Problemas intratables

Joaquim Borges
Robert Clarisó
Ramon Masià
Jaume Pujol
Josep Rifà
Joan Vancells
Mercè Villanueva

PID_00215269

Universitat Oberta de Catalunya

Índice

Introducción

En los cinco primeros módulos de esta asignatura se han estudiado un gran número de algoritmos para resolver una gran diversidad de problemas. La mayoría de problemas se han podido resolver de forma eficiente, pero también hay problemas para los cuales no se ha sabido encontrar un algoritmo eficiente (por ejemplo, el problema de encontrar un ciclo hamiltoniano en un grafo). Así pues, los algoritmos son una técnica muy potente para resolver problemas, pero su potencia no es ilimitada.

En el módulo "Complejidad computacional", se ha estudiado la complejidad de los problemas, lo que ha permitido clasificarlos en clases, y se ha visto que hay problemas que no se pueden resolver algorítmicamente de forma eficiente (no tienen o no se conocen algoritmos de complejidad polinómica que los puedan resolver). El estudio de las clases de complejidad de los algoritmos ha permitido introducir el concepto de problema intratable como aquel problema para el cual no se conoce ningún algoritmo de complejidad polinómica que lo resuelva.

Dentro de los problemas intratables, es especialmente interesante la clase de los problemas NP-Completos. Estos son problemas intratables tan difíciles como cualquier otro problema en NP. En este módulo se estudiarán en profundidad algunos problemas NP-Completos que permitirán entender mejor esta clase. Además, también se verá que los problemas NP-Completos son muy diversos y están presentes en una gran variedad de campos de la ciencia y la ingeniería.

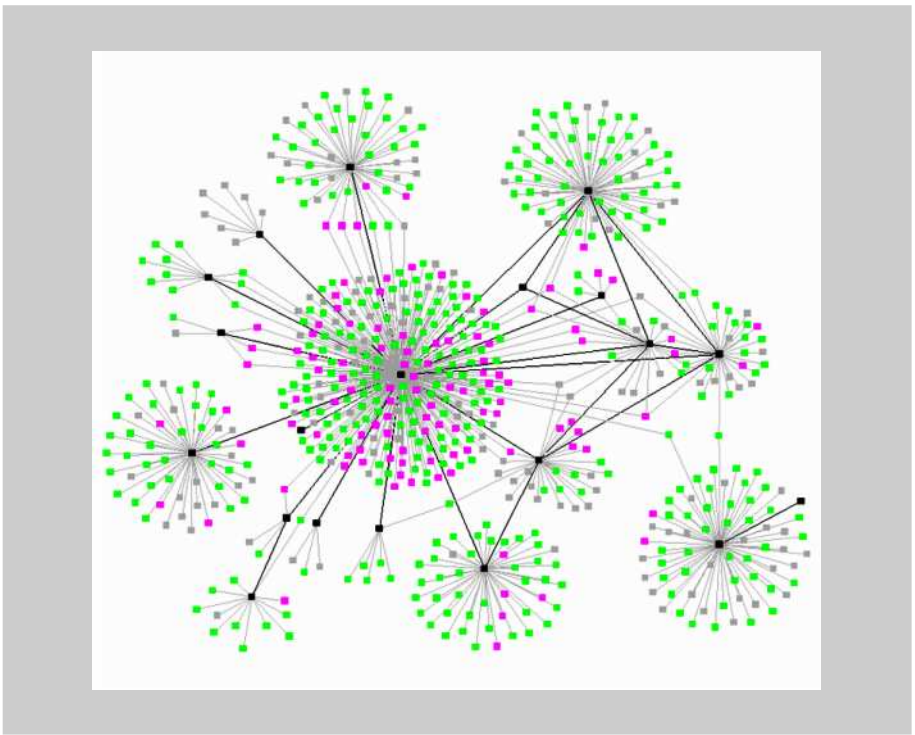
Finalmente, también se verá que hay problemas interesantes y muy útiles que no son NP-Completos. De hecho, uno de ellos (el de la primalidad) se ha podido demostrar recientemente que está en P. El otro (el de la factorización) se sabe que está en NP pero no se sabe (aunque se cree que no) si está en P o es NP-Completo.

1. Problemas intratables sobre grafos

1.1. Subgrafos completos

Las redes sociales (Facebook, Twitter, Tuenti,...) son sistemas de relaciones sociales, formados por personas, organizaciones o entidades, que facilitan la comunicación entre sus miembros. La mayoría de estas redes son dinámicas, crean y eliminan enlaces entre entidades de forma dinámica.

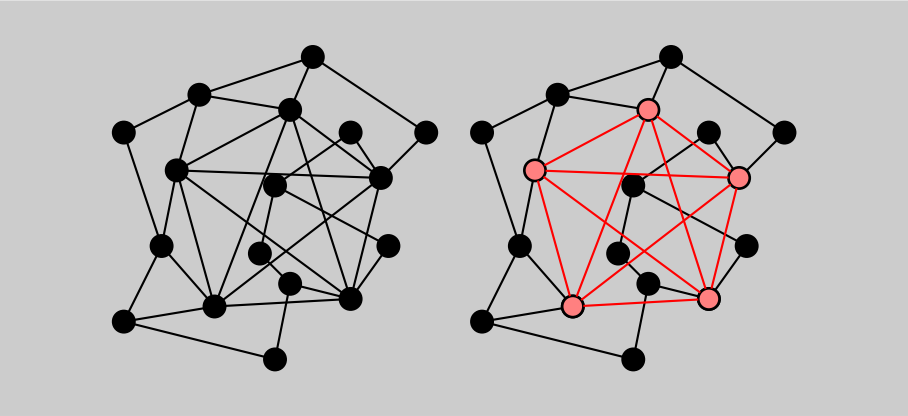
Estas redes se pueden modelar mediante un grafo donde los vértices representan las entidades y las aristas son diferentes tipos de relaciones. Conceptos de la teoría de grafos como por ejemplo el grado de un vértice, conectividad, caminos mínimos, subgrafos,... tienen su traducción inmediata en términos de redes sociales. Por ejemplo, el grado de un vértice representa el número de relaciones de una entidad determinada. Un subgrafo completo es un **grupo cohesionado**, es decir, un conjunto de personas o entidades relacionadas entre sí.



La cohesión de la red es un concepto que relaciona las entidades o grupos de entidades que tienen relaciones mutuas. Cuanto más cohesionado es un grupo, más relaciones hay entre los miembros del mismo. Todos los miembros de una empresa, o todos los alumnos

de una universidad podrían ser ejemplos de grupos cohesionados. Uno de los problemas que se puede plantear sobre la cohesión de la red es averiguar cuál es el grupo cohesionado que tiene más miembros.

En el gráfico de la izquierda de la figura siguiente se puede ver un modelo reducido de una red y a su derecha, en rojo, el grupo cohesionado que tiene más miembros.



1.1.1. Descripción del problema

Problema de optimización 1

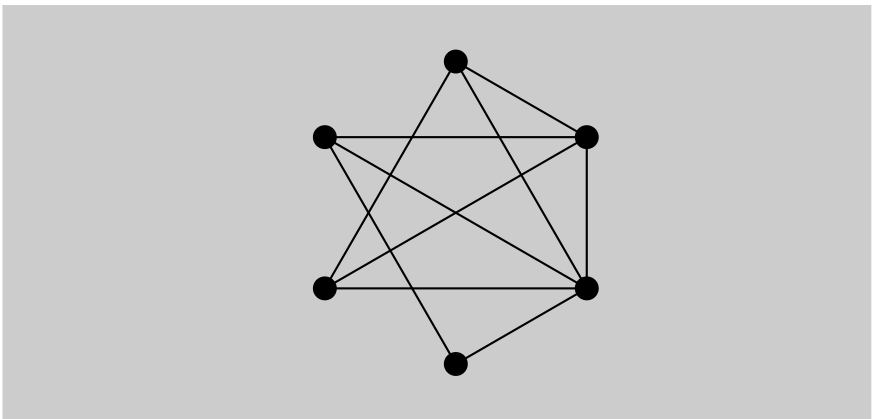
Dado un grafo $G = (V,A)$ ¿cuál es el subconjunto más grande $S \subseteq V$ tal que para cada pareja de vértices $u,v \in S$, la arista $\{u,v\} \in A$? De forma equivalente, ¿cuál es el subgrafo completo más grande de G?

Clique

Un subgrafo completo de G se llama en inglés **clique**.

Ejemplo 1

El grafo siguiente contiene subgrafos completos de orden 1, 2, 3 y 4. En este caso la solución al problema sería K_4.



Ejemplo 2

En un árbol, la solución al problema de encontrar el subgrafo completo más grande es K_2. En el grafo ciclo, C_n ($n > 3$), también es K_2. En el grafo rueda, R_n ($n > 3$), es K_3.

Ejercicios

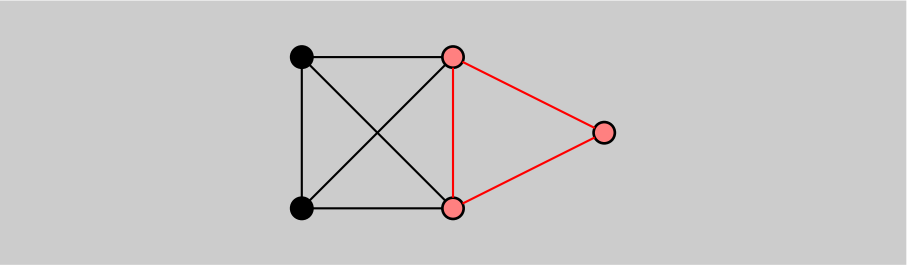
1. ¿Cuál es la solución al problema del subgrafo completo más grande en un grafo bipartito conexo? ¿Y en su complementario?

2. Dado un grafo $G = (V,A)$ de orden n con un subgrafo completo K_r ($r < n$) tal que si añadimos a K_r cualquier vértice $u \in V$ ($u \notin K_r$) no obtenemos un subgrafo completo mayor, ¿podemos afirmar que K_r es el subgrafo completo más grande de G?

Soluciones

1. En un grafo bipartito todos los vértices de los dos conjuntos que forman la partición no tienen ninguna arista en común. Por lo tanto, el subgrafo completo mayor será K_2. En su complementario, cada subconjunto de vértices de la partición será un subgrafo completo. Así, la solución será el subconjunto de la partición que tenga cardinal mayor.

2. La afirmación no es cierta, como lo demuestra el ejemplo siguiente:



Los vértices en rojo forman un subgrafo completo K_3, pero añadiendo cualquiera de los otros dos vértices no obtenemos un subgrafo completo más grande. La solución al problema es, evidentemente, K_4.

1.1.2. Discusión

La búsqueda del subgrafo completo más grande en un grafo G de n vértices es un problema de optimización y el correspondiente problema de decisión (dado un número k, decidir si hay algún subgrafo completo con k o más vértices) es un problema NP-Completo.

Búsqueda exhaustiva

La búsqueda exhaustiva es un método de diseño de algoritmos que consiste en generar todos los elementos de un conjunto para encontrar aquel que cumple una determinada condición.

La única solución exacta consiste en hacer una búsqueda exhaustiva sobre todos los subconjuntos de k vértices de G ($k \leq n$). Dado que el grado máximo de los vértices de G siempre es menor o igual que $n - 1$, podemos empezar cogiendo como referencia este grado máximo. El algoritmo siguiente calcula el subgrafo completo más

grande de G utilizando la búsqueda exhaustiva sobre el conjunto de todos los subconjuntos de vértices de G:

```
Entrada : G = (V,A)
Salida : s, conjunto de vértices del subgrafo completo más grande
algoritmo SubgrafoCompletoMayor(G)
  inicio
    k ← max{g(u) : u ∈ V} + 1
    encontrado ← FALSO
    mientras k ≥ 2 ∧ ¬encontrado
              S ← [subconjuntos de cardinal k de V]
              mientras S ≠ ∅ ∧ ¬encontrado
                        s ← Primero(S)
                        si SubgrafoCompleto(s)
                           entonces encontrado ← CIERTO
                                sino Eliminar(S,s)
                        finsi
              finmientras
              k ← k – 1
    finmientras
    retorno (s)
  fin
```

La complejidad de este algoritmo viene determinada por el número de subconjuntos de cardinal k de V. Dado que hay $\binom{n}{k}$ subconjuntos de cardinal k, la complejidad sería polinómica si k fuera constante. Pero, como que en el peor de los casos tenemos que calcular todos los subconjuntos de cardinal k, $k = 0,1,\ldots,n$, entonces la complejidad es del orden de $O(2^n)$ que es no polinómica.

Subconjuntos de un conjunto

Recordad que el número total de subconjuntos de un conjunto de cardinal n es 2^n. Este resultado también es consecuencia de la fórmula $\sum_{k=0}^{n} \binom{n}{k} = 2^n$.

Ejemplo 3

El grafo $K_4 \times T_2$ tiene orden 8 y medida 16. El grado máximo es 4, por lo tanto el algoritmo primero comprobaría los $\binom{8}{5} = 56$ subconjuntos de 5 vértices y después los $\binom{8}{4} = 70$ subconjuntos de 4 vértices hasta encontrar el subgrafo completo más grande que es K_4.

La prueba que este problema es NP-Completo la podemos hacer en tres pasos:

1) Definir el problema de decisión asociado.

2) Comprobar que este problema está en NP, es decir, que se puede verificar en tiempo polinómico.

3) Comprobar que cualquier problema en NP se puede reducir polinómicamente a nuestro problema, es decir, que nuestro problema es NP-Difícil.

Vamos a desarrollar cada uno de estos pasos.

1) El problema de decisión asociado se puede formular de la manera siguiente:

Problema de decisión 2

CLIQUE: Dado un grafo $G = (V,A)$ de orden n y un número entero k ($1 \leq k \leq n$), se tiene que determinar si existe un subgrafo completo (*clique*) de G de orden k.

2) Para hallar su complejidad, la medida de la entrada de este problema está determinada por el orden del grafo, n.

Para comprobar que el problema de decisión CLIQUE está en NP, es suficiente verificar, en tiempo polinómico, si, dado un subconjunto de k vértices del grafo, este subconjunto es un subgrafo completo.

Obviamente, fijados los k vértices, esta comprobación se puede hacer con una complejidad $O(k^2)$.*

*** Ved el ejercicio 10 de este módulo.**

3) El tercer paso consiste en encontrar un problema NP-Completo reducible polinómicamente a CLIQUE.

El problema NP-Completo escogido es el 3SAT, que se caracteriza porque cada cláusula de la fórmula booleana contiene tres literales.

