Representación de Grafos

Julieta Pages

Técnicas de Diseño de Algoritmos

FCEyN UBA

1C 2024

Esquema de hoy

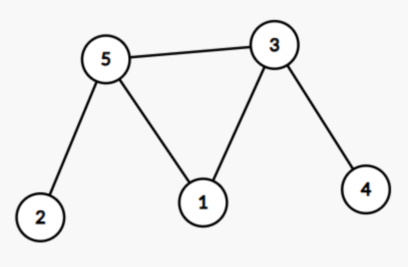
1. Repaso de definiciones
2. Representación de grafos
   - Lista de aristas
   - Lista de adyacencia
   - Matriz de adyacencia
3. Ejercicio
4. Bonus
   - Representación de árboles
   - Demostraciones en grafos

Definiciones

Grafo

Un grafo es un par (V, E) con V un conjunto de nodos (o vértices) y E conjunto de aristas (o ejes) de la forma (u, v) con u, v $\in$ V.

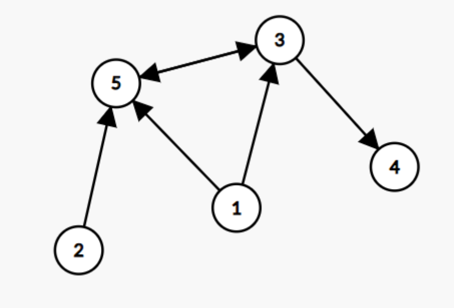
Si no se aclara, se asume que los ejes no son dirigidos, es decir que (u, v)=(v, u).



Digrafo

Es un grafo, es decir, un par (V, E) como antes, pero cuyas aristas están orientadas. Es decir que en este caso $(u, v) \neq (v, u)$

Puedo tener dos ejes entre dos nodos, uno en cada sentido.



Definiciones

Otros:

- Llamamos **multigrafo** a un grafo que permite tener múltiples aristas entre dos nodos.
- Llamamos **pseudografo** a grafos que permiten tener loops.
- Llamamos a un grafo **pesado** cuando G=(V, E, w) con w(e) una "función de pesos" que asigna a cada arista e=(u,v) un peso. (Observacion: se puede usar para simplificar multigrafos).

Por el momento nos vamos a concentrar en grafos y digrafos.

Más definiciones...

- **Recorrido:** una sucesión de vértices y aristas del grafo
- **Camino:** un recorrido que **no** pasa dos veces por el mismo vértice.
- **Circuito:** un recorrido que empieza y termina en el mismo nodo
- **Ciclo o circuito simple:** un circuito que **no** repite vértices. (Nota: para grafos, no consideramos como válido al ciclo de longitud 2)
- **Longitud:** la longitud de un recorrido se nota $l(P)$ y es la cantidad de *aristas* del mismo.
- **Distancia entre dos vértices:** longitud del camino más corto entre los vértices (si no existe se dice ∞).
- Un nodo es **adyacente** a otro si están conectados.
- Una arista es **incidente** a un nodo si conecta dicho nodo con algún otro.

Notación

- En general vamos a denotar con n la cantidad de nodos que tiene al grafo, y con m a la cantidad de aristas.

Para grafos:

- $N(v)$: vecindario del vértice v (conjunto de nodos adyacentes a v).
- $N[v]$: vecindario cerrado de v, $N[v] = N(v) \cup v$.
- $d(v)$: el grado de v (cantidad de vecinos), $d(v) = |N(v)|$.

Para digrafos:

- $N^{in}(v)$ y $N^{out}(v)$ son los vecindarios de entrada y salida respectivamente.
- $d^{in}(v)$ y $d^{out}(v)$ son los grados de entrada y salida respectivamente.

Operaciones

¿Qué operaciones podríamos llegar a querer hacer sobre un grafo?

- Inicializar un grafo.
- Agregar o sacar un nodo del grafo.
- Agregar o sacar una arista del grafo
- Obtener el vecindario de un nodo v.
- Evaluar si dos aristas u y v son adyacentes.

