Notas de la clase 3 – introducción a grafos

Francisco Soulignac

23 de marzo de 2019

Aclaración: este es un punteo de la clase para la materia AED3. Se distribuye como ayuda memoria de lo visto en clase y, en cierto sentido, es un reemplazo de las diapositivas que se distribuyen en otros cuatrimestres. Sin embargo, no son material de estudio y no suplanta ni las clases ni los libros. Peor aún, puede contener "herrorez" y podría faltar algún tema o discusión importante que haya surgido en clase. Finalmente, estas notas fueron escritas en un corto período de tiempo. En resumen: **estas notas no son para estudiar sino para saber qué hay que estudiar**.

Tiempo total: 190 minutos

1. Motivación (40 mins)

1.1. Definición inicial

- Formalmente, un *grafo general* G es una tripla (V, E, ϕ) donde:

  ▷ V es un conjunto de *vértices* y E es un conjunto de *aristas*. A los vértices a veces se los llama *nodos* o *puntos* y a las aristas se los llama *ejes* o *arcos*. En teoría, la única propiedad de los vértices y aristas es que sean distinguibles. En la práctica, los mismos representan entidades y sus relaciones (ya veremos ejemplos).

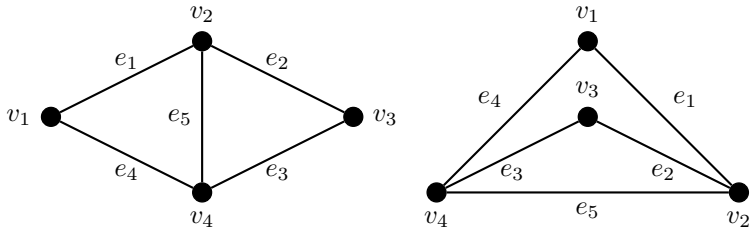
  ▷ ϕ es una función en $E \to V \times V$ donde cada imagen se interpreta como un par no ordenado. En lugar de escribir $\{v, w\}$ a un par no ordenado, lo escribimos vw

- Informalmente, $G = (V, E, \phi)$ es simplemente otro nombre para la relación dada por ϕ
- Más informalmente, G es un conjunto de puntos unidos por lineas, como se suelen representar gráficamente.

  ▷ En la representación gráfica, cada vértice v ocupa un punto p_v del plano y hay una línea que une p_v con p_w etiquetada con e para toda arista e con $\phi(e) = vw$. Si bien G tiene infinitas representaciones gráficas, cada una de ellas define *unívocamente* a G.

Representaciones gráficas de grafos generales

Dos representaciones del grafo (V, E, ϕ) con $V = \{v_1, \dots v_4\}$, $E = \{e_1, \dots, e_5\}$, y $\phi = e_i \to v_i v_{i+1}$ para $1 \leq i \leq 4$ y $e_5 = v_2 v_4$.



- La *teoría de grafos* es el estudio de los grafos generales (y otras estructuras aún más genéricas) desde un punto de vista matemático. Si bien los grafos generales son simplemente relaciones (en cierta medida), la teoría de grafos se enfoca en ciertas propiedades estructurales que transforman el objeto de estudio en otro distinto al de las relaciones matemáticas, aportando nueva terminología y problemas.

- La *teoría algorítmica de grafos* es el estudio algorítmico de problemas que involucran grafos y cuyas soluciones explotan la teoría de grafos.

- Los problemas de grafos son modelos (i.e. abstracciones) de problemas de la vida real. Por ende, su estudio abstracto aporta soluciones a problemas de diversas disciplinas.

- El **núcleo** de esta materia es el estudio de la teoría algorítmica de grafos.

1.2. Algunos problemas históricos que se modelan con grafos generales

1.2.1. Protección del imperio romano

- Defensa del imperio romano en el S. III [1, 5, 13, 17].

- Cuatro grupos de legiones para defender seis regiones. A cada grupo se lo puede representar en un mapa con una piedra.

- Una región esta *asegurada* cuando contiene una piedra y es *asegurable* si está conectada a una región que tiene una piedra.

- Las piedras se pueden mover en un paso de una región a otra cuando existe un camino.

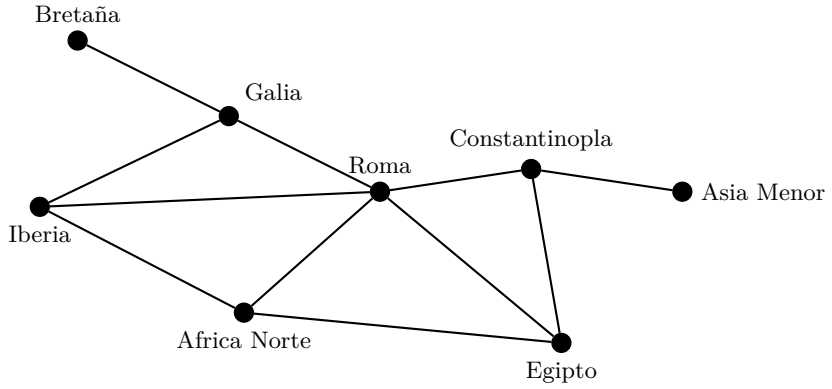
- Sin embargo, la estrategia romana consiste en mover una piedra de una región v a otra w cuando hay dos piedras en v. De esta forma, v queda defendida cuando se despliega una piedra hacia w.

- Constantino decidió mantener, en condiciones de paz, dos piedras en Roma y dos en Constantinopla [1]. Con esta solución, se requieren 4 movimientos para defender Britania de acuerdo a las reglas.

- Hoy se sabe que hay mejores estrategias [1, 17], aunque existe el problema de defender múltiples ataques en simultáneo o, incluso, una secuencia de ataques.

- El problema de la defensa romana se puede modelar con un grafo general donde: cada región se representa con un vértice y cada ruta directa se representa con una arista que une los vértice de las regiones correspondientes.

Grafo de caminos del imperio romano



- El problema de asegurar todas las regiones (obviando el hecho de que hayan dos piedras en una región para moverse) coincide con el problema de dominación en grafos simples. El mismo consiste en la selección un conjunto de vértices D de forma tal que todo vértice fuera de D esté unido con una arista a un vértice de D.

- En el caso particular estudiado, el grafo en cuestión admite un dibujo sin que se crucen líneas.

