ALGORITMIA

Esquemas Algorítmicos

Solución de Problemas de Exámenes de cursos anteriores

Curso Académico 2023-2024

Grado en Ingeniería Informática en Tecnologías de la Información
Escuela Politécnica de Ingeniería – Campus de Gijón
Universidad de Oviedo

Curso Académico 2020-2021

1) PROGRAMACIÓN DINÁMICA [5 puntos] El grupo de investigación AdA ha generado una elevada producción científica en el presente año y se dispone a diseñar una estrategia para derivar artículos generados a diferentes congresos. En su análisis, ha localizado C congresos (con $C \geq 1$) a los que poder enviar sus artículos. Se sabe que la inscripción en un congreso k, lo que le permite la presentación de un artículo, tiene un valor asociado INS[k] con $1 \leq k \leq C$. A su vez, cada congreso tiene asociado un índice de impacto, el cuál es un indicador de su calidad. Sea IMP[k] el índice de impacto asociado al congreso k con $1 \leq k \leq C$. Por otro lado, el grupo AdA dispone de un proyecto de investigación que tiene asociado un presupuesto de E euros en la partida de *Inscripciones a Congresos*, a gastar en el presente año.

Se pide diseñar un algoritmo basado en Programación Dinámica que determine cuántos artículos debe enviar el grupo AdA a cada congreso de tal forma que consiga maximizar la suma de los índices de impacto obtenidos, sin que el total de las inscripciones realizadas llegue a superar la cantidad de E euros disponible. Se considera que la producción del grupo es tal que dispone de un número inagotable de artículos. Se supone que los artículos enviados a un congreso son aceptados (algo que no siempre ocurre en la realidad). Por tanto, por cada artículo enviado se paga la inscripción correspondiente y por cada artículo enviado se obtiene el índice de impacto correspondiente. Responder a cada uno de los siguientes apartados con claridad y concisión:

a) Secuencia de decisiones (número de decisiones y significado de las mismas) (10%)
b) Función objetivo y restricciones (20%)
c) Demostración del principio de optimalidad (15%)
d) Definición recursiva y cómo se efectúa la primera llamada a la función recursiva (25%)
e) Árbol de llamadas COMPLETO, correspondiente a la definición recursiva anterior, para el siguiente ejemplo (10%): C = 3, E = 5, INS[1..3] = { 1, 2, 2 }, IMP[1..3] = { 1, 7, 3 }
f) Tipo de estructuras de almacenamiento elegidas y sus dimensiones y en qué orden se rellenan. Cómo se calculan dos posiciones concretas de las estructuras (una correspondiente a un caso trivial y a la otra a un caso no trivial). Cómo se obtiene la solución (valor y secuencia de decisiones). A este apartado f) se puede responder haciendo referencia al ejemplo del apartado e) o de forma genérica (20%)

a) SECUENCIA DE DECISIONES

$< x_1, x_2, \ldots, x_C >$, por consiguiente, tupla de longitud fija, donde x_1 indicará cuántos artículos se envían al congreso 1, x_2 indicará cuántos artículos se envían al congreso 2, …, x_i indicará cuántos artículos se envían al congreso i, …

b) FUNCIÓN OBJETIVO Y RESTRICCIONES

$$maximizar \sum_{i=1}^{C} IMP[i] * x[i]$$

sujeto a

$$\sum_{i=1}^{C} INS[i] * x[i] \leq E$$

c) DEMOSTRACIÓN DEL PRINCIPIO DE OPTIMALIDAD

Sea < x_1, x_2, …, x_C > la solución óptima del problema del envío de artículos a C congresos (del 1 al C) con un presupuesto de E euros con el objetivo de obtener el máximo impacto total. Denominaremos a dicho problema Congresos(C,E). El valor (impacto total) asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{C} IMP[i] * x_i$$

cumpliéndose además que

$$\sum_{i=1}^{C} INS[i] * x_i \leq E$$

Supongamos que prescindimos de la última decisión, esto es, x_C. Esta decisión indica cuántos artículos se envían al congreso C. La subsecuencia que queda, esto es < x_1, x_2, …, x_{C-1} > es la solución óptima para el problema asociado, es decir, para el subproblema

Congresos(C-1, E - INS[C]*x_C),

que es el subproblema del envío de artículos a C-1 congresos (del 1 al C-1) con un presupuesto de E-INS[C]*x_c euros con el objetivo de obtener el máximo impacto total. El valor (impacto total) asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{C-1} IMP[i] * x_i$$

cumpliéndose además que

$$\sum_{i=1}^{C-1} INS[i] * x_i \leq E - INS[C] * x_c$$

Supongamos que < x_1, x_2, …, x_{C-1} > **NO** es la solución óptima para el subproblema Congresos(C-1, E - INS[C]*x_C),sino que existe otra solución < y_1, y_2, …, y_{C-1} > que mejora su valor. Esto querrá decir que

$$\sum_{i=1}^{C-1} IMP[i] * x_i < \sum_{i=1}^{C-1} IMP[i] * y_i \quad (1)$$

cumpliéndose además que

$$\sum_{i=1}^{C-1} INS[i] * y_i \leq E - INS[C] * x_c$$

pero, si sumamos el término asociado al congreso C, esto es, IMP[C]*x_C a ambos lados de la desigualdad recogida en (1), se cumplirá que:

$$\sum_{i=1}^{C-1} IMP[i] * x_i + IMP[C] * x_c < \sum_{i=1}^{C-1} IMP[i] * y_i + IMP[C] * x_c$$

lo que significa que la secuencia < y_1, y_2, …, y_{C-1}, x_C > mejora el resultado de la secuencia <x_1, x_2, …, x_C > para el problema Congresos(C,E), lo cual contradice la hipótesis de partida, pues < x_1, x_2, …, x_C > era la solución óptima de dicho problema. En conclusión, se cumple el principio de optimalidad.

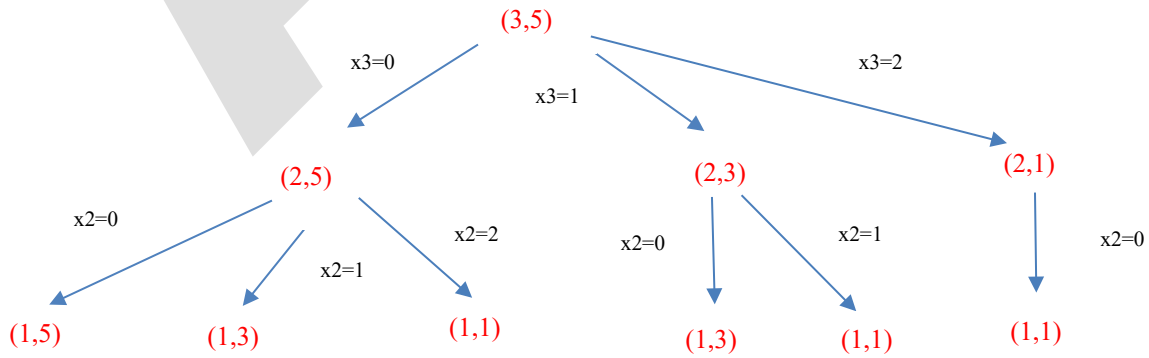
e) DEFINICIÓN RECURSIVA

$$Congresos\ (c,e) = \begin{cases} IMP[c] * (e/INS[c]) & si\ c = 1 \\ \max_{0 \leq x_c \leq e/INS[c]} \{Congresos(c-1, e - INS[c] * x_c) + IMP[c] * x_c\} & si\ c > 1 \end{cases}$$

donde Congresos(c,e), devuelve el máximo impacto total que se puede obtener al enviar artículos a c congresos (del 1 al c) sujeto a un presupuesto de e euros. La primera llamada a la función se producirá como Congresos(C,E), siendo C el número de congresos y E el presupuesto disponible. Indicar que la división recogida en la definición recursiva, esto es, e/INS[c], es la división entera.

f) ÁRBOL DE LLAMADAS

C=3, E = 5, INS[1..3] = { 1, 2, 2 } y IMP[1..3] = { 1, 7, 3 }



g) ESTRUCTURAS DE ALMACENAMIENTO

Dado que nuestra función recursiva tiene dos parámetros, se precisan dos estructuras bidimensionales. Estas tendrán C filas y E+1 columnas, esto es debido a que $1 \leq c \leq C$ y $0 \leq e \leq E$. En una de las estructuras, la denominaremos IMPmax, se guardará el valor asociado a cada subproblema, esto es, el máximo impacto global. Concretamente en IMPmax[c][e] estará recogido el valor correspondiente a Congresos(c,e). En la otra estructura, la denominaremos Dec, se almacenará la alternativa (0, 1, …, e/INS[c]) que proporciona el máximo para cada subproblema.

Orden en el que se rellenan:

- Primero, los problemas triviales, esto es Congresos(1,e), que se corresponde con rellenar la fila 1 de cada matriz. En el caso de IMPmax[1][e] con $0 \leq e \leq E$, sus valores serán: IMP[1]*(e/INS[1]).
- Dado que para solucionar el subproblema Congresos(c,e) se precisa conocer la solución de los subproblemas del tipo Congresos(c-1,z) donde $0 \leq z \leq e$, las matrices IMPmax y Dec se rellenan por filas en sentido creciente, desde la fila 2 hasta la fila C. (=> Dependencia de los subproblemas). Y dentro de cada fila el orden sería indistinto, es decir, de izquierda a derecha 0..E, o de derecha a izquierda E..0.

Cómo se calculan dos posiciones concretas de las estructuras (referencia al ejemplo del apartado e)):

- IMPmax[1][5]=IMP[1]*(5/INS[1])=1*(5/1)=5 (Caso trivial)

- IMPmax[3][5]=max{ IMPmax[2][5]+0, IMPmax[2][3]+3*1, IMPmax[2][1]+3*2 }= … (Caso no trivial)

Cómo se obtiene la solución (valor y secuencia de decisiones):

- El máximo impacto total (valor) estará en la posición [C][E] de la matriz IMPmax.

- La secuencia óptima de decisiones se obtiene recorriendo determinadas posiciones de la matriz Dec. Comenzaríamos por la posición [C][E], el valor ahí almacenado corresponderá a x_c. Seguidamente iríamos a Dec[C-1][E-x_C*INS[C]], ahí se encontrará el valor correspondiente a x_{C-1}. Seguidamente iríamos a Dec[C-2][E - x_C*INS[C] - x_{C-1}*INS[C-1]], ahí se encontrará el valor correspondiente a x_{C-2} . Y así sucesivamente.

2) VUELTA ATRÁS [2.5 puntos] Resolver el problema anterior aplicando la metodología de Vuelta atrás (Backtracking). Se pide responder con claridad y concisión a las siguientes cuestiones:

a) Secuencia de decisiones (número de decisiones y significado de las mismas) (10%)
b) Función objetivo y restricciones explícitas e implícitas (20%)
c) Preparar_recorrido_nivel_k, Existe_hermano_nivel_k, Siguiente_hermano_nivel_k y Solución (20%)
d) Describir qué hacen las funciones Correcto y/o Valor si es que fueran necesarias en la solución. Escribir el pseudocódigo de dichas funciones (≡ cómo lo hacen) (30%)
e) Para el mismo ejemplo del problema 1, dibujar el árbol de búsqueda que se explora hasta localizar al menos 7 soluciones factibles. Numerar los nodos reflejando el orden en el que se visitan (20%)

SOLUCIÓN.-

a) SECUENCIA DE DECISIONES

$< x_1, x_2, \ldots, x_C >$, por consiguiente, tupla de longitud fija, donde x_1 indicará cuántos artículos se envían al congreso 1, x_2 indicará cuántos artículos se envían al congreso 2, …, x_i indicará cuántos artículos se envían al congreso i, …

b) FUNCIÓN OBJETIVO y RESTRICCIONES EXPLÍCITAS E IMPLÍCITAS

$$maximizar \sum_{i=1}^{C} IMP[i] * x[i]$$

Restricciones explícitas $(\forall i)(x_i \epsilon \{0, 1, \ldots, E/INS[i]\}: 1 \leq i \leq C)$

Restricciones implícitas

$$\sum_{i=1}^{C} INS[i] * x[i] \leq E$$

c) PREPARAR_RECORRIDO_NIVEL_K, EXISTE_HERMANO_NIVEL_K, SIGUIENTE_HERMANO_NIVEL_K y SOLUCION

preparar_recorrido_nivel_k	x[k] = -1
existe_hermano_nivel_k	x[k] < E / INS[k]
siguiente_hermano_nivel_k	x[k] = x[k] + 1
SOLUCION(x,k)	k = C

d) FUNCIÓN CORRECTO Y FUNCIÓN VALOR

QUÉ HACE.- la función correcto, tras recibir la secuencia de decisiones x, el valor de k correspondiente y los datos del problema, devuelve falso, si la suma de las inscripciones de todos los artículos supera el valor de E euros; verdadero, en caso contrario.

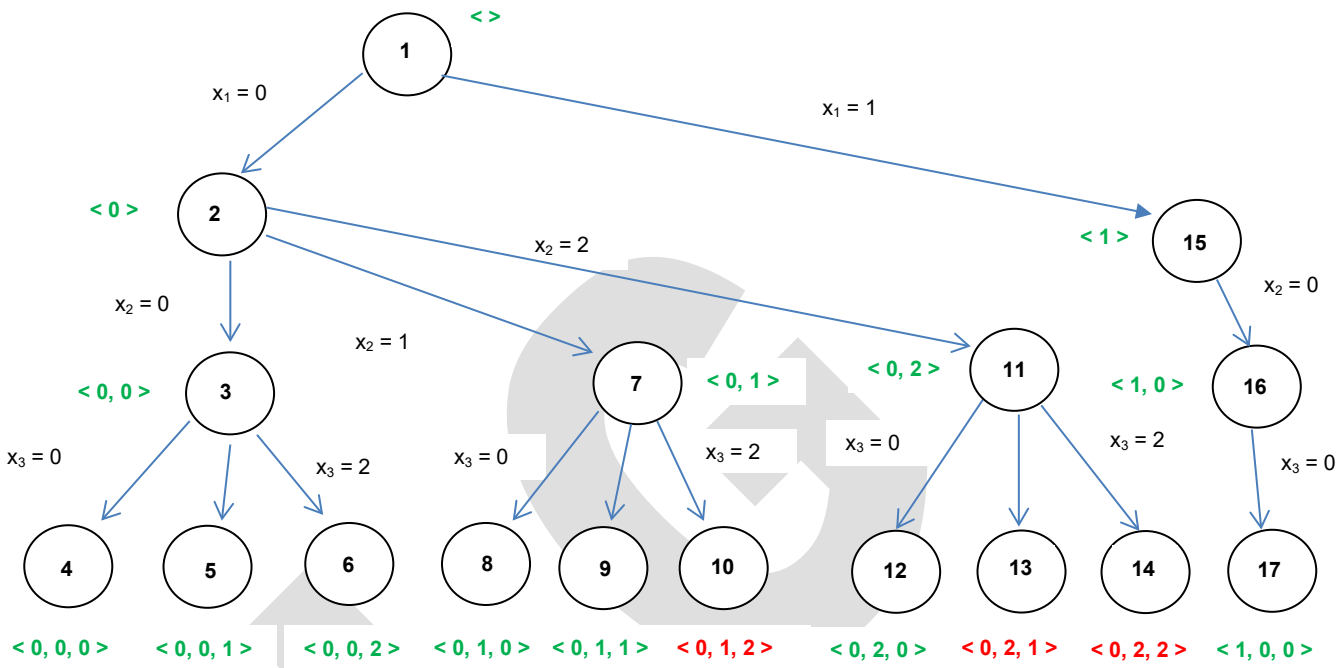
```
Funcion Correcto ( INS[1..C]: vector de enteros, C: entero, E:entero, x:tupla, k:entero )
                                                                    retorna ( b:booleano )
        var i, total: entero fvar
        total=0;
        para i=1 hasta k hacer
                total = total + INS[ i ]*x[ i ]
        fpara
        si total ≤ E entonces retorna verdadero
        sino retorna falso
        fsi
ffunción
```

QUÉ HACE.- la función Valor, tras recibir la secuencia de decisiones x, el valor de k, y los datos del problema, devuelve el valor de la función objetivo correspondiente a la secuencia de decisiones x, esto es, el impacto total.

```
Funcion Valor ( IMP[1..C]: vector de enteros, x:tupla, k:entero ) retorna ( b:entero )
        var i, total: entero fvar
        total=0;
        para i=1 hasta k hacer
                total = total + IMP[ i ] * x[ i ]
        fpara
        retorna total
ffunción
```

e) ÁRBOL DE BÚSQUEDA

C=3, E = 5, INS[1..3] = { 1, 2, 2 } y IMP[1..3] = { 1, 7, 3 }



Los nodos 10, 13 y 14 representan soluciones no factibles ya que el total de las inscripciones supera los 5 euros.