Fórmula booleana

Recordad que una fórmula booleana es una combinación de variables lógicas o booleanas ($x,y,z,\ldots$ que toman valores 0 y 1) y las operaciones lógicas $\vee$, $\wedge$, $\bar{x}$.

Ejemplo 4

Una fórmula booleana en forma normal conjuntiva (FNC) sería, por ejemplo, la siguiente:

$$x \wedge (\bar{y} \vee z).$$

Añadiendo variables auxiliares, esta fórmula se puede convertir en una fórmula booleana en la que cada cláusula contiene tres literales:

$$(x \vee v \vee w) \wedge (x \vee v \vee \bar{w}) \wedge (x \vee \bar{v} \vee w) \wedge (x \vee \bar{v} \vee \bar{w}) \wedge (\bar{y} \vee z \vee u) \wedge (\bar{y} \vee z \vee \bar{u})$$

La reducción del problema 3SAT al problema CLIQUE (3SAT $\leq_p$ CLIQUE) consiste en transformar cada fórmula booleana formada por k cláusulas de tres literales en un grafo $G = (V,A)$ de orden $3k$. Esta transformación se tiene que hacer de tal manera que la fórmula booleana se satisface si y sólo si el grafo contiene un subgrafo completo de orden k.

Observación

Recordad que una fórmula booleana se satisface si existe una asignación de valores 0 o 1 a cada variable de forma que, una vez evaluada la expresión, el valor resultante sea 1.

La construcción del grafo es la siguiente. Para cada cláusula $(x \vee y \vee z)$ definimos los vértices $x,y,z \in V$, y dos vértices $u,v \in V$ son adyacentes si:

1) u y v representan variables booleanas de cláusulas diferentes, y

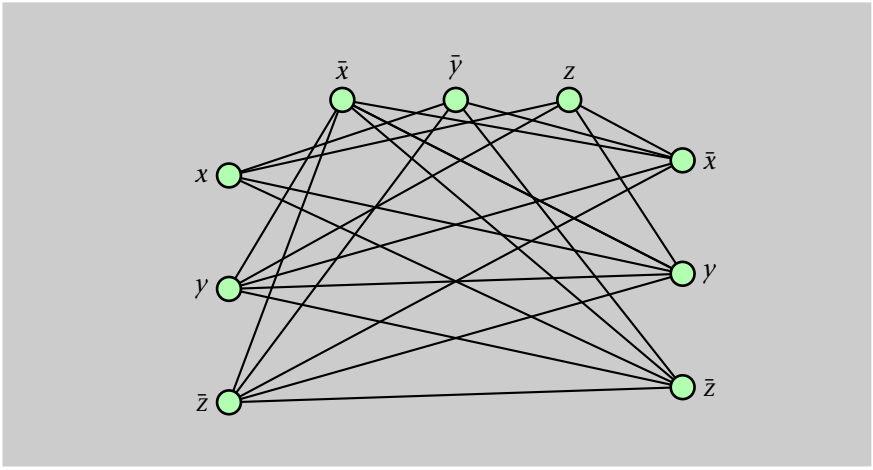
2) las variables booleanas representadas por u y v no son complementarias.

Ejemplo 5

Consideremos la fórmula booleana

$$(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee \bar{z}).$$

La transformación definida da lugar al siguiente grafo:



En este grafo, los tres vértices de la izquierda corresponden a la primera cláusula, los tres vértices superiores a la segunda cláusula y los tres vértices de la derecha a la tercera cláusula.

Para que esta transformación sea una reducción, hay que comprobar que la transformación se puede hacer en tiempo polinómico y que a cada asignación de las variables que satisface la fórmula booleana le corresponde un subgrafo completo de orden k.

La transformación se puede hacer en tiempo polinómico, dado que podemos construir un grafo de $3k$ vértices en tiempo polinómico.

Si hay una asignación de las variables que satisface la fórmula, entonces en cada cláusula habrá una variable (o más de una) que tendrá valor 1. Escogemos una variable de cada cláusula con valor 1 y su correspondiente vértice en el grafo. Los k vértices del grafo forman un subgrafo completo, puesto que dos vértices cualesquiera de este subgrafo representan variables de diferentes cláusulas que no pueden ser complementarias.

Recíprocamente, si escogemos un subgrafo completo de k vértices, entonces cada uno de estos k vértices estará en una cláusula dife-

rente. Si asignamos el valor 1 a la variable correspondiente de la cláusula que representa, esta asignación de las variables satisface la fórmula booleana.

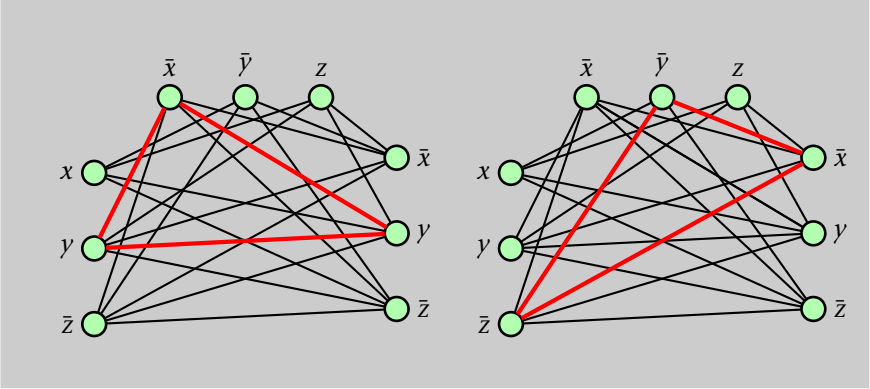
Ejemplo 6

Siguiendo con la fórmula booleana del ejemplo 5 que contiene 3 cláusulas

$$(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee \bar{z}),$$

el grafo obtenido por la transformación contendrá un subgrafo completo de orden 3 para cada posible asignación que satisface la fórmula.

Las asignaciones de las tres variables (x,y,z): (0,1,0), (0,0,0) satisfacen la fórmula booleana y corresponden a los subgrafos completos de orden 3 siguientes:



Ejercicios

3. Comprobar que la asignación $x = 1$, $y = 0$, $z = 0$ satisface la fórmula booleana:

$$(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee \bar{z})$$

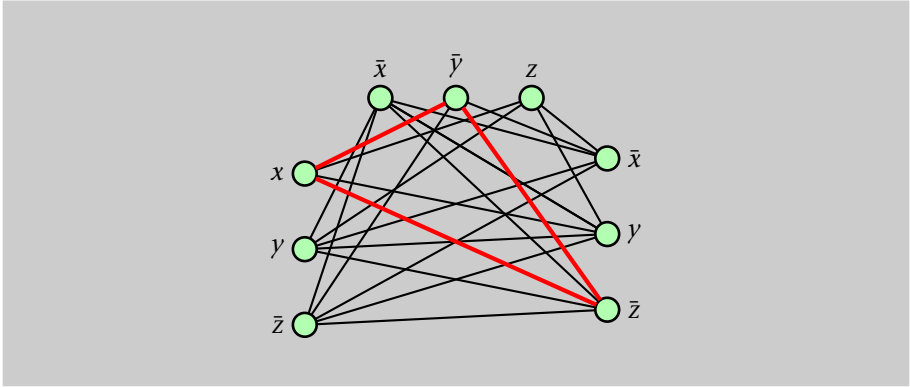
y que el subgrafo generado es un subgrafo completo de orden 3.

Soluciones

3. Sustituyendo los valores de las variables en la fórmula obtenemos:

$$(1 \vee 0 \vee 1) \wedge (0 \vee 1 \vee 0) \wedge (0 \vee 0 \vee 1) = 1.$$

El subgrafo generado es:



1.2. Conjuntos dominantes de vértices

Dado un grafo $G = (V,A)$, a veces nos interesará encontrar un conjunto de vértices $S \subseteq V$ "representantes", de forma que cualquier otro vértice sea adyacente a alguno(s) de los vértices de S. Por ejemplo, en una red social, nos podría interesar determinar un conjunto de entidades, de forma que cualquier entidad de la red esté en este conjunto o bien esté relacionada con alguna de las entidades de este conjunto.

1.2.1. Descripción del problema

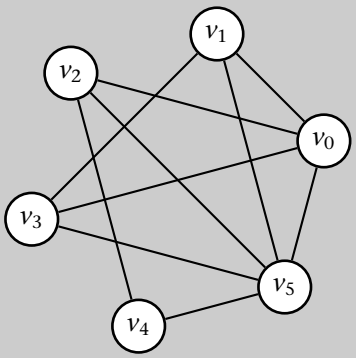
Definición 1

Un **conjunto dominante** de un grafo $G = (V,A)$ es un subconjunto $S \subseteq V$ tal que cada vértice de G se encuentra en S o es adyacente al menos a un vértice de S.

Naturalmente, lo más interesante será encontrar conjuntos dominantes con el menor número de vértices posible.

Ejemplo 7

En el grafo siguiente los vértices $\{v_1,v_2,v_3\}$ forman un conjunto dominante del grafo. Pero un conjunto dominante con el menor número de vértices podría ser $\{v_1,v_2\}$ o $\{v_2,v_3\}$.



Ejemplo 8

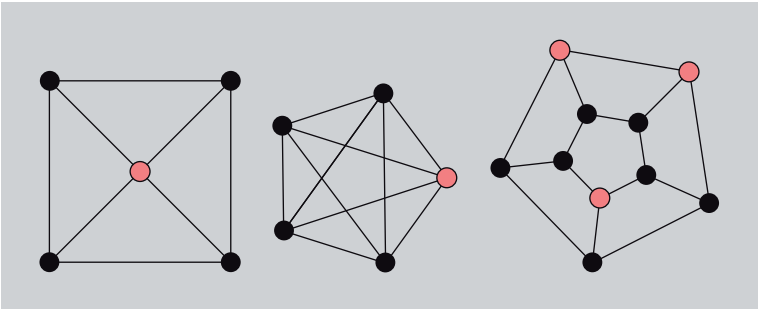
En el grafo completo K_n, cualquier vértice forma un conjunto dominante. En el grafo nulo N_n, en cambio, sólo hay un conjunto dominante que es el conjunto que contiene todos los vértices del grafo.

Problema de optimización 3

Dado un grafo $G = (V,A)$, ¿cuál es el subconjunto más pequeño $S \subseteq V$ tal que para cada vértice $x \in V$, se verifica que $x \in S$, o bien x es adyacente a algún(os) vértice(s) de S? Equivalentemente, ¿cuál es el conjunto dominante mínimo de G?

Ejemplo 9

En rojo se pueden ver conjuntos dominantes mínimos de cada uno de los tres grafos:



Para encontrar conjuntos dominantes mínimos, podemos trabajar con la matriz de adyacencia B del grafo G.

En este caso, es conveniente poner unos en la diagonal de B (consideramos que cada vértice es adyacente a si mismo). De este modo, encontrar un conjunto dominante mínimo equivale a encontrar un conjunto de columnas mínimo de forma que cada fila tenga un 1 en alguna de las columnas que hemos elegido.

El problema puede generalizarse, si pensamos en una matriz no necesariamente cuadrada y con costes asociados a las columnas. Entonces, se denomina *problema de cobertura* al problema de optimización que consiste en encontrar un conjunto de columnas de coste mínimo que contiene al menos un 1 para cada una de las filas. Un caso particular es el llamado *problema de partición de una matriz* dónde, además, se exige que ninguna fila sea cubierta más de una vez, es decir, que dos columnas de la solución no pueden tener unos en la misma fila.

Para estos problemas, además de generar todas las posibles soluciones para buscar la óptima, hay otros algoritmos basados en la técnica del *backtracking* para buscar la solución óptima. Pero más que estudiar estos algoritmos, vamos a ver algunas posibles simplificaciones que se puede intentar hacer en la matriz de un proble-

Backtracking

El backtracking es un método de diseño de algoritmos que consiste en generar una solución parcial del problema y evaluarla por si puede formar parte de la solución general. Si la evaluación es negativa, se descarta la solución parcial y se genera otra. Si la evaluación es positiva, se continúa con la solución parcial hasta encontrar la solución general.

ma de cobertura. Llamamos $R = \{R_1, \ldots, R_n\}$ al conjunto de filas y $S = \{S_1, \ldots, S_m\}$ al conjunto de columnas. Entonces:

1) Si hay alguna fila R_i que tiene todo ceros, entonces el problema no tiene solución.

2) Si R_i tiene un único 1 en la columna S_j, entonces S_j tiene que formar parte de la solución y podemos eliminar R_i y todas las otras filas cubiertas por S_j.

3) Si una fila R_j cubre R_i (es decir, en todas partes donde R_i tiene un 1, R_j también tiene un 1), entonces podemos eliminar la fila R_j, dado que todo conjunto de columnas que cubra R_i también cubrirá R_j.

4) Si las filas cubiertas por S_j de coste c_j pueden ser cubiertas por un grupo de columnas que tienen, entre todas, un coste inferior a c_j, entonces podemos eliminar la columna S_j.

Ejemplo 10

Vamos a resolver el problema de cobertura para la siguiente matriz:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
R_1	1	1	1	0	0	0	0	0
R_2	0	1	1	1	1	0	0	0
R_3	1	1	1	1	1	0	0	0
R_4	1	0	0	0	0	1	1	0
R_5	0	0	0	0	1	1	0	1
R_6	1	0	0	0	0	0	0	0
Coste	7	9	20	4	9	3	5	2

En primer lugar, podemos observar que la fila R_6 tiene un único 1 en la columna S_1, entonces S_1 formará parte de la solución y ya podemos borrar R_1, R_3, R_4 y R_6 que son cubiertas por S_1. Entonces nos queda la matriz:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
R_2	0	1	1	1	1	0	0	0
R_5	0	0	0	0	1	1	0	1
Coste	7	9	20	4	9	3	5	2

Fijémonos que ahora la columna S_7 no nos cubre ninguna de las filas restantes, por lo tanto, podemos eliminarla. Las columnas S_2, S_3 y S_4 nos cubren, cada una de ellas, la fila R_2. Dado que S_4 tiene coste menor, ya podemos eliminar las columnas S_2 y S_3. Pasa algo similar con las columnas S_6 y S_8, puesto que las dos cubren la fila R_5. Cómo S_8 tiene coste menor, eliminamos S_6. Después de esto, nos queda:

	S_1	S_4	S_5	S_8
R_2	0	1	1	0
R_5	0	0	1	1
Coste	7	4	9	2

Finalmente, puesto que S_5 tiene coste 9 y la suma de los costes de S_4 y S_8 es 6, eliminaríamos S_5. Ahora ya nos quedan las dos filas con un único 1 cada una y, por lo tanto, S_4 y S_8 tienen que formar parte de la solución. Así pues, la solución final es $\{S_1, S_4, S_8\}$ con coste $7 + 4 + 2 = 13$.

