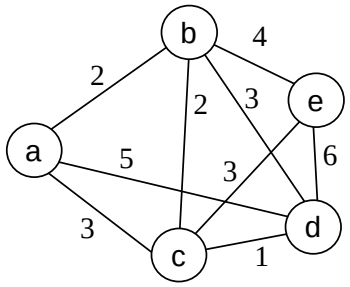
6.1 En un problema determinado, una solución está dada por una tupla de **n** elementos (x_1, x_2..., x_n). Para cada elemento existen en total **m** posibles valores. Comparar el número de nodos generados para resolver el problema usando un algoritmo de backtracking (suponiendo que no se realiza ninguna poda), un algoritmo de avance rápido (se supondrá que cada posibilidad probada es un nodo), y un algoritmo de ramificación y poda. En el último caso, ¿podemos predecir el número exacto de nodos generados o debemos dar un mejor y peor caso?

6.2 Aplicar la técnica de ramificación y poda al problema del viajante. Dado un grafo no dirigido y ponderado, encontrar un ciclo simple de coste mínimo que pase por todos los nodos una sola vez. Definir la representación de la solución y una forma de obtener la cota superior, inferior y el coste estimado a partir de una solución parcial. Escribir el algoritmo para resolver el problema. ¿Cuál es el orden de complejidad del algoritmo?
Mostrar la ejecución sobre el siguiente grafo.



6.3 Aplicar el algoritmo de ramificación y poda para el problema de la mochila 0/1, al siguiente ejemplo: n= 5, M= 20, v= (10, 7, 6, 4, 2), w= (30, 15, 11, 8, 2). Elegir el tipo de representación de la solución y las estrategias de ramificación y de poda que, previsiblemente, den lugar a un número mínimo de nodos generados.

6.4 Suponer un problema de satisfacción de restricciones como el de las **n** reinas, donde queremos obtener todas las posibles soluciones para un **n** dado. En esta situación, ¿se obtiene algún beneficio utilizando ramificación y poda en lugar de backtracking? Justificar la respuesta. Resolver la pregunta para el caso general de problemas de este tipo.

6.5 Resolver por backtracking el problema de obtener de una serie de números dada {x_1, x_2..., x_n} los que suman una cantidad S, utilizando el menor número de elementos x_i posible. Utilizar un esquema no recursivo, como el visto en clase, con funciones **Generar**, **MasHermanos**, **Solución** y **Criterio**, indicando también cómo se representa la solución y cuando finaliza el algoritmo.
Decir cómo se podrían obtener cotas inferiores y superiores y cómo se podría estimar el coste, para resolverlo utilizando ramificación y poda. Definir un estrategia de ramificación y de poda adecuadas a este problema concreto.
Aplicar sobre el siguiente ejemplo: x= {4, 12, 6, 3, 1}, S = 11.

6.6 Suponer un juego de tablero como el ajedrez, donde el número de posibilidades es muy elevado y el árbol de juego completo es inabordable para cualquier computador. Para poder resolver el juego de forma eficiente, ¿es necesario variar las definiciones de la cota superior e inferior del beneficio, y los criterios para

realizar la poda del árbol? ¿Qué problemas puede tener una estrategia de ramificación en profundidad? ¿Cómo será la estrategia de ramificación más adecuada?

6.7 Para el problema del cambio de monedas, especificar una buena forma de realizar el cálculo de las cotas y la estimación de beneficio que se puede obtener a partir de un nodo. Definir los demás aspectos necesarios para poder aplicar el esquema de ramificación y poda y escribir el algoritmo.
Suponer que a partir de un nodo obtenemos una solución mediante un algoritmo de avance rápido. ¿Qué información nos aporta esto, sobre la solución óptima que se puede alcanzar a partir de ese nodo?