3) ESQUEMA VORAZ [2.5 puntos] Una empresa dispone de una flota de camiones para distribuir una serie de lotes de productos a lo largo de un conjunto de puntos de venta (un lote por cada punto de venta). Cada camión tardará un número de horas determinado en hacer efectiva la entrega en cada uno de los puntos. Cabe destacar que cada lote se mantiene en un estado óptimo si su entrega se realiza antes de un determinado número de horas. Sin embargo, si la entrega se realiza de manera posterior, entonces su estado de calidad decrece de manera proporcional por cada hora transcurrida desde la finalización de su número de horas límite. El objetivo es determinar que lote debería de ser entregado a cada punto de venta con el fin de minimizar la pérdida total de calidad de los lotes.
Para ello se dispone de los siguientes datos:

1) Número de puntos de venta y número de lotes N.
2) Número de horas que un camión tarda en llegar a cada uno de los puntos de venta, Horas[1..N] , donde Horas[i] es el número de horas en llegar al punto de venta i.
3) Límite de horas ideal para la entrega de cada lote, Limite[1..N], donde Limite[i] es el número de horas límite para el lote i.
4) Pérdida de valor del lote por cada hora transcurrida fuera de su límite de reparto, Perdida[1..N], donde Perdida[i] es la pérdida sufrida por el lote i.

Se pide resolver los siguientes apartados:

a) Especificar de forma concisa, clara y razonada una función de selección de candidatos que resuelva dicho problema. (15%)
b) Especificar cuál sería el conjunto de candidatos, el conjunto solución y la condición de parada del bucle voraz. (15%)
c) Aplicar la estrategia voraz sobre un ejemplo de forma detallada (por etapas), y explicar de manera razonada su funcionamiento en las dos primeras etapas (inicial y primera). (30%)

Etapas	Conjunto candidatos	Decisión	Conjunto solución	Pérdida total

Ejemplo: N = 4, Horas = [4, 2, 7, 3], Limite = [1, 5, 3, 6], Perdida = [10, 20, 3, 8]

d) Escribir el algoritmo voraz que resuelva el problema. (40%)

SOLUCIÓN.-

a) Especificar de forma concisa, clara y razonada una función de selección de candidatos

Criterio de selección:
Dado que lo que se pretende es minimizar la pérdida global de entrega del conjunto de lotes a lo largo de todos los puntos de venta, entonces cada lote debería de ser entregado en un punto de venta cuyo número de horas de entrega no excediese de su límite, en caso contrario debería de ser entregado a aquel punto que supusiese la menor pérdida posible. Por lo tanto, dado un un lote i y un punto de venta j será necesario tener en cuenta el siguiente valor $\max(0, Horas[j] - Limite[i]) * Perdida[i])$. En definitiva, la función a optimizar sería la siguiente:

Minimizar $\sum_{i=1}^{N} \max(0, Horas[j] - Limite[i]) * Perdida[i])$,
siendo j = conjunto de los puntos de venta 1..N.

b) Especificar cuál sería el conjunto de candidatos, el conjunto solución y la condición de parada del bucle voraz

El **conjunto de candidatos** estaría formado por el conjunto de puntos de venta.
El **conjunto solución** estaría formado por la secuencia de decisiones con una longitud total de n decisiones, es decir, tantas decisiones como lotes. Una entrada i-ésima del conjunto solución representaría que lote (1..N) ha sido asignado al punto de venta i.
La **parada del bucle voraz** se produciría en el momento que ya hubiesen sido distribuidos los N lotes a lo largo de los N puntos de venta.

c) Aplicar la estrategia voraz sobre un ejemplo de forma detallada (por etapas), y explicar de manera razonada su funcionamiento en las dos primeras etapas (inicial y primera)

Criterio: elegir en cada etapa i a que punto de venta debería de ser asignado el lote i con el fin de minimizar la penalización

$$\max(0, Horas[j] - Limite[i]) * Perdida[i] \; con$$
$$j \; = \; conjunto\ de\ puntos\ de\ venta \text{ sin lote } asignado$$

Por lo tanto, el objetivo sería colocar cada lote en aquel punto de venta más próximo a su número de horas límite con el fin de que no penalice o que penalice lo menos posible, y de esta forma favorecer la ubicación de otros lotes que podrían tener un número de horas límite inferior.

Ejemplo: N = 4, Horas = [4, 2, 7, 3], Límite = [1, 5, 3, 6], Pérdida = [10, 20, 3, 8]

Etapas	Conjunto candidatos	Decisión	Conjunto solución	Pérdida total
Inicial	{1, 2, 3, 4}	---	[0, 0, 0, 0]	0
1	{1, 3, 4}	2	[0, 1, 0, 0]	0+10=10
2	{1, 3}	4	[0, 1, 0, 2]	10+0=10
3	{3}	1	[3, 1, 0, 2]	10+3=13
4	{}	3	[3, 1, 4, 2]	13+8=21

Etapa inicial: conjunto de candidatos estaría formado por todos los puntos de venta. El conjunto solución debería ser vacío, donde una entrada i-ésima representa el lote asignado al punto de venta i. La pérdida inicial es 0.

Primera etapa: De acuerdo al criterio de selección explicado anteriormente, se debería de calcular la pérdida que se produciría si se asigna el lote 1 a cualquiera de los cuatro puntos de venta, y elegir aquel que genere la menor pérdida posible. Por lo tanto, de acuerdo a los siguientes cálculos:
Punto de venta 1: Pérdida del lote 1 = max(0, (4-1))*10 = 30
Punto de venta 2: Pérdida del lote 1 = max(0, (2-1))*10 = 10
Punto de venta 3: Pérdida del lote 1 = max(0, (7-1))*10 = 60
Punto de venta 4: Pérdida del lote 1 = max(0, (3-1))*10 = 20

De acuerdo al resultado, el lote 1 debería de ser asignado al punto de venta 2.

Segunda etapa: Ahora se debería de calcular la pérdida que se produciría si se asigna el lote 2 a los puntos de venta 1, 3 y 4, seleccionando aquel que genere la menor pérdida posible. Por lo tanto, de acuerdo a los siguientes cálculos:
Punto de venta 1: Pérdida del lote 2 = max(0, (4-5))*20 = 0
Punto de venta 3: Pérdida del lote 2 = max(0, (7-5))*20 = 40
Punto de venta 4: Pérdida del lote 2 = max(0, (3-5))*20 = 0
Por ejemplo, se selecciona el punto de venta 4.

Tercera etapa: Ahora se debería de calcular la pérdida que se produciría si se asigna el lote 3 a los puntos de venta 1 y 3. Por lo tanto, de acuerdo a los siguientes cálculos:
Punto de venta 1: Pérdida del lote 3 = max(0, (4-3))*3 = 3
Punto de venta 3: Pérdida del lote 3 = max(0, (7-3))*3 = 12
Se elegirá el punto de venta 1.

Cuarta etapa: Ahora se debería de calcular la pérdida que se produciría si se asigna el lote 4 al punto de venta 3, y sería seleccionado directamente, dado que es el último que queda.
Punto de venta 3: Pérdida del lote 4 = max(0, (7-6))*8 = 8
Se elige el punto de venta 3.

d) Escribir el algoritmo voraz que resuelve el problema

El algoritmo voraz a utilizar sería:

```
Función Voraz ( x: T1 ) retorna ( y: T2)
   var z: T'1, S: T2, decision: T3 fvar
   z = prepara (x);
   S = solucion_vacia;
   mientras (¬es_solucion (S) ^ z ≠ ø ) 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_vacia fsi
ffuncion
```

El algoritmo resultante sería el siguiente:

```
Funcion Voraz (H[1..N], L[1..N], P[1..N]: vector de enteros, N: entero)
                                        retorna (PerdidaMin: entero, Solucion[1..n]: vector de enteros)
var C: conjunto de enteros;
     S[1..N]: vector de enteros;
     i, decision: entero;
    perdida: entero;
fvar
C = {1, 2, …, N};
S = [0, 0, 0, 0];
perdida = 0;
i= 1;
mientras ( i <= N) hacer
      decision = seleccionar aquel punto de venta del conjunto C cuya pérdida para el
                  lote i sea mínima;
      C = C – {decision};
      S[decision] = i;
      perdida = perdida +max(0, H[decision]-L[i])*P[i];
      i = i + 1;
fmientras
retorna (perdida, S);
ffuncion
```

Curso Académico 2019-2020

1) PROGRAMACIÓN DINÁMICA [5 puntos] Una academia, que inaugurará sus instalaciones próximamente, pretende impartir un número total de H horas lectivas. Para ello debe contratar profesores y para ello dispone de C tipos de contratos diferentes. Un contrato del tipo k implica impartir D[k] horas lectivas y supone una remuneración de R[k] euros. Ambos datos son conocidos, esto es, el número de horas según el tipo de contrato está recogido en el vector de enteros D[1..C] y la remuneración según el tipo de contrato está recogida en el vector de enteros R[1..C]. Determinar cuántos contratos, y de qué tipo, es necesario realizar para impartir las H horas lectivas, de manera que la remuneración total sea mínima. Resolver el problema utilizando la metodología de Programación Dinámica, explicando cada uno de los apartados del proceso de resolución con claridad y concisión:

a. Secuencia de decisiones (número de decisiones y significado de las mismas) (10%)
b. Función objetivo y restricciones (20%)
c. Demostración del principio de optimalidad (15%)
d. Definición recursiva y cómo se realiza la primera llamada a la función recursiva (25%)
e. Árbol de llamadas completo, correspondiente a la definición recursiva anterior, para el siguiente ejemplo (10%)

H = 10, C = 3, D[1..3] = { 2, 5, 6 } y R[1..3] = { 4, 50, 75 }

f. Explicar de manera concisa y clara: tipo de estructuras de almacenamiento elegidas, sus dimensiones, en qué orden se rellenan y cómo se obtiene la solución (valor y secuencia de decisiones) (20%)

SOLUCIÓN.-

a) SECUENCIA DE DECISIONES

$< x_1, x_2, \ldots, x_C >$, por consiguiente, tupla de longitud fija, donde x_1 indicará cuántos contratos de tipo 1 se realizarán, x_2 indicará cuántos contratos de tipo 2 se realizarán, …, x_i indicará cuántos contratos de tipo i se van a realizar.

También se podría haber planteado como tupla de longitud variable $< x_1, x_2, \ldots, x_S >$, donde x_1 indicará el primer tipo de contrato que se va a realizar, x_2 indicará el segundo tipo de contrato que se va a realizar, …, x_i indicará el i-ésimo tipo de contrato que se va a realizar. De ese modo $x_i \in \{ 1, 2, \ldots, C \}$.

El resto de apartados (del b al f) desarrollan la solución de longitud fija.

b) FUNCIÓN OBJETIVO Y RESTRICCIONES

$$minimizar \sum_{i=1}^{C} R[i] * x[i]$$

sujeto a

$$\sum_{i=1}^{C} D[i] * x[i] = H$$

c) DEMOSTRACIÓN DEL PRINCIPIO DE OPTIMALIDAD

Sea $< x_1, x_2, \ldots, x_C >$ la solución óptima del problema de impartir H horas lectivas a través de contratos del tipo 1 al tipo C con la menor remuneración. Denominaremos a dicho problema Academia(C,H). El valor asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{C} R[i] * xi$$

cumpliéndose además que

$$\sum_{i=1}^{C} D[i] * x_i = H$$

Supongamos que prescindimos de la última decisión, esto es, x_C. Esta decisión indica cuántos contratos del tipo C se van a realizar. La subsecuencia que queda, esto es, $< x_1, x_2, \ldots, x_{C-1} >$ es la solución óptima para el problema asociado, es decir, para el subproblema

Academia(C-1, H - D[C]*x_C),

que es el subproblema de impatir H-D[C]*x_C horas lectivas a través de contratos del tipo 1 al tipo C-1 con la menor remuneración. El valor asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{C-1} R[i] * x_i$$

cumpliéndose además que

$$\sum_{i=1}^{C-1} D[i] * x_i = H - D[C] * x_c$$

Supongamos que $< x_1, x_2, \ldots, x_{C-1} >$ **NO** es la solución óptima para el subproblema Academia(C-1, H - D[C]*x_C), sino que existe otra solución $< y_1, y_2, \ldots, y_{C-1} >$ que mejora su valor. Esto querrá decir que

$$\sum_{i=1}^{C-1} R[i] * x_i > \sum_{i=1}^{C-1} R[i] * y_i \quad (1)$$

cumpliéndose además que

$$\sum_{i=1}^{C-1} D[i] * y_i = H - D[C] * x_c$$

pero, si sumamos el término asociado al tipo de contrato C, esto es, R[C]*x_C a ambos lados de la desigualdad recogida en (1), se cumpliría que:

$$\sum_{i=1}^{C-1} R[i] * x_i + R[C] * x_c > \sum_{i=1}^{C-1} R[i] * y_i + R[C] * x_c$$

lo que significa que la secuencia < y_1, y_2, …, y_{C-1}, x_C > mejora el resultado de la secuencia <x_1, x_2, …, x_C > para el problema Academia(C,H), lo cual contradice la hipótesis de partida, pues < x_1, x_2, …, x_C > era la solución óptima de dicho problema. En conclusión, se cumple el principio de optimalidad.

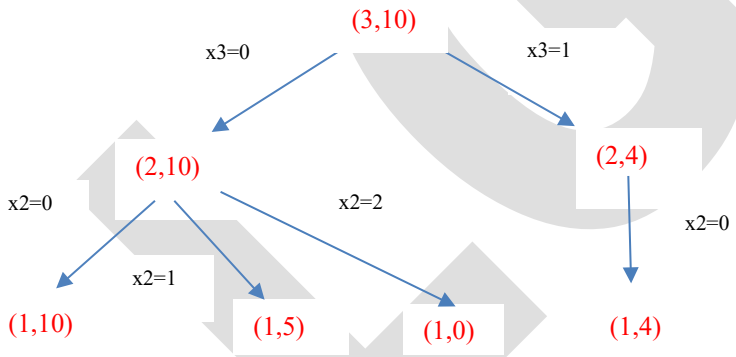
e) DEFINICIÓN RECURSIVA

$$Academia\ (c,h) = \begin{cases} R[c] * (h/D[c]) & si\ c = 1\ y\ h\%D[c] = 0 \\ +\infty & si\ c = 1\ y\ h\%D[c] != 0 \\ \min_{0 \le x_c \le h/D[c]} \{Academia(c-1, h - x_c * D[c]) + R[c] * x_c\} & si\ c > 1 \end{cases}$$

donde Academia(c,h) devuelve la mínima remuneración que se ha de realizar con c tipos de contratos (del 1 al c) para cubrir h horas lectivas. La primera llamada a la función se producirá como Academia(C,H), siendo C el número de tipos de contratos y H el número de horas lectivas a impartir.

f) ÁRBOL DE LLAMADAS

H = 10, C = 3, D[1..3] = { 2, 5, 6 } y R[1..3] = { 4, 50, 75 }



g) ESTRUCTURAS DE ALMACENAMIENTO

Dado que nuestra función recursiva tiene dos parámetros, se precisan dos estructuras bidimensionales. Estas tendrán C filas y H+1 columnas, esto es debido a que 1 ≤ c ≤ C y 0 ≤ h ≤ H. En una de las estructuras, Rmin, se guardará el valor asociado a cada subproblema, esto es, la mínima remuneración. Concretamente en Rmin[c][h] estará recogido el valor correspondiente a Academia(c,h). En la otra estructura, Dec, se almacenará la alternativa (0, 1, …, h/D[c]) que proporcione el mínimo para cada subproblema.

Orden en el que se rellenan:

- Primero, los problemas triviales, esto es Academia(1,h), que se corresponde con rellenar la fila 1 de cada matriz. En el caso de Rmin[1][h] con 0 ≤ h ≤ H, sus valores serán: R[1]*(h/D[1]) ó +∞, según h%D[1] sea igual a 0 o distinto de 0.
- Dado que para solucionar el subproblema Academia(c,h) se precisa conocer la solución de los subproblemas del tipo Academia(c-1,z) donde 0 ≤ z ≤ h, las matrices Rmin y Dec se rellenan por filas en sentido creciente, desde la fila 2 hasta la fila C. (=> Dependencia de los subproblemas).

La remuneración mínima estará en la posición [C][H] de la matriz Rmin.

La secuencia óptima de decisiones se obtiene recorriendo determinadas posiciones de la matriz Dec. Comenzaríamos por la posición [C][H], el valor ahí almacenado correspondería a x_c. Seguidamente iríamos a Dec[C-1][H-x_C*D[C]], ahí se encontrará el valor correspondiente a x_{C-1}. Seguidamente iríamos a Dec[C-2][H- x_C *D[C]- x_{C-1} *D[C-1]], ahí se encontrará el valor correspondiente a x_{C-2} . Y así sucesivamente.

2) VUELTA ATRÁS [2.5 puntos] Resolver el problema anterior aplicando la metodología de Vuelta atrás (Backtracking). Se pide responder con claridad y concisión a las siguientes cuestiones:

a. Secuencia de decisiones (número de decisiones y significado de las mismas) (5%)
b. Función objetivo (5%)
c. Restricciones explícitas e implícitas (20%)
d. Preparar_recorrido_nivel_k, Existe_hermano_nivel_k y Siguiente_hermano_nivel_k (15%)
e. Función Solución (5%)
f. Indicar qué hacen las funciones Correcto y/o Valor si es que fueran necesarias en la solución. Escribir el pseudocódigo de dichas funciones (≡ cómo lo hacen) (25%)
g. Para el mismo ejemplo del problema 1, dibujar el árbol de búsqueda que se explora hasta localizar la primera solución factible. Numerar los nodos reflejando el orden en el que se visitan e indicar cuándo se realiza una poda y por qué (25%).

