COLECCIÓN DE PROBLEMAS

DE

Algorítmica

Para la evaluación contínua

SECCIONES

I. DIVIDE Y VENCERÁS
II. VORACES
III. VUELTA ATRÁS y RAMIFICACIÓN Y PODA
IV. MISCELÁNEA

Los problemas que vienen a continuación aparecen en una sola sección, pero eso no significa que la única forma de resolverlos, ni tan siquiera la mejor, sea la que corresponde al tema en que aparecen incluidos. El propósito de su ubicación es que no sea inmediato asignar un esquema para su resolución, ¡ es necesario pensar!

I. DIVIDE Y VENCERÁS

I.1. Diseñar un algorítmo de búsqueda ternaria en un vector. La idea es partir el vector en 3 partes y activar recursivamente la función de búsqueda en cada una de las 3 partes. Analizar el costo de algorítmo en comparación con el de búsqueda binaria.

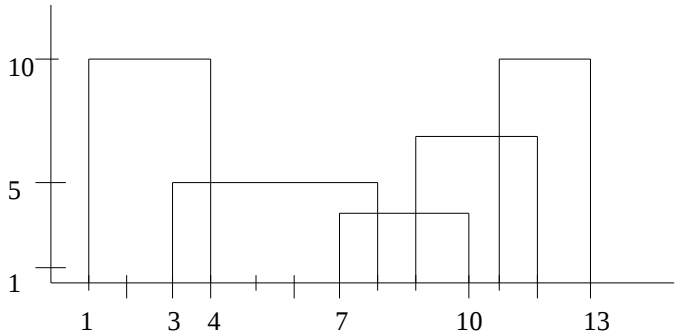
I.2. Dados n valores reales ($v_1,\ldots,v_n$) diseñar un algorítmo que calcule el valor de la diferencia *d* entre los dos valores má cercanos:

$$d = \min_{1 \le i, j \le n,\ i \ne j} | v_i - v_j |$$

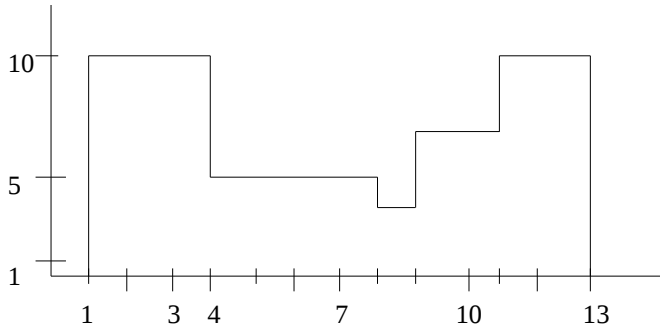
Sugerencia : Usar el algorítmo de partición del quicksort.

I.3. Sea a[1..*n*], $n \ge 1$, un vector de enteros diferentes y ordenados crecientemente, tal que algunos de los valores pueden ser negativos. Diseñar un algoritmo que devuelva un índice natural *k*, $1 \le k \le n$, tal que a[*k*]=*k*, siempre que tal índice exista. El coste del algoritmo debe ser O(*log n*) en el caso peor.
Se pide diseñar el algoritmo dando todo lujo de detalles. Justificar su corrección y el coste.
Repetir el problema pero para un vector de *n* enteros posiblemente repetidos y ordenado decrecientemente.

I.4. Dada la localización exacta y la altura de varios edificios rectangulares de la ciudad, obtener la *skyline*, linea de recorte contra el cielo, que producen todos los edificios eliminando las líneas ocultas. *Ejemplo* : La entrada es una secuencia de edificios y un edificio se caracteriza por una tripleta de valores (*xmin,xmax,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 :

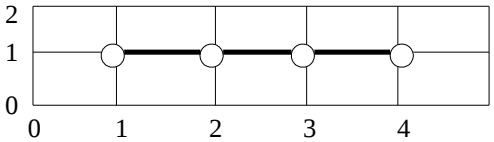


Su secuencia asociada es : (1,4,10), (4,8,5),(8,9,3), (9,11,7) y (11,13,10)

I.5. Se está jugando un torneo entre *n* jugadores. Cada jugador juega una vez contra los *n-1* jugadores restantes (es un *round-robin*). No hay empates y los resultados de los partidos se anotan en una matriz. En general no se pueden ordenar los jugadores porque falla la transitividad, es decir, *A* le gana a *B*, *B* le gana a *C* pero *C* le gana a *A*. Estamos interesados en un orden más débil que definimos de la siguiente forma : en la secuencia ordenada que se obtiene *P1*,*P2*,..,*Pn* se cumple que *P1* le ha ganado a *P2*, *P2* le ha ganado a *P3*, etc. y *Pn-1* le ha ganado a *Pn*. Diseñar un algoritmo que produzca esta secuencia ordenada con un coste de O(*n.log n*). La entrada del algoritmo es la matriz de resultados y el coste de acceder a una posición de la matriz es constante.

