Problema de asignación: resolución mediante backtracking

Problema de asignación

- Existen n personas y n tareas
- Cada persona `i` puede realizar una tarea `j` con más o menos rendimiento: `B[i,j]`
- **Objetivo:** asignar una tarea a cada persona (asignación uno-a-uno), de manera que se maximice la suma de rendimientos.

Personas \ Tareas	1	2	3
1	4	9	1
2	7	2	3
3	6	3	5

Ejemplo 1. (P1, T1), (P2, T3), (P3, T2)

B_{TOTAL}= 4+3+3= 10

Ejemplo 2. (P1, T2), (P2, T1), (P3, T3)

B_{TOTAL}= 9+7+5= 21

Problema de asignación

Datos del problema:

- **n**: número de personas y de tareas disponibles.
- **B**: matriz nxn de enteros (beneficio).
  - `B[i,j]`=beneficio de asignar a la persona `i` la tarea `j`.

Resultado:

- **Realizar n asignaciones:** $\{(p_1,t_1),\ (p_2,t_2),\ldots,(p_n,t_n)\}$.

Formulación matemática:

- **Maximizar** $\sum_{i=1..n} B[p_i, t_i]$ sujeto a la restricción $p_i \neq p_j,\ t_i \neq t_j,\ \forall i \neq j$

Problema de asignación

Aplicación de backtracking (proceso metódico):

1) Determinar cómo es la **forma del árbol** de backtracking → cómo es la representación de la solución.

2) Elegir el **esquema del algoritmo** adecuado, adaptándola en caso necesario.

3) Diseñar las **funciones genéricas** para la aplicación concreta: según la forma del árbol y las características del problema.

4) Posibles **mejoras**: usar variables locales con "valores acumulados", con información adicional, etc.

Problema de asignación

1) Representación de la solución
   a) Mediante matriz de asignaciones:

   s = $((a_{11},a_{12},\ldots,a_{1n}), (a_{21},a_{22},\ldots,a_{2n}), \ldots, (a_{n1},a_{n2},\ldots,a_{nn}))$, con $a_{ii} \in \{0, 1\}$,

   y con $\sum_{i=1..n} a_{ij} = 1, \sum_{j=1..n} a_{ij} = 1$

   a_{ij}=1→la tarea j se asigna a la persona i

   a_{ij}=0→ la tarea j no se asigna a la persona i

   Árbol binario, pero muy profundo: n^2 niveles

   Tiene muchas restricciones

a	1	2	3
1	0	1	0
2	1	0	0
3	0	0	1

   b) Vector: desde el punto de vista de las personas

   s=($t_1, t_2, \ldots, t_n$), siendo $t_i \in \{1,\ldots,n\}$, con $t_i \neq t_j, \forall i \neq j$

   - **t_i→número de tarea asignada a la persona i**
   - Da lugar a un árbol permutacional
   - Número de nodos→n+n(n-1)+n(n-1)(n-2)+…+n!

   c) Vector: desde el punto de vista de las tareas

   s=($p_1, p_2, \ldots, p_n$), siendo $p_i \in \{1,\ldots,n\}$, con $p_i \neq p_j, \forall i \neq j$

   - **p_i→número de persona asignada a la tarea i**
   - Representación análoga (dual) a la anterior

Problema de asignación

2) Elegir el esquema del algoritmo: caso optimización (maximización)

```
Backtracking (var s: array[1..n] de entero)
  nivel:= 1; s:= s_INICIAL
  voa:= -∞; soa:= ∅
  bact:0
  repetir
   Generar (nivel, s)
   si Solución (nivel, s) AND (bact>voa) entonces
     voa:=bact; soa:=s
   si Criterio (nivel, s) AND (nivel<n) entonces
     nivel:= nivel + 1
   mientras NOT MasHermanos (nivel, s) AND (nivel>0)
     hacer Retroceder (nivel, s)
  hasta nivel==0
```

bact: Beneficio actual

¡Ojo! En C, los índices empiezan en 0

- Representar las personas por las posiciones 0,1,…,n-1
- Y también las tareas por simplicidad: 0,1,…,n-1