SOLUCIÓN.-

a) SECUENCIA DE DECISIONES

< x_1, x_2,…, x_C >, por consiguiente, tupla de longitud fija, y donde x_1 indicará cuántos contratos de tipo 1 se realizarán, x_2 indicará cuántos contratos de tipo 2 se realizarán, …, x_i indicará cuántos contratos de tipo i se realizarán.

Al igual que en el ejercicio 1, también se podría haber planteado como tupla de longitud variable < x_1, x_2,…, x_S >, donde x_1 indicará el primer tipo de contrato que se va a realizar, x_2 indicará el segundo tipo de contrato que se va a realizar, …, x_i indicará el i-ésimo tipo de contrato que se va a realizar. De ese modo $x_i \in \{1, 2, \ldots, C\}$.

El resto de apartados (del b al g) desarrollan la solución de longitud fija.

b) FUNCIÓN OBJETIVO

$$minimizar \sum_{i=1}^{C} R[i] * x[i]$$

c) RESTRICCIONES EXPLÍCITAS E IMPLÍCITAS

Restricciones explícitas $(\forall i)(x_i \epsilon \{0, 1, \ldots, H/D[i]\}: 1 \leq i \leq C)$

Restricciones implícitas

$$\sum_{i=1}^{C} D[i] * x[i] = H$$

$$si\ k < C\ entonces \sum_{i=1}^{k} D[i] * x[i] \leq H$$

d) PREPARAR_RECORRIDO_NIVEL_K, EXISTE_HERMANO_NIVEL_K Y SIGUIENTE_HERMANO_NIVEL_K

preparar_recorrido_nivel_k	x[k] = -1
existe_hermano_nivel_k	x[k] < H / D[k]
siguiente_hermano_nivel_k	x[k] = x[k] + 1

e) FUNCIÓN SOLUCIÓN

k = C

f) FUNCIÓN CORRECTO Y FUNCIÓN VALOR

QUÉ HACE.- la función correcto, tras recibir la secuencia de decisiones x y el valor de k correspondiente, devuelve

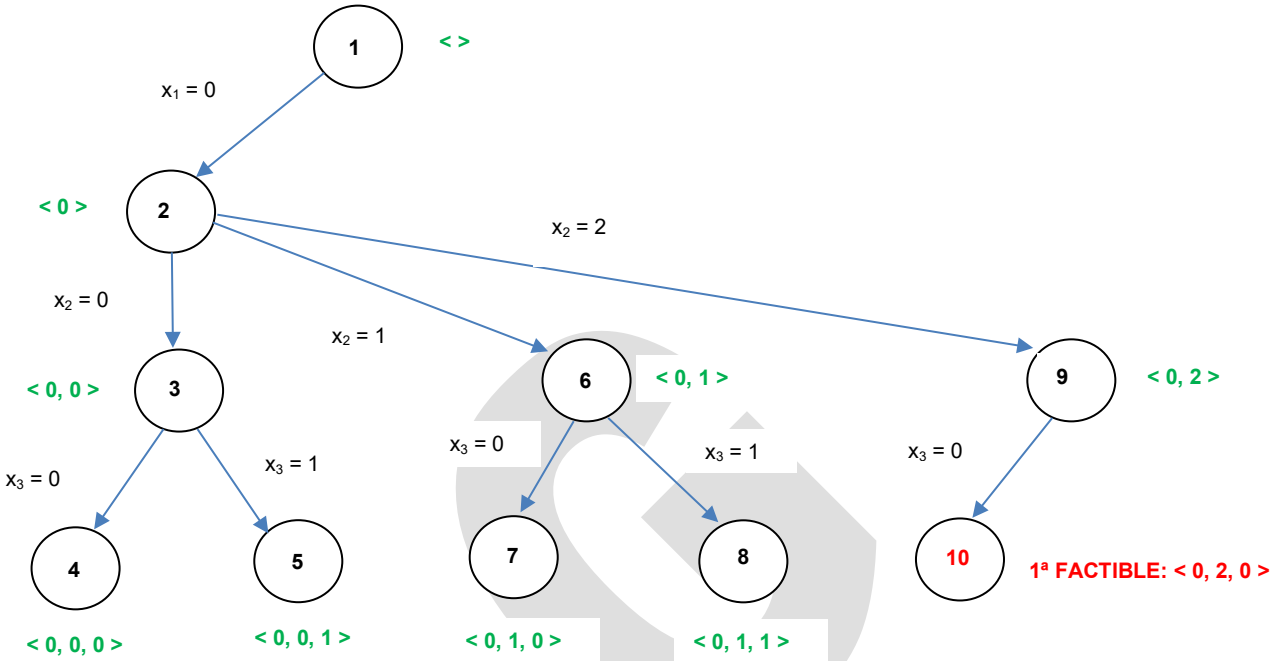
- en el caso de una tupla incompleta: falso, si la suma de las horas lectivas que suponen todos los contratos supera el valor de H; verdadero, en caso contrario.
- en el caso de una tupla completa: verdadero, si la suma de las horas lectivas que suponen todos los contratos coincide con el valor de H; falso, en caso contrario.

```
Funcion Correcto ( D[1..C]: vector de enteros, C: entero, x:tupla, k:entero ) retorna ( b:booleano )
        var i, total: entero fvar
        total=0;
        para i=1 hasta k hacer
                total = total + D[ i ]*x[ i ]
        fpara
        si k<C entonces
                si total ≤ H entonces retorna verdadero
                        sino retorna falso
                fsi
        sino si total = H entonces retorna verdadero
             sino retorna falso
             fsi
        fsi
ffunción
```

QUÉ HACE.- la función Valor, tras recibir la secuencia de decisiones x y el valor de k, devuelve el valor de la función objetivo correspondiente a la secuencia de decisiones x, esto es, la remuneración total.

```
Funcion Valor ( R[1..C]: vector de enteros, x:tupla, k:entero ) retorna ( b:entero )
        var i, total: entero fvar
        total=0;
        para i=1 hasta k hacer
                total = total + R[ i ] * x[ i ]
        fpara
        retorna total
ffunción
```

g) ÁRBOL DE BÚSQUEDA



En los nodos 4, 5, 7 y 8 se alcanza una secuencia de decisiones completa pero incorrecta ya que el número de horas lectivas que representa no coincide con H.

3) ESQUEMA VORAZ [2.5 puntos] Una tienda dispone de una serie de productos donde cada uno de los cuales tiene asociado un **precio de coste, un precio de venta** atendiendo a un margen de beneficio esperado **(precio coste / (1 - %margen)**) **y un coste de almacenaje** que se mide **en una escala del 1 al 3 (1-alto, 2-medio, 3-bajo).** El propietario desea realizar una selección de artículos a los cuáles se les aplicará un descuento. Por lo tanto, se deberá tomar una decisión sobre que artículos deberán ser rebajados, de manera que su venta suponga maximizar la ganancia total teniendo en cuenta el coste de almacenaje.

Para la resolución del problema se dispondrá de los siguientes datos de entrada:

n: número de productos.

pr: porcentaje de productos que se desea rebajar.

pm: el porcentaje de beneficio (margen) esperado de los productos.

pd: el porcentaje de descuento que se desea aplicar.

PC[1..n]: vector con los **precios de coste** de cada uno de los n productos, PC[i] con 1<=i<=n contendrá el precio de coste del producto i.

CA[1..n]: vector con los **costes de almacenaje** de cada uno de los n productos, CA[i] con 1<=i<=n contendrá el coste de almacenaje del producto i, y cuyo valor podrá ser 1 (alto), 2 (medio) ó 3 (bajo).

Se pide resolver los siguientes apartados:

e) Especificar de forma concisa, clara y razonada una función de selección de candidatos que resuelva dicho problema. (15%)

f) Especificar cuál sería el conjunto de candidatos, el conjunto solución y la condición de parada del bucle voraz. (15%)

g) Aplicar la estrategia voraz sobre un ejemplo de forma detallada (por etapas), y explicar de manera razonada su funcionamiento en las dos primeras etapas (inicial y primera). (30%)

Etapas	Conjunto candidatos	Decisión	Conjunto solución	Beneficio total

Ejemplo: n = 6, **pm** = 20%, **pd** = 10%,**pr** = 70%

Precio de coste (PC):	50	30	20	100	70	60
Coste de almacenaje (CA):	3	3	2	1	1	2
Precio de venta (PV):	62,5	37,5	25	125	87,5	75
Descuento (DTO):	6,25	3,75	2,5	12,5	8,75	7,5

h) Escribir el algoritmo voraz que resuelva el problema. (40%)

SOLUCIÓN.-

a) Especificar de forma concisa, clara y razonada una función de selección de candidatos

Criterio de selección:

Dado que lo que se pretende es optimizar el beneficio, pero teniendo en cuenta el coste de almacenaje, los productos podrían ser seleccionados en orden creciente de acuerdo al ratio CA_i/B_i, siendo el beneficio del producto i (B_i) igual a B_i = Precio de venta $_i$ – Precio de coste $_i$ – descuento $_i$. De hecho, dados dos productos que proporcionasen el mismo beneficio, debería ser seleccionado aquel con mayor coste de almacenaje. Así por ejemplo, si dos productos tuviesen un mismo beneficio de 10 ($B_1 = B_2 = 10$) y un coste de almacenaje de 1 ($CA_1 = 1$) y 2 ($CA_2 = 2$), entonces la relación entre ambos factores sería de 0,1 (CA_1 / B_1) y 0,2 (CA_2 / B_2) respectivamente, lo que implicaría que el producto con un beneficio de 10 y un coste de almacenaje de 1 debería ser seleccionado en primer lugar para ser rebajado antes que el segundo, dado que ante un mismo beneficio supone un mayor coste de almacenaje.

b) Especificar cuál sería el conjunto de candidatos, el conjunto solución y la condición de parada del bucle voraz

El **conjunto de candidatos** estaría formado por todos los productos de la tienda.
El **conjunto solución** estaría formado por la secuencia de decisiones con una longitud total de m decisiones, siendo m = E(pr * n), es decir, tantas como productos a rebajar. De hecho, una decisión d_i ($1 \leq i \leq m$) se correspondería con un valor comprendido entre 1 y n, donde d_i indica el producto que sería rebajado.
La **parada del bucle voraz** se produciría en el momento que ya hubiesen sido seleccionados m productos.

c) Aplicar la estrategia voraz sobre un ejemplo de forma detallada (por etapas), y explicar de manera razonada su funcionamiento en las dos primeras etapas (inicial y primera)

Criterio: MÍNIMO CA_i / B_i (Ratio), en caso de empate sería indiferente.

Ratio [1..6] = {0,48; 0,8; 0,8; 0,08; 0,11; 0,27}

Etapas	Conjunto candidatos	Decisión	Conjunto solución	Beneficio total
Inicial	{1, 2, 3, 4, 5, 6}	---	{}	0,0
1	{1, 2, 3, 4, 6}	4	{4}	12,5
2	{1, 2, 3, 6}	5	{4, 5}	21,25
3	{1, 2, 3}	6	{4, 5, 6}	28,75
4	{2, 3}	1	**{4, 5, 6, 1}**	**35**

Etapa inicial: El conjunto de candidatos debería de estar formado por todos los productos de la tienda, el conjunto solución debería de ser vacío dado que por el momento no ha sido seleccionado ningún producto para ser rebajado, y por lo tanto el beneficio que tendríamos por la venta de productos rebajados sería de 0.

Primera etapa: De acuerdo al criterio de selección explicado anteriormente, el primer producto que debería de ser elegido para ser rebajado sería el producto número 4, dado que es el que presenta un menor valor de acuerdo al valor del ratio (CA_4 / B_4 = 0,08). Por lo tanto, dicho producto tendría que ser eliminado de candidatos para no ser tenido en cuenta en etapas posteriores del bucle voraz e incorporado a la solución. Este hecho implicaría un incremento en el beneficio total de 12,5 (Precio $venta_4$– Precio $coste_4$ – $descuento_4$ = 125 – 100 - 12,5), en el caso de que fuese vendido.

d) Escribir el algoritmo voraz que resuelve el problema

El algoritmo voraz a utilizar sería:

```
Función Voraz ( x: T1 ) retorna ( y: T2)
   var z: T'1, S: T2, decision: T3 fvar
   z = prepara (x);
   S = solucion_vacia;
   mientras (¬es_solucion (S) ^ z ≠ ø ) 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_vacia fsi
ffuncion
```

El algoritmo resultante sería el siguiente:

```
Funcion Voraz (PC[1..n]: vector de enteros, CA[1..n]: vector de enteros, n, pr, pm, pd: entero)
                                                    retorna (Bmax: real, S: conjunto de enteros)
var C, S: conjunto de enteros;
     i, m, decision: entero;
     beneficio: real;
     PV[1..n]: vector de reales;
     DTO[1..n]: vector de reales;
fvar

para i = 1 hasta n hacer
     PV[i] = PC[i] / (1 – (pm / 100));
     DTO[i] = PV[i] * (pd / 100);
fpara

m = Parte entera(n * pr);
C = {1, …, n};
S = {};
beneficio = 0.0;
i= 0;
mientras ( i < m) hacer
      decision = seleccionar aquel producto aún no seleccionado cuyo ratio CAi/Bi sea menor;
      C = C – {decision};
      S = S + {decision};
      beneficio = beneficio + (PV[decision] – PC[decision] – DTO[decision]);
      i = i + 1;
fmientras
retorna (beneficio, S);
ffuncion
```

Curso Académico 2018-2019

1) PROGRAMACIÓN DINÁMICA [5 puntos] Un comercial que distribuye un producto específico tiene establecida una ruta fija que incluye C ciudades (numeradas de la 1 a la C), cada una de las cuales puede visitar opcionalmente según su criterio. El número de unidades disponibles del producto que representa es U cuando inicia la ruta. Por experiencia sabe que dependiendo de la ciudad varían tanto el precio de venta como la demanda. Ambos datos son conocidos. El precio de venta está recogido en el vector P[1..C], donde P[k] con $1 \leq k \leq C$, es el precio de venta unitario del producto en la ciudad k-ésima mientras que la demanda está recogida en el vector D[1..C], donde D[k] con $1 \leq k \leq C$, es la demanda en la ciudad k. Determinar las ciudades que debe visitar para obtener un beneficio máximo, teniendo en cuenta que se visitará una ciudad siempre y cuando su demanda se pueda atender al haber unidades disponibles del producto, sino no. Resolver el problema utilizando la metodología de Programación Dinámica, explicando cada uno de los apartados del proceso de resolución con claridad y concisión:

a) Secuencia de decisiones (número de decisiones y significado de las mismas) (10%)
b) Función Objetivo (10%)
c) Restricciones (10%)
d) Demostración del principio de optimalidad (15%)
e) Definición recursiva y cómo se realiza la primera llamada a la función (25%)
f) Árbol de llamadas completo para el siguiente ejemplo (10%)

U = 20, C = 4, P[1..4] = { 30, 12, 20, 15 } y D[1..4] = { 10, 7, 5, 12 }

g) Explicar de manera concisa y clara: tipo de estructuras de almacenamiento elegidas, sus dimensiones, en qué orden se rellenan y cómo se obtiene la solución (valor y secuencia de decisiones) (20%)

SOLUCIÓN.-

a) SECUENCIA DE DECISIONES

$< x_1, x_2, \ldots, x_C >$, por consiguiente, tupla de longitud fija, y donde x_1 indicará si se visita la primera ciudad o no, x_2 indicará si se visita la segunda ciudad o no, …, x_i indicará si se visita la ciudad i-ésima o no (valor 1, indicaría que sí; valor 0, indicaría que no).

b) y c) FUNCIÓN OBJETIVO Y RESTRICCIONES

$$maximizar \sum_{i=1}^{C} P[i] * D[i] * x[i]$$

sujeto a

$$\sum_{i=1}^{C} D[i] * x[i] \leq U$$

El problema tiene una resolución análoga al de la mochila visto en clase (consultar apuntes de clase y/o Brassard, pág. 299), reemplazando el concepto de objeto a incluir en la mochila por el de ciudad a visitar y asociando el valor U del enunciado con el límite de capacidad de la mochila.

d) DEMOSTRACIÓN DEL PRINCIPIO DE OPTIMALIDAD

La demostración del principio de optimalidad es análoga a la del problema de la mochila (consultar las referencias anteriores).

Sea < x_1, x_2, …, x_C > la solución óptima del problema de comercializar U unidades de un producto en una ruta de C ciudades (de la 1 a la C) obteniendo el mayor beneficio. Denominaremos a dicho problema Comercial(C,U). El valor asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{C} P[i] * D[i] * xi$$

cumpliéndose además que

$$\sum_{i=1}^{C} D[i] * x_i \leq U$$

Supongamos que prescindimos de la última decisión, esto es, d_C. La cual indica si se va a visitar la ciudad C o no y por tanto, si se cubre su demanda o no. La subsecuencia que queda, esto es, < x_1, x_2, …, x_{C-1} > es la solución óptima para el problema asociado, es decir, para el subproblema

Comercial(C-1, U - D[C]*x_C),

que es el subproblema de comercializar U-D[C]*x_C unidades del producto en una ruta de C-1 ciudades (de la 1 a la C-1) obteniendo el mayor beneficio.

