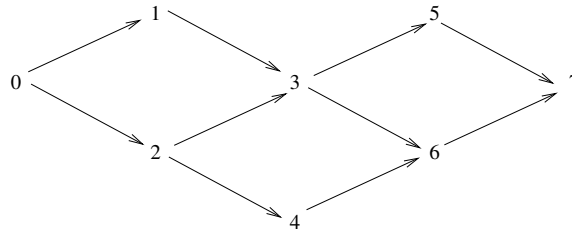
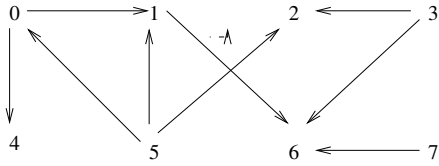


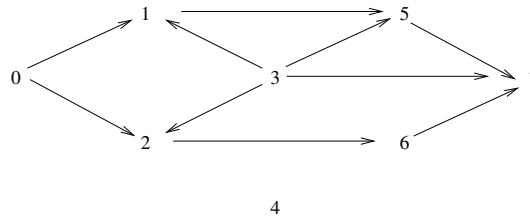
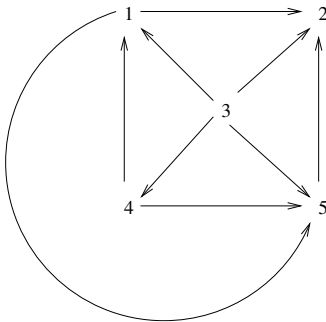
Problemas

Representación de grafos

- E** Para los siguientes grafos, dar su representación mediante una matriz de adyacencia y una lista de adyacencia.



- E** Para los siguientes grafos, dar su representación mediante una lista de adyacencia y dar también la evolución sobre los mismos del algoritmo de distancias mínimas en grafos no ponderados.



- E** Junto a la lista de adyacencia de un grafo $G = (V, R)$ se puede definir su lista de incidencia, en la que $L[i]$ apunta a los vértices v_j tales que $(v_j, v_i) \in R$. Escribir las listas de incidencia de los grafos del problema anterior. Definir de manera análoga la matriz de incidencia y calcularla en los mismos ejemplos.
- P**– Escribir el psc de una función que reciba la lista de adyacencia de un grafo y devuelva su lista de incidencia.
- P**– Escribir el psc de una función que reciba la matriz de adyacencia de un grafo y devuelva su matriz de incidencia.
- P** La *incidencia* de un vértice se define como el número de ramas que llegan a él. Escribir el pseudocódigo de una rutina que calcule la tabla $inc[]$ de incidencias en los vértices de un grafo a partir de la representación de un grafo en una lista de adyacencia.
- P** Demostrar que en un grafo no dirigido $G = (V, R)$ se cumple que $\sum_{u \in V} inc(u) = 2|R|$. Deducir que un grafo no dirigido tiene un número par de vértices de incidencia impar.

veces es un número par.

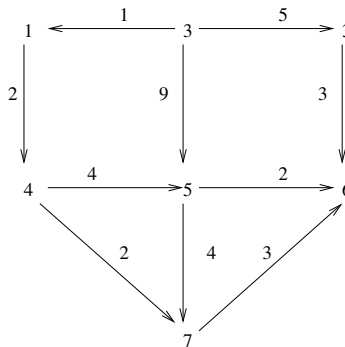
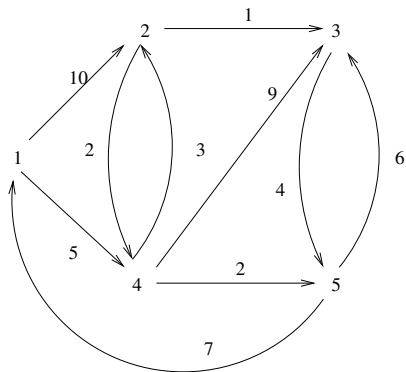
9. **E+** ¿Se pueden conectar entre sí 99 teléfonos de tal manera que cada uno se conecte exactamente con otros 11?
10. **E** ¿Puede un país del que salen 7 carreteras de cada ciudad tener exactamente 100 carreteras?
11. **P** El complemento de un grafo $G = (V, R)$ es un grafo $G' = (V, R')$ en el que $(u, v) \in R'$ sii $(u, v) \notin R$. Dar el pseudocódigo de una función que reciba la lista de adyacencia de un grafo y devuelva la de su complemento.
12. **P** Dar el pseudocódigo de una función que reciba la matriz de adyacencia de un grafo y devuelva la de su complemento.

Distancias mínimas y árboles abarcadores mínimos

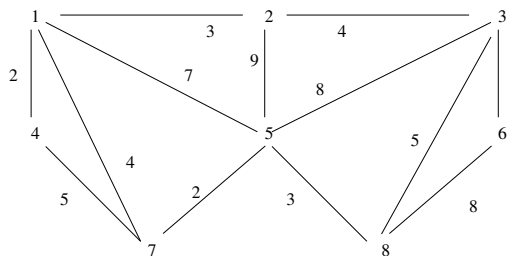
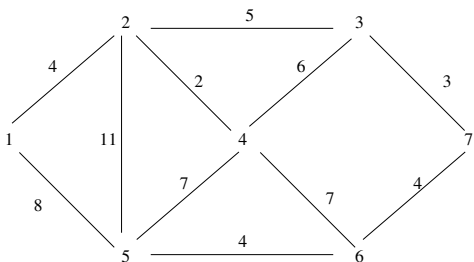
13. **P** ¿Cuál sería la eficacia del algoritmo de distancias mínimas en grafos no ponderados si se utilizara una matriz de adyacencia?
14. **P** Dar el pseudocódigo de una función que calcule la tabla `dist[]` de distancias mínimas entre un vértice de un grafo a los demás a partir de la tabla `prev[]`.
15. **P+** La tabla `prev[]` proporcionada por los distintos algoritmos de caminos mínimos sirve para que, dado un vértice, encontrar todos los caminos mínimos a él desde otros vértices del grafo.
 1. Escribir una función recursiva de prototipo `Camino(vertice u, tabla p)` que reciba un vértice `u` y una tal tabla `p`, y proporcione los sucesivos vértices mediante los que se llega a `u` según un camino mínimo.
 2. Transformar dicha función eliminando sus posibles llamadas recursivas.
 3. ¿Cómo habría que modificar dicha función para que escribiera los vértices de dicho camino en su orden natural?
16. **P** ¿Es posible escribir el algoritmo de distancia mínima sobre grafos no ponderados sin utilizar las tablas `v[]` y/o `d[]`?
17. **P** ¿Es posible reescribir el algoritmo de Dijkstra sin utilizar la tabla `visto`? Escribir un nuevo psc de ser así.
18. **P** Dar un ejemplo concreto de un grafo ponderado con costes de ramas posiblemente negativos en el que el algoritmo de Dijkstra no proporcione caminos mínimos. (Evitar el ejemplo trivial de grafos con ciclos de ramas de coste negativo.)
19. **E** Dada la lista 1 15 2 14 3 13 4 12 5 11 6 10 7 9 8, construir el heap asociado a la misma, y dar su evolución cuando se extrae repetidamente su elemento mínimo.
20. **P+** (a) Suponiendo que en cada nodo de un heap se guarda un cierto identificador del elemento que lo ocupa, escribir el psc de una función que determine si un cierto elemento está o no en un heap, y si lo está, que dé su posición.
(b) Ampliar la función anterior para que, si su prioridad relativa lo justifica, elimine del heap la vieja versión del elemento en cuestión y reubique la nueva en su lugar adecuado.
21. **P** Si las tareas anteriores se recogen en una función `ReInsHeap(dato D, heap H)`, estimar el coste de la inserción y reubicación de un elemento en un heap mediante la misma.

múltiples reencoles (esto es, reemplazando AddPQ por ReInsHeap), y estimar su coste.

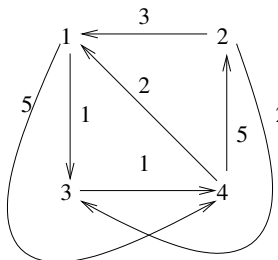
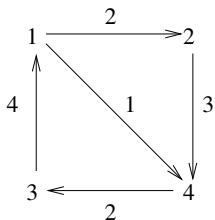
23. **E** Especificar mediante una tabla la evolución del algoritmo de Dijkstra de determinación de caminos mínimos a partir del vértice 1 en los grafos inferiores (usar múltiples reencoles de vértices si fuera necesario, y señalar en la cola de prioridad los costes asociados a los distintos vértices).



24. **P** Escribir el psc para el algoritmo de Prim modificando en forma adecuada el ya dado para Dijkstra.
25. **P** ¿Cómo se podría modificar el algoritmo de Prim para detectar la falta de conexión de un grafo?
26. **P** Dar el psc de un función que dé la lista de adyacencia de un AAm a partir de la tabla $p[]$ devuelta por el algoritmo de Prim.
27. **E** Dar sobre una tabla la evolución del algoritmo de Prim en los siguientes grafos.



28. **E** Encontrar las distancias mínimas entre todos los pares de vértices de los grafos inferiores:



29. **P** ¿Cuántas matrices deben de mantenerse en el algoritmo PD de determinación de distancias mínimas entre todos los pares de vértices de un grafo?

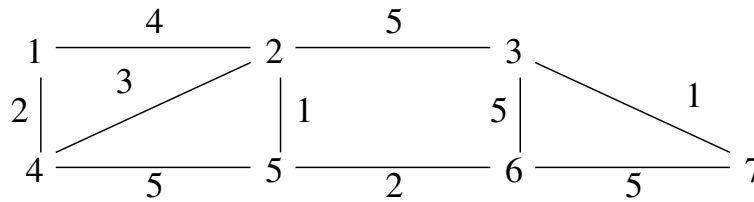
Algoritmo de Kruskal y TAD CD

31. **P** Escribir el pseudocódigo de una función de prototipo `Prev2AAM(vértices V, tabla prev)` que a partir del conjunto de vértices de un grafo y de la tabla `prev[]` devuelta por el algoritmo de Prim devuelva como la lista de adyacencia de un grafo el supuesto árbol abarcador mínimo que dicha tabla define. Prestar atención al control de errores, tanto interno al algoritmo, como de detección de errores en el árbol resultante.
32. **P** Una posible EdD para el TAD CD puede obtenerse almacenando los elementos de los subconjuntos en listas enlazadas. Para una mayor eficacia de la unión, dichas listas se acceden desde una tabla cuyos índices se identifican con los elementos del conjunto universal base U . En una entrada de dicha tabla de índice i se almacena
- un puntero al primer elemento de la lista donde se almacena el elemento i (dicho primer elemento es el representante del conjunto almacenado en la lista);
 - un puntero al último elemento de la última lista de la que i fue representante;
 - el tamaño de la última lista de la que i fue representante.