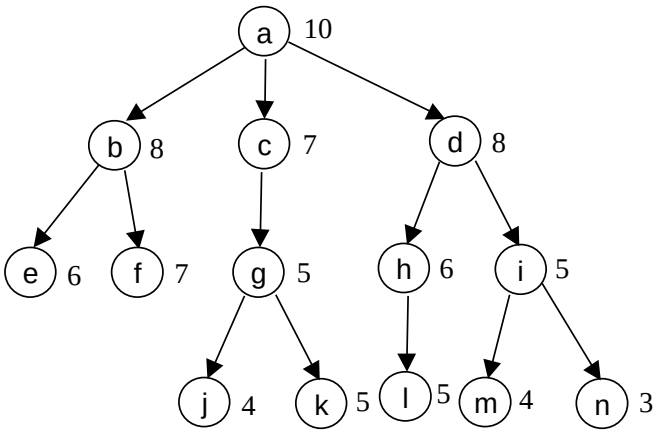
6.8 (TG 13.2) En un problema de ramificación y poda podemos utilizar dos técnicas distintas para calcular las cotas y para la estimación del beneficio a partir de un nodo.

a) Para el cálculo de las cotas podemos usar un método con un tiempo **t(n) = 2n** (cada cota tarda t(n) = n), que producirá una poda del **p*50%** de los nodos, o bien podemos usar otro método que tarda **t(n) = $2n^2$**, que produce una poda del **p*75%** de los nodos. El valor de **p** (entre 0 y 1) hace referencia a lo rápido que la estimación del beneficio nos dirige hacia la solución óptima (se supone que usamos una estrategia LC).
b) Para la estimación del beneficio se puede usar un procedimiento de tiempo **t(n) = n**, que da lugar a un valor de **p=0.5**, o se puede usar otro procedimiento con **t(n) = n^2**, que tiene un **p=0.25**.

Sea **M** el número total de nodos del árbol completo; suponer que se puede usar cualquier combinación de los métodos anteriores y que el tiempo de manejar la lista de nodos vivos es despreciable. Calcular para cada combinación el tiempo necesario para ejecutar el algoritmo. ¿De qué forma se consigue un mejor tiempo de ejecución? ¿Son los datos del problema son realistas (tiempos de ejecución, valores de **p**...) o existe alguna incoherencia?

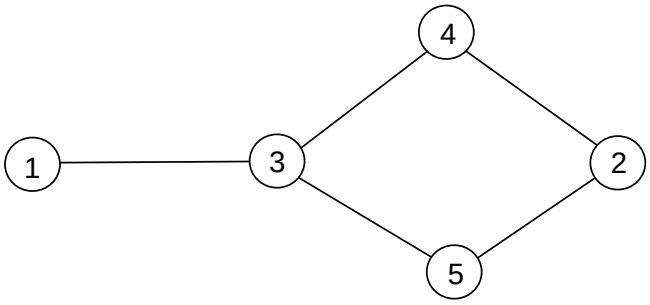
6.9 Suponer el problema de encontrar todos los subconjuntos de un conjunto dado de enteros positivos $\{x_1, x_2..., x_n\}$ que sumen una cantidad M. Para resolverlo utilizamos ramificación y poda, con una representación binaria de la solución. La cota inferior usada es la suma de los elementos de la solución actual, y la cota superior es la cota inferior más los elementos que faltan por tratar.
¿Por qué no tiene ningún sentido dar una estimación del beneficio y utilizar una estrategia LC? Dar una estrategia de ramificación y de poda adecuadas para este problema, que sean óptimas en cuanto a tiempo de ejecución y memoria requerida.

6.10 (TG 13.3) El siguiente árbol de soluciones corresponde a un problema de maximización:



El número que acompaña a cada nodo es la cota superior del beneficio que se puede alcanzar a partir de ese nodo, que será igual al beneficio estimado para ese nodo. Para los nodos hoja, será el valor de una solución final. No existe una cota inferior, puesto que a partir de un nodo puede no existir una solución. Enumerar el orden en que se recorren los nodos con los métodos: backtracking, ramificación y poda con FIFO, LIFO, LC-FIFO y LC-FIFO. Suponer que los hijos se generan por orden alfabético.

6.11 Supongamos que queremos resolver el problema de coloración de grafos usando ramificación y poda. Dar fórmulas para calcular las cotas y el beneficio estimado para cada nodo, y establecer una estrategia de poda, según las características del problema. Dar un esquema en pseudocódigo del algoritmo y mostrar la ejecución sobre el siguiente ejemplo sencillo.