El valor asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{C-1} P[i] * D[i] * x_i$$

cumpliéndose además que

$$\sum_{i=1}^{C-1} D[i] * x_i \leq U - D[C] * x_C$$

Supongamos que < x_1, x_2, …, x_{C-1} > **NO** es la solución óptima para el subproblema Comercial(C-1, U - D[C]*x_C), sino que existe otra solución < y_1, y_2, …, y_{C-1} > que mejora su valor. Esto querrá decir que

$$\sum_{i=1}^{C-1} P[i] * D[i] * X_i \; < \; \sum_{i=1}^{C-1} P[i] * D[i] * y_i \qquad (1)$$

cumpliéndose además que

$$\sum_{i=1}^{C-1} D[i] * y_i \leq U - D[C] * x_C$$

Pero entonces si sumamos el término asociado a la ciudad C, esto es, P[C]*D[C]*x_C a ambos lados de la desigualdad recogida en (1), se cumpliría que:

$$\sum_{i=1}^{C-1} P[i] * D[i] * x_i \; + P[C] * D[C] * x_C \; < \; \sum_{i=1}^{C-1} P[i] * D[i] * y_i \; + P[C] * D[C] * x_c$$

lo que significa que la secuencia < y_1, y_2, …, y_{C-1}, x_C > mejoraría el resultado de la secuencia <x_1, x_2, …, x_C > para el problema Comercial(C,U), lo cual contradice la hipótesis de partida, pues < x_1, x_2, …, x_C > era la solución óptima de dicho problema. En conclusión, se cumple el principio de optimalidad.

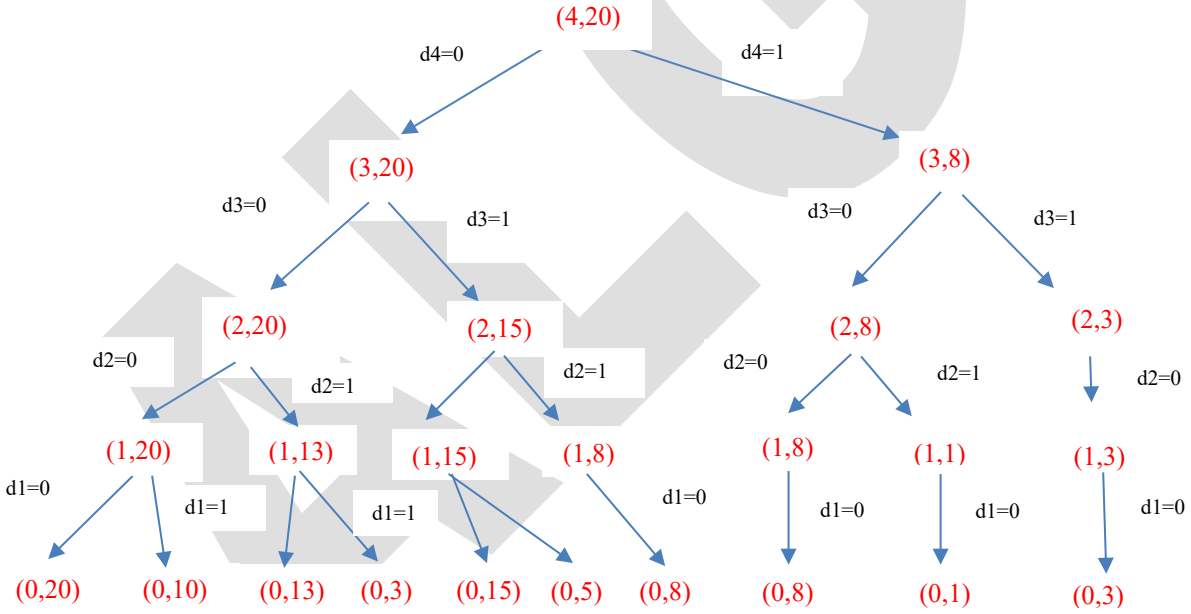
e) DEFINICIÓN RECURSIVA

$$Comercial\ (c,u) = \begin{cases} 0 & si\ c = 0 \\ Comercial(c-1,u) & si\ c > 0\ y\ u < D[c] \\ \max_{d_c \in \{0,1\}} \{Comercial(c-1, u - d_c * D[c]) + P[c] * D[c] * d_c\} & si\ c > 0\ y\ u \geq D[c] \end{cases}$$

donde Comercial(c,u) devuelve el máximo beneficio que se pueden obtener entre las c ciudades (del 1 al c) con u unidades disponibles del producto que se comercializa.
La primera llamada a la función se producirá como Comercial(C,U), siendo C el número de ciudades y U el número de unidades disponibles al inicio de la ruta.

f) ÁRBOL DE LLAMADAS

U = 20, C = 4, P[1..4] = { 30, 12, 20, 15 } y D[1..4] = { 10, 7, 5, 12 }



…

g) ESTRUCTURAS DE ALMACENAMIENTO

Dado que nuestra función recursiva tiene dos parámetros, se precisan dos estructuras bidimensionales. Estas tendrán C+1 filas y U+1 columnas, esto es debido a que $0 \le c \le C$ y $0 \le u \le U$. En una de las estructuras, Bmax, se guardará el valor asociado a cada subproblema, esto es, el beneficio máximo obtenido. En la otra estructura, Dec, se almacenará la alternativa (0 o 1) que proporcione el máximo para cada subproblema.

Orden en el que se rellenan:

a. Primero, los resultados de los problemas triviales que corresponde con rellenar la fila 0 de cada matriz con 0, dado que los problema triviales son del tipo Comercial(0,u), esto es, Bmax[0][u]=0 con $0 \le u \le U$ y Dec[0][u]=0 con $0 \le u \le U$.
b. El resto, dado que para solucionar el subproblema Comercial(c,u) se precisa conocer la solución de los subproblemas del tipo Comercial (c-1,x) donde $x \le u$, ambas matrices se rellenan por filas en sentido creciente, desde la fila 1 hasta la fila C. (=> Dependencia de los subproblemas).

El beneficio máximo obtenido estará en la posición [C][U] de la matriz Bmax.

La secuencia óptima de decisiones se obtiene recorriendo determinadas posiciones de la matriz Dec. Comenzaríamos por la posición [C][U], el valor ahí almacenado correspondería a d_c. Seguidamente iríamos a Dec[C-1][U-d_C*D[C]], ahí se encontrará el valor correspondiente a d_{C-1}. Seguidamente iríamos a Dec[C-2][U-dC*D[C]- d_{C-1} *D[C-1]], ahí se encontrará el valor correspondiente a d_{C-2} . Y así sucesivamente.

2) BACKTRACKING [3.5 puntos] Resolver el problema de los embarcaderos visto en clase en el tema de Programación Dinámica ahora desde la perspectiva de Backtracking (Vuelta Atrás). El enunciado del problema era el siguiente: A lo largo de la ribera de un río hay N embarcaderos (numerados del 1 al N). Se tiene una matriz C de tamaño N x N, de manera que C[i][j] indica el coste de ir directamente del embarcadero i al embarcadero j ($1 \le i \le j \le N$). Se asume que el coste de ir de un embarcadero a él mismo es cero, o sea, C[i][i]=0 $\forall i : 1..N$. La única limitación que existe en este problema es que desde un embarcadero no se puede volver a ninguno de los anteriores. Se pretende saber cuál es el coste mínimo para ir desde el primer embarcadero al último y cuál es el trayecto que proporciona este coste mínimo.
Se pide responder con claridad y concisión a las siguientes cuestiones:

a) Secuencia de decisiones (número de decisiones y significado de las mismas) (10%)
b) Función objetivo (10%)
c) Restricciones explícitas e implícitas (20%)
d) Preparar_recorrido_nivel_k, Existe_hermano_nivel_k y Siguiente_hermano_nivel_k (15%)
e) Función Solución (5%)
f) Indicar qué hacen las funciones Correcto y/o Valor si es que fueran necesarias en la solución. Escribir el pseudocódigo de dichas funciones (≡ cómo lo hacen) (20%)
g) Para el siguiente ejemplo, dibujar el árbol de búsqueda que explora la solución expuesta previamente hasta localizar las cuatro primeras soluciones factibles. Numerar los nodos reflejando el orden en el que se visitan, indicar cuándo se realiza una poda y por qué: (20%) N=6 y

C	1	2	3	4	5	6
1	0	10	6	15	23	30
2		0	5	1	11	18
3			0	8	12	25
4				0	6	3
5					0	7
6						0

SOLUCIÓN.-

a) SECUENCIA DE DECISIONES

<x_1, x_2,…, x_S> con $1 \leq S \leq N-1$, por consiguiente, tupla de longitud variable, y donde x_1 indicará el primer embarcadero a visitar desde el embarcadero 1, x_2 indica el segundo embarcadero a visitar y así sucesivamente. Indicar que x_S será N. Como se puede ver sigue el mismo planteamiento que la resolución aplicando Programación Dinámica (pag. 54-55 de las diapositivas del tema 3).

b) FUNCIÓN OBJETIVO

$$minimizar\ C[1][x_1] + \sum_{i=1}^{S-1} C[x_i][x_{i+1}] =$$

$$minimizar\ C[1][x_1] + C[x_1][x_2] + \cdots + C[x_{S-1}][x_s]$$

Como se puede ver es la expresión que figura en las diapositivas del tema 3.

c) RESTRICCIONES EXPLÍCITAS E IMPLÍCITAS

Restricciones explícitas $(\forall i)(x_i \epsilon \{2, \dots, N\}: 1 \leq i \leq S)$

Restricciones implícitas $(\forall i)(x_i < x_{i+1} : 1 \leq i \leq S-1)$

d) PREPARAR_RECORRIDO_NIVEL_K, EXISTE_HERMANO_NIVEL_K Y SIGUIENTE_HERMANO_NIVEL_K

preparar_recorrido_nivel_k x[k] = 1

existe_hermano_nivel_k x[k] < N

siguiente_hermano_nivel_k x[k] = x[k] + 1

e) FUNCIÓN SOLUCIÓN

x[k] = N

f) FUNCIÓN CORRECTO Y FUNCIÓN VALOR

QUÉ HACE.- la función correcto, tras recibir la secuencia de decisiones x y el valor de k correspondiente, devuelve falso si x[k] es inferior o igual a x[k-1]. En cualquier otro caso, devolverá cierto. Indicar que en el caso de la primera decisión (k=1, esto es x_1, siempre es correcta).

PSEUDOCÓDIGO (CÓMO).-

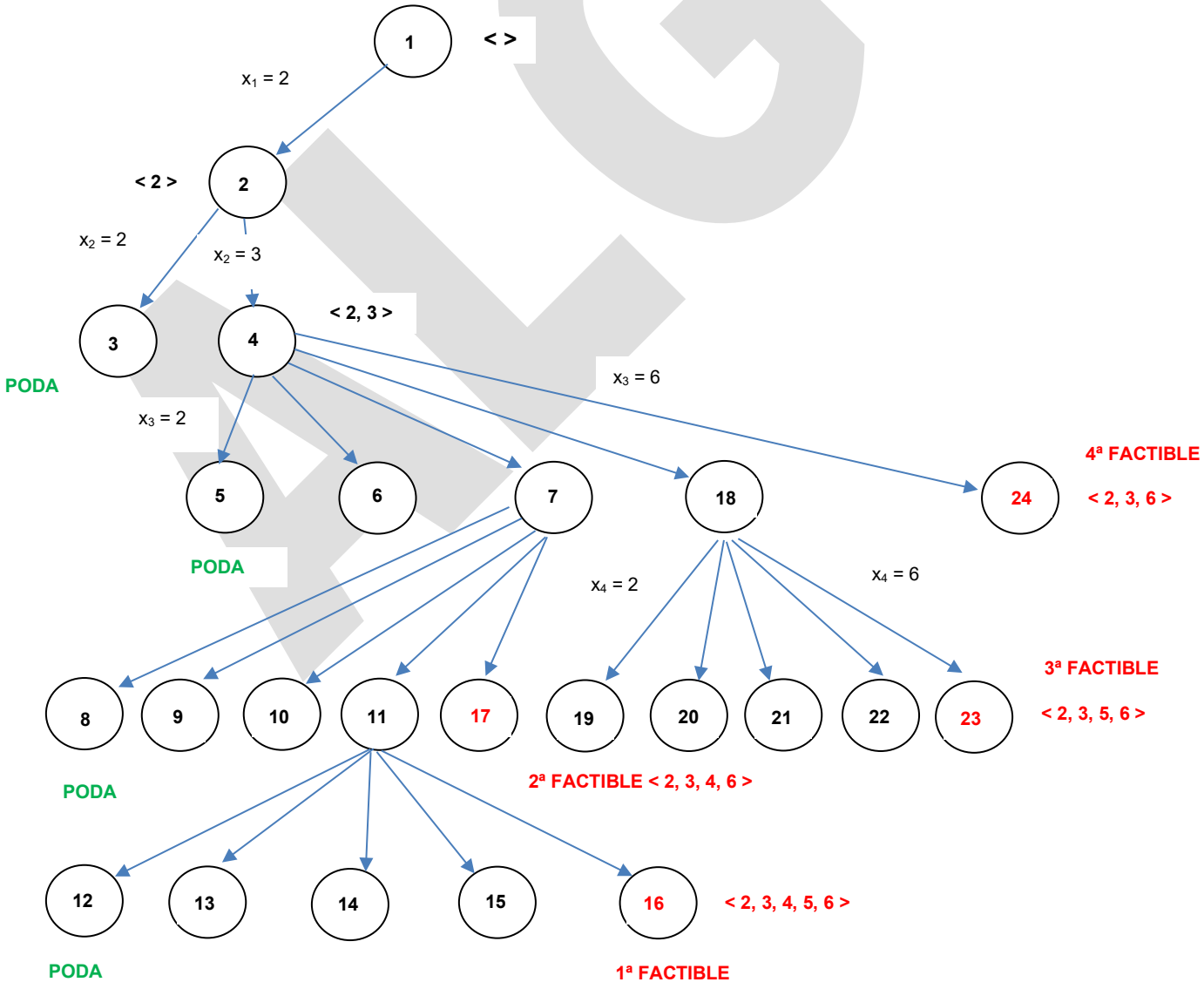
```
Funcion Correcto (x: tupla, k:entero) retorna (b:booleano)
        si ( k = 1 ) retorna verdadero
           sino si ( x[ k ] > x[ k-1 ] ) retorna verdadero
                      sino retorna falso
                fsi
        fsi
ffunción
```

QUÉ HACE.- la función valor, tras recibir la secuencia de decisiones x y el valor de k (recordemos que x[k] es igual a N), devuelve el valor de la función objetivo correspondiente a la secuencia de decisiones x, esto es, la suma de los costes de los diferentes trayectos que se van a realizar a lo largo del río.

PSEUDOCÓDIGO (CÓMO).-

```
Funcion Valor ( C[1..N][1..N]: matriz de enteros, x:tupla, k:entero ) retorna ( b:entero )
        var i, total: entero fvar
        total=C[1][x[1]];
        para i=2 hasta k hacer
                total = total + C[ x[ i-1 ] ][ x[i] ]
        fmientras
        retorna total
ffunción
```

g) ÁRBOL DE BÚSQUEDA



En los nodos 3, 5, 6, 8, 9, 10, 12, 13, 14, 15, 19, 20, 21, 22 se realiza una poda porque representan un estado que visita un embarcadero igual o menor que el embarcadero anterior. Ejemplos:

Nodo 2: < 2, 2>
Nodo 5: < 2, 3, 2 >
Nodo 8: < 2, 3, 4, 2 >
Nodo 20: < 2, 3, 5, 3 >

Nota.- Podríamos plantear como restricciones explícitas que las alternativas a valorar sean los embarcaderos posteriores al embarcadero actual (igual que cuando se resolvió este ejercicio a través de Programación Dinámica en clase), esto es,

$$(\forall i)(x_i \epsilon \{x_{i-1}+1, \dots, N\}: 1 \leq i \leq S).$$

De ese modo,

preparar_recorrido_nivel_k x[k] = x[k-1]

Así, para el ejemplo de 6 embarcaderos del enunciado esta mejora de la solución implica que el árbol de búsqueda tiene 32 nodos, frente a los 81 de la solución anterior.

La solución de este ejercicio también se puede plantear como una secuencia de decisiones de tamaño fijo, es decir, $<x_1, x_2, \dots, x_N>$ donde x_i indicaría si se para en el embarcadero i o no. De ese modo, cada x_i tomaría dos posibles valores, 0 o 1. El 0 indicará que no se para en el embarcadero i y 1 indicará que sí se para en el embarcadero i. Podríamos fijar x_1 y x_N con valor 1 ya que son los embarcaderos de origen y destino del problema, de uno de sale y al otro se ha de llegar. Obviamente, el resto de apartados (del b al g) serían diferentes a la resolución que se ha mostrado anteriormente.

3) ESQUEMA VORAZ [1.5 puntos] Se pide resolver el problema anterior utilizando una estrategia voraz. Se deberá contestar de forma clara y concisa a las siguientes cuestiones: candidatos (10%), criterio de selección razonable y razonado (10%), función factible (10%), función es_solución (10%) y mostrar, etapa a etapa, cómo dicha estrategia voraz construye la solución para el ejemplo que aparece en el ejercicio anterior (50%).

CANDIDATOS

Siguiendo la solución expuesta en el ejercicio anterior, la secuencia de decisiones será $<x_1, x_2, \dots, x_s>$ donde x_1 indicará el primer embarcadero a visitar desde el embarcadero 1, x_2 indica el segundo embarcadero a visitar y así sucesivamente. Indicar que x_s será N.

