1.- Demuestre las siguientes afirmaciones. ¿Qué quieren decir?

(a) Si f(n) = O(log a), a > 0, entonces f(n) = O(log n).
(b) Si f(n) = o(g(n)) entonces f(n) + g(n) = O(g(n)).
$n \to +\infty$
(c) Si O(f(n)) < O(g(n)) entonces O(f(n) + g(n)) = O(g(n)). (Aplicar el apartado b)
(d) Si f(n) es la función logarítmica, polinómica o suma y/o producto de ellas, y para n potencia de 2 tenemos que otra determinada función g(n) es O(f(n)). Entonces, si g(n) es creciente, es O(f(n)) para todo n.

2.– Para resolver un determinado problema P_i, se dispone de dos algoritmos: A y B, con complejidades respectivas $f_A(n)$ y $f_B(n)$. Determine, según el siguiente cuadro, para qué valores de n conviene usar A ó B.

	$f_A(n)$	$f_B(n)$
P_1	n^2	$10.n$
P_2	2^n	$2.n^3$

	$f_A(n)$	$f_B(n)$
P_3	$n^2/\log n$	$n.(\log n)^2$
P_4	$n^3/2$	$n^{2,81}$

3.– Calcule el orden de complejidad de los algoritmos de ordenación por selección, inserción e intercambio directos (en el mejor y peor de los casos). Hágase también con las versiones mejoradas inserción binaria y sacudida, comparando los resultados. (Elija como función de complejidad en tiempo la suma del número de comparaciones más el de asignaciones).

4.– Calcule el orden de complejidad de cada uno de los siguientes fragmentos de programa, eligiendo previamente el tamaño y función de complejidad adecuados, para los casos en que Alg(m) sea

1º) O(1) y 2º) $O(m^2)$

```
(a) j = n; k = 1;
    while (j >= 1) {
       k++; j = n/k;
       a = Alg(j);
    }
```

```
(b) j = n; k = 1;
    while (j >= 1) {
       k++; j = n/k;
       for (i = 1; i <= j; i++)
          a = Alg(i);
    }
```

```
(c) j = n;
    while (j >= 1) {
       j = j/5;
       a = Alg(j);
    }
```

```
(d) for (i = 1; i <= n; i++)
       for (j = 1; j <= i; j++) {
          t = n+1;
          while (t > i) {
             a = Alg(t);
             t--;
          }
       }
```

5.– De una función de complejidad en tiempo T(n), en forma recurrente, para el siguiente algoritmo, asumiendo que n es potencia de 2. Deduzca de ella un orden de complejidad (en tiempo) que sea peor o igual al del algoritmo, y otro que sea mejor o igual. El procedimiento E es de orden O(1). ¿Cambiarían los resultados si quitamos la hipótesis de que n sea potencia de 2?.

```
int Camino( int s, int t, int n ) {
   int Cam = true;
   if (n = 1) {
      if (E(s,t) == true) return true ;
      else return false;
   }
   else {
      for ( i = 1 ; i <= n ; i++ )
         Cam = Cam && Camino(s,i,n/2) && Camino(i,t,n/2);
      return Cam;
   }
}
```

6.– El siguiente algoritmo (de Prim) busca un árbol generador mínimo para un grafo conexo valorado. V es un vector que guarda los vértices, X la matriz de costes y F un vector de arcos donde guardamos la solución. Se pide calcular el orden de complejidad del algoritmo (en tiempo) para los casos en que el tamaño del problema sea

a) n = número de arcos del grafo, y
b) m = número de nodos del grafo.

1. i ← 1; x ← V(1);
2. Hacer hasta que i = n
   2.1. Elegir X(a,b) = mín { X(V(x), V(y))/ 1≤x≤i, i<y≤n}
   2.2. F(i) ← (a,b);
   2.3. i ← i + 1;
   2.4. Intercambiar V(i) y V(b).

7.– Calcule el orden de complejidad del siguiente fragmento de programa, en donde Proc(n) es i) O(1) y ii) O(n)

```
for ( i = 1 ; i != n ; i++ ) {
   a = i ;
   do {
      Proc(a);
      a = a/2;
   } while (a != 0);
   j = n/2;
   for ( k = j ; k >= 1 ; k-- )
      Proc( k );
}
```

¿Cambia el orden de complejidad si quitamos el bucle `for` interior?
Repita los cálculos sustituyendo '`a = i`' por '`a = n`'.

8.– Calcule el orden de complejidad de las siguientes funciones o procedimientos

A)

```
void LF( int n , int *s ) {
   int i, x, y;
   if ( n < 1 )  s = 1;
   else {
      x = 1;
      for ( i = 2 ; i != n ; i++ ) x = x * i;
      y = 0;
      do {
         y++;
         x = x/4;
      } while ( x != 0 );
      s = y;
   }
}
```

B)

```
void MoverDisco( int Origen , int Destino ) {
   printf( " Movimiento Desde-Hacia " );
   printf( " Origen " );
   printf( " Destino " );
}

void Transferir( int n, int origen, int destino, int otro ) {
   if ( n > 0 ){
      Transferir( n-1, Origen, Otro, Destino);
      MoverDisco( Origen, Destino);
      Transferir( n-1, Otro, Destino, Origen);
   }
}
```

C) Asumiendo que se cumple la propiedad `x > 1`.

```
int PeroQueHaceEsto( int x, int y ) {
   if ( y > x ) {
      int c = PeroQueHaceEsto(2*x,y);
      while ( y > x ) {
         int i = 0;
         do {
            c = c + n*x*y ;
            i++;
         } while ( i*x <= y );
         x = 2 * x;
      }
   }
   return c;
}
```

D)

```
int Jun91( int n ) {
   int x = 1, y = 1;
   if ( n > 2 ) {
      y = Jun91(n/2);
      while ( x*x < n ) {
        x++;
        y = x + y;
      }
      y = 2 * Jun91(n/2) + x;
   }
   Jun91 = x + y;
}
```

E)

```
int Jun91( int n ) {
   int x , y;
   while ( n > 1 ) {
      x = 1;
      y = n;
      while ( n > x*x ) {
         x++;
         y = y – x;
      }
      n = n/2 ;
   }
   return (x + y)
}
```

F) Calcule la complejidad de las dos funciones siguientes:

```
int Estudio (int n) {
   int i, contador;

   contador = 0;
   for (i = 1; i <= n; i++) contador++;
   return contador;
}

int Examen (int n) {
   int j, k, contador;

   if (n < 2) return 1;
   else {
      contador = Estudio (n);
      j = 1; k = 1;
      while (j < n) {
         j = 3*j; k++;
         contador += Estudio(n);
      }
      return (contador + Examen(n/3) + k);
   }
} // Examen
```

G) Calcule la complejidad de las dos funciones siguientes:

```
int Adios (int n) {
   if (n <= 1) return 1;
   else return (1 + Adios(n-1));
} // Adios


int Despedida (int n) {
   int i, j, acumulador;

   if (n < 1) return 1;
   else {
      acumulador = Despedida(n/2) + Adios(n*n);
      j = 1;
      while (j < n) {
         acumulador += Adios(n);
         for (i = 1; i <= j; i++) acumulador++;
         j++;
      }
      return (acumulador);
   }
} // Despedida
```

H) Calcule la complejidad de las dos funciones siguientes:

```
int Una (int n) {
   if (n < 2) return 1;
   else return (2 * Una(n-1));
}

int Examen (int n) {
   int i, m, suma;

   if (n < 3) {
      suma = 0;
      for (i = 1; i <= n; i++) suma += Una(i);
      return suma;
   }
   else {
      m = n; suma = 0;
      while (m > 1) {
         m = m/3;
         for (i = 1; i <= 10; i++) suma += Una(n);
      }
      return (suma + Examen(n/3));
   }
} // Examen
```

I) Calcule la complejidad de las dos funciones siguientes:

```
int LaOtra (int n) {
   if (n < 2) return 1;
   else return (n * LaOtra(n/3));
}

int Examen (int n) {
   int i, suma;

   suma = 0;
   if (n < 2) {
      for (i = 1; i <= n; i++) suma += LaOtra(i);
      return suma;
   }
   else {
      for (i = 1; i <= n/2; i++) suma += LaOtra(n);
      return (suma + Examen (n/3));
   }
 } // Examen
```

J) Calcule la complejidad de las dos funciones siguientes:

```
int Uno (int n) {
   int K, Suma;

   K = 0; Suma = 0;
   for (i = 1; i <= n; i++) {
       Suma = Suma + n; K++;
   }
   while (Suma > 0) {
      Suma = Suma/2; K++;
   }
   return K;
   }
} // Uno


int Dos (int n) {
   int K, Suma;
   if (n < 2)
      return Uno(n*n);
   else {
      K = n; Suma = Dos (n/3);
      while (K > 0) {
         Suma = Suma + Factorial(n);
         K = K / 2;
      }
      return Suma;
   }
} // Dos
```

K) Calcule la complejidad de las dos funciones siguientes, indicando cuál tiene mayor orden de complejidad (debe implementar Factorial(n) y calcular su complejidad).

```
int Uno (int n) {
   int Suma, x;
   Suma = 0; x = 0;
   while (x*x < n) {
      Suma = Suma + Factorial(x); x++;
   }
   return Suma;
} // Uno


int Examen (int n) {
   int K, Suma;
   if (n < 3)
      return Uno(n*n);
   else {
      K = n; Suma = Examen (n/2);
      while (K > 0) {
         Suma = Suma + Factorial(n);
         K = K / 3;
      }
      return Suma;
   }
} // Examen
```

L) Calcule la complejidad de las dos funciones siguientes, indicando cuál tiene mayor orden de complejidad, donde `Dudua(n)` es una función de complejidad lineal.

```
int Uno (int n) {
   int Suma, x;
   Suma = 0; x = 0;
   while (x*x < n) {
      Suma = Suma + Dudua(x); x++;
   }
   return Suma;
} // Uno

int Examen (int n) {
   int K, Suma;
   if (n < 2) return Uno(2*n);
   else {
      K = n; Suma = Examen (n/2);
      while (K > 0) {
         Suma = Suma + Dudua(n);
         K = K / 4;
      }
      return Suma;
   }
} // Examen
```

M) Calcule la complejidad de las dos funciones siguientes, indicando cuál tiene mayor orden de complejidad.

```
int Uno (int n) {
   int Suma, x;
   Suma = 0; x = 0;
   while (x*x*x < n) {
      Suma = Suma + x; x++;
   }
   return Suma;
} // Uno

int Examen (int n) {
   int K, Suma;
   if (n < 2) return Uno(n*n*n);
   else {
      K = n; Suma = Examen (n/3);
      while (K > 0) {
         Suma = Suma + n;
         K = K / 4;
      }
      return Suma;
   }
} // Examen
```

1.– Sea A una matriz cuadrada *n x n*, conteniendo la siguiente información en cada fila *i*

$$A[i,j] = \begin{cases} 1 & \text{para } 1 \le j \le k_i \le n \\ 0 & \text{para } k_i < j \le n \end{cases}$$

Construya un algoritmo "***Divide y Vencerás***" que ordene las filas de la matriz según el número de unos que tienen, sin pasar de una complejidad *O(n log n)*. No considere la complejidad que supone intercambiar filas.

2.– Construya un algoritmo "***Divide y Vencerás***" que multiplique dos polinomios de grado *n*, con una **complejidad estrictamente menor que cuadrática.** Calcule la complejidad del algoritmo desarrollado.

3.– Sean A y B dos vectores ordenados de *n* elementos. Diseñe un algoritmo "***Divide y Vencerás***" de complejidad menor estricta que lineal que calcule la mediana del vector ordenado que resultaría de mezclar A y B.

4.– Un método de ordenación es ESTABLE si el orden relativo entre elementos iguales se mantiene después de la ordenación. Compruebe la estabilidad de los algoritmos SHELL, HEAPSORT, QUICKSORT y MGSNatFILE.

5.– Busque contraejemplos que justifiquen todos los controles que se usan en el algoritmo QUICKSORT (≤ en vez de <, al revés, etc.).

6.– Para obtener el elemento pivote x en el algoritmo QUICKSORT pueden emplearse técnicas que impidan elegir el peor elemento. ¿Qué orden de complejidad deben tener éstas como máximo, para que la complejidad del QUICKSORT no varíe? Idear una forma de hacerlo.

7.– Construya una versión iterativa del QUICKSORT.

8.– ¿Qué inconvenientes tendría hacer una versión recursiva del algoritmo MGSNatFILE?

9.– Construya un algoritmo que localice el lugar *k* que ocupará un determinado elemento *x* de un vector si éste se ordena. Basarse para ello en la parte "Divide" del QUICKSORT. Estudie su complejidad en el mejor, en el peor de los casos y en media, y sacar conclusiones.

10.– Construya un algoritmo de ordenación "in situ" basado en la mezcla natural. (Debe mezclar dos tramos naturales de un array sin usar memoria auxilar que pase de orden constante). Inténtese que la complejidad total sea < $O(n^2)$.

11.– Disponemos de un vector A de *N* elementos enteros ordenados en sentido creciente estricto, i.e., no hay elementos repetidos. Se pretende encontrar una posición i, tal que A[i] = i. Diseñe un algoritmo "***Divide y Vencerás***" que devuelva dicha posición o retorne el valor 0 cuando no exista ninguna.

12.– Calcule el orden de complejidad para los casos peor, mejor y medio. En cualquier caso estas complejidades deben ser menores que la obtenida por un algoritmo de búsqueda exhaustiva.

13.– Disponemos de un vector A de *N* enteros ordenados en sentido decreciente, no necesariamente estricto. Construya una función "***Divide y Vencerás***" tal que si existe una posición i, tal que A[i] = i, devuelva dicha posición o retorne el valor 0 cuando no exista ninguna. La complejidad ha de ser menor estricta que la lineal.

14.– Sean X e Y dos vectores de enteros sin elementos repetidos de dimensión *N*. Se sabe que X está ordenado crecientemente y que Y lo está decrecientemente, al menos uno en sentido estricto. Construya una **función recursiva** que decida en **tiempo logarítmico** si hay un índice i tal que

$$X[i] = Y[i]$$

Construya una **versión iterativa** de dicha función.

15.– Construya un procedimiento que seleccione los *K* menores elementos de un vector de *N* elementos dado. La complejidad del procedimiento debe ser lineal.

16.– Construya un procedimiento que seleccione el *K*-ésimo menor elemento de un vector de *N* elementos dado. La complejidad del procedimiento debe ser lineal.

17.– Tenemos un vector de *N* elementos, y pretendemos comprobar si existe algún elemento en él que se repita al menos *N/2* veces. Diseñe un algoritmo que realice dicha comprobación, y devuelva, en su caso, cuál es el elemento repetido. Si existen dos elementos, devolverá cualquiera de ellos. El orden de complejidad del algoritmo, en el caso medio, no debe ser superior al lineal si el elemento existe.

Sobre el algoritmo diseñado, calcule la complejidad en los casos mejor y peor. Así mismo, calcule la complejidad en media sobre la hipótesis de que existen exactamente *N/2* elementos iguales, y los otros *N/2* son todos distintos.

No se permiten sobre los elementos del vector comparaciones de orden, sólo de igualdad.

18.– Escriba un procedimiento "***Divide y Vencerás***" que mezcle dos árboles binarios con ***estructura de montículo***, generando otro con la misma estructura.

Calcule el orden de complejidad en el caso medio considerando como tamaño del problema el número total de nodos entre los dos árboles.

Calcule el orden de complejidad en el peor caso considerando como tamaño del problema la suma de las profundidades de los árboles. Supóngase en este caso que los dos árboles tienen la misma altura y son completos.

El algoritmo debe tener la mínima complejidad posible.

NOTA: Un árbol binario tiene ***estructura de montículo*** si todo nodo es *mayor* que sus hijos izquierdo y derecho.

19.– El rey Midas se hizo famoso por su avaricia, que le llevó en cierta ocasión a proponerles a los sabios del reino el siguiente juego:

Tenemos una bolsa de *N* monedas de oro entre las que sabemos que hay una falsa, cuyo peso es menor que las demás. Teniendo presente que únicamente disponemos de una balanza de **DOS PLATOS** perfectamente equilibrada para pesarlas, diseñe una estrategia Divide y Vencerás segura, que encuentre la moneda falsa en el menor número de pesadas

NOTA: Se ha adoptar una estrategia del tipo ***Divide y Vencerás***, aunque el número de pesadas se reduce por una leve observación. Estudie pues bien la subdivisión del problema.

¿Es realmente óptima la estrategia adoptada?

20.– Se ha de organizar el calendario de un campeonato entre *N* jugadores. Cada uno ha de jugar exactamente una vez contra cada adversario. Además, cada jugador ha de jugar exactamente un partido diario, con la excepción posible de un sólo día en el que no jugará.

a) Si *N* es potencia de dos, construya un algoritmo que permita terminar el campeonato en *N - 1* días.

b) Siendo *N* cualquiera, construya un algoritmo que planifique el campeonato en *N - 1* días si *N* es par, o en *N* días si *N* es impar y mayor que 1.

21.– Un vector de enteros de dimensión impar, decimos que es "*centradito*" si sus tres elementos centrales luego de ordenarlo son consecutivos. Desarrolle una estrategia para decidir **en tiempo lineal** si un vector dado es "*centradito*".

22.– Dadas dos matrices A y B de tamaño *2k* x *2k*, pueden multiplicarse aplicando la descomposición siguiente:

$$\begin{pmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{pmatrix}\begin{pmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{pmatrix} = \begin{pmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{pmatrix}$$

siendo A_{ij} y B_{ij} matrices cuadradas de orden *k*.

Se pide:

a) Aplique dicha idea para desarrollar un algoritmo "***Divide y Vencerás***", para multiplicar matrices cuadradas de dimensión 2^n.

b) Idem para matrices cuadradas de dimensión $n = m\ 2^k$, con *m* y *k* naturales.

INDICACIONES Y NOTAS:

i) En ambos apartados debe cuidarse el gasto de memoria.

ii) En el apartado b) particione las matrices originales en *2k* x *2k* matrices de tamaño *m* x *m*. Aplique el método desarrollado en el apartado a) para multiplicar las matrices originales *n* x *n*, y el estándar para multiplicar los pares de submatrices *m* x *m* requeridos.

23.– Dada una matriz cuadrada de tamaño 2^n, con $n>0$. Diseñe utilizando ***divide y vencerás***, un algoritmo para rotarla 90º en el sentido de las agujas del reloj. Por ejemplo:

$$\begin{pmatrix} a & e & i & m \\ b & f & j & n \\ c & g & k & \tilde{n} \\ d & h & l & o \end{pmatrix}$$ se transforma mediante un giro de 90º en $$\begin{pmatrix} d & c & b & a \\ h & g & f & e \\ l & k & j & i \\ o & \tilde{n} & n & m \end{pmatrix}$$

24.– Diseñe un algoritmo basado en la técnica ***Divide y Vencerás*** que calcule la traspuesta de una matriz de dimensión 2^n.

25.– Diseñe un algoritmo basado en la técnica ***Divide y Vencerás*** que construya un árbol binario **casi-completo** a partir de un vector.

26.– Diseñe un algoritmo "***Divide y Vencerás***" para multiplicar dos enteros grandes.

27.– Dada una secuencia de enteros $E_1, E_2,..., E_n$. Se trata de encontrar una subsecuencia $E_j, E_{j+1},..., E_{j+k}$, para algunos j y k que verifican $1 \le j \le n$ y $1 \le j+k \le n$ tal que la suma

$$\sum_{j \le l \le k+j} E_l$$

sea **máxima**. Construya un algoritmo "***Divide y Vencerás***" que devuelva la **suma de la subsecuencia de suma máxima** con a lo sumo complejidad $O(n \log n)$.

28.– **Par de puntos más cercano**. Se tienen n puntos en el plano. Diseñe un algoritmo "***Divide y Vencerás***" que devuelva el par de puntos cuya distancia entre ambos es **mínima**.

29.– Dadas **dos cadenas ordenadas** x, de n símbolos, e y, de exactamente **dos** símbolos, se desea obtener un índice $j \in N$ tal que $0 < j < n$, $x_j = y_1$, $x_{j+1} = y_2$, o bien $j = 0$, si no existe tal j.

Siguiendo la estrategia "***Divide y Vencerás***", desarrolle algoritmos para resolver el problema propuesto en los siguientes casos:

a) Ninguna de las dos cadenas contienen símbolos repetidos.

b) Las dos cadenas pueden contener símbolos repetidos.

En ambos casos la complejidad de los algoritmos (que ha calcularse) deberá ser **logarítmica** en el peor de los casos.

Ejemplos:

a) x ="A D E F H P S T W", y ="H P", $\rightarrow j = 5$; y ="H T", $\rightarrow j = 0$.

b) x ="A D D D D R R S T W", y ="D D", $\rightarrow j = 2$; y ="R T", $\rightarrow j = 0$.

Nótese que con y ="D D", otras soluciones correctas son $j = 3$ ó $j = 4$.

30.– Se considera la clasificación de una maratón con miles de participantes, pero resulta que los datos de los participantes están ordenados por el dorsal con el que compitieron en la misma. Se desea conocer:

a) los cien primeros clasificados, no necesariamente ordenados, y
b) los diez primeros clasificados, éstos si ordenados.

Diseñe algoritmos eficientes que resuelvan estos problemas. La complejidad del algoritmo desarrollado debe ser lineal en el peor de los casos.

31.– Se consideran un vector *v* de *N* elementos y un entero positivo *k*. Diseñe un algoritmo que **devuelva los *k* elementos centrales** de *v*, luego de ordenarlo. La complejidad del algoritmo desarrollado debe ser lineal en el peor de los casos. La complejidad no es necesario que se calcule analíticamente, pero debe justificarse.

32.– Se consideran un vector *v* de *N* elementos y un entero positivo *k*. Diseñe un algoritmo que **devuelva los *k* elementos de *v* más próximos (en valor) a su mediana**. La complejidad del algoritmo desarrollado debe ser lineal en el peor de los casos. La complejidad no es necesario que se calcule analíticamente, pero debe justificarse.

33.– Se consideran un vector *v* de *N* elementos y un entero positivo *k*. Diseñe un algoritmo que **devuelva los *k* elementos de *v* más alejados (en valor) de su mediana**. La complejidad del algoritmo desarrollado debe ser lineal en el peor de los casos. La complejidad no es necesario que se calcule analíticamente, pero debe justificarse.

34.– Se consideran un vector *v* de *N* elementos y dos enteros positivos *k* y *p*. Diseñe un algoritmo que **devuelva los *p* elementos de *v* más próximos al k-ésimo menor elemento de *v* luego de ordenarlo**. La complejidad del algoritmo desarrollado debe ser lineal en el peor de los casos. La complejidad no es necesario que se calcule analíticamente, pero debe justificarse.

35.– Se consideran un vector *v* de *N* elementos y dos enteros positivos *k* y *p*. Diseñe un algoritmo que **devuelva los *p* elementos de *v* más próximos al k-ésimo mayor elemento de *v* luego de ordenarlo**. La complejidad del algoritmo desarrollado debe ser lineal en el peor de los casos. La complejidad no es necesario que se calcule analíticamente, pero debe justificarse.

36.– En una apartada localidad se ha realizado una encuesta para determinar cuál es la palabra más popular. Resulta que esa palabra puede ser inventada, y no estar recogida en el diccionario, ni en ninguna fuente. Los lugareños son de ideas fijas, y de vocabulario contagioso, así que por encuestas anteriores realizadas en la localidad se sabe que **cuando una palabra se pone de moda entre ellos, entonces al menos un tercio de la población la elige**. Diseñe un algoritmo **eficiente** que dado el resultado de la encuesta (la palabra elegida por cada entrevistado) determine la palabra de moda, si es que ésta existe. Indicación: La complejidad del algoritmo desarrollado debe ser lineal en el peor de los casos. La complejidad no es necesario que se calcule analíticamente, pero debe justificarse.

37.– Dadas dos secuencias ordenadas de *n* y *m* enteros, respectivamente. Diseñe un algoritmo ***Divide y Vencerás***, eficiente, que calcule el *k-ésimo* elemento de la mezcla resultante de las dos secuencias dadas. Calcule la complejidad del algoritmo desarrollado.

38.– Dadas dos secuencias ordenadas de *n* y *m* enteros, respectivamente. Diseñe un algoritmo ***Divide y Vencerás***, eficiente, que calcule la *mediana* de la mezcla resultante de las dos secuencias dadas. Calcule la complejidad del algoritmo desarrollado.

39.– Se tiene un vector *v* de números enteros distintos, con pesos asociados $p_1, \ldots, p_n$. Los pesos son valores no negativos y verifican que $\sum_{i=1}^{n} p_i = 1$. Se define la ***mediana ponderada*** del vector *v* como el valor *v[m], 1* $\leq m \leq n$, tal que

$$\sum_{v[i]<v[m]} p_i < \frac{1}{2} \quad \text{y} \quad \sum_{v[i]\leq v[m]} p_i \geq \frac{1}{2}$$

Por ejemplo, para $n = 5$, $v = [4, 2, 9, 3, 7]$ y $p = [0{,}15, 0{,}2, 0{,}3, 0{,}1, 0{,}25]$, la media ponderada es $v[5] = 7$ porque

$$\sum_{v[i]<7} p_i = p_1 + p_2 + p_4 = 0{,}15 + 0{,}2 + 0{,}1 = 0{,}45 < \frac{1}{2}, \text{ y}$$

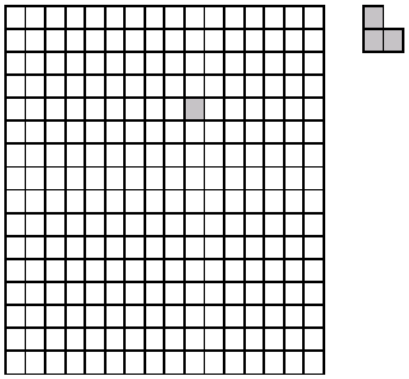
$$\sum_{v[i]\leq 7} p_i = p_1 + p_2 + p_4 + p_5 = 0{,}15 + 0{,}2 + 0{,}1 + 0{,}25 = 0{,}7 \geq \frac{1}{2}$$

Diseñe un algoritmo de tipo ***Divide y Vencerás*** que encuentre la mediana ponderada en un tiempo lineal en media. (Obsérvese que *v* puede no estar ordenado.) ¿Cómo se puede conseguir un coste lineal en tiempo en el caso peor?

40.– En una habitación oscura se tienen dos cajones en uno de los cuales hay *n* tornillos de varios tamaños, y en el otro las correspondientes *n* tuercas. Es necesario emparejar cada tornillo con su tuerca correspondiente, pero debido a la oscuridad no se pueden comparar tornillos con tornillos ni tuercas con tuercas, y la única comparación posible es la de intentar enroscar una tuerca en un tornillo para comprobar si es demasiado grande, demasiado pequeña, o se ajusta perfectamente al tornillo. Desarrolle un algoritmo ***Divide y Vencerás*** para emparejar los tornillos con las tuercas, que use O(*n log n*) comparaciones en término medio.

41.– En un jardín de infancia los niños se disponen a dormir la siesta después de comer. Para ello, el profesor les quita previamente los zapatos uno a uno y los guarda emparejados, por ejemplo, atando los cordones de cada par. Una vez terminada la siesta, el profesor debe poner de nuevo los zapatos a todos sus alumnos. El problema es que todos los zapatos son del mismo modelo y color, que es el del uniforme del jardín de infancia. Cuando intenta colocar un par de zapatos a un niño puede suceder que el par encaje perfectamente, que le queden grandes, o que le queden pequeños. El objetivo del profesor es que todos encajen perfectamente. Diseñe un algoritmo ***Divide y Vencerás*** que resuelva el problema suponiendo que cada par de zapatos solamente le sirve a un niño, que no use un número de comparaciones superior a O*(n log n)* en término medio. Calcule su complejidad.

42.– Tenemos un tablero cuadrado de dimensión 2^m (tiene, pues 2^{2m} celdas) y una tesela en forma de L, como puede observarse en la figura. Diseñe un algoritmo ***Divide y Vencerás***, que cubra todo el tablero utilizando teselas en forma de L como las de la figura, supuesto que ya tenemos cubierta una casilla.



43.– Cálculo del *n*-ésimo término de la sucesión de Fibonacci.

a) Diseñe un algoritmo "***Divide y Vencerás***" que calcule x^n con un coste $O(n \log n)$, en términos del número de multiplicaciones efectuadas.

b) Sea F la matriz $\begin{pmatrix} 0 & 1 \\ 1 & 1 \end{pmatrix}$. Considere el producto del vector (a b) por la matriz F. ¿Cuál es el resultado cuando a y b son dos términos consecutivos de la sucesión de Fibonacci.

c) Utilizando las ideas de los apartados anteriores, desarrolle un algoritmo "***Divide y Vencerás***" para calcular el *n*-ésimo término de la sucesión de Fibonacci. Calcule su coste en términos del número de operaciones aritméticas efectuadas.

44.– ***D.V. Loser*** es aficionado al juego y todas las noches va a jugar una partida de póquer. Como es un jugador responsable, para controlar sus pérdidas, anota lo que gana o pierde cada noche como un número entero de euros. Por ejemplo, en el último mes los resultados cosechados han sido los siguientes:

```
29 -7 14 21 30
-47 1 7 -39 23 -20 -36
-41 27 -34 7 48 35 -46
-16 32 18 5 -33 27 28
-22 1 -20 3 -42
```

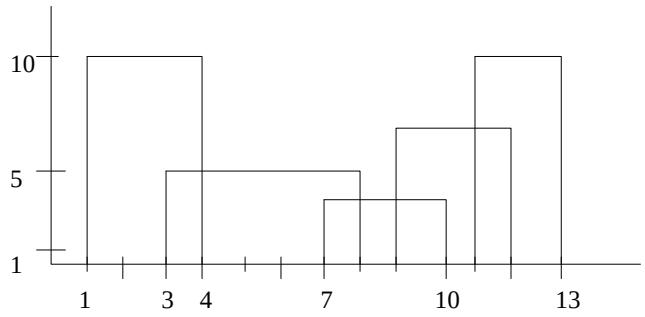
Se observa que empezó bien el mes con una ganancia de 29 euros, pero terminó con una pérdida de 42 euros. El beneficio total obtenido a lo largo de todo el mes es -47 euros.

Mirando esta información en retrospectiva, *D.V.* se da cuenta de que si hubiera empezado a jugar el día 16 y terminado el día 26, habría maximizado sus ganancias, obteniendo 105 euros, una diferencia de 155 euros con respecto a su situación actual.

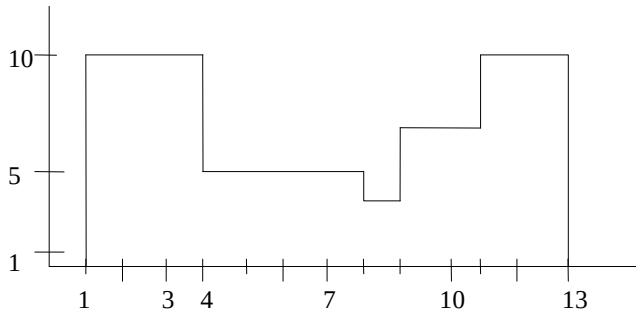
Dado un vector de ganancias/pérdidas, de longitud *N*, utilizando la técnica "***Divide y Vencerás***", se trata de devolver la secuencia de días (consecutivos) con los que se consigue **maximizar el beneficio total**, devolviendo los índices de comienzo y fin, y ese beneficio máximo. Calcule la complejidad del algoritmo desarrollado.

45.– Dada la localización exacta y la altura de varios edificios rectangulares de la ciudad, obtener la *skyline*, línea de recorte contra el cielo, que producen todos los edificios eliminando las líneas ocultas. *Ejemplo*: La entrada es una secuencia de edificios, donde un edificio se caracteriza por una tripleta de valores (x_{min},x_{max},h).

Una posible secuencia de entrada para los edificios de la siguiente figura podría ser : (7,10,3), (9,12,7), (1,4,10), (3,8,5) y (11,13,10).