Dar el pseudocódigo de unas primitivas del TAD CD que utilicen la EdD recién mencionada.

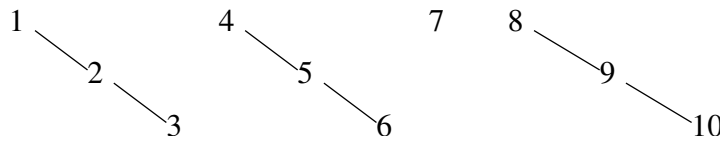
33. **P+** Si sobre la EdD anterior se efectúan siempre uniones por tamaños (esto es, la lista más corta se añade a la más larga), demostrar que el coste conjunto de todas las uniones posibles es $O(N \log N)$ con $N = |\mathcal{S}|$.
34. **P** Reaplicar el algoritmo de Kruskal a los distintos grafos no dirigidos de problemas anteriores indicando la evolución del CD a utilizar si el mismo se implementa mediante listas enlazadas.
35. **P** Modificar el pseudocódigo del algoritmo de Kruskal dado en clase para que detecte una posible falta de conexión en el grafo.
36. **E** Reaplicar el algoritmo de Kruskal a los distintos grafos no dirigidos de problemas anteriores representando en cada paso los conjuntos de vértices resultantes como árboles si se aplica unión por rangos y compresión de caminos.
37. **P** Si para el TAD CD se usa como EdD un bosque sobre tablas, dar el psc de las primitivas de dicho TAD.
38. **P** Una alternativa a la unión por rangos es la unión por tamaños: en cada unión, el árbol con menos nodos se añade al otro (en caso de empate se une el segundo al primero). Sobre el esquema en tabla $\mathcal{S}[]$ para representar los distintos conjuntos, ¿cómo se podría incorporar la información relativa al tamaño de los árboles?
39. **P+** Comprobar que empleando unión por tamaños también se cumple que para M finds y N uniones, se tiene $n(M, N) = O(N + M \log N)$.
40. **P** Justificar el que al aplicar unión por alturas la profundidad de un árbol de conjunto S cualquiera es $O(\lg |S|)$.
41. **P** Justificar el que al aplicar unión por rangos la profundidad de un árbol de conjunto S cualquiera es $O(\lg |S|)$.
42. **P+** ¿Cuál podría ser la función inversa de \lg^* ?

presión de caminos para conjuntos sobre listas enlazadas especificando en cada paso la evolución del bosque de árboles abarcadores parciales, y la de la tabla de punteros y listas enlazadas que recogen la gestión de los correspondientes conjuntos disjuntos.



44. **E** El bosque inferior representa el estado en un cierto momento de un conjunto disjunto. Indicar la evolución de dicho bosque tras efectuar las siguientes operaciones utilizando unión por rangos y compresión de caminos:

1. `union(find(2), find(6))`
2. `union(find(3), find(7))`
3. `union(find(6), find(10))`



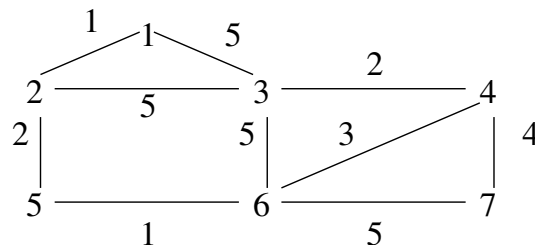
Representar el bosque inicial en una tabla y rehacer dicha evolución sobre la misma.

45. **E** Suponer que a partir de los conjuntos elementales formados por los 16 primeros elementos se efectúan finds y uniones de acuerdo al siguiente esquema:

1. se sitúan dichos elementos en una cola;
2. repetidamente y hasta llegar a un único conjunto, se extraen los dos primeros;
3. se unen;
4. se efectúa un find de un elemento de cada uno de ellos a profundidad máxima;
5. se reencola el resultado.

Realizar estas operaciones mediante unión por rangos y con y sin compresión de caminos, llevando la cuenta de cada acceso a nodo, y comparar los resultados.

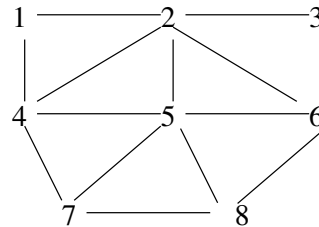
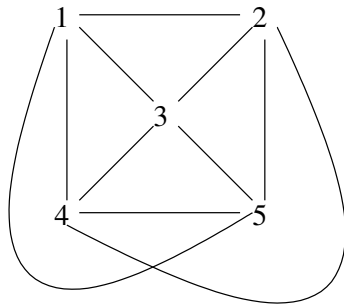
46. **E A.** Suponiendo una ordenación de ramas en la que las de igual coste se ordenan de manera lexicográfica, aplicar el algoritmo de Kruskal sobre el grafo inferior utilizando unión por tamaños y listas enlazadas para la gestión del conjunto disjunto de vértices y especificando en cada paso el estado de las mismas y el de la lista de ramas a añadir al árbol abarcador mínimo.



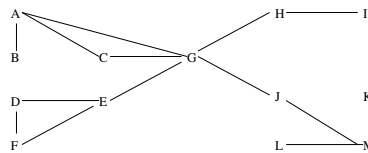
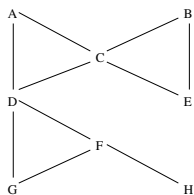
compresión de caminos, indicando sobre una tabla en cada paso el estado del conjunto disjunto de vértices.

Búsqueda en profundidad y conectividad

47. **P** Escribir el pseudocódigo de una función que a partir de la tabla `prev[]` devuelta por los algoritmos de búsqueda proporcione los correspondientes árboles.
48. **E** Escribir el pseudocódigo de una función driver para la búsqueda en profundidad que explore todos los vértices de un grafo dado.
49. **P** ¿Es posible detectar mediante búsqueda en profundidad las componentes conexas de un grafo no dirigido? Modificar para ello en forma adecuada el pseudocódigo de las rutinas a utilizar.
50. **P** ¿Cómo habría que modificar el algoritmo de búsqueda en profundidad para que proporcionara las ramas ascendentes y descendentes?
51. **P** ¿Cómo habría que modificar el algoritmo de búsqueda en profundidad para que proporcionara las ramas de cruce?
52. **P**– Modificar el algoritmo de búsqueda en profundidad para que asocie a cada vértice su numeración en preorden en el árbol de BP. ¿Es preciso mantener entonces la tabla `visto`?
53. **E** Dar la evolución del algoritmo de búsqueda en profundidad sobre los grafos no dirigidos inferiores, construyendo su árbol de búsqueda y marcando sobre él las ramas ascendentes.

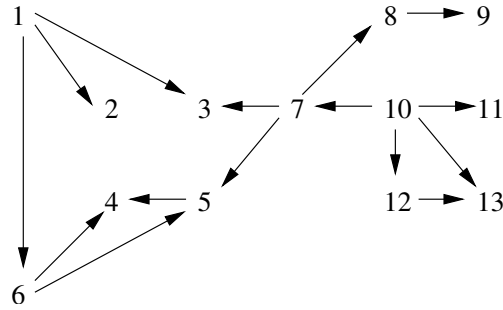
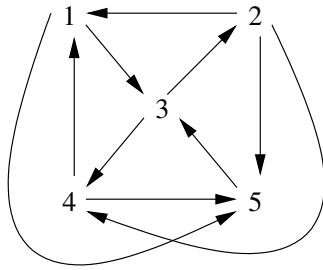


54. **E** Identificar los puntos de articulación de los grafos siguientes.

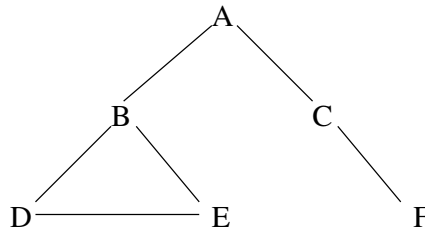


55. **E** La numeración en preorden de un árbol de BP marca el comienzo del proceso de BP de cada uno de sus nodos. En forma análoga podría computarse el tiempo `f[]` de finalización del proceso de cada vertice. Modificar el psc de BP para que proporcione también estos tiempos de finalización.

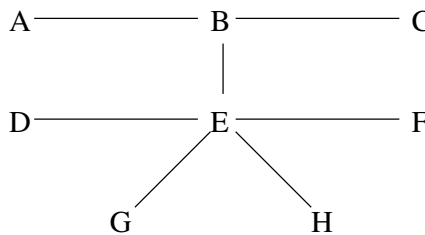
$G' = (V', R')$ de G donde $V' \subset V$, $R' \subset R$ y para todo par u, v de vértices de V' hay un camino de u a v cuyas ramas están contenidas en R' . Identificar por inspección las componentes fuertemente conexas de los siguientes grafos dirigidos.



57. **E** Encontrar razonadamente los puntos de articulación del grafo inferior. Para una mayor homogeneidad, usar listas de adyacencia ordenadas alfabéticamente, empezar el cálculo en el vértice A y representar los valores a calcular en una tabla. Aunque no se precisen en el cálculo, indicar adecuadamente las ramas ascendentes. Durante la ejecución de la búsqueda en profundidad indicar el comienzo y el retorno de las sucesivas llamadas recursivas, numerando su orden de ejecución.



