,r) | fcas
                [] k=8 → res |
                fcas |
                r(k):=r(k)+1 |
                fer (r(k)≤8) ∧c ¬lliure(k,r) → r(k):=r(k)+1 ffer

```

Nota: Es podria suprimir els  $\wedge_c$  definint una nova funció *lliure* de forma que per  $r(k)=9$  retornés sempre *fals* sense avortar. Ara, tal com l'hem definida, podria retornar *cert*. De fet, aquest  $\wedge_c$  no deixa de ser un purisme.

ffer

facció

- Cal reduir les comparances que són redundants.
- Cal transformar-lo a iteratiu.
- Provar els models [1] i [2].

**Marcatges:**

En ocasions pot millorar-se la determinació d'una solució si es procedeix a un marcatge (i desmarcatge) de la tupla en formació. Com solen ser costosos l'ús de marcatges es limitarà als casos en que es produeixi una millora real o en què siguin imprescindibles. Apareix una nova funció **acceptable**, fruit del marcatge, que permet desestimar un candidat *a priori*, equival a (*solució*  $\hat{U}$  *completable*). Vol dir que:

*acceptable*  $\hat{U}$  ¬*solució* **P** *completable*  
*acceptable*  $\hat{U}$  ¬*completable* **P** *solució*

Esquemes per solucions amb marcatges: al ser una immersió, cal la primera crida; *init\_marcatge; assaja(1)* [5]

acció assaja(k:nat) és [5]

```

prepara_recorregut
fer ∃_succ → següent_candidat
                cas acceptable → enregistra_candidat
                cas solució → tracta_solució
                [] ¬solució → res
                fcas
                cas completable → assaja(k+1)
                [] ¬completable → res
                fcas
                desenregistra_candidat
                [] ¬acceptable → res

```

ffer

facció



Aplicació al problema de les 8 reines:

Una alternativa a la funció lliure és marcar tots els quadres que mata una reina, vol dir columnes i diagonals, en tal cas amb l'esquema [1]:

acció reines(k:nat) és [5]

r(k):=0

fer r(k)<8 → r(k):=r(k)+1

cas lliure(k,r) → posa\_reina(k)

cas k=8 → impr\_tauler

[] k<8 → reines(k+1)

fcas

treu\_reina(k)

[] ¬lliure(k,r) → res

fcas

ffer

facció

- La mateixa columna | (col):

$$r(i)=r(k)$$

- La mateixa diagonal ascendent / (da):  $(k-i=r(i)-r(k)) \Rightarrow r(k)+k=r(i)+i$

- La mateixa diagonal descendent \ (dd):  $(k-i=r(k)-r(i)) \Rightarrow r(k)-k=r(i)-i$

Tres taules de booleans:

col:taula(1÷8) de booleà

da:taula(2÷16) de booleà

dd:taula(-7÷7) de booleà

posar\_reina: (col(r(k)),da(r(k)+k),dd(r(k)-k)):= (fals,fals,fals)

treu\_reina : (col(r(k)),da(r(k)+k),dd(r(k)-k)):= (cert,cert,cert)

lliure: col(r(k)) ∧ da(r(k)+k) ∧ dd(r(k)-k)

acció init\_marcatge és

var i:enter fvar

i:=1; fer i≤8 → col(i):=cert;i:=i+1 ffer

i:=2; fer i≤16 → da(i):=cert;i:=i+1 ffer

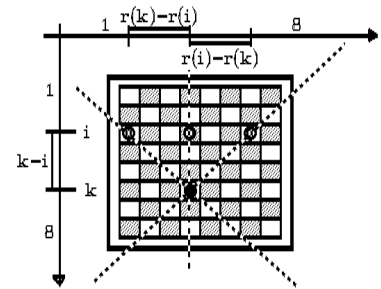
i:=-7; fer i≤7 → dd(i):=cert;i:=i+1 ffer

facció

■ Transformar a iteratiu.

■ Provar les altres alternatives.

■ La solució de marcar una taula de (1,8,1,8) és molt més complicada al moment de marcar i desmarcar, ja que cal saber quantes reines maten un quadre determinat, vol dir usar una taula de naturals i comptar-les.