En el anterior ejemplo, con las simplificaciones nos ha quedado una matriz muy simple y ha sido fácil encontrar la solución óptima pero, naturalmente, esto no pasa en el caso general.

1.2.2. Discusión

Volviendo ahora al problema original de encontrar un conjunto dominante mínimo en un grafo $G = (V,A)$, podemos observar que es un caso particular del anterior problema de cobertura donde la matriz es cuadrada, con unos en la diagonal y con todos los costes iguales. Se trata de un problema de optimización y veremos que el correspondiente problema de decisión es NP-Completo.

1) El problema de decisión podría enunciarse cómo:

Problema de decisión 4

SET_COVER: Dado un grafo $G = (V,A)$ y un número entero k $(1 \leq k \leq n)$, determinar si existe un conjunto dominante en G que tenga k vértices.

2) Para comprobar que el problema de decisión SET_COVER está en NP, es suficiente comprobar, en tiempo polinómico, si dado un conjunto S de k vértices, éste es un conjunto dominante.

Esta comprobación puede hacerse calculando la unión de S y sus adyacentes y comprobando que obtenemos todos los vértices V del grafo. Evidentemente esto se puede hacer con complejidad polinómica respecto del número de vértices del grafo.

3) Podemos reducir polinómicamente el problema 3SAT al problema SET _COVER (3SAT $\leq_p$ SET_COVER) de la siguiente forma:

Sea $U = \{u_1, \ldots, u_n\}$ el conjunto de variables del problema 3SAT y sea $C = \{C_1, \ldots, C_m\}$ el conjunto de cláusulas.

Para cada variable u_i del 3SAT, construimos un triángulo con vértices u_i, $\bar{u}_i$ y v_i. Para cada cláusula $C_j = u_i \vee u_k \vee u_l$, creamos un vértice C_j con aristas $\{u_i,C_j\}$, $\{u_k,C_j\}$ y $\{u_l,C_j\}$. El grafo G construido tiene $3n + m$ vértices. Vamos a ver que el problema 3SAT se satisface si y sólo si G tiene un conjunto dominante S de n vértices.

Supongamos primero que el problema 3SAT se satisface, es decir, que 3SAT(x) = SÍ, entonces definimos S como el conjunto de literales que tienen asignado un 1. Es decir, para cada $i = 1, \ldots, n$; o bien

u_i, o bien $\bar{u}_i$ estará en S. Cada triángulo, con vértices u_i, $\bar{u}_i$ y v_i, tiene exactamente un vértice en S y dado que cada cláusula debe contener un literal con valor 1, todos los vértices C_j serán adyacentes a algún vértice de S.

Recíprocamente, si S es un conjunto dominante con n vértices de G, entonces cada triángulo tendrá que tener al menos un vértice en S, puesto que tenemos n triángulos, en cada triángulo habrá exactamente un vértice de S. Los vértices cláusula C_j serán adyacentes a algún(os) vértice(s) de S. Asignamos el valor 1 a los vértices (literales) de S y 0 a los que no son de S. Claramente, toda cláusula contiene algún literal con valor 1 y, por lo tanto, el problema 3SAT se satisface.

Ejercicios

4. Comprobar que la asignación $x = 1$, $y = 0$, $z = 1$ satisface la fórmula booleana:

$$(x \vee y \vee \bar{z}).$$

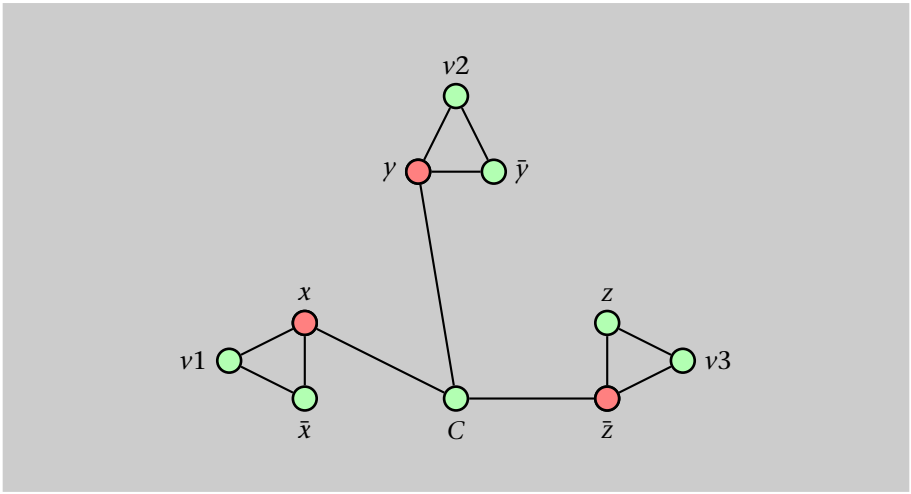
Dibujar el grafo asociado y comprobar que tiene un conjunto dominante de 3 vértices.

Soluciones

4. Substituyendo los valores de las variables en la fórmula obtenemos:

$$1 \vee 0 \vee 0 = 1.$$

El grafo asociado es el siguiente, dónde hemos dibujado en rojo los vértices del conjunto dominante.



El problema de los conjuntos dominantes de vértices tiene aplicaciones en el ámbito de la logística. Por ejemplo, nos puede interesar determinar un número mínimo de almacenes o centros médicos que den servicio a un conjunto de poblaciones.

1.3. Recubrimientos de vértices

Una variante del problema de la dominación es la siguiente. Se quiere escoger el grupo más pequeño de entidades que, dada una relación (arista) cualquiera de la red, contenga al menos una de las dos entidades (vértice) que tienen esta relación. Ya se ve que el interés estará en elegir entidades que tengan muchas relaciones para tener que seleccionar un mínimo número de entidades. Dicho de otro modo, dado un grafo $G = (V,A)$, queremos encontrar un conjunto de vértices $S \subseteq V$, de forma que cualquier arista $\{x,y\}$ verifica que $x \in S$ o bien $y \in S$.

1.3.1. Descripción del problema

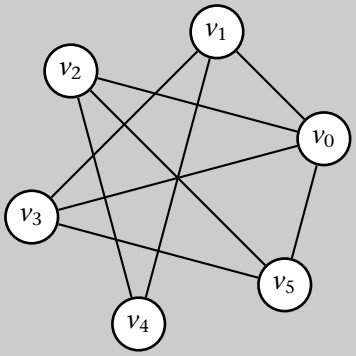
Definición 2

Un **recubrimiento** de los vértices de un grafo $G = (V,A)$ es un subconjunto $S \subseteq V$ tal que para cada $\{x,y\} \in A$ tenemos que $x \in S$ o $y \in S$.

Naturalmente, lo más interesante será encontrar recubrimientos con el menor número posible de vértices.

Ejemplo 11

En el grafo siguiente el conjunto de vértices $S_1 = \{v_0,v_1,v_2,v_3,v_5\}$ forman un recubrimiento de los vértices del grafo que no es mínimo. El conjunto $S_2 = \{v_0,v_1,v_2,v_3\}$ es un recubrimiento mínimo de los vértices del grafo.



Ejemplo 12

En un grafo bipartito $G = (V_1 \cup V_2,A)$ cada uno de los dos subconjuntos V_1, V_2 forman un recubrimiento de los vértices del grafo.

Ejemplo 13

En el grafo ciclo C_n, un recubrimiento mínimo tendrá $\frac{n}{2}$ vértices si n es par y $\frac{n+1}{2}$ si n es impar.

Ejercicios

5. Demostrar que en un grafo sin vértices aislados, todo recubrimiento es un conjunto dominante, pero no necesariamente mínimo. Por otro lado, demostrar también que no siempre un conjunto dominante es un recubrimiento de vértices.

Soluciones

5. Sea S un recubrimiento y sea $x \in V$ y $x \notin S$. Dado que G no tiene vértices aislados, sea $\{x,z\}$ una arista incidente con x. Entonces $z \in S$ y, por lo tanto, x es adyacente a un vértice de S. Es decir, el recubrimiento S también es un conjunto dominante puesto que todos los vértices $x \notin S$ están conectados a algún vértice z de S.

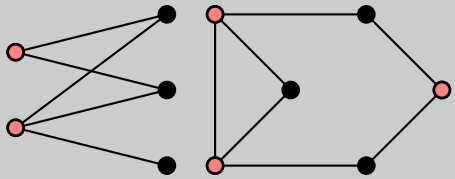
Respecto a la segunda afirmación sólo hay que considerar un contraejemplo: en K_3, un vértice cualquiera forma un conjunto dominante pero no es un recubrimiento.

Problema de optimización 5

Dado un grafo $G = (V,A)$, ¿cuál es el subconjunto más pequeño $S \subseteq V$ tal que para cada arista $\{x,y\} \in A$, se verifica que $x \in S$, o bien $y \in S$? Equivalentemente, ¿cuál es el recubrimiento mínimo de los vértices de G?

Ejemplo 14

En rojo se pueden ver los vértices de un recubrimiento mínimo en cada grafo.



Ejercicios

6. Dado un grafo $G = (V,A)$, denominamos $\beta(G)$ el *número de dominación* de G, es decir, el cardinal de un conjunto dominante mínimo de G. Denominamos $\gamma(G)$ el cardinal de un recubrimiento mínimo. Demostrar que $\gamma(G) \geq \beta(G)$.

Soluciones

6. Es consecuencia directa del ejercicio 5.

1.3.2. Discusión

De nuevo, el problema de encontrar un recubrimiento mínimo en un grafo arbitrario es un problema intratable. Su versión como problema de decisión (VERTEX_COVER) resultará ser NP-Completo.*

*** Ved el ejercicio de autoevaluación 6.**

Este problema vuelve a tener aplicaciones en el ámbito de la logística y la distribución de recursos. Por ejemplo, si queremos revisar todas las conexiones de una red interna de telefonía y tenemos que visitar al menos un extremo de cada conexión, ¿cuál es el número mínimo de centralitas de la red que hay que revisar?

1.4. Coloración de grafos

Un problema relacionado con los anteriores es el de la coloración de grafos. Este problema se plantea cuando se quiere asignar recursos (colores) a entidades (vértices) que tienen algunas relaciones de incompatibilidad entre ellos (aristas), de forma que no todos pueden compartir el mismo recurso.

Origen del término

El ejemplo clásico de coloración es el problema de pintar los países en un mapa sin repetir colores en dos países adyacentes. Cada país se representaría mediante un vértice conectado a los países adyacentes.

Lo veremos claramente en un ejemplo: supongamos que tenemos que guardar en almacenes una serie de productos químicos. Algunos de estos productos son incompatibles, de forma que los queremos en almacenes diferentes. El objetivo es encontrar el mínimo número posible de almacenes que necesitamos. Así, podríamos pensar en un grafo donde los vértices representan estos productos químicos y dos vértices son adyacentes si y sólo si los correspondientes productos químicos son incompatibles.

En este caso, queremos encontrar una partición de los vértices de forma que cada clase de la partición se pueda pintar de un color diferente, de tal manera que vértices adyacentes tengan siempre diferentes colores.

1.4.1. Descripción del problema

Definición 3

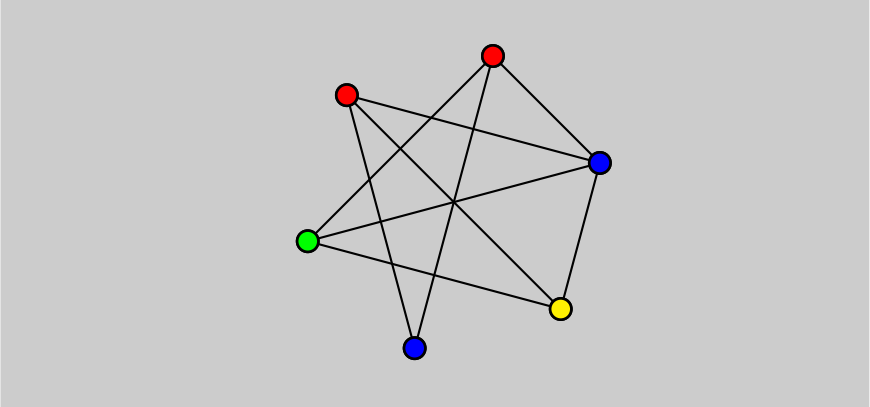
Una **coloración** de un grafo $G = (V,A)$ es una función $f : V \to C$, donde C es un conjunto finito de elementos que denominaremos colores, de forma que para toda pareja de vértices $x,y \in V$, si $\{x,y\} \in A$, entonces $f(x) \neq f(y)$.

Así pues, vértices adyacentes deben tener asignados diferentes colores. Si la imagen de f tiene cardinal r, es decir, si se utilizan como mucho r colores, se dice que G es *r-colorable*. El objetivo es encontrar el menor r para el que G es r-colorable. Este r mínimo, que denotaremos $\chi(G)$, se denomina el *número cromático* de G y también decimos que G es $\chi(G)$-*cromático*.

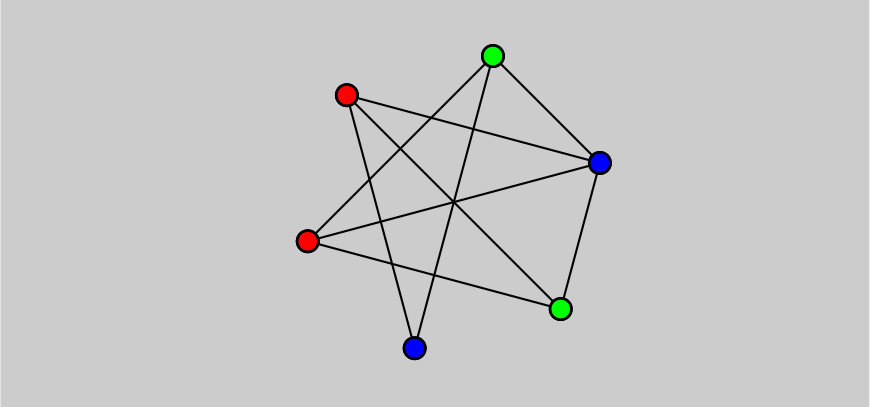
Observad que un grafo r-colorable es r-partito. Así, todo grafo k-partito se puede pintar con k colores diferentes.

Ejemplo 15

El siguiente grafo se puede pintar con 4 colores diferentes.