Problema de asignación

2) Elegir el esquema del algoritmo: caso optimización (maximización)

```
Backtracking (var s: array[0..n-1] de entero)
   nivel:= 0; s:= s_INICIAL
   voa:= -∞; soa:= ∅
   bact:0
   repetir
    Generar (nivel, s)
    si Solución (nivel, s) AND (bact>voa) entonces
       voa:=bact; soa:=s
    si Criterio (nivel, s) AND (nivel<n-1) entonces
       nivel:= nivel + 1
    mientras NOT MasHermanos (nivel, s) AND (nivel>=0)
       hacer Retroceder (nivel, s)
   hasta nivel==-1
```

El primer nivel será el nivel 0 (la primera persona, la primera posición en el vector solución)

Luego la condición de parada será hasta nivel==-1

3) Funciones genéricas del esquema de backtracking

Variables:

- `s`: vector de enteros:
  - cada `s[i]` indica la tarea (valor entre `0` y `n-1`) asignada a la persona `i` (valor entre `0` y `n-1`).
  - Hay que inicializarla a un valor adecuado, por ejemplo `-1` (un valor no válido como asignación)
- `bact`: beneficio de la solución actual

Problema de asignación

3) Funciones genéricas del esquema de backtracking

- Generar(nivel,s)→probar cada tarea en nivel: primero 0, luego 1, …, hasta n-1

```
s[nivel]=s[nivel]+1; //valor siguiente tarea
if (s[nivel]==0) //es la primera tarea que pruebo
   bact=bact+B[nivel][s[nivel]];
else                  //no es la primera tarea
   //resto lo que asigné en la prueba anterior
   bact=bact+B[nivel][s[nivel]]-B[nivel][s[nivel]-1];
```

- Criterio(nivel,s)→comprobar si la asignación es válida (tarea no usada)

```
for(i=0;i<nivel;i++)
   if (s[nivel]==s[i]) return 0;
return 1;
```

Tareas

Personas	B	0	1	2
	0	4	9	1
	1	7	2	3
	2	6	3	5

Problema de asignación

3) Funciones genéricas del esquema de backtracking

- Solución(nivel,s)→último nivel y asignación válida

```
return (nivel==n-1 & Criterio(nivel,s));
```

- MasHermanos(nivel,s)→quedan asignaciones por probar

```
return s[nivel]<n-1;
```

- Retroceder(nivel,s)→deshacer la asignación previa

```
bact=bact-B[nivel][s[nivel]];
s[nivel]=-1; //valor inicial
nivel=nivel-1;
```

Problema de asignación

En la 3 siguientes diapositivas se muestra:

a) El árbol completo con todas las posibles soluciones (las ramas que no cumplen el criterio ya no se dibujan)