- En teoría de grafos se estudian distintos problemas de dominación, muchos de los cuales involucran juegos de estrategia de defensa.

1.2.2. Puentes de Königsberg

- Traducción y refraseo de [7] de acuerdo a su traducción al inglés en [4]:

  ▷ Interés en la "geometría de posición" que estudia "relaciones entre las posiciones sin importar sus magnitudes".

  ▷ "No existe ninguna definición satisfactoria de los problemas de geometría de posición ni de los métodos usados para resolverlos".

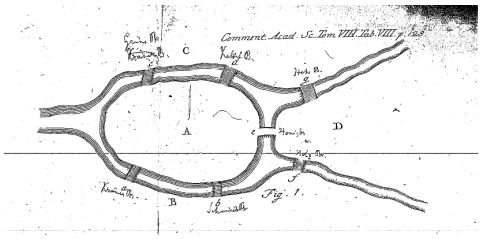
  ▷ "En la ciudad de Königsberg hay un río como el de la Figura 1 cruzado por 7 puentes a, b, c, d, e, f y g. La pregunta es si una persona puede planear un recorrido que cruce todos los puentes una vez pero no mas que una vez."

  ▷ "Me formulo un problema general a mi mismo: dada cualquier configuración de un río y de sus ramificaciones, al igual que cualquier cantidad de puentes, determinar si es posible o no cruzar cada puente exactamente una vez".

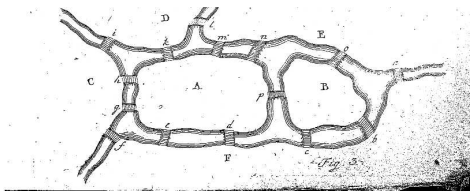
  ▷ "El problema de los 7 puentes puede ser resuelto a mano, pero resulta tedioso y difícil por la cantidad de combinaciones posibles, y puede no ser factible en el caso general. [...] Propongo un método más simple".

  ▷ "Uso letras en mayúscula A, B, C y D para designar las porciones de tierra separadas por el río. Cuando una persona va de un área A a otra B, designo este cruce con las letras AB. [...] A dos cruces sucesivos AB y BD los denoto ABD [...]".

Mapas del artículo de Euler [7]



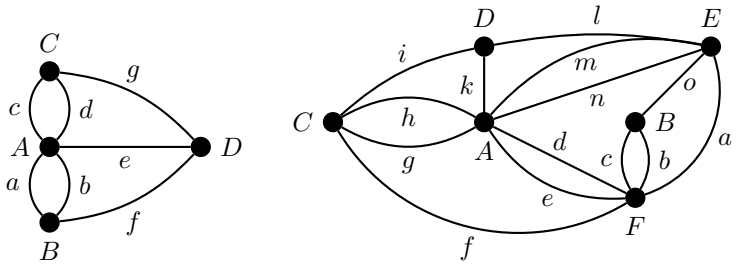
Königsberg



Generalización

- La idea fundamental es abstraer toda la porción de tierra (con sus posibles caminos internos) a puntos y los puentes a lineas que lo conectan.

Grafos de los puentes de Königsberg



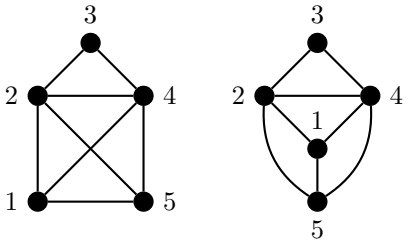
Königsberg Generalización

▪ Más aún, no importan las posiciones de los objetos, sólo sus relaciones. Si se cambia el dibujo pero se mantienen invariantes las relaciones, el problema no cambia.

▷ Considerar el problema de dibujar sin levantar el lápiz. Los problemas de dibujar cada una de las siguientes figuras son equivalentes a pesar de ser figuras distintas.

Grafos que se pueden dibujar sin levantar el lápiz

Formalmente, son dos representaciones gráficas del mismo grafo: una se puede dibujar sin levantar el lápiz del papel si y sólo si la otra también se puede dibujar sin levantar el lápiz del papel.



▪ A este proceso hoy lo llamamos modelar con grafos (generales). Actualmente se llaman vértices a los puntos, aristas a las líneas y camino (circuito) euleriano a un camino que pasa exactamente una vez por cada arista (y empieza y termina en el mismo vértice).

▪ Este artículo, presentado en 1735, se considera uno de los orígenes de dos disciplinas: la teoría de grafos y la topología.

▪ Hoy en día se estudian distintos problemas en donde es necesario recorrer todas las aristas de un grafo. Pensar, por ejemplo, en la recolección de residuos: tenemos que recorrer todas las calles al menos una vez con una flota de camiones. Minimizar la cantidad total de calles que se recorren más de una vez es un buen inicio.

1.2.3. Recorridos de caballos y el juego icosiano

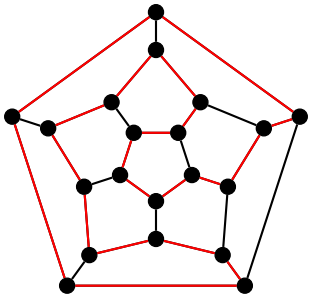
▪ ¿Puede un caballo recorrer todas las celdas de un tablero de ajedrez empezando y terminando en la misma celda? A este problema se lo conoce como el *problema del recorrido del caballo* (knight's tour).

▪ Problema conocido desde el S. IX (c.f. [21]). Una solución fue propuesta por Euler en 1759 con una estrategia de búsqueda local [8] (ver [21]).

▪ En 1857, Hamilton inventó el *juego icosiano*. El mismo consiste en recorrer los vértices de (una proyección 2D de) un dodecaedro regular, moviéndose por las aristas que los unen (con varios tipos de juegos y reglas). El juego se comercializó en Europa por John Jacques and Sons en dos formatos: el *icosian game* y *the travellers dodecahedron or a voyage around the world*. Las imágenes se pueden consultar en `https://www.puzzlemuseum.com/month/picm02/200207icosian.htm`.

Proyección del dodecaedro

Las aristas rojas representan un "recorrido cerrado" que pasa exactamente una vez por cada vértice.



▪ La razón del juego surge del estudio de estos circuitos y su relación con la simetría de los icosaedros (c.f. [4]).

▪ El problema de estudiar ciclos en poliedros ya había sido estudiado por Kirkman (c.f. [4]) que dio ejemplos de poliedros que no tienen estos circuitos.