Representaciones: versión abstracta

Por conveniencia, vamos a suponer que todos los nodos del grafo son números de [1, n].

Podríamos representar el grafo de dos maneras:

- **Conjunto de aristas:** guardamos el conjunto E del grafo.
- **Diccionario:** le asociamos a cada vértice 'v' su vecindario $N(v)$.

Luego, cada una de estas representaciones se podrían implementar haciendo uso de distintas estructuras como las que vieron en Algoritmos (ex Algo2), como listas, vectores, AVL, tablas de hash, etc.

Representaciones: lo más común

Principalmente vamos a utilizar las siguientes estructuras de representación:

- Lista de aristas
- Lista de adyacencia
- Matriz de adyacencia

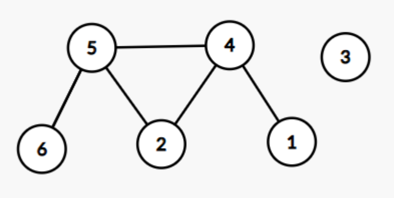
Veamos cada una de estas representaciones y sus diferencias.

Ejemplo: lista de aristas

lista de aristas

El conjunto de aristas como una secuencia (lista).

Tomemos el siguiente grafo como ejemplo:



Lista de aristas:

$$\{(6,5),(5,2),(2,4),(5,4),(4,1)\}$$

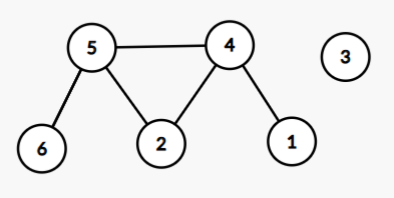
Nota 1: normalmente esta va a ser la única representación con la que se van a expresar los inputs de grafos.
Nota 2: si junto con la lista se pasa el tamaño del grafo, como por convención las etiquetas de los vértices están numeradas 1...n podemos deducir qué vértices no tienen vecinos.

Ejemplo: lista de adyacencia

Lista de adyacencia

El diccionario es un vector y los vecindarios son listas de tamaño d(v) conteniendo a los nodos vecinos.

Tomemos el siguiente grafo como ejemplo:



Lista de adyacencia:

Nodo : lista de vecinos

1 : 4

2 : $4 \longrightarrow 5$

3 :

4 : $5 \longrightarrow 1 \longrightarrow 2$

5 : $2 \longrightarrow 6 \longrightarrow 4$

6 : 5

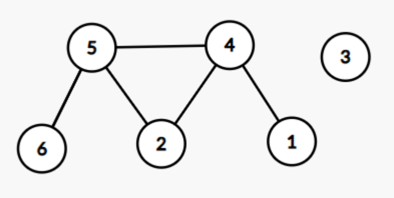
Nota: Se pueden hacer cosas como ordenar los vecindarios, pero es caro mantenerlo si el grafo cambia.

Ejemplo: matriz de adyacencia

Matriz de adyacencia

El diccionario y los vecindarios son vectores de tamaño n. Resultando así en una matriz de nxn donde $M_{ij} = 1$ si los vértices i y j son adyacentes y $M_{ij} = 0$ si no.

Tomemos el siguiente grafo como ejemplo:



Matriz de adyacencia:

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Complejidades

Las complejidades para las representaciones vistas quedarían:

	lista de aristas	matriz de ady.	listas de ady.
construcción	$O(m)$	$O(n^2)$	$O(n+m)$
adyacentes	$O(m)$	$O(1)$	$O(d(v))$
vecinos	$O(m)$	$O(n)$	$O(d(v))$
agregarArista	$O(m)$	$O(1)$	$O(d(u)+d(v))$
removerArista	$O(m)$	$O(1)$	$O(d(u)+d(v))$
agregarVértice	$O(1)$	$O(n^2)$	$O(n)$
removerVértice	$O(m)$	$O(n^2)$	$O(n+m)$

Nota: como el tamaño del grafo está dado por su cantidad de nodos y aristas, una complejidad O(n+m) es lineal respecto al tamaño del grafo.