I.6. Sea *T* un árbol binario completo de altura *h* y $n=2^h-1$ vértices. Queremos encajar el árbol *T* en un plano de la siguiente manera : A cada vértice le corresponde un único punto en el plano, vértices adyacentes están conectados por lineas rectas (paralelas al eje *x* o paralelas al eje *y*) y no pueden intersectar dos líneas. El problema general es encontrar el rectángulo de área mínima que contenga el árbol. Por ejemplo : un grafo de *k* vértices (*k*=4 en la figura) que forma una secuencia :


, podría ser encajado en un rectángulo de área 2(*k*+1). Ver en la figura el encaje con *k*=4.



Encajar grafos en el plano de esta forma es un problema importante en el diseño de chips, sobre todo en VLSI. Diseñar un algoritmo que te indique como encajar el árbol *T* en el plano de modo que éste ocupe un área de orden O(*n*). No se pide el de área mínima.

I.7. Diseñar un algoritmo eficiente para el problema de la *selección múltiple* : Dados un vector A[1..*n*] de elementos, *n*>0, y un vector *j*[1..*p*] de enteros, *p*>0, donde $1 \le j[1] < j[2] < ... < j[p] \le n$, hay que hallar los elementos *j*[1]-*ésimo, j*[2]-*ésimo, ..., j*[*p*]-*ésimo* del vector A si éste estuviera ordenado. La solución propuesta debe ser más "eficiente" que la evidente de ordenar el vector A o que la de iterar un algoritmo de selección *p* veces, una vez con cada valor *j*[*i*] dado. No obstante, no se pide que se demuestre que el algoritmo propuesto es efectivamente más eficiente que las dos soluciones triviales mencionadas ni que se calcule su coste (es un problema matemáticamente complejo).
Ejemplo :

El vector A contiene los siguientes elementos A[1]=8, A[2]=14, A[3]=5, A[4]=7, A[5]=3, A[6]=1, A[7]=25 y A[8]=2.
El vector j, qué está ordenado crecientemente, contiene los índices de los elementos de A que se quieren obtener si A estuviera ordenado. Así, *j*[1]=2, *j*[2]=5 y *j*[3]=7.
El resultado nos ha de dar el valor de A que ocuparía la posición 2 si A estuviera ordenado, o sea Res[1]=2, y el valor de A que ocuparía la posición 5 si A estuviera ordenado, o sea Res[2]=7 y el valor de A que ocuparía la posición 7 si A estuviera ordenado, o sea Res[3]=14.

I.8. El algoritmo de ordenación rápida, *quicksort*, no es demasiado eficaz para ordenar cadenas, ya que la comparación entre dos cadenas es una operación costosa (proporcional a la longitud de las cadenas comparadas, en caso peor). Peor aún, los intercambios son todavía más costosos. Por ello se pide que se desarrolle una variante de quicksort, *vqs*, que ordene eficazmente *n* cadenas.
Para empezar, el vector que reciba como entrada *vqs* no será un vector de *n* cadenas, sino un vector de *n* apuntadores al primer carácter de cada cadena. Un símbolo especial, *FDC*, marcará el fin de una cadena. Este símbolo es menor que cualquier otro carácter en el orden alfabético. La notación A[*i*][*j*] permite acceder al *j*-ésimo carácter de la cadena *i*-ésima en el caso en que la longitud de la cadena sea inferior a *j*, mientras que A[*i*][*j*]=*FDC* si *j* es la longitud de la cadena y estará indefinido en otro caso. Como pista, el procedimiento a diseñar, una vez hecha la inmersión de parámetros, tiene la siguiente especificación :

```
// Pre: ( 0≤ i, j ≤n-1 ) ∧ ( i+1< j ) ∧ todas las cadenas en A[i..j] tienen un prefijo común de
// longitud k
void vqs ( char* A[ ], int i, int j, int k )
// Post : A[i..j] está ordenado crecientemente según el orden alfabético
```

La llamada inicial sería entonces vqs(A, 0, n-1, 0).
Es necesario que vuestro algoritmo sea muy detallado y que se diseñen todas las funciones auxiliares que se usen.

II. VORACES

En todos los ejercicios que siguen definir con precisión cual es el criterio de selección, diseñar el algoritmo voraz y demostrar la optimalidad de las soluciones obtenidas.

II.1. Deseamos almacenar en una cinta magnética de longitud L un conjunto de n programas $\{P_1, P_2, \ldots, P_n\}$. Sabemos que cada P_i necesita un espacio a_i de la cinta y que

$$\left(\sum_{1 \le i \le n} a_i\right) > L$$

Construir un algoritmo que seleccione aquel subconjunto de programas que hace que el número de programas almacenado en la cinta sea máximo.

II.2. Diseñar un algoritmo para almacenar N programas de longitud l_i, $1 \le l_i \le H$ en una cinta magnética de forma que el tiempo medio de lectura de los mismos sea mínimo. Se supone que los programas se leen con igual frecuencia y que antes de leer cada programa se rebobina la cinta. Se entiende por tiempo medio de lectura:

$$T = (1/N) \sum_{k=1}^{N} \sum_{i=1}^{k} l_i$$

II.3. Un *vertex cover* (recubrimiento de vértices) de un grafo no dirigido $G=(V,E)$ es un conjunto de vértices U, $U \subseteq V$, tal que cada arista del grafo G incide en, como mínimo, un vértice de U. Diseñar un algoritmo voraz que calcule el *vertex cover* de tamaño mínimo, si es que eso es posible. Comprobar si la estrategia elegida conduce siempre al óptimo. Repetir el mismo proceso pero para grafos que sean árboles.

