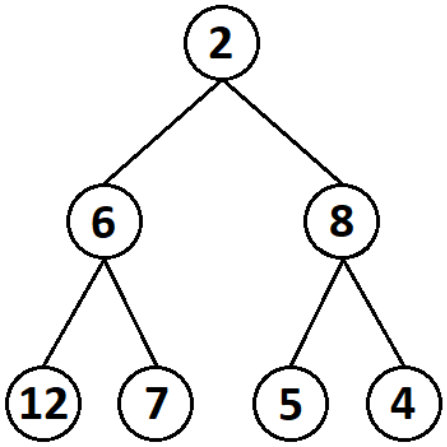
SEPTIEMBRE 2013 – ORIGINAL (MODELO A)

1. Considérese el vector *v=[2,6,8,12,7,5,4]*. ¿Cuál de las siguientes opciones es cierta?

   a) El vector *v* es un montículo de mínimos.
   b) *v* sería un montículo de mínimos si se flotara el elemento de valor 5.
   c) *v* sería un montículo de mínimos si se hundiera el elemento de valor 8.
   d) Ninguna de las opciones anteriores.



A)

- Montículo de Mínimos: Cada nodo padre es de un valor menor al de sus hijos.
- Montículo de Máximos: Cada nodo padre es de un valor mayor al de sus hijos.
- Flotar: Intercambiar por el nodo a un nivel inmediatamente superior.
- Hundir: Intercambiar por el nodo hijo de menor valor.

B) No es un montículo de mínimos

C) Sí es un montículo de mínimos

Respuesta C)

2. En el problema de colorear un grafo con n nodos utilizando m colores, de manera que no haya dos vértices adyacentes que tengan el mismo color, una cota superior ajustada del coste de encontrar una solución utilizando un esquema adecuado es del orden de:

   a) $O(n^2)$.
   b) $O(m^n)$.
   c) $O(m \log n)$.
   d) $O(n^m)$.

Respuesta B)

3. Sea el problema de la mochila en su versión de objetos no fraccionables solucionado mediante programación dinámica. Suponga que se dispone de 5 objetos con volúmenes {1,2,5,6,7} y que aportan unos beneficios de {1,6,18,22,28}, respectivamente. Suponga también que dispone de una mochila con una capacidad máxima de 11. Indique cuál sería el contenido de la tabla de resultados parciales en la fila correspondiente al objeto de peso 7, si dichos objetos se consideran en orden creciente de pesos.

   a) 0 1 6 7 7 18 19 24 25 25 28 29
   b) 0 1 6 7 7 18 22 24 28 29 29 40
   c) 0 1 6 7 7 18 22 28 29 34 35 40
   d) Ninguna de las anteriores.

El problema de la mochila se introduce inicialmente en el algoritmo Voraz, ya que es un algoritmo de optimización (ya que los algoritmos voraces son más eficientes para resolver este problema).

Este problema busca obtener el mayor beneficio posible (maximizar de beneficio).

No es posible resolver este tipo de problemas (el de la mochila) con algoritmos Voraces, ya que, como tendremos que estar realizando combinaciones entre los pesos y los beneficios de cada uno de los objetos, no llegaremos a encontrar una estrategia de selección que nos permita resolver el problema sin deshacer las decisiones que hayan sido tomadas.

Dicho de otra forma, no nos será posible elegir en cada iteración un objeto de forma que, al final, el conjunto de objetos que deciden pueda llegar a obtener un beneficio máximo sin que se supere la capacidad máxima de la Mochila.

Eso sí, hay una forma posible en la que se puede resolver este problema con el algoritmo Voraz, y esto es cuando los objetos son fraccionables, en dicho caso si existe una función de selección.

PROBLEMA DE LA MOCHILA

- Capacidad Máxima ($C_{máx}$)
- n objetos. Por cada objeto, tenemos:
  - Un Volumen (Vol): $v_1, v_2, \ldots, v_n$
  - Un Beneficio (Benef.): $b_1, b_2, \ldots, b_n$
- Se busca maximizar la suma de beneficios sin superar la capacidad máxima.

$$\text{Máx. } \Sigma\, b < C_{máx}$$

Resolución del Problema

1. Con Algoritmos Voraces: Cuando los objetos son fraccionables, es decir, que se nos permite meter para cada uno de los objetos una fracción del volumen (no completo).
   La función de selección sería calcular para cada objeto el valor por unidad de peso, es decir:
   Benef./Vol. o Peso → bi / Vi
   Selección: escoger los objetos de mayor a menor según bi/Vi hasta que llegue a la capacidad máxima.
2. Programación Dinámica (PD): Tienen un mayor coste espacial O(nC)
3. Ramificación y Poda (RyP): Tienen un mayor coste exponencial.

Tanto PD como RyP son algoritmos de optimización para objetos no fraccionables.