Aclaraciones

- La complejidad de pedir los vecinos queda en O(d(v)) porque, si bien podríamos devolver el puntero al inicio de la lista en O(1), eso rompe el encapsulamiento. Sería mejor devolver una copia del vecindario para garantizar que no se rompa la estructura externamente (ej. si le quitan elementos la lista perderíamos adyacencias o si es un grafo no dirigido se rompen invariantes).
- Si bien decimos que la complejidad de agregar vértices de una lista de adyacencia es O(n) por el costo de reinicializar el array, si se usaran arrays dinámicos quedaría en O(1) amortizado.

Eligiendo una representación

Qué representación conviene usar va a depender de:

- Las características del grafo (por ejemplo si es ralo o denso).
- Para qué lo vamos a querer usar y qué complejidades queremos para sus operaciones.

Ejercicio 1[1]

Ejercicio de parcial

Sea D un digrafo representado como una lista de aristas. Queremos determinar en tiempo O(m+n) cuáles aristas cumplen que su opuesta no está en D. Es decir, para una arista v $\to$ w, no está w $\to$ v.

- Para pensar:
  ¿Qué significa que la arista w $\to$ v esté en D, respecto de D^t (el digrafo traspuesto de D)?

[1] Tomado en el primer recuperatorio del 1C 2022

Ejercicio 1: solución con lista de aristas

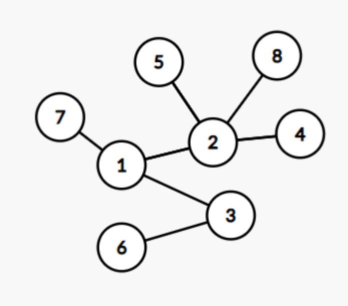
- Transformo el problema en uno más fácil $\rightarrow$ O(m)
  - Genero E(D^t)
  - Genero un 'multidigrafo' S concatenando D^t con D.
    Notar que si una arista está repetida, es porque en D tenía la ida y la vuelta.
- Ordenar en tiempo lineal $\rightarrow$ O(n+m)
  - Pensamos las aristas como palabras de 2 letras en el alfabeto: [1, n]. Y como n está acotado, podemos usar bucket sort para ordenar por una coordenada.
  - Para ordenar las aristas entonces, se puede usar radix sort (ordenar primero por la segunda coordenada y luego por la primera).
- Busco repetidos en una lista $\rightarrow$ O(m)
  - Toda arista que no esté repetida en S va a ser porque aparecía una sola vez en D.
  - Recorro la lista ya ordenada, y me quedo con las aristas que sean distintas a la anterior y a la siguiente.

Representación de árboles

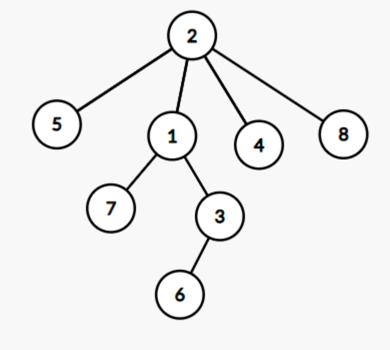
Árbol

Un árbol es un grafo conexo, acíclico y con n-1 aristas.
Nota: alcanza con saber que cumple con dos de dichas propiedades para afirmar que un grafo es un árbol.

Por ejemplo es un árbol:



Si lo **enraizamos** por ejemplo en el nodo 2 nos quedaría:

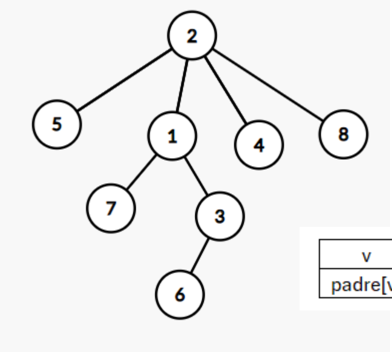


Representación de árboles

¿Podemos aprovecharnos de esto para representarlos de una manera más compacta? ¡Sí!