Esquemes per a generar una única solució:

La primera crida: èxit:=fals;assaja(1)

acció assaja(k:nat) és [9]

prepara\_recorregut

fer ¬èxit ∧ ∃\_succ → següent\_candidat

cas solució → èxit:=cert

tracta\_solució

[] ¬solució → cas completable → assaja(k+1)

[] ¬completable → res

fcas

fcas

ffer

facció

Iteratiu:

$k:=1$  [10]

$\text{\u00e8xit}:=\text{fals}$

$\text{prepara\_recorregut}$

$\text{fer } \neg \text{\u00e8xit} \wedge k > 0 \rightarrow \text{cas } \exists \_ \text{succ} \rightarrow \text{seg\u00fcent\_candidat}$   
 $\text{cas soluci\u00f3} \rightarrow \text{\u00e8xit}:=\text{cert}$   
 $\text{tracta\_soluci\u00f3}$   
 $[\ ] \neg \text{soluci\u00f3} \rightarrow \text{cas completable} \rightarrow k:=k+1$   
 $\text{prepara\_recorregut}$   
 $[\ ] \neg \text{completable} \rightarrow \text{res}$   
 $\text{fcas}$

$[\ ] \neg \exists \_ \text{succ} \rightarrow \text{fcas}$   
 $k:=k-1$

$\text{ffer}$

Per l'altre tipus de m\u00e0quina:

$\text{acci\u00f3 assaja}(k:\text{nat}; \text{var } \text{\u00e8xit}:\text{boolea}) \text{ \u00e9s}$  [11]

$\text{prepara\_recorregut\_2a}$

$\text{fer } \neg \text{\u00e8xit} \wedge \text{hi\_ha\_candidat} \rightarrow \text{cas soluci\u00f3} \rightarrow \text{\u00e8xit}:=\text{cert}$   
 $\text{tracta\_soluci\u00f3}$   
 $[\ ] \neg \text{soluci\u00f3} \rightarrow \text{cas completable} \rightarrow \text{assaja}(k+1, \text{\u00e8xit})$   
 $[\ ] \neg \text{completable} \rightarrow \text{res}$   
 $\text{fcas}$   
 $\text{cas } \neg \text{\u00e8xit} \rightarrow \text{seg\u00fcent\_candidat}$   
 $[\ ] \text{\u00e8xit} \rightarrow \text{res}$   
 $\text{fcas}$

$\text{fcas}$

$\text{ffer}$

$\text{facci\u00f3}$

Iteratiu:

$k:=1$  [12]

$\text{\u00e8xit}:=\text{fals}$

$\text{prepara\_recorregut\_2a}$

$\text{fer } \neg \text{\u00e8xit} \wedge k > 0 \rightarrow \text{cas hi\_ha\_candidat} \rightarrow \text{cas soluci\u00f3} \rightarrow \text{\u00e8xit}:=\text{cert}$   
 $\text{tracta\_soluci\u00f3}$   
 $[\ ] \neg \text{soluci\u00f3} \rightarrow \text{cas completable} \rightarrow$   
 $k:=k+1$   
 $\text{prepara\_recorregut\_2a}$   
 $[\ ] \neg \text{completable} \rightarrow$   
 $\text{seg\u00fcent\_candidat}$   
 $\text{fcas}$

$[\ ] \neg \text{hi\_ha\_candidat} \rightarrow \text{fcas}$   
 $k:=k-1$   
 $\text{cas } k > 0 \rightarrow \text{seg\u00fcent\_candidat}$   
 $[\ ] k \leq 0 \rightarrow \text{res}$   
 $\text{fcas}$

$\text{fcas}$

$\text{ffer}$

Amb marcatges: { recordant *acceptable*  $\bar{U} \neg$ solució  $\bar{P}$  completable }

acció assaja(k:nat;var èxit:booleà) és [13]

prepara\_recorregut

fer  $\neg$ èxit  $\wedge$   $\exists$ \_succ  $\rightarrow$  següent\_candidat

cas acceptable  $\rightarrow$  enregistra\_candidat

cas solució  $\rightarrow$  èxit:=cert  
tracta\_solució

$[]$   $\neg$ solució  $\rightarrow$  assaja(k+1,èxit)  
desenregistra\_candidat

fcas

$[]$   $\neg$ acceptable  $\rightarrow$  res

fcas

ffer

facció

Iteratiu: [14]

init\_marcatge

k:=1

èxit:=fals

prepara\_recorregut

fer  $\neg$ èxit  $\wedge$  k>0  $\rightarrow$  cas  $\exists$ \_succ  $\rightarrow$  següent\_candidat

