1.- **(7 puntos)** Se dispone de una tabla T con n filas y m columnas que representan las posibilidades de que ciertos trabajadores (n) realicen determinadas tareas (m). Si T[i, j]=1 entonces el trabajador i-ésimo puede realizar la tarea j-ésima. En caso contrario no puede realizarla. Cada tarea puede ser realizada por uno o ningún trabajador, y cada trabajador debe tener una tarea o ninguna. El objetivo es obtener una asignación de trabajadores con tareas, de forma que el número de tareas realizadas sea máximo. Utilizando la metodología de Backtracking, diseñar un algoritmo que resuelva este problema.

Deberá responderse a las siguientes cuestiones:

a) Identificar y escribir el esquema de Backtracking a utilizar
b) Identificar cada una de las operaciones del esquema en relación con los datos del problema
c) Dar el algoritmo resultante

Ejemplo

	Tarea 1	**Tarea 2**	**Tarea 3**	**Tarea 4**	**Tarea 5**
Trabajador 1	1	0	0	0	0
Trabajador 2	0	1	1	1	0
Trabajador 3	0	1	0	0	1
Trabajador 4	1	0	1	1	0

Una solución óptima sería:

Trabajador 1 →Tarea 1 / Trabajador 2 →Tarea 2 / Trabajador 3 →Tarea 5 / Trabajador 4 →Tarea 3

Solución

Tupla de tamaño fijo

Secuencia de decisiones:
Tomamos n decisiones, tantas como trabajadores.
La decisión di (i:1..n) corresponde a un valor comprendido entre 0 y m. Si di=0, se entiende que el trabajador i no realiza ninguna tarea. En caso contrario, indica la tarea que debe realizar ese trabajador.

Restricciones explícitas: $\forall \mathrm{i}: \mathrm{d_i} \in \{0,1,2,\dots,\mathrm{n}\}$ con $(1 \leq \mathrm{i} \leq \mathrm{n})$

Restricciones implícitas:

$\forall(\mathrm{i}): (d_i = 0) \vee (T(i, d_i) = 0)$ con $(1 \leq \mathrm{i} \leq \mathrm{n})$

$\forall(\mathrm{i},\mathrm{j}): [(\mathrm{d_i} = 0) \vee (\mathrm{d}_j = 0) \vee (\mathrm{d_i} \neq \mathrm{d_j})]$ con $(1 \leq \mathrm{i} \leq \mathrm{n}) \wedge (1 \leq \mathrm{j} \leq \mathrm{n}) \wedge (\mathrm{i} \neq \mathrm{j})$

Función a optimizar: $maximizar\ (\mathrm{Ni}): \mathrm{d_i} \neq 0$ con $(1 \leq \mathrm{i} \leq \mathrm{n})$

Cada nivel del grafo corresponde a un trabajador. Es decir, en el nivel i decidimos si el trabajador i realiza alguna tarea, y, en caso afirmativo, cuál de ellas.

Esquema de Backtracking a utilizar:

```
Procedimiento Backtracking_OPTIMA (D:datos_problema; k:entero; e/s x, x_mejor:tupla; e/s v_mejor:valor)
preparar_recorrido_nivel_k;
mientras ∃_hermano_nivel_k hacer
        siguiente_hermano_nivel_k;
        opción
            solución(D,x,k) ∧ correcto(D,x,k): si valor(D,x,k) > v_mejor entonces
                                                  x_mejor=x;
```

```
                                        v_mejor=valor(D,x,k);
                              fsi
       ¬solución(D,x,k) ∧ correcto(D,x,k): Backtracking_OPTIMA (D, k+1, x, x_mejor, v_mejor);
        fopción
    fmientras
    fprocedimiento
```

Notas.- la función valor calcula el valor asociado a la secuencia de decisiones (función objetivo)

Las funciones auxiliares se definen del siguiente modo:

```
 Función preparar_recorrido_nivel_k    retorna entero
     k=-1
 Finfuncion

 Función solución(x:tupla,k:entero): retorna booleano
   retorna   (k==n)
 FinFunción

 Función correcto(T:matriz,x:tupla,n:entero,k:entero):  retorna booleano
     factible=Cierto;
     i=1;
     Mientras ( (i<=k Y factible ) {
          Si ( (x[i]!=0) Y (T[i][x[i]]==0) ) entonces factible=false;
           i++;
           j=0;
           Mientras ( (j<=i) Y factible ) {
                 Si ( (x[i]==x[j]) Y (x[i]!=0) Y (x[j]!=0))  entonces factible=false;
                  j++;
            }
      }
      Retorna factible
  FinFuncion

  Función valor(T:matriz,x:tupla,k:entero): retorna entero
      contador=0;
      Para j desde 1 hasta k
            Si (x[i]≠0)  Entonces  contador++   FinSi;
      Retorna contador;
   FinFuncion;
```

