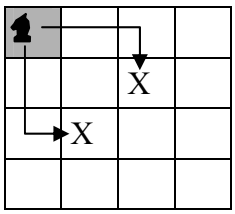
7.1. Modificar el algoritmo voraz del problema de la mochila visto en clase, para tratar el caso en que no se permita partir los objetos (es decir, un objeto se coge entero o no se coge nada). Comprobar que el algoritmo no garantiza la solución óptima en este caso. **Idea**: la demostración se puede hacer con un contraejemplo.

7.2. Se define el problema de la **minimización del tiempo en el sistema** de la siguiente manera. Tenemos un conjunto de **n** tareas, cada una de las cuales tarda un tiempo predefinido $\mathbf{t_i}$, y un procesador donde se ejecutan las tareas. El objetivo es dar una planificación de las tareas (un orden de ejecución de las mismas) de manera que se minimice el tiempo medio de finalización. El tiempo de finalización de una tarea es el tiempo que transcurre entre el instante inicial (instante 0) y el momento en el que concluye una tarea. Construir un algoritmo voraz para resolver el problema de la minimización del tiempo en el sistema. Mostrar la ejecución para el ejemplo, **n**= 4, **t**= (4, 10, 2, 20). Demostrar que el algoritmo voraz construido es óptimo.

7.3. Construir un algoritmo voraz para resolver el problema de la minimización del tiempo en el sistema pero con varios procesadores. Tenemos un conjunto de **n** tareas, cada una de las cuales tarda un tiempo $\mathbf{t_i}$, y disponemos de **m** procesadores para ejecutar las tareas. Las tareas se ejecutan secuencialmente, es decir, no se pueden partir. ¿Cuál es el orden de complejidad del algoritmo? Mostrar la ejecución para un ejemplo con **m**=4 y **n**=9, **t**=(7, 1, 6, 2, 3, 4, 2, 5, 7).

7.4. (TG Sec. 10.2.2) Resolver el siguiente problema utilizando un algoritmo voraz: En un tablero de ajedrez (de tamaño **n**x**n**) partimos de una casilla cualquiera. Tenemos una ficha de un caballo, que puede realizar los mismos movimientos que en el ajedrez. El objetivo es, partiendo de la posición inicial, visitar todas las casillas del tablero, sin repetir ninguna.
Sugerencia: transformar el problema en un problema sobre grafos, teniendo en cuenta las casillas y las posiciones accesibles desde cada una.



Ejemplo de movimientos válidos del caballo

7.5. Ejecutar el algoritmo para planificación de tareas con plazo fijo, visto en clase, sobre el siguiente conjunto de tareas:
n= 10
g = (10, 20, 8, 15, 30, 10, 5, 10, 2, 9)
d = (1, 4, 4, 1, 4, 2, 3, 4, 5, 4)

7.6. El algoritmo de planificación de tareas con plazo fijo, visto en clase, tiene el inconveniente de que cada vez que se considera un nuevo candidato puede ser necesario mover algunas tareas dentro de la solución actual, en la función factible. Si el candidato no es factible habrá que deshacer el cambio.
a) Propón alguna modificación de forma que no sea necesario mover tareas de la solución actual, cada vez que se considera un nuevo candidato. Sugerencia:

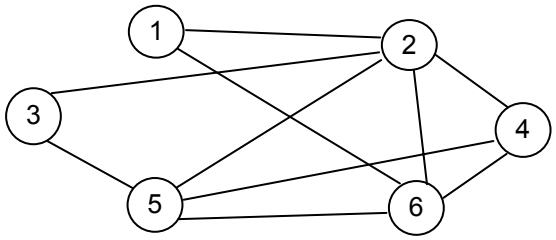
Considera la posibilidad de planificar cada tarea nueva lo más tarde posible, dentro de su plazo de ejecución.

b) Comprueba que la modificación anterior no cambia el resultado del algoritmo, ejecutándolo sobre el ejemplo del ejercicio anterior.