▪ Todos estos problemas se pueden modelar con un grafo. El problema es determinar si existe un recorrido que pase una vez por cada vértice y que termine en el vértice inicial. En el caso de knight's tour, cada celda del tablero se representa con un vértice y hay una arista entre dos vértices cuando el caballo puede saltar entre estas celdas.

▪ A los ciclos que recorren exactamente una vez cada vértice se los llaman *ciclos hamiltonianos* en honor a Hamilton. Aquellos grafos que tienen un ciclo hamiltoniano se llaman grafos hamiltonianos.

▪ Es facil ver que si todos los pares de vértices tienen una arista, entonces el grafo *completo* resultante tiene un ciclo hamiltoniano. De hecho, cualquier permutación de los vértices es un ciclo hamiltoniano.

▪ En el problema del *viajante de comercio* (TRAVELLING SALESMAN), todas las aristas del grafo tienen un costo que indica qué tan caro es transitar la arista. El objetivo es encontrar un ciclo hamiltoniano de costo mínimo. Este problema, que nadie sabe resolver en tiempo polinomial, se usa de base para probar distintas técnicas algorítmicas.

▪ Los *problema de ruteo de vehículos* son problemas de logística con alto impacto económico y ambiental. Consisten el determinar de qué forma visitar todos los clientes (vértices) con una flota de vehículos a fin de satisfacer su demanda, minimizando el costo de las rutas. Estos problemas son generalizaciones del problema de viajante de comercio, donde la flota esta conformada por un sólo vehículo.

1.2.4. El origen del término "grafo"

▪ El término grafo fue usado por primera vez por Sylvester [18], en un artículo que buscaba relacionar las estructuras químicas y el álgebra. El término se refiere a las representaciones gráficas de estos objetos.

▪ Siendo una herramienta para el estudio de distintas relaciones entre objetos, no sorprende que los mismos hayan encontrado aplicaciones en todas las áreas del conocimiento, incluyendo la química [20].

- Con el crecimiento del poder de cómputo en el mundo, la teoría algorítmica de grafos cobra cada vez un peso mayor dentro del área que impacta a todas las disciplinas del conocimiento.

Intervalo (10 mins)

2. Terminología básica de grafos I (80 mins)

Como mencionamos previamente, la teoría de grafos provee una terminología apropiada para estudiar relaciones entre entidades. Desde un punto de vista computacional, surgen una cantidad monumental de problemas algorítmicos. En esta sección presentamos parte de la terminología y algunos problemas a modo de motivación. Esta sección sirve de ayuda memoria y no vale la pena recordarla de memoria ya que se aprende con la práctica.

2.1. Grafos, digrafos y grafos mixtos

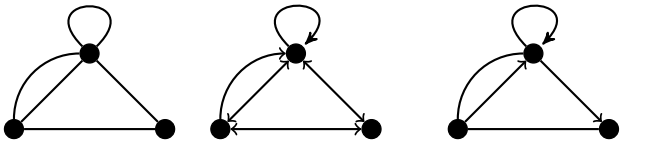
- Recordemos que un *grafo general* G es una tripla (V, E, ϕ) donde V es un conjunto de *vértices*, E es un conjunto de *aristas* y $\phi\colon E \to V \times V$ donde cada elemento de la imagen de ϕ se considera un par no ordenado.

- Podemos representar G gráficamente poniendo un punto p_v por cada $v \in V$ y una linea l_e entre los dos extremos de $\phi(e)$ por cada $e \in E$. Si se etiquetan los puntos y lineas, entonces tenemos el dibujo representa unívocamente al grafo.

- Un *grafo dirigido general*, o *digrafo general*, es lo mismo que un grafo, pero donde cada elemento de la imagen de ϕ es un par ordenado. Gráficamente, las aristas se representan con flechas, donde una flecha va de v a w cuando $f(e) = vw$.

- En el caso más general, un *grafo mixto general* tenemos el par $\phi(e)$ se interpreta como ordenado o no ordenado. Gráficamente, hay lineas y flechas.

Representaciones gráficas de variantes de grafos generales



Grafo general Digrafo general Grafo mixto general

- Dos aristas e_1 y e_2 con $\phi(e_1) = \phi(e_2)$ se llaman *duplicadas*.

- Un arista e con $\phi(e) = vv$ para algun $v \in V$ se llama *loops*.

Tipos de aristas

duplicadas duplicadas no duplicadas loop

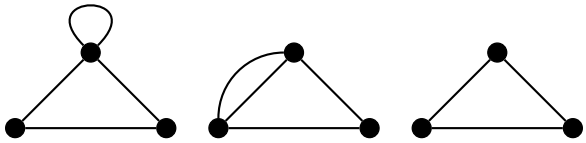
- De acuerdo a qué aristas puede tener un (di)grafo (mixto), tenemos distintas configuraciones que se suelen estudiar:

(di)grafo (mixto) simple: no contiene loops ni aristas duplicadas.

multi(di)grafo (mixto): puede contener aristas duplicadas pero no loops.

seudo(di)grafo (mixto): puede contener loops pero no aristas duplicadas.

Tipos de grafos



- Salvo en algunos casos muy puntales de la práctica, en AED3 nos restringimos a estudiar grafos y digrafos simples y finitos y, ocasionalmente hablamos de grafos mixtos simples. En todos los casos, vamos a omitir los términos "simple" y "finito".

- Notar que para describir las aristas alcanza con decir cuál es el conjunto de pares de vértices relacionados. Entonces, podemos simplificar la definición de grafo y digrafo.

- Formalmente, un (di)grafo es un par $G = (V, E)$, donde V es un conjunto finito y no vacío de *vértices* y E es un subconjunto de pares no ordenados (ordenados) de V, llamados *aristas* o *arcos*.

- Vamos a escribir $V(G)$ para denotar a V y $E(G)$ para denotar a G. Asimismo, se suele denotar $n = |V(G)|$ y $m = |E(G)|$. Observar que esta notación es ambigua y supone que G está claro por contexto.

- A cada par ordenado (v, w) y a cada par no ordenado $\{v, w\}$ lo vamos a denotar vw, sea o no que pertenezca a $E(G)$.