58. **E** Encontrar razonadamente los puntos de articulación del grafo inferior. Para una mayor homogeneidad, usar listas de adyacencia ordenadas alfabéticamente, empezar el cálculo en el vértice A y representar los valores a calcular en una tabla. Aunque no se precisen en el cálculo, indicar adecuadamente las ramas ascendentes.



59. **E** El siguiente esquema recoge esencialmente el algoritmo de Tarjan para la determinación de componentes fuertemente conexas de un grafo dirigido:

CFC(grafo G)

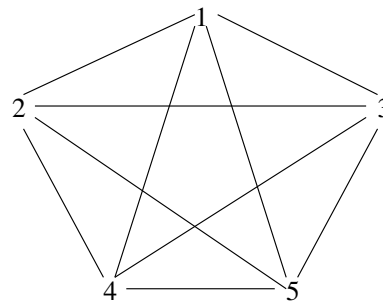
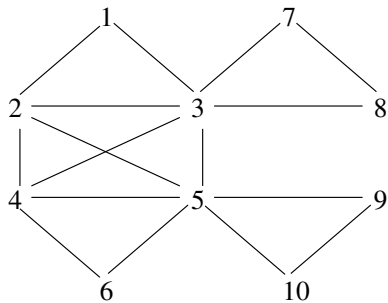
aplicar BP a G computando $f[]$;
 calcular el grafo traspuesto G' de G ;
 aplicar BP a G' en orden inverso al dado por $f[]$;

Comprobar sobre los grafos anteriores que este algoritmo proporciona efectivamente sus componentes fuertemente conexas.

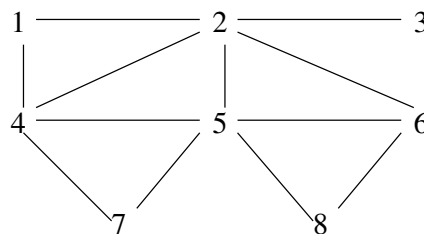
60. **P** Desarrollar en detalle el psc del algoritmo de Tarjan anterior.
61. **P+** Aplicar el algoritmo de búsqueda en profundidad a los grafos dirigidos acíclicos de hojas anteriores. A la vista de los correspondientes árboles de BP, ¿cómo se podría dar un algoritmo alternativo de ordenación topológica?

Circuitos eulerianos y hamiltonianos

62. **P** Se tiene un caballo de ajedrez en la esquina superior izquierda (ESI) de un tablero 3×3 . ¿Puede establecerse un camino que parta y vuelva a la ESI pasando por todas las demás casillas menos la central?
63. **E** Encontrar un circuito euleriano en los grafos inferiores.

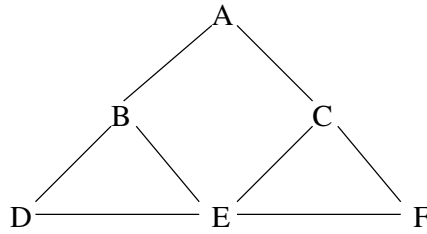


64. **E** Comprobar razonadamente para el grafo inferior la existencia de caminos eulerianos y encontrar razonadamente un tal camino a partir del posible punto de arranque de numeración más baja. Utilizar una tabla visto[u] [v] para indicar las ramas vistas, marcando con un 1 las vistas en el primer recorrido, con un 2 las vistas en el segundo, etcétera.

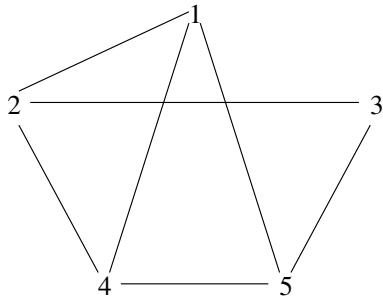


65. **P** Demostrar que un grafo no dirigido conexo G tiene un camino euleriano si y sólo si solamente tiene dos vértices de incidencia impar.
66. **P** Dar una rutina que determine si un grafo no dirigido conexo posee o no un circuito o un camino eulerianos. ¿Cuál sería su coste?
67. **P** Dar condiciones necesarias y suficientes para la existencia de caminos y circuitos eulerianos en multigrafos.

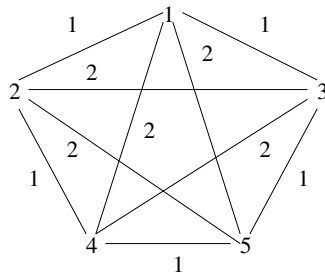
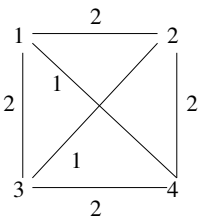
contrar razonadamente un tal camino a partir del posible punto de arranque de numeración más baja. Utilizar una tabla visto[u] [v] para indicar las ramas vistas, marcando con un 1 las vistas en el primer recorrido, con un 2 las vistas en el segundo, etcétera.



69. **P+** Dar el pseudocódigo de las rutinas necesarias para implementar de manera eficiente el algoritmo de determinación de circuitos eulerianos dado en clase. ¿Cuál sería su eficacia?
70. **P** ¿Cómo habría que modificar las rutinas anteriores para que detectaran errores debidos a la no existencia de circuitos eulerianos?
71. **E** En un árbol se duplica cada rama dando lugar a un multigrafo donde cada vértice tiene incidencia par (tal y como se hizo en el algoritmo aproximado para TSP). ¿Cuál podría ser un algoritmo simple que produjera un circuito euleriano?
72. **P** Comprobar que recorriendo en preorden un árbol abarcador mínimo de un grafo ponderado completo con función de coste verificando la desigualdad euclídea, se obtiene una solución a TSP de coste a lo sumo dos veces el óptimo.
73. **E** ¿Cuál podría ser la evolución de una búsqueda exhaustiva para la determinación de circuitos hamiltonianos en el grafo inferior?



74. **E** Dar una solución aproximada de TSP sobre los grafos inferiores.



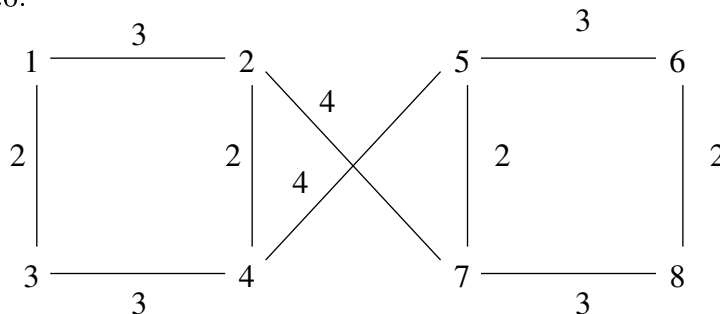
$\mathcal{S}_1 = \{ \text{ATG, TGG, TGC, GTG, GGC, GCA, GCG, CGT} \}$

$\mathcal{S}_2 = \{ \text{ATG, GGG, GGT, GTA, GTG, TAT, TGG} \}$

encontrar cadenas s_i cuyo espectro sea \mathcal{S}_i formulando el correspondiente problema como el de obtener un circuito hamiltoniano y encontrándolo por inspección.

76. **E** Encontrar cadenas s_i con los espectros del problema anterior formulando el correspondiente problema como el de obtener un circuito eurleriano y encontrándolo de manera adecuada.

77. **E** Encontrar razonadamente para el grafo inferior una solución aproximada al problema del viajante. El grafo inferior es completo, y el peso de las ramas no mostradas es 4, para así tener un grafo euclídeo.



78. **E** Dadas las cadenas

TTA, GAT, GTG, AGT

TAT, ATG, TGG, GTG, GGT

definir el grafo completo necesario para encontrar su supercadena más corta y encontrarla por inspección. Encontrar también su aproximación mediante el algoritmo codicioso.

79. **P+** Dada una cadena s obtenida como supercadena de serie de cadenas s_i , su compresión $c(s)$ se define como el número de caracteres ahorrados en s respecto a la cadena obtenida por simple concatenación de las cadenas s_i . Comprobar que si s_C es la supercadena obtenida mediante el algoritmo codicioso y s_O la cadena óptima, entonces $c(s_C) \geq 2c(s_O)$.

80. **P** La siguiente fórmula sugiere un método recursivo para calcular el producto $C = AB$ de dos matrices $2K \times 2K$ reduciéndolo al de 8 matrices $K \times K$:

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

Dar el psc de un tal algoritmo **MMR**, escribir una ecuación recurrente para su rendimiento y estimar su solución.

81. **P** El algoritmo de Strassen es una variante del algoritmo recursivo del problema anterior que consigue efectuar el producto de dos matrices $2K \times 2K$ reduciéndolo al de 7 matrices $K \times K$. Escribir una ecuación recurrente para su rendimiento y estimar su solución.
82. **P** ¿Cómo habría que modificar el algoritmo de Strassen para poder multiplicar matrices de tamaño $N \times N$ arbitrario realizando $\Theta(N^{\lg 7})$ productos?
83. **P** ¿Cómo se podrían multiplicar dos números complejos realizándose solamente tres multiplicaciones de números reales?
84. **P** Se ha demostrado que se pueden multiplicar dos matrices de tamaño 68×68 usando 132464 productos, dos matrices 70×70 realizando 143640 productos y 72×72 mediante 155424 productos. ¿Cuál de estos métodos produciría un rendimiento asintótico mejor? ¿Cómo se comparan con el método de Strassen?
85. **P** El profesor Tragacanto nos informa que ha descubierto una manera de multiplicar matrices 3×3 mediante un cierto número K de productos y que su algoritmo mejora el de Strassen. ¿Cuál sería el valor máximo de un tal K ?
86. **P** Suponiendo $N = 2^K$, un número A de N cifras puede descomponerse como $A = A_1 * D_K + A_2$, donde A_1, A_2 tienen $N/2$ cifras y $D_K = 10^{2^{K-1}}$. Si así se descomponen dos números A, B de N cifras, la fórmula

$$AB = A_1B_1 * D_K^2 + (A_1B_2 + A_2B_1) * D_K + A_2B_2$$

sugiere el siguiente algoritmo recursivo para multiplicar A y B :

```
int MMR(int A, int B, int N)
  si N==1:
    devolver A*B;
  else:
    calcular A1, A2, B1, B2;
    O = MMR(A1, B1);
    P = MMR(A1, B2) + MMR(A2, B1);
    Q = MMR(A2, B2);
    devolver (O * DK + P) * DK + Q;
```

donde DK denota $10^{2^{K-1}}$. Tomando como ob la multiplicación, ¿cuál es la complejidad $T_{MMR}(N)$ de aplicar MMR a dos números de $N = 2^K$ cifras?

$$AB = A_1B_1 * D_K^2 + ((A_1 + A_2)(B_1 + B_2) - A_1B_1 - A_2B_2) * D_K + A_2B_2$$

reduce la multiplicación de dos números de N cifras a esencialmente tres multiplicaciones de dos números de $N/2$ cifras. Dar el pseudocódigo de un algoritmo recursivo $MMRmR$ de multiplicación de números que aproveche dicha fórmula y estimar su complejidad $T_{MMRmR}(N)$ al aplicarlo a dos números de $N = 2^K$ cifras.