II.4. Se han de procesar n tareas. Cada tarea i tiene un instante de terminacion t_i y un valor $v_i \ge 0$ si se procesa en dicho instante o antes. Se dispone de un sólo procesador que necesita sólo una unidad de tiempo para ejecutar cada tarea. Una solución factible es una secuencia S de tareas que cumplen las restricciones de terminación. Una solucion óptima es aquella que maximiza V tal que :

$$V = \sum_{s \in S} v_s$$

Diseñar un algoritmo voraz para encontrar la secuencia S.

II.5. Se han de procesar n tareas con instantes de terminación t_i y tiempo de proceso p_i . Se dispone de un procesador y las tareas no son interrumpibles. Una solución factible planifica todas las tareas de modo que terminan en, o antes de, su instante asignado. Diseñar un algoritmo voraz para encontrar la solución. Demostrar que, si existe solución, el algoritmo voraz la encuentra.

II.6. Dado un conjunto de n cintas cuyo contenido está ordenado y con n_i registros cada una, se han de mezclar por pares hasta lograr una única cinta ordenada. La secuencia en que se

haga la mezcla determina la eficiencia del proceso. Diseñar un algoritmo voraz que minimice el número de movimientos.

Ejemplo : se han de mezclar 3 cintas: *A* con 30 registros, *B* con 20 y *C* con 10.

Opción 1 : - Mezclar *A* con *B* requiere 50 movimientos (se obtiene *A'*)
- Mezclar *A'* con *C* requiere 60 movimientos
TOTAL = 110 movimientos

Opción 2 : - Mezclar *C* con *B* requiere 30 movimientos (se obtiene *A'*)
- Mezclar *A'* con *A* requiere 60 movimientos
TOTAL = 90 movimientos.

II.7. Un automovilista ha de ir con su vehículo desde la población *A* hasta la población *B* siguiendo una ruta prefijada (por ejemplo la carretera N-340). Con el depósito completamente lleno puede hacer *X* km. El conductor conoce de antemano en qué lugares de la carretera existen gasolineras (la distancia entre dos gasolineras consecutivas es inferior a *X* km.). Diseñar un algoritmo que permita que el automovilista vaya de *A* a *B* repostando el *mínimo* número de veces posible (se supone que parte de *A* con el depósito lleno y que el coche no se averiará, ni tendrá un accidente, ni será abducido, etc. durante el trayecto). Indicar el coste del algoritmo y demostrar la optimalidad del criterio usado.

II.8. El informático megalómano comienza a estar harto de tener que esperar tanto rato para poder escuchar sus canciones favoritas en su extensa colección de casettes. Como no tiene suficiente dinero para comprarse un grabador de CD's ni tampoco quiere desperdiciar cinta ha decidido diseñar un algoritmo que dadas *n* canciones, sus respectivas duraciones en segundos (dur[*i*] es la duración de la canción *i*-ésima, $1 \leq i \leq n$), y unas "frecuencias" de acceso (f[*i*] es la frecuencia de acceso a la canción *i*-ésima), encuentre la disposición de las *n* canciones que minimiza el tiempo medio de acceso. Es decir, el resultado del algoritmo es un vector pos que representa a una permutación de las *n* canciones, de tal modo que si *pos*[*j*] es la canción que ocupa la posición *j*-ésima en la cinta entonces

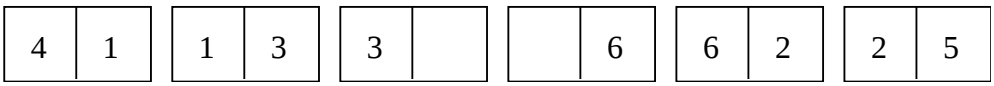
$$\sum_{j \leq 1 \leq n} f[pos[j]] \quad \sum_{k \leq 1 \leq j-1} dur[pos[k]]$$

es mínima.
Nuestro informático megalómano no ha hecho IEA y ya empieza a desesperar. Hay que echarle una mano!

II.9. La jefatura de estudios de la BIF encara el siguiente problema cada cuatrimestre : una vez fijados los horarios de las *n* clases a impartir (una clase es una tripleta <*día_semana*,*hora_inicio*,*hora_fin*>) debe asignarse un aula a cada clase. El único requisito para que la asignación sea válida es que no puede haber dos clases el mismo día y a la misma hora que compartan aula. Hallar un algoritmo eficiente (polinómico en *n*) que encuentre una asignación válida de aulas a las clases de modo que se emplee el mínimo número de aulas posible.
Suponemos que la *hora_inicio* y la *hora_fin* de una clase son horas en punto o horas y media (ej: 10:30-12:30) y que una clase puede durar como mínimo media hora y no más de 4 horas. Ninguna clase acaba en un día diferente al día en que se inicia.