cas acceptable  $\rightarrow$  enregistra\_candidat

cas solució  $\rightarrow$  èxit:=cert  
tracta\_solució

$[]$   $\neg$ solució  $\rightarrow$  k:=k+1  
prepara\_recorregut

fcas

$[]$   $\neg$ acceptable  $\rightarrow$  res

fcas

$[]$   $\neg \exists$ \_succ  $\rightarrow$  k:=k-1

cas k>0  $\rightarrow$  desenregistra\_candidat

$[]$  k≤0  $\rightarrow$  res

fcas

fcas

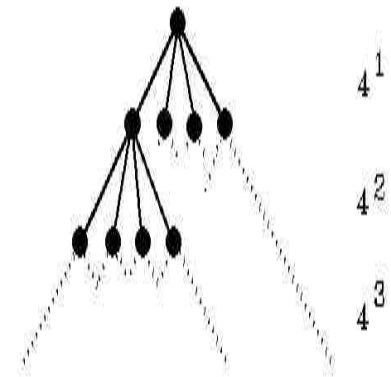
ffer

Exercici: Escriure els casos per l'altre tipus de màquina amb marcatges (corresponents als esquemes d'aquest capítol [7] i [8]).

**Optimització:**

- En alguns casos no es cerquen totes les solucions ni solament la primera, sinó aquella que és òptima en cert sentit (no totes les solucions tenen el mateix valor).
- Es generen totes les solucions, i es reté la millor fins el moment (Si és una funció f qui ens dona el sentit d'optimitat):
  - cas f(solució\_actual)>f(òptim)  $\rightarrow$  òptim:=solució\_actual
  - $[]$  f(solució\_actual)≤f(òptim)  $\rightarrow$  res
- Millorant l'algorisme si algun mecanisme ens permet saber si la solució actual pot ser o no millor que la seleccionada com a valor òptim provisional.  $\Rightarrow$  Aquesta és la idea base del mètode de brancament i poda.

**Exemple 1; Acolorit d'un mapa:** Donat un mapa, acolorir les regions de manera que dues regions adjacents siguin de color diferent, usant exclusivament quatre colors.



- Generarem totes les maneres possibles d'acolorir-lo.
- La solució és expressable en forma de tupla de dades homogènies amb tants components com regions. Cada element té el valor del color assignat.
- El mapa ve donat en forma d'una matriu de contigüïtat.
- La solució bruta són variacions amb repetició de 4 elements presos de  $n$  en  $n$  vol dir que són  $4^n$ , el que vol dir que són  $4^n$  tuples de  $n$  elements, és a dir la complexitat és  $n \times 4^n$ .

L'arbre té  $(\sum_{i=1}^n 4^i) = (4/3)(4^n - 1)$  (ja que  $a + ax + ax^2 + \dots + ax^{n-1} = a(x^n - 1)/(x - 1)$ ) però no es recorren tots els nusos.

$\{k \leq n\}$

acció colora (k:nat) és

$c(k) := 0$

fer  $c(k) < 4 \rightarrow c(k) := c(k) + 1$

cas  $(k = n) \wedge \text{compatible}(k) \rightarrow \text{imprimir\_solució}$

$[] (k < n) \vee \neg \text{compatible}(k) \rightarrow \text{res}$

fcas

cas  $(k < n) \wedge \text{compatible}(k) \rightarrow \text{colora}(k+1)$

$[] (k = n) \vee \neg \text{compatible}(k) \rightarrow \text{res}$

fcas

} (1) ↓  
|  
|  
|  
|

ffer

facció

(1) cas  $\text{compatible}(k) \rightarrow \begin{matrix} \text{cas } k = n \\ [] k < n \\ \text{fcas} \end{matrix} \rightarrow \begin{matrix} \text{imprimir\_solució} \\ \text{colora}(k+1) \end{matrix}$

$[] \neg \text{compatible}(k) \rightarrow \text{res}$

fcas

func compatible(k:nat) ret (c:booleà) és

var h:nat fvar

h:=1

fer  $\neg \text{contigu}(h,k) \vee (c(h) \neq c(k)) \rightarrow h := h + 1$  ffer

ret h=k

ffunc

⇒ Per simplificar s'ha considerat que contigu (la matriu de contigüïtats) és una variable global de definició: taula(1, n, 1, n) de booleà, que tindrà la diagonal principal de valor cert i contigu(i,j)=cert si país(i) és veí del país(j).