- Para un digrafo $D = (V, E)$, su *grafo subyacente* es $G = (V, E)$. Notar el uso de los términos "grafo" y "digrafo"; lo único que cambia es que nos olvidamos que los pares son ordenados. Gráficamente, cambiamos flechas por lineas.

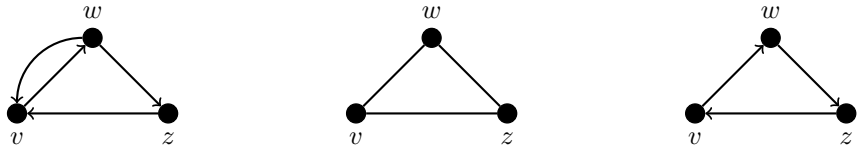
- Un *grafo orientado* es un digrafo tal que $vw \in E$ sólo si $wv \notin E$.

- Para un grafo G, cualquier grafo orientado D cuyo grafo subyacente es G se llama una *orientación* de G. Gráficamente, una orientación consiste en reemplazar cada linea por una flecha.

- Notar que un digrafo tiene un único grafo subyacente, pero un grafo tiene una cantidad exponencial de orientaciones.

Grafos y orientaciones

A la izquierda se ve el digrafo D cuyas aristas son wv, vw, wz y zv. Su grafo subyacente G, que se muestra en el centro, tiene el mismo conjunto de aristas. Dado dado que $wv = vw$, sólo una copia sobrevive en G. El grafo dirigido de la derecha es una de las 2^3 posibles orientaciones de G.



2.2. Adyacencias y vecindarios

- Si $vw \in E(G)$, entonces decimos que v y w son *adyacentes* o *vecinos*. Asimismo, decimos que v y w *inciden* en la arista vw.

- Si G es un digrafo, v es un *vecino de entrada* a w y w es un *vecino de salida* de v. Similarmente, v es la *cola* y w es la *cabeza* de vw

- El *vecindario (abierto)* de un vértice v es el conjunto $N_G(v) = \{w \in V(G) \mid vw \in E(G)\}$ formado por los vecinos de v, mientras que su *vecindario cerrado* es $N_G[v] = N(v) \cup \{v\}$.

- Si G es un digrafo, también podemos definir los *vecindarios de entrada* $N^{\text{in}}_G(v) = N^-_G(v)$ y de *salida* $N^{\text{out}}_G(G) = N^+_G(v)$ formados por los vecinos de entrada y salida de v, respectivamente.

- El *grado* de v es $d_G(v) = |N_G(v)|$, es decir, la cantidad de vecinos que v tiene. En digrafos, también podemos definir el *grado de entrada* $d^{\text{in}}_G(v) = d^-_G(v) = |N^{\text{in}}_G(v)|$ y el *grado de salida* $d^{\text{out}}_G(v) = d^+_G(v) = |N^{\text{out}}_G(v)|$.

- Cuando G esta claro por contexto, omitimos el subíndice G de d y N.

Adjacencias y vecindarios

Los vértices v y w son adyacentes e inciden en $e = vw$. En el caso dirigido, v es la cola y w la cabeza de $e = vw$. Por definición, $v \in N(w)$, $w \in N(v)$ y, en el caso dirigido, $v \in N^-(w)$ y $w \in N^+(v)$.

Observación 1. *Para todo (di)grafo G ocurre que $\sum_{v \in V(G)} d^+(v) = \sum_{v \in V(G)} d^-(v) = m$.*

Demostración. Notemos que cada arista tiene exactamente una cola (cabeza). Ergo, una forma de recorrer una vez cada arista es recorriendo una vez cada vértice v y visitando las $d^+(v)$ ($d^-(v)$) aristas de salida (entrada) en las que incide. □

Corolario 1. *Para todo (di)grafo G ocurre que $\sum_{v \in V(G)} d(v) = 2m$*

Demostración. Porque, considerando cualquier orientacion de G (si G no es ya un digrafo), $d(v) = d^+(v) + d^-(v)$. □

2.3. Representación computacional de (di)grafos I: lista de aristas

- La forma simple de representar a un (di)grafo es siguiendo la definición. Un grafo es un par formado por un conjunto de vértices V y un conjunto de aristas E.

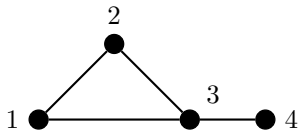
- Cuando la semántica de V es irrelevante (como en la materia[1]), alcanza con representar a V como los números entre 0 y $n - 1$. Con lo cual, alcanza con decir qué cantidad de vértices hay.

- Por otra parte, antiguamente E se representaba como una lista y, por este motivo, a esta representación se la llama *lista de aristas*. Obviamente, la lista se puede reemplazar por cualquier otra estructura.

[1] Cuando G representa la relación entre objetos de un dominio, es común extender la estructura de datos con atributos para los vértices y las aristas. Esta **no es una materia de objetos**, razón por la cual vamos a ignorar el diseño de estructuras de datos genéricas para problemas de grafos. A quién tenga interés, existen distintas bibliotecas "objetosas" de grafos.

Representación con lista de aristas

El conjunto $\{(3,1),(1,2),(4,3),(3,2)\}$ representa al siguiente grafo.



- En C++, una forma simple de representar G con lista de aristas es la siguiente.

Representación con lista de arista

```cpp
using edge = std::pair<int, int>;
using graph = std::vector<edge>;
int N;
graph G;
```

- La ventaja de la lista de aristas es que ocupa $O(m)$ espacio para representar un grafo con n vértices y m aristas. Esto es óptimo (salvo una constante) y, por lo tanto, se suele usar esta representación para definir el input de los algoritmos de grafos.

- La desventaja obvia es que la lista de aristas no es muy útil para operar. Por ejemplo, tenemos que recorrer todas las aristas para conocer el vecindario de un vértice.

2.4. Representación de grafos II: lista de adyacencias

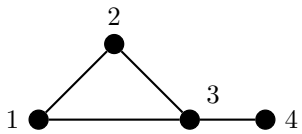
- Otra forma posible de representar un grafo $G = (V, E)$ es usando un diccionario $\mathbf{N}$ cuyo conjunto de claves es V tal que $\mathbf{N}[v] = N(v)$ para todo $v \in V$. En el caso de los digrafos, es necesario separar $N(v)$ en $N^+(v)$ y $N^-(v)$ y alcanza con guardar solo un tipo de vecindario; se omiten los detalles.