III. VUELTA ATRÁS y RAMIFICACIÓN Y PODA

III.1. El juego del dominó tiene 28 fichas diferentes. Las fichas son rectangulares y tienen grabado en cada extremo un número del 0 al 6. Siguiendo las reglas del juego, las fichas se colocan formando una cadena de tal manera que cada pareja de fichas consecutivas tienen el mismo número en los extremos que se tocan. Un ejemplo de cadena sería:



Diseñar un algorítmo que imprima todas las cadenas correctas de longitud 28 que se pueden formar con las 28 fichas del juego.

III.2. Un cuadrado latino es un tablero de nxn posiciones, cada una de las cuales posee un color escogido entre n colores diferentes. Ha de cumplir la restricción de que no puede haber colores repetidos en ninguna de sus filas ni en ninguna de sus columnas. Un ejemplo con n=4 es:

A	B	C	D
D	C	A	B
C	D	B	A
B	A	D	C

Diseñar un algoritmo que imprima todos los posibles cuadrados latinos de tamaño nxn.

III.3. Diez factorías de tecnología punta, recién creadas en la República de Fanfanisflan, requieren una instalación informática y personal que la dirija. El Gobierno Supremo de la Nación les ha asignado diez máquinas y diez técnicos recién licenciados.
Pero no todas las máquinas son apropiadas para las necesidades de todas las empresas. Se dispone de una función booleana apropiada f*(m,e)* que indica si la máquina *m* es apropiada para la empresa *e*. Asimismo, no todos los técnicos conocen todas las máquinas, por lo que disponemos de una función booleana conoce g(*t,m)* que indica si el técnico *t* conoce la máquina *m*. Finalmente, no todos los técnicos aceptan trabajar en todas las empresas, debido a que algunas de ellas se relacionan con la industria bélica; disponemos de la previsible función booleana aceptaria h*(t,e)* que indica si el técnico *t* aceptaría trabajar en la empresa *e*.
Se pide un programa capaz de encontrar una manera de asignar a cada empresa una máquina apropiada a sus necesidades, y un técnico que conozca la máquina y que acepte trabajar en la empresa, si es que tal asignación existe; y lo indique adecuadamente si tal asignación no existe.

III.4. 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. Diseñar un algoritmo que calcule todas las maneras posibles de obtener *7 y medio*.

Nota : Todas las cartas puntuan por su valor excepto el 10,11 y 12 que sólo valen medio punto.

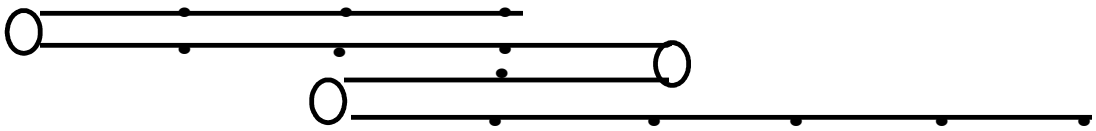
III.5. La editorial *Tothop Ubliquem* está preparando una nueva reedición de las obras completas del prolífico escritor *Vaes Criuremassa*. Como es bien sabido, este escritor únicamente cultivó un género literario, el ensayo, y sus obras son todas de reducidas dimensiones: poco más o menos, cada una de sus obras ocupa *p* páginas.
En sus agudos escritos, este erudito ensayista desarrollaba siempre sorprendentes y curiosas relaciones entre temas aparentemente diversos. Es famoso, por ejemplo, su ensayo *El mito de Dione*, que muestra cómo la astrología medieval tiene sus orígenes en las costumbres de los navegantes fenicios y en la fauna de los desiertos africanos, y explica asimismo cómo influye después tanto en la pintura flamenca como en determinadas concepciones políticas del liberalismo de principios del siglo XX.

La editorial en cuestión se enfrenta al siguiente problema. La publicación habrá de consistir en varios volúmenes, lujosamente encuadernados para que queden bonitos en las bibliotecas de sedicentes intelectuales. Para que no sean demasiado gruesos, es preciso que todos los volúmenes contengan a lo más *j* ensayos.
Previas ediciones de estos escritos, organizadas cronológicamente o a veces sin criterio alguno, han resultado poco rentables. Con el fin de publicar una edición novedosa, la editorial quiere organizar los volúmenes de acuerdo con criterios temáticos, de manera que cada volumen esté dedicado a un tema. Es decir, todos los ensayos qe se publiquen en el mismo volumen han de tener al menos un tema en común, que es al que se dedica el volumen.

Se dispone de algo de software ya escrito para este proyecto. En concreto, existen módulos que implementan los tipos *obra*, *tema* y *lista_de_temas*, con las operaciones apropiadas para su manejo. Además existe una estructura de datos, trabajosamente recopilada, que soporta la operación *temas_de_que_trata*: dada una obra, proporciona la lista de los temas que se tratan en esa obra. Por ejemplo, la expresión :
temas_de_que_trata("El mito de Dione") devuelve la lista [astrología_medieval, costumbres_fenicias, fauna_africana, pintura_flamenca, liberalismo_del_siglo_XX].
Se pide:
(a) Aplicando el esquema de Ramificación y Poda, desarrollar un algoritmo capaz de encontrar una solución para el problema de editar las obras completas de *Vaes Criuremassa* que minimice el número de volúmenes.
(b) Resolver el mismo problema mediante un algoritmo basado en un esquema diferente. Exprimirse la mollera en busca de ideas chulas (NO vale un Vuelta Atrás ciego y sin marcaje).