7.7. Suponer que, en el problema del cambio de monedas, disponemos de un sistema monetario en el que cada tipo de moneda vale más del doble que el tipo anterior. La primera moneda tendrá valor 1. Demostrar que estas condiciones no garantizan que el algoritmo voraz para el cambio de monedas encuentre siempre la solución óptima. Y ¿qué ocurre si cada moneda vale exactamente el doble de la anterior?

7.8. En un sistema monetario disponemos de monedas con valores 1, C, C^2, C^3, ..., C^n, siendo C > 1. Demostrar que en este caso el algoritmo voraz para el cambio de monedas obtiene siempre la solución óptima.

7.9. (EX) Diseñar una heurística voraz para el problema del ciclo hamiltoniano. Ejecutarla sobre el grafo de la siguiente figura y dar una estimación del tiempo de ejecución del algoritmo. ¿Garantiza este algoritmo una solución óptima? ¿Garantiza que se encuentre alguna solución en caso de existir?



7.10. Tenemos una tabla:

C[x, y]	a	b	c
a	2	1	1
b	4	3	2
c	2	3	1

Donde **a**, **b** y **c** son trabajos que se deben ejecutar de forma secuencial. Cada valor C[x, y] de la tabla indica el beneficio que se obtendría de ejecutar el trabajo **x** y a continuación el trabajo **y** (por ejemplo, si ejecutamos **ab**, obtendríamos un beneficio de 1, y ejecutando **ba** obtendríamos 4).
Idear un algoritmo basado en el método de avance rápido para maximizar el beneficio. Tendremos que ejecutar los trabajos **a**, **b** y **c** un número $\mathbf{n_a}$, $\mathbf{n_b}$ y $\mathbf{n_c}$ de veces, respectivamente. Mostrar que el algoritmo no es óptimo.

7.11. ¿Se puede considerar la ordenación por selección como un algoritmo voraz? En tal caso, describir cómo serían las funciones del esquema básico: solución, selección, factible, insertar y objetivo.

7.12. Supongamos, en el problema de la mochila, que tenemos una cantidad ilimitada de cada uno de los **n** tipos de objetos (en lugar de tener simplemente **n** objetos).

¿Cómo se debería modificar el algoritmo voraz para este problema? ¿Sigue siendo necesario?

7.13. En la planificación con plazo fijo modificamos la función de selección para que elija los candidatos por orden creciente de plazos de ejecución $\mathbf{d_i}$ (en lugar de hacerlo por orden creciente de $\mathbf{g_i}$). Definir el resto de funciones (solución, factible, insertar) para este caso. Comprobar que este algoritmo no garantiza la solución óptima.

7.14. (EX) Consideramos el problema de la mochila modificado, consistente en dado un damero de dimensiones M_1xM_2 y **n** piezas de dimensiones i_1xj_1, i_2xj_2, ..., i_nxj_n, cada una de ellas con beneficio b_1, b_2, ..., b_n respectivamente, 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 ixj con beneficio b es b/ij. Resolver el problema con avance rápido y explicar cómo funcionaría para el caso de un tablero 3x3 y piezas 2x3, 1x4, 3x1 y 2x2, con beneficios 6, 3, 4 y 2, respectivamente.

7.15. Para el algoritmo diseñado en el ejercicio anterior, comprobar si garantiza siempre la solución óptima. Modificar el algoritmo voraz para el caso en que no se puedan partir las piezas en trozos. Ahora, ¿garantiza el algoritmo la solución óptima?

7.16. Utilizando la idea de los algoritmos voraces, encontrar una solución para el juego del Master Mind: una persona piensa un número **A** de **n** cifras distintas y el ordenador debe tratar de adivinarlo en el menor número de intentos posible. En cada intento el ordenador dice un número **B**, y la persona debe decir cuántas cifras aparecen en **A** en la misma posición y cuántas aparecen pero en otra posición. Decidir cuáles serán los candidatos, cómo seleccionarlos, qué hacemos con la información que nos da el jugador (cifras bien colocadas y mal colocadas) y cuándo acaba el algoritmo.