- Si bien para representar $\mathbf{N}$ se puede usar un árbol o una tabla de hash, si $V = \{0, \ldots, n-1\}$ como en nuestro caso, alcanza con un vector.

- Análogamente, para representar cada conjunto $\mathbf{N}[v]$ se puede usar cualquier estructura conjunto. Comúnmente, se usa una secuencia sobre vector o una tabla de hash. Antiguamente se usaban listas y, por esta razón, a la estructura se la conoce con el nombre de *listas de adyacencias*.

Representación con listas de adyacencias

El diccionario $\{1 \to \{3,2\}, 2 \to \{1,3\}, 3 \to \{1,4,2\}, 4 \to \{3\}\}$ representa al grafo:



- En C++ podemos representar un grafo de la siguiente forma:

Representación de un grafo con listas de adyacencias

```cpp
using graph = std::vector<std::vector<int>>;
//using graph = std::vector<std::unordered_set<int>>;
```

- En este caso, el espacio requerido es $O(n + \sum_{v \in V(G)} d(v)) = O(n + m)$, que también es óptimo, salvo que G tenga pocas aristas.[2]

- Esta es la estructura que se suele usar para representar grafos en las bibliotecas de código.

- Vale notar que cada arista vw está representada dos veces, ya que $v \in N(w)$ y $w \in N(v)$. En algunas aplicaciones se pueden agregar punteros (iteradores) que relacionan estas aristas (para permitir, e.g., un borrado eficiente de la arista). Otras veces, esta estructura se extiende con algún invariante, e.g., $N(v)$ está ordenado de alguna forma para todo $v \in V(G)$.

- Remarcamos que uno no puede suponer que la estructura tiene tal o cual particularidad sin demostrar cómo se implementa. En particular, uno no debería suponer que el input ya viene procesado. Empero, sí podemos aplicar transformaciones entre distintas representaciones cuando estamos dispuestos a pagar los costos. Los siguientes teoremas sirven para transformar de listas de aristas (input usual) a listas de adyacencias.

Observación 2. *Dada una representación de un (di)grafo G por lista de aristas, se puede obtener una representación de G por listas de adyacencias en $O(n + m)$. Recíprocamente, se puede obtener la lista de aristas de G en $O(n + m)$ dada la representación por listas de adyacencias.*

Demostración. El algoritmo tiene dos pasos. Primero inicializamos un vector N con n lista vacías en tiempo $O(n)$. Luego, recorremos todas las aristas de $E(G)$ y, por cada aristas vw, insertamos v en $N(w)$ y $w \in N(v)$. Como el recorrido de una lista y la inserción de un elemento en un vector cuestan $O(1)$ amortizado, el costo total es $O(n + m)$. Para la reciproca, recorremos $N(v)$ para cada $v \in V(G)$ y, por cada $w \in N(v)$ tal que $v < w$, agregamos vw a la lista resultante. El código C++ queda como ejercicio. □

2.4.1. Representación de grafos III: matriz de adyacencia

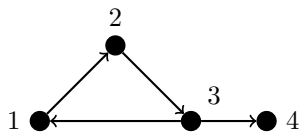
- En lugar de usar una tabla de hash, otra forma de representar un conjunto C sobre un universo $\{0, \dots, n-1\}$ es usando un arreglo de n posiciones, donde $v \in C$ si y sólo si la posición v-ésima es true.

- Esta representación de conjuntos lo podemos aplicar a cada vecindario $\mathbf{N}[v]$ de una lista de adyacencias.

- La representación resultante consume $\Theta(n^2)$ espacio. Por este motivo, no es apropiada para definir el input. Por otra parte, cualquier algoritmo que use esta representación requiere $\Omega(n^2)$ tiempo. En peor caso, cuando $m = O(n)$, el algoritmo es cuadrático. Es por ello que esta representación debería evitarse a no ser que: 1. se trabaje sobre grafos con $m = \Omega(n^2)$, en cuyo caso es lineal, o 2. no se conozcan algoritmos $o(n^2)$.

- A estas representación se la llama *matriz de adyacencia* y se la suele escribir como una matriz.

Un digrafo y su matriz de adyacencias



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

[2] Esto no suele pasar, y en tal caso se pueden compactar los vértices aislados.

- Observar que la matriz de adyacencia tiene 0's en la diagonal y es simétrica cuando G es un grafo.

- Las matrices de adyacencias tienen varias propiedades matemáticas con una semántica particular que se puede explotar en varios algoritmos.

- Asimismo, permiten aplicar álgebra lineal a teoría de grafos. Así que, si bien no son una estructura apropiada para describir sucintamente un grafo, sí son una herramienta computacional y matemática válida.

- En C++, la representación de un grafo es la siguiente.

Representación de un grafo con matriz de adyacencia

```
using graph = std::vector<std::vector<int>>;
graph G(n, vector<int>(n, 0));    //inicializacion
```

Observación 3. *Dada una representación de un grafo G por lista de aristas, se puede obtener una representación de G por matriz de adyacencias en $\Theta(n^2)$ tiempo. Recíprocamente, se puede obtener la lista de aristas de G en $\Theta(n^2)$ tiempo, dada la representación por listas de adyacencias.*

Demostración. Ejercicio: escribir el algoritmo en C++. □

2.5. Igualdad e isomorfismo

- Dos (di)grafos G y H son iguales cuando $V(G) = V(H)$ y $E(G) = E(H)$.

- A veces tenemos dos (di)grafos que no son iguales, pero que admiten los mismos dibujos. Para reflejar esta situación, se usa el concepto de isomorfismo, que en rigor es independiente del dibujo.

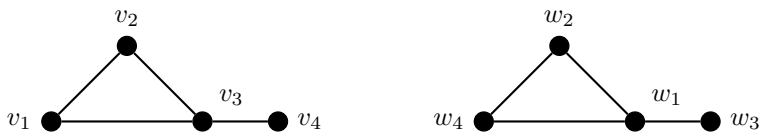
- Un (di)grafo H es *isomorfo* a G cuando existe una biyección $f\colon V(G) \to V(H)$ tal que $vw \in E(G)$ si y sólo si $f(v)f(w) \in E(H)$.

- A la función f se la llama *isomorfismo* de G a H.

- Denotamos con $=$ la a relación tal que $G = H$ cuando H es isomorfo a G..