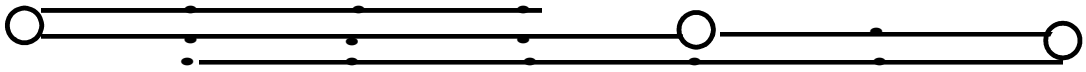
III.6. Disponemos de una regla de carpintero un tanto inusual. Está compuesta por una serie de segmentos consecutivos y articulados y cada uno de ellos tiene una longitud asociada que no tiene porqué ser siempre la misma. Se trata de diseñar un algoritmo eficiente que indique cómo hay que plegar la regla para que se minimice la longitud que ocupa una vez plegada. Justificad el esquema elegido.

Ejemplo clarificador : Supongamos que la regla se compone de 4 segmentos de longitudes 3,4,2,5 en esta secuencia que es inalterable. Una forma de plegarlo bastante intuitiva sería la que se muestra en la figura que viene a continuación:



Como puede apreciarse, la longitud que ocupa plegada es 7 y la solución sería {[s1,izq], [s2,der], [s3,izq], [s4,der]}

Sin embargo, plegándola como se indica en la siguiente figura se consigue una longitud de 6, que es la mínima, y que tiene como solución {[s1,izq], [s2,der], [s3,der], [s4,izq]}.



III.7. Vamos a construir un circuito eléctrico usando como soporte una placa de dimensiones AxB (podemos pensar en la placa como si fuera una cuadrícula o una matriz con A filas y B columnas). Hemos de colocar en ella N componentes electrónicos (caben todos y cada uno ocupa una única posición) y disponemos de las siguientes matrices de datos para facilitar su colocación :
- Matriz CON (conexiones) : CON*(i,j)* contiene el número de conexiones que han de existir entre la componente *i* y la componente *j*.
- Matriz DIS (distancias) : DIS*(r,s)* contiene la distancia que existe entre la posición *r* y la posición *s* de la placa.

Diseñar un algoritmo que encuentre una asignación de componentes a posiciones tal que minimice la longitud del cable empleado. Justificar la eleccción del esquema elegido. La longitud del cable empleado es la suma de los productos CON*(i,j)*xDIS*(r,s)* donde la componente *i* ocupa la posición *r* y la componente *j* ocupa la *s*.

III.8. Una matriz booleana M[1..*n*,1..*n*] representa un laberinto cuadrado. A partir de una casilla dada se puede llegar a las casillas adyacentes de la misma línea o columna. Además, si M[*i*,*j*]=FALSO, no se puede pasar por esa casilla; si M[*i*,*j*]=CIERTO se puede pasar.
a/ Diseñar un algoritmo que encuentre un camino, si hay alguno, de la casilla (1,1) a la (*n*,*n*).
b/ Modificar la solución anterior, si es necesario, para que en el caso de que exista camino encuentre el más corto.

III.9. En pleno mes de Agosto se nos ha estropeado un congelador repleto de tarrinas de helados de diferentes sabores. Formalizando la situación: tenemos N tarrinas, una tabla T[1..*N*] de naturales que nos indica el tiempo que tarda en derretirse cada una de ellas y otra tabla Sabor[1..*N*], también de naturales, que nos indica el sabor asociado.
Además disponemos de la función placer*(i)*, que cuantifica el placer que nos produce comer un helado de sabor *i* , y con la función compatible*(i,j)* que será cierta si y sólo si podemos comer una tarrina de sabor *j* inmediatamente después de haber ingerido una de sabor *i*.
Suponemos que necesitamos una unidad de tiempo para devorar una tarrina y que empezamos a comer en el instante en que se estropea el congelador.

Hay que decidir la secuencia de helados que hemos de comer para que se maximice el placer obtenido. Hay que tener en cuenta que un helado no puede comerse si ya se ha derretido.

III.10. Disponemos de *N* vasos diferentes con los que queremos construir una torre de modo que ésta tenga altura mínima. La torre se consigue apilando los *N* vasos invertidos, es decir, la abertura (boca) hacia abajo y el pie hacia arriba. Nos ofrecen las siguientes operaciones :

longitud : vaso ---> real /* devuelve la altura del vaso */

índice : vaso, torre_vasos ---> real /* Devuelve un valor entre 0 y 100 que es el % de la altura de *torre_vasos* que queda dentro de *vaso* al intentar apilar *vaso* sobre *torre_vasos* */

Diseñar un algoritmo, justificando el esquema utilizado, que encuentre la solución del problema planteado.