Los candidatos por tanto serán todos los embarcaderos excepto el embarcadero 1. No todos los embarcaderos formarán parte de la solución necesariamente.

CRITERIO DE SELECCIÓN

En cada etapa i se seleccionará aquel embarcadero de los embarcaderos posteriores al que hayamos seleccionado en la etapa anterior (llamémosle j) cuyo coste desde el embarcadero j a él sea menor. Dicho embarcadero se incorpora a la solución.

FUNCIÓN FACTIBLE

El criterio de selección propuesto hace que todas decisiones sean factibles ya que se garantiza que el embarcadero seleccionado será posterior al que nos encontremos. La función factible tendrá por tanto tratamiento vacío.

FUNCIÓN ES_SOLUCIÓN

Habremos encontrado la solución cuando el embarcadero elegido sea el embarcadero N, por lo que habremos llegado al fin del trayecto.

MOSTRAR ETAPA A ETAPA COMO LA ESTRATEGIA VORAZ CONSTRUYE LA SOLUCIÓN

N = 6 y

C	1	2	3	4	5	6
1	0	10	6	15	23	30
2		0	5	1	11	18
3			0	8	12	25
4				0	6	3
5					0	7
6						0

Etapa	Candidato seleccionado	Solución	Valor total
inicial	--	< -, -, - >	0
1	3	< 3, -, - >	0+6=6
2	4	< 3, 4, - >	6+8=14
3	6	< 3, 4, 6 >	14+3=17

La solución obtenida a través de la estrategia voraz expuesta es < 3, 4, 6 > y su coste es 17.

Curso Académico 2017-2018

Nota.- en el curso académico 2017/2018 y anteriores, se realizaba por separado la evaluación de Programación Dinámica (finales de noviembre) y la evaluación Backtracking y Esquema Voraz (enero, según fecha fijada en el calendario oficial de exámenes), de ahí los puntos vinculados a cada ejercicio.

PROGRAMACIÓN DINÁMICA [10 puntos] El próximo día 5 de diciembre se va a disputar el partido Chelsea vs Atlético de Madrid correspondiente a la jornada 6 de la Champions League. Los organizadores de dicho partido han concedido al equipo español un cupo de E entradas. El Atlético de Madrid nos ha encomendado gestionar sus entradas entre sus peñas. Se sabe que cada peña del equipo, hay P, ha solicitado un número de entradas, siendo este número S_k con $1 \leq k \leq P$, donde S_k indica las entradas solicitadas por la peña k. El número total de solicitudes que llegaron al club excede con creces las E entradas de las que se dispone, por lo que se ha tomado la siguiente decisión: se concederán las entradas solicitadas por una peña siempre y cuando todos sus solicitantes tengan cabida en el estadio. Teniendo en cuenta que el objetivo del Atlético de Madrid es maximizar la venta de sus E entradas, se trata de resolver el problema mediante Programación Dinámica indicando a qué peña se ha de vender entradas con objeto de vender el mayor número de ellas.

Se pide responder con claridad y concisión a las siguientes cuestiones:

- Secuencia de decisiones: tamaño y significado de la decisión i-ésima (10%)
- Función Objetivo (10%)
- Restricciones (10%)
- Demostración del principio de optimalidad (15%)
- Función recursiva y cómo se realiza la primera llamada a la función (25%)
- Árbol de llamadas completo para el siguiente ejemplo (10%)

$$E = 10, P = 4, S[1..4] = \{ 3, 5, 6, 3 \}$$

- Estructuras de almacenamiento necesarias y rellenado de las mismas para el ejemplo anterior mostrando únicamente los cálculos necesarios para resolver los subproblemas que aparecen en el árbol del apartado anterior. Concluir con el valor que optimiza la función objetivo, así como la forma de alcanzarlo (20%)

Antes de mostrar la solución del ejercicio conviene indicar que es igual que el problema de la mochila visto en clase:

- Disponemos de P peñas y se trata de decidir a qué peñas se venderán las entradas. Del mismo modo que disponíamos de N objetos y se trataba de decidir qué objetos se introducían en la mochila. La secuencia de decisiones será de tamaño P (tantas decisiones como peñas) y la decisión con respecto a cada peña consiste en indicar si a dicha peña se le dan las entradas solicitadas o no => Decisiones.
- No todas las peñas recibirán entradas ya que no hay suficientes entradas para todas, del mismo modo que no todos los objetos se podían introducir en la mochila ya que no todos cabían en ella => Restricciones.
- Se trata de vender/distribuir el mayor número de entradas igual que en el problema de la mochila se trataba de obtener el máximo beneficio => Función objetivo.

SOLUCIÓN.-

Secuencia de decisiones: tamaño y significado de la decisión i-ésima (10%)

Número fijo de decisiones = P

$<d_1, d_2, \ldots, d_P>$

d_i representa si se vende entradas a la peña i o no, representaremos con 0 el hecho de que no y 1 el hecho de que sí.

Función Objetivo (10%)

$$\text{maximizar} \sum_{i=1}^{P} S[i] * di$$

Restricciones (10%)

$$\sum_{i=1}^{P} S[i] * d_i \leq E$$

Demostración del principio de optimalidad (15%)

Sea < d_1, d_2, …, d_P > la solución óptima del problema de distribuir E entradas entre P peñas (de la 1 a la P) obteniendo la mayor venta. Denominaremos a dicho problema Venta(P,E). El valor

$$\sum_{i=1}^{P} S[i] * di$$

cumpliéndose además que

$$\sum_{i=1}^{P} S[i] * d_i \leq E$$

Supongamos que prescindimos de la última decisión d_P. La cual indica si a la peña P se le venden entradas o no. La subsecuencia que queda, esto es, < d_1, d_2, …, d_{P-1} > es la solución óptima para el problema asociado, es decir, para el subproblema

Venta(P-1, E - S[P]*d_P),

que es el subproblema de distribuir E - S[P]*d_P entradas entre P-1 peñas (de la 1 a la P-1) obteniendo la mayor venta.

El valor asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{P-1} S[i] * di$$

cumpliéndose además que

$$\sum_{i=1}^{P-1} S[i] * d_i \leq E - S[P] * d_P$$

Supongamos que < d_1, d_2, …, d_{P-1} > **NO** es la solución óptima para el problema asociado sino que existe otra solución < d^*_1, d^*_2, …, d^*_{P-1} > que mejora su valor. Esto querrá decir que

$$\sum_{i=1}^{P-1} S[i] * di \; < \; \sum_{i=1}^{P-1} S[i] * d_i^* \qquad (1)$$

cumpliéndose además que

$$\sum_{i=1}^{P-1} S[i] * d_i^* \leq E - S[P] * d_P$$

Pero entonces sumando S[P]*d_P a ambos lados de la desigualdad recogida en (1), se cumpliría que:

$$\sum_{i=1}^{P-1} S[i] * di + S[P] * d_P < \sum_{i=1}^{P-1} S[i] * d_i^* + S[P] * d_P$$

lo que significa que la secuencia < d^*_1, d^*_2, …, d^*_{P-1}, d_P > mejoraría el resultado de la secuencia <d_1, d_2, …, d_P > para el problema Venta(P,E), lo cual contradice la hipótesis de partida, pues < d_1, d_2, …, d_P > era la solución óptima de dicho problema. En conclusión, se cumple el principio de optimalidad.

Ecuación recursiva y primera llamada a la función (25%)

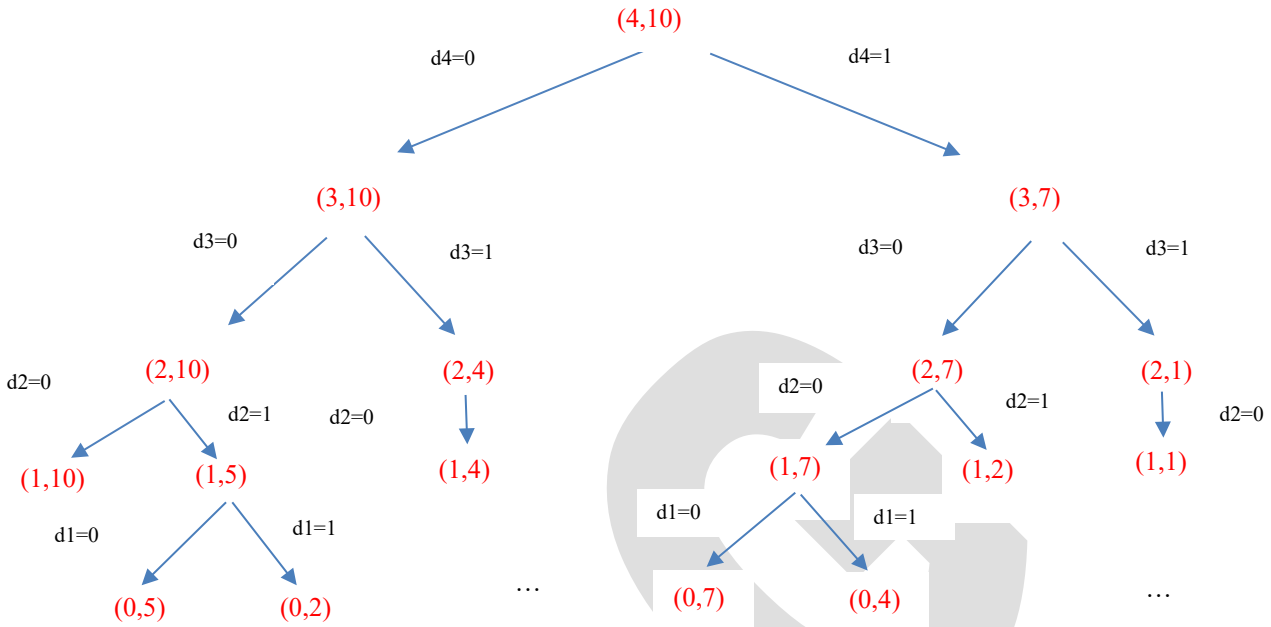
$$Venta\ (p,e) = \begin{cases} 0 & si\ p = 0 \\ Venta(p-1,e) & si\ p > 0\ y\ e < S[p] \\ \max_{d_p \in \{0,1\}} \{Venta(p-1, e - d_p * S[p]) + S[p] * d_p\} & si\ p > 0\ y\ e \geq S[p] \end{cases}$$

donde Venta(p,e) devuelve el máximo número de entradas que se pueden distribuir entre las p peñas (del 1 al p) de las e entradas posibles.

La primera llamada a la función se producirá como Venta(P,E), siendo P el número de peñas totales y E el número de entradas disponibles al comienzo.

Árbol de llamadas para el ejemplo dado (10%)

E = 10, P = 4, S[1..4] = { 3, 5, 6, 3 }



Estructuras de almacenamiento necesarias y rellenado de las mismas para el ejemplo anterior, mostrando únicamente los cálculos necesarios para resolver los subproblemas que aparecen en el árbol. Concluir con el valor que optimiza la función objetivo, así como la forma de alcanzarlo (20%)

Dado que nuestra función recursiva tiene dos parámetros, se precisan dos estructuras bidimensionales. Estas tendrán P+1 filas y E+1 columnas, esto es debido a que $0 \leq p \leq P$ y $0 \leq e \leq E$.

En una de las estructuras, Vmax, se guardará el valor asociado a cada subproblema, esto es, el máximo número de entradas distribuidas. En la otra estructura, Dec, se almacenará la alternativa (0 o 1) que proporcione el máximo para cada subproblema.

Inicializar las estructuras de almacenamiento con los resultados de los problemas triviales, corresponde a rellenar la fila 0 de cada matriz con 0, dado que los problema triviales son del tipo Venta(0,e), esto es, Vmax[0][e]=0 con $0 \leq x \leq 10$ y Dec[0][e]=0 con $0 \leq x \leq 10$

Completar el rellenado de la matrices: dado que para solucionar el subproblema Venta(p,e) se precisa conocer la solución de los subproblemas del tipo Venta(p-1,x) donde $x \leq e$, ambas matrices se rellenan por filas en sentido creciente, desde la fila 1 hasta la fila P. (=> Dependencia de los subproblemas).

Vmax[1][1]=Vmax[0][1]=0 Dec[1][1]=0
Vmax[1][2]=Vmax[0][2]=0 Dec[1][2]=0
Vmax[1][4]=max { Vmax[0][4], Vmax[0][1]+3 }=max{0, 0+3}=3 Dec[1][4]=1
Vmax[1][5]=max { Vmax[0][5], Vmax[0][2]+3 }=max{0, 0+3}=3 Dec[1][5]=1
Vmax[1][7]=max { Vmax[0][7], Vmax[0][4]+3 }=max{0, 0+3}=3 Dec[1][7]=1
Vmax[1][10]=max { Vmax[0][10], Vmax[0][7]+3 }=max{0, 0+3}=3 Dec[1][10]=1

Vmax[2][1]=Vmax[1][1]=0 Dec[2][1]=0
Vmax[2][4]=Vmax[1][4]=3 Dec[2][4]=0
Vmax[2][7]=max{Vmax[1][7],Vmax[1][2]+5}=max{3,0+5}=5 Dec[2][7]=1
Vmax[2][10]= max{Vmax[1][10],Vmax[1][5]+5}=max{3,3+5}=8 Dec[2][10]=1

Vmax[3][7]= max{Vmax[2][7],Vmax[2][1]+6}=max{5,0+6}=6 Dec[3][7]=1
Vmax[3][10]= max{Vmax[2][10],Vmax[2][4]+6}=max{8,3+6}=9 Dec[3][10]=1

Vmax[4][10]= max{Vmax[3][10],Vmax[3][7]+3}=max{9,6+3}=9 Dec[4][10]=0

Las matrices Vmax y Dec tras el proceso de rellenado quedarían del siguiente modo:

Vmax	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	3	3	3	3	3	3	3	3
2	0	0	0	3	3	5	5	5	8	8	8
3	0	0	0	3	3	5	6	6	8	9	9
4	0	0	0	3	3	5	6	6	8	9	9

El mayor número de entradas distribuido estará en la posición [P][E] de la matriz Vmax, esto es, 9.

Dec	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	1	1	1	1	1	1	1
2	0	0	0	0	0	1	1	1	1	1	1
3	0	0	0	0	0	0	1	1	0	1	1
4	0	0	0	0	0	0	0	0	0	0	0

La secuencia óptima de decisiones se obtiene recorriendo determinadas posiciones de la matriz Dec. Comenzaríamos por la posición [4][10], el valor ahí almacenado correspondería a d_4, es decir, d_4 será 0. Seguidamente iríamos a Dec[4-1][10-d_4*S[4]]=Dec[3][10], ahí se encontrará el valor correspondiente a d_3 , d_3 será 1 por tanto. Seguidamente iríamos a Dec[3-1][10-d_3*S[3]]=Dec[2][4], ahí se encontrará el valor correspondiente a d_2 , d_2 será 0 por tanto. Por último, iríamos a Dec[2-1][4-d_2*S[2]]=Dec[1][4], ahí se encontrará el valor correspondiente a d_1 , d_1 será 1. De tal modo que la secuencia de decisiones óptima corresponde a: $< d_1, d_2, d_3, d_4 > = < 1, 0, 1, 0 >$

[Backtracking 8 puntos] Una escuela universitaria debe realizar la planificación de un conjunto de E exámenes para un día concreto en la próxima convocatoria de julio. Para ello dispone de A aulas cuyas capacidades están recogidas en el vector Capacidad[1..A], siendo Capacidad[i] la capacidad del aula i-ésima. También se conoce el número de alumnos que se van a presentar a cada examen, dicha información está recogida en el vector NumAlumnos[1..E], donde NumAlumnos [i] es el número de alumnos que asistirán al examen i-ésimo.

A partir de estos datos se trata de diseñar un algoritmo utilizando la metodología *Backtracking* que indique a la escuela la forma de organizar los exámenes de ese día de tal modo que se minimice el número de aulas a utilizar. Habrá de tenerse en cuenta que no es posible dividir un examen en varias aulas y que en un aula se podrán meter varios exámenes si es que caben.

Se pide responder con claridad y concisión a las siguientes cuestiones:

a) Secuencia de decisiones (nº de decisiones y significado de las mismas) (10%)
b) Expresar la función objetivo en lenguaje natural (5%)
c) Restricciones explícitas (10%)
d) Expresar las restricciones implícitas en lenguaje natural (10%)
e) Tipo de solución buscada (Todas las factibles/Una factible/Óptima) (5%)
f) Preparar_recorrido_nivel_k (5%)
g) Existe_hermano_nivel_k (5%)
h) Siguiente_hermano_nivel_k (5%)
i) Función Solución (5%)
j) Indicar qué hacen las funciones Correcto y/o Valor si es que fueran necesarias en tu solución. Escribir el pseudocódigo de dichas funciones (≡ cómo lo hacen) (20%)
k) Para el siguiente ejemplo, dibujar el árbol de búsqueda que se explora hasta localizar la primera solución factible. Numerar los nodos reflejando el orden en el que se visitan, indicar cuándo se realiza una poda y por qué: (20%):

A = 4, E = 3, Capacidad[1..4] = { 20, 10, 40, 10 } y NumAlumnos[1..3] = { 20, 20, 20 }

SOLUCIÓN.-

SECUENCIA DE DECISIONES

$<x_1,x_2,\ldots,X_E>$ donde xi indica el aula en la que se realizará el examen i-ésimo.

