CAPÍTULO 4

Grafos y particiones

Los grafos son las estructuras de datos más generales posibles. Otras estructuras de datos, tales como las listas o los árboles, podrían considerarse casos particulares de los grafos. Se utilizan para representar relaciones binarias entre objetos. Un grafo puede definirse como un par $\langle V, A\rangle$, donde V es un conjunto de *vértices* y $A \subseteq V \times V$ es una relación binaria en V. Suele presentarse como un conjunto de *aristas*. Una arista es un par (u, v), con $u, v \in V$.

Si se considera que el orden de los dos vértices dentro de cada par importa, el grafo se denomina *dirigido*. Cada arista (u, v) tiene una orientación: u es su vértice *origen* y v su vértice *destino*; además, diremos que v es *adyacente* a u. En caso contrario, el grafo se dice *no dirigido*: si $(u, v) \in A$ es una arista, entonces $(v, u) \in A$ y viceversa, es decir, la relación binaria representada por A es simétrica. Diremos que v es adyacente a u y que u es adyacente a v o que ambos son adyacentes entre sí. En un grafo no dirigido no suelen permitirse aristas de la forma (u, u).

En ocasiones, es útil incluir una *etiqueta* asociada a cada arista. Diremos entonces que el grafo está *etiquetado*. Por ejemplo, se puede representar un autómata mediante un grafo dirigido etiquetado en el que los vértices representan estados, las aristas transiciones entre estados y las etiquetas eventos asociados a las transiciones. A veces, la etiqueta es un valor numérico que representa el *valor* o *peso* de la arista en cuestión. Diremos entonces que se trata de un *grafo valorado*. Existen numerosos algoritmos sobre grafos que requieren pesos en las aristas. El objetivo de estos algoritmos suele ser optimizar una cierta función de coste relacionada con estos pesos.

Los grafos aparecen en una amplia variedad de contextos. Su estudio ha dado lugar a sofisticados y elegantes algoritmos, que aportan soluciones eficientes a

muchos problemas de las sociedades modernas. Basten los siguientes ejemplos para corroborarlo:

Navegadores Encontrar la ruta óptima o simplemente una ruta, desde un destino a otro en un mapa de carreteras o en el plano de una ciudad, se ha convertido en una acción cotidiana trivial gracias al uso de los navegadores de nuestros teléfonos móviles o de nuestro automóvil. En este caso, cada ciudad, pueblo o simple cruce, es un vértice del grafo y las calles, caminos o carreteras que los unen, son las aristas.

Navegadores web Aquí, cada página web es un vértice del grafo y los enlaces que permiten navegar de una página a otra, son las aristas. Los buscadores "se mueven" de forma incansable por este inmenso grafo, creando índices con los contenidos de las páginas.

Redes sociales Los vértices son las cuentas de la red social y las aristas son los mensajes que circulan por la red. Estudiando la forma y volumen de estas conexiones se pueden detectar dónde están las cuentas más influyentes, dónde se ubican sus seguidores y hasta saber qué cuentas son falsas (las conocidas como *bots*).

Circuitos electrónicos Los vértices corresponden a los componentes y las aristas, a las conexiones entre ellos. Preguntas tales como, ¿cuál es la disposición que minimiza la longitud de las conexiones? o ¿puede crearse el circuito sobre una placa o sobre un chip, sin que se crucen dos conexiones?, permiten ser contestadas por los correspondientes algoritmos sobre grafos.

Transporte Las redes ferroviarias o la distribución de productos estratégicos, tales como los combustibles, los alimentos u otros, requieren una compleja logística, como por ejemplo la necesidad de almacenes intermedios, la maximización del flujo a través de las redes de transporte, la minimización de los tiempos y de los costes, etc. Muchos de los algoritmos involucrados tienen que ver con grafos.

Planificación Muchos proyectos empresariales, la construcción de complejas infraestructuras y otras actividades similares que involucran tareas, precedencias entre ellas, máquinas, personas, tiempos y costes, necesitan una compleja planificación que puede ser expresada mediante grafos. Su tratamiento algorítmico conduce a soluciones óptimas.

En este capítulo se presentan las posibles representaciones para los grafos y los algoritmos más elementales, que incluyen sus recorridos, la descomposición en componentes conexas o, en su caso, fuertemente conexas y la ordenación topológica. El algoritmo de Dijkstra de caminos mínimos y el cálculo de árboles de recubrimiento mínimos se verán en el capítulo 6, por pertenecer al método voraz. El algoritmo de

Floyd de caminos mínimos se verá en el capítulo 7, por pertenecer al método de programación dinámica.

4.1. Terminología

Al igual que en el caso de los árboles, donde han aparecido terminos variados tales como hijos, hermanos, descendientes, ancestros, raíces, hojas, caminos, etc., también se ha desarrollado una terminología bastante amplia para los grafos. Algunos de estos conceptos se definen a continuación. Si no se indica lo contrario, se entenderá que son válidos tanto para grafos dirigidos como para no dirigidos:

Camino Secuencia de vértices $v_0, \ldots, v_{n-1}$, con $n \geq 1$, tales que, para todo $i \in \{0..n-2\}$, (v_i, v_{i+1}) es una arista. La *longitud* del camino es el número de vértices menos uno. Se admite que siempre existe un camino de longitud cero de un vértice a sí mismo.