7.17. (EX) Un sistema dispone de **m** procesadores (P_1, P_2, ..., P_m), que deben ejecutar un conjunto de **n** tareas distintas (T_1, T_2, ..., T_n), disponibles en el instante inicial. De cada tarea se conoce el número de instrucciones que ejecuta $\mathbf{t_i}$ (se supone que todas las instrucciones requieren el mismo tiempo), y de cada procesador se tiene su velocidad de procesamiento $\mathbf{v_i}$, en número de instrucciones por segundo. Se supone que cada procesador ejecuta las tareas de manera secuencial, es decir sin partirlas.

El objetivo consiste en dar una planificación que minimice el tiempo medio de finalización de las tareas. Una planificación consistirá en asignar cada tarea a un procesador, y en un orden determinado. El tiempo medio de finalización será la media de los tiempos que cada tarea tiene que esperar, desde el instante inicial hasta que acaba de ejecutarse.

a) Dar una buena solución para el problema, usando la técnica que creas más adecuada de entre divide y vencerás o algoritmos voraces. Se pide explicar el funcionamiento del algoritmo y dar un esquema en pseudocódigo de su estructura.
b) Ejecutar el algoritmo diseñado sobre el siguiente ejemplo: m= 3, n= 6, t= (35, 40, 20, 25, 10, 50), v= (5, 1, 10).

c) Hacer una estimación del orden de complejidad del algoritmo.
d) Comprobar si el algoritmo diseñado es óptimo o no.

7.18. (EX) Encontrar una buena solución para el siguiente problema usando un algoritmo voraz. Explicar el funcionamiento del algoritmo: cuál es el conjunto de candidatos, la función de selección, la función para añadir un elemento a la solución, el criterio de finalización, el criterio de coste, etc.

En una votación existen **n** candidatos y **m** votantes. La probabilidad de que un votante **i** vote al candidato **j** la conocemos a priori, y viene dada por **P[i, j]**. Un votante cualquiera **a** puede ser coaccionado para que vote al candidato que queramos, por ejemplo el **p**, para lo cual tenemos que pagarle **C[a]** ptas. Con esto, nos aseguramos que **P[a, p] = 1**, y **P[a, j] = 0**, para **j ≠ p**.

El objetivo consiste en gastarse la mínima cantidad de dinero, coaccionando a los votantes necesarios, para garantizar que un candidato **p** dado se llevará al menos el 70% de los votos (de acuerdo con las probabilidades esperadas). La solución estará compuesta por la lista de votantes a los cuales hay que coaccionar.

Aplicar el algoritmo diseñado al siguiente ejemplo: **n** = 2 candidatos, **m** = 7 votantes, **p** = 1. Porcentajes y costes de coacción:

	Votantes						
	1	**2**	**3**	**4**	**5**	**6**	**7**
P[i, 1]	0.2	0.1	0.8	0.5	0.6	0.2	0
P[i, 2]	0.8	0.9	0.2	0.5	0.4	0.8	1
C[i]	4	3	2	5	3	3	5

7.19. (EX) Varios países europeos están sufriendo una epidemia de fiebre **aftosa**. La enfermedad se transmite por contacto directo o indirecto con animales enfermos y es altamente contagiosa. Para controlar la infección, las autoridades sanitarias han recabado información de todas las granjas existentes, que llamaremos **G** = ($\mathbf{g_1}$, $\mathbf{g_2}$, ..., $\mathbf{g_n}$). También se conocen las compras de animales que han tenido lugar entre las distintas granjas, y la fecha en la que tuvo lugar cada una.