Y ésta es la representación de la *skyline* de la salida que se debe producir:



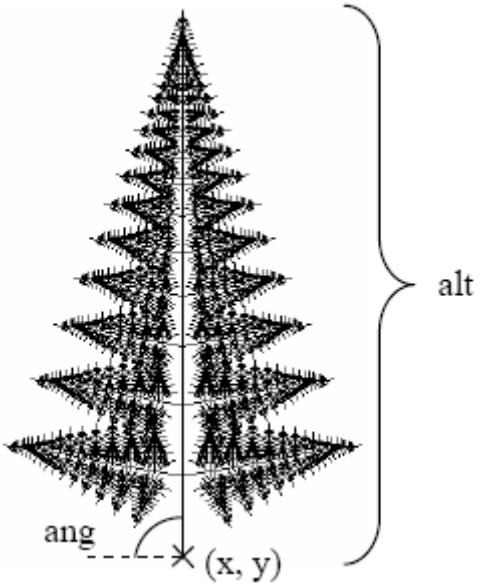
Su secuencia asociada es : (1,4,10), (4,8,5),(8,9,3), (9,11,7) y (11,13,10).

Diseñe un algoritmo ***Divide y Vencerás*** que resuelva el problema.

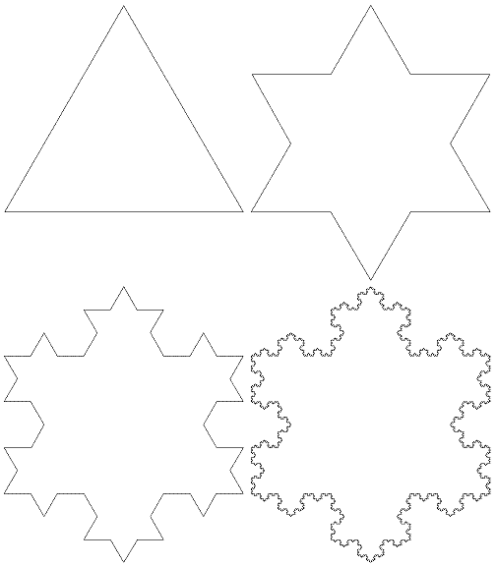
46.– Se considera la figura fractal "***helecho***" de la figura. Diseñe un algoritmo ***Divide y Vencerás*** que la dibuje. La cabecera de la función es de la forma:

void PintaHelecho (**float** x, **float** y, **float** ang, **float** alt)

que pinta un helecho de altura **alt**, cuyo tallo principal tiene la base en **(x, y)** y tiene ángulo **ang** respecto a la horizontal.



47.– Se considera la figura fractal "***copo de nieve***" que se muestra en la figura. Diseñe un algoritmo ***Divide y Vencerás*** que la dibuje.



48.– **Cierre Convexo**. Dado un conjunto de *n* puntos del plano. Diseñe un algoritmo ***Divide y Vencerás*** que compute el cierre convexo del mismo. Calcule la complejidad del algoritmo desarrollado.

1.– Aplique el algoritmo dado para el problema de la mochila en los siguientes casos:

a) M = 15, V = [10 5 15 7 6 18 3]

P = [2 3 5 7 1 4 1].

b) Seleccione un lote de 100 libros, lo más barato posible, de entre los siguientes lotes

Lote de 25 volúmenes	65.000	Ptas./Lote
Lote de 10 volúmenes	30.000	Ptas./Lote
Lote de 100 volúmenes	270.000	Ptas./Lote
Lote de 80 volúmenes	160.000	Ptas./Lote

2.– Construya procedimientos que resuelvan el problema de la **mochila**:

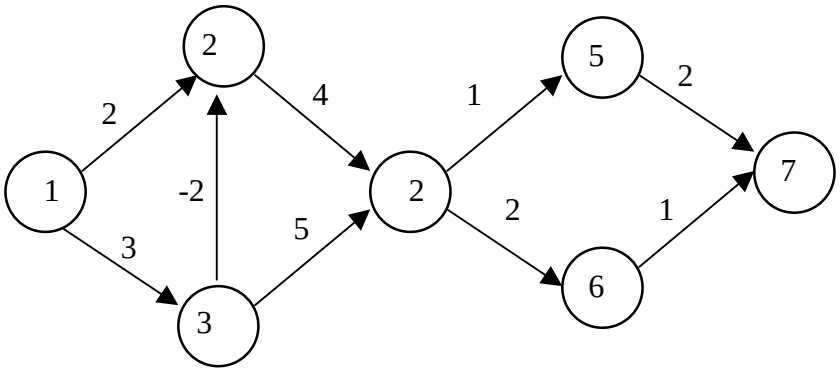
a) Ordenando primero los materiales (por precios).

b) Tomando los materiales que se supone se necesitarán. INDICACION: Una posible estimación de esta cantidad puede venir dada por

$$\frac{V}{\sum_{i=1}^{n} v_i}$$

3.– Plantee la función objetivo, para el problema de búsqueda de caminos óptimos con vértice común, como la suma de los arcos que forman los caminos de una solución factible. Diseñe un algoritmo devorador "inmediato" que obtenga una "buena" solución. Estime su calidad.

4.– Aplique el algoritmo de Dijkstra al siguiente grafo



para $x_0 = 1$. Busque otra solución mejor que la obtenida. ¿Por qué no se obtiene la solución óptima?

5.– Se quiere establecer una red de distribución de periódicos desde la ciudad A a otras 10 ciudades. Las distancias entre ellas en Kms. se da en el cuadro siguiente. Obtenga una red de distribución óptima para los casos en que

a) La distribución se hace directamente desde A a las demás ciudades, y

b) La distribución se va ramificando a través de las ciudades.

	A	B	C	D	E	F	G	H	I	J	K
B	40										
C	45										
D	50										
E	45										
F	30				40						
G		25				40					
H		60									
I			25	25				10			
J		50					20	10			
K	60			45	10						

6.– Implemente el **Algoritmo de Prim** para la obtención del árbol de expansión mínimo.

7.– Escriba un procedimiento Pascal que implemente el **Algoritmo de Huffman** para la obtención del **árbol extendido de mínimo peso**.

8.– **MEZCLA OPTIMA DE FICHEROS**.

Aplique el procedimiento anterior a la resolución del problema de la mezcla de N ficheros con tres canales: Tomamos dos ficheros X e Y, y devolvemos la mezcla de ambos en otro fichero Z. Cada fichero i tiene Q_i datos.

9.– Construya un procedimiento que decida si un grafo no dirigido dado es **conexo**.

10.– Un vértice p de un grafo dirigido G = (V,A) se dice que es un **pozo** o **sumidero** si hay arcos en el grafo que lo tienen como extremo y ninguno como origen. Diseñe un algoritmo que detecte la presencia de pozos en un grafo G.

11.– Un vértice p de un grafo dirigido G = (V,A) se dice que es un **manantial** o una **fuente** si hay arcos en el grafo que lo tienen como origen y ninguno como extremo. Diseñe un algoritmo que detecte la presencia de manantiales en un grafo G.

12.– **Planificación de Tareas con Caducidad**. Se tiene una serie de *N* trabajos cada uno de los cuales proporciona un beneficio b_i siempre y cuando se ejecute a lo más en un instante d_i, transcurrido el cual, su ejecución no proporcionaría ningún beneficio. En cada instante $t= 1,2,...,T$, se realiza un trabajo. Se trata de hallar los trabajos que hemos de realizar y la secuencia en que debemos hacerlos para obtener el máximo beneficio posible y calcular dicho beneficio. *N* y *T* son fijos.

13.– Los profesores de Algorítmica no saben que hacer para tener una estimación de las notas de sus alumnos. En un momento de locura han creído descubrir un buen método para hacer esta estimación.

Puesto que por las prácticas realizadas en el laboratorio pueden tener una idea aproximada de los conocimientos de cada alumno, saben que **dado un problema *P*, si un alumno es capaz de resolverlo, emplea en su resolución un tiempo t_P**, no admitirán medias tintas, un problema estará o Bien o Mal resuelto según esté o no completamente resuelto. Se entiende asimismo que un alumno puede resolver un problema y otro no, aunque este último tenga en las prácticas una nota global superior.

El examen constará de *M* problemas cada uno de los cuales tendrá un peso de *1/M* en la calificación global.

Se pide:

A) Si la dinámica del examen consiste en entregar un problema y proceder a su recogida transcurrido un tiempo, que dependerá del problema, y a continuación, proceder de la misma forma con el siguiente problema.

Dado un examen de *M* problemas calcular el número de aprobados esperado y los alumnos que superarían el examen junto con los ejercicios que resolverían.

B) Los alumnos han mostrado su disgusto por la poca libertad que les da el método de evaluación anterior, pidiendo que todos los problemas se les den al comenzar el examen y entregen su solución al final del examen. Si los profesores accediesen a su petición, calcular:

   i) el número máximo de aprobados,

   ii) el número mínimo de aprobados

En ambos casos debe también decirse cuáles son los alumnos que superarían el examen y justificar los algoritmos dados para i) y ii). Como en a) deben facilitarse los datos concernientes a cada alumno, esto es, cuál sería la secuencia de resolución de ejercicios propuestos en el examen.

C) Algunos alumnos han propuesto que se les dé el examen de antemano, a lo que los profesores, como era previsible, no han accedido. Pero han realizado una contraoferta, consistente en que no les darán los ejercicios, sino la estimación que han hecho sobre cada problema para cada alumno, esto es, el tiempo que necesita para resolverlo, y deben elegir entre N problemas los M que compondrán el examen. Remisos los alumnos a aceptar la contraoferta, los profesores han mejorado la misma añadiendo que en el examen se propondrán los *N* problemas entre los que se elegirán los ejercicios del examen, pero cada

alumno habra de resolver sólo $M < N$. El examen tendrá una duración fijada por los profesores de antemano que también ha sido facilitada como dato a los alumnos. Calcular:

i) el número máximo de aprobados,

ii) el número mínimo de aprobados

Los datos que deben darse como solución a c) son los mismos que en b).

14.– Disponemos de un ordenador multitarea con N procesadores, cada uno de los cuales puede direccionar una cantidad de memoria de P_i Kbytes ($1 \leq i \leq N$). Queremos procesar N trabajos, cada uno de los cuales necesita m_i Kbytes de memoria. Evidentemente, para poder procesar un trabajo en un procesador, la cantidad de memoria que aquel necesita debe ser a lo sumo la que éste puede direccionar, además alguno de los trabajos es posible que no pueda ser procesado. Un procesador sólo puede realizar un trabajo del lote.

Diseñe un algoritmo voraz que asigne cada trabajo a un procesador, de forma tal que el número de trabajos que no pueden ser procesados sea mínimo. Pruébese que el algoritmo diseñado encuentra la solución óptima o encuéntrese un contraejemplo en caso contrario.

15.– **CUBRIMIENTO DE NODOS**.

Dado un grafo $G=(V,A)$ no dirigido, un subconjunto C de V es un "Cubrimiento de Nodos" si cada arista de A tiene al menos un extremo en C. Plantee el problema y diseñe un algoritmo devorador que obtenga un cubrimiento de un grafo dado, tratando de que tenga la menor cantidad posible de nodos. Estime la calidad de la solución.

16.– **Aplicaciones del cubrimiento de nodos**.

A) Una agencia de información tiene N sedes en varias ciudades del mundo. Cada sede puede comunicarse con otras (una o varias a la vez) de la misma agencia, para recibir o mandar información, a través de un terminal que en ella existe. Una red de información activa (RIA) es la que forman un número de terminales (nodos de la red) que en un determinado momento estén mandando o recibiendo información. Una arista de la red sería un flujo de información directa (FID) entre dos terminales. Cuando una RIA está constituida por los N terminales que existen puede producirse un colapso en la red, el cual solo se soluciona desconectando al menos uno de los dos extremos (terminales) de cada FID. El coste de este colapso lo mide el número de terminales que deben ser desconectadas. Se pide:

i) Plantee el problema y Diseñe un algoritmo devorador "rápido" (de complejidad no superior a cuadrática) que, dada una RIA (implementada de alguna forma que debe precisarse), determine qué terminales deben desconectarse para solucionar un colapso, intentando que el coste sea el menor posible. Supóngase que los terminales están numerados de 1 a N.

ii) Estime la calidad del algoritmo: Complejidad en tiempo y calidad de la solución (contraejemplo si no es óptima y demostración si lo es).

B) Para los juegos olímpicos son necesarios un total de *n* intérpretes (que deberán saber alguno de los *k* idiomas oficiales). De este grupo existirá un subgrupo que se encargará de recibir instrucciones generales, para transmitírselas a los demás intérpretes. Es deseable que este subgrupo sea lo más reducido posible. Se pide lo mismo que el caso A, para el problema de obtener un subgrupo de intérpretes, con las mismas restricciones de complejidad.

NOTA: Aunque si se resuelve el problema de obtener un subgrupo de intérpretes, lo más reducido posible, que sepa los k idiomas, el problema original queda resuelto, no plantearlo con este fin, ya que ofrece una solución demasiado trivial. Piénsese que si todos los intérpretes saben, por ejemplo, Inglés, bastaría con un sólo intérprete para formar el subgrupo deseado, sin que tenga que saber todos los idiomas.

17.– En la Alta Edad Media, y en una región dividida en *N* reinos, un intrigante Cardenal planea establecer una alianza internacional entre todos ellos basándose en una política matrimonial. Cada monarca tiene un único hijo y una o varias hijas, cada hijo tiene un determinado Peso. La planificación es la siguiente:

i) Se establece una alianza entre dos reinos si el hijo del rey de uno se casa con una hija del rey de otro.

ii) Si dos reinos están aliados, todos los aliados de uno también lo están con todos los aliados del otro.

iii) El peso de un matrimonio es la medida de su estabilidad.

iv) El Cardenal pretende unir todos los reinos consiguiendo la mayor estabilidad posible.

Diseñe un algoritmo devorador para resolver el problema del Cardenal. Calcule el orden de complejidad del mismo y probar si la solución obtenida es óptima.

18.– Se define el **diámetro** de un árbol binario como la longitud del camino más largo entre dos nodos cualesquiera del mismo. Desarrolle una estrategia voraz que calcule el diámetro de un árbol con una complejidad lineal con el número de nodos del árbol binario.

19.– Una sala ha de estar constantemente iluminada por un mínimo de *M* bombillas. Para ello, disponemos de un conjunto finito de *N* bombillas $\{b_i\}$ que una vez encendidas no pueden apagarse y que se gastan al cabo de t_i días. Se trata de maximizar el número de días que podemos tener la sala iluminada.

Resuelva el problema usando una estrategia voraz eficiente, justificándola. Calcule la complejidad del algoritmo desarrollado. Razone si la estrategia es óptima, y en caso contrario, exponga un contraejemplo.

20.– Partiendo de una sucesión *S* de *N* números enteros y un conjunto A de *N* números, reordene los elementos de A en una sucesión *S'* de manera que si se comparan los pares S_i y S_i', para todo $i \in \{1,...,N\}$, se verifica:

$S_i \leq S_i$' para un mayor número de veces posibles.

Desarrolle una estrategia voraz para resolver el problema y prográmela.

NOTA: Sin pérdida de generalidad se puede suponer A ordenado en un vector. Por ejemplo:

A	0	1	1	2	3	8	10	11	12	13
S	1	3	7	10	9	2	4	17	14	10
S'	1	3	8	10	11	2	12	0	1	13

21.– **Cubrimiento de Conjuntos**. Sea $S = \{S_i\}_{i \in I}$, una colección finita de conjuntos finitos. Obténgase un recubrimiento mínimo de la misma.

Un **recubrimiento** de S es subcolección de la misma $S' = \{S'_j\}_{j \in J}$, (i.e. $\forall j \in J \; \exists \; i \in I,, S'_j = S_i$), tal que todo conjunto de S esté contenido en la unión de los conjuntos de S'.

Un **recubrimiento** S' de S se dice **mínimo** si el cardinal de S' es mínimo entre todos los recubrimientos de S, es decir, el número de conjuntos de la subcolección S' es mínimo.

Por ejemplo: Sea S = {{1,3,4}, {2,4,6}, {2,4}, {1,3}, {5,7}, {4,6}}. Recubrimientos del mismo son:

S' = {{1,3,4}, {2,4}, {5,7}, {4,6}},

S'' = {{1,3}, {2,4,6}, {5,7}}

S''' = {{1,3}, {2,4}, {5,7}, {4,6}}

Diseñe una **heurística voraz** para resolver el problema. Exponga un contraejemplo en el caso de que el algoritmo voraz no devuelva la solución óptima.

22.– En un campo tenemos un número determinado de flores, cada una con una determinada cantidad de polen. Tenemos un zángano que guiado por la ley del mínimo esfuerzo, sólo vuela a las flores adyacentes. Supuesto que debe saciar su apetito con una cantidad de polen *P*, visitando el menor número de flores, diseñe una **heurística voraz** que le ayude a conseguir su objetivo. Supóngase que parte de una flor dada.

¿Y si el zángano puede elegir la primera flor? Si quitamos la restricción de la elección a las flores adyacentes, ¿cuál sería la solución?

Calcule la complejidad del algoritmo desarrollado en cada uno de los casos.

23.– Tenemos dos bolsas capaces de soportar hasta un peso máximo *Pmax*, y un conjunto de *N* objetos de cada uno de los cuales conocemos su peso p_i. Se trata de decidir qué objetos se

ponen en cada bolsa para que el peso en ambas esté lo más equilibrado posible, sin sobrepasar el peso máximo *Pmax*, sabiendo que el peso de los objetos de cada bolsa debe ser al menos *Pmin* (*Pmin* ≤ *Pmax*). Resuelva el problema mediante una **heurística voraz**, y demuestre la optimalidad de la estrategia o exponga un contraejemplo. Calcule asimismo la complejidad del algoritmo desarrollado.

24.– Un camionero que está en Babia quiere regresar a su casa, que está por los cerros de Úbeda, siguiendo una ruta dada y llevando un camión que le permite, con el depósito lleno, recorrer *k* kilómetros sin repostar. El camionero dispone de un mapa de carreteras que le indica las distancias entre las gasolineras que hay en su ruta. Como va con prisa, el camionero desea pararse a repostar el menor número de veces posible.

Diseñe e implemente un **algoritmo voraz** que nos diga en qué gasolineras tiene que parar. Demuestre que el algoritmo encuentra siempre la solución optima y calcule la complejidad de dicho algoritmo.

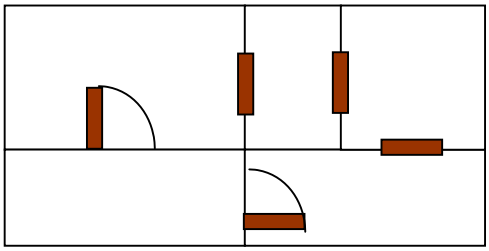
25.– Se considera una secuencia de palabras $p_1, p_2, ..., p_n$, de longitudes $l_1, l_2, ..., l_n$. Se desea agruparlas en líneas de longitud *L*. Las palabras están separadas por espacios cuya amplitud ideal (en milímetros) es *b*, pero los espacios pueden reducirse o ampliarse si es necesario (aunque sin solapamiento entre palabras), de tal forma que una línea $p_i, p_{i+1}, ..., p_j$ tenga exactamente longitud *L*. Sin embargo, existe una penalización por reducción o ampliación del número total de espacios que aparecen o desaparecen. El coste de fijar una línea $p_i, p_{i+1}, ..., p_j$ es $(j - i)|b' - b|$, siendo *b'* el ancho real de los espacios, es decir, $(L - l_i - l_{i+1} - ... - l_j)/(j - i)$. No obstante, si $j = n$ (la última palabra) el coste será cero a menos que $b' < b$ (ya que no es necesario ampliar la última línea).

Diseñe e implemente un **algoritmo voraz** que resuelva el problema. Demuestre la optimalidad de solución proporcionada o en caso contrario, exponga un contraejemplo.

26.– Para *Forrest Gump* la vida era como una caja de bombones. Cada caja tenía bombones de varias clases. *Forrest* frecuentaba una tienda que tenía muchas cajas distintas, y *Forrest* siempre deseó tener bombones de todas las clases, pero su paga no le alcanzaba. Por ello, tenía que comprar **el menor número de cajas** pues todas las cajas tenían el mismo precio. Diseñe una **heurística voraz** que le diga a *Forrest* qué cajas debe comprar y cuánto le costará comprarlas.

27.– Los *Reyes Magos* tienen sus encargos. Necesitan repartir un conjunto de *N* regalos voluminosos y de considerable peso. Para ello, pueden utilizar hasta *M* camellos, cada uno de los cuales puede soportar un peso dado (no es el mismo para todos los camellos). Los *Magos* deben de repartir la mayor cantidad posible de regalos. Se trata de decir cuál es la distribución de los objetos en los camellos, supuesto que en cada camello va un sólo objeto. Diseñe una ***estrategia voraz*** que resuelva el problema y calcule su complejidad.

28.– El plano de una vivienda puede verse como un conjunto de habitaciones rectangulares, que comparten paredes unas con otras, y que tienen una o más puertas para pasar de una habitación a otra. Se trata de calcular la temperatura que habrá en cada habitación partiendo de una temperatura inicial y de una determinada situación de cada puerta (abierta/cerrada). Diseñe un algoritmo que calcule la temperatura final de cada habitación, sabiendo que la temperatura de aquellas habitaciones que compartan una puerta abierta debe calcularse mediante una sencilla media aritmética ponderada al volumen de la habitación. Nótese que el número de habitaciones conectadas puede ser de dos o más.



29.– El concurso "*Los Glotones*", el sueño de los golosos, consiste en que cada equipo se coma, en un tiempo dado, la mayor cantidad de tartas. Cada componente del equipo debe comerse una tarta y sólo una tarta, tarta que solo se contabiliza si se la come completamente.

Al mismo se han dirigido los GVI (Glotones Voraces de Informática, GGP o G2P en terminología anglosajona), que han formado un equipo, pues les gustan más los pasteles que las cucarachas, y se han propuesto ganar el concurso aplicando sus conocimientos informáticos.

Para ello, han hecho una serie de pruebas para determinar la cantidad de tarta que cada uno es capaz de ingerir en el tiempo dado por el concurso. También han llevado a un ojeador, que además es el capitán del equipo, capaz de hacer una estimación, al instante, del peso de cada tarta. El capitán del equipo debe asignar qué componente del equipo se comerá cada tarta (las tartas están numeradas).

Diseñe el algoritmo que utilizará el capitán para realizar la asignación de los componentes del equipo a las tartas, para maximizar el número de tartas "devoradas" por completo. Calcule la complejidad del algoritmo, que debe ser rápido, porque el tiempo empieza a correr en cuanto se enseñan las tartas.

30.– ¡**Nos invaden**! El enemigo, armado hasta los dientes con palos y piedras ha desembarcado en las costas de nuestro país invadiendo N ciudades. Los servicios de "inteligencia" están informados que en cada una de las ciudades invadidas se encuentran e_i efectivos enemigos. Para contraatacar, el grupo de intervención rápida de defensa dispone de N equipos listos para intervenir. Cada uno de ellos consta de d_j efectivos completamente equipados y entrenados. Para garantizar el éxito de la intervención en una ciudad es necesario que contemos al menos con tantos efectivos de defensa como el enemigo.

Diseñe un **algoritmo voraz** que indique qué grupo de intervención debe ir a cada ciudad de forma que se maximice el número de éxitos garantizados. Asimismo, calcule la complejidad del algoritmo desarrollado.

38.– Se va a celebrar un maratón de cine en el que se van a proyectar durante todo un día películas, todas diferentes, en las *N* salas disponibles. Cada película se proyecta una única vez, y de ella se conoce la duración, la hora de comienzo y la sala en la que se proyectará. Nuestro viejo conocido, Juan, "*el gorrilla*", que es un cinéfilo empedernido. Harto de tanto mundial, con las pelas que se ha sacado de aparcacoches se ha comprado un bono que le permite ver todas las películas que desee. Y piensa sacarle el máximo partido al bono **viendo el mayor número de películas posibles**.

Diseñe un ***Algoritmo Voraz*** que ayude a Juan "*el gorrilla*" a conseguir su **objetivo** (**ver el máximo número posible de películas**). ¿Proporciona el algoritmo una solución óptima? Calcule la complejidad del mismo.

39.– Disponemos de *n* cajas distintas, cada una de las cuales tiene un peso P_i y puede soportar una carga encima de ella C_i. Diseñe una **heurística voraz** que sea capaz de encontrar la disposición de un subconjunto de las *n* cajas de manera que se apile el mayor número de cajas. Calcule la complejidad del algoritmo.

40.– Manolo y Benito necesitan hacer una "ñiapa" en el tejado del *Empire State*, y para ello nada mejor que montar un andamio desde el piso bajo hasta el tejado, ¿verdad?

En la furgoneta llevan un numero limitado de piezas para montar los andamios hasta llegar a esa altura, pero Benito, en su línea de seguir la *Ley del Mínimo Esfuerzo* quiere hacerlo con las mínimas piezas posibles. Conocidos para cada pieza el Peso que Soporta, el Peso de la pieza y su Altura, realice un programa usando un **Algoritmo Voraz** que le diga a Benito qué piezas usar para **montar el andamio de forma que aguante el peso de ambos, se use el menor número de piezas posibles y se llegue hasta el tejado del Empire State**.

41.– El encargado de una red de concesionarios ha decidido rebajar el precio de una serie de *N* coches que no han sido vendidos y que se han pasado de moda. Los coches son voluminosos, con una longitud y un peso determinados. Para la distribución de los coches se utilizan *M trailers*. Cada trailer tiene una longitud dada, y puede soportar un peso determinado (no tienen porqué ser los mismos para todos). Se deben repartir la mayor cantidad posible de coches. Diseñe una **heurística voraz** para resolver el problema. Calcule la complejidad del algoritmo desarrollado.

42.– Disponemos de n cajas de longitud C_i y anchura M cada una. Se pretende empaquetar K objetos, de longitud O_j ($O_j > M$) y anchura M cada uno, utilizando el mínimo número de cajas. Diseñe un algoritmo, utilizando una **heurística voraz**, que nos de la mejor disposición de los objetos en las cajas para conseguir el objetivo de utilizar el menor número de cajas. ¿Es óptimo? Calcule su complejidad.

43.– Debido a una avería en el Sistema Informático de Control del Tráfico Aéreo, los pilotos de los aviones no saben qué ruta es la adecuada para su vuelo. Afortunadamente *Alavi Oncito* es el controlador al mando. No pudo entrar en la Escuela de Pilotos por su reducida estatura, tuvo un desarrollo tardío, aunque después creció tanto que al encargado de las pruebas le sacaría la cabeza y parte del cuello (en sueños, no figuradamente). El caso es que los N aviones tienen un destino común para todos, pero hay diferentes rutas, y cada avión sólo puede volar en un rango de alturas para hacerlo con seguridad. Disponemos pues de M rutas al destino, que están determinadas para hacerse a una altura determinada, y en las que, por razones de seguridad, no podrá haber más de un avión. Diseñe un ***algoritmo voraz*** para evitar el mayor número de problemas en los vuelos.

44.– En la ciudad ha habido una invasión de *vampiros*, motivo por el cuál *La Autoridad* ha acordado permitir que en el jardín de cada casa (e incluso en los edificios de la Universidad) se puedan plantar ajos para espantar a los "*malignos*".

Así, en la reciente construcción de una nueva urbanización, financiada por el ayuntamiento, se ha tenido en cuenta plantar en la parcela de cada casa ajos que protegerán a sus habitantes del ataque de los vampiros. Aunque tal medida tiene como contrapartida un determinado olor a ajo en cada casa.

Las casas que cuentan con financiación del ayuntamiento deben ser asignadas a las familias que las han solicitado. Para evitar quejas (e incluso abandono de la casas de la familia a la que se ha asignado) **se debe cumplir que el olor a ajo de la casa debe ser soportado por la familia que la habite**. Para cada familia tenemos el grado de tolerancia al olor a ajo.

Se trata de **maximizar el número de casas asignadas** a las familias, con la restricción anterior. Resuelva el problema de forma eficiente en tiempo y espacio. **Calcule la complejidad del algoritmo desarrollado**. **NOTA**: Cada casa se asigna a una única familia.

45.– **Viernes13.com**. *Jason* había decidido cambiar sus hábitos. Últimamente le habían estado comiendo el tarro con las bondades de la aplicación de la informática a problemas de optimización, y él que siempre estaba pensando en lo mismo, pensaba ir ahora por los cibercafés y otros garitos.com.

Para ello, se había provisto de un plano en el que se detallaba la distancia (en tiempo) t_{ij} entre los garitos.com, y también tenía una estimación del número de posibles víctimas de cada garito.com para esta noche v_i, porque *Jason* sólo tenía esta noche para cometer sus fechorías. Y también sabe el tiempo que tardará en "liquidar" a todas las víctimas de cada garito t_i (distinto para cada garito).

Había pensado en un algoritmo de rápida ejecución en encontrar la secuencia de garitos.como a visitar. Esto es, una **heurística voraz**, tratando siempre de "cazar" la mayor cantidad de víctimas a lo largo de toda la noche. Programe un algoritmo que nos devuelva la secuencia de garitos.com que visitará *Jason*, suponiendo que puede empezar por uno cualquiera de los garitos.com. Razone la optimalidad del algoritmo diseñado, y en caso de que no lo sea, exponga un contraejemplo.

46.– Con motivo de las fiestas de san Fermín, se ha ideado una campaña de promoción de los vinos de la tierra, que consiste en que cada taberna ofrece una serie de "paquetes" de chupitos. Cada paquete tiene una serie de chupitos distintos, y cada chupito tiene una graduación alcohólica dada. Un mismo tipo de vino puede aparecer en distintos paquetes, y lo mismo en distintas tabernas.

Beb Edor y su amigo *Bor Achon*, acaban de llegar a Pamplona para correr los encierros, y no piensan desaprovechar la oportunidad de probar cuantos más caldos distintos mejor. Ahora bien, puesto que piensan correr los sanfermines no sobrepasarán una graduación alcohólica total de *G* grados. Además, para quedar bien en la taberna, si se piden un paquete de chupitos, se lo beberán entero.

Puesto que ambos no están para cálculos complicados, piensan seguir una **estrategia voraz** para lograr su objetivo de beber la mayor cantidad de chupitos distintos sin sobrepasar la graduación alcohólica *G aprovechando las ofertas de las tabernas*, como buenos amigos que son se van a pedir los mismos paquetes.

47.– Se tiene un conjunto de palabras que se quieren agrupar según su grado de semejanza. Desarrolle una **heurística voraz** de manera que las palabras queden agrupadas hasta un umbral dado δ, sabiendo que al unir dos grupos, la semejanza del nuevo grupo con todos los demás es el mínimo de los dos valores de semejanza.

La **semejanza** o proximidad de dos palabras $a = a_1a_2 ...a_m$ y $b = b_1b_2 ...b_n$ está dada por:

$$\frac{2Card(A \cap B)}{Card(A)+Card(B)}, \text{ siendo } A = \{ a_ia_{i+1} / 1 \le i \le m\text{-}1\} \text{ y } B = \{ b_ib_{i+1} / 1 \le i \le n\text{-}1\}$$

Nótese que *A* y *B* son conjuntos, y por tanto, no tienen elementos repetidos.

49.– **Red de Espías**. Los servicios de Contraespionaje han detectado que una potencia "*amiga*" ha creado una tupida red de espionaje en el aparato de Seguridad del Estado. Cada espía tiene una serie de contactos con otros espías. *SuperBond*, ante tales revelaciones, se ha mostrado muy contrariado ante sus colaboradores por un aparato de seguridad trufado, que parece un *QG*, un *queso gruyère*.

En un primer momento *SuperBond* había pensado en encargarle a la *TIA* la desarticulación de la red de espías, pero debido a que Mortadelo y Filemón se encuentran en la difícil misión de proteger al *SuperJefe* en su primer encuentro al más alto nivel con precisamente dicha potencia "*amiga*", se ha decidido por encargar la misión de neutralización al *Servicio de Inteligencia Militar e Investigaciones Off* (*SIMIO*, también conocido como *SIMeIO*), que últimamente debido al escaso presupuesto, y por motivos promocionales, está colaborando con voluntarios del *Servicio Social*; habiendo creado una unidad al efecto, la *SIMSESO*.