88. **P+** ¿Cuál es el coste en términos de productos realizados del algoritmo de cálculo de determinantes por menores? Escribir una recurrencia para el mismo y resolverla.
89. **P+** Demostrar que cualquier algoritmo de determinación del máximo de una tabla de N elementos mediante cdcs ha de efectuar como mínimo $N - 1$ cdcs. Concluir que el algoritmo habitual es óptimo.
90. **P** Un algoritmo para obtener el k -ésimo de una tabla podría consistir en ir leyendo los elementos manteniendo en cada momento en un max heap H los k elementos menores. De hacerlo así, el elemento buscado es el mayor elemento del heap H obtenido en el último paso. Desarrollar un tal algoritmo y estimar su coste.
91. **P+** Un algoritmo simple para la determinación simultánea del máximo y el mínimo de una tabla es aplicar los algoritmos habituales de obtención del máximo y del mínimo. ¿Es dicho algoritmo óptimo?
92. **P** ¿Cuántas cdcs son necesarias para calcular la mediana de 5 elementos?
93. **E** Dar la evolución para $K = 11$ del algoritmo `QuickSelect2` sobre la tabla 15 3 7 2 12 9 1 6 14 11 4 8 13 5 10.
94. **E** Indicar la evolución del algoritmo `QSelect` para la determinación del 9 elemento de la tabla [15 3 7 2 12 9 1 6 14] indicando los índices y posición a buscar en las sucesivas llamadas a `QSelect`, así como los resultados de las llamadas a las rutinas `Pivote5` y `Partir`.
95. **P+** ¿Cuál sería la eficacia en el caso peor de `QuickSort` si la selección del elemento medio se hiciera como en `QuickSelect2`?
96. **P** Demostrar que un polinomio de grado N queda totalmente determinado por sus valores en $N + 1$ puntos distintos.
97. **E** Los valores de un cierto polinomio cúbico $P(X)$ en los puntos 1, i , -1 y $-i$ son respectivamente 1, 2, 3 y 4. Calcular el valor de $P(2)$.
98. **P+** Describir una extensión del algoritmo FFT al caso en que N es una potencia de 3. ¿Cuál sería su rendimiento asintótico?
99. **P+** Describir cómo extender la FFT al caso de entradas cuyo tamaño no fuera una potencia de 2. ¿Cuál sería el rendimiento del algoritmo resultante?
100. **E** Comprobar la propiedad de inversión de la Transformada Discreta de Fourier aplicando esta y su inversa a los valores [1 1 1] en el caso $N = 3$ y a [1 2 3 4] en el caso $N = 4$.
101. **E** Calcular mediante la Transformada Discreta de Fourier el producto de los polinomios $X^2 + 2X + 1$ y $X + 2$.

FFT.

103. **P** El siguiente algoritmo, conocido como Regla de Horner, proporciona la evaluación de un polinomio de grado N y coeficientes $C[0], C[1], \dots, C[N]$ en un punto X :

```
float Horner(int N, float C[], float X)
  P = C[N];
  para i de N-1 a 0:
    P = P*X + C[i];
  devolver P;
```

Comprobar que el algoritmo es correcto, estimar su rendimiento y compararlo con el del algoritmo “simple” habitual.

104. **P** Si se conocen los valores $P_i, i = 0, \dots, N$ de un polinomio $P(X)$ de grado N en los puntos X_i , el propio polinomio $P(X)$ puede entonces calcularse en la forma $P(X) = \sum_0^N P_i L_i(X)$, donde los distintos $L_i(X)$ se definen como $L_i(X) = \prod_{j \neq i} (X - X_j) / \prod_{j \neq i} (X_i - X_j)$. Comprobar que en efecto esto es así (la expresión anterior para $P(X)$ se conoce como la fórmula de interpolación de Lagrange).
105. **E** De dos polinomios P y Q se sabe que el grado de su producto PQ es ≤ 3 y que el valor de PQ en los puntos $1, i, -1, -i$ es respectivamente $6, -1 + i, 0$ y $-1 - i$. Identificar PQ .
106. **P** Resolver para $N = K^Q$ la recurrencia $T(N) = mT(N/K) + N^P$ que mide la eficacia en general de un algoritmo recursivo.
107. **P+** Estudiar el crecimiento de las soluciones de la recurrencia $T(N) = mT(N/K) + \Theta(N^p \log^q N)$.
108. **P+** Los métodos recursivos pueden dar lugar en ocasiones a algoritmos totalmente horribles bajo el punto de vista de la eficacia computacional. Un ejemplo de este tipo de situaciones sería un algoritmo recursivo para el cálculo de los números de Fibonacci. Escribir un tal algoritmo y estimar su rendimiento.
109. **P** El pésimo rendimiento de algoritmos como el anterior suele deberse a que el cálculo de ciertos valores se repite innecesariamente. Una forma de evitar este hecho es usar una tabla *caching* $c[]$ para los valores intermedios, donde $c[i] = 1$ si un cierto término T_i ya ha sido calculado, o bien vale 0 si no lo ha sido. Modificar el algoritmo recursivo anterior para el cálculo de los números de Fibonacci usando un tal tabla global. ¿Cuál sería la eficacia del algoritmo resultante?

Programación dinámica

110. **P** Dar el pseudocódigo de un algoritmo PD para determinar el beneficio óptimo en el problema 0–1 de la mochila, y estimar su eficacia.
111. **P** Modificar el algoritmo anterior para que determine la composición de una carga óptima.
112. **E** Dar la evolución del algoritmo PD para el problema 0–1 de la mochila con cota 17 sobre los siguientes elementos

elemento:	1	2	3	4	5
peso:	3	4	7	8	9
valor:	4	5	10	11	13

elemento pueda haber un número cualquiera de ejemplares.

114. **P** Dar un algoritmo PD que, recibiendo una cantidad en pesetas, especifique qué monedas de curso legal deben usarse para que el número total de monedas para cambiar dicha cantidad sea mínimo.
115. **P** El Problema de la Suma (una variante del de la Mochila) consiste en, dada una serie finita $\mathcal{S} = S_1, \dots, S_N$ y un valor V , determinar si algún subconjunto S_{i_1}, \dots, S_{i_K} , $K \leq N$ de \mathcal{S} verifica que $S_{i_1} + \dots + S_{i_K} = V$.
- Encontrar un algoritmo general de programación dinámica para resolver dicho problema, recogiendo el pseudocódigo del mismo en una rutina `bool SumaSi(lista S, valor V)` y aplicarlo a la cadena 1,3,5,7 y el valor 11.
116. **P** Dar un algoritmo que recibiendo N enteros positivos y una cota C , determine qué subconjunto de dichos números proporciona la suma más alta.
117. **E+** Dada la lista 10, 15, 23, 35, 44, decidir si los números 77, 54, 68 pueden descomponerse como sumas de los de la lista y de ser así, dar dicha descomposición.
118. **E+** Encontrar los inversos de 8 módulo 13, módulo 39 y módulo 141.
119. **E+** La sucesión 4, 17, 18, 25, 27 se ha obtenido a partir de una sucesión supercreciente usando como módulo 50 y como multiplicador 9. Encontrar dicha sucesión supercreciente.
120. **E+** Usando las sucesiones del problema anterior, codificar y decodificar mediante el método de la mochila la cadena de bits 1 0 1 1 0 0 1 1 0 0 usando como clave pública la sucesión 4, 17, 18, 25, 27 y el módulo 50.
121. **E** (a) Dar la evolución del algoritmo PD para el problema 0–1 de la mochila con cota 20 sobre los siguientes elementos

elemento:	1	2	3	4	5
peso:	4	6	8	9	11
valor:	6	5	9	11	7

(b) ¿Cuál sería dicha evolución si cada elemento puede aparecer cualquier número de veces?

122. **P** Modificar el algoritmo dado en clase de obtención del número mínimo de operaciones en multiplicación de matrices para que también proporcione la ordenación óptima.
123. **E** Dar la evolución del algoritmo anterior y la ordenación óptima cuando se emplea para multiplicar 4 matrices de tamaños respectivos 10×15 , 15×20 , 20×25 y 25×30 .
124. **P+** Desarrollar un algoritmo recursivo para el cálculo del número óptimo de productos en la multiplicación de matrices respecto a la propiedad de subestructura óptima de dicho problema. ¿Cuál sería su rendimiento?
125. **P+** Escribir un algoritmo recursivo de determinación del coste óptimo de búsqueda en árboles binarios y estimar su rendimiento.
126. **P** Modificar el algoritmo habitual de programación dinámica para que permita especificar el árbol óptimo.

[0.22 0.18 0.35 0.25].

128. **P** La *distancia* entre una cadena P y otra T es el número mínimo de operaciones de uno de los tipos siguientes que transforma una de las cadenas en la otra:
- (a) modificar un carácter de una de las cadenas;
 - (b) añadir un carácter a una de las cadenas;
 - (c) quitar un carácter de una de las cadenas.