b) Una diapositiva resumen con el código y la matriz B

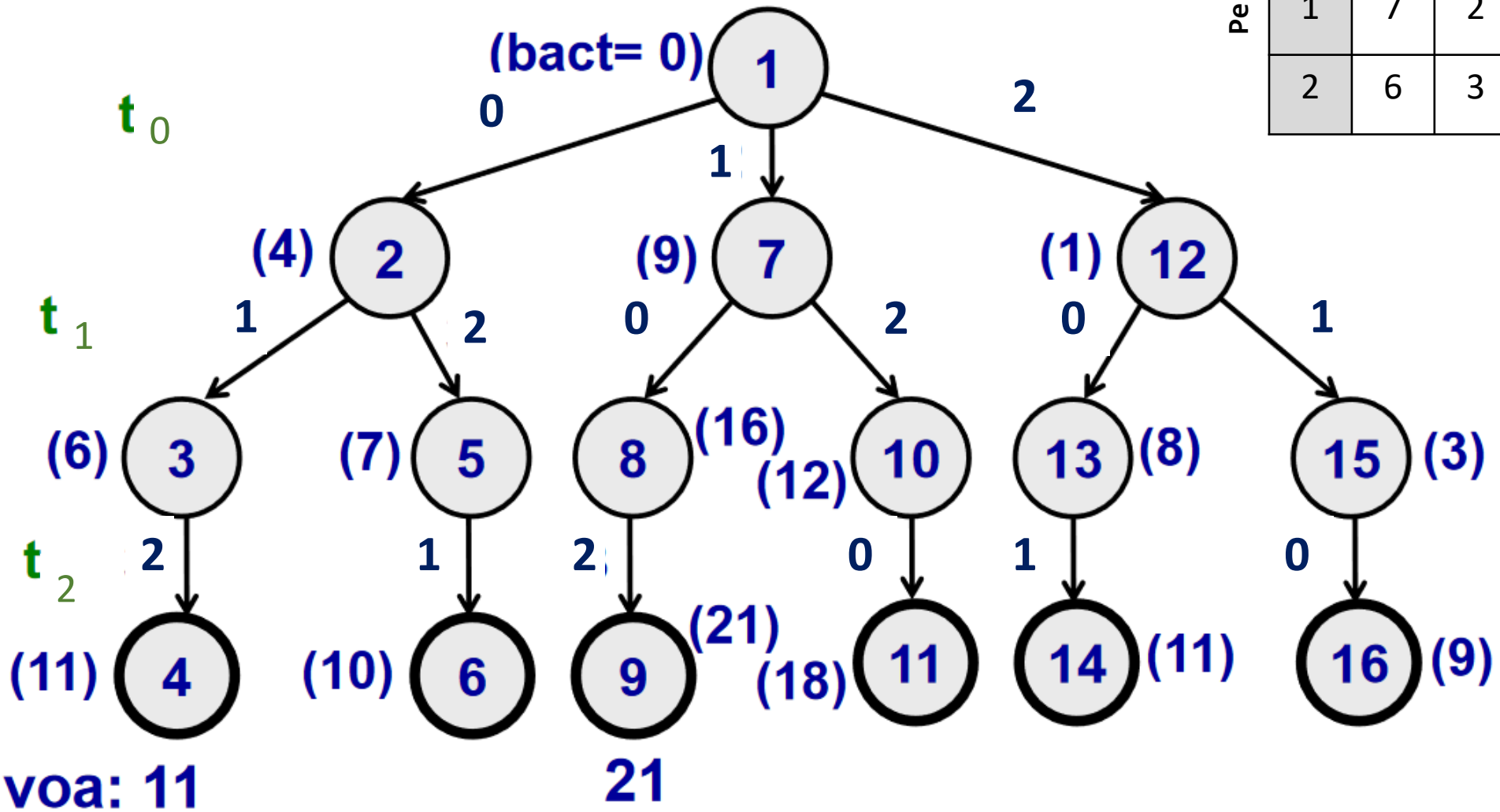
c) Una tabla resumen donde se va viendo cómo se va ejecutando el código

Problema de asignación

Ejemplo de aplicación. n = 3

Tareas

Personas	B	0	1	2
	0	4	9	1
	1	7	2	3
	2	6	3	5



Problema de asignación

```
Backtracking (var s: array[0..n-1] de entero)
 nivel:= 0; s:= s_INICIAL
 voa:= -∞; soa:= ∅
 bact:0
 repetir
  Generar (nivel, s)
  si Solución (nivel, s) AND (bact>voa)
     voa:=bact; soa:=s
  si Criterio (nivel, s) AND (nivel<n-1)
     nivel:= nivel + 1
  mientras NOT MasHermanos(nivel,s)AND(nivel>=0)
     Retroceder (nivel, s)
 hasta nivel==-1
```

```
Criterio(nivel,s)
 for(i=0;i<nivel;i++)
  if (s[nivel]==s[i]) return 0;
 return 1;

MasHermanos(nivel,s)
  return s[nivel]<n-1;

Retroceder(nivel,s)
  bact=bact-B[nivel][s[nivel]];
  s[nivel]=-1;
  nivel=nivel-1;
```