Se dibuja una tabla con dos dimensiones. En una de ellas, dibujamos los objetos de los que disponemos, donde la primera fila sería el caso base, es decir, cuando no tenemos objetos, al igual que la primera columna, también será dicho caso base.

Las filas serán los volúmenes, y las columnas las distintas capacidades de la mochila desde 0 hasta llegar a la capacidad máxima (11 en este caso).

		CAPACIDADES												
		0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	
		0	0	0	0	0	0	0	0	0	0	0	0	**BENEFICIOS**
PESOS	**1**	0	1	1	1	1	1	1	1	1	1	1	1	**1**
	2	0	1	6	7	7	7	7	7	7	7	7	7	**6**
	5	0	1	6	7	7	18	19	24	25	25	25	25	**18**
	6	0	1	6	7	7	18	22	24	28	29	29	40	**22**
	7	0	1	6	7	7	18	22	28	29	34	35	40	**28**

Para realizar el cálculo de cada celda, se realiza como:

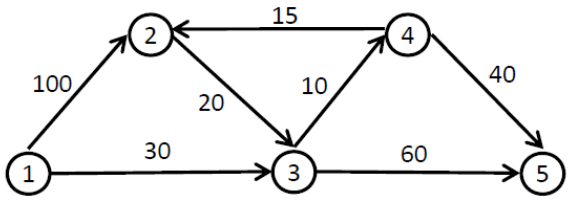
Valor máximo entre

- Valor de la misma columna en la fila anterior
- Beneficio del Peso + X

X = Valor en la columna Y de la fila anterior Y = Capacidad de la mochila (C) – Peso

Respuesta C)

4. Dado el siguiente grafo dirigido:



Indique el valor del vector de distancias *especial[]* en el paso del algoritmo de Dijkstra en el que se selecciona el nodo *v=4*, tomando como nodo origen el nodo 1:

a) [∞, 30,40, ∞]
b) [50, 30,40, ∞]
c) [55, 30,40, 80]
d) Ninguna de las anteriores

Nodos que han salido	**Nodos que no han salido**	**Vector Distancia especial [ ]**				**Predecesores**			
		2	**3**	**4**	**5**	**2**	**3**	**4**	**5**
1	2, 3, 4, 5	100	**30**	∞	∞	1	1	1	1
1, 3	2, 4, 5	100	30	**40**	90	1	1	3	3
1, 3, 4	2, 5	**55**	30	40	80	4	1	3	4
1, 3, 4, 2	5	55	30	40	80	4	1	3	4

Respuesta C)

5. Indica cuál de las siguientes afirmaciones es cierta con respecto a la resolución de colisiones:

   a) El método de hashing abierto es siempre más eficiente que el hashing cerrado para la resolución de colisiones.
   b) En el método de hashing cerrado con recorrido lineal existe una probabilidad muy baja de colisiones independientemente de los patrones de las claves.
   c) En el método de hashing cerrado con recorrido cuadrático es posible garantizar que se van a recorrer todos los elementos de la tabla cuando el número de elementos de la tabla, m, cumple determinadas condiciones.
   d) En el método de hashing cerrado con recorrido mediante doble hashing la función h y h' que se aplican pueden ser iguales cuando el número de elementos de la tabla, m, cumple determinadas condiciones.

a) **Falso.** Desde el punto de vista conceptual, el hashing abierto es un método válido para la resolución de colisiones. No obstante, si consideramos la eficiencia, este no es un método muy eficiente para el recorrido lineal de estas estructuras, comparado como si lo es el método del hashing cerrado.

b) **Falso.** El hashing cerrado permite resolver la colisión mediante la búsqueda en ubicaciones alternativas en la misma tabla, hasta que se encuentra un sitio libre en la misma. Se debe determinar que hay un sitio libre en la tabla con la presencia de un valor que lo determine, y, si es así, se ubica el valor en la posición indicada por la función Hash. En este tipo de direccionamiento, a medida que la tabla se llena, las probabilidades de colisión aumentan significativamente.

   Los métodos más conocidos para implementar este tipo de resolución de colisiones son:

   - ***Recorrido Lineal:*** en general, a la dirección obtenida por la función Hash h(k) se le añade un incremento lineal que proporciona otra dirección en la tabla, donde "m" es el tamaño de la tabla:

     $$s(k, i) = (h(k) + i) \bmod m$$

   - ***Recorrido Cuadrático:*** en el caso del recorrido lineal, la probabilidad de nuevas colisiones es bastante alta para determinados patrones de claves. Hay otro método basado en una expresión cuadrática g(k) que permite mayor dispersión de las colisiones por la tabla, al mismo tiempo que proporciona un recorrido completo por la misma.
   - ***Recorrido mediante Doble Hashing:*** se utiliza una segunda función Hash h'(x)' como función auxiliar. De esta manera tenemos:

     $$dir = (h(k) + ch'(k)) \bmod m$$

     Para que este método funcione, la función h'(x) debe cumplir las condiciones:
     - h'(x) ≠ 0, ya que de lo contrario se producen colisiones.
     - La función h' debe ser diferente de la función h.
     - Los valores h'(x) deben ser primos relativos de M (es decir, que no comparten factores primos), para que los índices de la tabla se ocupen en su totalidad. Si M es primo, cualquier función puede ser usada como h'.

c) **Cierto.** Explicado en el apartado b).

d) **Falso.** Explicado en el apartado b).