Aunque dispuesto *SuperBond* a que no se produzcan dispendios, les ha planteado que le den soluciones a diversas alternativas, porque ya ha tenido realizar cierta "*inversión*" en comprar la información (a otra potencia "amiga" de ambas) relativa a la estructura de la red. Esto es, los contactos que mantienen entre sí los espías, y su lugar de trabajo, y otros asuntos. Pero por más que ha insistido no ha podido "sacarles" la identidad de los *topos*. (NOTA: Asumimos que un espía aislado no genera ningún peligro puesto que no puede transmitir información.)

a) El primer supuesto es que *SuperBond* no está dispuesto más que a liberar recursos para neutralizar *M* espías. Así, se trata de decidir cuáles son los *M* espías a neutralizar para causar el mayor daño a la red, esto es, romper el mayor número de contactos.

b) El segundo supuesto es cuantos espías deberían neutralizar para destruir completamente la red de espionaje.

Resuélvase cada uno de los apartados utilizando una ***Heurística Voraz***. Debe calcularse la complejidad del algoritmo desarrollado (basta con uno de los apartados).

50.– En una clase de primaria el maestro se da cuenta de que dejando a los alumnos sentarse como quieran, la clase se le "sube a las barbas". Para evitarlo ha decidido hacer él la asignación de los alumnos a los pupitres. En cada pupitre se pueden sentar dos alumnos como máximo. Hay un total de *T* pupitres y *A* alumnos. Con la observación del comportamiento de los alumnos, el profesor ha completado una tabla, *C*, del efecto que cada alumno produce sobre cada uno de los demás cuando se sientan juntos. El efecto puede ser un número positivo (le ayuda con los ejercicios, no habla con él, ...) o negativo (le mete cizaña, habla con él, le quita la goma, ...). Para algunos alumnos no ha podido comprobar el efecto de que estén sentados juntos. Si no se tiene información acerca de poner juntos *a* y *b*, tenemos $C[a, b] = 0$.

El objetivo es hacer una asignación de los alumnos a los pupitres (a cada pupitre se asignará cero, uno o dos alumnos) de una manera rápida, intentando **maximizar el efecto positivo**. Diseñe un algoritmo basado en la técnica voraz para resolver el problema. El algoritmo

diseñado, ¿garantiza la obtención de la asignación óptima? Calcule la complejidad del algoritmo desarrollado.

51.– El *Jaro* "*Mantalombro*" es el alumno más "*gorrón*" de la Universidad. Su dedicación durante todo el día es estar tumbado en el sofá jugando a las consolas. Recientemente ha adquirido una consola "*Polystation*" de última generación, ya que había conseguido pasarse todos los juegos de las otras consolas que poseía. Hasta aquí todo bien, pero no había caído en un pequeño detalle y es que no poseía ningún juego para esta nueva consola y no está dispuesto a perder ni un solo minuto, ya que su cerebro le solicita jugar en todo momento. Su idea es la siguiente: sus "*contactos*" han localizado a *N* posibles "*pardillos*" de los cuales puede aprovecharse para grabar juegos compatibles con la "*polystation*". Cada pardillo *p* de estos posee m_p juegos y los contactos estiman un tiempo t_i en grabarlos. Plantee una ***estrategia voraz*** que le permita al "*Jaro*" grabarse la mayor cantidad de juegos distintos suponiendo que tiene un tiempo *máximo T*.

52.– José Peco Pión, piensa aprobar la asignatura sin más esfuerzo que el necesario para "*trasladar*" las ideas de otros a su examen. Para ello, en vez de estudiar ha ideado un procedimiento de "*traslación*" de respuestas de los problemas del examen, estableciendo previamente acuerdos con los posibles sujetos que pueden resolver los problemas ("empollones", en lo que sigue), y que también concurren al examen, que a cambio de una módica cantidad dineraria (dependiente de cada empollón), le entregarán uno de los problemas resueltos correctamente.

El procedimiento ideado consiste en hacer "*transacciones*" con empollones distantes al menos *DMin* metros (los "vigilantes" han tomado nota de las posiciones tomadas por los concurrentes al examen, y concienzudamente comprueban todos los exámenes que distan menos de la citada distancia *Dmin*) y menos de *DMax* metros (su movimiento sería detectado por algún vigilante). La transacción se reducirá a un solo ejercicio por empollón, para evitar problemas derivados de la "similitud" de los exámenes, y que se destape el "asunto".

En el examen se han propuesto *N* problemas, de los que se tienen que **resolver *M* correctamente para aprobar**. Cada "empollón" sólo está dispuesto a intercambiar unos cuantos problemas de entre los *N*, no necesariamente todos.

El tiempo que tarda en realizarse cada "intercambio" $T_{Intercambio}$ puede considerarse constante, y es muy inferior, aunque no debe ignorarse, con respecto al tiempo que se necesita para "trasladar" la respuesta del empollón, al examen de Peco (tiempo que tarda en copiarlo Peco), que está dado por t_{pe}, para el problema *p* resuelto por el empollón *e*. El importe cobrado por un empollón *e* es el mismo para cualquiera de los problemas que está dispuesto a intercambiar, C_e.

Resuelva el problema utilizando una estrategia **voraz**, supuesto que dispone de un tiempo *T* para realizar el examen (se conforma con aprobar) y desea gastar la menor cantidad de dinero. Calcule la complejidad del algoritmo desarrollado.

53.– Darío, "*el Presi*", después de fallar por sexagésima-sexta vez en su intento de fuga, ha sido obligado a trabajar como cartero dentro de la prisión. Su trabajo consiste en repartir la

correspondencia a todos los reclusos de la prisión. Para Darío, esto es una contrariedad, ya que no le permite concentrarse en su *hobby* preferido, planear una nueva fuga. Por esto, ha decidido realizar su trabajo de forma correcta (para evitar represalias por parte del alcaide de la prisión), pero empleando el menor tiempo posible. Para planificar el reparto de hoy, Darío ha estado midiendo la distancia existente entre los distintos puntos de entrega de la correspondencia, y ha construido una matriz de distancias, con el tiempo de desplazamiento, y el tiempo empleado en entregar la correspondencia en cada punto. Ayude a Darío a construir un algoritmo, basado en una ***Heurística Voraz***, que le permita repartir toda la correspondencia en el menor tiempo posible, sabiendo que Darío cuenta con un carrito en el que puede llevar toda la correspondencia.

54.– Nos encontramos en época de sequía y la Junta de Regantes ha racionado el suministro de agua. Concretamente a la finca en que realizan su trabajo los reclusos, en situación de Tercer Grado, de la prisión en que se encuentra recluido Darío "*El Presi*", le ha sido asignado un cupo de T unidades de agua. La finca está compuesta por un conjunto de P parcelas. Cada parcela p tiene una extensión e_p. En cada parcela hay sembrado un cultivo (al principio no estaban previstas restricciones de agua). De cada cultivo c se sabe tanto la necesidad de agua por hectárea, a_c, como su rendimiento neto (en euros), r_c, también por hectárea.

Lamentablemente, se sabe que si las necesidades de agua de un cultivo no son totalmente satisfechas, se perderá completamente la cosecha; no obteniéndose pues, rendimiento alguno. De la misma forma, y dado el diseño del sistema de regadíos de la finca, aunque es posible regar todas las parcelas al mismo tiempo, es imposible regar sólo una parte de una parcela, pues el agua dedicada a una parcela se distribuye uniformemente por ella.

El alcaide pretende **maximizar los beneficios que se pueden obtener de los cultivos de la finca de la prisión**, puesto que los ingresos esperados por los cultivos iban a ser empleados, a partes iguales, –según el alcaide–, en "rehabilitaciones y mejoras de las viviendas de los empleados de la prisión", y en "mejoras de las instalaciones penitenciarias". Todo ello, tras restar los "*gastos varios*" (que se supone que superarán el 30%, pues se rumorea incluyen un 20% para las dependencias [privadas] del alcaide, y alguna que otra comisión del 3%).

Así pues, y para motivar a los reclusos y hacerles partícipes de los éxitos de su gestión económica –y no sólo rehabilitadora–, ha convocado un **concurso entre los reclusos para determinar cuáles son las parcelas que hay que regar para maximizar la productividad (monetaria) de la finca**. El premio del concurso es una reducción de la pena a la mejor propuesta, junto con una celda individual de algo más de 25m^2 durante un año, sin compensación económica, puesto que según dice no estaría bien visto que los reclusos se lucraran gracias al sistema penitenciario, pues bien pudiera acabar derivando en un incremento de la criminalidad.

Conocidas la condiciones, Darío "*El Presi*", recordando las conversaciones (o intercambio de informaciones) que mantenía con su compañero de celda, "*El Teclas*", alias "*Rekcah Inverso*" en el exterior ("*El Teclas*" es un genio de la Informática con el que en tiempos pasados compartió celda, afortunadamente rehabilitado para la sociedad, y que realiza trabajos para alguna que otra empresa de seguridad. Aunque también es cierto que de vez en cuando se relaciona con "*El Puertas*", también conocido en ciertas esferas por "*El Señor del Lado Oscuro*"). Darío, en aquellas conversaciones había adquirido algún que otro conocimiento de programación, y estaba dispuesto a desarrollar un algoritmo basado en una heurística voraz para conseguir la intimidad que proporciona una celda de 25m^2. ***Diseñe uno de los buenos***

algoritmos basado en una heurística voraz de los que podría crear Darío para resolver el problema. Calcule la complejidad del algoritmo desarrollado.

55.– *Papá Noel (Santa* para los amigos) y su banda de rufianes renos, entre los que se encuentra el malvado *Rudor* está en crisis. Los tiempos cambian y los niños de hoy en día prefieren los regalos de *Los Reyes Magos*. En una acción desesperada por conseguir el mercado de los regalos, *La banda de Santa* intentará dar el golpe de su vida robándole el mayor número de fábricas de regalos a *Los Reyes Magos*.

Disponen de N grupos de renos listos para atacar. Cada uno de estos grupos consta de R_i renos enloquecidos por la furia de la victoria. El día clave para llevar a cabo la misión "*Feliz Año Nuevo*" será el 1 de Enero. Intentarán maximizar el número de fábricas conquistadas consiguiendo así sus regalos. *Los Magos* tienen M fábricas cada una defendida por C_j camellos. Para conseguir la victoria por parte de *La banda de Santa* tendrán que atacar con mayor número de renos que de camellos defiendan la fábrica. Los grupos de renos nunca se pueden dividir.

Diseñe algoritmos basados en una estrategia ***Voraz*** que nos devuelva la asignación que **maximiza el número de fábricas** que podrán tomar *Santa* y sus renos, calculando la complejidad del algoritmo desarrollado, bajo las siguientes hipótesis:

a) Los grupos de renos no se pueden unir. Calcule la complejidad.

b) Los grupos de renos pueden unirse para asaltar una fábrica.

56.– A la peluquería "*Va yapelo's*" han llegado a la vez y sin ningún orden N clientes habituales. Como los clientes son conocidos, el dueño de la peluquería sabe el tiempo que va a tardar en atender a cada cliente (t_i), el dinero que debe cobrarle (d_i) y la paciencia (tiempo de espera) que tiene cada persona (p_i).

Diseñe un ***Algoritmo Voraz*** que ayude al peluquero a recaudar la mayor cantidad de dinero posible, aunque se vaya algún cliente sin ser atendido.

Resuelva el problema para el caso particular en el que $t_i = t$, $0 \leq i < N$.

57.– Disponemos de un conjunto de N trabajos. Para cada trabajo i sabemos la fecha límite en que debería ser realizado d_i y la penalización p_i en que se incurre si no se realiza antes. Deben realizarse todos los trabajos. Los trabajos pueden procesarse en cada instante de tiempo (0, 1, 2, etc.), pero sólo uno en cada instante. **El objetivo es realizar todos los trabajos minimizando la penalización total**. Diseñe un ***Algoritmo Voraz*** que resuelva el problema. Calcule la complejidad del algoritmo desarrollado.

58.– En la empresa Losa Preta, O.S. andan escasos de espacio y necesitan distribuir los trabajadores de una forma óptima. Disponen de M puestos de trabajo y N trabajadores. Cada trabajador i tiene unas restricciones horarias r_i, una carga de trabajo c_i y una preferencia p_{ip} por cada puesto p de trabajo. Naturalmente en un puesto no pueden estar trabajando al mismo tiempo dos trabajadores distintos.

Diseñe ***heurísticas voraces*** para determinar el horario de tal forma que todos los trabajadores cubran su carga de trabajo y se **maximice la preferencia del conjunto**, según las dos alternativas siguientes:

a) Un trabajador sólo trabaja en un puesto. Esto es, si se le asigna el puesto *k*, entonces solo trabaja en el puesto *k*.

b) Un trabajador puede trabajar en varios puestos en distintas sesiones, pero no cambia en sesiones consecutivas. Por ejemplo, si de 9:00 a 10:00 trabaja en el puesto *k*, y si trabaja también de 10:00 a 11:00, entonces también lo hará en el puesto *k*.

Naturalmente puede ocurrir que las restricciones sean demasiado fuertes y el problema no tenga solución en los términos planteados. En tal caso, determine **cuál es el mínimo número de puestos de trabajo necesarios para satisfacer las restricciones horarias**, y supuesto que los empleados tienen una preferencia nula por los nuevos puestos añadidos determine la **distribución que maximice la preferencial global**.

NOTAS: La carga de trabajo es el número de horas semanales que tiene que trabajar. Las restricciones horarias se entiende que son las horas a las que puede trabajar durante la semana. La preferencia dada se entiende por unidad de tiempo.

59.– Disponemos de un conjunto de placas rectangulares de tipo *R*, cada una de dimensiones $A \times L$. Se desean recortar *n* piezas rectangulares de dimensiones $a_i \times l_i$, de tal forma que se minimice el número de las placas de tipo *R* de las que se ha obtenido alguna pieza. Diseñe una ***Heurística Voraz*** que resuelva el problema.

60.– Tenemos un procesador que debe realizar una serie de *N* trabajos. De cada trabajo *i* sabemos el instante exacto c_i en que debe comenzar, su duración d_i y el beneficio b_i que proporciona si se realiza. El procesador sólo puede realizar una tarea al mismo tiempo. Supuesto que el procesador está operativo entre los instantes *C* y *F*, diseñe un ***Algoritmo Voraz*** que maximice el beneficio de los trabajos que se procesen. Calcule la complejidad del algoritmo desarrollado. ¿Es óptimo el algoritmo desarrollado? Si no lo es, debe exponer un contraejemplo. En todo caso, debe razonarse la respuesta.

61.– Marcos es el encargado de colocar los cuadros en el Museo Artístico Nacional. El director de este museo tiene la mala costumbre de cambiar los cuadros demasiado a menudo y Marcos está harto de trabajar para colocar y recolocar los cuadros una y otra vez. El museo tiene *S* salas de dimensión $A_j \times L_j$ metros, para cada sala *j*. Cada sala puede tener hasta cuatro puertas, que dan a salas contiguas. Todas las puertas miden *P* metros. Cada cuadro tiene una anchura C_i.

Diseñe un algoritmo que ayude a Marcos a colocar los *N* cuadros de la exposición de manera que se utilice el menor número de salas posible.

Nota 1: No se puede colocar un cuadro encima de otro.

Nota 2: Considere que en el ancho de cada cuadro C_i ya está incluido el espacio de separación entre cuadro y cuadro.

62.– La empresa *Enredando Inc*. desea contar con las últimas tecnologías y ahora se ha embarcado en el proyecto de dotar a todas sus instalaciones con red inalámbrica (WiFi). La responsabilidad de la implantación de la red WiFi ha recaído en su departamento técnico, que ha realizado un estudio al respecto. En el estudio se incluye un mapa de las instalaciones de la empresa en el que se muestra la disposición de los puntos en los que se pueden instalar las antenas, los puntos de la empresa que deben tener cobertura y la distancia [lineal] existente entre cada par de puntos.

Las especificaciones técnicas de las antenas WiFi indican que tienen una cobertura circular de *R* metros de radio y pueden ser configuradas en 3 canales diferentes. Además, advierten que en el caso de que exista un solapamiento en la cobertura de las redes, este solapamiento no debe ser de señales pertenecientes al mismo canal, pues se producirá una interferencia en las señales y la zona de intersección quedará sin cobertura.

Diseñe un ***Algoritmo Voraz*** que encuentre la disposición óptima de las antenas (posición y canal), de tal manera que utilizando el menor número de antenas se proporcione cobertura a todos los puntos especificados, evitando las interferencias. El problema planteado puede no tener solución, exponga las alternativas para conseguir una solución al problema, teniendo presente que el principal objetivo es conseguir la cobertura total.

63.– La empresa de instalaciones eléctricas "*Los Bombillas*", ha resultado la adjudicataria del concurso para la instalación del alumbrado de la ciudad. Han realizado un estudio y disponen de un plano en el que se detallan los puntos que deben iluminarse, y que son los mismos en los que es posible instalar farolas, además tenemos la distancia lineal entre ellos, y se sabe que cada farola tiene un radio de iluminación de *R* metros. Se trata de **iluminar todos los puntos señalados minimizando el número de farolas empleadas**. Diseñe un ***Algoritmo Voraz*** que resuelva el problema.

65.– En el *Mundo de Oz* están pavimentando los caminos que unen las distintas ciudades con baldosas amarillas (todas de igual tamaño). El *Mago de Oz* se ha encontrado con un problema

económico, pues su mundo es más grande de lo que pensaba, y hay muchos caminos que embaldosar. Dado que sus arcas no están rebosantes de dinero, el *Mago* pretende embaldosar los caminos que comunican todas las ciudades del *Mundo de Oz* con la *Ciudad Esmeralda*, la gran capital del imperio, pero **utilizando el menor número de baldosas amarillas**.

Diseñe un ***Algoritmo Voraz*** que dado el mapa de caminos del *Mundo de Oz*, resuelva el problema que tiene el *Mago de Oz*. Calcule la complejidad del algoritmo desarrollado.

66.– En una ciudad el alcalde ha decidido asfaltar sus calles por motivos electorales, pero tiene el problema de que no puede asfaltar todas sus calles por motivos de presupuesto. Ha decidido asfaltar aquellas calles que unan una plaza con otra, quedando todas las calles asfaltadas unidas. Para cada calle y cada plaza sabe cuánto cuesta asfaltarla, y también su tiempo en hacerlo. Además, sabe por estadísticas de elecciones anteriores y encuestas realizadas recientemente, el número de potenciales votantes de su partido tanto para cada calle, como para cada plaza.

El alcalde se ha propuesto maximizar el número de sus partidarios con plaza o calle asfaltadas, puesto que cree que contribuirá a conseguir afianzar el voto entre los seguidores acérrimos, y en el resto, a inclinar definitivamente la balanza a su favor. Diseñe una ***Heurística Voraz*** que indique cuáles son las calles y plazas a asfaltar, supuesto que disponemos de un presupuesto *P*.

67.– *De Campaña*. Tenemos un mapa de carreteras con *N* puntos críticos, que podemos clasificar en *Puntos Negros* y/o *Recaudatorios*. Dado un periodo determinado de tiempo fijo, para cada punto sabemos la recaudación que se obtiene y el número de accidentes que se producen atendiendo a estas tres alternativas: (1) no cubrirlo; (2) cubrirlo con un radar; (3) cubrirlo con una dotación de agentes de tráfico. Disponemos de *R* radares y *P* dotaciones de agentes de tráfico ($P < R$). Se pide resolver los siguientes problemas:

a) Maximizar la recaudación sin que se sobrepase una cantidad *M* de accidentes prefijada de antemano.

b) Minimizar el número de accidentes.

Diseñe ***Algoritmos Voraces*** que resuelvan los problemas planteados.

68.– La Prima Vera ha venido a pasar un tiempo con nosotros, y para no variar, nos ha traído como presente un buen ramo de *N* flores. Las flores son voluminosas, pero de lo más hermosas, y cada una tiene un peso y volumen determinados. Como somos previsores hemos colocado por toda nuestra casa un conjunto de *M* jarrones para colocar la mayor cantidad de flores y complacer lo más posible a nuestra querida prima. Utilizando ***Algoritmos Voraces*** resuelva cada uno de los problemas siguientes:

i) sólo colocamos una flor en cada jarrón,

ii) podemos colocar más de una flor por jarrón.

69.– Juanito se ha comprado una cosechadora para vendimiar. Se aproxima la época de la recolección de la misma y a Juanito le llegan muchos clientes para pedirle que recoja su cultivo ya que éste tiene unos precios muy competitivos. Ante la avalancha de clientes, Juanito tiene que hacer una selección de los mismos ya que le va a ser imposible atender a todos porque la uva tiene una fecha a partir de la cual pierde su valor. De cada parcela se sabe precio que tiene que cobrar, el tiempo que emplea en la recolección y la fecha a partir de la cual no merece la pena recoger el producto porque "caduca" y no se producen beneficios.

Diseñe un ***Algoritmo Voraz*** que ayude a Juanito a maximizar el beneficio puesto que necesita amortizar cuanto antes el coste de la cosechadora.

Resuelva el problema suponiendo que un cliente puede tener varias parcelas, y que todas las parcelas del cliente tienen que recolectarse consecutivamente.

70.– **La granja de Pepito**. Una comarca tiene N_G de parcelas para pastar el ganado, N_C parcelas de cultivo y N_A parcelas de arboleda. Al final de la temporada cada una de ellas aporta un beneficio B_G, B_C y B_A, respectivamente. Así, si un granjero tiene G parcelas de pastos, C parcelas para cultivar y A parcelas de arboleda, el beneficio total al final de la temporada viene dado por:

$$Beneficio = G*B_G + C*B_C + A*B_A$$

Al comienzo de cada temporada, el granjero puede decidir comprar una (y sólo una) nueva zona con sus ahorros, teniendo cada tipo de zona un precio diferente: P_G, P_C y P_A.

Además, tenemos que:

a) cada parcela de ganado nueva supone inutilizar media parcela cultivable para dar de comer a los animales,

b) cada parcela cultivable adicional supone inutilizar media de arboleda, y

c) cada parcela de arboleda supone inutilizar media parcela de ganado para su plantación.

Dado un presupuesto inicial P, unas zonas iniciales de cada tipo G_0, C_0 y A_0, se trata de **maximizar el beneficio que podemos obtener al cabo de N temporadas**. Resuelva el problema mediante un ***algoritmo voraz***.

71.– Tenemos la hora de paso de los autobuses de una línea metropolitana por una parada dada recogidos durante un mes. Los autobuses pasan periódicamente por dicha parada, aunque la hora de paso suele variar por las condiciones del tráfico. Supuesto que se sabe el número de autobuses que pasan por dicha parada, diseñe una ***Heurística Voraz*** que determine el horario de los autobuses en dicha parada, esto es, las horas a las que los autobuses pasan por dicha parada.

72.– Proponga un problema que pueda ser resuelto por un Algoritmo Voraz e impleméntelo.

1.– Programe el Problema del Viajante utilizando funciones con memoria. Se ha de obtener un procedimiento incluido en una unidad al que proporcionándole la matriz de costes del grafo, nos devuelva en la solución la vuelta de mínimo coste y su coste asociado.

2.– Sean *u* y *v* dos cadenas de caracteres. Se desea transformar *u* en *v* con el mínimo número de operaciones elementales del tipo siguiente:

- eliminar un carácter
- añadir un carácter
- cambiar un carácter

Por ejemplo, para pasar de la cadena ababc a bccba podríamos hacer:

$ababc \rightarrow babc$ (eliminar *a*)
$\rightarrow bcabc$ (añadir *c*)
$\rightarrow bccbc$ (cambiar un carácter)
$\rightarrow bccba$ (cambiar un carácter)

Escriba un algoritmo de Programación Dinámica que calcule el número mínimo de operaciones necesarias para transformar *u* en *v* y cuáles son esas operaciones. Impleméntelo.

3.– Sobre el río Tajo hay *N* estafetas de correos. En cada estafeta se puede alquilar un bote que permite ir a cualquier otra estafeta río abajo (es imposible remontar la corriente). La tarifa indica el coste del viaje de *i* a *j* para cualquier punto de partida *i* y cualquier punto de llegada *j* más abajo del río. Puede suceder que un viaje de *i* a *j* sea más caro que una sucesión de viajes más cortos, en cuyo caso se tomaría un primer bote hasta una estafeta *k* y un segundo bote para continuar a partir de *k*. No hay coste adicional por cambiar de bote. Diseñe e implemente un algoritmo eficiente que determine el coste mínimo para ir de *i* a *j*.

4.– **Problema Simplificado de Transporte**.

Se tienen dos almacenes A_1 y A_2, y *n* comercios $c_1, c_2,..., c_n$. En cada almacén hay e_1 y e_2, respectivamente, unidades indivisibles de un producto; y en cada comercio se necesitan v_j ($j= 1, 2,..., n$). Además, se verifica:

$$e_1 + e_2 = \sum_{i=1}^{n} v_i$$

El coste de transporte de A_i a C_j viene dado por la función $Coste_{ij}(x)$, donde *x* es el número de unidades transportadas.

Se pide la solución óptima, i.e., de coste mínimo, para el problema de abastecimiento a los comercios, por un método de ***Programación Dinámica***.

5.– Resuelva el Problema de la Mochila versión 0/1 utilizando ***Backtracking***.

6.– N trabajos han de procesarse en un sistema que cuenta con dos procesadores. Para cada trabajo se saben los tiempos a_i y b_i que necesitaría el proceso en cada procesador. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que encuentre la solución óptima, esto es, la que minimiza el tiempo necesario para concluir todos los procesos.

7.– El **Centro de un grafo** se define como el vértice del grafo que minimiza la distancia media de todos los vértices del grafo a dicho vértice. Diseñe un algoritmo que halle el centro de un grafo. Calcule la complejidad del algoritmo desarrollado.

8.– Construya e implemente un algoritmo que dada una sucesión finita encuentre una de las subsucesiones crecientes más largas (puede haber varias), utilizando bien ***Programación Dinámica*** bien ***Backtracking***.

9.– La partición de un número positivo N consiste en escribir ese número como la suma de k números positivos, por supuesto distintos del original. Por ejemplo:

$$12 = 1 + 1 + 2 + 3 + 5 = 1 + 2 + 4 + 5 = 1 + 2 + 2 + 2 + 5$$

Construya un programa que genere todas las posibles particiones de un número natural dado.

10.– Unos niños juegan en una calle adoquinada a la rayuela. Situados en un adoquín pueden desplazarse al siguiente o saltárselo. Construya un procedimiento que imprima todos los caminos posibles para situados en el primer adoquín ir al n-ésimo. ¿Cuántos caminos hay para alcanzar dicho adoquín?

11.– Problema de **Asignación de Trabajos**.

N trabajos han de ser realizados por N personas, y se sabe que la persona i tardará un tiempo C_{ij}, en hacer el trabajo j.

Resuelva los siguientes problemas utilizando ***Backtracking***:

a) Cada trabajo debe hacerlo una persona distinta y se debe minimizar el tiempo total.

b) Como en el apartado a) pero sin la restricción de que cada persona deba hacer un trabajo.

c) Como en a) pero minimizando el máximo tiempo para concluir los trabajos (todos los trabajadores comienzan simultáneamente).

12.– En un Centro de Salud, un día normal, tenemos P pacientes y M médicos ($M \leq P$). Un paciente sólo será visto por un médico. Tenemos una estimación del tiempo t_{mp} que tarda en atender el médico m al paciente p.

Supuesto que todos los médicos comienzan su consulta a la misma hora, diseñe algoritmos, cuidando la eficiencia, para resolver de los siguientes problemas:

i) **Minimizar el tiempo total de atención a los pacientes**. (No hay ninguna restricción.)

ii) Como en i) pero **cada médico ha de ver por lo menos a un paciente**.

iii) Suponga ahora que los M médicos forman un equipo que se ha desplazado *ex profeso*, utilizando un transporte colectivo, a la localidad del Centro de Salud, y **desean regresar a su centro origen cuanto antes**. **NOTA**: Téngase presente que **no pueden regresar hasta que no se atienda a todos los pacientes**, o lo que es lo mismo, hasta que terminen de atender sus consultas todos los médicos.

iv) Como en el apartado anterior, pero suponiendo además, que entre los M, hay K dados, que sistemáticamente se niegan a ver más de L pacientes cada uno, teniendo que ser asumidas las otras consultas por el resto.

13.– El ministerio de Desinformación y Decencia se ha propuesto hacer trabajar en firme a sus N funcionarios, y se ha sacado de la manga N trabajos. Los funcionarios, ya se sabe que son capaces de realizar cualquier trabajo, pero unos tardan todavía más que otros, y los hay que lo hacen aún pero que otros, lo que se recoge en dos tablas T y E, siendo $T_{f,t}$ el tiempo que tarda el funcionario f en realizar el trabajo t, y $E_{f,t}$ es la eficacia con la que el funcionario f realiza el trabajo t. Su Excelencia el Sr. Ministro, siempre tan preocupado por la eficiencia, desea encontrar la asignación óptima de trabajos a funcionarios en dos sentidos diferentes (e independientes):

a) **minimizar la suma total de tiempos**,

b) **maximizar la suma total de eficacias**,

Diseñe algoritmos ***Backtracking*** que resuelvan dichos problemas.

14.– Un cuadrado latino de orden n es un tablero de $n \times n$ posiciones, cada una de las cuales posee un color escogido entre n colores diferentes, tal que ni en ninguna de sus filas ni en ninguna de sus columnas hay colores repetidos.

Ejemplo (n = 4):

Rojo	Blanco	Negro	Azul
Blanco	Negro	Azul	Rojo
Azul	Rojo	Blanco	Negro
Negro	Azul	Rojo	Blanco

Diseñe e implemente un algoritmo que imprima todos los posibles tableros latinos de tamaño $n \times n$.

15.– Disponemos de una baraja de cartas española con cuatro palos (oros, copas, espadas y bastos). De cada uno de los palos tenemos las siguientes cartas: 1, 2, 3, 4, 5, 6, 7, 10, 11 y 12.

Diseñe e implemente un algoritmo que calcule la manera de obtener "siete y media" de modo que se minimice el número de cartas empleadas.

NOTA: Todas las cartas puntúan por su valor excepto 10, 11 y 12 que sólo valen medio punto.

16.– Sea S un conjunto de enteros positivos, se desea averiguar si existe una partición $\{P,Q\}$ disjunta de S, i.e., $P \cap Q = \varnothing$ y $P \cup Q = S$, tal que

$$\sum_{s \in P} s = \sum_{s \in Q} s$$

17.– Tenemos un conjunto S de N enteros positivos, se trata de hacer una partición disjunta del mismo en dos subconjuntos P y Q de tal forma que se minimice la diferencia de la suma de los elementos de cada subconjunto, i.e., se trata de hallar $\{P,Q\}$ que verifican $P \cap Q = \varnothing$ y $P \cup Q = S$, y tales que minimicen la expresión:

$$\left| \sum_{s \in P} s - \sum_{s \in Q} s \right|$$

Diseñe un algoritmo ***Backtracking*** que resuelva el problema.

18.– Lolo Luke, Director General de CAMP-MUSIC, nos ha encargado la difícil tarea de organizar las grabaciones de su poderosa compañía, de tal manera que exista una compensación de tiempos en sus grabaciones en cinta. Para ello, nos pide que diseñemos un algoritmo que tomando como datos los tiempos de todas las canciones que vayan a ser grabadas sean distribuidas en las dos caras. (De un Ángel).

19.– El presupuesto para comprar libros de una determinada organización es menor que la cantidad necesaria para la adquisición de todos los libros que deseamos comprar. Diseñe un algoritmo que nos indique los libros a comprar teniendo en cuenta lo siguiente:

i) Se desea gastar la mayor parte posible del presupuesto disponible.

ii) Jamás se podrá pagar un importe superior a la cantidad presupuestada.

iii) No hay ningún libro preferido sobre otro.

iv) Se dispone de la lista completa de libros que deseamos con sus correspondientes precios.

20.– Se tiene un tablero rectangular de altura N y anchura K, en cuyas casillas hay letras. Se pide un programa que utilizando Backtracking, encuentre la sucesión de vocales más larga que puede formase partiendo del extremo inferior izquierdo y permitiéndose el acceso desde cada casilla a las adyacentes vertical, horizontalmente o en diagonal, sin pasar dos veces por una misma casilla.

Por otra parte si $K = 2$, existe un método mezcla de "***Método Devorador***" y "***Divide y Vencerás***" que permite resolver el problema. Constrúyalo y prográmelo.

