5.1 En un problema de backtracking estamos interesados en almacenar de forma explícita el árbol recorrido por el algoritmo. De cada nodo del árbol sólo necesitamos saber un número, que indica el orden en que ha sido generado. Para ello utilizamos una estructura de representación de punteros al padre (como la vista en el Tema 3 de la Parte I).
Describir cómo se debería modificar el esquema general de backtracking visto en clase para realizar esta función. ¿Cómo varía la eficiencia del algoritmo? Tener en cuenta el tiempo y el espacio utilizados por el algoritmo.

5.2 Aplicar el algoritmo de backtracking para el problema de la mochila 0/1 visto en clase (con poda según el beneficio estimado por el algoritmo voraz), al siguiente ejemplo: n= 5, M= 20, v= (10, 7, 6, 4, 2), w= (30, 15, 11, 8, 2). Obtener la solución, utilizando un árbol binario de soluciones y un árbol combinatorio.

5.3 (TG Sec. 12.1.3) Proponer un esquema recursivo genérico para el método de backtracking (para los mismos tipos de problemas vistos en clase). ¿Qué ventajas e inconvenientes tiene esta versión respecto a la no recursiva?

5.4 Diseñar una solución para el problema de la coloración de grafos utilizando backtracking. Dado un grafo no dirigido G = (V, A), el problema consiste en encontrar una coloración de los nodos, usando un número mínimo de colores.
Se supondrá que el grafo consta de **n** nodos y que utilizamos una representación con una matriz de adyacencia **A[1..n, 1..n]** de booleanos. En lugar de colores, se asignarán etiquetas numéricas a los nodos: 1, 2, 3... Una solución será representada como una tupla $\mathbf{S = (s_1, s_2, ..., s_n)}$, donde $\mathbf{s_i \in (1, 2, 3, ...)}$ indica el color asignado al nodo **i**.
Utilizar un esquema similar al visto en clase, dando el esquema y las funciones que aparecen en el mismo (Generar, MasHermanos, Criterio...).

5.5 Suponer que decidimos utilizar un algoritmo con backtracking para el problema del cambio de moneda, en un caso donde el algoritmo voraz no da la solución óptima.
a) Especificar la forma de resolver el problema, indicando cómo sería la representación de la solución y las funciones utilizadas en el esquema genérico (Genera, MasHermanos, Criterio, Solución...).
b) ¿Sería válido utilizar el algoritmo voraz para el cambio de monedas como una cota del beneficio que se puede obtener a partir de un nodo? Justificar la respuesta.

5.6 En el algoritmo de la mochila 0/1 con backtracking, decidimos no usar variables locales de peso y beneficio acumulados en cada nodo (**bact**, **pact**) sino calcularlas siempre que las necesitemos, aplicando las fórmulas correspondientes. Hacer un cálculo del tiempo de ejecución en el peor caso, con esta modificación. ¿Se modifica el orden de complejidad del algoritmo?

5.7 Diseñar una solución para el problema del ciclo hamiltoniano utilizando backtracking. Utilizar un esquema similar al visto en clase, dando el esquema y las funciones que aparecen en el mismo (Generar, MasHermanos, Criterio...).

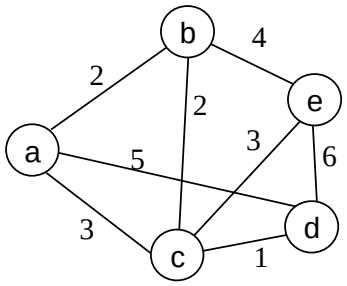
5.8 En una matriz cuadrada M, de tamaño nxn representamos un laberinto. Partimos de la posición (1, 1) y el objetivo es moverse a la posición (n, n). Podemos pasar por la casilla (i, j) si y sólo si M[i, j] = A (abierta). Si M[i, j] = C (cerrada) entonces no podemos pasar por esa casilla. Desde cada casilla existen 4 posibles movimientos: arriba, abajo, izquierda y derecha.

A	**C**	A	A	A
A	A	A	**C**	**C**
A	**C**	A	A	A
A	**C**	A	**C**	A
A	A	A	**C**	A

a) Describir la forma de resolver el problema utilizando backtracking (representación de la solución, funciones del esquema básico, ...). Idea: tener en cuenta que en cada momento sólo necesitamos conocer la posición actual en el tablero.
b) Puesto que, en general, es posible llegar a un mismo sitio por varios caminos distintos, el árbol de soluciones será realmente un grafo. ¿Es posible que existan ciclos en este grafo? En caso afirmativo, ¿qué significa un ciclo y qué consecuencias tiene en el algoritmo de backtracking? ¿Cómo solucionarlo?
c) ¿Es adecuada la aplicación de backtracking a este problema? ¿Existe alguna otra posible solución más eficiente?