Tareas

Personas

B	0	1	2
0	4	9	1
1	7	2	3
2	6	3	5

```
Generar(nivel,s)
 s[nivel]=s[nivel]+1;
 if (s[nivel]==0)
  bact=bact+B[nivel][s[nivel]];
 else
  bact=bact+B[nivel][s[nivel]]-B[nivel][s[nivel]-1];

Solución(nivel,s)
 return (nivel==n-1 & Criterio(nivel,s));
```

Problema de asignación

Nodo	nivel	s	bact	voa	soa	Criterio	Solución	NOT MasHermanos
①	0	[-1,-1,-1]	0	-INF	[-1,-1,-1]			
②	0	[0,-1,-1]	0+B[0,0]=0+4=4	-	-	SI	NO	NO
③	1	[0,0,-1]	4+B[1,0]=4+7=11	-	-	NO	NO	NO
		[0,1,-1]	4+B[1,1]=4+2=6	-	-	SI	NO	NO
④	2	[0,1,0]	6+B[2,0]=6+6=12	-	-	NO	NO	NO
		[0,1,1]	6+B[2,1]=6+3=9	-	-	NO	NO	NO
		[0,1,2]	6+B[2,2]=6+5=11	11	[0,1,2]	SI	SI	**SI**
		[0,1,-1]	11-B[2,2]=11-5=6	-	-			NO
⑤	1	[0,2,-1]	6+B[1,2]-B[1,1]=6+3-2=7	-	-	SI	NO	NO
⑥	2	[0,2,0]	7+B[2,0]=7+6=13	-	-	NO	NO	NO
		[0,2,1]	7+B[2,1]=7+3=10	-	-	SI	SI	NO
		[0,2,2]	7+B[2,2]=7+5=12	-	-	NO	NO	**SI**
	2	[0,2,-1]	12-B[2,2]=12-5=7	-	-			SI
	1	[0,-1,-1]	7-B[1,2]=7-3=4					SI
⑦	0	[1,-1,-1]	4+B[0,1]-B[0,0]=4+9-4=9	-	-	SI	NO	NO
⑧	1	[1,0,-1]	9+B[1,0]=9+7=16	-	-	SI	NO	NO
⑨	2	[1,0,0]	-	-	-	NO	NO	NO
		[1,0,1]	-	-	-	NO	NO	NO
		[1,0,2]	16+B[2,2]=21	**21**	**[1,0,2]**	SI	SI	**SI**
…								

4) Mejoras

- Problema→: la función Criterio es muy lenta, repite muchas comprobaciones.
- Solución: usar un array que indique las tareas que están ya usadas en la asignación actual:
  - `usada`: array de enteros de tamaño n
  - `usada[i]` indica cuántas veces la tarea i está usada en la planificación actual (es decir, en s). Si está usada sólo una vez, la asignación es válida; si está usada más de una vez, no lo será.
  - Inicialización: `usada[i]=0`, para todo `i`
  - Modificar las funciones del esquema.

Problema de asignación

3) Funciones genéricas del esquema de backtracking

- Criterio(nivel,s)→comprobar si la asignación es válida (tarea no usada)

```
for(i=0;i<nivel;i++)
    if (s[nivel]==s[i]) return 0;
return 1;
return usada[s[nivel]]==1;
```

- Retroceder(nivel,s)→deshacer la asignación previa

```
bact=bact-B[nivel[s[nivel]];
usada[s[nivel]]--;
s[nivel]=-1; //valor inicial
nivel=nivel-1;
```

- Solución →no se modifica
- MasHermanos→no se modifica

3) Funciones genéricas del esquema de backtracking

- Generar(nivel,s)→probar cada tarea: primero 0, luego 1, …, hasta n-1

```
if s[nivel]!=-1
   usada[s[nivel]]--;
s[nivel]=s[nivel]+1;
usada[s[nivel]]++;
if (s[nivel]==0) //es la primera tarea que pruebo
   bact=bact+B[nivel][s[nivel]];
else                  //no es la primera tarea
   //resto lo que asigné en la prueba anterior
   bact=bact+B[nivel][s[nivel]]-B[nivel][s[nivel]-1];
```