No obstante, también podemos encontrar una coloración con 3 colores que es el mínimo:



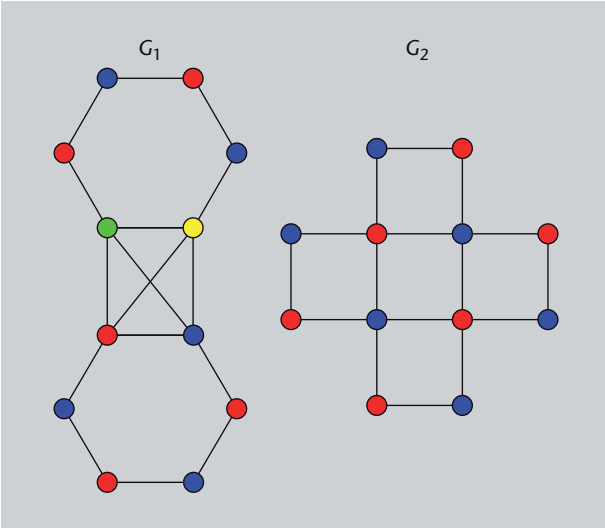
Ejemplo 16

Las siguientes afirmaciones son obvias:

- El número cromático del grafo completo K_n es $\chi(K_n) = n$.
- Un grafo es bipartito si y sólo si es 2-colorable.
- Si un grafo G contiene un subgrafo completo de t vértices, entonces $\chi(G) \geq t$.
- El número cromático del grafo ciclo C_n es $\chi(C_n) = 2$ si n es par y $\chi(C_n) = 3$ si n es impar.

Ejemplo 17

Se podría pensar que conociendo el orden, la medida y la secuencia de grados de G, se podría deducir $\chi(G)$. Pero esto es falso como se puede ver en los siguientes grafos:



Los dos grafos tienen 12 vértices y 16 aristas. En los dos casos, hay 8 vértices con grado 2 y 4 vértices con grado 4. En cambio, G_1 es 4-cromático (fijarse que contiene un *clique* de 4 vértices) mientras G_2 es 2-cromático (es un grafo bipartito).

Problema de optimización 6

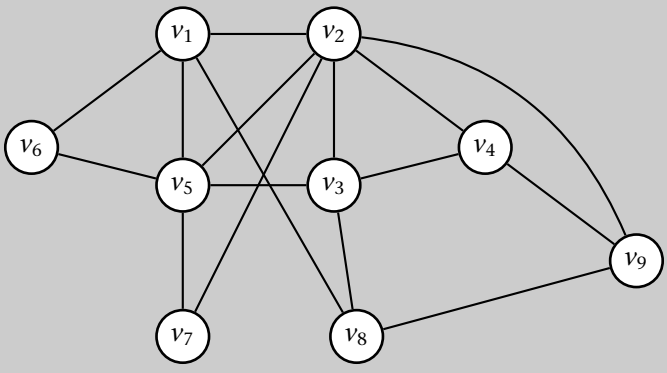
Dado un grafo $G = (V,A)$, ¿cuál es el mínimo número de colores r ($r \geq 1$) para que G sea r-colorable? Equivalentemente, ¿cuál es el número cromático $\chi(G)$ de G?

Ejemplo 18

El grafo estrella E_n es 2-cromático puesto que es bipartito. El grafo rueda R_n es 3-cromático si n es impar y 4-cromático si n es par.

Ejercicios

7. Encontrar el número cromático del grafo:



Soluciones

7. Los vértices v_1,v_3,v_7,v_9 pueden pintarse con el color c_1; los vértices v_2,v_6,v_8 con el color c_2 y los vértices v_4,v_5 con el c_3. Dado que el grafo no es bipartito, seguro que no existe ninguna 2-coloración. Así pues, es un grafo 3-cromático.

1.4.2. Discusión

Problema de decisión 7

COLORING: Dado un grafo $G = (V,A)$ de orden n y un número entero k ($1 \leq k \leq n$), determinar si G es k-colorable.

Naturalmente, encontrar una coloración óptima implica encontrar automáticamente el número cromático. El problema de decisión de sí dado un grafo existe una k-coloración es un problema NP-Completo, en general. Ver si un grafo es 1-colorable es trivial (no debe tener ninguna arista). También es fácil ver si existe una 2-coloración puesto que, como ya se ha dicho, esto es equivalente a ver si el grafo es bipartito. Pero el problema de si un grafo es 3-colorable ya es NP-Completo. Se puede encontrar la justificación en la obra de Garey y otros (1979), por ejemplo.

1.5. Ciclo hamiltoniano

Los problemas relacionados con la optimización de recorridos, es decir, la búsqueda del recorrido óptimo entre los nodos de una red (de carreteras, de ordenadores, redes sociales,...) es uno de los problemas que nos podemos encontrar de forma habitual. Los GPS contienen aplicaciones que utilizan la optimización de recorridos para buscar el mejor camino entre dos puntos. Google Maps es un ejemplo de aplicación que también utiliza la optimización de recorridos para ofrecer soluciones óptimas a la planificación de rutas.

En los módulos anteriores hemos visto algunos de estos problemas y sus soluciones. Concretamente, en el módulo "Recorridos y conectividad", hemos estudiado el problema del camino mínimo en un grafo y hemos visto que se puede resolver de forma eficiente utilizando algoritmos como el de Dijkstra o el de Floyd.

En el módulo "Grafos eulerianos y grafos hamiltonianos", hemos estudiado dos problemas muy parecidos, recorrer todas las aristas de un grafo sin repetir ninguno (circuito euleriano) y recorrer to-

dos los vértices de un grafo sin repetir ninguno (ciclo hamiltoniano). En el primer caso hemos visto que había algoritmos eficientes para resolver el problema, pero no se conoce ningún algoritmo de complejidad polinómica para encontrar un ciclo hamiltoniano. Este resultado no nos tiene que sorprender puesto que el problema de buscar un ciclo hamiltoniano en un grafo es un problema NP-Completo, como describiremos a continuación.

1.5.1. Descripción del problema

Problema de cálculo 8

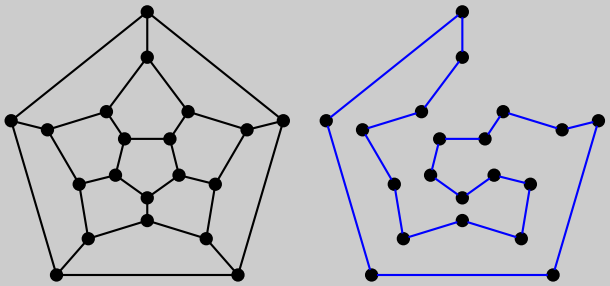
Dado un grafo $G = (V,A)$, hallar un ciclo hamiltoniano de G.

Grafo hamiltoniano

Recordad que un ciclo hamiltoniano en un grafo G es un ciclo que pasa por todos los vértices del grafo. Un grafo que contiene un ciclo hamiltoniano se denomina **grafo hamiltoniano**.

Ejemplo 19

El grafo de la izquierda es una representación de las caras del dodecaedro y el de la derecha es la representación de un ciclo hamiltoniano en este grafo:



Ejemplo 20

El grafo ciclo C_n ($n \geq 3$) contiene trivialmente un ciclo hamiltoniano. De hecho, si un grafo contiene un ciclo hamiltoniano entonces siempre podemos formar un ciclo C_n con sus vértices.

Ejercicios

8. ¿Cuántas aristas podemos eliminar en un grafo hamiltoniano sin que deje de serlo?

Soluciones

8. Supongamos que el grafo tiene orden n y medida m. Puesto que es hamiltoniano tiene que contener un ciclo formado por n vértices y n aristas diferentes. Por lo tanto, podremos eliminar $m - n$ aristas.

1.5.2. Discusión

Un método general para encontrar un ciclo hamiltoniano en un grafo $G = (V,A)$ podría consistir en enumerar todas las permutaciones de los n vértices de G y comprobar si cada permutación de los n vértices forma un ciclo hamiltoniano. Puesto que hay $n!$ permutaciones de los vértices, podemos concluir que este método da lugar a un algoritmo de complejidad no polinómica*.

*** Repasad el ejemplo 10 del módulo "Grafos eulerianos y grafos hamiltonianos".**

Como podíamos esperar, el problema de decisión asociado también es un problema NP-Completo como veremos a continuación.

1) El problema de decisión asociado será:

Problema de decisión 9

HAM_CYCLE: Dado un grafo $G = (V,A)$ de orden n, determinar si existe un ciclo hamiltoniano en G.

2) El problema HAM_CYCLE está en NP, puesto que dada una secuencia de n vértices de G, podemos verificar en tiempo polinómico que los n vértices son todos diferentes y que, añadiendo el primer vértice al final de la secuencia, forman un ciclo en G.

3) La reducción polinómica se puede hacer a partir del problema del recubrimiento de vértices (VERTEX_COVER $\leq_p$ HAM_CYCLE). No lo explicaremos aquí por su extensión, se puede encontrar en la obra de Cormen y otros (2001).

1.6. El viajante de comercio

Ya conocemos uno de los problmeas más notables de optimización de recorridos, el problema del viajante de comercio o TSP (*Traveling salesman problem*). Este problema no tiene una solución eficiente y hemos visto que, en ciertas situaciones, es posible encontrar soluciones aproximadas (*TSP-aproximado*).

Ved también

El problema del viajante de comercio o TSP (*Traveling salesman problem*) se estudia en el módulo "Grafos eulerianos y grafos hamiltonianos". En concreto, sobre las soluciones aproximadas del problema ved el subapartado 2.2.2 del módulo "Grafos eulerianos y grafos hamiltonianos".

En este subapartado justificaremos por qué este problema no tiene una solución eficiente, o sea, veremos que es un problema NP-Completo.

1.6.1. Descripción del problema

Recordemos que el problema se puede describir a partir de una situación real: un viajante de comercio tiene que visitar n ciudades una sola vez de la forma más económica posible. Es decir, ¿cuál es la ruta más corta que tiene que seguir el viajante para visitar todas las ciudades una sola vez volviendo al punto de partida?

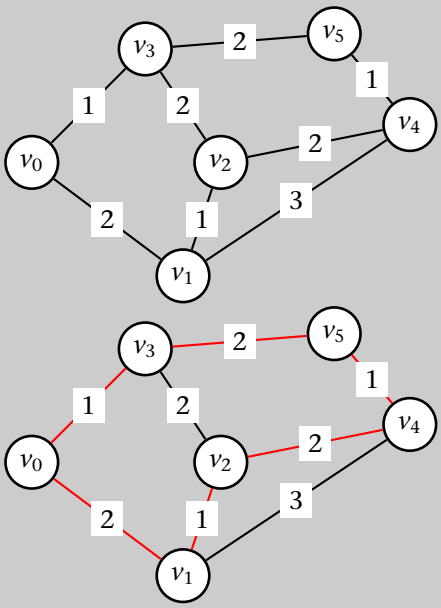
Observemos que si modelamos esta situación como un grafo ponderado entonces la solución del problema formará un ciclo hamiltoniano de peso total mínimo.

Problema de optimización 10

Dado un grafo ponderado $G = (V,A)$, ¿cuál es el ciclo hamiltoniano de G de peso total mínimo?

Ejemplo 21

En el grafo siguiente el ciclo hamiltoniano de peso total mínimo estará formado por el ciclo $\{v_0,v_3,v_5,v_4,v_2,v_1,v_0\}$ con un peso total de 9.



Ejercicios

9. Dado un grafo ponderado G tal que todas las aristas tienen el mismo peso, ¿cuál es la solución al problema del viajante de comercio?

Soluciones

9. Si todas las aristas tienen el mismo peso, podemos suponer que este peso es 1 y entonces cualquier ciclo hamiltoniano del grafo es una solución al problema del viajante de comercio.

1.6.2. Discusión

La búsqueda exhaustiva para este problema consiste en enumerar todos los ciclos hamiltonianos del grafo y, posteriormente, buscar el ciclo de peso más pequeño. Tal como hemos visto en el ejemplo 10 del módulo "Grafos eulerianos y grafos hamiltonianos", enumerar todos los ciclos hamiltonianos es un problema de complejidad exponencial, por lo que si aplicamos este método para resolver el problema del viajante de comercio obtendremos un algoritmo de complejidad no polinómica.

Vamos a ver que no podemos mejorar esta situación puesto que este también es un problema NP-Completo.

1) El problema de decisión asociado será:

Problema de decisión 11

TSP: Dado un grafo ponderado $G = (V,A)$ de orden n y un número entero k, determinar si existe un ciclo hamiltoniano en G de peso total k.

2) El problema TSP está en NP siguiendo el mismo razonamiento que para el problema HAM_CYCLE. Dada una secuencia de n vértices de G, podemos verificar en tiempo polinómico que los n vértices son todos diferentes y que, añadiendo el primer vértice al final de la secuencia, forman un ciclo en G de peso total k.

3) La reducción polinómica se puede hacer a partir del problema del ciclo hamiltoniano (HAM_CYCLE $\leq_p$ TSP).

Dado un grafo no ponderado $G = (V,A)$ de orden n, construimos el grafo ponderado $G' = (V',A')$ con los mismos vértices y aristas de G pero añadiendo a cada arista peso 1. La función de reducción será $f(G) = (G',n)$. Esta transformación se puede hacer lógicamente en tiempo polinómico.

Entonces, el grafo G contiene un ciclo hamiltoniano si, y solo si, el grafo G' tiene un ciclo hamiltoniano de peso total n.

1.7. Ejercicios

10. Diseñar un algoritmo para comprobar si k vértices de un grafo $G = (V,A)$ de orden n forman un grafo completo. Dar la complejidad de este algoritmo.

11. Un problema puede ser resuelto por un algoritmo que tiene una complejidad $O(n^{\log n})$. Justificar cuál de las siguientes afirmaciones es cierta:

a) El problema es tratable.

b) El problema es intratable.

c) Ninguna de las dos anteriores.

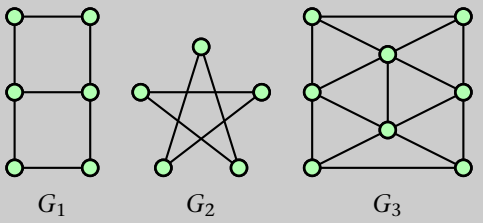
12. La organización de un congreso tiene que sentar 150 personas alrededor de una mesa redonda. Los organizadores saben que hay personas que están enemistadas entre sí y, por lo tanto, no se pueden sentar juntas. Expresar este problema como una instancia de algún problema intratable de los que habéis estudiado.