Hasta el momento se ha detectado el virus en un conjunto reducido de granjas **I**. Las autoridades pretenden encontrar todas las granjas susceptibles de estar infectadas, para declararlas en cuarentena. Hay que tener en cuenta que una granja es susceptible de padecer la enfermedad si ha comprado animales de una granja infectada, o de una granja susceptible de tener la enfermedad (después de que haya podido ser infectada).

a) Resolver el problema de determinar las granjas que deben ser declaradas en cuarentena. Describir y especificar de forma clara y precisa la estructura de datos que usa el problema, el algoritmo diseñado para resolverlo y la representación de la solución. **Sugerencia:** considerar el problema como un problema sobre grafos. **Ojo:** una buena solución debería ser de complejidad no mayor que $O(n^2)$.
b) Hacer una estimación del orden de complejidad del algoritmo diseñado. Aplicar el algoritmo sobre el siguiente caso:
n = 6; **G** = {**A**, **B**, **C**, **D**, **E**, **F**}; **I** = {**A**}

Granja que vende	E	D	B	C	F	E	B	F	A	C
Granja que compra	B	A	A	D	D	C	F	E	E	B

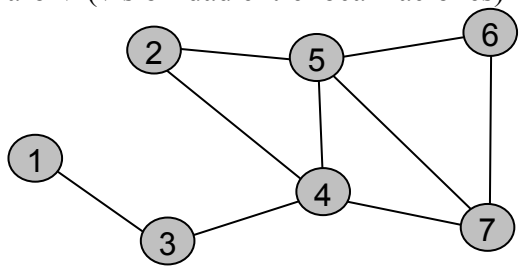
Número de animales	12	7	23	8	9	43	32	6	14	15
Fecha (nº de semana)	6	2	7	5	5	3	6	6	5	6

7.20. (EX S02) La ciudad de Nueva York está ambientada de tal forma que desde cualquier punto se puede ver al menos una bandera de los EEUU. Se pretende conseguir lo mismo en las demás ciudades del país, para lo cual se utilizará una aplicación informática.

Suponer que para cada ciudad a considerar tenemos definida una lista de **n** localizaciones **L** = {$\mathbf{l_1}$, $\mathbf{l_2}$, ..., $\mathbf{l_n}$}, donde se puede colocar una bandera. Mediante la tabla **B** [1..**n**] de booleanos se indica en la posición **B[i]** si ya existe una bandera en la localización **i**. Por otro lado, la matriz **V** [1..**n**, 1..**n**] de booleanos indica si una localización es visible desde otra o no, es decir **V[i, j]** = **V[j, i]** es **true** si desde el sitio **i** se ve el **j** (y viceversa), y **false** en caso contrario. El objetivo del problema es colocar banderas en el mínimo número de localizaciones posible, pero de manera que desde todas ellas sea visible al menos una bandera. Se pide:

a) Diseñar un algoritmo para resolver el problema, utilizando la técnica de avance rápido. Se valorará que las soluciones encontradas sean *buenas*. Dar una explicación y justificación del algoritmo, y hacer una implementación suficientemente detallada.
b) Hacer un estudio aproximado del orden de complejidad del algoritmo en función de **n**. Mostrar la ejecución del algoritmo sobre el siguiente ejemplo, donde **n** = 7. Se supone que **V[i, i]** = **true**, **B[i]** = **false**, para todo **i** = 1..**n**. En el grafo de abajo, se muestra una arista entre un nodo **i** y otro nodo **j** sí y sólo si **V[i, j]** = **true**.

Grafo **V** (visibilidad entre localizaciones)



