Implementaciones de gráficas y variantes

Introducción

Lo primero que debemos decidir al momento de trabajar con algoritmos en teoría de gráficas es cómo representaremos a una gráfica computacionalmente. En este capítulo discutimos algunas opciones de cómo hacerlo. Además, platicaremos de otros dos «sabores» de gráficas: las gráficas dirigidas y las gráficas con pesos. El brinco conceptual necesario para introducirlas es pequeño y añade mucha versatilidad a nuestras aplicaciones.

Listas de adyacencia vs. matrices de adyacencia

Recordemos que una gráfica consiste formalmente de vértices y aristas. Las aristas son parejas de vértices distintos, en donde no nos importa el orden. Cuando estamos trabajando con gráficas pequeñas es muy cómodo hacer una figura y entender todo lo que está sucediendo ahí. Sin embargo, para una computadora no es sencillo trabajar con un dibujo. Necesita de una mejor manera de representar la información de una gráfica para poderla procesar y ejecutar rápidamente consultas como saber si dos vértices son adyacentes.

Tomemos una gráfica G de orden n. Una posible de representarla es mediante una lista de vértices, y para cada uno de ellos una lista de sus vecinos. Si hacemos esto, diremos que la gráfica está representada mediante sus **listas de adyacencia**.

Otra posible forma de representar a G es mediante una **matriz de adyacencia**. Llamaremos a sus vértices $v_1,v_2,\ldots,v_n$. En una matriz M colocaremos en su entrada m_{ij} un 1 si v_iv_j es una arista y 0 si no. Una forma de representar una matriz de $n\times n$ computacionalmente es mediante una `Lista` de n elementos en donde cada elemento es una `Lista` de n elementos.

Ejemplo. Consideremos la gráfica (teórica) con vértices

$$V=\{1,2,3,4,5\}.$$
y aristas

$$A=\{\{1,2\},\{2,3\},\{2,4\},\{3,4\},\{3,5\},\{4,5\},\{1,5\}\}.$$
Una implementación (muy) básica en Python para representarla en términos de listas de adyacencia sería la siguiente:

```
V=[1,2,3,4,5]
L1=[2,5]
L2=[1,3,4]
L3=[2,4,5]
L4=[2,3,5]
L5=[1,3,4]
L=[L1,L2,L3,L4,L5]
```

Para representarla en términos de su matriz de adyacencia, hay que etiquetar los vértices. Como en Python la numeración de listas comienza en 0, hacemos eso para los vértices también:

$$v_0=1,v_1=2,v_2=3,v_4=5.$$
Bajo este etiquetado (que hay que recordar), la gráfica podría quedar representada como sigue:

```
M=[
    [0,1,0,0,1],
    [1,0,1,1,0],
    [0,1,0,1,1],
    [0,1,1,0,1],
    [1,0,1,1,0]
    ]
```

Observa que la matriz M es simétrica. Esto siempre sucede pues cuando hablamos de gráficas se tiene que v_i es adyacente a v_j

Observa que la matriz \(M\) es simétrica. Esto siempre sucede pues cuando hablamos de gráficas se tiene que \(u\) es adyacente a \(v\) si y sólo si \(v\) es adyacente a \(u\).

En cualquiera de los casos estamos guardando toda la información necesaria para recuperar por completo la estructura de \ \(\square\) (G\). Además, es sencillo diseñar un algoritmo que pase de una representación en listas de adyacencia a una en matriz de adyacencia. De esta manera, todo problema que podamos resolver con la primera lo podemos resolver con la segunda. Entonces, ¿cuál es la diferencia? Que habrá ventajas y desventajas dependiendo del problema a resolver o el algoritmo a ejecutar.

Pensemos en un par de ejemplos sencillos. Si queremos saber si dos vértices \(u\) y \(v\) son adyacentes, en la representación por listas de adyacencia tendríamos que buscar a \(v\) en la lista de \(u\). Ya que son listas, esto potencialmente podría tomar tiempo \(O(n)\). O quizás tiempo \(O(\log n)\) si las listas están ordenadas y hacemos búsqueda binaria. En una representación por matrices de adyacencia el tiempo de consulta requiere de dos lecturas de una `Lista`, lo cual corre en tiempo \(O(1)\).

Si queremos contar cuántas aristas tenemos, en el caso de las listas de adyacencia basta con contar cuántos elementos hay en cada lista, lo cual toma tiempo \(O(m)\): uno por cada arista. Sin embargo, si lo hacemos con matrices de adyacencia tomará tiempo \(O(n^2)\), pues hay que pasar por todas las entradas para saber si son \(0\) o \(1\). En una gráfica con pocas aristas conviene más lo primero.

Hay más formas de representar gráficas, pero nos quedaremos con las dos anteriores. Son las que más nos interesan para los algoritmos que discutiremos posteriormente.