Isomorfismo de grafos

Las funciones $\{v_1 \to w_4, v_2 \to w_2, v_3 \to w_1, v_4 \to w_3\}$ y $\{v_1 \to w_2, v_2 \to w_4, v_3 \to w_1, v_4 \to w_3\}$ son isomorfismos el grafo de la izquierda al de la derecha.



Observación 4. *La relación de isomorfismo $=$ es de equivalencia.*

Demostración. Si f es la identidad, entonces $vw \in E(G)$ si y sólo si $f(v)f(w) = vw \in E(G)$. Ergo, G es isomorfo a G, i.e., $=$ es reflexiva. Por otra parte, si f es un isomorfimo de G a H, entonces $f^{-1}\colon V(H) \to V(G)$ es una función bien definida porque f es biyectiva. Más aún, $f(v)f(w) \in E(H)$ si y sólo si $f^{-1}(f(v))f^{-1}(f(w)) = vw \in E(G)$. Ergo, G es isomorfo a H y, en consecuencia, $=$ es simétrica. Finalmente, si g es un isomorfismo de H a J, entonces $vw \in E(G)$ si y solo si $f(v)f(w) \in E(H)$ que ocurre si y sólo si $g(f(v))g(f(w)) \in E(J)$. Es decir que $g \circ f$ es un isomorfismo de G a J y, por lo tanto, $=$ es transitiva. □

- En función de la Observación 4, simplemente decimos que G y H son *isomorfos* sin especificar quién es isomorfo a quien.

- Notar que nunca se usa $=$ para igualdad, porque es una operación poco común en (di)grafos. Además, es reemplazable por su definición, i.e., $V(G) = V(H)$ y $E(G) = E(H)$.

- Problema de isomorfismo de (di)grafos:

  Input: (di)grafos G y H.

  Output: verdadero si y sólo si $G = H$. En caso de ser $G = H$, se espera un isomorfismo de G a H.

- Este problema se puede resolver con backtracking y admite múltiples podas.

- No se conocen algoritmos polinomiales para resolver el problema de isomorfismo.

- En Noviembre de 2015, Babai [2] afirmó que el problema se puede resolver en tiempo quasi-polinomial (i.e., $O(2^{(\log n)^c})$ con $c > 1$ constante). En Enero de 2017, mantuvo su afirmación luego de corregir un error (ver `http://people.cs.uchicago.edu/~laci/`, accedido 05/03/2019). El resultado todavía no está publicado formalmente (i.e., con referato), pero la versión pre-print fue citada 290 veces al día de la fecha según Google.

3. Intervalo (10 mins)

4. Terminología básica de grafos II (50 mins)

4.1. Subgrafos

- Un (di)grafo H es un *sub(di)grafo* de G cuando $V(H) \subseteq V(G)$ y $E(H) \subseteq E(G)$. Más aún:

  ▷ Si $V(H) = V(G)$, entonces H es un sub(di)grafo *generador* de G. El *(di)grafo generado por* $E \subseteq E(G)$ es (V, E).

  ▷ Si $E(H) = \{vw \in E(G) \mid v, w \in V(H)\}$, entonces H es un sub(di)grafo *inducido* de G. El *sub(di)grafo de G inducido por V* es el sub(di)grafo inducido H de G con $V(H) = V$. Se lo suele denotar como $G[V]$.

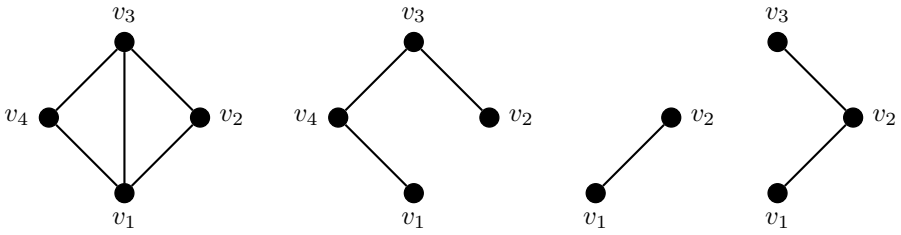
- Dado un conjunto de aristas E, denotamos $G \setminus E$ al sub(di)grafo generado por $E(G) \setminus E$.

- Análogamente, si $V \subseteq V(G)$, entonces $G \setminus V$ es el sub(di)grafo inducido por $V(G) \setminus V$.

- A veces notamos $G - e$ a $G \setminus \{e\}$ para $e \in E(G)$ y $G - v$ a $G \setminus \{v\}$ para $v \in V(G)$.

Tipos de subgrafos

De izquierda a derecha: un grafo G, su subgrafo generador $G \setminus \{v_1v_2, v_1v_3\}$, su subgrafo inducido $G \setminus \{v_3, v_4\}$, y su subgrafo $(G - v_4) - v_1v_3$ que no es ni inducido ni generador.



4.2. Completos y conjuntos independientes de grafos

- Un vértice v es *universal* cuando es adyacente a todos los otros vértices de G, i.e., $N[v] = V(G)$. Equivalentemente, v es universal cuando $d(v) = n - 1$.

- En cambio, v es *aislado* cuando no es adyacente a ningún vértice, i.e., $N(v) = \emptyset$. Equivalentemente, v es aislado cuando $d(v) = 0$.

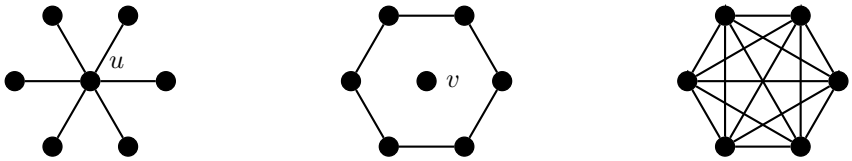
- Un grafo G es *completo* cuando todos sus vértices son universales.

- Se suele denotar K_n a un grafo completo genérico con n vértices. Ergo, G es completo cuando $G = K_n$.

- Hay una versión análoga al completo para digrafos (llamado torneo), pero no la estudiamos en este curso.

Universales y aislados

El vértice u es universal, v es aislado, y el grafo con 6 universales es K_6.



Corolario 2. *Si $G = K_n$, entonces $m = \binom{n}{2}$.*

Demostración. Porque $2m = \sum_{v \in V(K_n)} d(v) = \sum_{v \in V(K_n)} (n-1) = n(n-1)$ □

- Dada una propiedad P y un conjunto U, decimos que $S \subseteq U$ es *maximal* (*minimal*) cuando $P(S)$ es verdadero y ningún conjunto conteniendo (contenido) propiamente a (en) S satisface P.