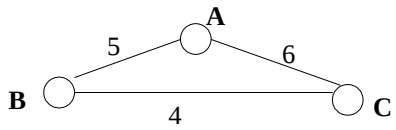
III.11. Sea $G=(V,E)$ un grafo no dirigido, completo y etiquetado con $n=|V|$. Las etiquetas pertenecen a los naturales. Se dispone de la operación etiq : *vértice*, *vértice* ---> *nat*, que dados dos vértices del grafo devuelve un natural que es la etiqueta de la arista. Se trata de generar una *secuencia*, que denominaremos *S*, con todos los vértices del grafo tal que sea mínima la siguiente suma de productos :

$$[\text{MIN}] \sum |i - j| \,.\, \text{etiq}(S(i), S(j))$$

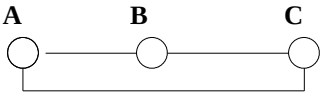
$$\forall i, j : 1 \leq i < j \leq n$$

donde $S(i)$ es el vértice que ocupa la posición i en la secuencia devuelta. Diseñar un algoritmo que calcule la secuencia de suma mínima. Calcular su coste y justificar la corrección de la solución propuesta.

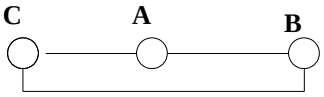
Ejemplo. Dado el grafo completo de grado 2 que viene a continuación :



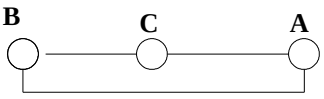
Las tres posibles sumas diferentes son :



SUMA = 5+4+(2.6)= 21



SUMA = 6+5+(2.4) = 19



SUMA = 4+6+(2.5) = 20

De las 3 sumas distintas la solución sería la segunda que es la de suma mínima.

III.12. Dados 2 grafos no dirigidos, $G=(V,E)$ y $H=(N,A)$, con idéntico número de vértices escribir un algoritmo, dando todo lujo de detalles, que determine si *G* y *H* son *isomorfos* o no lo son. Determinar el coste del algoritmo y justificar el esquema que se ha utilizado en su resolución.

Se dice que dos grafos son *isomorfos* sii :

- a cada vértice $u \in V$ le corresponde un único vértice $\prod(u) \in N$ y a la inversa,
- a cada arista $(u,v) \in E$ le corresponde una única arista $(\prod(u),\prod(v)) \in A$ tal que $\prod(u)$ es el vértice que le corresponde a u y $\prod(v)$ es el que le corresponde a *IV*.

Dicho de otro modo, dos grafos son *isomorfos* si existe una permutación $\prod: V \rightarrow N$ tal que $(u,v) \in E \Leftrightarrow (\prod(u),\prod(v)) \in A$.

III.13. Una compañía telefónica ha recibido el encargo de ubicar cabinas telefónicas en el recién construído campus de Chiquitistán. Todos los posibles puntos de ubicación para las cabinas y las distancias entre ellos se conocen; también se sabe cuál es el coste de instalar una cabina. Toda esta información está disponible en una matriz *dist*, y un vector *cost*:
dist[*i, j*] = distancia (> 0) entre las ubicaciones *i* y *j*, $1 \le i \le j \le n$;
cost[*i*] = coste de instalar una cabina en el punto *i*, $1 \le i \le n$.
El contrato establece que desde cualquier punto (de los *n* identificados) hasta la cabina más proxima no puede haber una distancia superior a *D*. A la compañía telefónica le interesa hallar las ubicaciones en las cuáles colocar cabinas de manera que se cumplan las condiciones del contrato, pero que minimice los costes de instalación, pues no confía en la rentabilidad de la operación, que sólo aceptó por una cuestión de imagen. Diseñad un algoritmo que solucione el problema.

III.14. *Problema de la reconstrucción de distancias.* Se nos da un conjunto de $N = n(n-1)/2$ valores que representan a las distancias entre *n* puntos ubicados sobre una línea. El problema consiste en hallar los valores $x_1, x_2, ..., x_n$ tales que dan origen a las *N* distancias dadas o bien determinar que no es factible colocar los *n* puntos sobre la recta de manera que las distancias mutuas sean las dadas. Se puede suponer sin pérdida de generalidad que $x_1 = 0$. Por ejemplo, si *D*={ 2, 4, 5, 6, 7, 11 } (*N* = 6, *n* = 4) una solución posible es $x_1=0$, $x_2=5$, $x_3=7$ y $x_4=11$.

III.15. Tenemos un sistema constituído por *n* dispositivos conectados en serie, de manera que el dispositivo D_i se conecta al dispositivo D_{i+1}, $1 \le i < n$. Sea f_i $(0<f_i \le 1)$ la fiabilidad del dispositivo D_i (es decir, la probabilidad de que funcione correctamente). Entonces la fiabilidad del sistema es

$$f = \prod_{1 \le i \le n} f_i$$

Aunque cada dispositivo sea muy fiable, la fiabilidad del sistema puede ser bastante menor (p.e. si $n=10$ y $f_i = 0.99$ para $1 \le i \le n$, tenemos $f \approx 0.904$). Para aumentar la fiabilidad decidimos modificar el sistema de modo que la fase *i*-ésima está compuesta por $m_i > 1$ réplicas del dispositivo D_i (en el sistema original $m_i = 1$). La fiabilidad de la fase *i* viene dada por $\phi_i(m_i)$, donde las funciones ϕ_i satisfacen las siguientes propiedades : 1) $\phi_i(1) = f_i$; 2) $\phi_i(m) < \phi_i(m')$ si $m < m'$; 3) $\lim_{m \to \infty} \phi_i(m) = 1$. Ahora la fiabilidad del sistema es

$$f = \prod_{1 \le i \le n} \phi_i(m_i)$$

De la propiedad (3) de las ϕ_i se desprende que podemos acercar la fiabilidad de la fase *i* a 1 tanto como queramos (y otro tanto podemos decir de *f*), colocando más y más réplicas de D_i.