<u>Respuesta C)</u>

6. Se dispone de cuatro tipos de monedas de valores 1, 2, 4 y 8. Se desea resolver el problema de pagar una cantidad C > 0 utilizando un número mínimo de monedas y suponiendo que la disponibilidad de cada tipo de moneda es ilimitada. ¿Cuál de los siguientes esquemas es más eficiente de los que puedan resolver el problema correctamente?

   a) Esquema voraz.
   b) Esquema de programación dinámica.
   c) Esquema de vuelta atrás.
   d) Esquema de ramificación y poda.

El esquema más apropiado es el Voraz, ya que las monedas son múltiplos unas de otras. En caso de no ser así, habría que utilizar el esquema de Programación Dinámica.

Respuesta A)

Problema (4 puntos). La agencia matrimonial Celestina & Co. quiere informatizar parte de la asignación de parejas entre sus clientes. Cuando un cliente llega a la agencia se describe a sí mismo y cómo le gustaría que fuera su pareja. Con la información de los clientes la agencia construye dos matrices M y H que contienen las preferencias de los unos por los otros, tales que la fila M[i, .] es una ordenación de mayor a menor de las mujeres cliente según las preferencias del i-ésimo hombre, y la fila H[i, .] es una ordenación de mayor a menor de los hombres cliente según las preferencias de la i-ésima mujer. Por ejemplo, M[i,1] almacenaría a la mujer preferida por el hombre i y M[i,2] a su segunda mujer preferida.
Dado el alto índice de divorcios, la empresa se ha planteado como objetivo que los emparejamientos sean estables. Se considera que una pareja (h , m) es estable si no se dan las siguientes situaciones:

1. Que exista una pareja (h', m') con una mujer m' tal que el hombre h la prefiere sobre la mujer m y además la mujer m' también prefiere a h sobre h'.
2. Que exista una pareja (h'', m'') con un hombre h'' tal que la mujer m lo prefiere sobre el hombre h y además el hombre h'' también prefiere a m sobre m''.

La agencia quiere que dadas las matrices de preferencia, un programa establezca parejas evitando las dos situaciones descritas anteriormente. Se pide:

a) Elección del esquema más apropiado, el esquema general y explicación de su aplicación al problema (0,5 puntos).

   El esquema más apropiado es el de Vuelta Atrás. En el espacio de búsqueda asociado, en cada nivel k, se intentará asignar una mujer al hombre k-ésimo.

   Los elementos principales del esquema de vuelta atrás son:

   - *IniciarExploraciónNivel()*: recoge todas las opciones posibles en que se puede extender la solución k-prometedora.
   - *OpcionesPendientes()*: comprueba que quedan opciones por explorar en el nivel.
   - *SoluciónCompleta()*: comprueba que se haya completado una solución al problema.
   - *ProcesarSolución()*: representa las operaciones que se quieran realizar con la solución, como imprimirla o devolverla al punto de llamada.
   - *Completable()*: comprueba que la solución k-prometedora se puede extender con la opción elegida cumpliendo las restricciones del problema hasta llegar a completar una solución.

```
fun VueltaAtras (v: Secuencia, k: entero)
    {v es una secuencia k-prometedora}
    IniciarExploraciónNivel(k)
    mientras OpcionesPendientes(k) hacer
        extender v con siguiente opción
        si SoluciónCompleta(v) entonces
            ProcesarSolución(v)
        sino
            si Completable (v) entonces
                VueltaAtras(v, k+1)
            fsi
        fsi
    fmientras
ffun
```

b) Descripción de las estructuras de datos necesarias (0,5 puntos solo si el punto 1 es correcto).

Las dos matrices que plantea el enunciado M y H que serán respectivamente un array de dimensiones n x n.

La solución se puede almacenar en un array S de n elementos, siendo n el número de parejas que hay que asignar. El elemento S[i] indicará el número de la mujer asignada al hombre i-ésimo.

Para comprobar la disponibilidad de mujeres se puede utilizar un array ML de n elementos booleanos.

Para facilitar el control de las restricciones se puede utilizar un array de n elementos que contenga los hombres asignados a cada mujer HA.

Para facilitar el procesamiento, también se pueden utilizar dos matrices auxiliares que expresan el orden de preferencia de mujeres a los hombres y viceversa.