Gráficas dirigidas

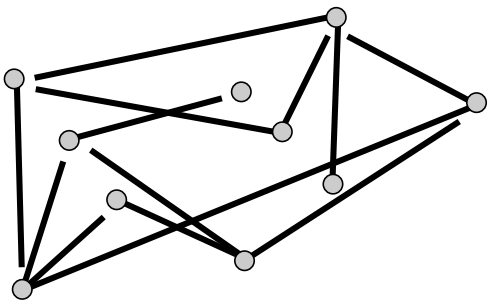
Hay situaciones en las que una relación entre parejas de objetos de un conjunto no es simétrica. Por ejemplo, puede que si tenemos dos ciudades \(A\) y \(B\) haya un camino de \(A\) a \(B\), pero no de \(B\) a \(A\). Si tenemos dos personas \(X\) y \(Y\) es posible que \(X\) admire a \(Y\), pero que \(Y\) no admire a \(X\). En estas situaciones es conveniente tener una estructura matemática que incluya la noción de «dirección». La definimos a continuación.

Definición. Una **gráfica dirigida** \(G\) consiste de un conjunto \(V\) de **vértices** y un conjunto \(E\) de **flechas**. Cada elemento de \(E\) es una pareja ordenada de elementos de \(V\).

Seguimos usando los términos **orden** y **tamaño** como en el caso no dirigido.

Observa que formalmente una flecha es una pareja del estilo \((u,v)\). Cuando no haya riesgo de confusión, la llamaremos simplemente la flecha \(uv\). Si \(G\) tiene a la flecha \(uv\), no necesariamente tiene a la flecha \(vu\).

Hay varias formas de hacer los dibujos de una gráfica dirigida. Por ejemplo, en algunos casos cuando se tiene tanto la flecha \(uv\) como la flecha \(vu\). En esos casos a veces se realiza el dibujo poniendo una flecha con dos puntas, a veces poniendo ambas flechas y a veces poniendo sólo el segmento \(uv\). Un dibujo de una gráfica dirigida podría ser el siguiente.



Muchas de las nociones no dirigidas se extienden naturalmente al contexto dirigido. Por ejemplo, un **camino dirigido** es una sucesión \(v_1,e_1,v_2,e_2,\ldots,e_{n-1},v_n\) en donde cada \(v_i\) es un vértice y cada \(e_i\) es la flecha \(v_1v_2\). Esto a su vez da una noción natural de **distancia dirigida** de un vértice \(u\) a un vértice \(v\) como la longitud del camino dirigido más corto que comience en \(u\) y termine en \(v\).

Podemos representar computacionalmente a las gráficas dirigidas con los equivalentes a listas de adyacencia o a matrices de adyacencia. Para el primer caso, si tenemos una flecha \(uv\), entonces ponemos a \(v\) en la lista de adyacencia de \(u\) (pero no

necesariamente a \(u\) en la de \(v\)). Para el segundo caso, colocamos en la entrada \((i,j)\) de la matriz un \(1\) si tenemos la flecha \(v_iv_j\) y un \(0\) si no. A diferencia del caso no dirigido, la matriz obtenida no será simétrica.

Gráficas con pesos

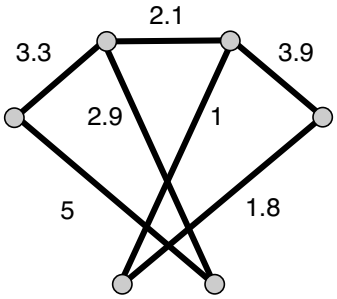
Otra situación de mucho interés es colocar números en los vértices o aristas de una gráfica para incluir información adicional de algún problema de aplicación. Por ejemplo, si tenemos una autopista entre dos ciudades \(A\) y \(B\), es disinto que sea de \(10\) kilómetros, a que sea de \(100\) kilómetros. Si los vértices de una gráfica representan personas, quizás queramos agregar la información de su edad o de su estatura.

A continuación trataremos el caso de gráficas en las que colocamos pesos a las aristas, pero se puede hacer una discusión análoga para considerar gráficas en donde se ponen pesos a los vértices.

Definición. Una **gráfica con aristas ponderadas** \(G\) consiste de un conjunto \(V\) de **vértices**, de un conjunto \(E\) de **aristas** y de una función \(w:E\to \mathbb{R}\). Cada elemento de \(E\) es una pareja ordenada de elementos de \(V\).

Es decir, básicamente consiste de tomar una gráfica y asignar un número real a cada arista. Dependiendo de la aplicación, es posible que los números que consideremos estén restringidos de alguna manera, como ser no negativos, o ser enteros.