Ciclo Camino de longitud no nula que comienza y termina en el mismo vértice. En los grafos no dirigidos, un ciclo ha de incluir al menos tres vértices. En los dirigidos puede constar también de uno o dos vértices. Cuando un grafo no tiene ciclos, diremos que es *acíclico*.

Subgrafo Diremos que $\langle V', A' \rangle$ es subgrafo del grafo $\langle V, A \rangle$, cuando es un grafo y, además, $V' \subseteq V$ y $A' \subseteq A$.

Grafo conexo (*solo grafos no dirigidos*) Para cada par de vértices del grafo, existe un camino que los une.

Componente conexa (*solo grafos no dirigidos*) Cada uno de los subgrafos conexos máximos de un grafo. Si el grafo es conexo, solo tendrá una componente. La descomposición de un grafo no dirigido en sus componentes conexas siempre es posible porque un solo vértice ya constituye un subgrafo conexo. Tal descomposición genera una partición de V en clases de equivalencia.

Grafo fuertemente conexo (*solo grafos dirigidos*) Dados dos vértices cualesquiera, $u,\ v \in V$, existe un camino de u a v y otro de v a u.

Componente fuertemente conexa (*solo grafos dirigidos*) Cada uno de los subgrafos fuertemente conexos máximos de un grafo. La descomposición de un grafo dirigido en sus componentes fuertemente conexas siempre es posible porque un solo vértice ya constituye un subgrafo fuertemente conexo. Tal descomposición genera una partición de V en clases de equivalencia.

Árbol libre Es un grafo no dirigido, conexo y acíclico. Si se destaca un vértice como *raíz*, el árbol libre se convertirá en un *árbol*.

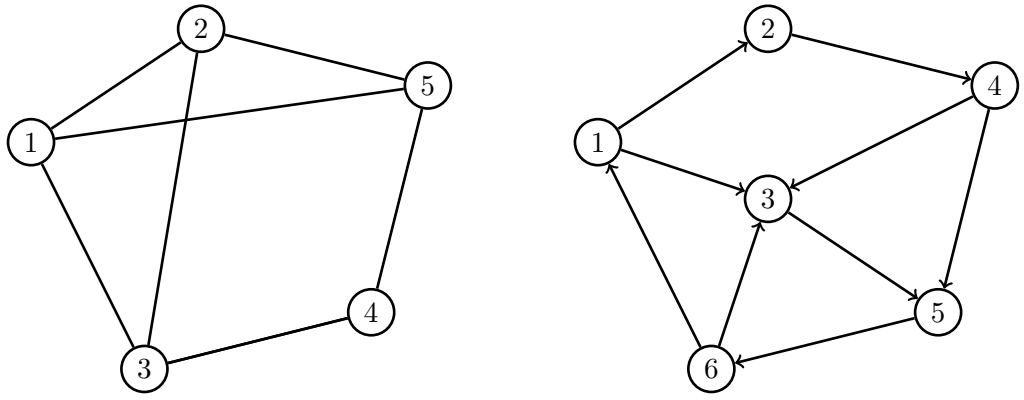

Figura 4.1: Ejemplos de grafo dirigido y no dirigido.

Árbol de recubrimiento de un grafo [1] (*solo grafos no dirigidos*) Es un árbol libre que es subgrafo del grafo original y que incluye todos sus vértices.

En la figura 4.1 se muestran ejemplos de grafos dirigidos y no dirigidos.

4.2. Representaciones de grafos

Existen numerosos algoritmos interesantes sobre grafos: por ejemplo, los dedicados a encontrar caminos de coste mínimo entre dos vértices, a decidir si un grafo es conexo o cíclico, a recorrer sus vértices o a encontrar el árbol de recubrimiento de menor coste. La eficiencia de estos algoritmos depende de la representación que elijamos para el tipo *grafo*.

Sin pérdida de generalidad, supondremos que el conjunto de vértices es el subrango de los enteros $0..n-1$. Si no fuera así y, por ejemplo, los vértices fueran objetos más complejos tales como cadenas de caracteres u otros, siempre será posible construir una tabla auxiliar que haga corresponder a cada objeto un natural único en dicho rango. Existen dos representaciones básicas para un grafo dirigido:

Matriz de adyacencia Un grafo dirigido G se representa mediante una matriz de booleanos:

```
type Graph = array2<bool>;
var G:Graph := new bool [n,n];
```

en la que $G[i,j] = \mathit{true}$ si y solo si existe la arista (i,j).

Listas de adyacencia Un grafo G se representa mediante un vector de listas de vértices:

[1] En inglés, *spanning tree*.

```
type Graph = array<List<nat>>;
var G:Graph;
```

en el que $G[i]$ nos da la lista de los vértices adyacentes al vértice i. Para las listas, puede utilizarse un tipo algebraico recursivo o una clase con nodos enlazados. En los algoritmos escritos en Dafny utilizaremos frecuentemente el tipo `set<nat>` para el conjunto de los vértices adyacentes, dado que su orden, en general, no será relevante.