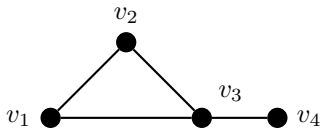
- En cambio, S es *maximo* (mínimo) cuando $P(S)$ es verdadero y $|S'| \leq |S|$ ($|S'| \geq |S|$) para todo conjunto $S' \subseteq U$ satisfaciendo P.

- Una *clique* Q es un conjunto de vértices mutuamente adyacentes.

- Un *conjunto independiente* S es un conjunto de vértices mutuamente no adyacentes.

Cliques y conjuntos independientes

El siguiente grafo tiene dos cliques maximales ($\{v_3, v_4\}$ y $\{v_1, v_2, v_3\}$) y tres conjuntos independientes maximales ($\{v_3\}$, $\{v_1, v_4\}$, $\{v_2, v_4\}$).

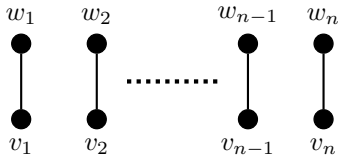


- Un grafo es completo cuando tiene una única clique maximal, que es máxima y tiene n vértices.

- En general, un grafo puede tener una cantidad exponencial de cliques (o conjuntos independientes) máximas.

Un grafo con 2^n conjuntos independientes maximales

Todo conjunto independiente maximal es máximo y contiene exactamente un vértice entre v_i y w_i para $1 \leq i \leq n$. Ergo, el grafo tiene 2^n conjuntos independientes.



- Problema de clique máxima (resp. conjunto independiente máximo):

  Input: grafo G.

  Output: una clique máxima (resp. conjunto independiente máximo) de G.

- El problema de la fiesta de la clase pasada es el problema de conjunto independiente máximo. No se conocen algoritmos polinomiales para este problema.

- El problema de interval scheduling es un caso particular del problema de conjunto independiente máximo, donde el input corresponde al grafo G cuyos vértices son intervalos y donde IJ son adyacentes cuando $I \cap J \neq \emptyset$. Este problema se puede resolver en tiempo lineal.

4.3. Complemento de un grafo

- El complemento de un grafo $G = (V, E)$ es otro grafo $\overline{G} = (V, \overline{E})$ donde $vw \in \overline{E}$ si y sólo si $vw \notin E$.

Complemento de un grafo

Un grafo a la izquierda y su complemento a la derecha.

- Notar que: 1. $H = \overline{G}$ sii $\overline{H} = G$; 2. $\overline{\overline{G}} = G$; 3. $E(G) \cap E(\overline{G}) = \emptyset$; 4. $E(G) \cup E(\overline{G}) = E(K_n)$; 5. v es universal en G sii v es aislado en $\overline{G}$; y 6. Q es una clique en G sii Q es un conjunto independiente de G.

Observación 5. *Para todo grafo G ocurre que $|E(\overline{G})| = \binom{n}{2} - |E(G)|$.*

Demostración. Porque $E(G) \cap E(\overline{G}) = \emptyset$, con lo cual $\binom{n}{2} = |E(K_n)| = |E(G) \cup E(\overline{G})| = |E(G)| + |E(\overline{G})|$. □

- Dado un grafo G, se puede computar $\overline{G}$ en $O(n^2)$ en peor caso. El algoritmo queda de ejercicio y se sugiere describirlo aprovechando la matriz de adyacencia. El algoritmo resultante es óptimo porque $\overline{G}$ tiene $\Omega(n^2)$ aristas en peor caso (Observación 5).

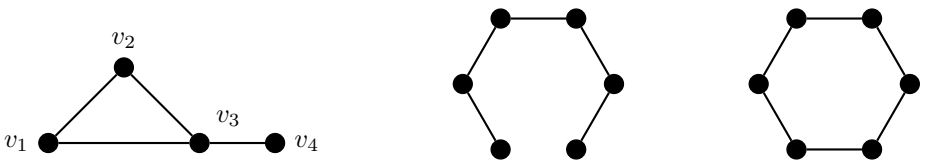
- En consecuencia, cualquier algoritmo de complejidad $O(f)$ para el problema de clique máxima implica la existencia de un algoritmo $O(f + n^2)$ para el problema de conjunto independiente y, recíprocamente, todo algoritmo para el problema de conjunto independiente maximo implica la existencia de un algoritmo $O(f + n^2)$ para el problema de clique máxima.

4.4. Caminos y circuitos

- En un (di)grafo G, un *camino* de *longitud* k es una secuencia $P = v_0, \ldots, v_k$ de vértices tales que $v_i v_{i+1} \in E(G)$ para $0 \leq i < k$. Se dice que P es un *camino de* v_0 *a* v_k.
- Si $v_0 = v_k$, entonces P es un *circuito*.
- Si $v_i \neq v_j$ para todo $i \neq j$, entonces P es un *camino simple*.
- Si $k \geq 3$, $v_0 = v_k$ y $v_0, \ldots, v_{k-1}$ es un camino simple, entonces P es un *circuito simple* o *ciclo*.
- Se suele denotar P_n (C_n) a un grafo genérico formado por un único camino (ciclo) de n vértices.
- Si la longitud de P es mínima entre todos los caminos de v_0 a v_k, entonces P es un *camino más corto* (de v_0 a v_k). Observar que cualquier camino más corto es simple (ejercicio).
- La *distancia* $d(v, w)$ es la longitud de un camino más corto de v a w; en caso de no existir ningún camino de v a w, entonces $d(v, w) = \infty$.

Caminos y circuitos

Al la izquierda un grafo que tiene: un camino no simple $v_1, v_3, v_2, v_1, v_3, v_4$, un camino simple v_1, v_2, v_3, v_4, un camino más corto v_1, v_3, v_4, un circuito no ciclo $v_1, v_2, v_3, v_1, v_2, v_3, v_1$ y un ciclo v_1, v_2, v_3, v_1. Al centro el grafo P_6 y a la derecha el grafo C_6.



Observación 6. *Si G es un grafo, la función d de distancia permite definir una* métrica *ya que:*

1. $d(v, w) >= 0$ *y* $d(v, w) = 0$ *si y sólo si* $v = w$,
2. $d(v, w) = d(w, v)$, *y*
3. $d(v, w) \leq d(v, u) + d(u, w)$.