FUNCIÓN OBJETIVO

Minimizar el número de aulas utilizadas. Esto implica que deberemos contabilizar las diferentes aulas que se van a usar, no importando, claro está, que un aula pueda aparecer varias veces en la secuencia de decisiones. Esto únicamente indicará que en ese aula se realizan varios exámenes.
Ejemplos:

< 2, 4, 2, 1 > En este caso, se utilizan 3 aulas: aula 1, aula 2 y aula 4.
< 5, 5, 2, 5> En este caso, se usan 2 aulas: aula 5 y aula 2.

RESTRICCIONES EXPLÍCITAS

$$(\forall i)(x_i \epsilon \{1,2,\ldots,A\}: 1 \leq i \leq E)$$

RESTRICCIONES IMPLÍCITAS

La suma de los alumnos de los diferentes exámenes destinados a un mismo aula (mismo valor en la secuencia de decisiones) no tiene que exceder la capacidad de dicha aula.

TIPO DE SOLUCIÓN

La solución óptima.

PREPARAR_RECORRIDO_NIVEL_K

x[k] = 0

EXISTE_HERMANO_NIVEL_K

x[k] < A

SIGUIENTE_HERMANO_NIVEL_K

x[k] = x[k] + 1

SOLUCIÓN

k = E

FUNCIÓN CORRECTO

La función Correcto necesita la secuencia de decisiones (x), la posición hasta la que se ha completado la secuencia de decisiones (k) y los vectores Capacidad y NumAlumnos.
La función Correcto produce un valor booleano: Verdadero, si la suma de todos los alumnos que realizan su examen en el aula x[k] no superan la capacidad del aula; Falso, en caso contrario.

```
Funcion Correcto ( x : tupla, k : entero; Capacidad[1..A], NumAlumnos[1..E] : vector de enteros)
                                                                    retorna (b:booleano)
        var i : entero; fvar
        Total_alumnos = NumAlumnos[k];
        i = 0;
        mientras (i < k-1) hacer
                i = i + 1;
                si ( x[ i ] = x[ k ] ) entonces
                        Total_alumnos = Total_alumnos + NumAlumnos[i];
                fsi
        fmientras
        si Total_alumnos ≤ Capacidad [x[k]] entonces retorna Verdadero sino retorna Falso fsi
ffunción
```

FUNCIÓN VALOR

La función Valor necesita la secuencia de decisiones (x) la cual estará completa, lo que quiere decir que k=E. La función Valor produce el número de aulas diferentes que se han utilizado en la organización de los exámenes. Para ello se utilizará un vector auxiliar (Aula_usada) en el que se almacenará un 0 en la posición i, en el caso que el aula i no se utilice y se almacenará un 1 en la posición i, en el caso que el aula i sí se utilice en la planificación de exámenes.

```
Funcion Valor ( x : tupla, k : entero) retorna (t : entero)
        var
           i, Total_aulas: entero;
           Aula_usada[1..A]: vector de enteros
        fvar
        Total_aulas = 0;
        para i=1 hasta A hacer
                Aula_usada[i]=0;
        fpara
        para i=1 hasta k hacer
                Aula_usada[x[i]]=1;
        fpara
        para i=1 hasta A hacer
                Total_aulas = Total_aulas  +  Aula_usada[i];
        fpara
        retorna Total_aulas
ffunción
```

Complejidad temporal: $T(A,E) \in \theta(max(A,E))$
Complejidad espacial: $T(A) \in \theta(A)$

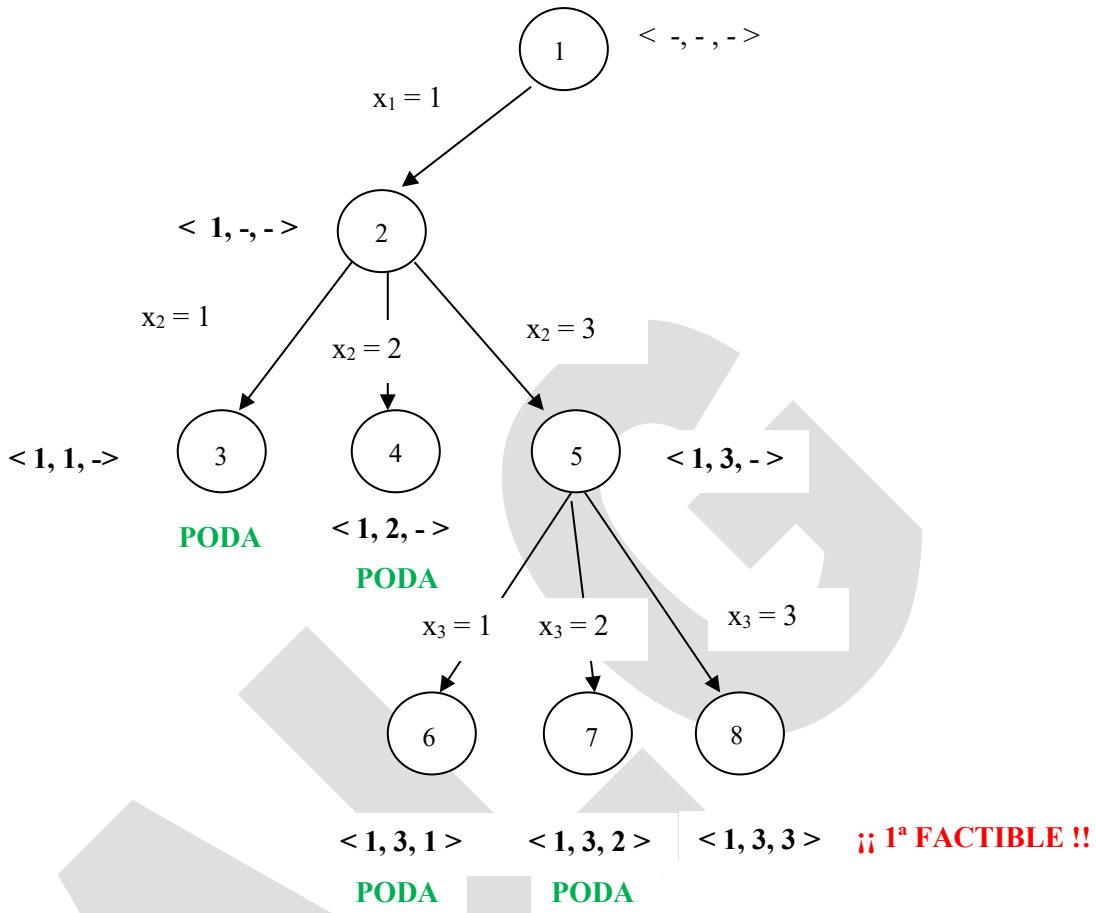
Otra posible solución para la función Valor, sin utilizar un vector de tamaño A, sería contabilizar únicamente la última aparición en el vector x de cada aula utilizada. Esta solución implica el uso de dos bucles anidados.

```
Funcion Valor ( x : tupla, k : entero) retorna (t : entero)
        var
           i, Total_aulas: entero;
        …..ultima_aparicion : booleano
        fvar
        Total_aulas = 0
        para i = 1 hasta k hacer
             unica_aparicion = Verdadero;
             j = i
             mientras j < k y unica_aparicion hacer
                  j = j + 1
                  si ( x[i] = x[j] ) unica_aparicion = Falso fsi
             fmientras
             si ( unica_aparicion ) Total_aulas = Total_aulas  +  1  fsi
        fpara
        retorna Total_aulas
ffunción
```

Esta segunda versión de la función Valor presenta mejor y peor caso. Si contabilizamos solamente las operaciones básicas que suceden en la instancia peor (cada aula aparece una única vez), la complejidad temporal es:

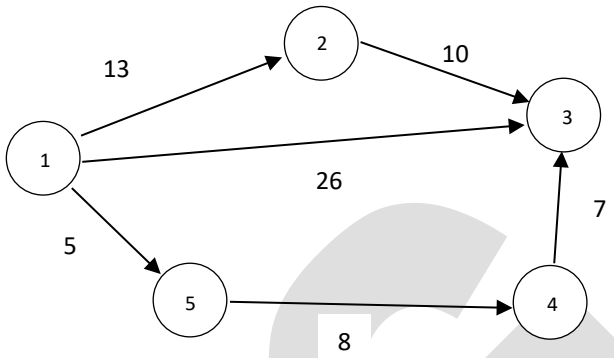
Complejidad temporal: $T_{PC}(E) \in \theta(E^2)$

ÁRBOL DE BÚSQUEDA



Nodo 3: Los alumnos de los exámenes 1 y 2 no caben en el aula 1
Nodo 4: Los alumnos del examen 2 no caben en el aula 2
Nodo 6: Los alumnos de los exámenes 1 y 3 no caben en el aula 1
Nodo 7: Los alumnos del examen 3 no caben en el aula 2

[Esquema voraz 2 puntos] Dado el siguiente grafo, rellenar la tabla adjunta indicando paso a paso cómo el
algoritmo de Dijkstra encuentra los caminos mínimos desde el nodo 1 a los restantes nodos del grafo.



etapa	Nodo seleccionado	Nodos ya seleccionados	Nodos aún no seleccionados	Vector de distancias (D) D[2…5]	Vector de predecesores (P) P[2…5]
inicial	---	{ 1 }	{ 2, 3, 4, 5 }	[13, 26, ∞, 5]	[1, 1, -, 1]
1	5	{ 1, 5 }	{ 2, 3, 4 }	[13, 26, 13, 5]	[1, 1, 5, 1]
2	2 (*)	{ 1, 5, 2 }	{ 3, 4 }	[13, 23, 13, 5]	[1, 2, 5, 1]
3	4	{ 1, 5, 2, 4 }	{ 3 }	[13, 20, 13, 5]	[1, 4, 5, 1]

A partir de la solución obtenida, explicar con detalle cuál es el coste del camino mínimo desde el nodo 1 al nodo 3 y por donde discurre dicho camino mínimo.

SOLUCIÓN.-

La longitud del camino mínimo desde el nodo 1 hasta el nodo 3 está recogida en la posición 3 del vector D, esto es, 20. Para saber por donde discurre dicho camino mínimo hay que empezar acudiendo a la posición 3 del vector P. Dicho valor nos indica cuál es el nodo predecesor al nodo 3 (en dicho camino mínimo), esto es, 4.

4 → 3

Seguidamente habría que acudir a la posición P[4], que nos indica cuál es el nodo predecesor al nodo 4, es decir, 5.

5 → 4 → 3

Y por último habría que ir a la posición P[5], que nos indica cuál es el nodo predecesor al nodo 4, siendo 1.

1 → 5 → 4 → 3

(*) En la etapa 2 se podría haber seleccionado el nodo 4 en lugar del nodo 2, ya que en ambos el coste del camino desde el nodo 1 es 13. La solución final sería la misma.

Curso Académico 2016-2017

[PROGRAMACIÓN DINÁMICA 10 puntos].- Joffrey I Rey de los Andalos, Lord protector del Reino, se dispone a expandir su corona. Para ello tiene en el punto de mira una lista de R reinos para destruir y hacerse con sus riquezas. Se sabe que cada uno de esos reinos ofrecerá una determinada resistencia. Dicha resistencia está recogida en el vector Res[1..R], donde Res[k], con $1 \leq k \leq R$, es la resistencia que ofrecerá el reino k. Se sabe que esa resistencia puede ser vencida con un número igual de soldados. Cada reino le aportaría al Rey Joffrey un determinado enriquecimiento gracias a los recursos de los que dispone: oro, fuego Valyrio, armas, … y este enriquecimiento está cuantificado y almacenado en Enri[1..R] donde Enri[k], con $1 \leq k \leq R$, es el enriquecimiento que se conseguirá al derrotar al reino k.

Como todos sabemos el Rey Joffrey no tiene inteligencia suficiente para solucionar el problema, así que ha pedido ayuda a su tío Tyron Lannister, el cual debe diseñar una estrategia óptima para la conquista sabiendo que no dispone de suficientes soldados en el reino para poder destruir los R reinos simultáneamente.

Con todo ello, Tyron te traslada a ti el diseño de un algoritmo basado en Programación Dinámica que proporcione la mejor estrategia para una conquista simultánea de los reinos objetivo, indicando qué reinos deberán ser destruidos para conseguir el mayor enriquecimiento sin utilizar más soldados de los disponibles, siendo éstos, S.

Se pide responder con claridad y concisión a las siguientes cuestiones:

- Secuencia de decisiones: tamaño y significado de la decisión i-ésima (5%)
- Función Objetivo (10%)
- Restricciones (10%)
- Demostración del principio de optimalidad (15%)
- Ecuación recursiva y primera llamada a la función (30%)
- Árbol de llamadas para un ejemplo (10%)
- Explicar de manera concisa y clara: tipo de estructuras de almacenamiento elegidas, sus dimensiones, cómo se rellenan y cómo se obtiene la solución (20%)

Antes de mostrar la solución del ejercicio conviene indicar que es igual que el problema de la mochila visto en clase:

- Disponemos de R reinos y se trata de decidir qué reinos deben ser destruidos. Del mismo modo que disponíamos de N objetos y se trataba de decidir qué objetos se debían introducir en la mochila. La secuencia de decisiones será de tamaño R (tantas decisiones como reinos) y cada decisión con respecto a cada reino consiste en indicar si dicho reino se destruye o no => Decisiones.
- No todos los reinos se pueden destruir ya que no hay suficientes soldados para llevar a cabo la conquista simultánea, del mismo modo que no todos los objetos se podían introducir en la mochila ya que no todos cabían en ella => Restricciones.
- Se trata de obtener el máximo enriquecimiento igual que en el problema de la mochila se trataba de obtener el máximo beneficio => Función objetivo.
- El enriquecimiento aportado por el reino k se consigue al derrotar al reino k, del mismo modo que el beneficio aportado por el objeto k se obtenía si dicho objeto era introducido en la mochila.

SOLUCIÓN.-

Secuencia de decisiones: tamaño y significado de la decisión i-ésima (5%)

Número fijo de decisiones = R

$<d_1, d_2, \ldots, d_R>$

d_i representa si se destruye el reino i o no, representaremos con 0 el hecho de que no y 1 el hecho de que sí.

Función Objetivo (10%)

$$\text{maximizar} \sum_{i=1}^{R} Enri[i] * di$$

Restricciones (10%)

$$\sum_{i=1}^{R} Res[i] * d_i \leq S$$

Demostración del principio de optimalidad (15%)

Sea < d_1, d_2, …, d_R > la solución óptima del problema de conquistar R reinos con S soldados disponibles obteniendo el máximo enriquecimiento. Denominaremos a dicho problema Destruccion(R,S). El valor asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{R} Enri[i] * di$$

cumpliéndose además que

$$\sum_{i=1}^{R} Res[i] * d_i \leq S$$

Supongamos que prescindimos de la última decisión d_R. La subsecuencia que queda, esto es, < d_1, d_2, …, d_{R-1} > es la solución óptima para el problema asociado, es decir, para el problema

Destruccion(R-1, S - Res[R]*d_R),

que es el subproblema de conquistar R-1 reinos con S - Res[R]*d_R soldados disponibles obteniendo el máximo enriquecimiento.

El valor asociado a dicha secuencia de decisiones es

$$\sum_{i=1}^{R-1} Enri[i] * di$$

cumpliéndose además que

$$\sum_{i=1}^{R-1} Res[i] * d_i \leq S - Res[R] * d_R$$

Supongamos que < d_1, d_2, …, d_{R-1} > NO es la solución óptima para el problema asociado sino que existe otra solución < d^*_1, d^*_2, …, d^*_{R-1} > que mejora su valor. Esto querrá decir que

$$\sum_{i=1}^{R-1} Enri[i] * di \; < \; \sum_{i=1}^{R-1} Enri[i] * d_i^* \qquad (1)$$

cumpliéndose además que

$$\sum_{i=1}^{R-1} Res[i] * d_i^* \leq S - Res[R] * d_R$$

Pero entonces sumando Enri[R]*d_R a ambos lados de la desigualdad recogida en (1), se cumpliría que:

$$\sum_{i=1}^{R-1} Enri[i] * di + Enri[R] * d_R < \sum_{i=1}^{R-1} Enri[i] * d_i^* + Enri[R] * d_R$$

lo cual significa que la secuencia < d^*_1, d^*_2, …, d^*_{R-1}, d_R > mejoraría el resultado de la secuencia <d_1, d_2, …, d_R > para el problema Destruccion(R,S) lo cual contradice la hipótesis de partida, pues < d_1, d_2, …, d_R > era la solución óptima de dicho problema. En conclusión, se cumple el principio de optimalidad.