13. Resolver el problema de cobertura definido por la matriz:

	A	B	C	D	E	F
1	1	1	0	0	0	0
2	0	0	0	0	1	1
3	0	0	0	1	1	0
4	1	1	1	0	0	0
5	0	0	1	1	0	0
6	0	0	0	1	1	0
7	1	0	1	0	1	1

Es decir, buscar el número mínimo de columnas que cubren todas las filas.

14. Encontrar el número cromático de los grafos siguientes:



G_1 G_2 G_3

15. Proponer un algoritmo eficiente para encontrar una 2-coloración de un grafo bipartito conexo.

1.8. Soluciones

10. El algoritmo podría ser el siguiente:

```
Entrada : G = (V,A),s secuencia de k vértices del grafo
Salida : CIERTO si s forma un grafo completo, FALSO en caso contrario
algoritmo esClique(G,s)
  inicio
    completo ← CIERTO
    k ←|s|
    i ← 1
    mientras i < k ∧ completo
              j ← i + 1
              mientras j ≤ k ∧ completo
                        si ¬(esAdyacente(s[i],s[j]))
                           entonces completo ← FALSO
                        finsi
                        j ← j + 1
              finmientras
              i ← i + 1
    finmientras
    retorno (completo)
  fin
```

Si la representación del grafo es una matriz de adyacencias, entonces la operación *esAdyacente*() tiene una complejidad constante. Si la representación es una lista de adyacencias, esta operación tiene una complejidad lineal. Los dos bucles juntos se ejecutarán, como máximo, $(k–1)+(k–2)+\cdots+2+1 = \frac{k(k–1)}{2} = \frac{k^2–k}{2}$. Así la complejidad será del orden de $O(k^2)$ si la representación del grafo es una matriz de adyacencias y $O(k^2 \cdot n)$ si la representación es una lista de adyacencias.

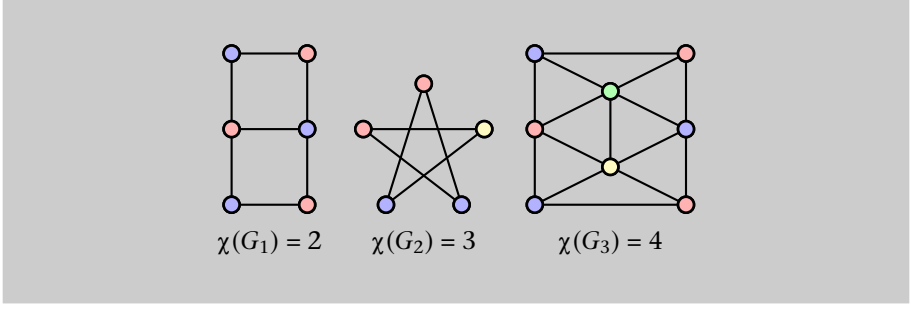
11. Un problema es tratable si puede ser resuelto con un algoritmo de complejidad polinómica, es decir, existe una constante $k \in \mathbb{N}$ tal que el algoritmo tiene una complejidad $O(n^k)$. Dado que $\log n$ es una función estrictamente creciente, fijado k, siempre podemos encontrar un n tal que $\log n > k$. Por lo tanto, un algoritmo de complejidad $O(n^{\log n})$ no tiene una complejidad polinómica. No obstante, con la información del ejercicio, no podemos concluir que el problema es intratable. Podría haber otro algoritmo que resuelve el problema y que tuviera una complejidad polinómica.

De acuerdo con el enunciado del ejercicio, la respuesta correcta sería la *c*.

12. Podemos definir el grafo $G = (V,A)$ de orden 150 donde V estará formado por las 150 personas y dos vértices serán adyacentes si las dos personas pueden sentarse juntas. Será posible sentar las 150 personas alrededor de una mesa sin que dos personas enemistadas se sienten juntas, si existe un ciclo hamiltoniano en el grafo G. El problema propuesto se puede considerar una instancia del problema intratable HAM_CYCLE.

13. Utilizando las simplificaciones de la página 16, el número mínimo de columnas es 3, *B*, *D* y *F*.

14. Los números cromáticos son los que se muestran en el siguiente gráfico:



15. Todo grafo bipartito es 2-colorable y en un grafo bipartito todos los ciclos son de longitud par. Por lo tanto, podemos utilizar el algoritmo BFS para calcular la distancia de un vértice inicial al resto de vértices del grafo. Los que estén a distancia par los pintaremos con uno de los dos colores, y los que estén a distancia impar los pintaremos con el otro color.

Ved también

El algoritmo BFS se estudia en el módulo "Recorridos y conectividad".

2. Otros problemas intratables

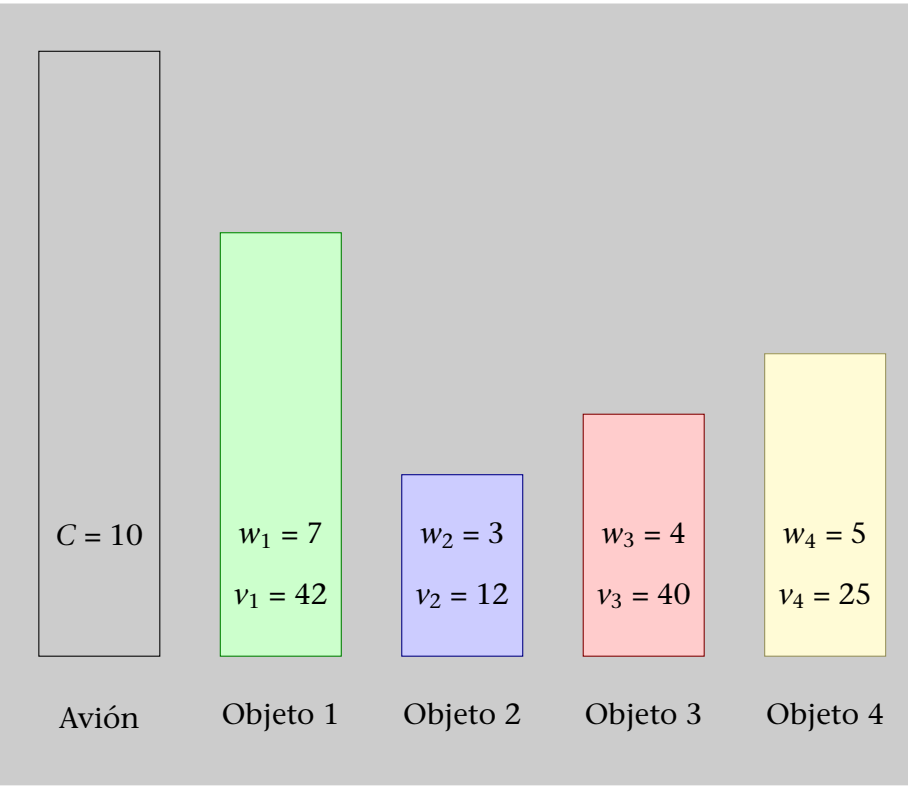
En el apartado 1 hemos visto numerosos problemas intratables y hemos utilizado la teoría de grafos para demostrar que efectivamente son problemas NP-Completos. En este apartado, introduciremos otros problemas que aparecen a menudo en informática y que también son intratables. El objetivo no será hacer una demostración formal de su intratabilidad, sino poner en evidencia la dificultad intrínseca de estos problemas.

2.1. El problema de la mochila

El problema de la mochila (*knapsack*, en inglés) es un problema relacionado con la asignación de recursos con restricciones. Suponer que un avión tiene que transportar una carga de objetos de varios pesos (w) y diferentes valores (v). Puesto que el avión puede transportar una carga máxima C, nos interesará transportar aquellos objetos que, sin sobrepasar la carga máxima, tengan el máximo valor. En el siguiente gráfico podéis ver una posible situación de partida:

El problema de la mochila

Se denomina así porque podemos pensar en una mochila con capacidad C en la cual queremos poner los objetos más valiosos sin sobrepasar su capacidad. Cada objeto tendrá un peso w y un valor v.



El avión tiene una capacidad de 10 toneladas y tenemos 4 objetos de peso 7, 3, 4 y 5 toneladas. Además, cada uno de los objetos tiene valor 42, 12, 40 y 25, respectivamente. El objetivo es transportar los objetos de más valor en el avión sin sobrepasar el máximo de la carga del avión.

2.1.1. Descripción del problema

Problema de optimización 12

Dados un conjunto de objetos $U = \{1,\ldots,n\}$ de peso w_i y valor v_i ($i = 1,\ldots,n$), y una mochila de capacidad C, ¿cuál es el subconjunto $S \subseteq U$ tal que $\sum_{i\in S} v_i$ sea máximo y $\sum_{i\in S} w_i \leq C$?

Ejemplo 22

Siguiendo con el ejemplo de la introducción, $C = 10$, $U = \{1,2,3,4\}$ con pesos $w_1 = 7$, $w_2 = 3$, $w_3 = 4$ y $w_4 = 5$. Los valores respectivos serán $v_1 = 42$, $v_2 = 12$, $v_3 = 40$ y $v_4 = 25$. La solución, en este caso, es el conjunto $S = \{3,4\}$ con un peso total de 9 toneladas y un valor total de 65.

Ejercicios

16. Suponer que todos los objetos del ejemplo de la introducción tuvieran el mismo valor, $v_1 = v_2 = v_3 = v_4$, ¿cuál sería ahora la solución del problema?

Soluciones

16. Cualquier pareja de objetos sería una solución al problema. Si queremos aprovechar al máximo la capacidad del avión, la solución sería el conjunto $S = \{1,2\}$.

2.1.2. Discusión

Este es un problema de optimización. La solución lógica es hacer una búsqueda exhaustiva entre todos los subconjuntos del conjunto U para encontrar el subconjunto con el peso total menor que C y con el valor total máximo. Puesto que el número de subconjuntos de un conjunto U de cardinal n es 2^n, podemos concluir que este algoritmo tendría una complejidad no polinómica.

Ejemplo 23

Siguiendo con el ejemplo de la introducción, la siguiente tabla muestra todas las posibilidades para buscar la solución del problema:

Subconjunto	Peso total	Valor total
$\emptyset$	0	0
{1}	7	42
{2}	3	12
{3}	4	40
{4}	5	25
{1,2}	10	54
{1,3}	11	no factible
{1,4}	12	no factible
{2,3}	7	52
{2,4}	8	37
{3,4}	**9**	**65**
{1,2,3}	14	no factible
{1,2,4}	15	no factible
{1,3,4}	16	no factible
{2,3,4}	12	no factible
{1,2,3,4}	19	no factible

En total hay $2^4 = 16$ posibilidades y la solución óptima está en negrita.

Vamos a describir brevemente en el proceso de demostración que el problema de la mochila es NP-Completo:

1) El problema de decisión asociado será:

Problema de decisión 13

KNAPSACK: Dados un conjunto de objetos $U = \{1,\ldots,n\}$ de peso w_i y valor v_i ($i = 1,\ldots,n$), una mochila de capacidad C, y un número entero k, determinar si existe un subconjunto $S \subseteq U$ tal que $\sum_{i\in S} v_i \geq k$ y $\sum_{i\in S} w_i \leq C$.

2) Este problema KNAPSACK está en NP, ja que dado un subconjunto $S \subseteq U$, podemos verificar en tiempo polinómico que $\sum_{i\in S} v_i \geq k$ y $\sum_{i\in S} w_i \leq C$.

3) La reducción polinómica se puede hacer a partir del problema del recubrimiento de vértices (VERTEX_COVER $\leq_p$ KNAPSACK).

La demostración de esta reducción es muy técnica y la daremos como resultado:

Proposición 4

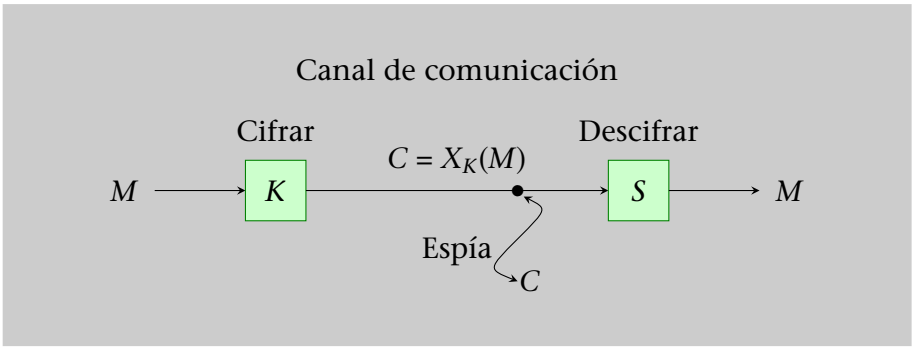
El problema de decisión KNAPSACK es un problema NP-Completo.

Demostración: Podéis ver la obra de Cormen y otros (2001). ∎

2.2. Factorización y primalidad

Uno de los problemas que la aparición de las tecnologías de la información y de las comunicaciones ha puesto de manifiesto es la vulnerabilidad de las comunicaciones. Las redes de ordenadores, los sistemas distribuidos, los espacios de comunicación en Internet, las transacciones electrónicas son algunos ejemplos de sistemas vulnerables a la pérdida de información o a recibir ataques de autenticidad o confidencialidad. Toda esta problemática ha dado lugar a nuevos campos de investigación y desarrollo dentro del ámbito de la seguridad informática.

Los especialistas e investigadores en seguridad informática utilizan básicamente técnicas matemáticas (criptografía) para incrementar la seguridad de los sistemas informáticos. De forma resumida, un sistema criptográfico responde al esquema siguiente:



En este esquema, un mensaje M se cifra con la clave K (clave pública del receptor) para obtener el mensaje cifrado C que se envía por el canal de comunicación. A la salida del canal, el mensaje cifrado C se descifra utilizando la clave S (clave privada del receptor). Un posible espía que tenga acceso al canal puede capturar el mensaje C, pero no puede obtener el mensaje original M si no conoce la clave S.

La seguridad del sistema depende básicamente del secreto de las claves y de la robustez de los algoritmos utilizados en el proceso de cifrado y de descifrado.

Uno de los sistemas criptográficos más utilizados es el RSA (Rivest-Shamir-Adelman). Este es un criptosistema de clave pública en el cual se utilizan claves diferentes para cifrar y descifrar los mensajes. La seguridad del RSA se basa en la diferencia de complejidad computacional entre la facilidad para encontrar números primos grandes y la dificultad de factorizar números enteros que se han formado como producto de dos números primos.

El esquema anterior corresponde a un esquema de cifrado utilizando RSA. La clave de cifrado es K y la de descifrado es S. Usualmente, las claves tienen una longitud de 1024 bits o más.