Pero la vida no es tan fácil … Cada dispositivo D_i nos cuesta c_i euros y el coste total del sistema no puede superar E euros, por lo que tendremos que decidir cuidadosamente cuántos dispositivos ponemos en cada fase. Supondremos que $E \geq c_1+\ldots+c_n$. Además, por razones diversas (*stock* disponible, solvencia del fabricante, etc.), puede estar limitado el número de unidades del dispositivo i que se pueden usar.

Diseñad un algoritmo que dados dos vectores $C[1..n]$ y $F[1..n]$ con los costes y fiabilidades de los dispositivos, un valor real $E>0$, un vector $B[1..n]$ con el máximo de unidades disponibles de cada dispositivo y una función **phi**(i,F,m) que calcula $\phi_i(m)$, halle el vector $m[1..n]$ tal que la fiabilidad del sistema es máxima sin que su coste exceda el valor E y dentro de los límites de suministros representados por el vector B. ($B[i]=+\infty$ si tenemos un suministro potencialmente ilimitado de unidades del dispositivo i).

IV. MISCELÁNEA

IV.1. Tenemos un tablero de NxN casillas y una ficha situada en la esquina superior izquierda, sobre la casilla *(1,1).* Se pretende llevar la ficha a la esquina opuesta, es decir la *(N,N).* Los únicos movimientos posibles son desplazar la ficha una casilla hacia abajo o una casilla hacia la derecha. En el tablero hay casillas normales y casillas especiales. Que la ficha pase por una casilla normal cuesta 1 y que pase por una casilla especial tiene coste 2. Se sabe que hay *S* casillas especiales repartidas por el tablero.
Prononer un algoritmo que encuentre un camino de coste mínimo entre las dos casillas fijadas. Justificar el esquema elegido así como su coste y todas las decisones tomadas.

IV.2. Disponemos de un juego de fichas de dominó con las tradicionales 28 fichas. Supongamos que nos entregan N fichas al azar tal que $2<N<28$ y que de esas N fichas se fijan dos cualesquiera (ficha origen y ficha destino), colocadas de una cierta manera, como extremos de una cadena.
El problema que se nos plantea es el siguiente : Devolver una secuencia de fichas de longitud mínima tal que permita pasar de la ficha origen a la destino satisfaciendo las reglas del dominó para formar la secuencia. Si la secuencia no existe devolver la secuencia vacía.
Diseñar una algoritmo para resolver el problema indicando, además, qué esquemas podían aplicarse, cuál es el elegido y el motivo de su elección.

IV.3. Sea a un vector , $a[1..n]$, de elementos con $n>0$. Se dice que un vector es mayoritario si tiene un elemento mayoritario y se dice que un elemento x es mayoritario en el vector a cuando más de la mitad de los elementos del vector son iguales a x. Formalmente, si existe un x tal que $(\mathrm{N}i : 1\le i\le n : a[i]=x)>n/2$, entonces x es elemento mayoritario y, por tanto, el vector es mayoritario. Como es evidente, si el vector es mayoritario, sólo existe en él un único elemento mayoritario.

(a) Supongamos que existe una relación de orden entre los elementos que figuran en el vector. Queremos determinar si un vector dado es mayoritario o no y, caso de que lo sea, saber cuál es ese elemento. ¿Qué algoritmo, de los vistos en clase, habría que utilizar para conseguir resolver este problema con un coste $O(n)$?. Identificar claramente el algoritmo final y justificar la respuesta indicando, por ejemplo, porqué hace falta que exista una relación de orden entre los elementos.

(b) Supongamos ahora que ya no tenemos una relación de orden entre los elementos y que la única operación disponible para comparar los elementos del vector es la igualdad. Diseñar un algoritmo que, con coste $O(n \cdot log\ n)$, resuelva el mismo problema que se plantea en a/ (Obviamente no se puede ordenar el vector a). Indicar el esquema empleado, justificar la corrección y determinar su coste.

IV.4. Sea S una secuencia de naturales no repetidos tal que $longitud(S)=n$ con $n\ge 0$. Se pide diseñar un algoritmo que calcule la tira creciente de naturales, contenida en S, de mayor longitud. Una tira es una secuencia formada por elementos que aparecen en la secuencia de

entrada tal que se mantiene el orden de aparición pero no la posición en la que aparecen. *Ejemplo* : sea *S*={7,2,1,9,5,6,11,14,8} la secuencia de entrada. Algunas de las tiras crecientes que se pueden obtener a partir de ella son :
T1={7,9,11,14}, T2={2,9,11,14}, T3={2,5,6,11,14}, T4={2,5,6,8}, T5={1,9,11,14}, T6={1,5,6,11,14} siendo, en este caso, T3 y T6 las de mayor longitud (5 elementos) de todas las posibles.
Para resolver este problema se pueden utilizar las operaciones típicas sobre secuencias. Identificar y justificar claramente el esquema elegido. Introducir todas las mejoras de eficiencia que sean posibles. Explicar todas las operaciones, con código incluido, si es necesario, así como todas las estructuras de datos utilizadas.