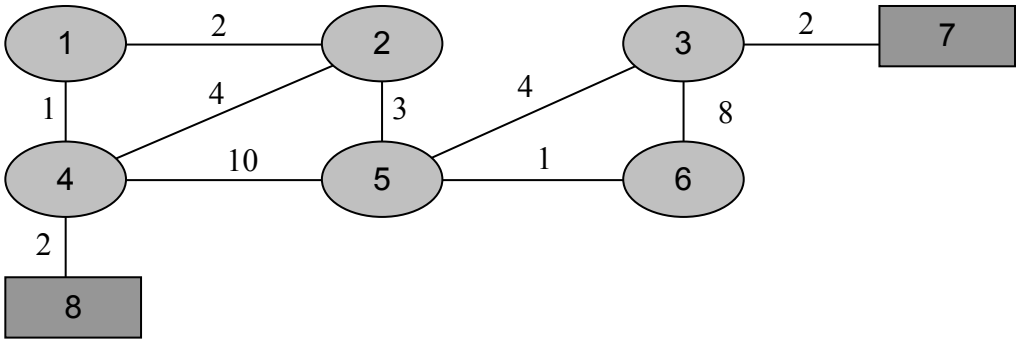
7.21. (EX D02) La experimentación con ratones es básica para crear nuevos fármacos, que resultan en importantes mejoras en la calidad de vida de los roedores. En un experimento con fármacos cerebrales, se colocan ratones dentro de un laberinto formado por **n** celdas, **m** salidas del laberinto y varios pasadizos entre las celdas (o las salidas). Las celdas son numeradas de **1** a **n** y las salidas de **n+1** a **n+m**. En cada pasadizo los ratones tardan un tiempo dado, que llamaremos **P[i, j]**: tiempo de ir de la celda **i** a la celda (o salida) **j**. Se supone que el tiempo que tardan en las celdas es despreciable.

El experimento consiste en colocar inicialmente un ratón en cada celda, y contar el número de ratones que son capaces de salir del laberinto en un tiempo dado $\mathbf{t_{max}}$. Suponiendo que los ratones eligen siempre el camino más corto hacia alguna de las salidas, las preguntas a resolver son: i) ¿Cuántos ratones habrán salido en el instante dado $\mathbf{t_{max}}$? ii) ¿Cuánto tiempo se necesita para conseguir que todos los ratones hayan salido?

Se pide:

a) Dar un algoritmo para resolver los dos problemas anteriores. Se supone que el instante inicial es **t**= 1. Hacer una estimación aproximada del orden de complejidad del algoritmo. No se pueden dar por supuestos los algoritmos vistos en clase, aunque sí las implementaciones de los tipos de datos lista y grafo.

b) Aplicar el algoritmo al siguiente ejemplo, donde **n**= 6, **m**= 2 y $\mathbf{t_{max}}$= 5. Las celdas han sido representadas con elipses, las salidas con rectángulos y los pasadizos con aristas etiquetadas. Además, en este caso se supone que **P[i, j]** = **P[j, i]**.



7.22. (EX J04) En un concurso de programación hay **m** participantes que se organizan en equipos de 3 personas o menos. Cada persona tiene una cierta afinidad con cada uno de los otros participantes (**A[i, j]** afinidad de la persona **i** hacia la **j**, no necesariamente simétrica y que puede ser negativa). El objetivo es organizar los equipos de manera que se maximice la suma de afinidades entre los miembros de los equipos formados.

Diseñar un algoritmo eficiente (con orden de complejidad polinómico) que encuentre una buena solución al problema, aunque no sea necesariamente la óptima. Aplicarlo al ejemplo de abajo, con **m**= 5.

A	**1**	**2**	**3**	**4**	**5**
1	-	1	3	6	3
2	3	-	-12	-8	1
3	2	8	-	1	2
4	1	4	3	-	0
5	2	1	2	-2	-

Resolver también el problema con un algoritmo que garantice la solución óptima, aunque no sea muy eficiente.

7.23. (EX J05) Un alumno tiene que aprobar un conjunto de asignaturas, con sus exámenes, prácticas, entrevistas y demás. En total tiene que hacer **n** tareas. Para cada una de ellas estima que le llevará cierto tiempo, $\mathbf{t_i}$. Como está un poco apurado de tiempo, se las piensa repartir entre las convocatorias de junio, septiembre y diciembre. En cada convocatoria puede sacar **M** unidades de tiempo. Se supone que todas las tareas se deben hacer antes o después, que las tareas no se pueden fraccionar (cada tarea va a una sola convocatoria), y que la suma de tiempos asignados a cada convocatoria no puede superar **M**.