Podemos tener una función *padre* tal que padre[v] es el (único) padre de v para todo v que no es la raíz. Para la raíz *r* podríamos poner padre[r] $=$ r por ejemeplo.

Con el ejemplo anterior:



v	1	2	3	4	5	6	7	8
padre[v]	2	2	1	2	2	3	1	2

Demostraciones

Hay distintas maneras de demostrar que una afirmación es correcta. Algunas opciones que tenemos son las siguientes:

- Por inducción.
- Por absurdo.
- Constructivamente ($p \rightarrow q$).
- Por contrarecíproco ($\neg q \rightarrow \neg p$).

Las demos se pueden dividir en casos, y probar cada uno como sea conveniente.

Qué método conviene usar va a depender de cada ejercicio. Es algo que con la práctica un le va tomando la mano.

Inducción en grafos

En grafos muchas veces vamos a tener que hacer inducción en la cantidad de nodos o aristas de un grafo.
Tengan en cuenta que:

- **NUNCA:** tomo un grafo G_k de k vértices (o k aristas) y digo que si una propiedad P se prueba para este grafo, agregándole un v vértice (o arista) sigue valiendo P para G_{k+1}. (*)
- **SIEMPRE:** tomo un grafo G_{k+1} de k + 1 vértices (o k + 1 aristas) con ciertas características y le saco un vértice (o arista) con algún tipo de estrategia particular y veo que cumple P. Luego, agrego el vértice o arista y veo que sigue cumpliendo P.

(*) el problema con esto es que estamos queriendo probar la propiedad para todo grafo G_{k+1} de k+1 nodos (o aristas). Sin embargo al tomar un grafo G_k y agregarle un nodo o arista estariamos solo considerando los grafos G_{k+1} que tenían a G_k como subgrafo, y eso no necesariamente eso abarca a todo G_{k+1} posible.

Ejercicio 2

Veamos un ejemplo sencillo:

Ejercicio 2

Probar que si G es un grafo de n nodos y tiene al menos n ejes, entonces tiene un ciclo.

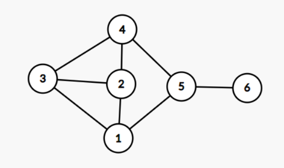
Ejercicio 2: solución

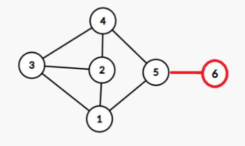
- **Hipótesis inductiva:** P(n)= para todo grafo de n nodos vale que si tiene al menos n ejes entonces tiene un ciclo.
- ¿Vale P(1)? El primer caso interesante es P(3), que también vale.
- Para el caso inductivo, queremos probar que $P(n) \rightarrow P(n+1)$. Es decir, tenemos que tomar un grafo cualquiera de n+1 nodos, y demostrar (aprovechando que vale P(n)) que si tiene n+1 o más ejes entonces tiene un ciclo.

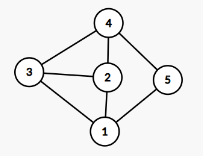
Ejercicio 2: solución

- Sea G un grafo con n+1 nodos, y al menos n+1 ejes. Vamos a separar en dos casos:
- Hay un nodo v de grado a lo sumo 1: entonces G v es un grafo de n nodos, con al menos n ejes. Luego, tiene un ciclo (por la hipótesis inductiva). Ese ciclo también está en G.

  A modo de ejemplo:







Ejercicio 2: solución

- Caso contrario, todos los nodos tiene al menos grado 2. Se puede demostrar fácilmente entonces tiene al menos un ciclo:
  La idea es, podemos tomar un vértice v_1, movernos a otro vértice v_2, y luego siempre vamos a poder movernos a otro vértice v_i por una arista distinta a la que usamos para llegar.
  Pero como la cantidad de nodos es finita, si podemos ir de nodo en nodo de manera infinita, eventualmente vamos a tener que repetir un nodo. Por lo tanto existe un ciclo.