Por ejemplo, la distancia entre las cadenas inferiores es 3:

```
P =   un  escessaraly
      | |      |
T =   unne cessarily
```

Dar un algoritmo PD que determine la distancia entre dos cadenas así definida y estimar su eficacia. (Sugerencia: definir $D(i, j)$ como el mínimo número de diferencias entre los i primeros caracteres de P y los j primeros caracteres de T).

129. **E** a. Encontrar razonadamente la distancia mínima entre las cadenas *noria* y *radio*.
b. Dar una mochila óptima de cota 14 para elementos e_1, e_2, e_3 y e_4 de pesos 1, 2, 3 y 6 y de valores 2, 4, 6 y 10 respectivamente, suponiendo que de cada elemento hay un número cualquiera de ejemplares.
130. **E** Encontrar razonadamente la distancia mínima entre las cadenas *forraje* y *zarzajo*.
131. **E** Se quiere construir un árbol binario de búsqueda óptimo para los elementos 1, 2, 3, 4, 5 suponiendo que las frecuencias de aparición de los mismos son 0.15, 0.30, 0.25, 0.10, 0.20. Recoger en una tabla la evolución del algoritmo PD que determine dicho árbol y construirlo.
132. **P** Estudiar razonadamente los requerimientos de memoria de los distintos algoritmos de programación dinámica de los problemas anteriores.

Algoritmos codiciosos

133. **P** Identificar con precisión el carácter codicioso de los algoritmos de Dijkstra, Prim, Kruskal y del problema fraccionario de la mochila.
134. **P** Completar la demostración de la propiedad de elección codiciosa del algoritmo para el problema fraccionario de la mochila.
135. **P** Dar un algoritmo codicioso que, recibiendo una cantidad en pesetas, especifique qué monedas de curso legal deben usarse para que el número total de monedas para cambiar dicha cantidad sea mínimo.
136. **P** Decidir razonadamente si el algoritmo codicioso anterior puede dar correctamente el cambio en las monedas de cualquier país. (Pista: No.) Dar un algoritmo correcto para este problema. (Pista: PD)

el control del procesador, no lo cede hasta que termina. Dar un algoritmo que recibiendo N trabajos T_1, T_2, \dots, T_N de tiempos de ejecución $t_i, i = 1 \dots N$, proporcione un orden de ejecución que minimice el tiempo medio de espera (Idea: dar un algoritmo codicioso demostrando que proporciona una solución correcta).

138. **E** Dar la evolución de los algoritmos FF, NF y BF de empaquetamiento en cajas de tamaño 1 sobre los paquetes de tamaños 0.5, 0.2, 0.4, 0.7, 0.1. 0.2 y 0.6.
139. **E+** a. Deducir razonadamente una codificación Huffman de los caracteres a, b, c, d, e y f si los mismos aparecen 1, 2, 5, 12, 27 y 58 veces respectivamente.
- b. En general, supongamos que, como sucede en el caso anterior, se tienen N caracteres c_1, \dots, c_N y que la frecuencia relativa de c_i es proporcional a $2^i - i$. Deducir razonadamente una posible codificación Huffman.
140. **E+** a. Encontrar un código prefijo óptimo para un archivo con los caracteres a, b, c, d, e, f, g, h todos con una misma frecuencia 10 (en caso de coincidencia de frecuencias, unir los dos primeros elementos por orden alfabético).
- b. En general, dado un archivo con 2^N elementos equiprobables, dar razonadamente la estructura del correspondiente código prefijo.
141. **P** a. Indicar la evolución de los algoritmos NextFit y FirstFit sobre una serie de cajas de los siguientes tamaños comparando sus efectos con los de una ordenación óptima:
(6m cajas) $1/2 + \epsilon$, (6m cajas) $1/3 + \epsilon$, (6m cajas) $1/6 + \epsilon$, (6m cajas) $1/6 - 2\epsilon$.
- b. Dar una codificación Huffman para un archivo formado por N caracteres c_i de de frecuencias $f_i = 3^i$.
142. **P** Analizar el rendimiento de la versión on line del algoritmo First Fit.

143. **E** Construir el código Huffman de compresión máxima para un fichero con los siguientes caracteres y frecuencias:

caracter:	a	b	c	d	e	f
frecuencia:	45	13	12	16	9	5

144. **E** Construir el código Huffman de compresión máxima para un fichero con los siguientes caracteres y frecuencias:

caracter:	a	b	c	d	e	f
frecuencia:	13	8	5	3	2	1

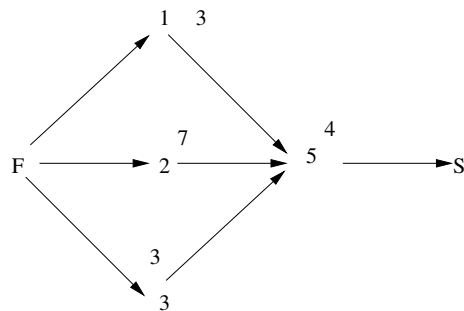
indicando razonadamente en cada paso los distintos árboles de metacaracteres.

145. **P** En el problema anterior las frecuencias de los caracteres se corresponden con los primeros números de Fibonacci. En general, si en un fichero con N caracteres la frecuencia del carácter i -ésimo es F_i , ¿cuál sería su codificación Huffman?
146. **E** Un fichero consta de dos símbolos, uno con probabilidad 0.9999 y otro con probabilidad 0.0001. ¿Cuál es la longitud media mínima por símbolo? ¿Cuáles son las longitudes de los códigos Huffman? ¿Cuáles son las longitudes de los códigos Shannon?

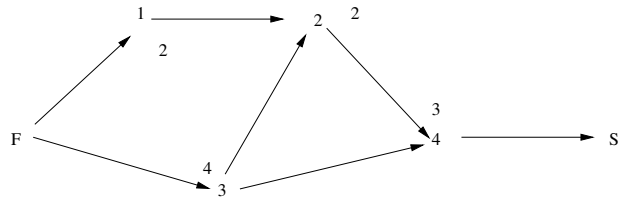
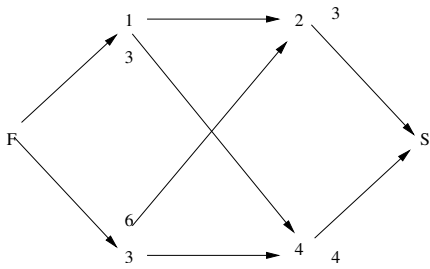
gitudes distintas para los caracteres. Ilustrar esta situación para un archivo de 4 símbolos y probabilidades respectivas ($1/3$, $1/3$, $1/4$ y $1/12$). Calcular las longitudes teóricas óptimas y observar que para algún código, su longitud óptima Huffman puede ser mayor que su longitud teórica.

148. **P+** Comprobar para una distribución discreta p que $H(p) \geq 0$. ¿Cuál es la distribución discreta de entropía mínima? ¿Cuál es la entropía máxima de una distribución discreta de K valores?
149. **E** Encontrar el código Huffman para una distribución de frecuencias ($1/3$, $1/5$, $1/5$, $2/15$, $2/15$). Argumentar que dicho código también es óptimo para la distribución ($1/5$, $1/5$, $1/5$, $1/5$, $1/5$).
150. **E** ¿Cuáles de los siguientes códigos NO puede ser de Huffman: $\{0, 10, 11\}$, $\{00, 01, 10, 110\}$, $\{01, 10\}$?
151. **P** El código $\{0, 01\}$, ¿es prefijo?, ¿es decodificable?
152. **P** Un código sufijo cumple que ningún de sus símbolos es sufijo de ninguno otro. Argumentar que un código sufijo es decodificable. Argumentar cómo construir a la Huffman un código sufijo óptimo y aplicar el método a alguna distribución de los problemas anteriores.
153. **P** Argumentar cómo establecer una codificación a la Huffman sobre un alfabeto ternario (esto es, de 3 símbolos). Aplicar dicha codificación a la distribución ($1/21$, $2/21$, $3/21$, $4/21$, $5/21$, $6/21$).
154. **P** Dar una versión de la desigualdad de Kraft para códigos ternarios y, en general, para códigos sobre un alfabeto de M símbolos.
155. **P** ¿Cómo se definiría la codificación Shannon para un alfabeto de M símbolos?

156. **E** Si G es un grafo dirigido acíclico, una ordenación topológica es una enumeración de sus vértices en la que sólo si hay un camino de u a v , u precede a v . Dar por inspección una ordenación topológica de los grafos del problema 2.
157. **P** Se puede dar un algoritmo de ordenación topológica de un grafo dirigido acíclico partiendo de un vértice de incidencia 0 y procesando los demás mediante una cola tal y como se efectúa en el algoritmo de distancia mínima en grafos no ponderados.
Escribir el psc de dicho algoritmo y recoger mediante una tabla su evolución en los grafos del problema 3.
158. **P** Si G es un grafo dirigido acíclico, demostrar que la ordenación topológica define una relación matemática de orden.
159. **P+** Demostrar que en un grafo dirigido acíclico siempre hay un vértice de incidencia 0 (esto es, un vértice al que no llegan ramas).
160. **P+** Demostrar que en un grafo dirigido acíclico siempre hay un vértice de excedencia 0 (esto es, un vértice del que no salen ramas).
161. **P** Si G es un grafo dirigido acíclico, una ordenación topológica inversa es una enumeración de sus vértices en la que si hay un camino de v a u , u precede a v . Dar un algoritmo de ordenación topológica inversa basado en listas de incidencia y vértices de excedencia 0.
162. **E** Dar la evolución del algoritmo anterior del problema precedente sobre los grafos dirigidos acíclicos del problema 2.
163. **P** Un grafo de tareas es un grafo del estilo del de la figura inferior donde cada vértice del grafo indica una cierta tarea y su coste, y donde las ramas han de entenderse como indicando que una tarea no puede comenzar hasta que no terminen las de las ramas que inciden en ella. Por ejemplo, en la figura, la tarea 4 no puede empezar hasta que no acaben las tareas 1, 2 y 3. ¿Cómo podría almacenarse un tal grafo?