21.– El emperador se dispone a expandir su imperio. Dispone de una lista de países para atacar, cada uno de los cuales tiene una resistencia r_i. Esta resistencia puede ser vencida con un número igual se soldados. Cada país proporciona un beneficio b_i en cuanto a recursos que dispone: oro, granjas, esclavos… Pero con cada país conquistado genera un nivel de descontento d_i en el ejército y si sobrepasa cierto límite el ejercito puede sublevarse. Como todos sabemos el emperador no tiene inteligencia para solucionar el problema así que ha pedido ayuda a su lacayo, que debe diseñar un algoritmo que **maximice el beneficio que se consigue sin sobrepasar el número de soldados y sin que se subleve el ejército**.

22.– Tenemos una colección de n objetos que debemos empaquetar en envases de capacidad C. El objeto i tiene un volumen v_i. Diseñe e implemente un algoritmo que calcule el empaquetamiento óptimo, esto es, que minimice el número de envases a utilizar. Evidentemente, los objetos no se pueden fraccionar. Para simplificar el problema considere que un objeto puede incluirse en un envase si el espacio libre que queda en el mismo es mayor o igual que el tamaño del objeto, ignorando la forma que el objeto puede tener.

23.– Construya un programa que utilice el procedimiento anterior para hacer una copia de seguridad de los ficheros del disco duro de un directorio y sus subdirectorios en el menor número de discos flexibles necesarios.

24.– Tenemos N productos que debemos empaquetar en bolsas. Cada producto i tiene un volumen v_i y un peso p_i. Todas las bolsas tienen la misma capacidad V y la misma resistencia en peso P. Diseñe un algoritmo que nos devuelva el **número mínimo de bolsas necesarias para empaquetar los *N* objetos, junto con la distribución de los mismos en las bolsas**.

25.– *Papa Noel* tiene que repartir los juguetes a los niños y lo hará con una flota de trineos cada uno de los cuales puede soportar una carga máxima *Pmax* (la misma para todos). Tenemos N juguetes, cada uno de los cuales tiene un peso dado. Diseñe un algoritmo ***Backtracking*** que distribuya los N juguetes entre los trineos de tal forma que se **minimice el número de trineos utilizados**.

26.– Disponemos de N de prendas y una lavadora. Cada prenda i tiene un volumen v_i y un peso p_i. El tambor de la lavadora tiene un volumen V y es capaz de soportar un peso P. Diseñe un algoritmo que **minimice el número de lavados** necesarios para lavar las N prendas.

27.– Disponemos de N de prendas y una lavadora. Cada prenda i tiene un volumen v_i y nos aporta un beneficio b_i si la lavamos. El tambor de la lavadora tiene un volumen V. Diseñe algoritmos que resuelvan los problemas siguientes:

a) Se trata de llenar la lavadora de tal forma que se maximice el beneficio de las prendas lavadas, respetando la restricción de volumen.

b) Se trata de minimizar el número de lavados necesarios para lavar todas las prendas, respetando la restricción de volumen.

28.– Tenemos dos bolsas capaces de soportar hasta un peso máximo *Pmax*, y un conjunto de N objetos de cada uno de los cuales conocemos su peso p_i. Se trata de decidir qué objetos se ponen en cada bolsa para que el peso en ambas esté lo más equilibrado posible, sin sobrepasar el peso máximo *Pmax*, sabiendo que el peso de los objetos de cada bolsa debe ser al menos *Pmin* (*Pmin* ≤ *Pmax*). Diseñe un algoritmo que resuelva el problema basado en i) ***Programación Dinámica*** y ii) ***Backtracking***. (Nótese que no necesariamente se deben incluir todos los objetos en las dos bolsas.)

29.– En el año 50 antes de Jesucristo toda la Galia está ocupada por los romanos... ¿Toda? ¡No! Una aldea poblada por irreductibles galos resiste todavía al invasor.

Tras años y años de escaramuzas con *César*, *Abracurcix*, el jefe de la aldea gala, ha decidido invitar a éste a un banquete en la aldea para firmar por fin la paz. Como homenaje a tan ilustre invitado *Abracurcix* ha encargado a sus mejores hombres, *Astérix* y *Obélix*, que construyan enseguida un pórtico de mármol a la entrada de la aldea para recibir a *César*.

Dado que la economía de la aldea no es muy boyante, el jefe le ha encargado a su cuñado que trabaja en la cantera, *Chapuzovic*, que le abastezca de los bloques de mármol necesarios para construir las dos columnas que componen el pórtico. Este último, aún molesto por no haber sido invitado al último banquete celebrado en la aldea, le ha dado los bloques defectuosos que no puede sacar a la venta por su baja calidad.

De este modo, *Astérix* y *Obélix* cuentan con N bloques de mármol de grosores distintos g_i. Tendrán que construir las dos columnas iguales, o si esto no es posible, de forma que haya la mínima diferencia entre ellas, a fin de que no se note mucho. El jefe también les ha informado de la normativa gubernamental vigente a cerca de las columnas de homenaje, que dice que no pueden ser más altas de una determinada altura H_{max}, y que tienen que tener un mínimo de H_{min}.

Como *Panoramix* está de fin de semana, *Algoritmix*, el druida suplente de la aldea, les ha indicado que podrían aplicar una estrategia basada en ***Programación Dinámica*** para resolver su problema. Como no tienen claro a qué se refería el druida con eso de "*Programación*

Dinámica", debe ayudar a nuestros intrépidos galos a salir victoriosos una vez más de tan importante misión.

30.– A los profesores de Algorítmica los cambios de estación les producen extraños efectos, en ellos también ha influido el ajuste presupuestario, por lo que han considerado necesario reducir el gasto de papel en sus exámenes.

Aprovechando tal situación se han propuesto demostrar a sus alumnos la utilidad práctica de la asignatura por lo que han decidido cambiar el tipo de examen que sistemáticamente repetían cada año.

En el examen se proponen N problemas. Cada problema tiene un valor de v_i y ocupa un número de páginas p_i. Se trata de maximizar el valor de los problemas resueltos teniendo en cuenta que sólo pueden rellenarse P páginas. Ahora bien, puesto que consideran que resolver más de una cantidad dada de problemas debe primarse de algún modo, han añadido que si se resuelven M problemas pueden emplearse P' páginas más. Además, sólo tienen en cuenta los problemas completamente resueltos.

Resuelva el problema utilizando ***Programación Dinámica***: Plantee la ecuación recurrente óptima y desarrolle el algoritmo que resuelve el problema.

Aplique el algoritmo desarrollado a la siguiente situación:

Se proponen 5 problemas con puntuaciones 90, 15, 40, 30 y 10 y que ocupan resueltos 90, 15, 45, 35 y 15 unidades de medida, respectivamente. Disponiendo únicamente de 100 unidades de medida, aunque si se realizan tres problemas se pueden utilizar 110 unidades de medida.

31.– **Dentro del Laberinto**. Representamos un laberinto cuadrado con una matriz booleana de N Filas por N Columnas. En el laberinto hay obstáculos. Desde una casilla se puede acceder a las adyacentes dentro de la misma fila o columna, nunca en diagonal, siempre y cuando en la casilla destino, no exista ningún obstáculo.

Partiendo de una casilla (i,j), construya un algoritmo que nos devuelva el **camino mínimo (aquel que tiene el menor número de casillas) que debemos recorrer para alcanzar cualquiera de las salidas del laberinto (puede haber varias)**. La complejidad del algoritmo debe ser polinómica con respecto a N, y debe darse una estimación de la misma.

32.– De una relación de N problemas se van a extraer los P problemas de que consta un examen. Los M alumnos que van a concurrir al examen han entregado resueltos los N problemas y los profesores de la asignatura los han calificado. Cada problema se ha calificado con una nota entre 0 y 10. Si un problema no se ha entregado se ha calificado con 0. Tenemos pues la nota c_{ap} del alumno a en el problema p, para cada alumno a y para cada problema p.

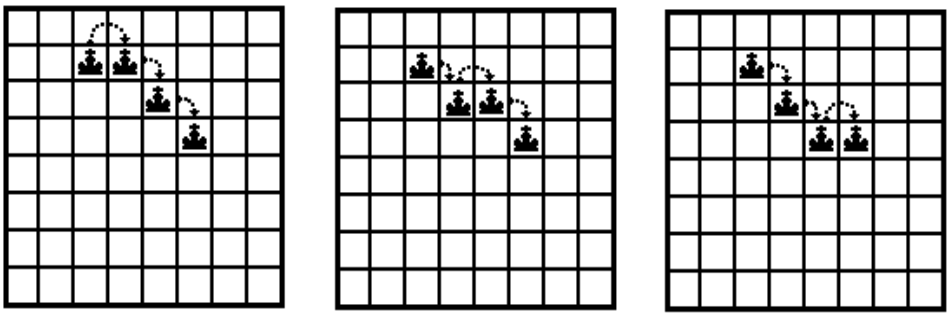
Diseñe un algoritmo basado en "***Programación Dinámica***", que devuelva los P problemas que deben componer el examen para **maximizar el número de aprobados**.

33.– Una sala ha de estar constantemente iluminada por un mínimo de M bombillas. Para ello, disponemos de un conjunto finito de N bombillas $\{b_i\}$ que una vez encendidas no pueden apagarse y que se gastan al cabo de t_i días. Se trata de maximizar el número de días que podemos tener la sala iluminada.

Resuelva el problema usando: a) ***Backtracking*** y b) ***Programación Dinámica***.

34.– Construya un procedimiento que nos devuelva el recorrido mínimo que ha de efectuar el rey negro para atacar todas las casillas del tablero de ajedrez partiendo de la casilla inferior izquierda del tablero.

35.– **Recorridos mínimos de un rey en un tablero de ajedrez**. Sobre un tablero de ajedrez tenemos únicamente el rey negro y deseamos trasladarlo de una casilla *(i,j)* a otra *(m,n)*, visitando el mínimo número de casillas posibles. Generalmente el problema no tiene una única solución, como por ejemplo puede observarse en la siguiente figura que nos muestra las posibilidades que tenemos de pasar de la casilla (2,3) a la casilla (4,6) por secuencias con mínimo número de pasos:



Construya un algoritmo que nos devuelva todas las secuencias mínimas de pasos por las que podemos trasladar el rey desde una casilla dada *(i,j)* a otra dada *(m,n)*. La complejidad de la solución obtenida debe ser O(n), siendo n el número de secuencias mínimas de pasos.

NOTA: Para la obtención de una secuencia mínima partiendo de una casilla dada, hemos de dirigirnos a otra casilla que reduzca la "**distancia**" con la casilla destino.

36.– **Camino mínimo de un caballo para llegar de una casilla a otra del tablero.** Dadas dos casillas del tablero de ajedrez, **Origen** y **Destino**, construya un algoritmo que nos devuelva el camino mínimo que debe recorrer un caballo de ajedrez para que partiendo de la casilla **Origen** llegue a la casilla **Destino**. Si no existe ningún camino entre ambas casillas, el algoritmo debe advertirlo.

37.– **Caminos mínimos de un caballo para llegar de una casilla a otra del tablero.** Igual que el problema anterior, pero deben generarse todos los caminos mínimos entre las dos casillas. La

complejidad del algoritmo que nos genera un camino mínimo debe ser lineal con respecto al número de casillas que componen el camino. ¿La complejidad del algoritmo es lineal con respecto al número de caminos mínimos existentes entre las dos casillas?

38.– Se considera un tablero de ajedrez, un conjunto *C* de casillas del mismo, y dos casillas, *origen* y *destino*. Diseñe un algoritmo que devuelva el **camino mínimo del caballo para ir de la casilla *origen* a la casilla *destino* sin pasar por casillas del conjunto *C*** de casillas dado. NOTA: Se primará la eficiencia.

39.– Diseñe e implemente un algoritmo que devuelva una de las posibles soluciones al problema de "***La vuelta del caballo***", siempre que sea posible, para un tablero de tamaño *n*.

40.– **Recorrido del Caballo de Ajedrez con Condiciones**. Se considera la **lista ordenada** formada por **cuatro** casillas del tablero de ajedrez. Se trata de encontrar el camino que debe seguir el caballo para realizar la "*vuelta*" al tablero de tal forma que la primera casilla del camino sea la primera de la lista, y las restantes casillas de la lista se encuentran en el camino en el mismo orden relativo que en la lista, si bien no necesariamente son contiguas en el camino. Dicho de otra forma, dada la lista ordenada $\{c_0, c_1, c_2, c_3\}$ de casillas del tablero de ajedrez, hay que **determinar el camino que sigue el caballo en su "*vuelta*" al tablero tal que c_0 es su casilla inicial y la lista dada es una subsucesión del mismo**.

Diseñe un algoritmo que resuelva el problema para el caso de un tablero de dimensión *N*.

Considere la resolución del problema para el caso en el que únicamente sean **tres** casillas las que forman la lista considerada, una de las cuales es la casilla de partida.

En ambos casos, si el problema no tiene solución, el algoritmo desarrollado debe indicarlo.

NOTA: El problema de la "*vuelta*" al tablero del caballo de ajedrez consiste en que partiendo de una casilla origen recorrer todas las casillas del tablero visitando cada casilla una única vez y regresar a la casilla origen una vez visitadas todas las casillas del tablero, esto es, la casilla origen es accesible por el caballo desde la última casilla del recorrido.

41.– En Monilandia no conocen los billetes por lo que tan solo usan monedas; de modo que sus habitantes suelen ir encorvados por el peso de sus bolsillos. Por ello, se exige a los tenderos del país que a la hora de devolver el cambio a un cliente, se le entregue el menor número de monedas posible. Resuelva el problema utilizando:

i) ***Programación Dinámica***,

ii) ***Backtracking***.

Ciertamente en nuestro país el problema admite una solución más simple y eficiente. Exponga la condición necesaria para poder aplicar esta segunda política y programe dicha política, demostrando su optimalidad.

42.– En un país se emiten sellos de K valores distintos, y se dispone de sobres en los que caben un máximo de M sellos. Se trata de calcular las tarifas postales que pueden establecerse. Resuelva el problema:

i) Con ***Programación Dinámica*** (y calcule su complejidad).

ii) Con ***Backtracking***.

43.– Suponga ahora que el empleado de correos es un maniático, y le gusta que la tarifa empiece y termine con un sello de la misma clase. Diseñe un algoritmo que nos devuelva todas las tarifas postales que dicho empleado admite.

44.– Desarrolle ahora un algoritmo que calcule todas las tarifas postales que pueden establecerse sin sellos repetidos, de tal forma que se minimice el número de sellos de la misma.

45.– Tenemos dos jarras de capacidades 3 y 4 litros. Construya un algoritmo que devuelva la secuencia de operación para aislar 2 litros.

46.– Dada una sucesión $X = < x_1, x_2, ..., x_m >$, otra sucesión $Z = < z_1, z_2, ..., z_k >$ es una **subsucesión** de X ($k \leq m$) si existe una sucesión estrictamente creciente $< i_1, i_2, ..., i_k >$ de los índices de X tal que para todo $j = 1, 2, ..., k$, tenemos que $x_{i_j} = z_j$.

Por ejemplo, $Z = <B, C, D, B>$ es una subsucesión de $X = <A, B, C, B, D, A, B>$ con la correspondiente secuencia de índices <2, 3, 5, 7>.

Dadas dos sucesiones X e Y, decimos que una sucesión Z es una **subsucesión común** de X e Y, si Z es una subsucesión de ambas. Por ejemplo, si $X = <A, B, C, B, D, A, B>$ *e* $Y = <B, D, C, A, B, A>$, la sucesión $Z = < B, C, A >$ es una subsucesión común de ambas. La sucesión $< B, C, A >$ no es la subsucesión común más larga de X e Y, ya que tiene longitud 3 y la sucesión $< B, C, B, A>$, que es común a X e Y, tiene longitud 4. La sucesión $< B, C, B, A >$ es una de las subsucesiones comunes más largas de X e Y, al igual que la sucesión $<B, D, A, B >$, ya que no hay subsucesiones comunes de longitud 5 o superior.

Utilizando ***Programación Dinámica***, diseñe un algoritmo que dadas dos sucesiones determine la subsucesión de longitud máxima común a ambas.

47.– Se tienen n objetos que se desea ordenar según las relaciones < e =. Por ejemplo, existen 13 ordenaciones diferentes entre 3 objetos:

A = B = C A = B < C A < B = C A < B< C A < C < B
A = C < B B < A = C B < A < C B < C < A B = C < A
C < A = B C < A < B C < B < A

Diseñe un algoritmo de ***Programación Dinámica*** que determine en función de *n* el número de ordenaciones diferentes. El tiempo a emplear ha de ser $O(n^2)$ y el espacio $O(n)$.

48.– Dados *N* enteros distintos. Calcule el número de árboles distintos que se pueden formar con un entero en cada nodo, supuesto que etiquetamos cada nodo con uno de los enteros.

49.– Construya e implemente un algoritmo que decida si dos matrices de adyacencia representan o no el mismo grafo, salvo enumeración de sus vértices.

50.– Una línea de trenes que da servicio a las ciudades comprendidas entre dos, está servida por k tipos de trenes que parten cada hora (los trenes de un mismo tipo) de la estación de origen. Los trenes nunca se adelantan y se conoce el tiempo que tardan en llegar a cada estación, y la lista de estaciones en que se paran. Como no todo tren para en todas las estaciones, resulta que ciertos recorridos exigen trasbordos en estaciones intermedios. Se trata de conocida la hora de llegada a una estación de partida P, encontrar la combinación óptima para llegar a un destino Q. Resuelva el problema:

a) Con ***Programación Dinámica***.

b) Con ***Backtracking***.

51.– Un **camino hamiltoniano** en un grafo no dirigido es un camino que usa cada vértice del grafo exactamente una vez. Construya un algoritmo que decida si un grafo dado tiene un camino hamiltoniano.

52.– Un **camino euleriano** en un grafo no dirigido es un camino que usa cada arco del grafo exactamente una vez. Construya un algoritmo que decida si un grafo dado tiene un camino euleriano y otro que en caso de que exista lo imprima. Hemos fragmentado el problema decidiendo primero si existe dicho camino puesto que se puede aplicar una estrategia devoradora al mismo con una complejidad "pequeña", sobre todo comparada con el que nos devuelve el camino.

53.– **El juego del Mastermind**. Programe el juego del Mastermind, de *N* colores con *M* posiciónes, bajo las hipótesis:

i) Los colores no pueden repetirse, y

ii) Los colores pueden repetirse.

En ambos planteamientos debe tenerse bien presente que la combinación que se propondrá en cada jugada será consistente con los resultados obtenidos en las anteriores.

54.– Dados seis números naturales entre 1 y 100, se trata de encontrar la expresión en la que pueden intervenir, una, ninguna o varias veces, los operadores +, -, * y / (división exacta) y un subconjunto de los seis números, éstos sin repetirse, que evaluada nos devuelva el número más cercano a otro número natural dado menor que 1000. Construya un procedimiento que nos resuelva el problema utilizando:

a) ***Programación Dinámica***,

b) ***Backtracking***.

55.– Tres misioneros y tres caníbales se encuentran en la orilla de un río. Han acordado que a todos les gustaría llegar a la otra orilla. Pero los misioneros no están seguros de qué otras cosas han acordado los caníbales. Por tanto los misioneros quieren arreglar el viaje a través del río de forma tal que el número de misioneros en cada lado del río nunca sea menor que el número de caníbales que están en el mismo lado. La única barca disponible sólo de dos plazas. ¿Cómo podrán atravesar el río sin que los misioneros corran el riesgo de ser comidos?

Construya un procedimiento que nos resuelva el problema para N misioneros y N caníbales.

57.– Paseábamos tranquilamente por la EXPO'92, un florido día de mayo perdiéndonos entre los demás visitantes, hasta que nos encontramos a un francés gritando: *Vive la France!!!*. Como quiera que no paraba y que como transcurría el tiempo se excitaba más, dándoselas también de listo, y dándole, no al Burdeos, sino al tinto de Valdepeñas, aprovechamos el encontrarnos en una terraza de baldosas de colores:

Blanco, Negro, Rojo, Amarillo, Verde y Azul,

y puesto que la bandera francesa es de color **Azul, Blanco y Rojo**, en esta secuencia, le propusimos que se situara en cualquiera de las casillas azules de los lados del embaldosado y permitiendo el acceso desde cada baldosa a las adyacentes vertical, horizontal o en diagonal, sin pisar dos veces la misma baldosa, contruyése la secuencia que nos da la cantidad máxima de banderas francesas consecutivas.

58.– **La huerta del tío Joaquín**.

El tío Joaquín tiene una huerta rectangular de 100 m de ancho dividida en zonas de longitud múltiplo de 100 m. Cada zona tienen una calidad (de 1ª a 5ª). Puede sembrar N productos de los cuáles se conoce el rendimiento por Ha, según la calidad del terreno. En cada temporada se hacen S siembras que posteriormente se recogen, y por otra parte, llueve ll veces, hiela h y hace buen tiempo bt veces. Se tiene que:

i) Al helar se pierde la mitad del rendimiento de lo sembrado, y el terreno no sembrado pierde un grado de calidad.

ii) Al llover crece un 20% del rendimiento, y al hacer sol un 15%.

iii) Tres temporadas de sol o lluvia consecutivas arruinan la cosecha.

iv) Dos temporadas de lluvia sin sol en medio sobre un terreno no sembrado mejoran la calidad un grado.

Se pide:

A) Implemente la huerta y las operaciones que modifican su estado.

B) Buscar el óptimo funcionamiento (alternativa de siembras, cosechas y sucesos atmosféricos) durante dos temporadas. Cada siembra puede ser de 1 a H hectáreas.

NOTA: La parte A) del problema se refiere a Estructuras de Datos, y la resolución de B) a los esquemas algorítmicos.

59.– El dueño de un Salón de Banquetes planea las celebraciones de toda la semana. Su mayor problema es disponer de suficientes manteles limpios. El buen señor sabe de antemano los banquetes concertados para dicha semana, y por tanto, ha podido calcular los manteles necesarios para cada día (la función f, f(i) es el número de manteles necesarios el día i-ésimo, i= 1, 2,..., 7). Además, se ha informado de los precios y velocidad en la entrega de las dos lavanderías más cercanas:

L_1 - x pts. por mantel, listos al día siguiente,

L_2 - y pts. por mantel, limpios en dos días, (un día más que L_1 pero con $y < x$).

Por otra parte, los manteles nuevos cuestan z pts. ($x, y < z$).

Ayude al señor a decidir:

A) La cantidad S de manteles nuevos que debe comprar al principio (se supone que el negocio es nuevo y todavía no dispone de manteles; además, la compra se hace el primer día y no se vuelven a comprar más).

B) El número de manteles que cada día de la semana se mandan a la lavandería (cada día se mandan a lavar todos los manteles ensuciados).

Por supuesto estas decisiones deben minimizar el gasto total en toda la semana.

60.– "*El nombre de la rosa*" o lo que no sabía Guillermo de Baskerville cuando visitó el laberinto.

Guillermo recordaba vagamente un procedimiento para recorrer el laberinto, pero no le daba gran fiabilidad ya que "probablemente no funciona en algunos casos". El algoritmo era el siguiente:

- Partimos de la escalera de la que parte un único pasillo.
- Al llegar a una habitación por primera vez, marcamos el pasillo por el que accedemos a élla.
- Al salir de una habitación inspeccionamos primero todas sus salidas, buscando si alguna nos lleva a una habitación nueva. Si encontramos una tal, pasamos a élla, y si no es así, retrocedemos por el camino de acceso marcado.
- Una vez en la escalera habremos recorrido el laberinto.

Se pide:

a) Un procedimiento que recorra el laberinto y nos describa el mismo numerando las habitaciones la primera vez que se accede a ellas, y cite tal enumeración cuando las volvamos a cruzar. Para distinguir los pasillos supondremos que poseemos una perfecta orientación numerando los pasillos en una habitación según se recorran, en el orden indicado por el sentido de las agujas del reloj. Los pasillos se cuentan tantas veces como se recorren.

Pero Guillermo deseaba explorar muchas veces el laberinto, y, para ello, lo mejor sería que averiguase primero el recorrido óptimo: minimizando el número de pasillos atravesados.

b) Encuentre tal recorrido con un algoritmo de Backtracking.

c) Programe dicho algoritmo utilizando Backtracking.

NOTA: El laberinto es desconocido y, por tanto, aunque puede representarse en el ordenador como se desee, debe tenerse cuidado de no utilizar dicha información salvo en la medida en que nuestro recorrido nos lo suministra.

EJEMPLO de INCORRECTO uso de la implementación:

Laberinto es la matriz de adyacencia del grafo de pasillos, y se usa su tamaño para saber cuando se ha visitado todo el laberinto; o equivalentemente, se utiliza información asociada a habitaciones no visitadas.

61.– Cuenta la leyenda que en la Cámara Funeraria de la pirámide del faraón Ammenotitis XIV se sepultaron gran parte de las riquezas que logró reunir en su corta vida, un tesoro de casi incalculable valor. Si bien se pensó durante largo tiempo que la leyenda era una fábula y no tenía realidad, una expedición española al frente de la cual figura el famoso arqueólogo don Luis de Rigobares Langredo ha logrado descubrir el emplazamiento exacto de la pirámide, oculta a los ojos del mundo por la arena del desierto.

También ha descubierto que tiene **varias entradas**, y por las inscripciones (maldiciones) que rezan en las entradas de la gran pirámide ha deducido **que el interior es un laberinto de cámaras**, alguna de las cuáles son **pozos sin fondo**, comunicadas entre éllas por **galerías** entre las que hay **algunas que están obstruidas por escombros** caídos de las paredes fruto de un terremoto o no se sabe bien que fuerzas extrañas. Ha comprobado que **en atravesar una galería sin escombros tarda *T* minutos**, en tanto que **para una galería de la que ha de quitar los escombros tarda *M* veces más**.

Don Luís de Rigobares es un arqueólogo muy metódico y en sus exploraciones siempre que puede aplica un algoritmo ***Backtracking*** **(Vuelta Atrás)** apropiado.

a) Construya un algoritmo que simule el recorrido del gran arqueólogo Rigobares en busca de la cámara funeraria del faraón, y nos diga cuál es la secuencia de cámaras que hemos de visitar sin recorridos inútiles hasta alcanzar el tesoro y el tiempo total que hemos tardado en encontrarla.

   NOTAS:

   En un **pozo** pueden desembocar varias galerías, ahora bien no podremos tomar ninguna de éllas porque nos es imposible atravesar el **pozo**.

   Adviértase también que los escombros de una galería sólo se quitan la primera vez que se atraviesa la galería.

   Algo que habrá que tener bien presente es que el laberinto se ha de representar de alguna forma en el ordenador pero que dicha información no podrá ser utilizada más que según la vayamos obteniendo al recorrer el laberinto.

b) Suponga ahora que el malvado Malandrín, jefe de los ladrones de tumbas, se ha confabulado con sus secuaces para robar el sarcófago del faraón pero ha de hacerlo en el mínimo tiempo posible no para evitar la vigilancia de los guardianes sino porque una de los descubrimientos del insigne don Luís es el descifrar una inscripción en la cámara que advierte que tras retirar el sarcófago del lugar destinado al reposo del faraón la pirámide se derrumbaría atrapando a los usurpadores de la paz de Ammenotitis en pocos minutos.

Nótese que lo que se ha de minimizar es el tiempo que se tarda en salir al exterior partiendo de la cámara funeraria, y que hay varias posibles salidas (las entradas descubiertas por el gran Rigobares).

62.– Construya un algoritmo capaz de transformar un entero inicial cualquiera N en otro entero final M, con el menor número de transformaciones de la clase:

*a) f(i) = 3 * i*

b) *g(i) = Parte entera de i/2*.

Por ejemplo, para transformar 15 en 5 podemos hacer:

$$15 \to 7 \to 21 \to 10 \to 5\,, \quad \text{haciendo} \quad ggfg.$$

Si la transformación es imposible el algoritmo debe detectarlo sin ciclar indefinidamente.

63.– Se considera el problema de Programación Lineal Entera: por medio de un algoritmo de ***Backtracking*** **(Vuelta Atrás)**.

$$max \sum_{i=1}^{n} b_i x_i$$

sujeto a:

$$\sum_{i=1}^{n} c_i x_i \le d$$

$$x_i \ge 0,\ x_i \ entero \ \text{i} = 1,\ \ldots, \text{n},$$

siendo b_i, $c_i > 0$.

Diseñe un algoritmo lo resuelva basado en:

a) ***Programación Dinámica***, o

b) ***Backtracking***.

Aplíquelo al problema siguiente:

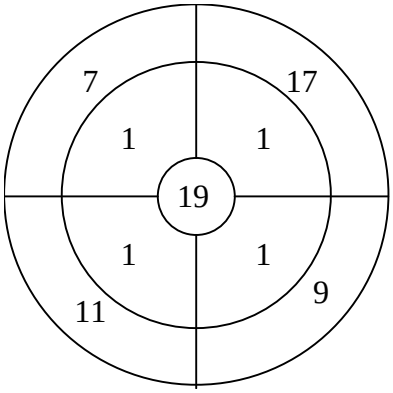
$$max\ 4x + 3y + 2z$$

sujeto a:

$$3x + 4y + 2z \le 8$$

$$x, y, z \ge 0,\ \text{enteros}$$

64.– Dada la siguiente diana:



Diseñe un algoritmo que devuelva cómo sumar exactamente 100 puntos con el menor número de dardos. Resuelva el problema utilizando: i) ***Programación Dinámica*** y ii) ***Backtracking***.

65.– Se considera una tabla de rendimientos R de dimensiones $M \times N$, donde R_{ei} indica el rendimiento obtenido al invertir en la entidad e una cantidad i. Diseñe un algoritmo basado en ***Programación Dinámica*** que maximice el rendimiento al invertir una cantidad C, entre las M entidades financieras de tal forma que se maximice el beneficio obtenido. NOTA: la inversión en una entidad se considerará global, esto es, no se puede invertir 2 y 3, en una misma entidad, puesto que a los efectos consideraremos el valor de 5 como inversión.

Aplíquese a la siguiente tabla de rentabilidades, suponiendo que dispongamos de 4 millones.

	Valor de la Inversión en Millones				
	0	**1**	**2**	**3**	**4**
Entidad 1	0	2	5	6	7
Entidad 2	0	1	3	6	7
Entidad 3	0	1	4	5	8
Entidad 4	0	0	2	5	9

66.– Tenemos un tablero de $M \times N$ casillas, en cada una de las cuales hay una cantidad de monedas, no necesariamente la misma cantidad en cada casilla, y además algunas casillas están vacías. Situados en una casilla, sólo podemos desplazarnos a una de las casillas adyacentes tanto en diagonal, horizontal como verticalmente, que tenga monedas, y cuando lo hacemos tomamos todas las monedas que hay en la misma.

Construya un algoritmo ***Backtracking***, que devuelva el **recorrido mínimo** (el de menor longitud) necesario para reunir un número dado D de monedas, partiendo de una casilla dada.

67.– Tenemos N asignaturas optativas en la carrera, cada una con K_i créditos y un cierto horario H_i. Se trata de mostrar todas las posibles elecciones de asignaturas que podemos hacer para conseguir T créditos exactamente, de forma que dos asignaturas nunca coincidan en el horario y no se sobrepasen un número máximo M de asignaturas ($M < N$).

Resuelva el problema anterior suponiendo además que horario debe tener el menor número de asignaturas. En el caso de que haya varios, bastará con que muestre el primero

68.– **Partida de Dominó**.

Dado un reparto de fichas de dominó, construya una de las partidas más cortas en las que el jugador que sale gane, si es que existe alguna para dicho reparto.

69.– El juego del dominó tiene 28 fichas distintas. Dada una lista de las mismas halle:

a) Todas las posibles **cadenas correctas** que se pueden formar con fichas de tal lista.

b) Una de las **cadenas correctas** más largas (puede haber varias) que puede formarse con fichas de la lista.

NOTA: Una **cadena es correcta** si los extremos adyacentes de dos fichas consecutivas son del mismo número.

70.– **Cubrimiento de Conjuntos**. Sea $S = \{S_i\}_{i \in I}$, una colección finita de conjuntos finitos. Obténgase un recubrimiento mínimo de la misma.

Un **recubrimiento** de S es subcolección de la misma $S' = \{S'_j\}_{j \in J}$, (i.e. $\forall j \in J \; \exists \; i \in I,, \; S'_j = S_i$), tal que todo conjunto de S esté contenido en la unión de los conjuntos de S'.

Un **recubrimiento** S' de S se dice **mínimo** si el cardinal de S' es mínimo entre todos los recubrimientos de S, es decir, el número de conjuntos de la subcolección S' es mínimo.