Una de ellas, en la posición [i, j], almacenaría el orden de preferencia de la mujer “i” por el hombre “j”. La otra, en la posición [i, j], almacenaría el orden de preferencia del hombre “i” por la mujer “j”.

c) Algoritmo completo a partir del refinamiento del esquema general (2,5 puntos solo si el punto 1 es correcto).

En primer lugar, hemos de decidir cómo representaremos la solución del problema mediante una n-tupla de valores X = [x_1, x_2, ..., x_n]. En este ejemplo el valor xi va a representar la tarea asignada a la i-ésima persona.

Empezando por la primera persona, en cada etapa el algoritmo irá avanzando en la construcción de la solución, comprobando siempre que el nuevo valor añadido a ella es compatible con los valores anteriores.

Por cada solución que encuentre anotará su coste y lo comparará con el coste de la mejor solución encontrada hasta el momento, que almacenará en la variable global mejor.

El algoritmo puede ser implementado como sigue:

```
CONST n = ...; (* numero de personas y trabajos *)
TYPE  TARIFAS = ARRAY[1..n],[1..n] OF CARDINAL;
      SOLUCION = ARRAY[1..n] OF CARDINAL;

VAR   X,mejor:SOLUCION;
      minimo:CARDINAL;
      tarifa:TARIFAS;

PROCEDURE Asignacion(k:CARDINAL);
   VAR c:CARDINAL;
BEGIN
   X[k]:=0;
   REPEAT
      INC(X[k]);
      IF Aceptable(k) THEN
         IF k<n THEN Asignacion(k+1)
         ELSE
            c:=Coste();
            IF minimo>c THEN mejor:=X; minimo:=c END;
         END
      END
   UNTIL X[k]=n
END Asignacion;
```

Siendo los procedimientos *Aceptable* y *Coste* los que respectivamente comprueban las restricciones para este problema y calculan el coste de las soluciones que van generándose.

En cuanto a las restricciones, sólo se va a definir una, que asegura que las tareas sólo se asignan una vez. Por otro lado, el coste de una solución coincide con el coste de la asignación:

```
PROCEDURE Aceptable(k:CARDINAL):BOOLEAN;
(* comprueba que esa tarea no ha sido asignada antes *)
   VAR i:CARDINAL;
BEGIN
   FOR i:=1 TO k-1 DO
      IF X[k]= X[i] THEN RETURN FALSE END
   END;
   RETURN TRUE
END Aceptable;


PROCEDURE Coste():CARDINAL;
   VAR suma,i:CARDINAL;
BEGIN
   suma:=0;
   FOR i:=1 TO n DO
      suma:=suma+tarifa[i,X[i]]
   END;
   RETURN suma
END Coste;
```

Para resolver el problema basta con invocar al procedimiento *Asignación* tras obtener los valores de la tabla de tarifas e inicializar la variable *minimo*:

```
...
minimo:=MAX(CARDINAL);
Asignacion(1);
ComunicarSolucion(mejor);
...
```

Al igual que en el problema anterior, existe una modificación a este algoritmo que permite realizar podas al árbol de expansión eliminando aquellos nodos que no van a llevar a la solución óptima.

Para implementar esta modificación –siempre interesante puesto que consigue reducir el tamaño del árbol de búsqueda– necesitamos hacer uso de una cota que almacene el valor obtenido por la mejor solución hasta el momento, y además llevar la cuenta en cada nodo del coste acumulado hasta ese nodo.

Si el coste acumulado es mayor que el valor de la cota, no merece la pena continuar explorando los hijos de ese nodo, pues nunca nos llevarán a una solución mejor de la que teníamos.

Como la cota la tenemos disponible ya (es la variable *minimo* del algoritmo anterior), lo que haremos es ir calculando de forma acumulada en vez de al llegar a una solución, y así podremos realizar la poda cuanto antes:

```
PROCEDURE Asignacion2(k:CARDINAL;costeacum:CARDINAL);
   VAR coste:CARDINAL;
BEGIN
   X[k]:=0;
   REPEAT
      INC(X[k]);
      coste:=costeacum+tarifa[k,X[k]];
      IF Aceptable(k) AND (coste<=minimo) THEN
         IF k<n THEN Asignacion2(k+1,coste)
         ELSE mejor:=X; minimo:=coste
         END
      END
   UNTIL X[k]=n
END Asignacion2;
```

También hacer una última observación sobre el problema de la asignación en general. Este problema siempre posee solución puesto que siempre existen asignaciones válidas.

De esta forma no nos tenemos que preocupar de si el algoritmo acaba con éxito o no.

d) Estudio del coste del algoritmo desarrollado (0,5 puntos solo si el punto 1 es correcto).

El tamaño del espacio de búsqueda es del orden n!, por lo que se puede considerar que una cota superior es O(n!).