En el resto del capítulo utilizaremos las variables n para referirnos al número de vértices del grafo y a para referirnos a su número de aristas. En general, $a \in O(n^2)$, pero en un grafo concreto a podría ser del orden de $O(n)$ o menor. Los costes de los algoritmos estarán por ello expresados en términos de ambos tamaños. La representación mediante matriz de adyacencia necesita un espacio en $O(n^2)$, lo cual puede ser inadecuado si el número a de aristas va a ser mucho menor que n^2. Si el grafo es no dirigido, solo se necesita la mitad de la matriz puesto que, si se tuviera entera, sería simétrica. Si el grafo fuese etiquetado o valorado, la matriz de adyacencia contendría etiquetas o pesos en lugar de booleanos y las celdas de las listas de adyacencia tendrían un valor adicional para almacenar la etiqueta o el peso de la arista.

Las preguntas ¿$(u, v) \in G$? o *coste* (G, u, v), que interrogan por la existencia o el coste de una arista dada, pueden contestarse, con la representación matricial, en un tiempo en $O(1)$. En cambio, la consulta *adj* (G, v), que devuelve el conjunto de los vértices adyacentes a un vértice dado v, necesita un tiempo en $O(n)$, pues la operación ha de recorrer una fila completa de la matriz.

La situación es inversa con la representación mediante listas de adyacencia: se accede a la lista de los vértices adyacentes a uno dado en tiempo constante. En cambio, consultar si una determinada arista (u, v) está presente en el grafo necesita recorrer la lista asociada al vértice u, lo que, en el caso peor, necesita un tiempo en $O(n)$. El espacio ocupado por esta representación es del orden de $O(a + n)$.

La eficiencia de los algoritmos que trabajan con grafos depende tanto del número n de vértices, como del número a de aristas y de la representación elegida para el grafo. En ocasiones, no es obvio cuál es la magnitud que mide mejor el tamaño del problema. Es frecuente emplear simultáneamente ambas cantidades en las expresiones de coste de estos algoritmos. Por ejemplo, veremos en el capítulo 6 que el algoritmo de Dijkstra para calcular los caminos de coste mínimo entre un vértice y todos los restantes de un grafo dirigido necesita un tiempo en $O(n^2)$ si se emplea la representación mediante matriz de adyacencia y un tiempo en $O(a \log n)$ o en $O(a + n \log n)$ si se emplean listas de adyacencia.

4.3. Recorridos

En esta sección supondremos que los algoritmos se aplican a grafos *dirigidos*. Si un grafo no dirigido se representa como uno dirigido en el que cada arista $u - v$ no dirigida se reemplaza por el par de aristas dirigidas $u \to v$ y $v \leftarrow u$, los algoritmos funcionarán igualmente, si bien su salida necesitará ser adaptada ligeramente para que sea comprensible por el usuario.

Al igual que en los recorridos de árboles, los algoritmos de recorrido de grafos visitan exactamente una vez cada vértice del mismo. Al no existir un vértice raíz, el recorrido parte de un vértice dado cualquiera. Una dificultad adicional es que los grafos pueden contener ciclos y, si no se presta atención a este problema, los algoritmos podrían ciclar eternamente, visitando repetidamente los mismos vértices. Para evitarlo, se puede utilizar, por ejemplo, un vector de booleanos, donde se marcan los vértices que ya han sido visitados. Aparece otra dificultad por el hecho de que los grafos, a diferencia de los árboles, pueden no ser conexos. Ello quiere decir que, partiendo de un vértice dado, solo se visitarán los vértices alcanzables desde él siguiendo las aristas del grafo, pero no el resto. Si se desea visitar todos, los algoritmos de recorrido no deben terminar al acabar la primera sesión de visitas, sino que deben investigar si quedan más vértices por visitar, consultando para ello el vector de marcado. En caso afirmativo, lanzan otra sesión de visitas, comenzando en el primer vértice no marcado. Proceden de este modo hasta que todos los vértices han sido marcados. A partir de lo dicho, es fácil imaginar que los algoritmos de recorrido sirven, con ligeras modificaciones, para resolver otros problemas relacionados:

- Detectar si un grafo no dirigido es conexo.
- Conocer el número de componentes conexas de un grafo no dirigido.
- Determinar los vértices y las aristas de dichas componentes conexas.
- Determinar si un grafo dirigido tiene ciclos.
- Ordenar topológicamente los vértices de un grafo dirigido acíclico.
- Calcular las componentes fuertemente conexas de un grafo dirigido.

Existen dos tipos de recorridos de grafos:

En anchura Similar al recorrido por niveles de un árbol: se visitan primero los vértices adyacentes de uno dado, antes de visitar los descendientes de estos.

En profundidad Similar al recorrido en preorden de un árbol: tras visitar un vértice, se recorre en profundidad su primer vértice adyacente no visitado, antes de pasar al segundo vértice adyacente. Se prosigue de igual modo con los restantes vértices adyacentes.