5.9 Mostrar el esquema general del algoritmo de backtracking aplicable a cada uno de los siguientes tipos de problemas. Indicar cómo se debe utilizar para resolver esos problemas. ¿Es suficiente con especificar la representación de la solución y las funciones básicas del esquema? En caso contrario indicar qué partes se deben modificar.
Para cada uno de ellos, mostrar el árbol recorrido por el algoritmo con algún ejemplo sencillo.
a) Dado un conjunto de números enteros $\{x_1, x_2..., x_n\}$, encontrar todos los subconjuntos que sumen una cantidad M.
b) Encontrar los **n** enteros positivos distintos $x_1, x_2..., x_n$, tales que, dado otro entero positivo N, minimicen $\sum_{i=1}^{n} x_i^2$ y cumplan que $\sum_{i=1}^{n} x_i = N$. Ojo: el único dato del problema aquí es N.
c) Dado un conjunto de números enteros positivos $\{x_1, x_2..., x_n\}$ y un entero N, encontrar un subconjunto $\{y_1, y_2..., y_m\}$ que minimice $|N - \sum_{i=1}^{n} y_i|$.
d) Dado un conjunto de números enteros positivos $\{x_1, x_2..., x_n\}$ y un entero positivo N, encontrar un subconjunto $\{y_1, y_2..., y_m\}$ que minimice $D = N - y_1*y_2*...*y_m$, sujeto a la restricción $D \geq 0$.
e) Dado un conjunto de números enteros positivos $\{x_1, x_2..., x_n\}$ encontrar un subconjunto $\{y_1..., y_m\}$ que cumpla cierta propiedad $P(\{y_1..., y_m\})$. ¿Qué diferencia existe en este caso respecto a los anteriores, en cuanto a tiempo de ejecución?

5.10 Estudiar la aplicación del método de backtracking sobre el problema del viajante. Mostrar la ejecución del algoritmo propuesto sobre el siguiente grafo de ejemplo.



Hacer una estimación del orden de complejidad del algoritmo en el peor caso.

5.11 (TG 12.2) Considerar la siguiente variante del problema de asignación. Tenemos una tabla T con **n** filas y **m** columnas que representan las posibilidades de que ciertos trabajadores (**n**) realicen determinados trabajos (**m**). Si T[i, j]=1 entonces el trabajador i-ésimo puede realizar el trabajo j-ésimo. En caso contrario no puede realizarlo. Cada trabajo puede ser realizado por uno o ningún trabajador, y cada trabajador debe tener un trabajo o ninguno. El objetivo es obtener una asignación de trabajadores con trabajos, de forma que el número de trabajos realizados sea máximo. Diseñar un algoritmo con backtracking para resolver este problema. ¿Cuál es el orden de complejidad?

5.12 En una liga de fútbol participan **n** equipos (suponemos que **n** es par). En cada jornada se juegan **n/2** partidos, que enfrentan a dos equipos, dirigidos por un árbitro. Existen **m** árbitros disponibles, siendo **m>n/2**. Cada equipo **i** valora a cada árbitro **j** con una puntuación P[i, j] entre 0 y 10, indicando su preferencia por ese árbitro. Un valor alto indica que le gusta el árbitro y un valor bajo que no le gusta. Si el árbitro y el equipo son de la misma región entonces P[i, j]=-∞.
El objetivo es (para cada jornada concreta) asignar un árbitro distinto a cada partido, de manera que se maximice la puntuación total de los árbitros asignados, teniendo en cuenta las preferencias de todos los equipos.

a) Dar una solución óptima para el problema usando backtracking. Exponer cómo es la representación de la solución, cuál es el esquema que habría que utilizar, cuál es la condición de fin y cómo son las funciones del esquema (**Generar**, **MasHermanos**, **Criterio**, **Solución...)**.
b) Hacer una estimación del orden de complejidad del algoritmo.
c) Partiendo de la solución de backtracking del punto a), comentar cómo se podría resolver el problema con ramificación y poda, indicando el modo de calcular las cotas y las estrategias que se deberían definir.