- Millor eficiència amb una bona ordenació de les regions. Millor quan més prop de l'arrel podem descartar les solucions ⇒ Països en ordre descendent del nombre connexions amb veïns.

**Exemple 2; La motxilla:**

Donats  $n$  objectes de volums (o de pes)  $v(i)$  i de pes (o de valor)  $p(i)$  i una motxilla de capacitat  $M$  (aprofitable al 100%) (o que suporta un pes màxim  $M$ ) elegir un subconjunt d'aquests objectes tal que es carregui el màxim pes (valor) possible. Equival a que si  $S$  és el subconjunt d'objectes seleccionat i:

$$M \geq (\sum_{i \in S} v(i)) \quad (2)$$

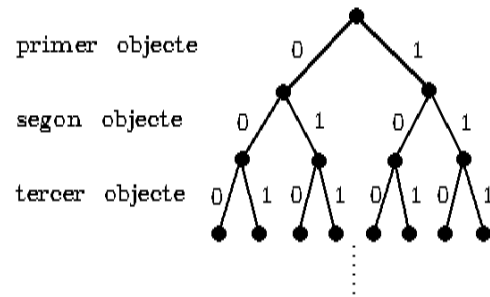
$$P = (\sum_{i \in S} p(i))$$

No existeix un altre  $S$  que complint (2) faci més gran el valor  $P$ , s'escriurà:

$$(\forall S': M \geq (\sum_{i \in S'} v(i)) \wedge (S' \neq S) : (\sum_{i \in S'} p(i)) \leq P)$$

- Es pot representar la solució com a tupla amb dues possibilitats:

- a) Tupla de  $n$  elements de valors que indiquen si l'element pertany o no a la solució amb els valors 0 o 1 (que podrien ser *cert* o *fals*) (0 vol dir que l'element  $i$  no pertany a la solució  $S$ , 1 vol dir que sí hi pertany).
- b) Tupla amb els  $k$  valors dels objectes que formen la solució (els  $k$  subíndex de  $v$  i  $p$ ).



**Versió I** (representació a):

Variables globals:

var t, t\_òptim: taula(1÷n) de -1÷1  
 v, p: taula(1÷n) de real  
 M:nat  
 profit\_òptim:real

fvar

Usarem les funcions:

volum(k) =  $(\sum_{i:1 \leq i \leq k} v(i) \cdot t(i))$   
 profit(k) =  $(\sum_{i:1 \leq i \leq k} p(i) \cdot t(i))$

{Primera crida: profit\_òptim:=0; motxilla(1)}  
 [1]

acció motxilla1(k:nat) és

t(k):=-1

fer t(k)<1 →

t(k):=t(k)+1

cas (k=n) ∧ (volum(k)≤M) →

cas profit(k)>profit\_òptim → t\_òptim:=t  
 profit\_òptim:=profit(k)

[] profit(k)≤profit\_òptim → res

fcas

[] (k<n) ∨ (volum(k)>M) →  
 res

fcas

cas (k<n) ∧ (volum(k)≤M) → motxilla1(k+1)

[] (k≥n) ∨ (volum(k)>M) → res

fcas

ffer

facció

Simplificat: acció motxilla1(k:nat) és

t(k):=-1

fer t(k)<1 → t(k):=t(k)+1

| cas (k=n) ∧ (volum(k)≤M) ∧ (profit(k)>profit\_òptim) →

| t\_òptim:=t

| profit\_òptim:=profit(k)

| [] (k<n) ∨ (volum(k)>M) ∨ (profit(k)≤profit\_òptim) →

| res

| fcas

| cas (k<n) ∧ (volum(k)≤M) → motxilla1(k+1)

| [] (k≥n) ∨ (volum(k)>M) → res

| fcas

ffer

facció

Milliores:

- Usar una immersió o marcatge que eviti el càlcul redundant de  $volum(k)$  i  $profit(k)$  (ús d'acumuladors).