164. **E** Para poder pasar en un grafo de tareas los costes de tareas a ramas, se utilizan los denominados grafos de hitos, en los que si una tarea T depende de otras anteriores, se introduce un hito de finalización T' , en el que inciden con coste 0 las ramas que antes incidían en T y se añade una rama (T', T) de coste del de la tarea T . Dar el grafo de hitos de los siguientes grafos de tareas.
165. **P/E** El tiempo mínimo de finalización de un hito es el mínimo tiempo necesario para alcanzar dicho hito suponiendo alcanzados todos sus hitos anterior (y terminadas las correspondientes



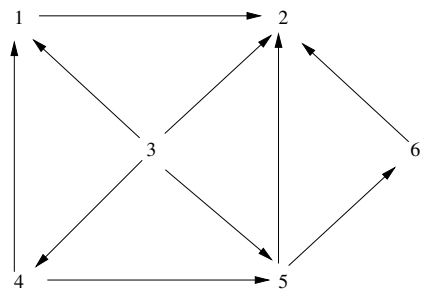
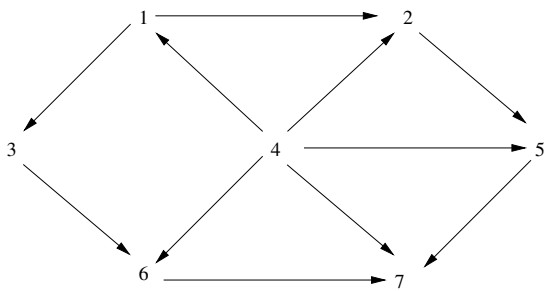
tareas). Comprobar que si $tm(H)$ denota el tiempo mínimo de finalización de un hito H , se cumple que $tm(H) = \max\{tm(H') + c(H', H)\}$. donde el máximo se toma sobre todas las ramas (H', H) incidentes en H .

Calcular por inspección los tiempos mínimos de los grafos del problema anterior.

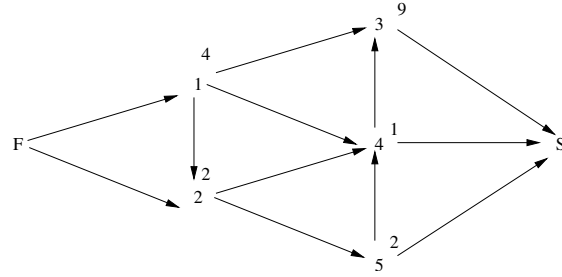
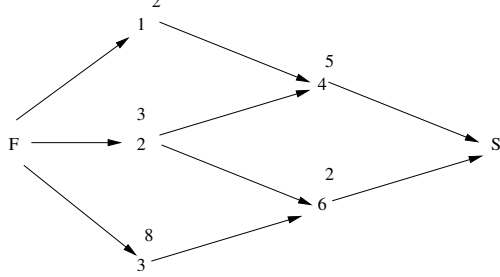
166. **P/E** El tiempo máximo de finalización de un hito es el tiempo máximo en que dicho hito puede terminarse sin afectar al tiempo mínimo global del grafo $tm(F)$. Comprobar que si $TM(H)$ denota el tiempo máximo de finalización de un hito H , se cumple que $TM(H) = \min\{c(H, H') + TM(H')\}$. donde el mínimo se toma sobre todas las ramas (H, H') que salen de H .

Calcular por inspección los tiempos máximos de los grafos del problema anterior.

167. **P+** ¿Cómo se podría modificar el algoritmo de distancias mínimas dado en clase para grafos no ponderados de tal manera que proporcione la distancia mínima de un vértice a los demás sobre GDAs ponderados?
168. **P** Dar el pseudocódigo de un algoritmo que transforme un grafo de tareas representado convenientemente en el correspondiente grafo de hitos
169. **P** Escribir el psc de un algoritmo de cálculo de tiempos máximos de finalización de un grafo de hitos.
170. **E** Sobre los siguientes grafos dirigidos acíclicos, reflejar sobre una tabla la evolución del algoritmo de distancias mínimas empezando desde los vértices 4 y 3 respectivamente.

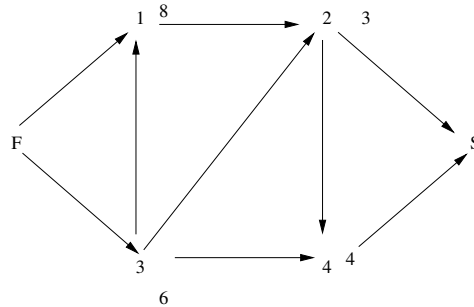


171. **E** Sobre los grafos de actividades inferiores,
1. construir el grafo de hitos asociado;
 2. especificar mediante una tabla la evolución del algoritmo de determinación de tiempos mínimos de ejecución;
 3. calcular por inspección tiempos máximos de ejecución;
 4. deducir razonadamente cuáles de las tareas originales son críticas.



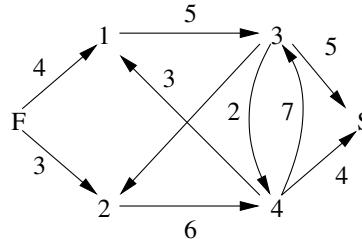
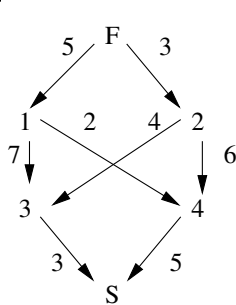
172. **E** Sobre el grafo de actividades inferior,

1. construir el grafo de hitos asociado;
2. calcular por inspección tiempos mínimos de ejecución;
3. especificar mediante una tabla la evolución del algoritmo de determinación de tiempos máximos de ejecución sobre la base de la ordenación topológica inversa;
4. deducir razonadamente cuáles son las ramas críticas.

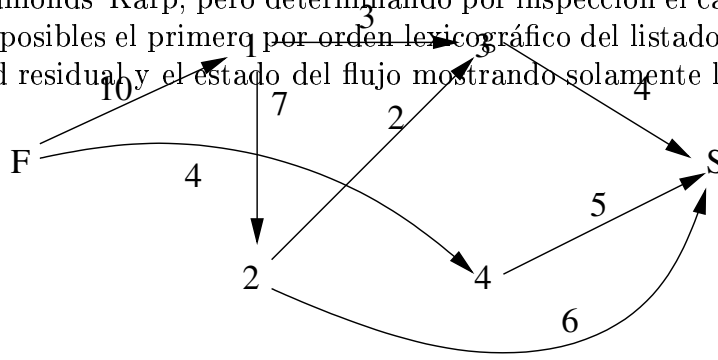


Flujos en grafos

173. **P** Demostrar que si f es un flujo en un grafo de flujo $G = (V, R)$, el flujo total que entra en un vértice distinto de la fuente y el sumidero es igual al que sale del mismo.
174. **P** Dar el pseudocódigo de una rutina que encuentre un camino aumentador en una cierta red residual.
175. **P** Dar el pseudocódigo completo de una rutina o rutinas que implementen el algoritmo de Ford–Fulkerson.
176. **E** Encontrar mediante el algoritmo de Ford–Fulkerson flujos máximos para las siguientes redes de flujo especificando en cada paso los sucesivos caminos aumentadores y las correspondientes redes residuales.

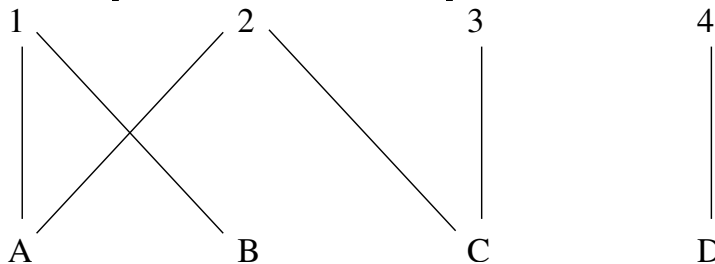


algoritmo de Edmonds–Karp, pero determinando por inspección el camino mínimo, y eligiendo entre los varios posibles el primero por orden lexicográfico del listado de sus vértices. Dar para cada paso la red residual y el estado del flujo mostrando solamente las ramas de flujo positivo

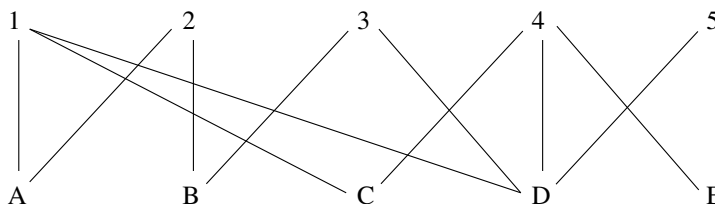


178. **P+** Un grafo se dice *bipartido* si sus vértices están divididos en dos subconjuntos V_1 y V_2 y sus ramas solamente conectan vértices del conjunto V_1 con vértices de V_2 . El problema de *emparejamiento máximo* consiste en encontrar el mayor número posible de ramas de tal manera que en cada vértice sólo incida una. ¿Cómo se podría encontrar un tal emparejamiento mediante técnicas de flujos en grafos?

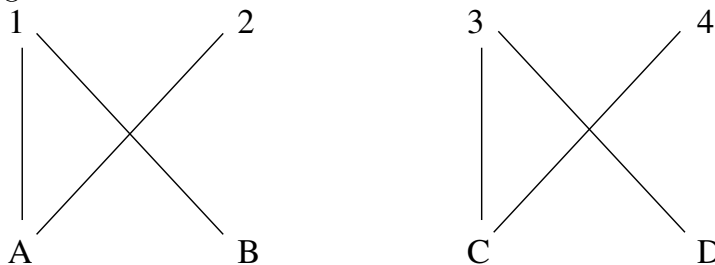
cográfica para deshacer posibles elecciones múltiples.



180. **E** Encontrar un emparejamiento máximo para el grafo bipartido inferior. ¿Cuál es el corte mínimo?



181. **E** Encontrar de manera razonada un emparejamiento máximo en el grafo inferior mediante un algoritmo de determinación de un flujo máximo. Utilizar el algoritmo de Edmonds–Karp, determinando por inspección el camino mínimo, y eligiendo entre los varios posibles el primero por orden lexicográfico del listado de sus vértices.