El objetivo es conseguir un reparto haciendo las tareas cuanto antes, en vez de dejarlas para el final. Es decir, minimizar el tiempo dedicado en la convocatoria

de diciembre. En caso de empate en diciembre, minimizar el tiempo en la de septiembre.

Resolver el problema mediante algoritmos voraces, aunque no se garantice siempre la solución óptima. Proponer y contrastar por lo menos dos criterios de selección diferentes. Aplicar el algoritmo al siguiente ejemplo: **n** = 5, **M** = 10, **t** = {5, 3, 1, 3, 4}.

7.24. (EX S05) El Sudoku se ha convertido en el pasatiempo del verano. Se trata de una matriz de 9x9 celdas, donde en cada celda puede haber un número entre 1 y 9. Se añade la restricción de que cada número sólo puede aparecer una vez en cada fila, en cada columna y en cada bloque de celdas. Los bloques son grupos de 3x3 celdas adyacentes, como se muestran abajo (ver que, en total, hay 9 filas, 9 columnas y 9 bloques). Algunas celdas del Sudoku están inicializadas con ciertos valores y otras están vacías. El objetivo es completar las celdas vacías, con valores que cumplan las restricciones del juego.

Se pide diseñar un algoritmo voraz para intentar resolver el problema. Indicar cuáles son los candidatos, cómo es la función de selección, cómo es la inserción de un elemento en la solución, y las demás partes del esquema. Suponer que la entrada es una matriz de 9x9, donde un valor 0 indica una celda no inicializada.

	3		9	5	2	7		
6	5			8	1	2		
							5	
		8			6	4		7
	4	6	1		5	8	9	
7		3	2			5		
	8							
		9	8	2			1	5
		5	4	6	3		8	

Sudoku de ejemplo

http://websudoku.com

7.25. (EX) En un tablero de ajedrez de tamaño **a** x **b** queremos colocar el mayor número posible de reinas. Como en el problema clásico de "las **n** reinas", no queremos que las reinas se puedan comer unas a otras. Pero, a diferencia de ese problema, permitimos que cada reina esté amenazada como máximo por otras dos reinas. Resolver el problema mediante un algoritmo voraz, encontrando una buena solución, aunque no se garantice la óptima.

7.26. (EX) Un campesino quiere hacer un corral para guardar los animales de su opá. Para ello dispone de **n** tablones, que formarán las paredes del corral. Cada tablón tiene cierta longitud, $\mathbf{l} = (\mathbf{l}_1, \mathbf{l}_2, ..., \mathbf{l}_n)$. El corral debe tener forma rectangular. Aunque los tablones se pueden partir, nos interesa no partirlos mucho.

El objetivo del problema es maximizar el área total del corral, es decir, el área del rectángulo construido. Diseñar y programar un algoritmo voraz que encuentre la solución óptima para el problema, suponiendo que los tablones se pueden partir como máximo 3 veces en total. Mostrar la solución del algoritmo para el siguiente caso: **n**= 7, **l**= (3, 5, 1, 8, 2, 4, 5).

7.27. (EX J06) Disponemos de un tablero de juego que consta de **n** casillas consecutivas (tipo juego de la oca). Cada casilla tiene asociado un valor entero (positivo o negativo). Un jugador parte de una casilla inicial (casilla uno) y avanza a través de las casillas según la tirada de un dado. Cada tirada del dado es un valor entre 1 y 6. El juego termina cuando el jugador sobrepasa la casilla **n**. La puntuación obtenida es igual a la suma de los valores asociados a las casillas en las que el jugador ha ido cayendo con las tiradas del dado. El propósito del juego es obtener la mayor puntuación posible.

Diseñar un algoritmo voraz que encuentre una secuencia de tiradas con la mejor puntuación posible. Aplicar el algoritmo al siguiente ejemplo: **n**= 15, **T**= (1, 3, 5, -4, -3, 8, 2, -1, -8, -1, -7, -2, -6, -3, 2). El algoritmo diseñado, ¿garantiza la solución óptima?