Si G es un digrafo, sólo satisface las condiciones 1 y 3.

Demostración.

1. Por definición, cualquier camino de v a $w \neq v$ tiene al menos una arista, mientras que $P = v$ es un camino de v a v de longitud 0.
2. Notar que si $P = v_0, \ldots, v_k$ es un camino de $v_0 = v$ a $v_k = w$ en un grafo, entonces $Q = v_k, \ldots, v_0$ es un camino de w a v, ergo $d(v, w) = d(w, v)$.
3. Finalmente, si $P = v_0, \ldots, v_k$ es un camino de $v = v_0$ y $u = v_k$ con longitud $k = d(v, u)$ y $Q = v_k, \ldots, v_{k+j}$ es un camino de v_k y $z = v_{k+j}$ con longitud $j = d(u, w)$, entonces $P + Q$ es un camino de v a w con longitud $k + j = d(v, u) + d(u, w)$. Por lo tanto, $d(v, w) \leq d(v, u) + d(u, w)$.

□

5. Comentarios bibliográficos

Para aquellos que deseen conocer la historia del surgimiento y la expansión de la teoría de grafos en sus inicios, se recomienda [4].

Como siempre, ningún libro tiene los temas en exactamente el mismo orden, terminología, enfoque y profundidad que se presentan en la materia. Sin embargo, todos los temas se encuentran en los libros. En general hay cuatro grandes enfoques para presentar los temas. Uno más algorítmico que apunta a la implementación computacional de los algoritmos, dejando un poco de lado algunos aspectos teóricos que involucran propiedades de grafos (e.g. [6, 15, 16]). Otro más teórico, que se enfoca en las propiedades matemáticas de los grafos que, si bien tratan sobre temas algorítmicos e incluyen la demostración de los mismos, soslayan o ignoran su implementación computacional (e.g. [3, 12, 22]). Un tercero, que combina ambas perspectivas profundizando más en una o en otra de acuerdo al discurso que se quiere dar (e.g. [9, 10, 19]). Finalmente, un enfoque que apunta más a la resolución de problemas de competencia, que están interesados en la implementación simple y rápida de (algunos) algoritmos (e.g. [11, 14]). En la materia estamos interesados en los enfoques que combinan los conceptos teóricos con la implementación de los algoritmos. En particular, nos interesa dar un discurso en donde la teoría surja de la resolución algorítmica de problemas.

Lamentablemente, en esta versión del presente documento, tengo pendiente incluir punteros específicos al material bibliográfico. Para esta clase la idea es tener un manejo básico de la terminología de grafos y cómo los mismos se pueden representar en una computadora.

Referencias

[1] John Arquilla and Hal Fredricksen. "Graphing" an optimal grand strategy. *Military Operations Research*, 1(3):3–17, 1995.

[2] László Babai. Graph isomorphism in quasipolynomial time. *CoRR*, abs/1512.03547, 2015.

[3] Claude Berge. *Graphs and hypergraphs*. North-Holland Publishing Co., Amsterdam-London; American Elsevier Publishing Co., Inc., New York, 1973. Translated from the French by Edward Minieka, North-Holland Mathematical Library, Vol. 6.

[4] Norman L. Biggs, E. Keith Lloyd, and Robin J. Wilson. *Graph theory. 1736–1936*. The Clarendon Press, Oxford University Press, New York, second edition, 1986.

[5] Ernie J. Cockayne, Paul A. Dreyer, Jr., Sandra M. Hedetniemi, and Stephen T. Hedetniemi. Roman domination in graphs. *Discrete Math.*, 278(1-3):11–22, 2004.

[6] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to algorithms*. MIT Press, Cambridge, MA, third edition, 2009.

[7] Leonhard Euler. Solutio problematis ad geometriam situs pertinentis. *Comment. Acad. Sci. U. Petrop*, 8:128–140, 1736. En Latín. Traducción en Ingles en Biggs, N. L., Lloyd, E. K., y Wilson, R. J., *Graph theory. 1736–1936*, **Oxford University Press**, 1986.

[8] Leonhard Euler. Solution d'une question curieuse que ne paroit soumise a aucune analyse. *Mem. Acad. Sci. Berlin*, 15:310–337, 1766.

[9] Jean-Claude Fournier. *Graph theory and applications with exercises and problems*. ISTE, London; John Wiley & Sons, Inc., Hoboken, NJ, 2009.

[10] Alan Gibbons. *Algorithmic graph theory*. Cambridge University Press, Cambridge, 1985.

[11] Steven Halim and Felix Halim. *Competitive Programming*. Self published, 3rd edition, 2013.

[12] Frank Harary. *Graph theory*. Addison-Wesley Publishing Co., Reading, Mass.-Menlo Park, Calif.-London, 1969.

[13] Michael A. Henning and Stephen T. Hedetniemi. Defending the Roman Empire—a new strategy. *Discrete Math.*, 266(1-3):239–251, 2003. The 18th British Combinatorial Conference (Brighton, 2001).

[14] Antti Laaksonen. *Guide to Competitive Programming: Learning and Improving Algorithms Through Contests*. Springer, 2017.

[15] R. Sedgewick and K. Wayne. *Algorithms*. Pearson Education, 2011.

[16] Steven S. Skiena. *The Algorithm Design Manual*. Springer, second edition, 2008.

[17] Ian Stewart. Defend the Roman Empire! *Sci. Am.*, 281:136–138, December 1999.

[18] James Sylvester. On an application of the new atomic theory to the graphical representation of the invariants and covariants of binary quantics, with three appendice. *Amer. J. Math*, 1(1):64–104, 1878.

[19] Jayme Luiz Szwarcfiter. *Teoria Computacional de Grafos*. Elsevier, 2018. Com programas Python por F. S. Oliveira e P. E. D. Pinto.

[20] Nenad Trinajstić. *Chemical graph theory*. Mathematical Chemistry Series. CRC Press, Boca Raton, FL, second edition, 1992.

[21] John J. Watkins. *Across the board: the mathematics of chessboard problems*. Princeton University Press, Princeton, NJ, 2004.

[22] Douglas B. West. *Introduction to graph theory*. Prentice Hall, Inc., Upper Saddle River, NJ, 1996.