En este subapartado estudiaremos, desde el punto de vista de la complejidad computacional, estos dos problemas que están muy relacionados: la primalidad y la factorización de números enteros.

2.2.1. Primalidad

Recordemos que un número natural p es *primo* si y solo si tiene exactamente dos divisores: 1 y p. Si un número natural no es primo entonces se dice que es un número *compuesto*.

Número primo más grande

¿Sabíais que el número primo más grande que se conoce (agosto 2008, E. Smith) es $2^{43,112,609} - 1$ que tiene 12,978,189 dígitos?

Descripción del problema

Problema de decisión 14

PRIMO: Dado un número natural N, determinar si N es un número primo.

Tal como lo hemos formulado, ya es una problema de decisión. Además, determinar si N es primo es equivalente a determinar si N es compuesto.

Ejemplo 24

Los números naturales 2, 3, 5, 7 y 11 son primos. Son los cinco números primos más pequeños. Por definición, el número 1 no es primo y cualquier número primo más grande que 2 tiene que ser impar.

Los números 204, 505, 561, 1024 son compuestos, puesto que 204 y 1024 tienen un divisor 2; 505 es múltiplo de 5; 561 es múltiplo de 17.

Ejercicios

17. El problema de decidir si un número natural N es primo, cuando N es grande, es realmente complicado. En algunos casos, no obstante, es relativamente fácil demostrar que son compuestos y, por lo tanto, que no son primos.

Demostrar que no son primos los números 31415926535, 42567892, 3737691.

18. La *criba de Eratóstenes* es uno de los métodos más antiguos para obtener números primos. El método consiste en escribir consecutivamente todos los

números e ir eliminando de la lista todos los múltiplos de 2, después de 3,... Por ejemplo, partiendo de la lista

$$2,3,4,5,6,7,8,9,10,11,12,13,\ldots$$

obtendríamos

$$2,3,\ ,5,\ ,7,\ ,\ ,\ ,11,\ ,13,\ldots$$

Utilizando la criba de Eratóstenes, calcular todos los números primos más pequeños que 100.

Soluciones

17. El número 31415926535 acaba en 5. Por lo tanto, es múltiplo de 5.

El número 42567892 es par y mayor que 2. Por lo tanto, no puede ser primo.

La suma de las cifras del número 3737691 es 36 que es múltiplo de 3. Por lo tanto, tampoco puede ser primo.

18. Los números primos más pequeños que 100 son: 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97.

Discusión

Antes que nada analizamos el algoritmo básico para comprobar si un número natural N es primo. El siguiente algoritmo devuelve `CIERTO` si el número N es primo:

```
Entrada : N (N > 1)
Salida : CIERTO, si N es primo,FALSO en caso contrario
algoritmo esPrimo(N)
  inicio
    primo ← CIERTO
    k ← 2
    mientras k < N ∧ primo
                si N mod k = 0
                   entonces primo ← FALSO
                finsi
                k ← k + 1
    finmientras
    retorno (primo)
  fin
```

La complejidad de este algoritmo viene determinada por el número de iteraciones del bucle. En el peor del casos, el bucle se ejecutará $N - 2$ veces, hecho que determina una complejidad lineal, $O(N)$.

Parecería pues, que la cuestión de la intratabilidad del problema de decisión estaría resuelta.

Siguiendo el razonamiento que hemos hecho en el módulo "Complejidad computacional", si calculamos la complejidad en función de la medida de la representación binaria de N, entonces resulta que la complejidad pasa a ser $O(2^n)$, donde n es la longitud de la representación binaria de N. En este sentido este algoritmo tiene una complejidad no polinómica.

Ejercicios

19. El algoritmo propuesto se basa en el cálculo de un posible divisor de N. Demostrar que es suficiente hacer $\lfloor\sqrt{N}\rfloor$ divisiones para asegurar que N es primo. Calcular la complejidad de esta variante del algoritmo.

Soluciones

19. Observar que si a es un divisor de N, entonces $N = ab$ y, por eso, $a \leq \sqrt{N}$ o $b \leq \sqrt{N}$. En cualquier caso, es suficiente tomar k tal que $1 < k \leq \sqrt{N}$. Ahora la complejidad del algoritmo sería $O(N^{1/2})$ o $O(2^{n/2})$. El algoritmo continuará teniendo, no obstante, una complejidad no polinómica.

Puesto que el enunciado ya es un problema de decisión, quedaría por ver si es intratable (si está en NP y si es NP-Completo). Demostrar que el problema está en NP no es trivial. En 1975, el profesor V. R. Prat publicó el primer método para verificar que el problema PRIMO estaba en NP.

Sorprendentemente, el problema equivalente, determinar si un número entero es compuesto, se puede verificar trivialmente en tiempo polinómico. Es suficiente tomar un divisor a de N y comprobar que N es un múltiplo de a.

La cuestión sobre la NP-Completitud del problema PRIMO había quedado abierta hasta el año 2002. La respuesta fue sorprendente, este año se publicó un algoritmo de complejidad polinómica que resuelve el problema de la primalidad, es decir, el problema PRIMO está en la clase P.

El problema PRIMO **está en** P

En el año 2002, el profesor Manindra Agrawal y sus estudiantes, Neeraj Kayal y Nitin Saxena, del Instituto de Tecnología de la India en Kapur, demostraron que el problema PRIMO está en la clase P.

Este algoritmo, no obstante, tiene una complejidad que es del orden $O(n^{21/2})$ donde n es la medida de la representación binaria del número primo. Esta complejidad es muy alta lo cual hace este algoritmo poco útil en la práctica. Por suerte, se conocen alternativas algorítmicas para comprobar si un número entero es primo con una probabilidad muy alta. Estos algoritmos son los que se utilizan en la práctica.

Algoritmo de Miller-Rabin

El algoritmo de Miller-Rabin es un algoritmo que hace un test de primalidad aleatorio. Dado un entero impar N ($N > 2$) el test asegura que N es primo con una probabilidad de error igual a $\frac{1}{2^s}$, donde s es el número de veces que se aplica el test. Para $s = 50$ esta probabilidad es tan pequeña que podemos asegurar que N es primo si el test se aplica 50 veces y responde afirmativamente cada vez.

2.2.2. Factorización

Recordemos que todo número entero (que supondremos mayor que 0) se puede descomponer de forma única como producto de números primos.

Descripción del problema

Problema de cálculo 15

Dado un número entero N ($N > 1$), determinar la descomposición de N en factores primos.

Ejemplo 25

El número 2200 se puede descomponer como $2^3 \times 5^2 \times 11$. El número entero 2147483647 es primo, tal como demostró L. Euler alrededor de 1772, y su descomposición es él mismo. El número 4294967297 se descompone como producto de dos números primos 641×6700417.

Ejercicios

20. Decir si los siguientes números son compuestos y, en este caso, dar su descomposición en producto de factores primos: 122, 360 y 8633.

Soluciones

20. Para facilitar el trabajo podemos usar dos hechos que ya conocemos: los factores tienen que ser menores o iguales que la raíz cuadrada del número inicial y el método de la criba de Eratóstenes permite buscar factores entre los números primos conocidos.

$\lfloor\sqrt{122}\rfloor = 11$. Los divisores a probar son 2,3,5,7,11. El número $122/2 = 61$ es primo. Por lo tanto, $122 = 2 \times 61$.

$\lfloor\sqrt{360}\rfloor = 18$. Empezando a dividir por 2, después 3,... obtenemos $360 = 2^3 \times 3^2 \times 5$.

$\lfloor\sqrt{8633}\rfloor = 92$. Cualquiera de los números primos obtenidos con la criba de Eratóstenes del ejercicio 18 (menos el último) podría ser un divisor. En este caso $8633 = 89 \times 97$. Este último cálculo pone de manifiesto la dificultad para factorizar un número compuesto que es producto de dos números primos grandes.

La solución al problema de primalidad del subapartado anterior nos permite asegurar si un número natural N ($N > 1$) es un número primo y, en caso contrario, que es un número compuesto. Lo que no nos dice la solución al problema de primalidad es saber cuáles son los factores primos de N.

Algoritmo de Pollard

Aunque no se conoce ningún algoritmo general para factorizar números enteros de forma eficiente, sí que hay algoritmos que pueden factorizar números enteros que tienen factores pequeños. El algoritmo de Pollard utiliza una heurística que, en mucho casos, factoriza un número entero de forma eficiente.

Discusión

El algoritmo básico *esPrimo*(N) se puede extender fácilmente para obtener un factor de N en caso de que N no sea primo. Repitiendo la ejecución de este algoritmo para cada factor de N obtendríamos la factorización completa de N en números primos. Claramente, este método tiene una complejidad exponencial en función de la longitud de la representación binaria de N.

En este subapartado veremos que el problema de decisión asociado está en la clase NP.

El criptosistema RSA

Se basa en la diferencia de complejidad de dos problemas: primalidad (facilidad para obtener números primos grandes de forma eficiente) y factorización (dificultad para factorizar números que son producto de dos primos grandes). La clave del RSA más grande que se ha factorizado es RSA-768 que tiene 232 dígitos. Actualmente se utilizan, como mínimo, claves RSA-1024 que tienen 309 dígitos y que se cree que no son factorizables en un tiempo limitado.

Problema de decisión 16

FACTORES: Dados dos números naturales N ($N > 1$) y M, determinar si existe un entero k ($1 < k < M$) tal que N es un múltiplo de k.

Antes que nada, observar que si tenemos una solución al problema FACTORES, entonces por repetidas aplicaciones del método, obtendríamos una solución al problema de la factorización de números enteros.

El problema FACTORES está en NP puesto que dado un número entero k podemos verificar que es un factor de N en tiempo polinómico. Efectivamente, solo hay que hacer la división entre N y k.

El interés de este problema está en el hecho que si bien es fácil demostrar que el problema de decisión asociado está en la clase NP, nadie ha demostrado que se trate de un problema NP-Completo. De hecho, se cree que no es NP-Completo (si lo fuera podría significar que $P = NP$ lo cual es poco creíble hoy en día) pero tampoco se ha demostrado que esté en P. Gran parte de la criptografía moderna se basa en la dificultad de factorizar números enteros grandes que sean producto de dos números primos.

2.3. Más problemas intratables

Los problemas intratables son frecuentes en informática y en otros campos de las ciencias y las ingenierías. Además de los que hemos descrito en estos subapartados, enumeraremos algunos:

En el campo de la informática:

1) Sistemas operativos: planificación de tareas (*multiprocessor scheduling*), gestión de la memoria o espacio de disco (política de asignación de la memoria para minimizar la fragmentación).

2) Optimización: programación lineal entera.

3) Teoría de la codificación: cálculo del peso mínimo de un código lineal binario.

4) Compiladores: asignación óptima de registros de un procesador de las instrucciones que forman un bloque básico (sin saltos) de un lenguaje de programación (*register allocation*).

5) Compresión de la información: cálculo de la subsecuencia más larga de un conjunto de k secuencias de caracteres.

6) Robótica: localización exacta de un robot en una habitación.

En otros campos:

1) Ingeniería aeroespacial: encontrar la malla óptima por elementos finitos.

2) Biología: problema del plegado de proteínas (dada la secuencia de aminoácidos de una proteína, calcular su estructura tridimensional).

3) Ingeniería civil: encontrar el equilibrio del flujo de tránsito urbano.

4) Ingeniería eléctrica: diseño VLSI óptimo (dada una función lógica, encontrar un circuito que la implementa de forma óptima atendiendo a uno o más parámetros: superficie del circuito, tiempo de cálculo, consumo energético,...).

5) Ingeniería ambiental: localización óptima de sensores de contaminantes.

6) Teoría de juegos: equilibrio de Nash que maximiza el bienestar social.

7) Estadística: diseño experimental óptimo.

2.4. ¿Qué hacer ante un problema intratable?

La conclusión de este módulo podría ser que hay muchos problemas en ciencia e ingeniería de los cuales no se conoce (y seguramente no hay, excepto que *P = NP*) ninguna solución algorítmica eficiente. Estos problemas, no obstante, son muy frecuentes y no querríamos acabar este apartado con una visión pesimista sobre su resolución.

Afortunadamente, para la mayoría de los problemas intratables más frecuentes se conocen algoritmos que, en situaciones concretas, permiten obtener la solución exacta o aproximada de forma eficiente. Ante un problema intratable e intentando responder al título del subapartado, podríamos proceder de la siguiente forma:

1) Demostrar que P = NP (nada aconsejable si se pretende obtener una solución en un tiempo prudencial).

2) Estudiar casos particulares del problema por si tienen una solución eficiente (pueden estar en P).

3) Utilizar técnicas de diseño de algoritmos que, si bien dan algoritmos de complejidad exponencial, pueden solucionar el problema o acercarse a la solución: programación dinámica, *backtracking*, *branch and bound*.

4) Utilizar técnicas heurísticas como los algoritmos genéticos o la búsqueda tabú para obtener una buena solución, a pesar de que no necesariamente tiene que ser la óptima.

5) Utilizar algoritmos probabilísticos basados en el método de Monte Carlo o en el método de Las Vegas.

6) Utilizar paralelismo o computación distribuida para repartir el tiempo de cálculo entre muchos ordenadores.

7) Si todo esto falla, tendremos que esperar a la computación cuántica, puesto que entonces algunos de estos problemas podrán ser resueltos por algoritmos eficientes.

Branch and bound

El *branch and bound* es una técnica de diseño de algoritmos que consiste en la enumeración sistemática de todas las soluciones candidatas y el mantenimiento de cotas superiores e inferiores de la solución óptima que permiten descartar las candidatas que no están entre las cotas.

Algoritmo de Shor

El 1994, Peter Shor demostró que el problema de la factorización de números enteros tenía una complejidad polinómica utilizando los principios de la computación cuántica.

2.5. Ejercicios

21. Lleváis una mochila de capacidad 34 l y queréis poner tres tipos de comida de 7 l, 5 l y 3 l (tenemos varias unidades de cada tipo). Cada clase de comida suministra energía por valor de 6 Kcal, 4 Kcal y 2 Kcal. Queremos cargar la mochila con la comida que tenga el máximo de energía. Plantear este problema como una instancia de un problema de optimización. ¿Podemos afirmar que este problema es NP-Completo?