IV.5. Se ha de organizar el horario 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. Suponiendo que *n* es potencia de 2, diseñar e implementar un algoritmo que construya el horario y permita terminar el campeonato en *n-1* días. Indicar el esquema utilizado. Analizar el coste del algoritmo.

IV.6. Diseñad un algoritmo que determine si un grafo no dirigido y conexo *G* contiene un *cuadrado* como subgrafo (*i.e.* determinad si *G* contiene al menos un ciclo de longitud 4). Se valorará positivamente que la solución propuesta tenga coste $O(n \cdot e)$, donde *n* es el número de vértices de *G* y *e* es el número de aristas, si bien se considerarán aceptables soluciones cuyo coste sea más elevado en caso peor.
Pista (para obtener una solución "astuta"): Pensad en primer lugar el algoritmo suponiendo que el grafo *G* viene dado a través de su matriz de adyacencia. Aplicad una idea similar sobre un grafo implementado con listas de adyacencia para obtener una versión más eficiente.

IV.7. Dado un conjunto de probabilidades de acceso $\{p_1, \ldots, p_n\}$ a un conjunto de elementos $X=\{x_1<x_2<\ldots<x_n\}$ podemos construir un árbol binario de búsqueda T^* para X tal que el coste medio de búsqueda :

$$C(T^*) = 1 + \sum_{1 \le i \le n} p_i \cdot \text{profundidad}(x_i, T^*),$$

sea mínimo, empleando el esquema de programación dinámica. Es decir, los elementos más accedidos se encuentran a menor profundidad que los elementos poco accedidos, de tal modo que si multiplicamos lo que nos cuesta acceder a x_i en T^* por su probabilidad de acceso p_i y sumamos para toda *i*, la mejor estructuración posible es el árbol T^*.
Sin embargo, el coste del algoritmo de programación dinámica es $\theta(n^2)$ (tanto en tiempo como en espacio), y preferiríamos un algoritmo de coste $o(n^2)$ aunque el árbol construido no sea necesariamente óptimo. Eso sí, el algoritmo debería tener en consideración los pesos : no vale cosntruir un árbol que sea perfecta o casi perfectamente equilibrado independientemente de los p_i's. Diseña al menos un algoritmo con estas características (eficiencia, suboptimalidad, dependencia del conjunto de probabilidades). Analiza su eficiencia, justifica que efectivamente construye(n) árboles binarios de búsqueda y razona el porqué de la "heurística" en la que se basa(n). Es decir, aporta alguna buena razón para suponer que en general se generarán árboles con costes medios de acceso cercanos al óptimo.

IV.8. Se denomina *arbitraje* al uso de las discrepancias en las tasas de cambio de divisas para obtener un beneficio. Por ejemplo, durante un corto espacio de tiempo puede suceder que un euro (1 €) valga 0.95 dólares, un dólar valga 0.75 libras esterlinas y una libra esterlina valga 1.45 €. Entonces partiendo de 1 € podemos terminar con :

1 € = 1 € x (0.95 US$ / 1 €) x (0.75 £ / 1 US$) x (1.45 euro / 1 £) = 1.033125 €,

con un beneficio neto de algo más del 3.3%. Suponed que hay n tipos distintos de moneda y que se dispone de una tabla C[1..n,1..n] donde C[i,j] es la tasa de cambio de (una unidad de) la divisa i a la divisa j. Por ejemplo si el euro es la divisa 1 y la libra esterlina la divisa 2 entonces C[2,1]=1.45. Se cumple que C[i,j]=1/C[j,i] para toda i y j y que C[i,i]=1. Diseñad un algoritmo de programación dinámica que dada la tabla C y el identificador i de una divisa, $1 \leq i \leq n$, nos proporcione el valor del mejor arbitraje, donde el mejor arbitraje es una secuencia $i_1, i_2, \ldots, i_k$ de identificadores distintos entre sí y distintos de i tal que su valor $C[i,i_1]\cdot C[i_1,i_2]\cdot\ldots\cdot C[i_{k-1},i_k]\cdot C[i_k,i]$ es máximo. Analizad el coste del algoritmo.

IV.9. Dados un vector A[1..n] de elementos (n>0) y un vector ω[1..n] de pesos asociados (números reales positivos), que satisface $\sum_{1\leq i\leq n} \omega[i] = 1$, la *mediana ponderada de* A es el mayor elemento x = A[j] tal que

$$\sum_{A[l]<x} \omega[l] < \tfrac{1}{2} \quad \text{y} \quad \sum_{A[r]\geq x} \omega[r] \geq \tfrac{1}{2}$$

es decir, es el mayor elemento x de A tal que la suma de los pesos asociados a los elementos menores que x es < ½ y la suma de los pesos asociados a los elementos mayores o iguales que x es ≥ ½. Escribid el pseudocódigo de un algoritmo eficiente (al menos, su coste en caso medio tendría que ser inferior a $n \log n$) para hallar la mediana ponderada de A y analizad el coste.