6.12 En Windows, cuando se quieren copiar varios archivos en disquetes, se empiezan a copiar por el orden seleccionado. Cuando un disquete se llena, se mete otro y se sigue copiando. Con este método, puede que necesitemos más disquetes de los necesarios. Por ejemplo, suponiendo que en un disquete caben 1.4 Mbytes, y los archivos son de tamaño: 400 Kb, 400 Kb, 800 Kb, 800 Kb, necesitaríamos 3 discos, cuando podríamos hacerlo con sólo 2. Se supone que los archivos no se pueden partir y que son de menor tamaño que la capacidad del disquete.
Diseñar una solución para el problema anterior utilizando ramificación y poda. Dado un array **T[1..n]**, con los tamaños de los archivos, y una cantidad **M** que indica el espacio libre de los disquetes, el problema consiste en encontrar el

número mínimo de disquetes necesarios para copiar los archivos, y la asignación **S[1..n]**, indicando el número de disquete en el que se debe copiar cada archivo.

a) Exponer qué forma tendrá el árbol de búsqueda, con un ejemplo sencillo de árbol. ¿Qué representa cada nivel del árbol? ¿Qué descendientes son generados para cada nodo? ¿Cuándo un nodo es una solución final?
b) Escribir el esquema del algoritmo de ramificación y poda. Especificar cómo son las funciones para: generar los hijos de un nodo, comprobar si un nodo es solución final. Dar fórmulas (lo más ajustadas posible) para calcular cotas y el beneficio estimado para cada nodo.
c) Establecer una estrategia de ramificación y de poda adecuadas, y mostrar la ejecución del algoritmo para el ejemplo anterior.

6.13 Considerar el esquema de ramificación y poda visto en clase, con la condición de poda para un problema de minimización y suponiendo que a partir de un nodo siempre existe alguna solución. Suponer que hacemos un cálculo de las cotas muy preciso, de manera que para ciertos nodos interiores ocurre que CI(i) = CS(i). ¿Puede esta situación causar algún problema en la ejecución del algoritmo (sin tener en cuenta el tiempo de ejecución)? En caso afirmativo, explicar brevemente cuál es el problema y cómo se podría solucionar.

6.14 La compañía *Eoloeléctrica Española* ha estado estudiando el problema de los cortes de suministro eléctrico durante este verano. Se ha comprobado que la potencia que producen las centrales es suficiente para cubrir la demanda. El problema se encuentra en la falta de capacidad de las líneas de alta tensión, que llevan la electricidad desde las centrales eléctricas hasta las ciudades. Se supone que hay **m** centrales productoras **P**= (p_1..., p_m), cada una de las cuales produce **PP**[i] Mwatios, con i= 1.. m. Por otro lado, existen **n** ciudades **C**= (c_1..., c_n), cuya potencia máxima consumida es **PC**[j], con j= 1..n.
Para solucionar el problema, se pueden construir nuevas líneas de alta tensión, desde una central excedentaria **i** hasta una ciudad deficitaria **j**. La construcción de esa línea tiene un coste específico **D[i, j]**. La capacidad de la nueva línea es fija, ya que por ley no se puede superar un tope **T**. La capacidad de las líneas existentes en la actualidad está almacenada en **L[i, j]**. Entre una central y una ciudad puede existir más de una línea.
Se pide:
a) Diseñar un algoritmo, con alguna de las técnicas vistas en clase, para resolver el problema de satisfacer la demanda de todas las ciudades, minimizando el coste requerido. Se entiende que la solución está formada por las nuevas líneas que se deben construir y el coste total de su construcción.
b) Hacer una estimación aproximada del orden de complejidad del algoritmo diseñado. Aplicar el algoritmo sobre el siguiente ejemplo:

m= 3 centrales; **n** = 3 ciudades

Central	p_1	p_2	p_3
PP (Mwatios)	16	20	13

Ciudad	c_1	c_2	c_3
PC (Mwatios)	14	16	13

Capacidad de cada línea nueva **T** = 5 Mwatios

Líneas existentes

L (Mwatios)	c_1	c_2	c_3
p_1	5	0	0
p_2	0	4	0
p_3	0	0	5

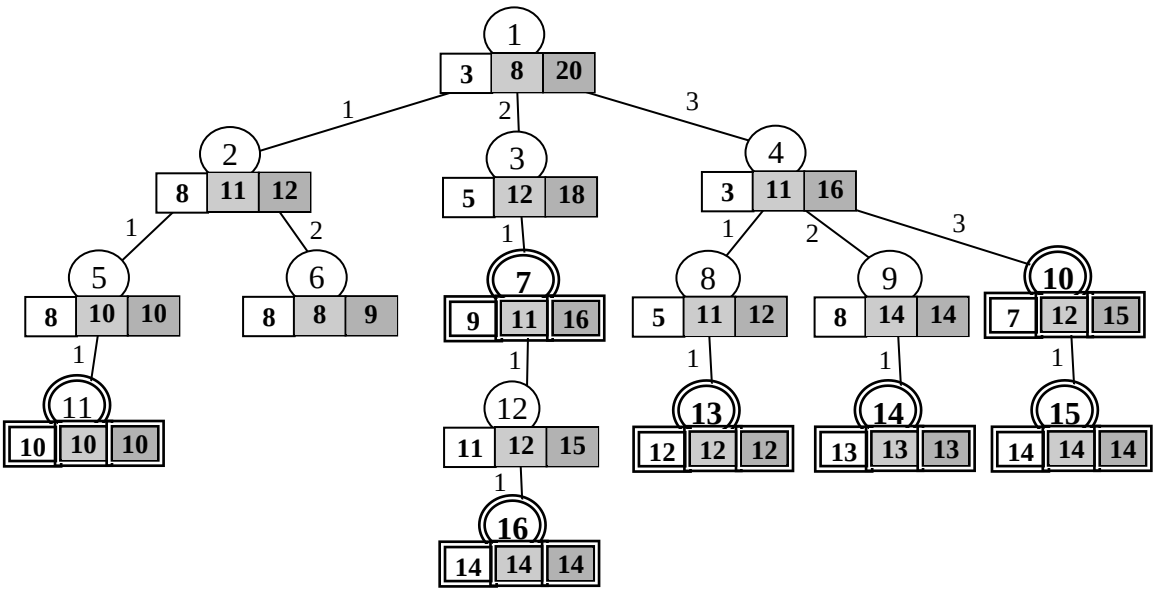
Coste de las líneas nuevas

D (mill. euros)	c_1	c_2	c_3
p_1	2	5	4
p_2	3	7	3
p_3	5	6	8

6.15 (TG 13.6) a) Consideramos el problema de la mochila modificado consistente en dado un damero de dimensiones $M_1 \times M_2$ y n piezas de dimensiones $i_1 \times j_1$, $i_2 \times j_2$..., $i_n \times j_n$, cada una de ellas con beneficio b_1, b_2..., b_n, maximizar el beneficio de poner las piezas en el tablero teniendo en cuenta que las piezas se pueden separar en cuadrículas para ponerlas en el tablero y que el beneficio de poner una cuadrícula de una pieza de dimensión i × j con beneficio b es b/ij. Resolver el problema con avance rápido y explicar cómo funcionaría para el caso de un tablero 3×4 y piezas 2×3, 1×4, 3×1 y 2×2, con beneficios 6, 3, 4 y 2, respectivamente.

b) Explicar cómo se podría utilizar el avance rápido anterior en la resolución por ramificación y poda del mismo problema pero sin poder dividirse en cuadrados las piezas.

6.16 En cierto problema de maximización representamos la solución mediante una tupla $\mathbf{S} = (s_1, s_2, ..., s_k)$. Cada s_j puede tomar ciertos valores enteros. El valor de **k** no es conocido a priori. Es más, a partir de una solución pueden existir otras soluciones. El problema es resuelto con ramificación y poda, de manera que el espacio de soluciones tiene la siguiente forma.



En cada nivel **q** se prueban las posibilidades para $\mathbf{s_q}$. Para cada nodo se muestran (de izquierda a derecha) la cota inferior, el beneficio estimado y la cota superior. Los nodos que son una posible solución final aparecen con una doble línea. El beneficio final de estos nodos coincide con la cota inferior.

a) Modificando el esquema de ramificación y poda visto en clase, diseñar una estructura general del algoritmo para resolver un problema como el anterior. Obviamente, las funciones básicas (*Generar*, *Solución*, *CI*, *CS*, etc.) deben quedar sin especificar.

b) Mostrar la ejecución del algoritmo (orden en que son recorridos los nodos, lista de nodos vivos, variable de poda, solución final obtenida) utilizando las estrategias de ramificación LC-FIFO y LC-LIFO. ¿Cuál parece más adecuada? ¿Por qué?

6.17 El árbol de abajo representa el espacio de soluciones en un problema de minimización. Para cada nodo se muestra (de izquierda a derecha) la cota inferior, el coste estimado y la cota superior. Todos los nodos hoja son soluciones válidas. El problema se resuelve con ramificación y poda, usando la estrategia LC-LIFO.
Mostrar, para cada paso del algoritmo, la lista de nodos vivos, el valor de la variable de poda **C**, los nodos podados y la solución óptima.