Por ejemplo: Sea S = {{1,3,4}, {2,4,6}, {2,4}, {1,3}, {5,7}, {4,6}}. Recubrimientos del mismo son:

S' = {{1,3,4}, {2,4}, {5,7}, {4,6}},

S'' = {{1,3}, {2,4,6}, {5,7}}

S''' = {{1,3}, {2,4}, {5,7}, {4,6}}

Resuelva el problema mediante i) ***Programación Dinámica*** y ii) ***Backtracking***.

71.– Para *Forrest Gump* la vida era como una caja de bombones. Cada caja tenía bombones de varias clases. *Forrest* frecuentaba una tienda que tenía muchas cajas distintas, y *Forrest* siempre deseó tener bombones de todas las clases, pero su paga no le alcanzaba. Por ello, tenía que comprar **el menor número de cajas** pues todas las cajas tenían el mismo precio. Resuelva el problema utilizando una estrategia de ***Programación Dinámica*** o ***Backtracking***.

72.– Se define el **Peso de un camino** como la suma de los pesos de sus vértices.

Construya un algoritmo de ***Programación Dinámica***, en el que dados un grafo *G* dirigido y con pesos en los vértices y un vértice *v* del mismo devuelva el camino, sin ciclos, de mayor peso de entre todos caminos con origen en *v*.

Nótese que un camino sin ciclos es aquel en el que no se repiten los vértices.

74.– En un campo tenemos un número determinado de flores, cada una con una determinada cantidad de polen. Tenemos un zángano que guiado por la ley del mínimo esfuerzo, sólo vuela a las flores adyacentes. Supuesto que debe saciar su apetito con una cantidad de polen *P*, visitando el menor número de flores, diseñe un algoritmo basado en la Programación dinámica que le ayude a conseguir su objetivo. Supóngase que parte de una flor dada.

¿Y si el zángano puede elegir la primera flor? Si quitamos la restricción de la elección a las flores adyacentes, ¿cuál sería la solución?

Calcule la complejidad del algoritmo desarrollado en cada uno de los casos.

75.– En un país netamente vitivinícola, cada uno de los *N* pueblos principales se ha especializado en la crianza de cada uno de los *T* tipos de vinos propios del país ($T \leq N$). Un extranjero por diversas circunstancias ha sido conminado por las autoridades a que abandone el país en el

menor tiempo posible. Este sujeto desea llevarse P tipos distintos de vino dados ($P \leq T$), que son los que le gustan especialmente. Dispone de un mapa del país con la distancia entre pueblos y el tiempo necesario para ir de uno a otro.

Sabiendo que se encuentra en una ciudad dada, construya un algoritmo que indique el tiempo mínimo que necesita para recoger los P tipos de vino y abandonar el país, y devuelva también el camino que seguiría. Resuelva el problema utilizando: i) ***Programación Dinámica*** y ii) ***Backtracking***.

Modifique el algoritmo si basta con llevarse P tipos cualesquiera de vino. ($P \leq T$)

Suponga ahora que el ciudadano extranjero desea abandonar el país cuanto antes, luego no se detendrá a comprar vino, pero tiene que decidir el mismo cuál es el camino que debe seguir, y consecuentemente tiene que hacerlo en el menor tiempo posible. ¿Cuál es el algoritmo que habrá empleado para decidir el camino a seguir para abandonar el país y cuál es el itinerario que seguirá? Calcule la complejidad del algoritmo.

76.– Un sujeto ha tenido que hacer una escala en el aeropuerto de su ciudad natal y dispone de H horas hasta que salga su avión. Ha pensado en visitar a la **mayor cantidad posible** de parientes y amigos en sus casas de la ciudad. Para ello, conoce la **distancia en tiempo** que hay entre dos cualesquiera de las casas a visitar y el tiempo que debe estar en cada casa para que se considere una visita. También sabe el tiempo para ir desde el aeropuerto hasta cada casa y el necesario para ir desde cada casa al aeropuerto. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que devuelva el recorrido que debe hacer para conseguir su objetivo.

77.– "*Epe*", "*El Programador Enmascarado*", es el nuevo héroe de los videojuegos. Ha llegado de incógnito a la ciudad y se ha alojado en un hotel. Tiene que "*neutralizar*" a M cualesquiera de los N "*malvados*" que planean un complot para sembrar el caos en la ciudad, con ello desbarataría el complot. Lo hará al abrigo de las sombras de la noche, mientras los "*malvados*" duermen en sus casas, y regresando a su hotel (para no levantar sospechas), en el **menor tiempo posible** Sabe la distancia en tiempo entre cada casa, el tiempo que separa cada casa de su hotel y el tiempo que tarda en "*neutralizar*" a cada *malvado*. Diseñe un algoritmo que resuelva el problema utilizando una estrategia basada en ***Programación Dinámica*** o ***Backtracking***. **NOTA**: Considérese que el tiempo cuenta desde que sale de su hotel y termina cuando regresa al mismo.

78.– Diseñe un algoritmo que resuelva el **Juego del Sudoku**.

79.– "*El Luis*" es un *vampiro* de la leche, porque forzado por las circunstancias –proximidad de los exámenes– se ha visto obligado a cambiar su dieta habitual, por dos litros de leche diarios, aunque el fin de semana vuelve a sus costumbres, e incluso también se permite algún que otro *lingotazo*.

La tensión ante los exámenes le ha llevado a plantearse, seriamente, cambiar sus estudios de Informática por maquinista de trenes, pero ya que le quedan pocas asignaturas piensa obtener el título (aún contando con la dificultad añadida por ciertos profesores en los exámenes de sus asignaturas).

En las últimas fechas la tensión se ha dejado sentir más, porque también necesita graduarse como *vampiro*, en otras palabras necesita víctimas a las que chupar la sangre. Y con la llegada del buen tiempo se le despierta el apetito: empieza a odiar la leche; y cada vez que ve algo rojo (un simple bolígrafo o rotulador) piensa en devorarlo –no se sabe bien si al objeto o al portador del mismo–.

Así pues, se ha planteado "matar varios pájaros de un tiro": obtener el título, graduarse como vampiro, y dar rienda suelta a sus instintos, y todo esto, aplicando los conocimientos aprendidos en sus estudios de la carrera.

Se ha provisto de un mapa de las casas de la ciudad, ha medido en tiempo la distancia entre las mismas; sabe también el tiempo que tarda en "chuparle la sangre" a cada una de sus posibles víctimas; y sobretodo, tiene bien presente que ciertos *"sujetos"* deben **necesariamente** encontrarse entre sus víctimas.

Aunque tiene un problema: *la Unidad Antivampiros de la policía local*. Dicha unidad es capaz de localizar instantáneamente a todos los vampiros que no se encuentran en lugar seguro una vez que se ha disparado la alarma y darles caza. Así, "*El Luis*" debe estar en lugar seguro (su casa) antes de que se active la alarma.

Estudiando los métodos de actuación de la *Unidad Antivampiros* ha llegado a deducir que la alarma se dispara transcurridas *T* unidades de tiempo tras cometerse el primer crimen de la noche.

Diseñe la estrategia que empleará "*El Luis"* para, permaneciendo a salvo de la *Unidad Antivampiros*, lograr **maximizar el número de víctimas**, entre las que se encontrarán necesariamente los sujetos a los que les tiene especial "cariño" (un conjunto prefijado de las víctimas), con los siguientes supuestos:

i) Una única víctima por casa.

ii) *"El Luis"* chupa la sangre a todos los moradores de la casa, teniendo presente que lo que ahora tiene "*El Luis"* es el tiempo total que tarda en cometer su crimen en la casa, y no individualizado por víctimas.

Resuelva el problema por ***Programación Dinámica*** y por ***Backtracking***.

80.– En la ciudad ha habido una invasión de *vampiros*, motivo por el cuál *La Autoridad* ha acordado permitir que en el jardín de cada casa (e incluso en los edificios de la Universidad) se puedan plantar ajos para espantar a los "*malignos*".

Así, en la reciente construcción de una nueva urbanización, financiada por el ayuntamiento, se ha tenido en cuenta plantar en la parcela de cada casa ajos que protegerán a sus habitantes del ataque de los vampiros. Aunque tal medida tiene como contrapartida un determinado olor a ajo en cada casa.

Las casas que cuentan con financiación del ayuntamiento deben ser asignadas a las familias que las han solicitado. Para evitar quejas (e incluso abandono de la casas de la familia a la que se ha asignado) **se debe cumplir que el olor a ajo de la casa debe ser soportado por la familia que la habite**. Para cada familia tenemos el grado de tolerancia al olor a ajo.

Supuesto que conocemos el número de componentes de la unidad familiar (los animales de compañía quedan excluidos –entre éstos no se encuentra la suegra–), pretendemos **maximizar el número de personas protegidas** (cada casa es ocupada por una única familia), teniendo siempre presente que la familia que habite una casa debe soportar el olor a ajos de la misma. Desarrolle algoritmos que determinen la asignación utilizando estrategias basadas en ***Programación Dinámica*** y ***Backtracking***.

81.– Tenemos un conjunto de n componentes electrónicas para colocar en n posiciones sobre una placa. Se nos dan dos matrices cuadradas C y D, de orden n, donde C_{ij} indica el número de conexiones necesarias entre la componente i y la componente j, y D_{pq} indica la distancia sobre la placa entre la posición p y la posición q (ambas matrices son simétricas y con diagonales nulas). Un cableado $(x_1,\ldots,x_n)$ de la placa consiste en la colocación de cada componente i en una posición x_i. La longitud total de este cableado viene dado por la fórmula:

$$\sum_{i<j} C_{ij} D_{x_i x_j}$$

Diseñe un algoritmo ***Backtracking*** que devuelva el **cableado de longitud mínima**.

88.– Tenemos un rompecabezas de colores cuyas fichas son cuadrados de *1* cm (*UN* cm) de lado, con un color central y las aristas pintadas de colores. Se trata de encajarlas en un tablero cuadrado de *M* cm de lado, teniendo en cuenta que dos aristas en contacto han de tener el mismo color. Se supone que tenemos una cantidad fija *N* de fichas. Se pide:

a) Construya un algoritmo que decida si puede resolverse el rompecabezas.

b) Resuelva el rompecabezas utilizando el mínimo número de fichas rojas.

89.– María Cruci Grama es una gran aficionada a los crucigramas, y a la hora de reformar su salón se le ha ocurrido una forma original de embaldosarlo. Ha comprado *N* baldosas con distintas letras pintadas en ellas (una letra en cada baldosa) y pretende que su salón tenga **la mayor cantidad de palabras formadas en horizontal (de izquierda a derecha) o vertical (de arriba abajo)**. El salón de María tiene una superficie de *PxQ* baldosas.

Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que le ayude a embaldosar su salón obteniendo la **mayor cantidad de palabras**.

Notas:

a) María dispone de un diccionario electrónico que mediante la llamada a la función **Boolean EsPalabra(String palabra)** puede saber si una combinación de letras forman una palabra.

b) Se asume que una misma baldosa puede formar parte de varias palabras, incluso en la misma dirección.

90.– Dada una lista de *P* palabras, se trata de colocarlas en forma de crucigrama en una cuadrícula de *M* x *N* en la que hay *C* casillas negras (utilizadas para separar las palabras). Diseñe un algoritmo ***Backtracking*** que realice esta colocación o indique si esto no es posible. Resuelva el problema bajo los siguientes supuestos:

a) las casillas negras ya se encuentran fijadas,

b) el algoritmo debe poner él las casillas negras.

92.– ¡¡Una horda extraterrestre ha atacado la tierra indiscriminadamente bombardeando aquí y allá todas las ciudades que han encontrado a su paso!!

Por suerte el "*ELE*", *Ejército de Liberación Europeo*, ha conseguido derribar *N* naves cayendo estas en *N* sitios distintos del globo terrestre.

Por suerte para nosotros, Mulder y Scully estaban por ahí realizando unas escuchas, cuando interceptaron un mensaje intergaláctico que avisaba de la invasión definitiva de la tierra en *D* días.

En este momento decidieron ir a M cualesquiera de los N sitios ($M < N$) donde habían caído las naves enemigas para analizar los restos de estos seres y poder fabricar un arma biológica que terminase definitivamente con ellos.

Diseñe un algoritmo basado en ***Backtracking*** o ***Programación Dinámica*** para ayudar a Mulder y Scully a visitar los M sitios necesarios, en el menor tiempo posible y siempre antes de que lleguen los extraterrestres, para recoger las muestras y preparar las armas biológicas.

Todo debe estar apunto para cuando lleguen "*los hostiles*".

93.– Tenemos un conjunto de N hombres y N mujeres. Para cada uno de estos conjuntos tenemos sendas matrices cuadradas de $N \times N$ elementos, H y M, donde M_{mh} indica la preferencia de la mujer m sobre el hombre h, y análogamente con H para los hombres. Construya un algoritmo que encuentre el emparejamiento de hombres y mujeres que maximice la suma de los productos de las preferencias.

94.– El encargado de una red de concesionarios ha decidido rebajar el precio de una serie de N coches que no han sido vendidos y que se han pasado de moda. Los coches son voluminosos, con una longitud y un peso determinados. Para la distribución de los coches se utilizan M *trailers*. Cada trailer tiene una longitud dada, y puede soportar un peso determinado (no tienen porqué ser los mismos para todos). Se deben repartir la mayor cantidad posible de coches. Diseñe una estrategia basada en **Backtracking** o **Programación Dinámica** para resolver el problema.

95.– Disponemos de n cajas de longitud C_i y anchura M cada una. Se pretende empaquetar K objetos, de longitud O_j ($O_j > M$) y anchura M cada uno, utilizando el mínimo número de cajas. Diseñe un algoritmo ***Backtracking***, que nos de la mejor disposición de los objetos en las cajas para conseguir el objetivo de utilizar el menor número de cajas. ¿Es óptimo? Calcule su complejidad.

96.– Tenemos N libros y K estantes de una estantería. Cada libro tiene un ancho A_i y peso P_i distinto y cada estante puede soportar un peso máximo PM_j. Todos los estantes tienen la misma longitud L. Diseñe un algoritmo utilizando ***Backtracking*** o ***Programación Dinámica*** que encuentre la disposición de los libros en las estanterías de manera que podamos colocar el mayor número de libros.

97.– **Viernes13.com**. Jason había decidido cambiar sus hábitos. Últimamente le habían estado comiendo el tarro con las bondades de la aplicación de la informática a problemas de optimización, y él que siempre estaba pensando en lo mismo, pensaba ir ahora por los cibercafés y otros garitos.com.

Para ello, se había provisto de un plano en el que se detallaba la distancia (en tiempo) t_{ij} entre los garitos.com, y también tenía una estimación del número de posibles víctimas de cada garito.com para esta noche v_i, porque Jasón sólo tenía esta noche para cometer sus fechorías. Y también sabe el tiempo que tardará en "liquidar" a todas las víctimas de cada garito t_i (distinto para cada garito).

Diseñe un algoritmo de ***Programación Dinámica*** o ***Backtracking*** que nos devuelva la secuencia de garitos.com que visitará Jasón, para "cazar" la mayor cantidad de víctimas a lo largo de toda la noche, suponiendo que puede empezar por uno cualquiera de los garitos.com.

98.– **Un repartidor de pizzas tiene que distribuir *N* encargos,** que le han asignado previamente, y tratará de hacerlo **en el menor tiempo posible,** porque desea "pegarle a la hebra" con la vecina del quinto. Para ello, se ha provisto de un mapa de la ciudad, en el que ha señalado las casas a las que tiene que llevar encargos (un encargo por casa), y en el que ha anotado la distancia, en tiempo, entre casas a las que tiene que llevar encargos, y también la distancia (en tiempo) de la central de reparto hasta cada casa. Supuesto que en cada viaje puede llevar como mucho *M* pizzas, diseñe un algoritmo que resuelva el problema utilizando una estrategia basada en ***Programación Dinámica*** o ***Backtracking***.

99.– *El tío Eusebio* quiere rememorar sus tiempos mozos, y se ha propuesto hacer un bombo para mostrarle a sus nietos la técnica de construcción de los mismos. Además, lo hace con otro propósito que más adelante se verá.

Cuando era joven no tenía miedo a las inclemencias del tiempo y era capaz de seguir en plena faena cuando la más negras nubes amenazaban tormenta en pleno mes de siega, o ensarmentando aunque estuvieran cayendo chuzos de punta. Pero los años no perdonan, y ahora tiene bastantes achaques, aunque conserva su buen sentido para predecir el tiempo, el mismo que usa para dar los consejos a los jóvenes, que las más de las veces ignoran.

Tiene varias hazas y desea que el bombo le sirva para resguardarse de las inclemencias del tiempo, principalmente agua, porque nunca se le ocurriría meterse en el bombo cuando viniera una tormenta con aparato eléctrico, salvo que el bombo fuera "último modelo" dotado de pararrayos –había perdido a alguno de sus mejores amigos que no habían seguido el consejo–.

Principalmente tiene viñas, aunque también tiene alguna que otra siembra, y con la jubilación, tiene tiempo para visitarlas regularmente –casi a diario–, por lo que sabe la distancia en tiempo, entre ellas. Lo mide en tiempo porque se ha tenido que acostumbrar a las horas para tratar con los jóvenes, esos que se desesperan cuando hablan de "estar un rato", pero a él nunca lo han tenido que esperar como él tiene que hacer con ellos, y eso que se pasan la vida mirando el reloj pero nunca llegan a tiempo...

Se trata de **hacer el bombo en el haza a la que se tarde lo menos posible en llegar a él desde cualquiera de las otras hazas**. El algoritmo desarrollado debe tener una **complejidad polinomial**, con respecto al número de hazas.

100.–Enterado *el hermano Paco* de las intenciones de su amigo *Eusebio*, quinto suyo, y acompañante de sus caminatas por el campo, le ha propuesto que en vez de uno hagan dos: "Porque no sólo vamos a tus tierras, sino también, y las más de las veces, a las mías, y así, con dos, mejor nos resguardaremos. O no te acuerdas del refriao que pillaste esta primavera, en *Las Chimeneas*, cuando fuimos a ver el centeno". –Le ha dicho con el acento que le caracteriza.

La idea que les guía sigue siendo hacer los dos bombos en aquellas hazas que les permitan llegar a uno de ellos en el menor tiempo posible desde cualquiera de las otras hazas.

Ellos tenían ya elegido las hazas donde hacerlos, pero uno de sus nietos que estudia Informática, y que al principio creía que eso de los bombos eran como tiendas de campaña que se compran en las tiendas de aventura, les ha dicho que él les dirá el lugar donde hacer los bombos. Y han accedido. Pero se temen lo peor porque todavía se están riendo de cuando su nieto les preguntó que ¿cuánto cemento necesitarían para hacer un bombo?

101.–K atracadores consiguen robar n_1 billetes del tipo *1*, n_2 billetes del tipo *2*, ..., n_p billetes del tipo p. Quieren repartir el botín de forma que todos se lleven exactamente la misma cantidad de dinero, donando el resto a la asociación *ALELADOS* (Asociación de Ladrones Escasamente Listos y Atrapados). Naturalmente, se trata de donar a dicha asociación la menor cantidad de dinero posible. Construya algoritmos que resuelvan el problema según: i) ***Programación Dinámica*** y ii) ***Backtracking***.

102.–Una empresa constructora de barcos fabrica en sus N astilleros M modelos de barcos, construyendo mensualmente b_{ak} barcos en cada astillero a de cada modelo k. La fabricación de estos barcos es de M_a millones por astillero y por mes.

Calcule como repartir la fabricación en los astilleros para reducir al mínimo el coste teniendo en cuenta que se debe hacer en un número T de meses, y se quieren construir como mínimo b_k barcos de cada modelo k.

103.–**Hace mucho tiempo, en una galaxia muy, muy lejana...**

La *Alianza Rebelde* ha obtenido información acerca del emplazamiento de varias bases imperiales en *Endor*. Deben destruirse todas las bases, pero nuestras naves de carga han sido interceptadas...

Nuestra única posibilidad es un bombardero *Y-Wing* cargado con T torpedos. Hay M tipos de torpedos, cada uno de los cuales tiene una potencia t_k. El bombardero lleva n_k torpedos de cada tipo. Cada base imperial debe ser bombardeada con una potencia P_d para ser destruida.

Dado que el número de bombas es limitado, diseñe un algoritmo que **minimice el número de bombas a usar, si puede destruir todas las bases, o en su defecto maximice el número de bases destruidas**.

Que la Fuerza te acompañe...

104.– *Caperucita Roja* ha sido enviada por su abuelita al bosque para recoger flores que inunden su casa con la mayor intensidad de fragancia posible. Dispone de N floreros, cada uno de los cuales puede llegar a contener un peso de flores P_i y un volumen V_i. Cada flor del bosque tiene un volumen v_j, un peso p_j y una intensidad de fragancia f_j. Como la primavera acaba de llegar la mayoría de las plantas aún no han brotado, por lo que en el bosque tan sólo hay F flores. Construya un algoritmo utilizando ***Programación Dinámica*** o ***Backtracking*** que ayude a *Caperucita* a seleccionar las flores y decidir en que florero las meterá para maximizar la cantidad de fragancia en la casa de su abuelita.

Una vez que *Caperucita* ha decidido las flores que tiene que recolectar, se le presenta el problema de recogerlas en el menor tiempo posible, puesto que desea irse a jugar con ese que llaman "*El Lobo Feroz*", y tiene el problema que las flores se encuentran muy dispersas en un bosque tan grande. *Caperucita* que es muy previsora, en sus continuos paseos por el bosque ha ido anotando la localización de las flores, y sus distancias (en tiempo) entre ellas, así como la distancia a casa de su abuelita.

Puesto que la cesta de *Caperucita* tiene una capacidad en volumen V y que no quiere portar más de un peso P, para no estropear las flores. Calcule el tiempo mínimo para recolectar las flores seleccionadas en el apartado anterior.

105.– "*El Dedos*" ha pensado que lo de las cajas fuertes tiene más riesgo que asaltar sistemas informáticos, así que se ha puesto manos a la obra y se ha apuntado a un curso acelerado de Informática, eso sí, sin pagar por adelantado, porque casualmente había visitado jornadas antes de su clausura las oficinas de Opening y había visto lo que había (o mejor, lo que no había) en su caja fuerte.

Como ejercicio para iniciarse en el nuevo arte se había propuesto romper la protección del ordenador de "El Profe". Y como buen observador que era, se había percatado que la contraseña que protegía el acceso al equipo tenía M caracteres, entre los que había L letras y el resto eran dígitos, y que además, no había N números seguidos.

Diseñe el algoritmo que debe programar "*El Dedos*" que genere todas las posibles contraseñas. Debe cuidarse la eficiencia.

106.– Darío, "*el presi*", ya ha intentado escapar de la cárcel 54 veces, por el laberinto de alcantarillas que pueblan el subsuelo de la prisión. Siempre lo han pillado, pero en la última ocasión casi lo consigue, si no hubiera sido por un gato que maulló cuando no debía. Mañana piensa volver a intentarlo, y esta vez quiere que sea la definitiva.

No obstante, los guardias de la prisión piensan poner fin a esta situación y han contaminado el aire de unas cuantas habitaciones del laberinto con un gas somnífero, con el fin de evitar la fuga de Darío. Pero Darío no está dispuesto a quedarse a la sombra más tiempo y ha utilizado sus contactos para conseguir un mapa de las alcantarillas donde están marcadas las salas contaminadas, además, le han informado que podrá soportar la exposición al gas somnífero el tiempo equivalente a atravesar tres salas antes de quedarse dormido como un angelito.

Diseñe un algoritmo basado bien en ***Backtracking*** o bien en ***Programación Dinámica*** que ayude a Darío a encontrar la salida del laberinto por el camino más corto, sin pasar por más de tres salas contaminadas, de manera que la quincuagésima quinta vez sea la vencida y consiga la, tan ansiada, libertad.

107.– Darío, "*el presi*", ha sido atrapado otra vez en su ya quincuagésimo quinto intento. Nuevamente, piensa volver a intentarlo, y nuevamente los guardias de la prisión piensan poner fin a esta situación y han contaminado el aire de unas cuantas habitaciones del laberinto con un gas somnífero, en otras han puesto trampas, y en otras animales carnívoros, con el fin de evitar la fuga de Darío. Pero Darío no está dispuesto a quedarse a la sombra más tiempo y ha utilizado sus contactos para conseguir un mapa de las alcantarillas donde están marcadas las salas contaminadas, las trampas y los bichos. Antes de la fuga Darío cuenta con 100 puntos de vida. Teniendo en cuenta que atravesar una sala con gas le resta 20 puntos de vida, una con trampa 25 y otra con bichos 30. Diseñe un algoritmo utilizando ***Backtracking*** o ***Programación Dinámica*** que nos devuelva el camino de huida para Darío, tal que le quede el máximo de vida disponible para poder disfrutar de su libertad.

108.– Darío, "*el presi*", ha sido atrapado otra vez en su quincuagésimo sexto intento. Nuevamente, piensa volver a intentarlo, y nuevamente los guardias de la prisión piensan poner fin a esta situación. Esta vez los guardias de la prisión han soltado perros rabiosos por algunas salas del laberinto (los perros están entrenados para aguardar a Darío sin moverse de la sala). Suerte que un compañero de prisión de Darío, Filomeno "*el carni*", le ha facilitado un mapa con las salas donde puede encontrar chuletones de ternera para entretener a los perros. De esta forma, Darío podrá pasar por una sala donde haya un perro sólo si tiene un chuletón para echarle.

Diseñe un algoritmo ***Programación Dinámica*** *o* ***Backtracking*** que nos devuelva el camino de huida para Darío. Naturalmente, se trata de huir en el menor tiempo.

109.– Darío, "*el presi*", está harto de que le pillen cada vez que intenta escaparse de la cárcel y ha decidido que si tras el quincuagésimo-séptimo intento (en el que nada podía fallar, pero, inexplicablemente, algo falló...) no ha conseguido escaparse, nunca conseguirá burlar la vigilancia a la que es sometido por los agentes de la prisión de máxima seguridad.

Tras desistir de hallar la libertad por la vía rápida, Darío, se ha convertido en un preso modelo y va poner en marcha un nuevo plan para verse cuanto antes en la calle. Su idea consiste en apuntarse al plan de trabajos de reinserción que ofrecen en la prisión, para poder reducir cuanto antes, por buena conducta, los trescientos veinticuatro años, siete meses y un día que aún le quedan por pasar a la sombra.

En la prisión se ofrece una serie de n trabajos de reinserción de carácter social, que abarcan desde el macramé a la cría de gamusinos, pasando por el ikebana y el cultivo de tulipanes tornasolados. Pero a Darío le da igual lo que tenga que hacer en cada actividad, lo que realmente le interesa es que cada uno de esos trabajos le reducen la pena en un tiempo t_i . Cada uno de los trabajos tiene un horario semanal fijado de antemano, y Darío no piensa renunciar a ciertas actividades, su partidita de mus, comer, dormir y hacer vida social en el patio de la prisión.

Realice un algoritmo basado en ***Backtracking*** o ***Programación Dinámica*** que ayude a Darío a elegir el conjunto de trabajos que debe realizar para conseguir la mayor reducción de su pena, sabiendo que no puede realizar dos trabajos a la vez (los horarios de los trabajos pueden solaparse) y que una vez que se compromete a realizar uno de ellos debe realizarlo por completo para conseguir la reducción de la pena.

110.– Darío, "*el Presi*", después de fallar por sexagésima-sexta vez en su intento de fuga, ha sido obligado a trabajar como cartero dentro de la prisión. Su trabajo consiste en repartir la correspondencia a todos los reclusos de la prisión. Para Darío, esto es una contrariedad, ya que no le permite concentrarse en su *hobby* preferido, planear una nueva fuga. Por esto, ha decidido realizar su trabajo de forma correcta (para evitar represalias por parte del alcaide de la prisión), pero empleando el menor tiempo posible. Para planificar el reparto de hoy, Darío ha estado midiendo la distancia existente entre los distintos puntos de entrega de la correspondencia, y ha construido una matriz de distancias, con el tiempo de desplazamiento, y el tiempo empleado en entregar la correspondencia en cada punto. Darío cuenta con un carrito en el que puede llevar toda la correspondencia, y pretende repartir toda la correspondencia en el menor tiempo posible. Resuelva el problema utilizando un algoritmo basado den ***Programación Dinámica*** o ***Backtracking***.

111.– Darío, "*el Presi*", después de fallar por sexagésima-séptima vez en su intento de fuga… sigue intentándolo. Quiere fugarse de la cárcel por el sistema de alcantarillado, pero tiene dos problemas: El diámetro de la bola que arrastra es de 50 cm y resulta ser demasiado grande para pasar por algunos pasillos. Y otro es que los pasillos que comunican unas alcantarillas con otras tienen demasiada pendiente y sólo puede circular por ellos en un sentido. Varias alcantarillas tienen su salida fuera de la prisión. Darío ha conseguido hacerse con los planos que le indican las salidas, el diámetro de los pasillos, el sentido de las pendientes y el tiempo para recorrer cada pasillo.

Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que ayude a Darío a encontrar el camino que le lleva hacia la libertad en el menor tiempo posible.

112.– Miguel "*el estudiante*", que el único título que tiene es la etiqueta de anís "*El Mono*" en la pared de su celda, es el mejor amigo de Darío "*el Presi*". Miguel ha terminado de cumplir condena. Se ha dedicado todos estos años a la gestión de la Biblioteca del centro penitenciario. Debido a su buena conducta y su amor por los libros se le ha concedido el deseo que tanto añoraba: tener su propia colección de L libros. Antes de marcharse tiene que decidir que libros quiere llevarse, para ello, se le dan C cajas. Cada caja i tiene unas dimensiones $L \times H \times A_i$, es decir, únicamente cambia el ancho de la caja ($A_i<H<L$). Además cada libro j tiene unas dimensiones $L \times H \times g_j$. (**Nota**: Los libros solo caben "*de pie*").

Como Miguel es un negado de la lógica y cuando salga de la cárcel quiere pasar el máximo tiempo posible leyendo, le pide a su amigo Darío "el presi" que le ayude para poder llevarse todas las cajas bien colmadas de libros. Diseñe un algoritmo basado en ***Programación Dinámica*** *o* ***Backtracking*** que le ayude a Miguel a llevarse la mayor cantidad de libros posible.

113.– Un sistema informático tiene que descargar un conjunto de N paquetes de actualización cada uno de los cuales tiene un tamaño Tam_i Kb. Para la descarga de esos N paquetes se dispone de S servidores remotos cada uno de los cuales cuenta con un ancho de banda de V_j Kb/s. El sistema de descarga solo permite descargar un paquete de cada servidor a la vez, pero podemos descargar varios paquetes a la vez procedentes de diversos servidores. Desarrolle un algoritmo que resuelva el problema propuesto indicando qué paquetes debemos descargar de cada servidor, de tal manera que se **minimice el tiempo de descarga**.

114.– Tenemos un procesador que debe realizar una serie de N trabajos. De cada trabajo i sabemos el instante exacto c_i en que debe comenzar, su duración d_i y el beneficio b_i que proporciona si se realiza. El procesador solo puede realizar una tarea al mismo tiempo. Supuesto que el procesador está operativo entre los instantes C y F, diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que maximice el beneficio de los trabajos que se procesen.

116.– **La Herencia**. El abuelo *Milbi Lletes* está en las últimas y sus dos nietos le están insistiendo constantemente para que le cuente qué parte de su inmensa fortuna les deja. El abuelo por su parte no suelta prenda, pero sí les comenta que les dejará una parte de su fortuna a partes iguales.

Días más tarde el abuelo pasa a mejor vida y el notario hace presencia en la mansión "*Villa Billete*". Una vez leído el testamento y llegando a la parte en la que a los nietos se hace referencia, el notario lee: "Yo *Milbi Lletes* quiero dejar todo lo contenido en esta lista de bienes indivisibles, cada uno tasado según la relación que se adjunta, a mis dos nietos, *Loqui Erotodo* y *Paca To*. El reparto, que queda en manos del notario, debe ser lo más equitativo posible.".

El notario viendo lo que tardaría en cuadrar dicho problema, inteligente donde los haya, decidió llamar a dos estudiantes de Ampliación de la Programación para resolverlo. Éstos bien aleccionados en sus clases, le preguntaron que en caso de no ser exacta la división qué debían hacer. El notario tras pensarlo decidió que el mayor de los dos tuviera la mayor parte.

Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que realice el reparto de las pertenencias.