Ecuación recursiva y primera llamada a la función (30%)

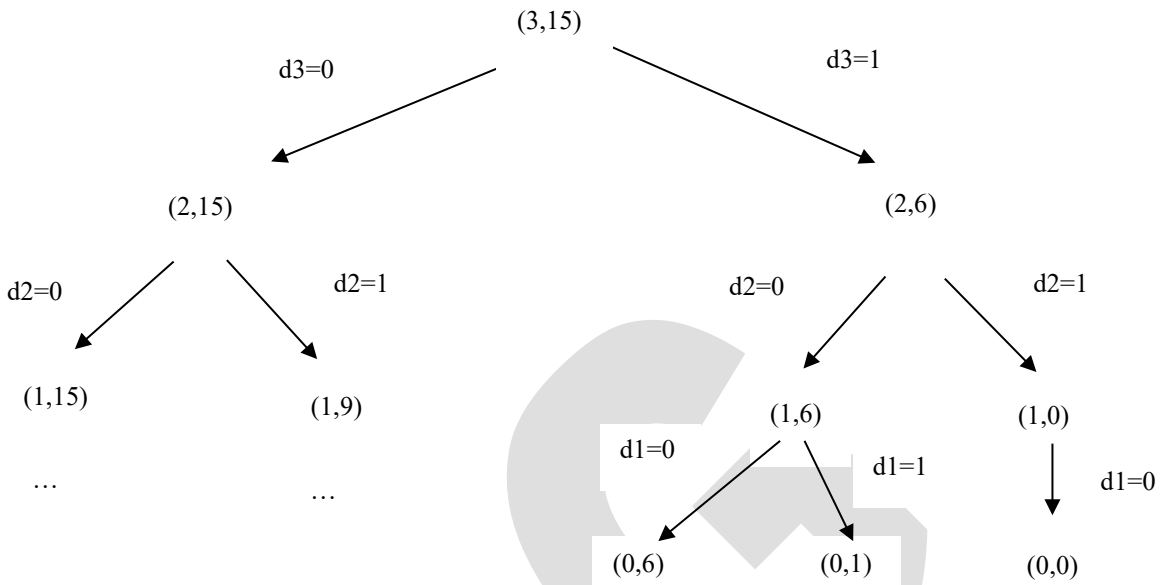
$$Destruccion\ (r,s) = \begin{cases} 0 & si\ r = 0 \\ Destruccion(r-1,s) & si\ r > 0\ y\ s < Resis[r] \\ \max_{d_r \in \{0,1\}} \{Destruccion(r-1, s - d_r * Resis[r]) + Enri[r] * d_r\} & si\ r > 0\ y\ s \geq Resis[r] \end{cases}$$

donde Destruccion(r,s) devuelve el máximo enriquecimiento asociado a la pretensión de destruir r reinos (del 1 al r) con s soldados posibles.

La primera llamada a la función se producirá como Destrucción(R,S), siendo R el número de reinos totales y S el número de soldados disponibles al comienzo.

Árbol de llamadas para un ejemplo (10%)

R=3, S=15, Enri[1..3]={24,40,38}, Res[1..3]={5,6,9}



Explicar de manera concisa y clara: tipo de estructuras de almacenamiento elegidas, sus dimensiones, cómo se rellenan y cómo se obtiene la solución (20%)

Dado que nuestra función recursiva tiene dos parámetros, se precisan dos estructuras bidimensionales. Éstas tendrán R+1 filas y S+1 columnas, esto es debido a que $0 \leq r \leq R$ y $0 \leq s \leq S$.

En una de las estructuras, Emax, se guardará el valor asociado a cada subproblema, esto es, el máximo enriquecimiento obtenido. En la otra estructura, Dec, se almacenará la alternativa (0 o 1) que proporcione el máximo para cada subproblema.

Inicializar las estructuras de almacenamiento con los resultados de los problemas triviales, corresponde a rellenar la fila 0 de cada matriz con 0, dado que los problema triviales son del tipo Destruccion(0,s).

Completar el rellenado de la matrices: dado que para solucionar el subproblema Destruccion(r,s) se precisa conocer la solución de los subproblemas del tipo Destruccion(r-1,x), ambas matrices se rellenan por filas en sentido creciente, desde la fila 1 hasta la fila R. (=> Dependencia de los subproblemas).

El enriquecimiento máximo buscado estará en la posición [R][S] de la matriz Emax. La secuencia óptima de decisiones se obtiene recorriendo determinadas posiciones de la matriz Dec. Comenzaríamos por la posición [R][S], el valor ahí almacenado correspondería a d_R. Seguidamente iríamos a Dec[R-1][S-d_R*Res[R]], ahí se encontrará el valor correspondiente a d_{R-1} y así sucesivamente dando valor a: d_{R-2} , d_{R-3}, …, d_2 , d_1 .

[Backtracking 7 puntos] La empresa ALG_APPs quiere realizar una aplicación para el juego del amigo invisible y nos encomienda su diseño. Los datos a manejar son varios. En primer lugar, el número de las personas que integran el juego, denominaremos P a ese número. En segundo lugar, se dispone de la información de los excluidos, esto es, se conoce si alguien no debe regalar a otra persona. Dicha información está recogida en el vector Excluir[1..P], donde Excluir[k] indica a qué persona no debe regalar la persona k. Los valores contenidos en dicho vector son valores comprendidos entre 0 y P. Indicar que si Excluir[k] fuese 0, querría decir que la persona k puede regalar a cualquiera de los demás de la pandilla. Y por último, también se va a manejar el afecto que parece haber entre las personas que integran el juego. Dicha información se encuentra almacenada en la matriz Afecto[1..P][1..P] , donde Afecto[i][j] es un número ($\geq$0) que representa el afecto que tiene la persona i hacia la persona j, con 1$\leq$ i , j $\leq$ P.

Se pide diseñar un algoritmo de Vuelta_Atrás (*Backtracking*) que genere la forma de llevar a cabo el juego del amigo invisible, esto es, a quien debe regalar cada uno (teniendo en consideración todo lo relatado en el párrafo anterior) de manera que se maximice la suma de los afectos en los casos que haya un regalo de por medio, es decir, si resulta que la persona 3 tiene que regalar a la persona 7 entonces el afecto que se contabiliza es Afecto[3][7].

Se pide responder con claridad y concisión a las siguientes cuestiones:

a) Secuencia de decisiones (nº de decisiones y significado de las mismas) (10%)
b) Función objetivo (10%)
c) Restricciones explícitas (5%)
d) Restricciones implícitas (15%)
e) Tipo de solución buscada (Todas las factibles/Una factible/Óptima) (5%)
f) Preparar_recorrido_nivel_k (5%)
g) Existe_hermano_nivel_k (5%)
h) Siguiente_hermano_nivel_k (5%)
i) Función Solución (5%)
j) Indica qué es lo que hacen las funciones Correcto y Valor si es que fueran necesarias en tu solución. Escribir el pseudocódigo de dichas funciones ($\equiv$ cómo lo hacen) (20%)
k) Dibujar el árbol de búsqueda que se explora hasta localizar la primera solución factible para el caso en el que P=4, esto es, hay 4 personas en el juego. Y donde Excluir[1..4] = { 2, 0, 2, 0 }. Numerar los nodos reflejando el orden en el que se visitan e indicar cuándo se realiza una poda (15%).

[Esquema voraz 3 puntos] Se pide resolver el mismo problema utilizando una estrategia voraz. Se deberá contestar de forma clara y concisa a las siguientes cuestiones: candidatos (10%), criterio de selección razonable y razonado (20%), función de factibilidad (10%), función de solución (10%) y mostrar, etapa a etapa, cómo la estrategia voraz descrita construye la solución para el siguiente ejemplo (50%):
P = 4, Excluir[1..4] = { 2, 0, 2, 0} y

Afecto[1..4][1..4] =

	1	2	3	4
1	0	10	8	6
2	9	0	7	5
3	1	2	0	3
4	6	5	4	0

SOLUCIÓN.-

SECUENCIA DE DECISIONES

<x_1, x_2,…, x_P> donde xi indica al amigo al que tiene que regalar la persona i.

FUNCIÓN OBJETIVO

$$maximizar \sum_{i=1}^{P} Afecto[i][x_i]$$

RESTRICCIONES EXPLÍCITAS

$$(\forall i)(x_i \epsilon \{1,2,\dots,P\}: 1 \leq i \leq P)$$

RESTRICCIONES IMPLÍCITAS

$$(\forall i)\left((\forall j)(i \neq j \rightarrow x_i \neq x_j : 1 \leq j \leq P\right): 1 \leq i \leq P)$$

$$(\forall i)(i \neq x_i): 1 \leq i \leq P)$$

$$(\forall i)(x_i \neq Excluir[i]): 1 \leq i \leq P)$$

TIPO DE SOLUCIÓN BUSCADA

La solución óptima, por tanto esquema 3.

PREPARAR_RECORRIDO_NIVEL_K

x[k]=0

EXISTE_HERMANO_NIVEL_K

x[k] < P

SIGUIENTE_HERMANO_NIVEL_K

x[k] = x[k] + 1

FUNCIÓN SOLUCIÓN

k = P

FUNCIÓN CORRECTO

QUÉ HACE.- la función correcto, tras recibir la secuencia de decisiones x y el valor de k correspondiente, devuelve falso si la persona k se regala a sí misma o si la persona k regala a una persona a la que no debe regalar, esto es, a Excluir[k]. Además, si el amigo asignado a la persona k, esto es, x[k], aparece en el tramo [1..k-1] de la secuencia x de decisiones, la función devolverá falso. En cualquier otro caso, devolverá cierto.

PSEUDOCÓDIGO (CÓMO).-

```
Funcion Correcto (Excluir[1..P]:vector de enteros, x:tupla, k:entero) retorna (b:booleano)
        var i:entero;ok:booleano fvar
        i = 0;
        ok = cierto;
        si ( k = x[ k ] ∨ x[ k ] = Excluir[ k ]) ok=falso; fsi
        mientras ( i < k-1 ∧ ok ) hacer
                i = i + 1;
                si ( x[ i ] = x[ k ] ) entonces
                        ok = falso;
                fsi
        fmientras
        retorna ok
ffunción
```

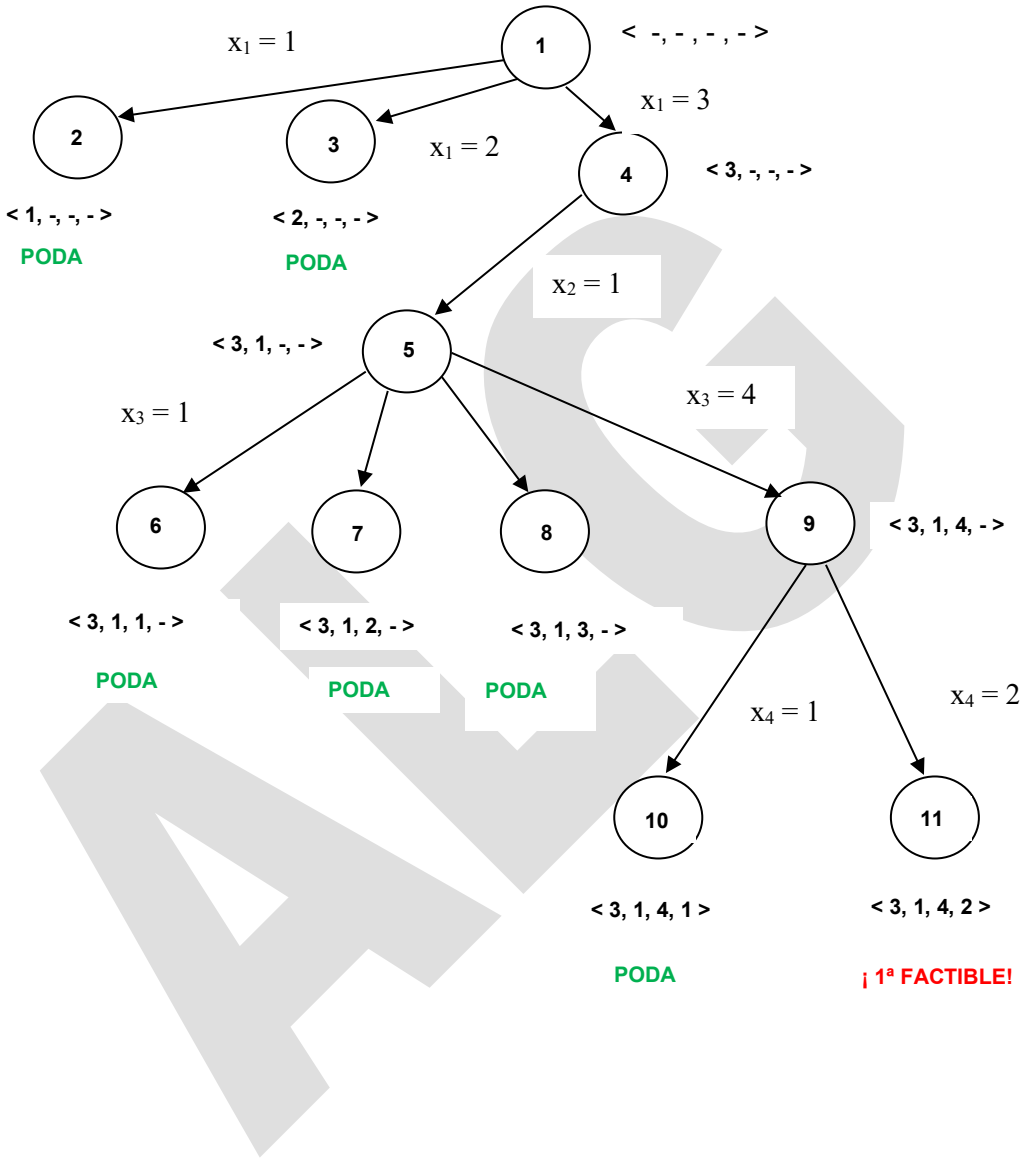
FUNCIÓN VALOR

QUÉ HACE.- la función valor, tras recibir la secuencia de decisiones x y el valor de k (k=P), devuelve el valor de la función objetivo correspondiente a la secuencia de decisiones x, esto es, la suma de los afectos conseguidos con la asignación de regalos.

PSEUDOCÓDIGO (CÓMO).-

```
Funcion Valor ( Afecto[1..P][1..P]: matriz de enteros, x:tupla, k:entero ) retorna ( b:entero )
        var i, total: entero fvar
        i = 0; total=0;
        mientras (i ≤ k-1) hacer
                i=i+1
                total = total + Afecto[ i ][ x[i] ]
        fmientras
        retorna total
ffunción
```

ÁRBOL DE BÚSQUEDA



Nodo 2: Poda porque la persona 1 no se puede regalar a sí misma
Nodo 3: Poda porque la persona 1 no puede regalar a la persona 2
Nodo 6: Poda porque la persona 1 no puede recibir más de un regalo
Nodo 7: Poda porque la persona 3 no puede regalar a la persona 2
Nodo 8: Poda porque la persona 3 no puede regalar a sí misma
Nodo 10: Poda porque la persona 1 no puede recibir más de un regalo

CANDIDATOS

Siguiendo la solución de backtracking, la secuencia de decisiones será $<x_1, x_2, \ldots, x_P>$ donde xi indica el amigo al que regalará la persona i-ésima.

Los candidatos por tanto serán las P personas. Todas ellas formarán parte de la solución, la cuestión es en qué orden forman parte de la solución.

CRITERIO DE SELECCIÓN

En cada etapa i se seleccionará aquel amigo aún no seleccionado, distinto de la persona i-ésima y no excluido para la persona i (Excluir[i]) cuyo afecto de la persona i hacia él sea el máximo.

FUNCIÓN DE FACTIBILIDAD

El criterio de selección propuesto hace que todas decisiones sean factibles.

FUNCIÓN DE SOLUCIÓN

Habremos encontrado la solución cuando en el conjunto solución estén las P personas.

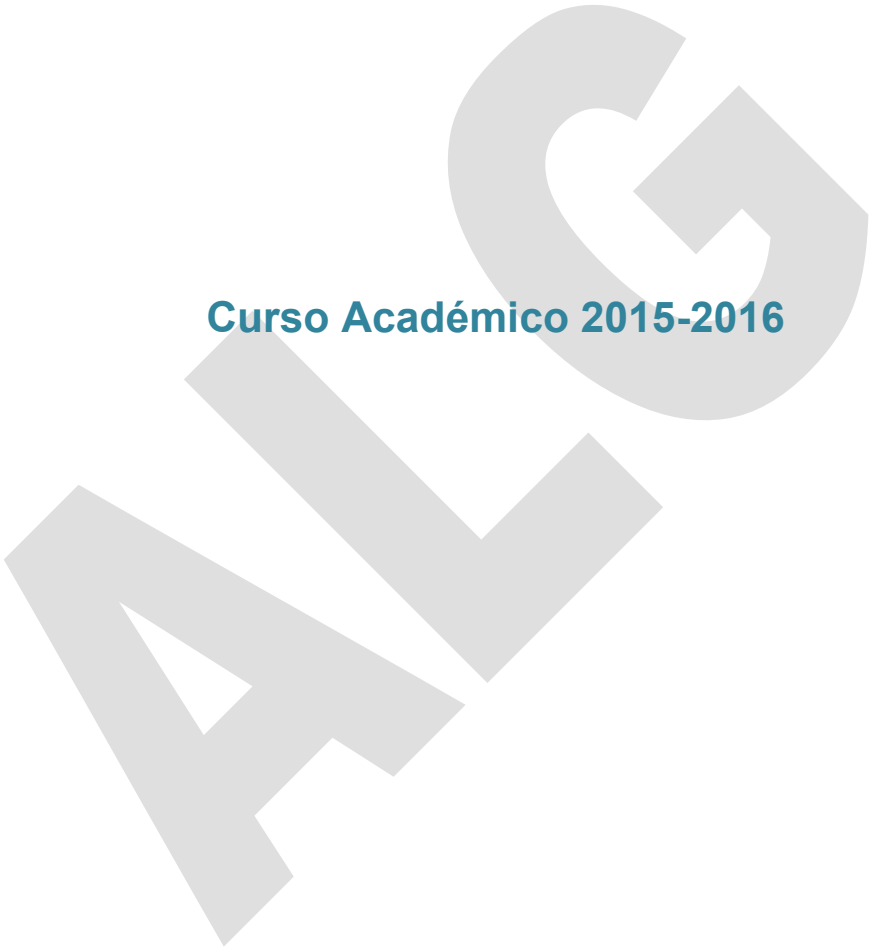
MOSTRAR ETAPA A ETAPA COMO LA ESTRATEGIA VORAZ CONSTRUYE LA SOLUCIÓN

P = 4, Excluir[1..4] = { 2, 0, 2, 0 } y

Afecto[1..4][1..4] =

	1	2	3	4
1	0	10	8	6
2	9	0	7	5
3	1	2	0	3
4	6	5	4	0

Etapa	Candidato seleccionado	Solución	Valor total
inicial	--	< -, -, -, - >	0
1	Amigo 3	< 3, -, -, - >	0+8=8
2	Amigo 1	< 3, 1, -, - >	8+9=17
3	Amigo 4	< 3, 1, 4, - >	17+3=20
4	Amigo 2	< 3, 1, 4, 2 >	20+5=25



Curso Académico 2015-2016

PROGRAMACIÓN DINÁMICA.- El gimnasio GYM_ALG, que dispone de S sedes, tiene solicitudes de algunos de sus socios para recibir clases de *Military Training*. Sea Sol[i], con $1 \leq i \leq S$, el número de solicitudes en la sede i-ésima del gimnasio. El gimnasio no dispone de monitores de dicha disciplina deportiva, por lo que decide contratar a un monitor externo. La facturación del monitor va en función de: a qué sede ha de ir a impartir clase y a cuántos alumnos va a tener. Dicha facturación está recogida en la matriz FACT[1..S][0..20], donde FACT[i][k] es el coste de acudir a la sede i-ésima y de tener a k alumnos. Se asume que el monitor podrá atender a lo sumo 20 alumnos en cada sede del gimnasio y que FACT[i][0]=0 para $1 \leq i \leq S$.