22. Sea U un conjunto de n objetos, todos de la misma medida, de valor v_i. Queremos llenar una mochila de capacidad C con los objetos del conjunto U de tal manera que el valor total de los objetos que quepan sea máximo.

a) Definir un problema de optimización para el planteamiento del enunciado.

b) Definir un problema de decisión (problema K) asociado al problema de optimización.

c) Demostrar que el problema K está en P.

d) Definir una reducción polinómica del problema K al problema KNAPSACK (K $\leq_p$ KNAPSACK).

e) ¿Por qué esta reducción no demuestra que el problema KNAPSACK está en P? Justificar la respuesta.

23. Justificar cuáles de los números enteros siguientes son primos y cuáles son compuestos. Y en este segundo caso, obtener una descomposición en factores primos: 13110, 4758, 1455, 4937.

24. Justificar si son ciertas o falsas las siguientes afirmaciones:

a) El problema PRIMO está en NP.

b) El problema PRIMO es NP-Completo.

c) El problema FACTORES está en NP.

d) El problema FACTORES está en P.

e) El problema FACTORES es NP-Completo.

2.6. Soluciones

21. Denotamos por $C = 34$ y por $w_1 = 7$, $w_2 = 5$ y $w_3 = 3$ con $v_1 = 6$, $v_2 = 4$ y $v_3 = 2$. La diferencia con el caso del problema de la mochila es que podemos tener varias unidades de comida de cada tipo. Denotamos por I, J y K los tres tipos de comida. Podemos suponer que tenemos x unidades del tipo I de tal manera que $7x \leq 34$, y unidades del tipo J de tal manera que $5y \leq 34$ y z unidades del tipo K con tal que $3z \leq 34$. Así, tendremos un conjunto $U = \{I_1, \ldots, I_x, J_1, \ldots, J_y, K_1, \ldots, K_z\}$ de cardinal $x + y + z$ y tenemos que elegir un subconjunto $S \subseteq U$ tal que $\sum_{i \in S} w_i \leq C$ y $\sum_{i \in S} v_i$ sea máximo. De este modo se trata de un problema de optimización que denominaremos COMIDA.

Tal como hemos planteado el problema de optimización, hemos hecho una reducción del problema COMIDA al KNAPSACK, COMIDA $\leq_p$ KNAPSACK. Por lo tanto, aunque ya sabemos que el problema KNAPSACK es NP-completo no podemos afirmar que el problema COMIDA es NP-completo.

22. Supongamos que el conjunto es $U = \{1, 2, \ldots, n\}$ y que la medida de cada objeto es 1.

a) Problema de optimización: Encontrar un subconjunto $S \subseteq U$ tal que $\sum_{i \in S} 1 = |S| \leq C$ y $\sum_{i \in S} v_i$ sea máximo.

b) Problema de decisión (K): Dado un número entero k ($k > 1$), determinar si existe un subconjunto $S \subseteq U$ tal que $|S| \leq C$ y $\sum_{i \in S} v_i = k$.

c) Para demostrar que el problema K está en P tenemos que encontrar un algoritmo eficiente que resuelva el problema. El algoritmo consiste en ordenar en orden decreciente de sus valores los n objetos de U y elegir el subconjunto de U formado por los $|C|$ primeros objetos. Este algoritmo tendrá una complejidad $O(n \log n)$ para la ordenación y $O(|C|)$ para elegir el subconjunto de U. En definitiva, será un algoritmo de complejidad polinómica.

d) La reducción polinómica de K a KNAPSACK consistirá en transformar los datos de entrada del problema K en los mismos datos de entrada del problema KNAPSACK cogiendo $w_i = 1$. Esta transformación es obviamente polinómica y cada solución del problema K se corresponde con una solución del problema KNAPSACK que hemos definido.

e) La reducción polinómica demuestra que el problema KNAPSACK tiene un subproblema que está en P, pero no demuestra que cualquier conjunto de datos de entrada del problema KNAPSACK tiene solución eficiente. Para hacerlo, habría que conseguir una reducción KNAPSACK $\leq_p$ K, que es diferente de lo que hemos hecho en el apartado *d*.

23. Estudiamos cada uno de los números enteros:

a) 13110 es par y múltiplo de 3 y de 5. Por lo tanto, $13110 = 2 \times 3 \times 5 \times 437$. Puesto que $\lfloor\sqrt{437}\rfloor = 20$ los posibles divisores serán 7, 11, 13, 17, 19. Después de las comprobaciones, $13110 = 2 \times 3 \times 5 \times 19 \times 23$.

b) 4758 es par y múltiplo de 3. Por lo tanto, $4758 = 2 \times 3 \times 793$. Puesto que $\lfloor\sqrt{793}\rfloor = 28$ los posibles divisores serán 7, 11, 13, 17, 19, 23. En este caso, 793 es múltiplo de 13 y obtenemos $4758 = 2 \times 3 \times 13 \times 61$.

c) 1455 es múltiplo de 3 y de 5. Por lo tanto, $1455 = 3 \times 5 \times 97$ y 97 ya es primo.

d) $\lfloor\sqrt{4937}\rfloor = 70$. Puesto que no es múltiplo ni de 2, ni de 3, ni de 5, tenemos que probar los factores 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, 61, 67. En este caso, ninguno de estos números es un factor de 4937 por lo cual podemos afirmar que 4937 es un número primo.

24. Justificamos cada afirmación:

a) Cierta. Desde el año 2002 se conoce un algoritmo de complejidad polinómica que resuelve el problema PRIMO. Por lo tanto, PRIMO está en $P \subseteq NP$.

b) No se sabe. Si PRIMO fuera NP-Completo, entonces significaría que P = NP.

c) Cierta. Podemos verificar en tiempo polinómico que un número entero tiene un factor.

d) No se sabe, aunque probablemente sea falsa. No se conoce ningún algoritmo eficiente para descomponer números enteros.

e) No se sabe, aunque probablemente sea falsa.

Ejercicios de autoevaluación

1. Justificar si cada uno de los siguientes problemas de decisión son tratables e indicar el algoritmo que utilizaríais para demostrarlo:

a) ¿El número entero n es múltiplo del número entero m?

b) ¿Los números enteros x e y son primos entre sí, es decir, su máximo común divisor es 1?

c) ¿El sistema de ecuaciones lineales $Ax = B$ tiene solución?

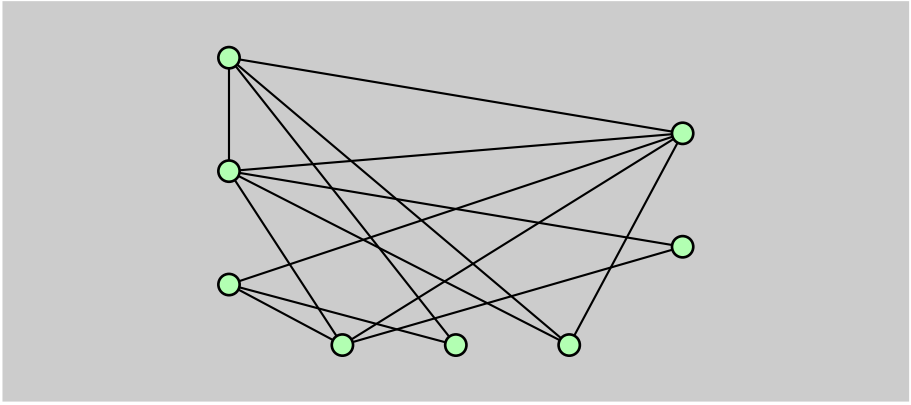
d) ¿La distancia entre dos ciudades x e y en un mapa de carreteras es k?

e) ¿Es posible dibujar un grafo G con un solo trazo, sin repetir ninguna línea y empezando y acabando en un mismo punto?

f) ¿Es posible enviar una información a través de una red de ordenadores de tal manera que pase por todos los ordenadores, sin repetir ninguno y volviendo a su punto de partida?

2. Un **conjunto independiente** de un grafo $G = (V,A)$ es un subconjunto $S \subseteq V$ tal que no existe ninguna arista entre los vértices de S. El **problema del conjunto independiente** consiste en encontrar el conjunto independiente más grande de G.

a) Dado el grafo siguiente indicar cuáles son los conjuntos independientes de este grafo y cuál es la solución al problema del conjunto independiente en este grafo.



b) Demostrar que S es un conjunto independiente de $G = (V,A)$ si y solo si toda arista de G es incidente como mucho a un vértice de S.

c) Demostrar que S es un conjunto independiente de $G = (V,A)$ si y solo si S es un subgrafo completo en G^c.

d) El problema del conjunto independiente es un problema de optimización. Indicar cuál es el problema de decisión (INDEPENDENT_SET) asociado al problema del conjunto independiente.

e) Demostrar que el problema de decisión INDEPENDENT_SET está en NP.

f) Demostrar que el problema de decisión INDEPENDENT_SET es un problema NP-Completo. (Indicación: Utilizar la reducción del 3SAT $\leq_p$ CLIQUE para encontrar una reducción polinómica de 3SAT a INDEPENDENT_SET.)

3. Considerar la fórmula booleana:

$$(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee \bar{z}).$$

a) Comprobar que la asignación $x = 0$, $y = 1$, $z = 0$ satisface esta fórmula booleana.

b) Comprobar que las reducciones descritas en el ejercicio anterior permiten obtener un conjunto independiente de cardinal 3.

4. Encontrar un conjunto de columnas óptimo (con el menor coste posible) que cubra todas las filas. Es decir, resolver el problema de cobertura para la siguiente matriz:

	S_1	S_2	S_3	S_4	S_5	S_6	S_7	S_8
R_1	1	1	1	0	0	1	0	1
R_2	1	0	1	0	0	1	0	1
R_3	0	0	0	1	0	0	0	0
R_4	0	1	0	0	1	0	1	1
R_5	0	0	0	0	1	1	1	0
R_6	1	1	0	0	0	0	1	0
Cost	4	7	5	8	3	2	8	5

Decir si la solución encontrada es también una solución del problema de partición.

5. En una universidad se quiere automatizar la confección de los horarios de exámenes y han pedido al servicio informático que diseñe un programa para resolver este problema con las siguientes restricciones:

- En cada jornada de exámenes, todos los exámenes empezarán y acabarán a la misma hora.
- Los alumnos tienen que poder asistir a todos los exámenes de las asignaturas en que están matriculados.
- El periodo de exámenes tiene que ser lo más corto posible (mínimo número de días).

a) Definir un problema de optimización para la situación descrita en el enunciado.

b) Describir el problema de decisión asociado.

c) El servicio de informática dice que ha resuelto el problema de forma eficiente de tal manera que lo puede resolver siempre en tiempo polinómico respecto al número de estudiantes y asignaturas de la universidad. ¿Creéis que el algoritmo propuesto por el servicio de informática será bastante general para resolver cualquier posible entrada del algoritmo?

6. Recordar que un recubrimiento de vértices de un grafo $G = (V,A)$ es un subconjunto $S \subseteq V$ tal que toda arista de A es incidente a al menos un vértice de S.

a) Enunciar el problema del recubrimiento de vértices como un problema de decisión (VERTEX_COVER).

b) Justificar que VERTEX_COVER es un problema NP.

c) Dado un 3SAT con conjunto de cláusulas $C = \{C_1, \ldots, C_n\}$, construimos el grafo $G = (V,A)$ de forma que, para cada cláusula $C_j = x \vee y \vee z$, definimos los vértices x_j, y_j y z_j, los tres adyacentes entre ellos (formando un triángulo). Además, si dos vértices de triángulos diferentes corresponden a literales complementarios, entonces también son adyacentes. Demostrar que G tiene un recubrimiento con $2n$ vértices si y solo si el 3SAT se satisface.

d) La asignación $x = 1$, $y = 0$, $z = 0$ satisface la fórmula booleana:

$$(x \vee y \vee \bar{z}) \wedge (\bar{x} \vee \bar{y} \vee z) \wedge (\bar{x} \vee y \vee \bar{z}).$$

Construir el grafo $G = (V,A)$ correspondiente, según las anteriores transformaciones, y determinar el recubrimiento, con $\frac{2|V|}{3}$ vértices, asociado.

7. El ejercicio anterior demuestra que VERTEX_COVER es un problema NP-Completo. Otra forma de verlo es la siguiente.

a) Demostrar que en un grafo $G = (V,A)$, $S \subseteq V$ es un recubrimiento si, y solo si, $V \setminus S$ es un conjunto independiente.

b) Utilizando el apartado anterior, demostrar que VERTEX_COVER es un problema NP-Completo.

8. Considerar el siguiente problema de decisión: COMPUESTO: Dado un número natural N ($N > 1$), determinar si N es un número compuesto. Por ejemplo, el número 437669 es un número compuesto dado que es múltiplo de 541. Concretamente, $437669 = 541 \times 809$. Observar que si N es un número compuesto entonces existe un número natural k ($1 < k < N$) tal que N es múltiplo de k.

a) Diseñar el algoritmo *esMultiplo*(N,k) que permite justificar si N es múltiplo de k.

b) Justificar que el problema COMPUESTO $\in$ NP.

c) ¿Este algoritmo nos permite garantizar que el problema FACTORES está en P?

9. Bonnie and Clyde han robado un banco y quieren repartirse el botín consistente en n monedas de valores distintos. Se las quieren repartir de forma que cada uno tenga la misma cantidad total de dinero. Para cada una de las siguientes situaciones, decir si el problema correspondiente está en P o en NP.

a) Todas las monedas tienen el mismo valor.

b) Las monedas tienen diferentes valores p_i ($1 \leq i \leq n$).

c) Las monedas tienen dos únicos valores x e y, y hay un número par de monedas de valor x y un número par de monedas de valor y.

d) Todas las monedas tienen un valor diferente y estos valores son n números enteros consecutivos con n múltiplo de 4.

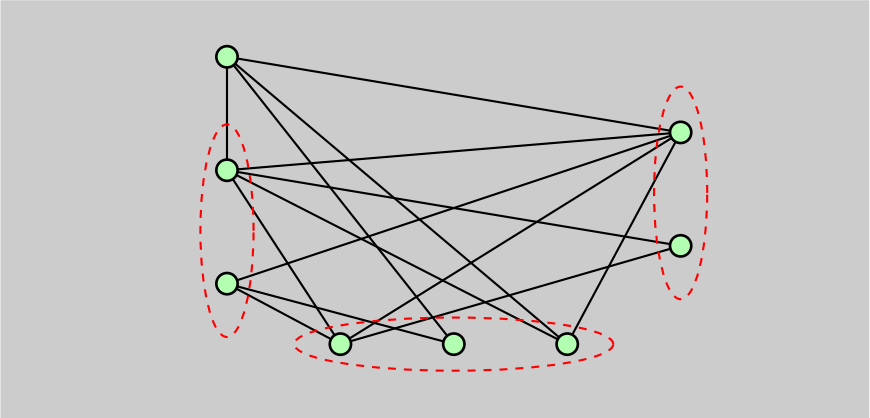
Soluciones

1. La tabla siguiente muestra el resultado:

Problema	Tratable	Algoritmo
a	Sí	Algoritmo de la división entera entre n y m.
b	Sí	Algoritmo de Euclides: máximo común divisor entre x e y.
c	Sí	Eliminación por el método de Gauss.
d	Sí	Algoritmo de Dijkstra.
e	Sí	Comprobar si el grafo es euleriano.
f	No	Comprobar si el grafo es hamiltoniano.