117.– Abundio, después de su "idea feliz" de vender el coche para comprar gasolina, se ha dado cuenta que lo que hizo no estuvo del todo bien y quiere darle salida a toda esa gasolina que ahora no le sirve para nada. Para ello irá por las N casas de sus amigos (Pichote, Tom Tolaba, el que pellizcaba ventanas,...), con el fin de venderla toda y conseguir el dinero que se gastó. Cada amigo le comprará una cantidad c_i distinta de gasolina, según la necesidad de éste. Abundio conocedor de su pueblo, sabe la distancia en tiempo entre las casas de sus amigos por lo que no le será difícil elegir una ruta a seguir.

Pero existe un problema: Abundio, que no se fía de su instinto, quiere vender toda la gasolina antes que empiece su película favorita en televisión, "Dos tontos muy tontos", ya que ésta le puede dar alguna idea de qué hacer con ella. Por ello, suponemos dispondrá de un tiempo T para vender la gasolina.

Ayude a Abundio diseñando un algoritmo de ***Programación Dinámica*** o ***Backtracking*** que devuelva la mejor ruta a seguir con el fin de vender la máxima gasolina posible en el tiempo del que se dispone. **NOTA**: Disponemos de un depósito con ruedas en el que cabe todo el combustible y que vamos desplazando de una casa a otra.

119.– Ante la inminente llegada de la Semana Santa, el Ayuntamiento –quizá conmovido tras años de insistencia, o quizá ávido de votos ante la proximidad de las elecciones– ha decidido por fin, permitir la salida de nuestro paso: el de la *Virgen de los Amargados por Ampliación*.

Por ser nuestro primer año, el alcalde nos concede el beneficio de elegir la ruta por la que irá nuestra procesión. La única restricción que nos impone es el tiempo máximo T que nuestro paso puede estar en la calle.

Por suerte, contamos en nuestra cofradía con Bernabé Ato Acérrimo, vieja gloria de los matriculados en Ampliación, y gran conocedor de la cultura religiosa. Bernabé ha puesto a nuestra disposición un mapa del pueblo, donde se detalla la distancia en tiempo entre los balcones de los saeteros que tiene el pueblo. También sabemos el nivel de ruido r_i de cada saetero, y el tiempo que estará cantando t_i cada saetero. Sabiendo que queremos empezar y

acabar la procesión en la Iglesia del pueblo (las distancias en tiempo desde cada balcón a la Iglesia, así como las de cualquier balcón hasta la Iglesia, son conocidas), supuesto que no pasaremos dos veces por el mismo balcón, diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que nos dé la ruta que debemos seguir para que nuestra procesión sea la que más ruido "meta" esta Semana Santa.

120.– Dado un plano del metro, supuesto que conocemos el tiempo que tarda cada tren en recorrer cada trayecto entre dos estaciones contiguas de una misma línea, el tiempo que está parado en el andén en cada estación, y también, el tiempo que tardamos en efectuar un transbordo de una línea a otra en cada una de las estaciones en las que esto es posible. Diseñe algoritmos que resuelvan los problemas que se indican a continuación:

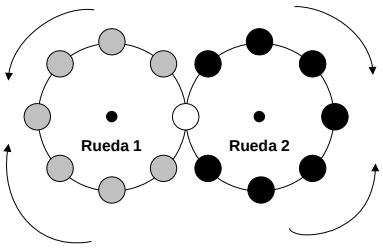
a) Determinar el camino mínimo (en tiempo) entre dos estaciones dadas de la red. Utilice ***Programación Dinámica***.

b) Resuelva el problema del apartado a), supuesto que no podamos hacer más de k transbordos. Utilice ***Backtracking***.

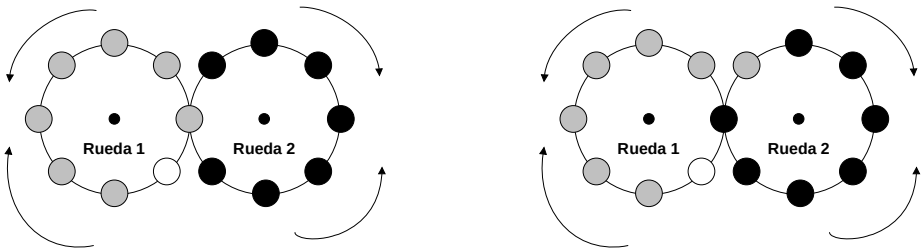
121.– **Cruzar un Puente Estrecho y Peligroso**. Tenemos un grupo de N personas que tienen que cruzar un puente por el que solo pueden ir en fila de a uno, y que además, no soporta más que el peso de dos personas. Sabemos que el grupo solo dispone de una lámpara para alumbrarse y que cada persona p tarda un tiempo t_p en cruzar el puente. Además, cuando dos personas cruzan el puente tienen que llevar la lámpara consigo, y el tiempo que emplean en cruzarlo es el mayor de las dos personas.

Diseñe un algoritmo que devuelva cómo deben cruzar el puente para que el tiempo que empleen en ello sea mínimo. Aplíquelo al caso de un grupo de cinco personas, cuyos tiempos en cruzar el puente de cada componente del grupo son: 1, 3, 6, 8 y 12 minutos.

122.– **Puzzle de las ruedas**. Diseñe un algoritmo que sea capaz de resolver el puzzle de las ruedas. Este puzzle se compone de dos ruedas giratorias y N bolas por cada rueda. De tal manera que al final del algoritmo queden en la siguiente configuración:



Para ver el funcionamiento, observe que tras una giro de la rueda 1 en sentido horario y otro de la rueda 2 en el mismo sentido las configuraciones serían:



123.–Las cartas enviadas por los N niños de un albergue se han extraviado, y no han llegado ni a *Santa*, ni a *Los Reyes Magos*. Aunque éstos por otras colaboraciones saben de las preferencias de juguetes de los niños. Así, tienen datos p_{ij} que les indican la preferencia que tiene el niño i por el juguete j. Tienen el problema que sólo disponen de J juguetes para los N niños, aunque éstos han pedido en sus cartas bastantes juguetes más, pero afortunadamente $J \geq N$. Diseñe algoritmos basados en ***Programación Dinámica*** o ***Backtracking*** que resuelvan cada uno de los problemas que siguen:

a) Supuesto que reparten un juguete por niño:

   i) Devuelva la asignación que maximice la suma de las preferencias.

   ii) Devuelva la asignación que hace posible tener entretenidos a todos los niños el máximo de tiempo posible, puesto que un niño juega con un juguete tanto tiempo como indica su preferencia, tiempo que está entretenido con el juguete, y transcurrido ese tiempo se "cansa" del mismo y pasa a aburrirse o a incordiar al resto de niños. Todos los niños reciben los juguetes al mismo tiempo.

b) Resuelva ii) supuesto que $N \leq J < 2N$ y se pueden repartir hasta **DOS** juguetes por niño.

124.–*Homer* ya está harto de que su malévolo jefe, el señor *Burns*, le "*mangonee*" continuamente, y ha jurado por el Dios *Duff* que esta vez va a ser la última. La última jugada que le ha hecho el señor *Burns* a *Homer* es robarle el juguete preferido por su hija *Maggie*, el oso Bobo. Bien es cierto que este oso perteneció en un tiempo muy lejano al señor *Burns*, pero cada vez que lo vuelve a encontrar lo vuelve a perder. *Homer* ha decidido hacerle probar un poco de su propia medicina al señor *Burns*, y siguiendo los "*buenos*" consejos de sus camaradas *Carl* y *Lenny*, *Homer* está dispuesto a asaltar la mansión del señor *Burns* y robarle la mayor cantidad posible de joyas que éste guarda en N baúles dentro de la habitación 17 del tercer piso. El problema está en que *Homer* sólo dispone de un tiempo T antes de que el fiel sirviente del señor *Burns*, *Smithers*, vuelva para arropar a su querido jefe. Pero este no es el único problema, ya que *Homer* está obsesionado últimamente con su suerte y piensa que asaltar un baúl es tarea fácil, pero que hace que la suerte de *Homer* se tambalee. Además, *Homer* en un esfuerzo inigualable, ha conseguido pensar que si pide consejo a alguien más sabio que él, es más probable que obtenga un mayor número de joyas. Para ello *Homer* ha recurrido a su querida hija *Lisa*. Ayude a *Lisa* a diseñar un algoritmo basado en ***Backtracking*** o ***Programación Dinámica***, de forma que *Homer* consiga robar el mayor número posible de joyas dentro del tiempo que dispone y sin "*tentar*" a la suerte de la que dispone siendo conocido las joyas que hay en cada baúl, el tiempo que se tarda en asaltar cada uno de ellos y la suerte que menguará a nuestro amigo *Homer*.

125.–**El bar de Moe**. En la ciudad de *Springfield* es costumbre para muchos trabajadores de la central nuclear (entre ellos *Homer Simpson*) pasar la noche de los viernes en el bar de *Moe* tomando cañas sin parar hasta acabar tirados por los suelos. Pero *Moe* últimamente ha tenido muchas peleas dentro del bar y ha perdido muchas jarras en las que sirve cerveza. Como todos saben *Moe* es muy rácano, y por eso no quiere comprar más jarras. Por lo que de los N clientes que pueda tener en una noche solo podrá atender a M de ellos al mismo tiempo ($M<N$) siendo M es el número de jarras de que dispone. Por supuesto cada cliente deja una propina p_i diferente a parte del precio de cada jarra y tarda un tiempo t_i determinado en beberse la cerveza.

Ayude a *Moe* a maximizar su beneficio, diseñando un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** para determinar a qué clientes tiene que servir cerveza, siempre y cuando no se sobrepase un tiempo T, ya que el jefe de policía *Clancy Wiggum* le obliga a cerrar a una cierta hora o sino le cierra el local.

126.–*Pluma Blanca* es un pequeño indio que ya tiene edad para pasar a una edad adulta. Sólo le falta la prueba que probará ante los demás miembros de su tribu su madurez. Se trata de dejar a *Pluma Blanca* sólo en el desierto con una pequeña cantimplora de agua y esperar que vuelva al poblado sano y salvo trayendo consigo, un mechón de pelo del puma *Malo Malo*, una ramita del viejo árbol del arroyo en el que se mira la *Luna*, un pedazo de roca del *Monte Olvidado*, una flor de más allá del lago y una serpiente de las *Tierras Lejanas*. Todos estos destinos están muy lejos unos de otros y por lo tanto en algún punto *Pluma Blanca* deberá volver a rellenar su cantimplora. Durante largo tiempo los ancianos de su poblado le han hablado de multitud de fuentes naturales a lo largo de los diversos lugares que tiene que visitar, y *Pluma Blanca* los ha aprendido al igual que las distancias que hay entre estos y los lugares a los que tiene que ir.

Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que le permita a *Pluma Blanca* encontrar todos los objetos para demostrar su madurez, sin deshidratarse y sin tardar más de un tiempo T del que dispone antes de la próxima luna llena.

127.–**El Pirata Barbarroja**. El pirata *Barbarroja*, famoso en el mundo entero por lo malo que llega a ser con sus enemigos, ha encontrado a un náufrago en una isla y en lugar de acogerlo en su barco lo ha dejado en mitad del Mar Caribe, dentro de un pequeño barril y con un mapa. El mapa le indica los posibles recorridos que puede hacer para llegar al país de "*Nunca Jamás*". Pero *Barbarroja* le ha advertido que durante ese recorrido puede caer en las manos de otros piratas más malvados que él, como por ejemplo el *Capitán Garfio*, y lo que sería peor, en las fauces de grandes tiburones sedientos de sangre.

El náufrago tiene siete vidas, por lo que podría salir de siete piratas malvados o tiburones pero lo está pasando muy mal porque se aburre mucho en mitad del mar y está deseando salir lo antes posible de allí. Ayude al náufrago diseñando un algoritmo basado en ***Programación Dinámica*** o ***Backtraking*** para que tarde lo menos posible en llegar a "*Nunca Jamás*" sin que pierda sus siete vidas, utilizando el mapa que le ha dado el pirata *Barbarroja*.

128.–**El Ladrón Astuto**. El *"Pies Largos*" está pensando en hacerse ladrón profesional, sabe algo de programación y ha pensado obtener un algoritmo que le ayude en su tarea.

Ha realizado una lista de *N* posibles joyerías en las que robar y ha estudiado detenidamente el plan para robar en cada una de ellas. Dispone de un plano en el que se especifica el tiempo que tarda en desplazarse entre las joyerías, y para cada joyería tiene el tiempo que tardará en robarla, y el botín que obtendrá. Cada joyería tiene un sistema de seguridad (medio, alto, excelente), y "*Pies Largos*" las ha clasificado atendiendo al mismo. Además, estima que la policía tardaría un tiempo *T* en localizarle y detenerle, esté donde esté, tiempo que empieza a contar desde el momento en el que comience sus fechorías.

Diseñe un algoritmo basado en ***Backtracking*** o ***Programación Dinámica*** para maximizar el botín (volver a su casa antes de tiempo sin ser detenido).

130.–**La Expo de Zaragoza**. Un grupo de amigos, se ha propuesto visitar el mayor número pabellones de la Expo del Agua. Para cada uno de los *N* pabellones se sabe el tiempo que tendrían que esperar en la cola antes de entrar y el tiempo que tardarían en visitarlo. Se sabe

además, la distancia en tiempo que se tarda en desplazarse entre dos pabellones cualesquiera y también la distancia (en tiempo) desde la entrada de la Expo a cada pabellón. Desean visitar **el mayor número de pabellones en un día**, supuesto que disponen de *H* horas. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema.

131.– Flipi Flipao quiere comprarse un coche deportivo, de esos que flipas, pero no tiene suficiente con el dinero que gana en su curro. Por eso se ha buscado otro para obtener un plus y conseguir su "sueño". Con el pluriempleo se ven resultados.

Va a repartir los folletos del partido *LEP* (*Ladrones en Paro)* que se presentará en las próximas elecciones, pero claro su trabajo habitual le ocupa casi todo el tiempo, y para repartir dichos folletos solo dispone de un tiempo *T*. Cuantos más folletos reparte, más dinero le pagan.

Flipi se ha hecho de un plano que le dice la distancia entre dos urbanizaciones, y también la distancia entre cada urbanización y la central en la que debe recoger los folletos. Además, ha realizado sus averiguaciones y sabe cuántos folletos repartirá en cada urbanización, y el tiempo que tardará en repartirlos.

El problema está en los partidarios radicales del partido *TU* (*Timadores Unidos*), rivales del *LEP*, de éstos se sabe que organizan "*partidas*" para "*informar*" convenientemente a cualquier repartidor de propaganda ajena que no debe hacerlo. Al último repartidor que se topó con ellos, todavía le duelen los huesos…

Pero Flipi Flipao no es un gil, y sabe de antemano el recorrido de la partida, concretamente, los instantes inicial y final, que la "*partida*" estará en cada urbanización (puede ser que no vayan a todas las urbanizaciones).

Diseñe un algoritmo que ayude a Flipi a **repartir el mayor número de folletos en el tiempo *T* de que dispone**.

132.– En una ciudad el alcalde ha decidido asfaltar sus calles por motivos electorales, pero tiene el problema de que no puede asfaltar todas sus calles por motivos de presupuesto. Ha decidido asfaltar aquellas calles que unan una plaza con otra, quedando todas las calles asfaltadas unidas. Para cada calle y cada plaza sabe cuánto cuesta asfaltarla, y también su tiempo en hacerlo. Además, sabe por estadísticas de elecciones anteriores y encuestas realizadas recientemente, el número de potenciales votantes de su partido tanto para cada calle, como para cada plaza.

El alcalde se ha propuesto maximizar el número de sus partidarios con plaza o calle asfaltadas, puesto que cree que contribuirá a conseguir afianzar el voto entre los seguidores acérrimos, y en el resto, a inclinar definitivamente la balanza a su favor. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que indique cuáles son las calles y plazas a asfaltar, supuesto que disponemos de un presupuesto *P*.

133.– Tenemos un plano de una ciudad en el que tenemos señaladas las posibles paradas de autobús y las distancias entre ellas. El conjunto *P* de las posibles paradas se ha elegido de tal forma que dado un punto cualquiera de la ciudad la distancia a dicho conjunto no sea superior a *d*.

Se trata de establecer la **ruta circular de menor longitud** que debe seguir un autobús para que cualquier parada de la misma quede a una distancia no superior a *D* de la parada más cercana del autobús. Con ello tendríamos que cualquier punto de la ciudad a un punto de la ruta no estaría a distancia superior a *D*+*d*. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** para resolver el problema.

134.– "**Yo no soy Tonto**" (*¡Quien lo diría!*…) Una conocida cadena de electrodo-mésticos, famosa por sus agresivas campañas publicitarias, ha tenido que devolver el 25% del precio de todos los televisores vendidos entre el 1 de mayo y el 20 de junio, debido a su compromiso publicitario a devolver el "cuarto" del precio del televisor si la Selección Española de fútbol pasaba de "cuartos" en la Eurocopa 2008. Afortunadamente para España y desafortunadamente para los del departamento de publicidad, España pasó.

Firmemente convencidos de su estrategia, los "fantásticos" publicistas convencieron al equipo directivo de que redoblara su apuesta e idearon otra campaña "**1x2**" en la que si los clientes compraban, y hasta un día antes de la final, cierto paquete de electrodomésticos, y España ganaba la final, se les regalaba otro paquete de electrodomésticos del mismo importe que servirían a domicilio sin coste adicional para el cliente. Y el caso es que la Selección, tras años de "pertinaz" sequía, ha ganado la Eurocopa.

Se ven ahora con que tienen que servir *N* paquetes a otras tantas casas (un paquete por casa) y hacerlo cuanto antes, puesto que la clientela se impacienta ante los rumores de que están intentando "echarse para atrás", al hacerse pública una reprimenda del Director General a los publicistas en la que les decía que "*no eran tontos, eran …*". Parece que alguien grabó con un móvil un vídeo con la "escena" y lo ha puesto a circular por Internet, aunque nadie ha podido confirmar su veracidad.

Se sabe la distancia entre todas las casas, y también entre el almacén central y cada casa. Lamentablemente por la imprevisión, sólo disponen una pequeña furgoneta en la que caben hasta *M* objetos.

Se trata de repartir los *N* paquetes a las casas en el menor tiempo posible. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema. **NOTA**: Puede suponerse que sólo se regresa al almacén cuando la furgoneta está vacía, y que cuando regresa al almacén, se carga al máximo.

135.– "*Estoy Queme Rompo*" se ha metido en el mundo de la publicidad. Ha montado una empresa de marketing "*Elmas Guapo S.A*" y está buscando una serie de actores que plasmen su último proyecto de nueva creación durante la presentación del producto a los medios de comunicación.

Se han presentado *N* actores a las pruebas, y según los resultados de las mismas se conocen de antemano la efectividad e_i de un actor al realizar una de las *M* posibles presentaciones, y el tiempo t_i que emplea en realizarla. Además, ha comprobado que el efecto de una exposición

se ve reforzado o disminuido en un porcentaje según la exposición realizada inmediatamente antes, y que es independiente del actor que la realiza. La tabla r_{ij} indica el refuerzo o disminución que experimenta la exposición j realizada inmediatamente después de la exposición i. Considérese la tabla r_{ij} como una constante multiplicativa de la efectividad (mayor que uno positiva, menor que uno negativa).

Se trata de determinar la composición de la presentación, para que **el acto resulte lo más efectivo (convincente) posible**, supuesto que cada actor no realizará más de una exposición y que el acto social deberá durar como máximo T unidades horarias. Diseñe un algoritmo que resuelva el problema.

136.–**El Jardín Oriental.** Tenemos un jardín oriental que pretendemos decorar de tal manera que consigamos un equilibrio del *Chi* según las enseñanzas del *Ikebana*. Según el maestro *Tzin-Tze Tou*, nuestro jardín debería contar con una combinación de colores amplia, pero esto no siempre es posible por ciertas limitaciones, principalmente monetarias.

En nuestro jardín tenemos N jardineras y cada jardinera está dividida en M zonas. No todas las zonas son iguales, E_{ji} es el espacio de la zona i de la jardinera j. Podemos elegir entre P plantas distintas. Cada planta p tiene color c_p y un precio p_p, y necesita de un espacio e_p (en cm^2) de la jardinera. En cada zona solo se puede plantar una planta. Nuestro objetivo es decorar todas las jardineras del jardín, de manera que no haya dos plantas del mismo color en una jardinera y procurando gastar lo menos posible.

Diseñe un algoritmo basado en ***Backtracking*** que nos indique cuál es la colocación de plantas óptima (menor gasto) para conseguir nuestro objetivo.

137.–Dado un tablero cuadrado de N filas, en cada una de sus casillas hay un indicador luminoso, que está apagado (***OFF***) o encendido (***ON***). Al pulsar en una casilla, esa casilla y todas sus adyacentes cambian su estado (de apagado a encendido y viceversa).

Diseñe un algoritmo basado en ***Backtracking*** que determine si es posible apagar todo el tablero, sin pulsar dos veces una casilla. Nótese que alguna casilla puede no pulsarse ninguna vez.

Si es posible apagar el tablero, **determine la secuencia mínima de pulsaciones que apagan el tablero**.

138.–Un grupo de amigos, formado por n parejas (n mujeres y n hombres), se reúnen para cenar. El protocolo del buen comensal exige que, a la hora de sentarse a la mesa (redonda), hombres y mujeres deben alternarse y que nadie debe sentarse al lado de su pareja habitual. Para que la velada sea lo más agradable posible, hay que **maximizar el grado de bienestar total**, obtenido sumando los grados de afinidad mutuos entre los comensales sentados en posiciones adyacentes. Al efecto, se dispone de sendas matrices que nos indican la afinidad entre hombres y mujeres y entre mujeres y hombres. Una vez establecida cada pareja de vecinos, la afinidad mutua se calcula multiplicando la de cada miembro por el contrario. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que encuentre una solución óptima al problema.

139.– El príncipe de un lejano país ha encontrado a su media naranja y al fin ha decidido casarse. Lo ha tomado como un negocio para sacar la mayor pasta posible, y además de poder pagarse el viaje de bodas, que le sobre para pagar el pequeño palacete que se ha construido.

Ha decidido invitar al mayor número de altos dignatarios y demás gente de postín con la idea de que sus regalos estarán a la altura de lo que de ellos se espera. Ahora bien, éstos no están acostumbrados a regalar dinero, y menos aún a desplazarse sin su propio séquito, o no contar con unas mínimas medidas de seguridad. Así cada uno ha demandado tener un número mínimo de agentes que cuiden de su seguridad y hospedarse en un sitio acorde a su *status*. Aunque hay algunos que solo tendrán necesidades de seguridad ya que su viaje será breve estando acompañando a los novios en la ceremonia y el banquete.

El príncipe ha hecho cuentas y ha visto que no tiene suficientes "maromos" para cubrir las necesidades de los invitados, y tampoco es que disponga de demasiadas "residencias".

Determinado como está a optimizar recursos para conseguir sacar la mayor tajada del magno acontecimiento, ha reunido a sus asesores para que le proporcionen la mejor asignación de recursos. Los asesores acostumbrados a "dorarle la píldora" no están por la labor de tener que realizar esfuerzos mentales de tal clase. Así, por unanimidad han resuelto recomendar que sea el recientemente creado Departamento Informático del ministerio de Recaudación, Información y Planificación (RIP), que a algunos de los asesores les está ocasionando algún que otro dolor de cabeza, que **diseñe un programa de devuelva la asignación óptima de recursos para conseguir que asistan los invitados que proporcionen los mayores ingresos**. NOTA: El citado departamento está instalado en las mazmorras del castillo y en él trabaja algún que otro alumno de Ampliación, ya familiarizado con las condiciones de trabajo.

140.– Con la finalización de las obras de los nuevos edificios, la dirección de la ESI tiene que llevar a cabo una reasignación de los despachos a sus profesores. Tenemos N profesores y M despachos. Los despachos son individuales o permiten dos puestos. Naturalmente, un despacho de dos puestos puede también ser asignado a un solo profesor, pero bajo ningún concepto un despacho individual se asignará a dos profesores.

Cada profesor k tiene una categoría laboral $CatProf_k$ y cada despacho es apropiado para hasta una categoría $CatDesp_k$ donde la categoría 0 representa la inferior y la $CatMax$ la mayor. A un profesor de una categoría nunca se le asignará un despacho de categoría inferior.

Entre las restricciones se tiene que cada profesor ha enviado a la dirección una lista de profesores con los que le resulta imposible compartir despacho. Además, la dirección sabe que algunos profesores deben ocupar despacho de forma individual.

Diseñe un algoritmo que determine la asignación que minimiza el número de despachos utilizados.

141.– En la empresa Losa Preta, O.S. andan escasos de espacio y necesitan distribuir los trabajadores de una forma óptima. Disponen de M puestos de trabajo y N trabajadores. Cada trabajador i tiene unas restricciones horarias r_i, una carga de trabajo c_i y una preferencia p_{ip} por cada puesto p de trabajo. Naturalmente en un puesto no pueden estar trabajando al mismo tiempo dos trabajadores distintos.

Diseñe algoritmos basados en ***Programación Dinámica*** o ***Backtracking*** para determinar el horario de tal forma que todos los trabajadores cubran su carga de trabajo y se **maximice la preferencia del conjunto**, según las dos alternativas siguientes:

a) Un trabajador sólo trabaja en un puesto. Esto es, si se le asigna el puesto *k*, entonces solo trabaja en el puesto *k*.

b) Un trabajador puede trabajar en varios puestos en distintas sesiones, pero no cambia en sesiones consecutivas. Por ejemplo, si de 9:00 a 10:00 trabaja en el puesto *k*, y si trabaja también de 10:00 a 11:00, entonces también lo hará en el puesto *k*.

Naturalmente puede ocurrir que las restricciones sean demasiado fuertes y el problema no tenga solución en los términos planteados. En tal caso, determine **cuál es el mínimo número de puestos de trabajo necesarios para satisfacer las restricciones horarias**, y supuesto que los empleados tienen una preferencia nula por los nuevos puestos añadidos determine la **distribución que maximice la preferencial global**.

NOTAS: La carga de trabajo es el número de horas semanales que tiene que trabajar. Las restricciones horarias se entiende que son las horas a las que puede trabajar durante la semana. La preferencia dada se entiende por unidad de tiempo.

142.– A *Frodo Bolsón* le ha tocado la pesada carga de llevar el antiguo anillo de su tío *Bilbo* desde la lejana tierra de la comarca hasta la posada del *Pony Pisador*. Su gran amigo el mago *Gandalf* le ha advertido de los grandes peligros que le acechan por el camino y de que puede utilizar el poder del anillo solamente durante cinco ocasiones antes de ser descubierto por el *Señor Oscuro*. Así, ha decido idear un planteamiento basado en ***Programación Dinámica*** desarrollado durante largo tiempo por los grandes señores *elfos* y transmitido hasta él para hacer más seguro su viaje. Para ello, *Frodo* se ha provisto de un mapa de la comarca con los diferentes lugares por los que puede pasar, y las distancias en tiempo entre ellos, en el que ha anotado los posibles lugares donde estarán los malvados *orcos* esperándole. Diseñe un algoritmo basado en ***Programación Dinámica*** para encontrar el camino mínimo hasta el *Pony Pisador*.

143.– En la *Cima de los Vientos*, la compañía del anillo ha sufrido el ataque de los temibles *Nazgûl* como consecuencia a la imprudencia de los *hobbits*. El resultado ha sido que el portador del anillo, *Frodo*, se debate ahora entre la vida y la muerte por el cuchillo de *Morgul* que tiene clavado cerca del corazón.

Aragorn sabe que en ese estado no durará mucho y antes de partir hacia *Rivendel*, debe preparar un ungüento que frene un poco el avance de *Frodo* hacia *el camino sin retorno*, hacia la muerte.

Para preparar dicho ungüento debe buscar M hierbas curativas, entre los N lugares más cercanos que conoce, sabiendo que en cada lugar crece un único tipo de planta de las M necesarias ($M < N$). Ahora lo importante es conseguir un remedio que consiga hacer que *Frodo* gane algo de fuerzas hasta llegar al refugio. *Aragorn* como buen montaraz, conoce la distancia, en tiempo, a cada uno de los N lugares, y el tiempo que puede llegar a tardar en cada uno, para buscar la hierba curativa que crece en dicho lugar. Debemos ayudar a *Aragorn*

con el uso de ***Backtracking*** o ***Programación Dinámica*** para que consiga las *M* hierbas, en el menor tiempo, de forma que pueda volver a la *Cima de los Vientos* lo antes posible, pues no sabe cuanto tardarán en volver los *Nazgûl*, pero volverán.

144.– Nuestros amigos, tras el triunfo del *Ejército del Oeste* sobre las tropas del *Señor Oscuro*, *Sauron*, en la guerra por *La Tierra Media*, deciden regresar a la ciudad de *Minas Tirith* para celebrar la victoria. Al anochecer celebrarán en el palacio una fiesta en honor del portador del anillo, *Frodo Bolson*. Cansados por los preparativos, *Merry y Pippin*, dos *hobbits* de reputación impecable y expertos catadores de cerveza, deciden ir a "explorar la ciudad" visitando las tabernas de la ciudad, acompañados por enano *Gimli*.

Faramin, *Senescal de Gondor*, les ha proporcionado un mapa turístico de la ciudad, con las *N* tabernas de la Ciudad en el que se indica, además del tiempo que se tarda en llegar a una taberna desde otra cualquiera, el precio de la jarra de cerveza en cada taberna, y el "ambiente" que se respira en la taberna. Como los *hobbits* son muy fiesteros, dependiendo del "ambiente" de la taberna, tardan más o menos tiempo en beberse la cerveza. *Gimli*, sin embargo, siempre tarda lo mismo en beberse una cerveza sea de la taberna que sea. Los tres no abandonan la taberna hasta que todos han terminado con su cerveza. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelvan los problemas que siguen:

a) Suponiendo que disponen de *T* horas hasta el anochecer y que tienen *D* dinero cada uno, se trata de maximizar el número de tabernas visitadas, suponiendo que en cada taberna se toman una cerveza.

b) Resuelva el apartado anterior teniendo presente las siguientes consideraciones:

   i) cada taberna tiene una medida propia de jarra de cerveza,

   ii) cada uno de los "exploradores" tiene un "aguante" determinado, i.e., es capaz de beber una cantidad de cerveza dada, sin emborracharse, y que

   iii) en el momento en el que uno se emborracha deciden volver al palacio.

145.– Es de noche y una horrenda criatura vaga por las zonas pantanosas de la *Tierra Media* cobijada por la oscuridad. Este ser una vez fue un *hobbit*, que respondía al nombre de *Smeagol*, pero ahora se le conoce como *Gollum* por el inquietante sonido que produce al engullir pescado crudo, su único alimento. Las gentes del lugar que aún recuerdan cómo era *Gollum* cuando todavía era *Smeagol* coinciden en que era de carácter afable e incluso estaba estudiando Ampliación de la Programación en *Hobbitton* pero una extraña influencia maligna terminó por ennegrecer su pequeño corazón.

"*El hobbit malo nos ha robado nuestro tessoro, ahora el buen Smeagol no podrá ir a pescar porque sin tessoro, los sucios orcos le darán caza. Ahora sólo podré viajar entre las charcas para cazar pescado mientras dure la noche y los orcos duerman*" – repetía una y otra vez *Gollum*.

Son muchos años los que lleva *Gollum* por la zona y ha terminado por aprenderse la distancia entre las charcas donde está el pescado y la cantidad de peces que puede encontrar en cada una, además del tiempo que tarda en pescar los peces de esa charca. *Gollum* debe cargar con todos los peces que ha pescado, lo que le supone que si en un trayecto entre dos charcas

tardaba de vacío *t*, al desplazarse entre ellas cargado con *p* peces, tardará *t(1+kp)*, donde *k* es constante expresada en tantos por uno.

Diseñe un algoritmo basado en **Backtracking** o **Programación Dinámica** que ayude a *Gollum* a obtener la mayor cantidad de peces posible sabiendo que tiene que estar de vuelta en su cueva antes de que amanezca para lo cual falta un tiempo *T*.