182. **P** Demostrar que si f es un flujo en un grafo de flujo $G = (V, R)$, y $G_f = (V, R_f)$ es su red residual, se tiene que $|R_f| \leq 2|R|$.

183. **P** Demostrar que si f es un flujo en un grafo de flujo $G = (V, R)$, $G_f = (V, R_f)$ es su red residual y en ella hay un camino aumentador π , el flujo f_π obtenido incrementando o decrementando f en c_π en las ramas adecuadas, proporciona un nuevo flujo.

184. **P** Demostrar que si f es un flujo en un grafo de flujo $G = (V, R)$, y (V_1, V_2) es un corte, entonces $|f| = f(V_1, V_2)$.

185. **P** Demostrar que si f es un flujo en un grafo de flujo $G = (V, R)$, se tiene que $|f| = \sum_{v \in V} f(v, S)$.

186. **E** ¿Existe una matriz 2×3 de ceros y unos cuyas filas sumen respectivamente 3, 2 y cuyas columnas sumen 2, 1 y 2 respectivamente?

187. **P** En general, ¿cómo se podría determinar si existe una matriz $M \times N$ de ceros y unos cuyas filas sumen f_1, \dots, f_M y sus columnas c_1, \dots, c_N , con $\sum f_i = \sum c_j$?

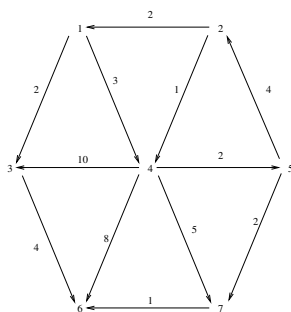
188. Si se desea ver los grafos como un tipo abstracto de datos, ¿cuáles podrían ser sus operaciones primitivas?

189. Suponiendo que una implementación del algoritmo de Prim devuelve una tabla $p[]$ de previos cuyos índices son los vértices del grafo, y cuyos valores son otro índice o NULL, dar el pseudocódigo de una función ListaEnlazada AAm(grafo G, tabla $p[]$) que reconstruya en una lista enlazada el árbol abarcador mínimo representado por $p[]$.

Suponer para ello la disponibilidad de las primitivas de listas enlazadas `status IniLE(ListaEnlazada L)`, `status VacíaLE(ListaEnlazada L)`, `status RemLE(elemento e, ListaEnlazada L)`, `status AddLE(elemento e, ListaEnlazada L)`.

En la función pedida no hay que considerar errores debidos a las primitivas de listas enlazadas, pero sí errores lógicos debidos a incoherencias en la tabla $p[]$ que imposibiliten en todo o en parte la obtención de un árbol.

190. Aplicar el algoritmo de Dijkstra de determinación de distancias mínimas sobre el grafo inferior especificando en cada momento el estado de la cola de prioridad (considerar múltiples reencoles):



191. Modificar el algoritmo de búsqueda en amplitud para su correcto funcionamiento en grafos no conexos, dirigidos o no.

192. Discutir una posible clasificación de las ramas de un grafo respecto al algoritmo de búsqueda en amplitud.

193. Eliminar la recursión del algoritmo general de recorrido en preorden de árboles.

194. Sobre la base del código anterior, escribir una versión no recursiva de la búsqueda en profundidad y aplicarla a los grafos anteriores. ¿Qué árboles de búsqueda se obtienen?

195. Si fuera necesario, ¿cómo habría que modificar el algoritmo anterior para que produjera el mismo árbol que la búsqueda en profundidad?

196. ¿Se podrían encontrar circuitos hamiltonianos en un grafo a partir de un algoritmo aproximado para TSP con un factor de aproximación arbitrario λ ?

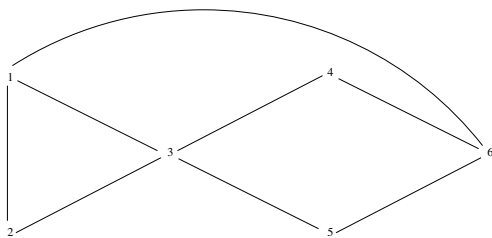
197. Los coeficientes binómicos $C(n, k) = \frac{n!}{k!(n-k)!}$ verifican las relaciones $C(n, 0) = 1$ si $n \geq 0$, $C(0, k) = 0$ si $k > 0$, y $C(n, k) = C(n-1, k-1) + C(n-1, k)$ si $n > 0$ y $k > 0$.

Escribir el psc de un algoritmo DyV para su cálculo y estimar su rendimiento.

198. Modificar el algoritmo anterior utilizando caching. ¿Qué mejora se consigue en su rendimiento?

¿Cuál sería su rendimiento?

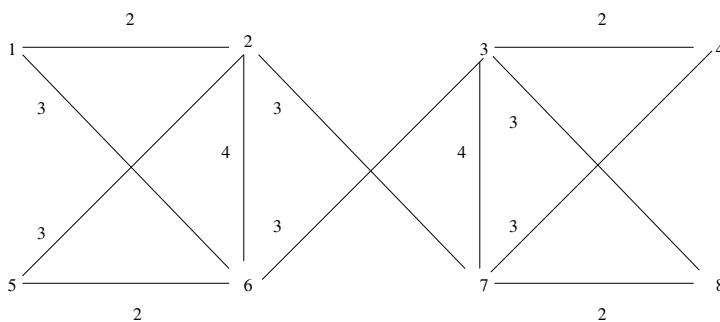
200. Estimar la eficacia del algoritmo PD dado en clase para la estimación del coste óptimo de búsqueda en árboles binarios.
201. Para el grafo inferior,
1. comprobar razonadamente que el mismo posee algún camino euleriano, y señalar desde qué puntos podría comenzar;
 2. encontrar razonadamente un camino euleriano a partir del posible punto de arranque de numeración más baja.



202. Si k es un entero positivo, una coincidencia aproximada de orden k entre una cadena patrón P y una cadena objetivo T se produce cuando T posee una subcadena T' cuya distancia a P según el problema anterior es k .

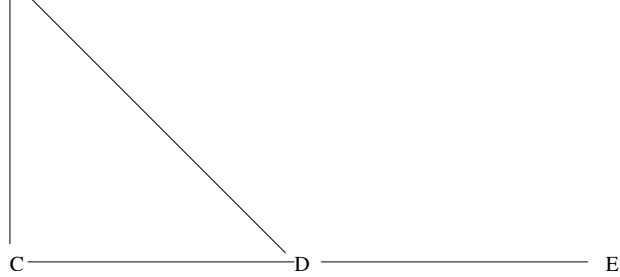
Dar un algoritmo que determine la primera coincidencia aproximada de orden k entre P y T .

203. En un árbol se duplica cada rama dando lugar a un multigrafo donde cada vértice tiene incidencia par (tal y como se hizo en el algoritmo aproximado para TSP). ¿Cuál podría ser un algoritmo simple que produjera un circuito euleriano?
204. Encontrar razonadamente para el grafo inferior una solución aproximada al problema del viajante. El grafo inferior es completo, y el peso de las ramas no mostradas es 4, para así tener un grafo euclídeo.



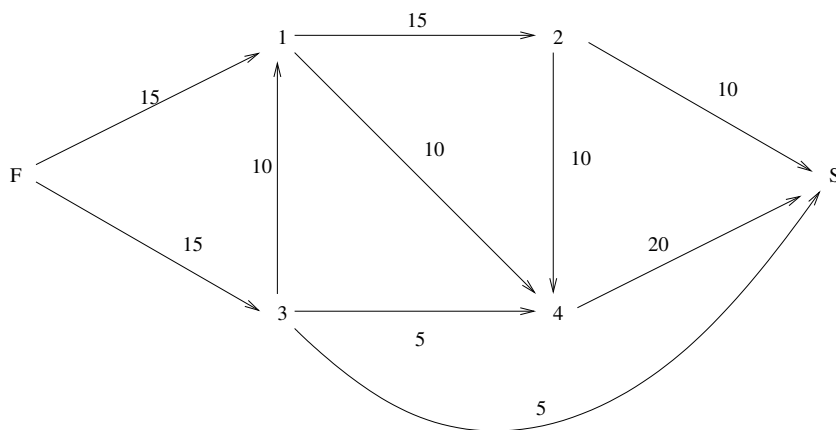
completo con función de coste verificando la desigualdad euclídea, se obtiene una solución a TSP de coste a lo sumo dos veces el óptimo.

206. ¿Cuál podría ser un “algoritmo de programación dinámica” para el cálculo de los coeficientes de Fibonacci?
207. ¿Cuál podría ser un “algoritmo de programación dinámica” para el cálculo de coeficientes binómicos?
208. ¿Cuál debiera ser el número máximo de productos al multiplicar dos matrices 8×8 para que el algoritmo recursivo de multiplicación de matrices resultante fuera más eficaz que el de Strassen?
209. Encontrar razonadamente los puntos de articulación del grafo inferior. Para una mayor homogeneidad, usar listas de adyacencia ordenadas alfabéticamente, empezar el cálculo en el vértice A y representar los valores a calcular en una tabla. Aunque no se precisen en el cálculo, indicar adecuadamente las ramas ascendentes. Durante la ejecución de la búsqueda en profundidad indicar el comienzo y el retorno de las sucesivas llamadas recursivas, numerando su orden de ejecución.

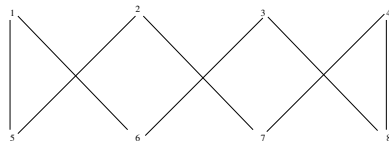


210. Para el grafo de flujo inferior,
1. encontrar un flujo máximo;
 2. determinar el corte mínimo correspondiente.

Nota: en cada paso, de entre los caminos aumentadores posibles, escoger el “primero” respecto al orden lexicográfico.