2. La finalidad de este ejercicio es demostrar que el problema del conjunto independiente es un problema NP-Completo.

a) En el gráfico siguiente podéis ver el grafo con tres conjuntos de vértices que no están conectados entre ellos. Dos de los conjuntos tienen dos vértices y el otro tiene tres. Este es el conjunto de vértices más grande que no tiene conexiones entre ninguno de sus vértices.



b) Si alguna arista de G fuera incidente con dos vértices u y v de S entonces la arista $\{u,v\}$ estaría en S, y viceversa.

c) Consecuencia del hecho que un conjunto independiente es un subgrafo nulo de G y del hecho que el complementario del grafo nulo N_r ($r > 0$) es el grafo completo K_r.

d) INDEPENDENT_SET: Dado un grafo $G = (V,A)$ de orden n y un número entero k ($k > 1$), determinar si existe un conjunto independiente de G de orden k.

e) Para comprobar que el problema de decisión es un problema NP, es suficiente verificar, en tiempo polinómico, si dado un subconjunto de k vértices del grafo es un conjunto independiente, es decir, si los vértices no son adyacentes entre sí.

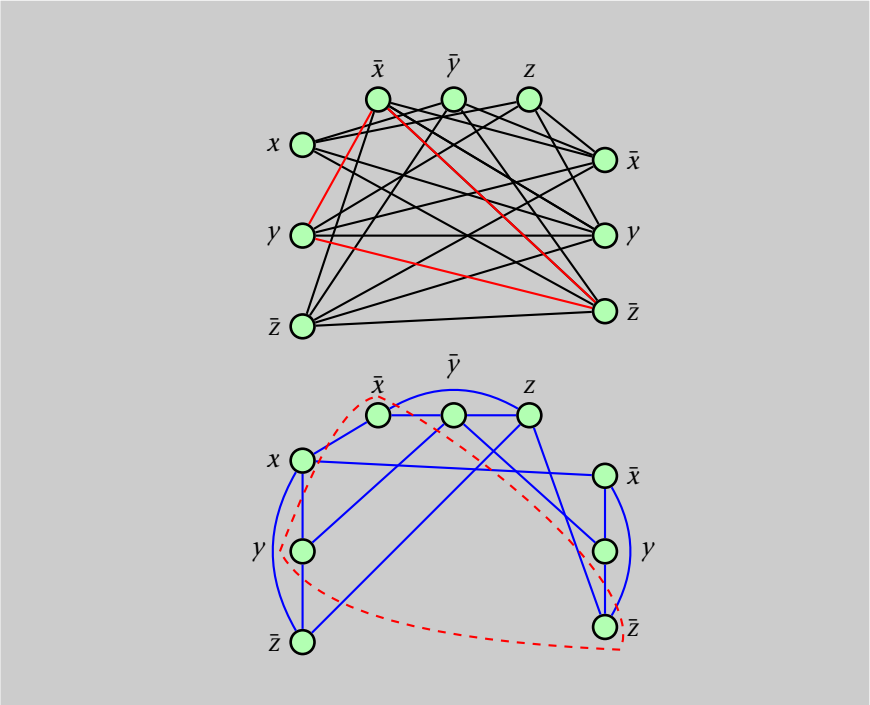
f) Según la correspondencia entre conjuntos independientes de un grafo y subgrafos completos del complementario, la reducción consistiría en transformar cada fórmula booleana formada por k cláusulas en un grafo $G = (V,A)$ de $3k$ vértices y posteriormente transformar este grafo en su complementario $G^c = (V,A')$ donde las aristas de A' son todas las aristas de K_{3k} que no están en A. Estas dos transformaciones se pueden hacer en tiempo polinómico.

Además, de acuerdo con las propiedades de la reducción de 3SAT a CLIQUE y de la propiedad anterior, G^c tiene un conjunto independiente de cardinal k si, y solo si, la fórmula booleana se satisface.

3. a) Substituyendo los valores de las variables en la fórmula obtenemos:

$$(0 \vee 1 \vee 1) \wedge (1 \vee 0 \vee 0) \wedge (1 \vee 1 \vee 1) = 1.$$

b) El siguiente gráfico muestra el resultado de las transformaciones:



4. En primer lugar, vemos que R_3 tiene un único 1 correspondiente a la columna S_4. Por lo tanto, S_4 forma parte de la solución (lo indicaremos en la tabla con un asterisco) y podemos eliminar las filas cubiertas por S_4 (solo R_3). Observamos también que la fila R_1 cubre la fila R_2. Por lo tanto, podemos eliminar R_1. Después de esto nos queda la matriz:

	S_1	S_2	S_3	S_4^*	S_5	S_6	S_7	S_8
R_2	1	0	1	0	0	1	0	1
R_4	0	1	0	0	1	0	1	1
R_5	0	0	0	0	1	1	1	0
R_6	1	1	0	0	0	0	1	0
Coste	4	7	5	8	3	2	8	5

Ahora no tenemos filas con un solo 1, ni filas cubiertas por otras filas. Pasamos a mirar las columnas. Podemos ver que la columna S_3 solo cubre R_2 que se puede cubrir con menor coste, por ejemplo S_1 tiene menor coste y cubre R_2. Por lo tanto, eliminamos la columna S_3. Observamos también que las filas cubiertas por S_7 pueden cubrirse utilizando S_1 y S_5. Como el coste de S_1 y S_5 es $4+3=7$ y, en cambio, S_7 tiene coste 8, podemos eliminar también la columna S_7. Entonces nos queda la matriz:

	S_1	S_2	S_4^*	S_5	S_6	S_8
R_2	1	0	0	0	1	1
R_4	0	1	0	1	0	1
R_5	0	0	0	1	1	0
R_6	1	1	0	0	0	0
Coste	4	7	8	3	2	5

Continuamos sin tener filas con un solo 1, ni filas cubiertas por otras. Mirando la columna S_2, podemos ver que las filas que cubre (R_4 y R_6) pueden ser cubiertas por S_1 y S_5 con el mismo coste (7); por lo tanto, podemos eliminar la columna S_2. Nos queda ahora la matriz:

	S_1	S_4^*	S_5	S_6	S_8
R_2	1	0	0	1	1
R_4	0	0	1	0	1
R_5	0	0	1	1	0
R_6	1	0	0	0	0
Coste	4	8	3	2	5

La última fila, R_6, tiene un único 1 en la columna S_1; por lo tanto, podemos eliminar R_6 y S_1 forma parte de la solución, con lo cual, eliminamos también la fila R_2 puesto que está cubierta por S_1. Nos queda la matriz:

	S_1^*	S_4^*	S_5	S_6	S_8
R_4	0	0	1	0	1
R_5	0	0	1	1	0
Cost	4	8	3	2	5

La fila que cubre S_8 (es decir, R_4) está cubierta por S_5 con coste menor. Por lo tanto, eliminamos S_8. Después de esto, R_4 tendrá un único 1 en la columna S_5. Por lo tanto, S_5 tiene que formar parte de la solución y ya tenemos cubiertas todas las filas.

Así pues la solución es $\{S_1,S_4,S_5\}$ con un coste $4 + 8 + 3 = 15$. Observamos, en la matriz original, que con esta solución no cubrimos ninguna fila más de una vez, o sea que $\{S_1,S_4,S_5\}$ es también la solución del problema de partición.

5. Antes que nada vamos a modelar la situación de partida con un grafo $G = (V,A)$. El conjunto de vértices estará formado por el conjunto de las asignaturas de la universidad. Dos vértices (asignaturas) serán adyacentes si hay algún alumno matriculado en las dos asignaturas. Entonces, podremos poner dos exámenes el mismo día si los vértices correspondientes a aquellas dos asignaturas no son adyacentes. Por lo tanto, se tendrá que encontrar una partición del conjunto de vértices de G, es decir, $V = \{V_1 \cup V_2 \cup \ldots \cup V_k\}$, con $V_i \cap V_j = \emptyset$ $(i \neq j)$, de forma que si $x \in V_i$ y $y \in V_j$ entonces x e y no son adyacentes.

a) Problema de optimización: Dado el grafo $G = (V,A)$, ¿cuál es la partición del conjunto de vértices de G con el mínimo número de subconjuntos?

b) Problema de decisión: Dado el grafo $G = (V,A)$ y un número entero k ($k > 0$), determinar si existe una partición del conjunto de vértices de G formada por k subconjuntos.

c) Esta situación es equivalente a encontrar el número cromático del grafo G, que es un problema NP-Completo. Por lo tanto, es muy dudoso que el servicio de informática haya podido resolver este problema de forma eficiente.

6. a) VERTEX_COVER: Dado un grafo $G = (V,A)$ y un número entero $k > 1$, determinar si existe un recubrimiento en G que tenga k vértices.

b) Para comprobar que el problema de decisión está en NP, es suficiente verificar, en tiempo polinómico, si dado un subconjunto S de k vértices, este es un recubrimiento.

Esta comprobación puede hacerse mirando que el conjunto de aristas incidentes a S es todo A. Como la medida máxima del conjunto de aristas A es $\frac{|V|^2}{2}$, podemos hacer la comprobación en tiempo polinómico.

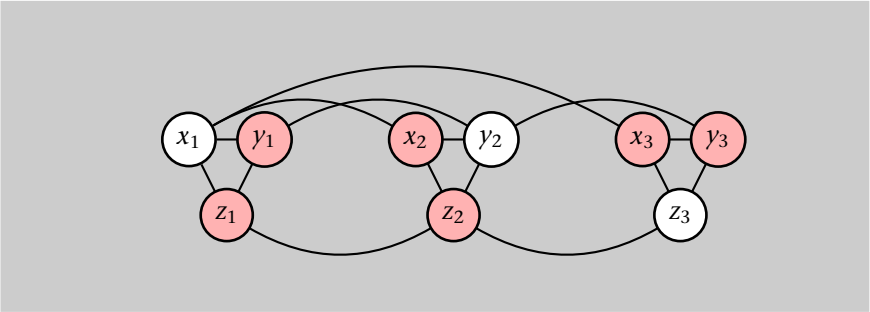
c) El grafo que hemos construido tiene $|V| = 3n$ vértices y se puede construir en tiempo polinómico. Vamos a ver que G tiene un recubrimiento con $2n$ vértices si y solo si el 3SAT se satisface:

Supongamos primero que el 3SAT se satisface, entonces, en cada cláusula, podemos seleccionar un literal que vale 1. Sea U el conjunto de vértices correspondientes a estos n literales. Los restantes $2n$ vértices de V que no

están en U forman un recubrimiento, dado que de cada grupo de 3 vértices correspondientes a una cláusula solo uno es de U y no puede ser que U contenga dos vértices que representan literales complementarios. De esta forma, es fácil ver que todas las aristas son incidentes a algún vértice de V pero no de U.

Supongamos ahora que G tiene un recubrimiento S con $2n$ vértices. Para cada 3 vértices correspondientes a una cláusula, solo uno estará fuera del conjunto dominante S (si hubiera dos, la arista entre estos dos no sería incidente a ningún vértice de S). Asignamos un 1 al literal correspondiente a este vértice que no es de S. Esto lo hacemos en todas las cláusulas. No nos puede pasar que queramos asignar 1 a un literal y a su complementario, puesto que si ninguno de los dos vértices fuera de S, la arista que había entre ellos no sería incidente a ningún vértice de S. Como todas las cláusulas tienen un literal que vale 1, el 3SAT se satisface.

d) El conjunto dominante está formado por los vértices en rojo.



7. a) Si S es un recubrimiento, dados dos vértices que no son de S, no pueden ser adyacentes puesto que tendríamos una arista que no sería incidente a ningún vértice de S. Recíprocamente, si S' es un conjunto independiente de vértices, entonces toda arista solo puede ser incidente a un vértice, como máximo, de S'. Por lo tanto, toda arista es incidente a un vértice de S.

b) Como ya se ha visto que el INDEPENDENT_SET es NP-Completo y es equivalente al VERTEX_COVER (cualquiera de los problemas se puede reducir al otro en tiempo polinómico), la conclusión es que VERTEX_COVER también es NP-Completo.

8. En este ejercicio se relacionan los tres problemas de primalidad, números compuestos y factorización.

a) El algoritmo sería:

```
Entrada : N,k
Salida : CIERTO si N es un número compuesto, FALSO en caso contrario
algoritmo esMultiplo(N,k)
  inicio
    compuesto ← FALSO
    si (k > 1 ∧ k < N) entonces
      si (N mod k = 0) entonces
        compuesto ← CIERTO
      finsi
    finsi
    retorno (compuesto)
  fin
```

b) El algoritmo *esMultiplo* es un algoritmo de complejidad polinómica que verifica que k es un factor de N. Esto demuestra que COMPUESTO $\in$ NP.

c) Esta versión del problema COMPUESTO no asegura cuál es el factor k de un número entero N y, por este motivo, no podemos concluir que FACTORES esté en P.

9. a) Hay que elegir simplemente $\frac{n}{2}$ monedas para cada uno si el número de monedas es par. Si es impar, entonces no es posible repartirlas equitativamente. El problema está en P.

b) Este enunciado coincide con el problema PARTICION, que ya vimos en el módulo "Complejidad computacional" que era NP-completo.

c) En este caso también podemos elegir la mitad de las monedas de valores x y la mitad de las monedas de valor y. El problema está en P.

d) En este caso podemos ordenar las monedas según su valor de tal manera que $v_1 < v_2 < \cdots < v_n$. Como los enteros son consecutivos se cumple que $v_i + v_{n-i+1}$ es constante. Como n es múltiplo de 4, sabemos que tenemos un número par de parejas (v_i, v_{n-i+1}), donde cada pareja tiene la misma suma total. Esto es lo mismo que repartir un número par de monedas del mismo valor. El problema está en P.

Bibliografía

Cormen, T.H.; Leiserson, C.E.; Rivest, R.L.; Stein, C. (2001). *Introduction to Algorithms*. Cambridge: MIT Press.

Garey, M.R.; Johnson, D.S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York: W.H. Freeman and Company.