146.– **Lectura de don Quijote**. Tenemos N lectores que van a leer otros tantos fragmentos del Quijote. El lector l lee el fragmento f con una intensidad i_{lf}, y tarda en leerlo un tiempo t_{lf}. **Dado un tiempo máximo T se trata de conseguir la máxima intensidad de lectura**. Resuelva el problema para cada uno de los siguientes apartados:

a) Cada lector leerá un fragmento.

b) Si tenemos M lectores, ($M < N$), devuelva la asignación que determina la máxima intensidad, respetando que si a y b son dos lectores cualesquiera, y n_a y n_b el número de fragmentos leídos por cada uno de ellos, se verifique que $|n_a - n_b| \leq 1$.

147.– Don Quijote y Sancho han vuelto al lugar del que partieron. Y aunque el cura, el ama y el bachiller quemaron los libros de los aposentos de don Quijote. Éste ha dado con unos baúles que dejó olvidados un viajero de aspecto triste, y ha encontrado una colección de *Novelas de Caballerías*, alguna de las cuales hace mención a las hazañas de cierto caballero andante por las tierras de La Mancha.

Alonso Quijano, ha colocado los libros encontrados en una estantería, y ha comenzado a leerlos según el orden en el que se encuentran en la estantería. Si bien, ansioso como estaba por saber de las aventuras de caballeros andantes, no ha reparado en el orden de colocación de los libros en la estantería.

El cura y el bachiller, se temen lo peor. Saben que no pueden volver a quemar los libros, porque Alonso Quijano se volvería loco si de un día para otro desaparecieran los libros, pero han advertido, como se ha dicho más arriba, que Alonso lee los libros según el orden de colocación en la estantería. Y creen haber encontrado una estrategia para demorar en lo posible que se vuelva loco: **reordenarán los libros que aún le quedan por leer a Alonso Quijano para demorar al máximo que se vuelva don Quijote**. A continuación, el bachiller Sansón Carrasco partirá a Salamanca en busca de sabios doctores en busca de consejo para la enfermedad que aqueja a nuestro hombre, con el margen que les proporciona la reordenación de los libros.

Tras averiguaciones varias, han dado con otros casos documentados del mismo mal que aqueja a Alonso Quijano, y han logrado saber para cada libro i el tiempo t_i (en días) que tardará en leerlo, y el desorden psicológico (locura) l_i que le causará su lectura (sólo se la causa cuando ha completado su lectura).

Sabiendo que Alonso Quijano está a L pasos de convertirse en don Quijote, diseñe un algoritmo basado ***Programación Dinámica*** o ***Backtracking***, que determine la **ordenación de los libros que proporcione al bachiller el mayor tiempo** para completar la misión de

encontrar remedio al mal que acecha a Alonso Quijano, ya que si se convierte en don Quijote, el proceso será irreversible hasta que se encuentre en el lecho de muerte.

148.–*Don Quijote* se encuentra camino de *El Toboso* y sigue en su intento de conquistar el corazón de su amada *Dulcinea*, y qué mejor que demostrarle su valentía enderezando entuertos a lo largo y ancho de Castilla. *Sancho* le ha proporcionado un plano con diversos lugares donde poder afrontar una batalla para así obtener mayor fama. Se sabe cuál es la distancia de cada uno de esos lugares hasta *El Toboso* y la distancia de esos lugares entre sí. Cada vez que *Don Quijote* dé feliz término a una aventura, lo cual le requerirá un tiempo t_i, obtendrá una fama f_i y mandará que el vencido vaya hasta *El Toboso* para dar noticia de la victoria de *Don Quijote*. El problema es que como la distancia es el olvido, el vencido aunque cumple con su promesa va disminuyendo la fama que publica de Don Quijote en una constante proporcional *k*. Es decir, si *d* es la distancia hasta el toboso, y *f* la fama profesada por el vencido, entonces la fama que profesará ante *Dulcinea* disminuye en *kfd*. Además *Don Quijote* se ha propuesto estar en *El Toboso* antes de *D* días, pues no puede esperar más para encontrarse con *Dulcinea* y hacerle llegar todo el amor que siente por ella. Diseñe un algoritmo por ***Backtracking*** o ***Programación Dinámica*** que maximice la cantidad de fama obtenida y permita a *Don Quijote* estar en *El Toboso* para el día señalado. **NOTA**: Considérese que si la fama a profesar es negativa, entonces el vencido no irá a ver a *Dulcinea*.

149.–**Don Quijote y Sancho Panza cabalgan de nuevo**.

Don Quijote y Sancho se han decidido a emprender nuevas aventuras. Su fama ha llegado a los más recónditos lugares, sobre pasando los límites de La Mancha, y extendiéndose más allá de las tierras de Castilla y Aragón. Han estado sopesando hacer Las Américas, porque las verdaderas aventuras están donde ningún mortal puso el pie, pero han desistido porque ninguno de los dos es muy amigo de andar sobre el agua.

Continuamente les llegan mensajes desesperados de gentes que no ven solución a sus males y precisan la desinteresada ayuda de un esforzado caballero. No pueden atender a todas las peticiones puesto que no disponen de tanto tiempo como quisieran, y tampoco las fuerzas son las mismas que en sus años mozos. Pero están resueltos a desfacer el máximo número de entuertos que puedan en el tiempo de que disponen y con las fuerzas que les acompañan.

Si algo han aprendido nuestros hombres de sus experiencias pasadas es que necesitan saber donde ir y que las aventuras que emprendan no deben sobrepasar sus fuerzas. Por ello, se han provisto de un mapa en el que han anotado datos que consideran esenciales para culminar con éxito sus aventuras. Así por distintas averiguaciones saben el tiempo T_{ij} que tardan en ir de un lugar *i* a otro *j*, y los esfuerzos Q_{ij} de don Quijote, y S_{ij} de Sancho, que les supone ir de *i* a *j*.

Así que cargados cada uno con fuerzas F_Q, don Quijote, y F_S, Sancho, y sabiendo que disponen de un tiempo *T*, puesto que se han comprometido con sus más próximos para estar de vuelta en su casas y celebrar un año más el aniversario de su primera salida, *en ese lugar de La Mancha*...

Pero nuestros hombres no saben cuál es el camino que tienen que seguir para lograr su propósito de desfacer el mayor número de entuertos, según las consideraciones anteriores. Aunque ha querido el destino que el día anterior al señalado para su partida un viajero ha

llegado al *lugar de La Mancha*, con extrañas vestimentas, y algunos artilugios que de vez en cuando lanzan algún gruñido.

A cambio de un buen caldo, alguna conversación y la promesa de un techo bajo el que pasar la noche les ha proporcionado el camino a seguir, tras estar manipulando la caja que lleva consigo un rato, y lanzar entre tanto, algún que otro improperio a la misma. Improperios que a Sancho le han recordado los tiempos en que a don Quijote le sorbieron el seso los libros de Caballerías.

Interrogado el viajero por cómo sabía la solución al problema, respondió que venía de un tiempo en el que había máquinas que surcaban los cielos, y que con una caja como la que el llevaba podía hablar con otros de lejanas tierras, y resolver problemas como los de ellos tenían, con el uso de la Programación Dinámica y el Backtracking. Porque él había ido a la Universidad y seguido un curso de Algorítmica, entre otras materias.

Sancho Panza al oír semejantes nombres, le ha dicho poco más o menos lo mismo que lo que le dijo a Sansón Carrasco con la gramática. Y ha empezado a sospechar que lo que quiere el viajero es comida y cama gratis. Pero don Quijote le ha aceptado sin recelos cuando les ha contado sus peripecias.

Para el viajero todo empezó cuando bajó a una cueva a rescatar unas doncellas cuyos lamentos se oían a distancia, y que en sus sollozos y lamentos, suplicaban las liberaran de su cautiverio, y al salir de la cueva no encontró sino lagunas en las que el agua hacía mucho ruido, que bien pudiera confundirse con los lamentos de las doncellas.

Los ojos de don Quijote comenzaron a brillar, por fin había encontrado evidencia de lo que en tiempos pasados pensó, y le dijo a Sancho. Y alabando la determinación del viajero, para socorrer gente en apuros, para lo que recordó lo que en cierta ocasión le dijo a Sancho:

"*La libertad, Sancho, es uno de los más preciosos dones que a los hombres dieron los cielos; con ella no pueden igualarse los tesoros que encierra la tierra ni el mar encubre; por la libertad, así como por la honra, se puede y debe aventurar la vida, y, por el contrario, el cautiverio es el mayor mal que puede venir a los hombres.*"

Don Quijote está resuelto a no perder un momento y partir a desfacer el mayor número de entuertos en el tiempo T de que dispone.

También es cierto que el viajero continuamente les preguntaba a nuestros hombres si eran reales, y no fruto de algún sueño o alucinación que a veces tenía resolviendo problemas de programación.

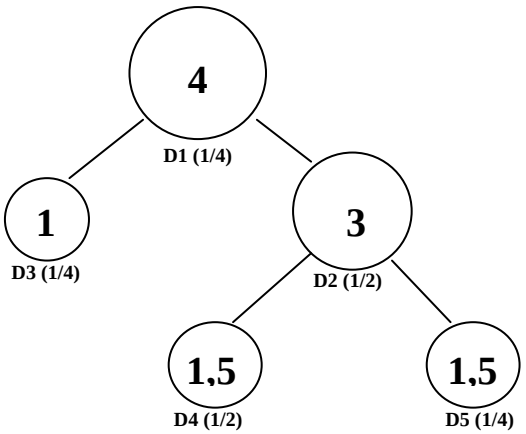
Si vos hubiérais sido el viajero ¿qué programa, basado en ***Programación Dinámica*** o ***Backtracking***, habríais creado para ayudar a don Quijote y Sancho?

150.– En el juego del **Pang** tenemos a un explorador que tiene que usar una pistola con diferentes tipos de disparo para eliminar gradualmente B bolas de diferentes volúmenes V_1, V_2, …, V_B. Cuando el disparo del jugador impacta contra una bola, dicha bola se divide en dos bolas de menor volumen. Existen n tipos de disparos que se van realizando de forma consecutiva y circular, de forma que cuando se ha realizado el n-ésimo disparo, el siguiente será igual al primero de la secuencia. Cada disparo divide la bola en 2 bolas repartiéndose el volumen

entre las dos bolas según el tipo de disparo. La secuencia de disparos es la siguiente: {p_1, p_2, …, p_n}, lo que quiere decir que para el disparo k comprendido entre *1* y n una bola de volumen V se divide en *2* bolas de volúmenes $V_1 = V * p_k$ y $V_2 = V - V_1$. Cuando las bolas tienen el tamaño menor o igual a V_{min}, si vuelven a ser disparadas, desaparecen.

El problema consiste en **encontrar el mínimo número de disparos necesarios para conseguir eliminar todas las bolas**.

Por ejemplo: Tenemos una bola (B=1) de volumen V_1=4, siendo V_{min} = 2, y tenemos 2 tipos de disparos: {1/4, 1/2}. La mejor solución serían 5 disparos, como muestra la figura.



Sin embargo, si el disparo 2 hubiera sido a la bola de volumen 1, habríamos tenido que hacer al menos 7 disparos (ya que la bola de volumen 3 se dividiría en 2 bolas de volúmenes 0,75 y 2,25 respectivamente).

Diseñe e implemente un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema del **Pang**.

152.– La empresa *Enredando Inc*. desea contar con las últimas tecnologías y ahora se ha embarcado en el proyecto de dotar a todas sus instalaciones con red inalámbrica (WiFi). La responsabilidad de la implantación de la red WiFi ha recaído en su departamento técnico, que ha realizado un estudio al respecto. En el estudio se incluye un mapa de las instalaciones de la

empresa en el que se muestra la disposición de los puntos en los que se pueden instalar las antenas, los puntos de la empresa que deben tener cobertura y la distancia [lineal] existente entre cada par de puntos.

Las especificaciones técnicas de las antenas WiFi indican que tienen una cobertura circular de R metros de radio y pueden ser configuradas en tres canales diferentes. Además, advierten que en el caso de que exista un solapamiento en la cobertura de las redes, este solapamiento no debe ser de señales pertenecientes al mismo canal, pues se producirá una interferencia en las señales y la zona de intersección quedará sin cobertura.

Diseñe un algoritmo basado en ***Programación Dinámica*** *o* ***Backtracking*** que encuentre la disposición óptima de las antenas (posición y canal), de tal manera que utilizando el menor número de antenas se proporcione cobertura a todos los puntos especificados, evitando las interferencias.

153.–La empresa de instalaciones eléctricas "*Los Bombillas*", ha resultado la adjudicataria del concurso para la instalación del alumbrado de la ciudad. Han realizado un estudio y disponen de un plano en el que se detallan los puntos que deben iluminarse, y que son los mismos en los que es posible instalar farolas, además tenemos la distancia lineal entre ellos, y se sabe que cada farola tiene un radio de iluminación de R metros. Se trata de **iluminar todos los puntos señalados minimizando el número de farolas empleadas**. Diseñe un algoritmo ***Backtracking*** que resuelva el problema.

154.–*Fotoman* es un fotógrafo *freelance*, que cobra según los eventos que cubre. En la ciudad hay N eventos que pueden ser cubiertos. Para cada evento sabe el beneficio que le reporta el cubrirlo, el instante del inicio $t_{e,inicio}$ y del final $t_{e,fin}$ del evento, y del tiempo t_e que tarda en cubrirlo ($t_e \leq t_{e,fin} - t_{e,inicio}$). Dispone también de la distancia en tiempo que tarda en desplazarse entre cada dos eventos t_{ij}. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que permita a *Fotoman* **maximizar el beneficio de los eventos cubiertos**.

155.–*De Campaña*. Tenemos un mapa de carreteras con N puntos críticos, que podemos clasificar en *Puntos Negros* y/o *Recaudatorios*. Dado un periodo determinado de tiempo fijo, para cada punto sabemos la recaudación que se obtiene y el número de accidentes que se producen atendiendo a estas tres alternativas: (1) no cubrirlo; (2) cubrirlo con un radar; (3) cubrirlo con una dotación de agentes de tráfico. Disponemos de R radares y P dotaciones de agentes de tráfico ($P < R$). Se pide resolver uno y sólo uno de los dos siguientes problemas:

a) Maximizar la recaudación sin que se sobrepase una cantidad M de accidentes prefijada de antemano.

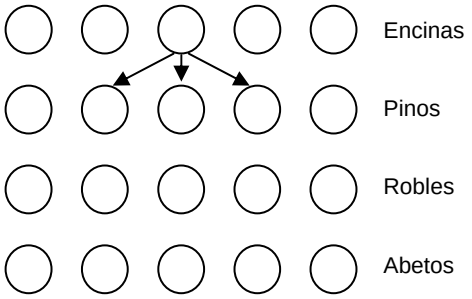
b) Minimizar el número de accidentes.

Diseñe el algoritmo que resuelva la alternativa elegida.

156.–El Tío Antonio tiene en su granja dos vacas: *Devoradora* y *Listilla*. Antes de ordeñarlas las alimenta llenando una hilera de *N* cubos con pienso (*N* es par). Cada cubo *i* contiene una cantidad p_i de pienso indicada en el cubo (todas las cantidades son distintas). Cada vaca en su turno debe elegir el cubo de uno de los extremos, y comerse su contenido. El cubo se retira y el turno pasa a la otra vaca. Se sigue así hasta agotar los cubos. La vaca que comienza comiendo se determina a priori por un procedimiento cualquiera. El objetivo de ambas vacas es comer en total lo máximo posible. La estrategia de *Devoradora* consiste en pensar poco y escoger el cubo de los extremos que esté más lleno. En cambio, *Listilla* prefiere pensárselo un poco más, para lo que ha adquirido lo último en computadoras vacunas portátiles.

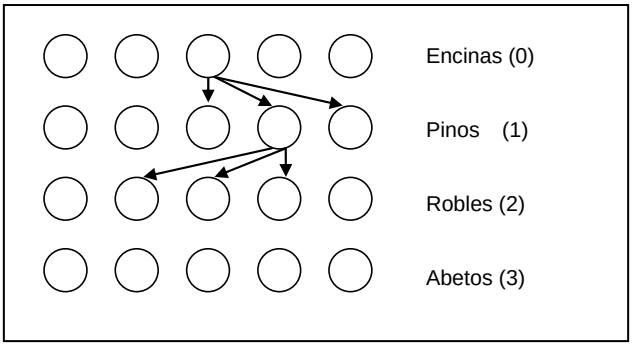
Listilla ha cursado un *master en Informática por la Escuela Superior Bovina*, pero en dicha escuela no estudian la ***Programación Dinámica***, por lo que solicita su ayuda para diseñar un algoritmo utilizando dicha técnica, suponiendo que le toca empezar a escoger.

157.–**Problema del leñador**. Un leñador trabaja en una plantación en forma de cuadrícula de *MxN* árboles. Cada fila del bosque está formada por *M* árboles del mismo tipo, y el bosque tiene *N* filas de árboles de diferente tipo. Cada árbol tiene una anchura propia *A* que determina el tiempo que tarda en ser talado. Un leñador tiene que talar *N* árboles todos de tipos diferentes. Para ello, el leñador va talando el bosque de fila en fila, de forma ordenada. El leñador sólo puede talar los árboles de la fila siguiente que están o bien en la vertical o en la diagonal con respecto al árbol que acaba de talar. Véase la figura adjunta. El problema consiste en encontrar una solución que minimice el tiempo que tarda el leñador en talar *N* árboles, cada uno de diferente tipo.



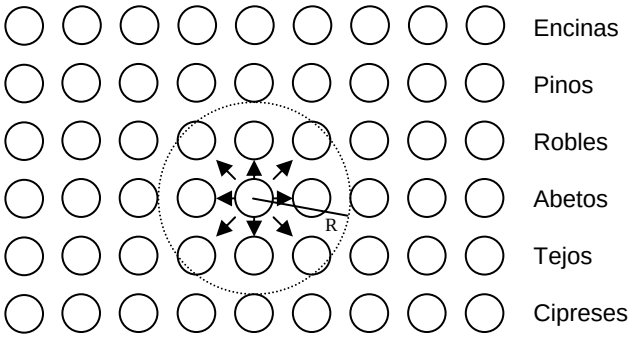
Resuelva el problema mediante estrategias de ***Programación Dinámica*** o ***Backtracking***.

158.–**Problema del Leñador v.2**: Suponga que ahora en las filas pares el leñador puede talar el árbol de la fila siguiente que está en la vertical y los dos a su derecha, y en las filas impares el árbol de la fila siguiente en la vertical y los dos a su izquierda. Véase la siguiente figura:



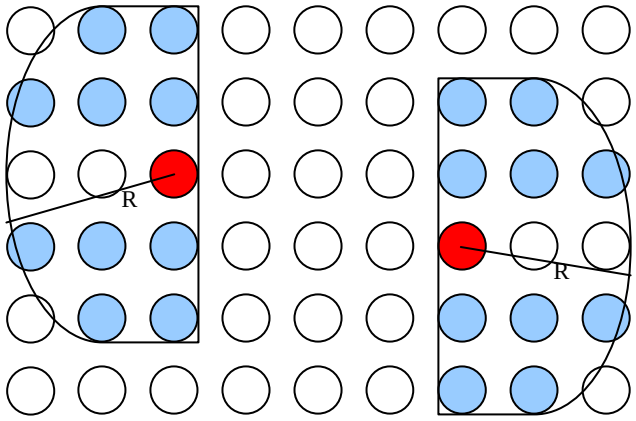
Resuelva el problema mediante estrategias de ***Programación Dinámica*** o ***Backtracking***.

159.–**Problema del Leñador Variante**: El leñador puede talar cualquier árbol que esté en un radio *R* con respecto al que acaba de talar. Para ello, los árboles están separados una distancia constante entre sí. Véase la figura que sigue. El problema consiste en encontrar una solución que minimice el tiempo que tarda el leñador en talar *N* árboles, cada uno de diferente tipo.



Diseñe un algoritmo ***Backtracking*** que resuelva esta variante.

160.–**Problema del Leñador Variante 2**: Si la fila en la que acaba de talar un árbol es par, entonces los siguientes árboles que puede talar son los que estén dentro del radio y en la misma columna o columnas a la izquierda. Si la fila es impar, solo puede talar los árboles que estén dentro del radio y en la misma columna o columnas a la derecha. Véase la siguiente figura:



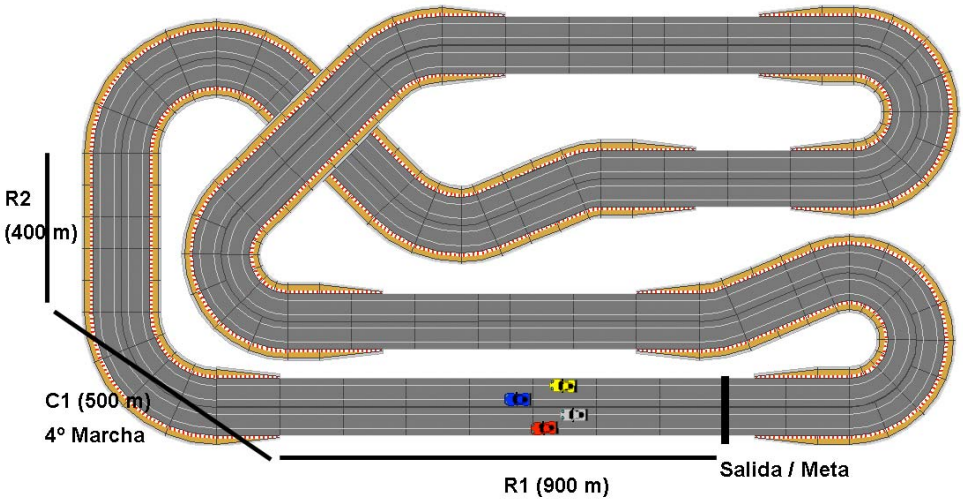
Resuelva esta variante del problema del leñador planteada utilizando ***Backtracking***.

161.–**Problema de los Remontes**. Un esquiador quiere subir hasta la cima, ubicada a *M* metros de altitud. Para ello debe coger una combinación de varios remontes partiendo de altitud 0. La estación está organizada en *L* niveles distanciados unos de otros en *N* metros de altitud (N = *M* / *L*). De cada nivel salen *R* remontes. Cada remonte sube una cierta altura en un tiempo

concreto según una tabla conocida. **El problema consiste en encontrar la ruta que minimice el tiempo que tardará el esquiador en subir los *M* metros**. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema.

162.–**Fórmula 1**. La nueva normativa de la Federación Internacional de Automovilismo obliga a los equipos de F1 a reducir considerablemente su presupuesto. El equipo Ferrari se ha visto obligado a prescindir de su simulador de carreras y se pone en manos de los estudiantes de Algorítmica.

Disponemos de la representación de un circuito. Se trata de una consecución alternativa de rectas R_i y curvas C_j, conociéndose los metros que tienen. Además conocemos que un monoplaza recorre un número de metros establecido por unidad de tiempo dependiendo de la marcha *M* que lleve. Suponiéndose seis posibles marchas ($M_1, M_2,... M_6$), siempre se cumple que si mr_i son los metros recorridos con la marcha *i* por unidad de tiempo, tenemos $mr_1<mr_2<...<mr_6$. Finalmente, conocemos la marcha máxima con la que se puede llegar a una curva para no salirse de la misma. En cada unidad de tiempo sólo se puede subir una marcha, bajar una marcha o mantener la marcha actual. Se desea saber cuál es la secuencia de marchas que debe poner Alonso para minimizar las unidades de tiempo que tarda en recorrer una vuelta del circuito, sin salirse en ninguna curva. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema.



163.–La **Liga Fabulosa** es una competición en la que los participantes disponen de un presupuesto *P* para configurar un equipo de fútbol a partir de un listado dado de jugadores. De cada jugador se conoce la posición en la que puede jugar, que puede ser Portero, Defensa, Centrocampista o Delantero. El equipo debe tener exactamente once jugadores, respetando que debe haber un portero y un mínimo de dos y un máximo de cinco jugadores de cada una de las posiciones restantes. Para cada judador *j*, se conoce el precio p_j de su ficha, y su valoración v_j ($v_j \in [0,10]$) en base a sus resultados en las ligas anteriores. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que determine los once jugadores que deben ficharse para conformar el equipo, sin sobrepasar un presupuesto dado *P* y

maximizando la valoración total del equipo, es decir, la suma de las valoraciones de los jugadores del mismo.

164.– Pedro, ya mayor, ha venido a Valladolid desde Ávila, también lo han hecho Paco y Régula desde el cortijo, y con Cipriano Salcedo se han dirigido a ver a su admirado maestro Miguel. Durante la conversación han comentado que en *El Norte de Castilla* ha aparecido una crónica relativa a un curioso caso: un sujeto anónimo ha pensado en regalar a sus amigos libros de don Miguel. Para cada libro sabe su precio y el placer o preferencia que cada uno de sus amigos tiene por cada libro. Tiene N libros y M amigos ($M \leq N$), se trata de **maximizar el placer total según los libros regalados, regalando un libro a cada uno**, si bien también sabe que sus amigos tienen algunos de los libros, y por tanto, el regalarle uno de esos libros a alguien que lo ya lo tiene, no le proporciona ningún placer, aunque le estará agradecido. Diseñe un algoritmo que resuelva el problema utilizando ***Backtracking*** o ***Programación Dinámica***.

165.– En la comarca de "*El Buen Queso*", cada pueblo de los N existentes, se ha especializado en fabricar un tipo de queso de los M tipos por los que es conocida. Además en cada pueblo se vende el queso a un determinado precio. Pepe Carachancla, conocido como "*El Quesero*", debe de hacer un reparto para la gran ciudad de P tipos de quesos distintos ($P \leq N$) que le han encargado sus clientes. Dispone de un mapa con la distancia en tiempo de un pueblo a otro de la comarca, y el tiempo que tarda en un pueblo en comprar el queso, diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que permita a Pepe, conseguir recoger los P quesos en el menor tiempo posible.

Suponga ahora que Pepe dispone únicamente de un tiempo T, sabe el precio al que le cuesta cada tipo de queso (el mismo todo el pueblo), y el precio al que se lo vende a su cliente, y trata de conseguir maximizar su beneficio. Diseñe un algoritmo que resuelva el problema.

166.– **Los Mayos**. Comienza el mes de mayo, y una cuadrilla de mozos se dispone ha efectuar la ronda de los mayos, a las mozas del pueblo. El *Informático* ha venido al pueblo a tomar fuerzas para enfrentarse a los exámenes finales, y le han encargado la tarea de conseguir rondar de forma "efectiva" a todas las mozas. Ha recopilado algunos datos como el tiempo de desplazamiento entre las casas de las mozas a rondar y de cada moza el tiempo que tardan en cantarle el mayo. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que devuelva para un tiempo dado, cuál ha de ser la ruta para rondar a la mayor cantidad de mozas de una lista dada.

MODIFICACIÓN: Resuelva el problema teniendo en cuenta que al *Informático* algunos de los mozos de la cuadrilla lo han sobornado para que necesariamente ronden a una serie de mozas.

167.– Juanito se ha comprado una cosechadora para vendimiar. Se aproxima la época de la recolección de la misma y a Juanito le llegan muchos clientes para pedirle que recoja su

cultivo ya que éste tiene unos precios muy competitivos. Ante la avalancha de clientes, Juanito tiene que hacer una selección de los mismos ya que le va a ser imposible atender a todos porque la uva tiene una fecha a partir de la cual pierde su valor. De cada parcela se sabe precio que tiene que cobrar, el tiempo que emplea en la recolección y la fecha a partir de la cual no merece la pena recoger el producto porque "caduca" y no se producen beneficios.

Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que ayude a Juanito a maximizar el beneficio puesto que necesita amortizar cuanto antes el coste de la cosechadora.

Resuelva el problema suponiendo que un cliente puede tener varias parcelas, y que todas las parcelas del cliente tienen que recolectarse consecutivamente.

168.– El presidente de la sociedad ha encargado una encuesta a una agencia de sondeos afín (tiene estrechos lazos con el presidente de la agencia puesto que lo nombró él). Pretende que la encuesta avale su gestión y sirva para desprestigiar a la competencia. La agencia ha parcelado el dominio en N zonas. Por sondeos anteriores, de cada zona se sabe el porcentaje de partidarios del presidente y de los partidarios de la competencia. Se trata de elegir M zonas de entre esas N de tal forma que la diferencia entre partidarios del presidente y de los de la competencia, se maximize. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema.

169.– Una potencia invasora ha planificado la invasión de una serie de países, cada invasión supone un coste en descontento d_i entre su propias tropas y sus ciudadanos (cuesta vidas e impuestos), y un beneficio b_i, puesto que puede disponer de sus materias primas. Además el país ofrece una resistencia a la invasión r_i. El departamento de planificación también ha establecido un orden entre algunos países para ser invadidos, representado en una secuencia, s. Así, si $i < j$, entonces el país s_i debe invadirse antes (no necesariamente inmediatamente antes) que el s_j, puesto que sino es inviable el éxito. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema.

170.– Un barco de transporte de mercancías, con una capacidad máxima de carga C, ha sido adquirido por una empresa que quiere maximizar los beneficios obtenidos con su utilización. El barco transporta mercancías entre una lista de P puertos. En cada puerto se pueden comprar y vender M productos diferentes. El precio de producto varía de un puerto a otro. En cada puerto lo que hace es vender el producto que lleva de carga (vende toda su carga) y carga otro producto de ese puerto (la cantidad a cargar dependerá de la capacidad del barco, y de la disponibilidad de capital). Conocemos los días que tarda el barco en ir desde cada puerto a todos los demás (se incluye el tiempo de carga y descarga). El capitán dispone de un capital inicial D para comprar mercancías.

El problema consiste en encontrar la ruta que maximiza los beneficios en un tiempo T (en días), teniendo en cuenta que la ruta debe comenzar y terminar en el mismo puerto, y además, no se puede repetir ningún puerto. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema.

171.–**Ascensores Inteligentes**. Un edificio de *P* plantas dispone de dos ascensores, cada uno con una capacidad de *C* personas, que tardan un tiempo de *t* segundos en desplazarse de una planta a la siguiente. Se ha detectado que la hora punta es entre las 11:00 y las 11:05 habiendo un tránsito masivo de personas que se desplazan a través de las distintas plantas. Cada persona que desea hacer un trayecto, indica la hora a la que desea coger un ascensor, la planta origen y la planta destino.

Supuesto que tenemos la lista de *N* trayectos solicitados, correspondientes a otras tantas personas (esto es, una persona solo solicita un trayecto), se trata de planificar el funcionamiento de los ascensores para que se minimice el tiempo de espera para entrar en el ascensor del conjunto de las personas. Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema.

Resuelva el problema si se trata de minimizar el tiempo de servir las *N* peticiones, sin que una persona sobrepase un cierto tiempo de espera *TEsperaMaximo* antes de subir al ascensor, ni esté en el ascensor más de un tiempo *TDentroMaximo*.

Nota: Despréciese el tiempo de apertura y cerrado de puertas, y el tiempo empleado en entrar y salir del ascensor.

172.–Tras conseguir proclamarse campeones del mundo, la selección de baloncesto ha regresado a casa. "*Las Autoridades*" han pensado en homenajearles por todo lo alto. Conscientes de que cada homenaje le proporciona al equipo nacional en su conjunto (dirección técnica incluida) una satisfacción dada, que suele ser distinta según el homenaje, quieren maximizar la satisfacción total de la selección nacional en los homenajes que les hagan, como reconocimiento.

Puestos a pensar han elegido *N* homenajes posibles, de cada uno de los cuales saben su duración d_i y la satisfacción s_i que les proporciona. Además, los homenajes se van a realizar en distintos lugares, y también se sabe el tiempo de desplazamiento t_{ij} entre los lugares en que se desarrollarán los homenajes *i* y *j*. El instante en que comienza el homenaje es indiferente, puesto que tienen carta blanca para detener el tráfico en la Gran Vía o el Paseo de la Castellana, o incluso suspender la sesión parlamentaria del día –si es que a esas horas sus señorías se encontraran trabajando o debatiendo…

Hay un homenaje fijo, que es la recepción ("*La Recepción*") a la selección por parte de *Las Autoridades*. Dada la apretada agenda de las mismas, con compromisos ineludibles, "*La Recepción*" debe comenzar a las *C* horas y finalizar a las *F* horas, exactamente.

El viaje de la selección ha sido largo y se merecen un bien ganado descanso. Así pues, los homenajes comenzarán por la mañana a partir de la hora *HoraInicio* y terminarán por la tarde a la hora *HoraFinal*, puesto que desean regresar con los suyos y disfrutar de unas breves pero merecidas vacaciones.