5.13 Queremos resolver el problema de la mochila 0/1, pero con la posibilidad de usar varias mochilas, en lugar de una sola. Tenemos **n** objetos, cada uno con su peso $\mathbf{w_i}$ y su beneficio $\mathbf{v_i}$, y un número indeterminado de mochilas con una capacidad máxima de peso **M**. El objetivo es decidir cuántas mochilas utilizar para obtener el máximo beneficio de los objetos transportados, pero equilibrando los beneficios entre las mochilas. Para ello, si decidimos usar **p** mochilas, y cada una de ellas tiene beneficio ($\mathbf{B_1}$, $\mathbf{B_2}$..., $\mathbf{B_p}$), el valor que queremos optimizar es: **p***min{$\mathbf{B_1}$,

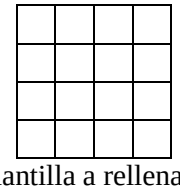
$\mathbf{B_2}$..., $\mathbf{B_p}$}. Resolver el problema de forma óptima, usando alguna de las técnicas vistas en clase. Tener en cuenta que no necesariamente todos los objetos deben ser metidos en alguna mochila, y que los objetos no se pueden partir.
Aplicar el algoritmo diseñado sobre el siguiente ejemplo: **n**= 3; **v**= (2, 3, 4); **w**= (1, 2, 3), **M**= 5.

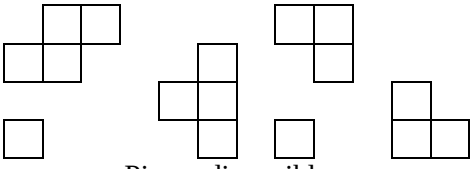
5.14 El problema del empaquetamiento en recipientes consiste en lo siguiente. Tenemos **n** objetos de pesos $\mathbf{w_1}$, $\mathbf{w_2}$,…, $\mathbf{w_n}$, y un número ilimitado de recipientes con capacidad máxima **R** (siendo $\mathbf{w_i} \leq \mathbf{R}$, para todo **i**). Los objetos se deben meter en los recipientes sin partirlos, y sin superar su capacidad máxima. Se busca el mínimo número de recipientes necesarios para colocar todos los objetos.
Diseñar un algoritmo de backtracking para encontrar la solución óptima del problema. Indicar la forma del árbol, la representación de la solución, el esquema del algoritmo y las funciones del mismo. Mostrar la ejecución del algoritmo sobre algún ejemplo sencillo.

5.15 Gracias a un duro entrenamiento físico hemos conseguido, entre otras cosas, ser capaces de levantar del suelo dos mochilas cargadas de objetos. Así que nos decidimos a resolver el problema de la mochila 0/1, pero utilizando dos mochilas en lugar de una. Los datos son: **n** objetos candidatos, cada uno de peso $\mathbf{w_i}$ y beneficio $\mathbf{v_i}$, los objetos se pueden meter enteros en una u otra mochila, siempre que no se sobrepase el peso máximo **M** (igual para ambas mochilas). El objetivo final es maximizar la suma de los beneficios de los objetos transportados en ambas mochilas. Diseñar un algoritmo que resuelva el problema de forma óptima.

5.16 (TG 12.4) Resolver por backtracking el problema de la devolución de monedas con el siguiente planteamiento: minimizar el número de monedas a devolver para dar una cantidad **C** si tenemos monedas de **n** tipos, estando los tipos de las monedas en un array **tipos: array** [1..**n**] **de** enteros, y teniendo de cada tipo una cierta cantidad de monedas, estando estas cantidades almacenadas en un array **cantidad: array** [1..**n**] **de** enteros (de la moneda de tipo **tipos[i]** podemos dar una cantidad entre 0 y **cantidad[i]**).
Hay que decir qué estructuras de datos se utilizarían, cómo sería el árbol de soluciones, cómo se representan las soluciones, cuál es la condición de fin, qué esquema se usa y programar las funciones.

5.17 (TG 12.1) Se quiere hacer un programa por backtracking que resuelva un rompecabezas consistente en rellenar completamente una plantilla cuadriculada con unas ciertas piezas dadas (ver el dibujo de abajo). Las piezas se pueden rotar y cada pieza sólo se puede usar una vez. Explicar cómo vendrían dados los datos del problema, cómo se podría representar una solución, cómo sería el árbol de soluciones, cuál sería la condición de final, y cómo serían los procedimientos **Generar**, **Criterio**, **Solución**, **MasHermanos** y **Retroceder** del esquema.


Plantilla a rellenar


Piezas disponibles