Con todo ello, se pide diseñar un algoritmo basado en Programación Dinámica que determine cuántas solicitudes atenderá el monitor en cada sede, de tal manera que atienda al mayor número de solicitantes en total sin que la factura que posteriormente pase al gimnasio exceda la cantidad de E euros.

Se pide responder con claridad y concisión a las siguientes cuestiones:

- Número de decisiones a tomar y significado de la decisión i-ésima (5%)
- Número de alternativas para la decisión i-ésima (5%)
- Función Objetivo (10%)
- Restricciones (10%)
- Demostración del principio de optimalidad (15%)
- Ecuación recursiva (30%)
- Primera llamada a la función (5%)
- Árbol de llamadas para un ejemplo (10%)
- Qué tipo de estructuras de almacenamiento se necesitan, cuál sería su dimensión y cómo se rellenarían (10%)

SOLUCIÓN.-

Número de decisiones a tomar y significado de la decisión i-ésima

El número de decisiones es S.

Secuencia de decisiones: $<d_1,d_2,\ldots, d_S>$

d_1 representa el número de solicitudes atendidas en la sede 1
d_2 representa el número de solicitudes atendidas en la sede 2
…
d_i representa el número de solicitudes atendidas en la sede i-ésima

Número de alternativas para la decisión i-ésima

Habrá un número de alternativas variable para la decisión i-ésima, dependerá de Sol[i], es decir, del número de solicitudes en la sede i. También hay que considerar que el monitor podrá atender a lo sumo 20 solicitudes en cada sede. Por tanto $d_i \in \{ 0,1,2,\ldots, \min (20, Sol[i]) \}$

Función objetivo

$$\text{maximizar} \sum_{i=1}^{S} d_i$$

Restricciones

$$\sum_{i=1}^{S} FACT[i][d_i] \leq E$$

Demostración del Principio de Optimalidad

Sea $<d_1, d_2, \ldots, d_S>$ la secuencia óptima de decisiones para el problema Alumnos(S,E), siendo su valor (número de solicitudes atendidas) asociado

$$\sum_{i=1}^{S} d_i$$

sujeto a

$$\sum_{i=1}^{S} FACT[i][d_i] \leq E$$

Vamos a prescindir de la última decisión, es decir, d_S. La subsecuencia que queda, esto es, $<d_1, d_2,\ldots, d_{S-1}>$, es la solución óptima para el subproblema Alumnos(S-1, E-FACT[S][d_S]), cuyo valor (número de solicitudes atendidas) es

$$\sum_{i=1}^{S-1} d_i$$

sujeto a

$$\sum_{i=1}^{S-1} FACT[i][d_i] \leq E - FACT[S][d_S]$$

Supongamos que <d_1, d_2,…, d_{S-1}> NO es la solución óptima del problema

Alumnos(S-1, E-FACT[S][d_S]),

sino que hay otra secuencia, siendo ésta, <d^*_1, d^*_2, …, d^*_{S-1}>, que mejora el resultado para el problema Alumnos(S-1, E-FACT[S][d_S]).

En este caso se cumpliría:

$$\sum_{i=1}^{S-1} d_i < \sum_{i=1}^{S-1} d_i^*$$

Y además:

$$\sum_{i=1}^{S-1} FACT[i][d_i^*] \leq E - FACT[S][d_S]$$

Por tanto, la secuencia <d^*_1, d^*_2,…, d^*_{S-1}, d_S > cumple:

$$\sum_{i=1}^{S-1} d_i + d_S < \sum_{i=1}^{S-1} d_i^* + d_S$$

por lo que mejora el valor (número de solicitudes atendidas) de <d_1, d_2, …, d_S>. En consecuencia, ésta no sería la solución óptima del problema Alumnos(S,E), en contra de lo supuesto inicialmente. En conclusión, se cumple el principio de optimalidad.

Definición recursiva

$$\text{Alumnos(s,e)} = \begin{cases} 0 & \text{si } s = 0 \\ \max_{0 \leq d_s \leq \min(20, Sol[s]) \,/\, e - FACT[s][d_s] \geq 0} \{ \text{Alumnos}(s-1, e - FACT[s][d_s]) + d_s \} & \text{si } s > 0 \end{cases}$$

donde Alumnos(s,e) representa el máximo de solicitudes atendidas para s sedes (de la 1 a la s) con un presupuesto disponible de e euros

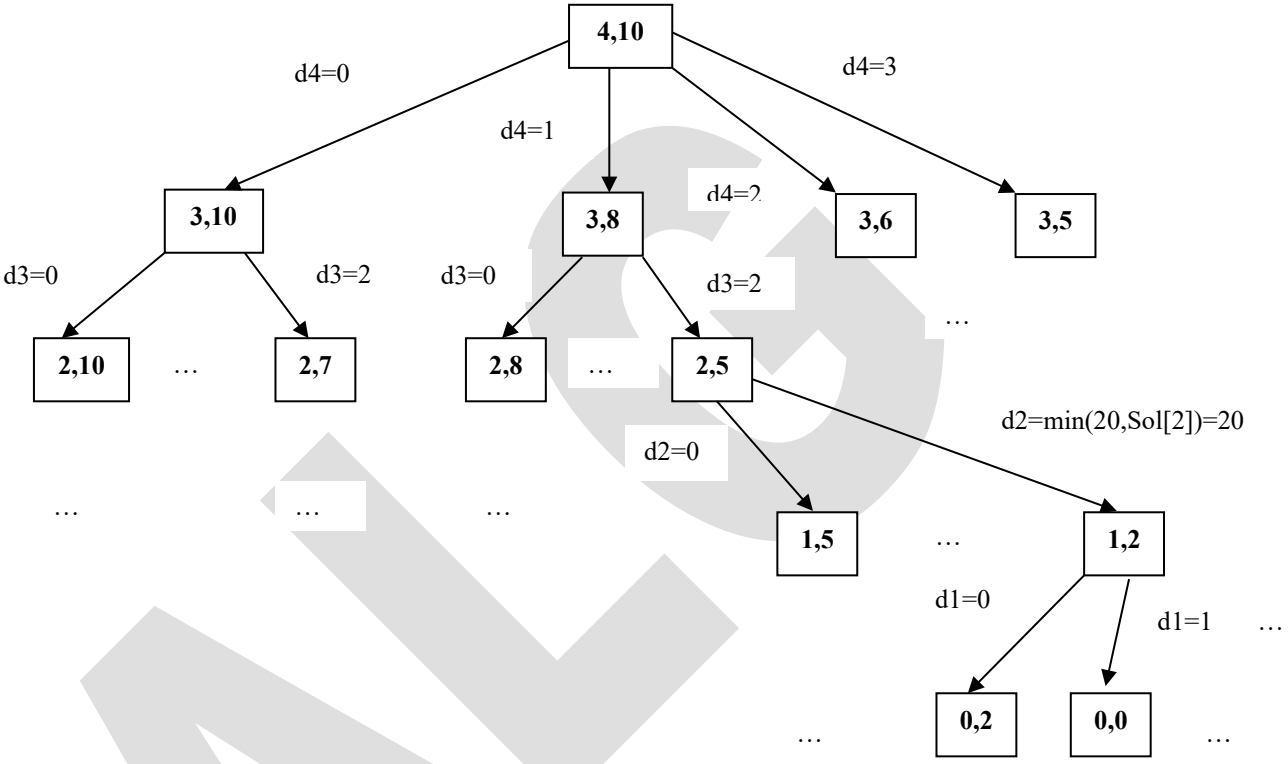
Primera llamada a la función

Alumnos(S,E) donde S es el número de sedes total y E es el presupuesto disponible inicialmente.

Árbol de llamadas para un ejemplo

S = 4, E = 10, Sol[1..4] = { 15, 32, 2, 3 } y FACT[1..4][0..20] es

FACT	0	1	2	3	…	20
1	0	2	9	2	…	4
2	0	2	1	1	…	3
3	0	3	3	8	…	5
4	0	2	4	5	…	9



Qué tipo de estructuras de almacenamiento se necesitan, cuál sería su dimensión y cómo se rellenarían.

Dado que nuestra función recursiva tiene dos parámetros, se precisan dos estructuras bidimensionales.

Éstas tendrán S+1 filas y E+1 columnas, esto es debido a que $0 \leq s \leq S$ y $0 \leq e \leq E$.

En una de las estructuras, se guardará el valor asociado a cada subproblema, esto es, el máximo de solicitudes atendidas. En la otra estructura, se almacenará la alternativa que proporcione el máximo para cada subproblema.

Inicializar las estructuras de almacenamiento con los resultados de los problemas triviales, corresponde a rellenar la fila 0 de cada matriz con 0, dado que los problema triviales son del tipo Alumnos(0,e).

Para completar el rellenado de la matrices: dado que para solucionar el subproblema Alumnos(s,e) se precisa conocer la solución de los subproblemas del tipo Alumnos(s-1,x), ambas matrices se rellenan por filas en sentido creciente, desde la fila 1 hasta la fila S.

VUELTA ATRÁS.- Se dispone de una hilera (vector V) de N (N>0) celdas consecutivas pintadas en el suelo identificadas por un índice del 1 al N. Suponemos que nos encontramos fuera de la hilera (en una posición virtual de índice 0) y que podemos avanzar por ella con movimientos de cualquier longitud, siempre que no salgamos de la misma. No existe posibilidad de retroceso.
En este caso, cada celda de la hilera contiene un número que puede ser positivo o negativo (no confundir con el índice de la celda).
Un jugador dispone de un capital inicial C>0, que puede incrementar o reducir en cada movimiento que realice. Sabemos que cada vez que visite una celda que contenga un valor positivo, su capital se incrementa en una cuantía igual a ese valor. En cambio, si la celda visitada contiene un valor negativo, el capital se reduce en esa cuantía.
El jugador no puede efectuar un movimiento si conduce a que el capital acumulado tras realizarlo sea menor o igual que 0.
El objetivo es alcanzar obligatoriamente la última celda (la de índice N) siguiendo una estrategia de movimientos que nos permita conseguir el mayor capital posible. Si por las características específicas del problema la última celda no fuera alcanzable, se entiende que el problema no tiene solución (sucedería cuando cualquier estrategia de movimientos para llegar a la última celda nos llevase en algún momento a un capital negativo o nulo).
Si existe solución óptima, el algoritmo debe imprimir el valor correspondiente. Si no existe, debe imprimir un mensaje informando de tal circunstancia.
Encontrar la solución óptima del problema, empleando la metodología de Backtracking.

Deberá responderse a los siguientes apartados:

- Secuencia de decisiones a tomar (5%)
- Naturaleza de cada una de las decisiones (5%)
- Restricciones explícitas del problema (10%)
- Restricciones implícitas del problema (10%)
- Función objetivo (15%)
- Esquema de Backtracking a utilizar (15%)
- Algoritmo que implemente cada una de las partes y funciones del esquema anterior, de acuerdo con las características del problema a resolver (40%)

Solución A (Solución (o tupla) de tamaño variable).-

Secuencia de decisiones:
$<d_1, d_2, \ldots, d_S>$ donde s es el número total de movimientos realizados. $1 \le s \le N$
d_i representa el índice de la celda a la que se accede en el movimiento i-ésimo.

Restricciones explícitas:

$$(\forall i) (d_i \in \{ 1, 2, , N \} : 1 \le i \le s)$$

Restricciones implícitas:
Una tupla NO es factible si alguno de sus elementos es menor o igual que el anterior, por lo que ha de cumplir que

$$(\forall i) (d_i > d_{i-1} : 2 \le i \le s)$$

Una tupla es factible si todos los capitales acumulados parciales son positivos

$$(\forall k) \left(C + \sum_{i=1}^{k} V[d_i] \right) > 0 \quad 1 \le k < s$$

Función objetivo:

$$\text{maximizar} \quad C + \sum_{i=1}^{s} V[d_i]$$

El óptimo se alcanza cuando el capital acumulado es máximo.
Una tupla es solución cuando $d_S=N$ (la última decisión conduce al elemento N)

Esquema a utilizar: Solución óptima

```
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
```

Función correcto (V[1..N]:vector de enteros, C:entero, x:tupla, k:entero) retorna (b:booleano)

```
var i:entero,correcto:booleano fvar

correcto = verdadero

para i = 2 hasta k  hacer
   si (x[i] <= x[i-1]) entonces correcto = falso  fsi
fpara
para i=1 hasta k-1 hacer
   si valor(V,C,x,i) <= 0 entonces correcto = falso   fsi
fpara
retorna correcto
ffuncion
```

Función valor (V[1..N]:vector de enteros, C:entero, x:tupla, k:entero) retorna (s:entero)

```
var total:entero fvar
total = C
para i = 1 hasta k  hacer
     total += V[x[i]];
fpara
retorna total
ffuncion
```

Función tratar (x:tupla, N:entero)

```
para i=1 hasta N hacer
     imprimir  x[i]
fpara
ffuncion
```

Solucion(x,k): x[k] = N

Preparar_recorrido_nivel_k: x[k]=0

Siguiente_hermano_nivel_k: x[k]+=1

Existe_hermano_nivel_k: x[k]<N

Solución B (Solución (o tupla) de tamaño fijo).-

Secuencia de decisiones: $<d_1, d_2, \ldots, d_N>$

$d_i = 1$ indica que la celda i-ésima forma parte del trayecto; $d_i = 0$, que no está incluida.

Restricciones explícitas:

$$(\forall i) (d_i \in \{ 0, 1 \} : 1 \leq i \leq N)$$

Restricciones implícitas:

Una tupla $<d_1, d_2, \ldots, d_s>$ (con $s < N$) es factible si en todos los puntos intermedios del trayecto el capital acumulado es positivo

$$(\forall k) \left(C + \sum_{i=1}^{k} d_i * V[i] \right) > 0 \quad : (1 \leq k < s) \wedge (d_k = 1)$$

Para ser factible, la tupla $<d_1, d_2, \ldots, d_N>$ necesita, además de la condición anterior, que $d_N = 1$

Función objetivo:

$$\text{maximizar} \quad C + \sum_{i=1}^{s} d_i * V[i]$$

El óptimo se alcanza cuando el capital acumulado es máximo.
Una tupla $<d_1, d_2, \ldots, d_s>$ es solución cuando $s = N$

Esquema a utilizar: Solución óptima

```
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
```

Función correcto (V[1..N]:vector de enteros, C:entero, N: entero, x:tupla, k:entero) retorna (b:bool)

```
var i, total : entero; correcto : booleano fvar
correcto=verdadero
para i =1 hasta k-1 hacer
   si (x[i]==1) Y (valor(V,C,x,i)<=0) entonces correcto=falso  fsi
fpara
si (k=n) Y (x[n]<>1) entonces correcto=falso   fsi
retorna correcto;
ffuncion
```

Función valor (V[1..N]:vector de enteros, C:entero, x:tupla, k:entero) retorna (s:entero)

```
Var total:entero
total=C
para i = 1 hasta k  hacer
     total+= x[i]*V[i];
fpara
retorna total;
ffuncion
```

Función tratar (x:tupla, N:entero)

```
para i = 1 hasta N
      imprimir  x[i]
fpara
ffuncion
```

Solucion(x,k): k = N

Preparar_recorrido_nivel_k: x[k] = -1

Siguiente_hermano_nivel_k: x[k] += 1

Existe_hermano_nivel_k: x[k] < 1