- Evitar comparacions redundants.

Variables globals:

var profit, volum: real fvar

acció motxilla2(k:nat) és

t(k):=-1

fer t(k)<1 →

t(k):=t(k)+1

cas t(k)=1 → profit:=profit+p(k) }  
 volum:=volum+v(k) } profit:=profit+p(k)\*t(k)

[] t(k)≠1 → res } volum:=volum+v(k)\*t(k)

fcas

cas volum≤M → cas k=n → cas profit(k)>profit\_òptim → t\_òptim:=t  
 profit\_òptim:=profit(k)

[] profit(k)≤profit\_òptim → res

fcas

[] k<n → motxilla2(k+1)

fcas

[] volum>M → res

fcas

cas t(k)=1 → profit:=profit-p(k) }  
 volum:=volum-v(k) } profit:=profit-p(k)\*t(k)

[] t(k)≠1 → res } volum:=volum-v(k)\*t(k)

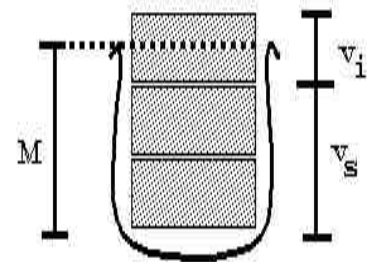
fcas

ffer

facció

És possible obtenir una funció que permeti retallar més l'arbre que simplement amb  $volum \leq M$ .

Introduint el concepte de densitat  $\{p(i)/v(i)\}$  i ordenant els objectes en ordre decreixent d'aquest valor (primer els més densos) podem obtenir fàcilment la cota superior del màxim profit que es pot obtenir expandint un nus amb qualsevol dels seus fills o descendent, que és el profit que es pot obtenir amb el dret a fragmentar el darrer objecte, vol dir el profit ideal en el cas que poguéssim aprofitar els més densos a partir de la situació actual:



func profit\_límit(i:nat;vs,ps:real) ret (l:real) és

fer (vs<M) ∧ (i<n) → i:=i+1

cas vs+v(i)≤M → ps:=ps+p(i)  
 vs:=vs+v(i)

[] vs+v(i)>M → ps:=ps+(M-vs)\*p(i)/v(i)  
 vs:=M

fcas

ffer

ret ps

ffunc

Sembla raonable esperar que siguin els més densos els més probables de pertànyer a la solució, pel que sembla millor seleccionar en ordre invers (els primers es més fàcil que hi siguin), ens cal modificar les definicions de t i t\_òptim:

var t:taula(1÷n) de 0÷2

t\_òptim:taula(1÷n) de 0÷2

fvar

acció motxilla3(k:nat) és

t(k):=2

fer t(k)>0 →

t(k):=t(k)-1

profit:=profit+p(k)\*t(k)

{el marcatge serveix per determinar si és acceptable}

volum:=volum+v(k)\*t(k)

cas (volum≤M) ∧ profit\_límit(k,volum,profit)>profit\_òptim →

cas k=n

→

cas profit>profit\_òptim

→

t\_òptim:=t

profit\_òptim:=profit(k)

[] profit≤profit\_òptim → res

fcas

[] k<n

→

motxilla3(k+1)

fcas

[] (volum>M) ∨ profit\_límit(k,volum,profit)≤profit\_òptim → res

fcas

profit:=profit-p(k)\*t(k)

volum:=volum-v(k)\*t(k)

ffer

facció

- Transformar-ho a iteratiu.

Fins aquí l'esquema. Pot semblar complex encara que no ho és, ja que es tracta simplement de recórrer l'arbre de recerca.