Algoritmo resultante:

```
Función AsignaTareas (T:datos_problema; , n:entero; m:entero,  k:entero; e/s x, x_mejor:tupla; e/s v_mejor:valor)
x[k]=-1;
mientras (x[k]<m) hacer
      x[k]:=x[k]+1;
      Si
             solución(x,k) ∧ correcto(x,k): si valor(T,x,k) > v_mejor entonces
                                                      x_mejor=x;
                                                      v_mejor=valor(D,x,k);
                                              fsi
```

¬(solución(x,k) ∧ correcto(,x,k)): AsignaTareas (T:matriz de enteros, n:entero; m:entero; k+1, x, x_mejor, v_mejor);
fopción
fmientras
fprocedimiento

Llamada inicial: AsignaTareas(T,n,m,1,x,x_mejor,0)

2.- **(3 puntos)** Se plantea una situación similar a la del problema 1, pero en este caso la tabla T también con n filas y m columnas representa los beneficios (valores enteros) de que ciertos trabajadores (n) realicen determinadas tareas (m). Si T[i, j]=s entonces el beneficio de que el trabajador i-ésimo realice la tarea j-ésima es s. Cada trabajador puede realizar varias tareas o ninguna pero las tareas deben ejecutarse todas y cada una sólo puede ser realizada por un trabajador. El objetivo es asignar trabajadores a las tareas, de forma que el beneficio global sea máximo. Utilizando la metodología Voraz, diseñar un algoritmo que resuelva este problema.
Deberá responderse a las siguientes cuestiones:

a) Identificar y escribir el esquema voraz a utilizar
b) Identificar cada una de las operaciones del esquema en relación con los datos del problema
c) Dar el algoritmo resultante

Ejemplo

	Tarea 1	Tarea 2	Tarea 3	Tarea 4	Tarea 5
Trabajador 1	1	2	6	6	5
Trabajador 2	3	5	1	4	0
Trabajador 3	4	1	8	5	1
Trabajador 4	2	7	7	2	3

Una posible solución a este problema sería:
Trabajador 3 → Tarea 1 / Trabajador 4 → Tarea 2 / Trabajador 3 → Tarea 3 / Trabajador 1 → Tarea 4 Trabajador 1 → Tarea 5

Solución

Secuencia de decisiones:
Tomamos m decisiones, tantas como tareas.
La decisión di (i:1..m) corresponde a un valor comprendido entre 1 y n.
di indica el trabajador que debe realizar la tarea i-ésima.

Naturaleza de las decisiones: $\forall i\colon d_i \in \{1,2,\ldots,n\}$ con $(1 \leq i \leq m)$
En cada decisión, el conjunto de candidatos Z está formado por los n trabajadores y consideramos como candidato más prometedor, aquel que ofrece el máximo beneficio para esa tarea, de acuerdo con los valores de la matriz T.

Condición de factibilidad: El criterio de selección de candidatos propuesto hace que todas las decisiones sean factibles.

Función a optimizar:

$$maximizar\left(\sum_{i=1}^{n} T[\mathrm{d_i}, i]\right)$$

La solución S se representa por medio de un vector con n componentes.

El esquema voraz a utilizar es:

```
Función Voraz ( x : T1 ) retorna ( y : T2 )
    var z : T´1, S : T2, decision : T3 fvar
    z = prepara (x);
    S = solucion_vacía;
    mientras ( ¬es_solucion (S) ∧ z ≠ ∅ ) hacer
            decision = selecciona_siguiente (z);
            z = elimina(z, decision);
            si factible (S, decision) entonces
                    S = añade (S, decision);
            fsi
    fmientras
    si es_solucion (S) entonces retorna S  sino retorna solucion_vacía  fsi
ffuncion
```

El algoritmo resultante es:

```
funcion solución_vacia(t:int)
  t=0;  // establece que la tarea curso es la 0;
finfuncion

funcion valor(T:matriz, x:tupla, k:int) {
   contador=0;
  Para i desde 0 hasta k hacer
      contador+= T[x[i]][i];
  retorna contador;
finfuncion

funcion es_solucion(n:int, k:int) {
  retorna (k==m-1);
finfuncion

funcion selecciona_siguiente(T:matriz, n:int, tarea:int) {
   indice_max,max=-1;
   para i desde 0 hasta n-1 hacer
       Si (T[i][*tarea]>max) entonces max=T[i][*tarea];
   para i desde 0 hasta n-1 hacer
       Si (T[i][*tarea]==max) entonces indice_max=i; printf("\nEl maximo para la tarea %d esta en la posicion  %d
                                                         y vale %d\n",*tarea+1,indice_max+1,T[indice_max][*tarea]);
   return indice_max;
}
```

```
funcion Voraz (T:matriz, n:int, m:int, trab:int, x:int)  {
  decisión:entero;
  solución_vacia(tarea);
  Mientras (*tarea)<=m-1   hacer
     decision = selecciona_siguiente (T,n,tarea);
     x[*tarea]=decision;
     (*tarea)++;
  FinMientras
  retorna valor(T,x,(*tarea)-1);
Finfuncion
```