211. Encontrar un emparejamiento máximo en el grafo inferior.



212. En ocasiones un mismo polinomio P fijo debe evaluarse repetidamente en puntos distintos. En estas circunstancias una técnica de interés es el **preproceso de coeficientes**. Supongamos para fijar ideas que el grado de P es $N = 2^K - 1$ para un cierto K y que P es mónico (esto es, que el coeficiente de X^{2^K-1} es 1). Entonces, el primer paso a dar es factorizar P como

$$P(X) = \left(X^{2^{K-1}} + (c_{2^{K-1}-1} - 1) \right) Q(X) + R(X),$$

donde $C = c_{2^{K-1}-1}$ es el correspondiente coeficiente de P , y donde Q y R son mónicos y de grado $2^{K-1} - 1$. Comprobar que estos hechos son ciertos.

análoga a la anterior, obteniéndose nuevos polinomios mónicos de grados $2^{K-2} - 1$, $2^{K-3} - 1$, etc., hasta llegar finalmente a polinomios de grado 1. El cálculo de todos estos polinomios es lo que se conoce como **preproceso de coeficientes**.

Supongamos que este preproceso ya se ha completado. Dado entonces un valor w , $P(w)$ podría calcularse mediante el siguiente pseudocódigo:

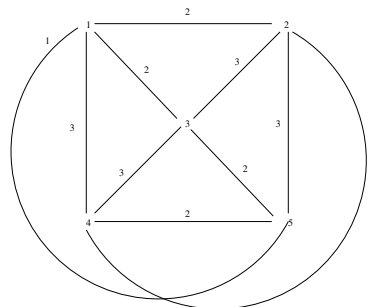
```

calcular Q(w) y R(w);
calcular W = w * ... * w ((N+1)/2 veces);
devolver (W + C) * Q(W) + R(W);

```

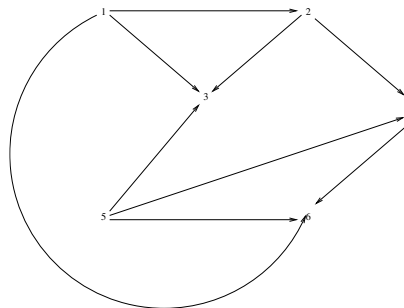
¿Cuál sería el número de multiplicaciones realizadas en esta evaluación de $P(w)$? ¿Cuál sería el número de sumas? (Ambos pueden calcularse mediante una recurrencia.)

214. Dar el pseudocódigo de un algoritmo codicioso (¡y erróneo!) de construcción del un árbol binario de búsqueda óptimo.
215. Dar el pseudocódigo de una implementación del algoritmo de Huffman que utilice una cola de prioridad para gestión de caracteres y metacaracteres. Estimar la eficacia del pseudocódigo resultante.
216. Modificar el pseudocódigo anterior para que proporcione el tamaño en bits del fichero comprimido.
217. (a). Dar el pseudocódigo de una rutina para el cálculo de la función \lg^* .
(b). ¿Cómo habría que modificar el algoritmo de búsqueda en amplitud para grafos no dirigidos para poder detectar componentes conexas?
218. A. ¿Cuál sería el máximo número de productos a realizar para multiplicar dos matrices 5×5 para poder definir un algoritmo de producto de matrices más eficaz que el de Strassen?
B. Se quiere encontrar el décimo elemento de la tabla 1 6 14 2 12 9 10 11 4 8 13 5 15 3 7 2 12 9 10 mediante el algoritmo `QuickSelect2`. Detallar las sucesivas llamadas a `QuickSelect2` que se producirían sobre dicha tabla.
219. Encontrar una solución aproximada del problema del viajante para el grafo inferior. Para ello, usar la versión básica del algoritmo de Kruskal, y dar en detalle el proceso de obtención de un circuito euleriano en un multigrafo.



inc) que devuelve en forma de lista enlazada un circuito euleriano parcial para el grafo G empezando y comenzando en el vértice U respecto a la tabla inc. Dar detalladamente el pseudocódigo de una rutina que llamando a la anterior, proporcione un circuito euleriano completo para el grafo G como una lista enlazada. (Describir primero las tareas a realizar por dicha rutina y dar luego su código).

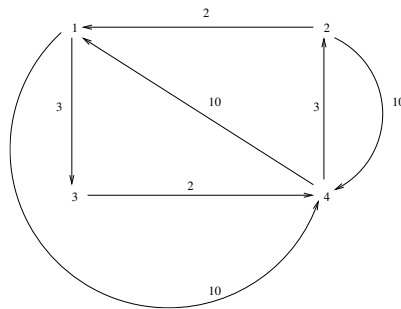
221. De un problema de maximización (o equivalentemente, minimización) \mathcal{M} donde a partir de una entrada \mathcal{I} se intenta maximizar (o minimizar) una variable V , se puede definir un problema de decisión \mathcal{D} donde a partir de una entrada \mathcal{I} y un valor fijo V_0 de la variable V se intenta decidir si sobre \mathcal{I} se puede superar (o bajar) el valor V_0 . Dar versiones de decisión de los problemas de la mochila, del viajante y de la ordenación óptima de un producto de matrices.
222. En el escenario anterior, y suponiendo que se dispone de un algoritmo \mathcal{A} para el problema \mathcal{M} , dar un algoritmo para el problema \mathcal{D} asociado.
223. a. Encontrar razonadamente la distancia mínima entre las cadenas `trino` y `orion`, señalando en particular las fórmulas utilizadas.
- b. Supongamos que a partir de las fórmulas anteriores se quiere escribir una función de prototipo `int DistMinRec (cadena S, long N, cadena T, long M)` que calcule recursivamente la distancia entre las cadenas S y T de longitudes respectivas N y M . Dar su pseudocódigo y calcular cuántas llamadas recursivas produciría su aplicación a las cadenas de la primera cuestión.
- c. En general, y para la rutina anterior, dar una cota inferior significativa para el número de llamadas recursivas que se producirían al aplicarla a dos cadenas de la misma longitud N .
224. Encontrar razonadamente las componentes fuertemente conexas del grafo inferior, ordenando en las listas de adyacencia a utilizar los vértices de manera creciente.



225. Los elementos a , b , c , d se presentan con las frecuencias relativas $[0.15, 0.35, 0.10, 0.40]$. Encontrar para los mismos un árbol binario de búsqueda óptimo.
226. La evaluación en los puntos $1, i, -1$ y $-i$ de dos polinomios P y Q proporciona las tablas $(4, 2i, 0, -2i)$ y $(0, i - 1, -2, -i - 1)$ respectivamente. Encontrar razonadamente el polinomio producto PQ sabiendo que su grado es menor que 4.
227. a. Encontrar razonadamente la distancia mínima entre las cadenas `perrazo` y `zorrera`.
- b. Encontrar un código prefijo óptimo para un archivo con los caracteres a, b, c, d, e, f, g, h cuyas frecuencias respectivas son 20, 12, 33, 5, 10, 6, 25, 40. Indicar el tamaño en bits del archivo comprimido.

de Fourier.

229. Encontrar razonadamente las distancias mínimas entre todos los pares de puntos del grafo inferior.



230. a. Indicar razonadamente el orden óptimo de multiplicación de cuatro matrices A , B , C y D de dimensiones 20×10 , 10×30 , 30×20 y 20×40 respectivamente (dar la condición de subestructura óptima para este problema).
b. Encontrar razonadamente la distancia mínima entre las cadenas **gorrino** y **rigodon** (dar la condición de subestructura óptima para este problema).

231. Los valores de un cierto polinomio de dimensión ≤ 7 en los puntos

$$0, \frac{\sqrt{2} + i\sqrt{2}}{2}, i, \frac{-\sqrt{2} + i\sqrt{2}}{2}, -1, \frac{-\sqrt{2} - i\sqrt{2}}{2}, -i, \frac{\sqrt{2} - i\sqrt{2}}{2}$$

son

$$4, 1 + 2\sqrt{2} - i, 0, 1 - 2\sqrt{2} + i, 0, 1 - 2\sqrt{2} - i, 0, 1 + 2\sqrt{2} + i.$$

respectivamente. Hallar dicho polinomio utilizando la Transformada Rápida de Fourier.

232. Los elementos a , b , c , d se presentan con las frecuencias relativas $[0.30, 0.35, 0.15, 0.20]$. Encontrar para los mismos un árbol binario de búsqueda óptimo.

233. A. ¿Cuál sería el máximo número de productos a realizar para multiplicar dos matrices 7×7 para poder definir un algoritmo de producto de matrices más eficaz que el de Strassen?

B. Indicar la evolución de los algoritmos NextFit y FirstFit sobre una serie de cajas de los siguientes tamaños comparando sus efectos con los de una ordenación óptima:

(12 = 4×3 cajas) $1/2 + \epsilon$, (12 cajas) $1/4 - \epsilon$, (12 cajas) $1/2 - \epsilon$, (12 cajas) $1/4 + \epsilon$.

234. Un cierto país usa N tipos de monedas, de valores $v_1 = 1, v_2, \dots, v_N$. Dar una propiedad de subestructura óptima y el pseudocódigo de un algoritmo `int Cambio(int Val)` que devuelva el número mínimo de monedas necesario para cambiar una cantidad Val .

Aplicar dicho algoritmo si $N = 3, v_1 = 1, v_2 = 2, v_3 = 5$ y $Val = 7$.

235. Una implementación del algoritmo de Kruskal nos devuelve una lista de ramas sobre nodos con campos `vert1`, `vert2`, `next2` y un dato de tipo `listaE2` apunta a un tal nodo. Dar el pseudocódigo de una función `grafo LE2AAm(int N, listaE2 L)` que a partir de una tal lista L devuelta por Kruskal sobre un grafo de N vértices, devuelva el grafo del árbol abarcador mínimo como un puntero a su tabla de listas de adyacencia.

Observaciones:

1. Dar primero una versión sin control de errores y depurarla luego adecuadamente.

tipo `listaE` apunta a un tal nodo (no considerar costes de ramas). 3. Suponer disponibles macros `vert(listaE L)`, `next(listaE L)`, `vert1(listaE2 L)`, `vert2(listaE2 L)`, `next(listaE2 L)` para el acceso a los campos de nodos.

4. Si se juzga necesario, utilizar funciones `InsListaE(dato D, listaE L)`, `InsListaE2(dato D, listaE2 L)` para inserción de un dato `D` en unas tales listas.