Diseñe un algoritmo basado en ***Programación Dinámica*** o ***Backtracking*** que resuelva el problema, es decir, **devuelva la secuencia de homenajes que maximice la satisfacción del equipo nacional y la satisfacción obtenida**.

Nótese que puesto que hay un homenaje fijo, que es "*La Recepción*", se trata de determinar los homenajes que deben realizarse de tal forma que se maximice la satisfacción de los

componentes del equipo nacional, supuesto de que disponemos de *TiempoAntes* (= *C* – *HoraInicio*) antes de "*La Recepción*" y *TiempoDespués* (= *HoraFinal* – *F*) después de la misma.

173.– En lo profundo del bosque, muchos de ellos lejos de su casa, se agolpan los *hobbits* en la entrada de la cueva para mantener distraída a *La Bestia* con sus acertijos. Reunidos previamente los *hobbits* se han aprendido todos *N* acertijos que sabían individualmente. Cada *hobbit h* tarda un tiempo p_{ha} en plantear el acertijo *a*, y *La Bestia* tarda en resolvérselo r_{ha}, no necesariamente los mismos para cada *hobbit*, puesto que cada uno tiene una forma de expresarse, y de acuerdo a como se exprese la bestia tarda más en entenderlo, aún siendo el mismo acertijo.

Diseñe un programa que indique a cada *hobbit* cuál es el acertijo que debe plantear para mantener a *La Bestia* el mayor tiempo ocupada, bajo los siguientes supuestos:

b) A la entrada de la cueva se agolpan *N hobbits* y cada uno plantea un acertijo distinto.

c) En la entrada de la cueva hay *M hobbits (M < N)*, plantean todos los acertijos, cada uno distinto, pero puede haber *hobbits* que no planteen ninguno.

d) Como en b) pero cada *hobbit* plantea al menos un acertijo.

e) Como en c) pero ahora el tiempo de respuesta de *La Bestia* a los segundos y sucesivos acertijos planteados por un mismo *hobbit* se reduce a la mitad para cada uno respecto al tiempo considerado inicialmente.

174.– Suponga ahora que *La Bestia* tiene *C* cabezas (*C << M*), con cada una de las cuales atiende a un *hobbit*, y que en cuanto una cabeza no esté ocupada se dará cuenta de la estrategia de distracción de los *hobbits*. Cada uno de los *M hobbits* plantea un acertijo distinto, y aunque pueden ser varios acertijos, no puede plantearlos al mismo tiempo a distintas cabezas. El tiempo sólo depende del acertijo y del *hobbit* que lo plantea, no de la cabeza. El problema es el mismo que en el problema anterior, pero las hipótesis son:

a) Hasta que no se ha resuelto completamente el acertijo, el *hobbit* no puede plantear otro acertijo a otra cabeza.

b) El acertijo se plantea y a continuación el *hobbit* puede plantearle otro a otra cabeza.

175.– **El vicioso empedernido**.

Juan es un padre y esposo ejemplar. Su suegra espera impaciente el día 4 de mayo de cada año, fecha en la que ésta celebra su cumpleaños pues, además de hacerle un estupendo regalo, Juan, al felicitarla, llora inconsolable al teléfono por no poder estar a su lado y con el resto de su familia que van a pasar tan señalado día a casa de la abuela. Pero lo que no sabe la buena señora, ni la esposa ni los hijos pueden imaginar es que la tristeza le dura a Juan lo que tarda en colgar el teléfono ya que dispone de 24 horas para hacer lo que le venga en gana, tiempo en el que se dedica a sus vicios: **beber**, **fumar** y **jugar**.

Pero Juan es una persona metódica y "juiciosa", y ha comprobado tras un exhaustivo análisis que cada uno de sus vicios v le consume un determinado tiempo t_v y le proporciona un determinado grado de placer p_v. No obstante, también ha observado las siguientes consideraciones:

a) Se emborracha si la razón entre las copas bebidas y el tiempo en horas que tarda en bebérselas es mayor que k, siempre que se haya bebido al menos C copas en las h últimas horas.

b) Cuando se emborracha, en la primera hora experimenta un placer de p_b, y el resto del día duerme la mona.

c) Borracho cualquier otro vicio no le proporciona placer.

d) No puede fumar y beber a la vez.

e) Si durante una partida fuma o bebe el placer proporcionado por este segundo vicio se reduce al 50%.

f) La tercera partida que juegue seguida no le proporciona placer alguno.

¿Cuál es la secuencia que le proporciona mayor placer al cabo de las 24 horas de desenfreno de las que dispone?

176.– Proponga un problema que pueda ser resuelto por Backtracking o Programación Dinámica e impleméntelo.

1.– **Juego de los Palillos**: Tenemos k filas con n_k, palillos cada una y dos jugadores que, alternativamente, pueden quitar de una y solo una fila tantos palillos como desee (por lo menos uno), ganando el que se lleva el último. Construya un algoritmo que evalúe el juego, para una posición del mismo dada, indicando la jugada a realizar.

2.– **Juego**: Tenemos N montones cada uno con n_i cerillas ($1 \leq i \leq N$). Dos jugadores cogen alternativamente tantas cerillas como se desee (al menos una), pero de un solo montón. Gana el que se lleva la última cerilla. Construya un algoritmo que evalúe el juego, para una posición del mismo dada, indicando la jugada a realizar.

3.– **Juego**: Tenemos N cerillas ($N > 2$) y dos jugadores. El primero puede iniciar el juego cogiendo tantas cerillas como desee, pero por lo menos una y dejar otra. Van jugando alternativamente, pudiendo cada jugador coger tantas cerillas como quiera pero cogiendo siempre alguna y menos que el doble de las tomadas en la jugada anterior por su adversario. Gana el que se lleva la última cerilla. Construya un algoritmo que evalúe el juego, para una posición del mismo dada, indicando la jugada a realizar.

4.– **Juego**. Tenemos N baldosas cada una con un tamaño b_i ($1 \leq i \leq N$), y un camino para cubrir M metros. Dos personas juegan alternativamente a poner una baldosa de entre las dadas (y no utilizadas aún) hasta cubrir el camino. Pierde aquel que no pueda colocar una baldosa en su turno. Construya un algoritmo que evalúe el juego, para una posición del mismo dada, indicando la jugada a realizar.

Construya también un algoritmo que simule el juego entre una persona y la máquina, suponiendo que la máquina sigue la estrategia utilizada en el párrafo anterior.

5.– **Juego**. Sobre la mesa tenemos un montón de N monedas, cada una de un determinado valor. Dos jugadores cogen alternativamente una moneda del montón. Gana el primer jugador cuyas monedas suman **exactamente** una cantidad dada K, prefijada de antemano. Construya un algoritmo que dada una configuración del juego la clasifique, y devuelva la decisión que produce el resultado indicado. **NOTA**: Puede haber varias monedas con el mismo valor.

Por ejemplo: Si el Montón inicial de monedas es (representadas por su valor) $C = \{1,2,2,3,4,5,6,6,7\}$ y $K = 15$, el desarrollo del juego puede ser como sigue:

- A elige *5* y queda $C = \{1,2,2,3,4,6,6,7\}$ $S_a = 5$ $S_b = 0$
- B elige *7* y queda $C = \{1,2,2,3,4,6,6\}$ $S_a = 5$ $S_b = 7$
- A elige *6* y queda $C = \{1,2,2,3,4,6\}$ $S_a = 11$ $S_b = 7$
- B elige *6* y queda $C = \{1,2,2,3,4\}$ $S_a = 11$ $S_b = 13$
- A elige *4* y **Gana**, puesto que la suma de las monedas que ha elegido *es* $S_a = 15$ $(=5+6+4)$

6.– **Juego**. Tenemos un tablero de *M x N* casillas, en cada una de las cuales hay una cantidad de monedas, no necesariamente la misma en cada casilla, y además algunas casillas están vacías. Dos jugadores juegan a ver quien consigue el mayor número de monedas.

El desarrollo del juego es como sigue: Un jugador situado en una casilla *C*, puede desplazarse a una de las casillas adyacentes que tenga monedas y toma todas las monedas que hay en dicha casilla, y a continuación, pasa el turno al otro jugador; si no puede desplazarse a ninguna casilla, pasa el turno al otro jugador. Se supone que el estado inicial del juego es que los dos jugadores están ya posicionados sobre el tablero y las casillas sobre las que están no tienen monedas. Construya un algoritmo que dada una configuración del juego la evalúe, y devuelva la decisión que produce el resultado indicado.

7.– **Juego**. Tenemos un tablero de $M \times N$ casillas, en alguna de las cuales hay obstáculos. Dos jugadores juegan alternativamente al siguiente juego. Un jugador situado en una casilla puede desplazarse a cualquiera de las casillas adyacentes, tanto en diagonal, horizontal como verticalmente, que no tenga obstáculos y no haya pasado por ella anteriormente ninguno de los dos jugadores. Si un jugador no puede moverse pasa el turno. Gana el jugador que visita más casillas. Construya un algoritmo que evalúe el juego, para una posición del mismo dada, indicando la jugada a realizar.

8.– **Juego**. Sobre una mesa tenemos una colección de *N* objetos, cada uno de los cuales tiene un peso y valor dados. Dos jugadores cogen alternativamente un objeto de la colección. El peso total de los objetos que coge un jugador no puede sobrepasar una cantidad dada *K*, prefijada de antemano. Gana el jugador que logra reunir la colección de objetos de mayor valor. Construya un algoritmo que dada una configuración del juego la evalúe, y devuelva la decisión que produce el resultado indicado.

9.– **Juego.** Tenemos un conjunto de *N* fichas de colores, cada una de un determinado valor, de entre *K* valores posibles ($K<N$). Dos jugadores cogen alternativamente fichas del conjunto, ganando el que coge la última ficha. En cada jugada un jugador puede coger bien todas las fichas de un color, o bien todas las fichas del mismo valor. Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

10.– **Juego**. Tenemos un conjunto de *N* enteros positivos. Dos jugadores juegan alternativamente a elegir un número del conjunto y eliminar todos sus divisores. Gana el que elimina el último número. Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

11.– **Juego.** Tenemos un conjunto de *N* enteros positivos. Dos jugadores eligen alternativamente un entero del conjunto y lo eliminan junto con todos sus divisores. Pierde el jugador que elimina el último número. Construya un algoritmo que dada una configuración del juego la evalúe, y devuelva la decisión que produce el resultado indicado.

12.– **Juego**. Se considera un número entero de *N* dígitos. Dos jugadores juegan alternativamente. En su turno, cada jugador puede efectuar dos tipos de movimientos:

a) Cambiar un dígito por otro menor. P.e. en el número 9735478 podemos cambiar 5 por 4, 3, 2, 1 ó 0. En concreto, si cambiamos 5 por 2 quedaría 9732478.

b) Si uno de los dígitos es 0, se puede eliminar, eliminando al mismo tiempo todos los dígitos situados a su derecha. P.e., si en número inicial 85972046 elimina el 0, queda el número 85972.

Pierde el jugador que elimina el último dígito.

Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

13.– **Juego**. Se considera una secuencia de *N* dígitos. Dos jugadores juegan alternativamente. En su turno, cada jugador puede efectuar dos tipos de movimientos:

a) Cambiar un dígito impar por otro menor. P.e. en la secuencia 9735478 podemos cambiar 5 por 4, 3, 2, 1 ó 0. En concreto, si cambiamos 5 por 2 quedaría 9732478.

b) Si uno de los dígitos es par ó 0, se puede eliminar, eliminando al mismo tiempo todos los dígitos situados a su izquierda. P.e., si en la secuencia 85972046 eliminamos el 2, queda la secuencia 046.

Pierde el jugador que elimina el último dígito.

Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

14.– **Juego**. Se considera un tablero cuadrado de dimensión *N*. Dos jugadores juegan alternativamente a poner fichas de dimensiones *2 x 1* en el tablero. Cada uno dispone de $N^2/2$ fichas. En su turno, cada jugador pone una de sus fichas ocupando dos casillas contiguas libres. Uno de los jugadores, por ejemplo el jugador A, puede poner las fichas sólo en horizontal, y el otro jugador, p.e. jugador B, sólo puede poner fichas en vertical. Pierde el jugador que no puede poner ficha en su turno.

Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

15.– **Juego**. La *Devoradora* del tío Antonio aprende rápido, y tras varios días de competir por la comida con la *Listilla*, ha cambiado de estrategia y ahora las dos se han convertido en jugadoras "*de la leche*", y buscan como único fin comer más que la otra, de hecho las dos puede que utilicen la misma estrategia. El juego consiste en que como antes cada una en su turno elige un cubo de uno de los extremos, se lo come y es retirado, ganando la que come más. Diseñe un algoritmo que para una posición del mismo evalúe el juego e indique la jugada a realizar.

16.– **El juego de las reinas**. Darío, "el presi", entre sus múltiples actividades sociales dentro de la prisión, se ha inventado un nuevo juego con el que está intentando ganar vales de tabaco a sus compañeros de reclusión. El juego que Darío se ha inventado se basa en el problema de las *n* damas de ajedrez que deben ser colocadas en un tablero de *n x n* celdas sin que se ataquen entre ellas, sabiendo que la dama del ajedrez ataca a todas las piezas que se encuentren en su misma fila, columna o alguna de las dos diagonales que pasan por la casilla en la que se encuentra la dama.

En el juego cada jugador, va colocando una dama, y sólo una, en su turno, siempre que la dama colocada no ataque a ninguna de las damas anteriormente colocadas. Aquel jugador que no puede colocar una dama, pierde el juego. Construya un algoritmo que dada una configuración del juego la evalúe, y devuelva la decisión que produce el resultado indicado.

17.– **Juego. Dominó, DOS jugadores**. Se considera el juego del dominó con dos jugadores a los que previamente se les han distribuido todas las fichas. Construya un algoritmo que dada una configuración del juego la evalúe, y devuelva la decisión que produce el resultado indicado.

18.–**Juego**. Los *Pintados* y los *Dibujados* están pugnando por conseguir el mayor número de escaños en las próximas elecciones comarcales. Los estrategas electorales de cada grupo han recopilado datos sobre cada uno de los pueblos de la comarca y han estimado los votos que son capaces de conseguir según los mítines que van a dar sus máximos dirigentes (solo darán mítines los máximos dirigentes, uno por cada partido) en cada uno de los pueblos en campaña electoral.

Así, para cada pueblo disponen del poder de convicción, en número de votos, de cada uno de sus dos máximos dirigentes. Han realizado la distinción para cada pueblo en si son los primeros en dar el mitin o son los segundos, y también han determinado que un mismo candidato no va a dar dos mítines en un pueblo, porque entonces los electores se "saturan" y "pasan" de votar a dicho candidato.

Puesto que los dirigentes sufren un gran desgaste en cada uno de los mítines que dan, solo van a dar un mitin cada día, y además, van a tomarse un día de descanso entre cada mitin. Así, han suscrito un documento, remitido a la Junta Electoral Central en el que se comprometen a dar mítines en días alternos: un día dará un mitin un candidato y al siguiente lo dará el otro candidato.

Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar. Gana el que consiga más votos.

19.– **Juego. Cuatro en Raya**. Disponemos de un tablero vertical de dimensiones *N x N*, Construya un algoritmo que dada una configuración del juego la evalúe, y devuelva la decisión que produce el resultado indicado.

Por ejemplo, si tenemos la posición

o				x
x	x			o
x	o	o	x	o

Posibles continuaciones son:

o	x			x
x	x			o
x	o	o	x	o

O bien

o				x
x	x		x	o
x	o	o	x	o

NOTA: En este caso se trata de ejemplos de jugadas, no de la respuesta que debiera dar el algoritmo desarrollado.

20.– **Juego.** *Jug Ador* y *Lud Opata* han inventado un nuevo juego con el que pasar entretenidos las tardes de verano. El juego se juega sobre un tablero cuadrado de *NxN* casillas en las que inicialmente hay colocadas una serie de fichas de distintos colores. Cada color tiene un valor asignado y el objetivo del juego es conseguir la mayor cantidad de puntos.

El juego se desarrolla de la siguiente manera: *Jug* y *Lud* cogen una ficha alternativamente, de tal manera que al coger una ficha, cogen también las fichas de sus casillas adyacentes. El juego termina cuando no quedan más fichas en el tablero y gana aquel jugador que ha conseguido reunir el mayor número de puntos.

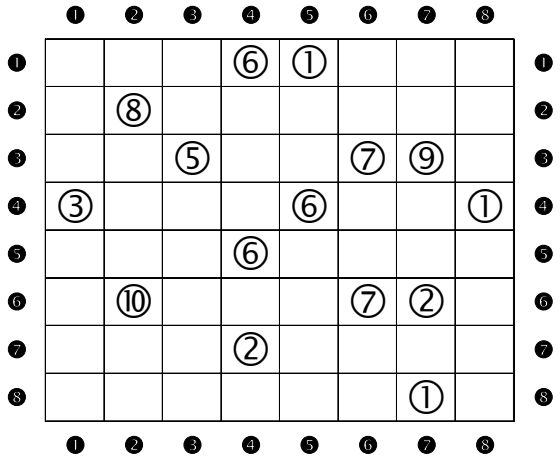
Por ejemplo, dada la posición

az	p	a	na	b
b	m	v	v	a
o	az	na	r	a
o	r	v		z
f	z	p	rs	o

Al coger la ficha de la casilla m tomaríamos las de su alrededor. Si tomáramos la de la casilla rs habría una casilla que no tendría nada, y tomaríamos todas las restantes.

Diseñe un algoritmo que dada una posición del juego, la evalúe, y devuelva la jugada que hace llegar a esa situación.

21.–**Juego GravidBall**. El ***GravidBall*** es un juego de tablero para dos jugadores. En este juego, los dos jugadores juegan alternativamente pulsando uno de los botones del tablero. En cada turno, un jugador tiene la obligación de pulsar un botón y sólo puede pulsar un botón por turno. El tablero de juego es una rejilla de *NxN* casillas en las que hay unas bolas con una puntuación. En cada casilla hay a lo sumo una bola. El tablero tiene alrededor un conjunto de botones, cada uno de los cuales está asignado a una fila o una columna. Cuando un jugador acciona un botón, todas las bolas de esa fila o columna caen a un depósito y se suma la puntuación de las bolas al jugador que pulsó el botón. El juego termina cuando no quedan bolas en el tablero y ganará aquel jugador que consiga la mayor puntuación.



Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

22.– **Juego**: En un grafo, cada vértice tiene un cierto valor, el botín. Dos ladrones, partiendo del mismo vértice y de forma alternativa, podrán ir a cualquiera de los vértices adyacentes que aún tengan botín. Si alguno no pudiera moverse, por no tener botín ninguno de sus adyacentes, pasaría el turno al otro. El juego acaba cuando ninguno de los jugadores pueda moverse. Ganará el que más dinero consiga.

23.– **Juego**. Se tiene un grafo dirigido valorado, que representa un mapa de carreteras, en el que el peso de los arcos es la distancia entre ciudades. Dos coches salen de una ciudad *A* y deben llegar a otra *B*. Cada vez avanza uno de ellos y atraviesa uno de los tramos (arcos), siempre que no haya pasado ya por él ninguno de los dos coches. Gana el que llegue a *B* recorriendo el menor número de kilómetros.

24.– **Juego de los Buhoneros**. Dos buhoneros se han apostado sus carromatos a ver quien vende más botellas del milagroso bálsamo del Dr. *Bestfit*. La comarca sobre la que han hecho la apuesta tiene un conjunto de aldeas, unidas por caminos angostos y pedregosos que bordean las montañas. Disponen del mapa con las aldeas y los caminos que los comunican, con el coste de desplazamiento entre dos aldeas contiguas, que es el peaje (monetario) a pagar a *ABACO* (*Asociación de BAndidos de la COmarca*). De la misma forma, y teniendo en cuenta la población de cada aldea, y el nivel de convicción que tiene cada uno sobre los vecinos de

cada aldea, saben cuantas botellas venderá cada uno en una aldea (suele ser distinta para cada uno).

Por experiencia saben que no deben visitar dos veces la misma aldea, ni siquiera pasar por ella, porque si los aldeanos no perciben una mejoría en sus males, la emprenden a pedradas con el buhonero y su carromato, destrozándolo.

El mecanismo del juego es el que sigue: *Pluma de Águila* (puesto que lleva una en su sombrero) visita una aldea un día, y *Pico de Oro* descansa, y el día siguiente, es *Pico de Oro* el que visita otra aldea, y *Pluma de Águila* descansa, y así sucesivamente, hasta que ambos no tienen posibilidades de ganar más dinero.

Puesto que hay dificultades para saber cuando no deben pasar por una aldea (entre las "*virtudes*" del bálsamo milagroso no se encuentra la de la comunicación telepática), han acordado jugar sobre el mapa de la comarca que tienen, ya que en el mismo se recogen todos los datos necesarios para jugar.

Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

25.– **Juego de los Buhoneros 2.0**. "***El hombre es el único animal que tropieza dos veces en la misma piedra (o con la misma botella)".*** Dos buhoneros se han apostado sus carromatos a ver quien vende más botellas del milagroso bálsamo del Dr. *Bestfit*. La comarca sobre la que han hecho la apuesta tiene un conjunto de aldeas, unidas por caminos angostos y pedregosos que bordean las montañas. Disponen del mapa con las aldeas y los caminos que los comunican, con el coste de desplazamiento entre dos aldeas contiguas, que es el peaje (monetario) a pagar a *ABACO* (*Asociación de BAndidos de la COmarca*). De la misma forma, y teniendo en cuenta la población de cada aldea, y el nivel de convicción que tiene cada uno sobre los vecinos de cada aldea, saben cuantas botellas venderá cada uno en una aldea (suele ser distinta para cada uno).

El sentido común les dice que no deberían pasar dos veces por la misma aldea, ni siquiera pasar por ella, porque si los aldeanos no perciben una mejoría en sus males, deberían emprenderla a pedradas con el buhonero y su carromato, destrozándolo. Pero lo que su experiencia les ha enseñado es que pueden pasar hasta dos veces por una aldea, aunque la segunda vez, deben hacer un descuento del 50% en la mercancía que venden (curiosamente, venden el mismo número de botellas, y a los mismos que la vez anterior, porque "el hombre es el único animal que tropieza..."). Así pues, la segunda vez que pasan por una aldea el beneficio que obtienen es la mitad del de la primera vez.

El mecanismo del juego es el que sigue: *Pluma de Águila* (puesto que lleva una en su sombrero) visita una aldea un día, y *Pico de Oro* descansa, y el día siguiente, es *Pico de Oro* el que visita otra aldea, y *Pluma de Águila* descansa, y así sucesivamente, hasta que ambos no tienen posibilidades de ganar más dinero.

Puesto que hay dificultades para saber cuando no deben pasar por una aldea (entre las "*virtudes*" del bálsamo milagroso no se encuentra la de la comunicación telepática), han acordado jugar sobre el mapa de la comarca que tienen, ya que en el mismo se recogen todos los datos necesarios para jugar.

Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

26.– **Juego.** *Santa* y *Los Magos* se han retado estas navidades a ver quien reparte más ilusión entre los niños (de entre 0 y 500 años). Puesto que no pueden hacerlo en la realidad, puesto que trabajan en fechas distintas, van a hacer una simulación mediante un juego.

En contexto en el que se desarrolla el juego es que tenemos *N* casas, habitadas cada una por una familia, a cuyos miembros tienen que repartir un conjunto de los juguetes (o cumplir con sus deseos) que han pedido, lo que les proporciona cierta felicidad (a todos los niños de la casa, de entre 0 y 500 años), no necesariamente la misma para todas las casas. Naturalmente, cumplir con los deseos de los niños (o repartirles los juguetes) requiere un esfuerzo. Puesto que *Santa* y *Los Magos* pueden emplear diferentes estrategias (y por tanto tener costes distintos) con los moradores de una casa, la felicidad que consiguen puede ser distinta. Si bien supondremos que siempre emplean la estrategia que proporciona una mayor felicidad para los moradores (si hay que hacer algo, mejor hacerlo bien). Así, se sabe que para una casa *c*, la felicidad que reparte *Santa* es FS_c, con un coste CS_c, y la felicidad que reparten *Los Magos* es FM_c con un coste CM_c. Además, al principio, *Santa* tiene una fuerza *FS* y *Los Magos* otra *FM*, para repartir juguetes.

Aunque se sabe que ni *Santa* ni *Los Magos* tienen ninguna restricción para desplazarse de un lugar a otro, para hacer el juego más interesante, se han impuesto la restricción de que sólo visitarán casas adyacentes a la que se encuentre en un momento dado, según un mapa que muestra las conexiones entre las casas, y que de entre las casas adyacentes, sólo pueden visitar aquellas que ninguno ya haya visitado anteriormente.

El juego se desarrolla considerando que los movimientos entre las casas se realizan de forma alternativa entre los jugadores, esto es, un jugador hace un movimiento (visita una casa), y el otro espera, y a continuación éste realiza un movimiento, y el otro espera, y así sucesivamente. Siempre y cuando un jugador pueda moverse, porque en otro caso, pasa el turno. **Gana el que consigue repartir una mayor felicidad**.

Construya un algoritmo que dada una configuración del juego, la evalúe, y devuelva la decisión que produce el resultado indicado.

27.– **Juego**. Para cada Escuela de Informática se sabe el número de alumnos que finalizan los estudios este año. Varias empresas necesitan personal para desarrollar sus proyectos informáticos y han contratado a dos empresas cazatalentos (ECT) rivales para que en un plazo de a lo sumo *K* días les proporcionen personal. Las ECT competirán por captar el mayor número de recién titulados para sus clientes.

Saben el porcentaje de alumnos, sobre los recién titulados, que son capaces de captar tras sus contactos con el alumnado de cada escuela. Este porcentaje no es el mismo para cada Escuela, ni tampoco para cada ECT. Por ejemplo, la ECT A puede captar el 80% de una Escuela, en tanto que a la ECT B tiene un 60% de captación en la misma.

Los Directores Generales de las ECT se han propuesto deslumbrar a sus empresas clientes con su trabajo para demostrarles lo engrasada que está su maquinaria de captación de materia gris. Y han planteado la captación como un juego con una serie de reglas, expuestas a sus clientes. Han dibujado sobre un mapa las posiciones de las Escuelas con las relaciones de proximidad entre ellas. Una ECT que está en una Escuela puede ir a otra Escuela adyacente, que no haya sido visitada anteriormente. Ganará el primero que consiga el número de

titulados exigido en los *K* días. Se supone que la captación en cada Escuela tarda un día. Diseñe una función que dada una posición del juego la evalúe y devuelva la decisión a tomar.

28.– **Juego**. Jason (Viernes 13) y Freddy Krugger (Pesadilla en Elm Street), se han retado a ver quién consigue más víctimas. Pero para darle una versión más académica, se habían propuesto realizar sus fechorías en el campus de la Universidad, concretamente en sus Escuelas y Facultades, y como posibles víctimas los pringaos de turno que se quedan hasta altas horas de la noche trabajando en dichos centros (a partir de ahora, el trabajo del pringao, además de poco agradecido, sería altamente "peligroso"...).

Sabían el número *N* de centros del campus y habían acordado visitar un centro en cada turno, con tal de que no haya sido visitado anteriormente por uno cualquiera de los dos.

Debido a los diferentes métodos empleados con sus víctimas, la "efectividad" de cada uno es distinta para cada Escuela. Así, para el centro *i*, el número de víctimas de Jasón es J_i, y el de Freddy F_i, no teniendo porqué coincidir.

Construya un algoritmo que dada una configuración del juego la evalúe, y devuelva la decisión que produce el resultado indicado.

29.– **Juego**. Jason (Viernes 13) y Freddy Krugger (Pesadilla en Elm Street), se han retado a ver quién consigue más víctimas. Pero para darle una versión más académica, se habían propuesto realizar sus fechorías en el *Campus de la Universidad*, concretamente en sus Escuelas y Facultades, y como posibles víctimas los pringaos de turno que se quedan hasta altas horas de la noche trabajando en dichos centros (a partir de ahora, el trabajo del pringao, además de poco agradecido, sería altamente "peligroso"...).

Sabían el número *N* de centros del campus y habían acordado visitar un centro en cada turno, con tal de que no haya sido visitado anteriormente por uno cualquiera de los dos.

Debido a los diferentes métodos empleados con sus víctimas, la "efectividad" de cada uno es distinta para cada Escuela. Así, para el centro *i*, el número de víctimas de Jasón es J_i, y el de Freddy F_i, no teniendo porqué coincidir.

Además, con motivo del esfuerzo para desplazarse entre los centros, su efectividad para cada centro se reduce según van recorriendo más distancia. Así, tenemos que para centro *c*, la efectividad de Jasón tras haber recorrido una distancia *d* es $J_c/j(d)$ ($F_c/k(d)$ para Krugger, respectivamente) siendo $j(d)$ la función para Jasón (resp. $k(d)$ para Krugger).

Construya un algoritmo que dada una configuración del juego la evalúe, y devuelva la decisión que produce el resultado indicado.

30.– **Juego entre dos personas.**

Tenemos un tablero cuadrado en el que en determinadas casillas hay fichas. En cada jugada una persona retira una o varias fichas de una columna o de una fila no pudiendo dejar en

medio de las fichas que retira otras fichas. Siempre se ha de retirar al menos una ficha. Los jugadores se van alternando y gana quien coge la última ficha.

Construya un procedimiento que dada una determinada posición del juego determine si ésta es ganadora o perdedora, y si es ganadora la decisión que nos lleva a ganar.

Recuérdese que:

– Una posición es **ganadora** si puede dejarle a su contrincante una posición perdedora.

– Una posición es **perdedora** si no es ganadora, esto es, si cualquier posición que puede dejarle a su adversario es ganadora.

Ejemplos:

1) - - -
x x -
- x x

Posición Ganadora: tomamos las fichas de la segunda columna.

2) x x
x x

Posición Perdedora: siempre dejamos una continuación Ganadora.

3) x - -
x - -
- x x

Posición Perdedora: lo mismo que en 2.

4) x x -
- - -
- - -

Posición Ganadora: tomamos todas las fichas.

5) x x x
x - x
x x x

Una **jugada ilegal** es tomar en la fila 1 sólo las fichas de las columnas 1 y 3, pues está entre ellas otra ficha. Por contra sería **legal** tomar en la segunda fila las dos fichas.

31.– **Juego**. Dos jugadores juegan al siguiente juego. Cada jugador tiene un cartón formado por huecos y operadores (suma, resta, multiplicación y división) de forma alternativa. El cartón tiene un número par de huecos *H* y *H-1* operadores. Cada jugador puede ir rellenando los huecos del cartón, de izquierda a derecha, con diferentes números comprendidos entre *1* y

H/2. Además cada jugador no puede jugar dos veces el mismo número. Cuando un jugador coloca un número N comprendido entre *1* y *H/2*, al rival se le coloca de forma automática el valor *(H/2 + 1) – N* en su mismo hueco del cartón.

Gana el juego aquel que tras evaluar el cartón, el resultado de las operaciones (sin tener en cuenta la precedencia) se queda más cercano de un valor V dado.

Diseñe una función que dada una posición del juego la evalúe, y devuelva la decisión a tomar.

Ejemplo:

Cartón:		+		*		-		*		+		-		/		*		+	

H = 10, V = 25

Valores posibles que se pueden jugar [1-5]

Comienza el jugador 1, y gana el jugador que se queda más cerca de *V = 25*.

El desarrollo del juego podría ser:

J1 -> 3, J2 -> 5, J1 -> 5, J2 -> 1, J1 -> 1, J2 -> 3, J1 -> 2, J2 -> 4, J1 -> 4, J2 -> 4.

3	+	1	*	5	-	5	*	1	+	3	-	2	/	4	*	4	+	2	=18
3	+	5	*	1	-	1	*	5	+	3	-	4	/	2	*	2	+	4	=38

Jugador 1 -> Resultado = 18 -> |25-18| = 7 --------> GANADOR
Jugador 2 -> Resultado = 38 -> |25-38| = 13

Otro ejemplo:

5	+	3	*	1	-	4	*	4	+	2	-	2	/	5	*	3	+	1	=10,6
1	+	3	*	5	-	2	*	2	+	4	-	4	/	5	*	3	+	1	=22,6

Jugador 1 -> Resultado = 10,6 -> |25-10,6| = 14,4
Jugador 2 -> Resultado = 22,6 -> |25-22,6| = 2,4 --------> GANADOR