Un dibujo de una gráfica con aristas ponderadas se parece a uno de una gráfica. Simplemente añadimos los pesos a cada una de las arisas. Hacemo esto o bien exactamente enmedio de las aristas, o bien suficientemente cerca de cada arista para que se entienda el etiquetado. Un ejemplo es el siguiente.



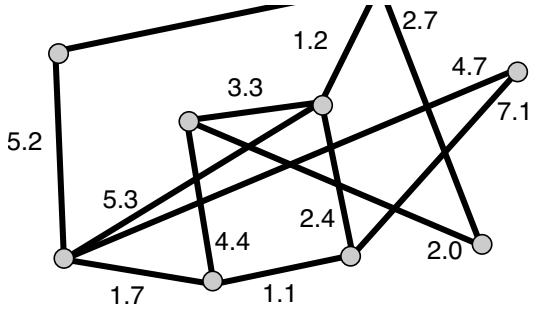
También las nociones sin pesos se extienden naturalmente al contexto de aristas ponderadas. Por ejemplo, si tenemos un camino de la gráfica, podemos definir su **peso** como la suma de los pesos en sus aristas. En el caso no dirigido la distancia entre dos vértices \(u\) y \(v\) está dada como la longitud del camino más corto entre ellos. En el caso dirigido podemos definir la **distancia ponderada** como el peso del camino más ligero posible que comience en \(u\) y termine en \(v\).

Para representar computacionalmente a las gráficas ponderadas podríamos seguir un enfoque de listas de adyacencia, almacenando para cada vértice \(u\) parejas del estilo \((v,w(u,v))\) para cada vecino \(v\) de \(u\). Esto complica un poco el acceso al peso de la arista. Otra posibilidad, quizás un poco más práctica, es seguir un enfoque de matrices de adyacencia, almacenando en la entrada \((i,j)\) de la matriz el peso de la arista \(v_iv_j\). Si hacemos esto, hay que tener cuidado de cómo representamos la falta de una arista. En un problema de carreteras y peajes no es lo mismo una arista \(uv\) de peso \(0\) (una carretera en donde no se pagan peajes), a que no haya arista \(uv\) (que no haya carretera entre las ubicaciones). En esos casos conviene incluir un símbolo adicional para la falta de arista.

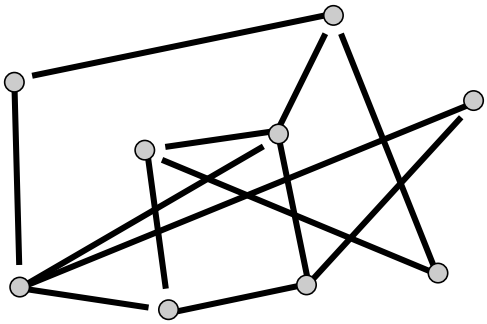
Tarea moral

Los siguientes problemas te ayudarán a practicar lo visto en esta entrada. Para resolverlos, necesitarás usar herramientas matemáticas, computacionales o ambas.

1. Pasa la siguiente gráfica ponderada \(W\) a su representación teórica y a una representación en matriz de adyacencia. Necesitarás ponerle nombre a los vértices.



2. Pasa la siguiente gráfica dirigida \(D\) a su representación teórica y a una representación en listas de adyacencia. Necesitarás ponerle nombre a los vértices. Una arista sin extremos de flecha indica que tienes las fechas en ambas direcciones.



3. Diseña un algoritmo para cada una de las siguientes tareas computacionales:
   - Pasar una gráfica de listas de adyacencia a matriz de adyacencia.
   - Pasar una gráfica de matriz de adyacencia a listas de adyacencia.
   - Pasar una gráfica dirigida a la gráfica no dirigida subyacente, es decir, aquella que tiene la arista \(uv\) si en la original está alguna de las flechas \(uv\) o \(vu\) (o ambas).
   - Pasar una gráfica con aristas ponderada a una sin pesos en la que sólo queden las aristas con peso mayor o igual que \(5\).
   - Pasar de una gráfica a una gráfica con aristas ponderada en donde cada arista tiene peso 1.

   Estudia tus algoritmos en correctitud, complejidad en espacio y complejidad computacional.

4. Sea \(A\) la matriz de adyacencia de una gráfica \(G\) con etiquetado de vértices \(v_1,\ldots,v_n\). Demuestra que la entrada \((i,j)\) de \(A^2\) cuenta la cantidad de caminos de longitud \(2\) entre \(v_i\) y \(v_j\). Generaliza esta observación para la matriz \(A^k\).
5. Crea una clase `Grafica` en Python que soporte operaciones de añadir vértices, quitarlos, añadir aristas, quitarlas y de dar los vecinos de un vértice.

Anterior
Algoritmos en teoría de gráficas

Siguiente
Uso básico de NetworkX

Por Leonardo Ignacio Martínez Sandoval