Algoritmos y Estructuras de Datos III
Apunte Teórico

2018
Primer cuatrimestre

Índice

Capítulo 1

Grafos

1.1. Conceptos básicos

Definición 1.1.1. Un *grafo* $G = (V, E)$ es un par de conjuntos donde E es un conjunto de pares no ordenados de elementos *distintos* de V. Los elementos de V se llaman *nodos* o *vértices* y los de E *ejes* o *aristas*. Dados $u, v \in V$, si $e = (u, v) \in E$ se dice que u y v son *adyacentes* y que e es *incidente* a u y v.

Notación. Dado un grafo G notamos $V(G)$ a su conjunto de vértices y $E(G)$ a su conjunto de aristas. Cuando no hay ambigüedad escribimos simplemente V y E, respectivamente, y llamamos $n = |V|$ y $m = |E|$.

Definición 1.1.2. Un *multigrafo* es un grafo en el que puede haber varias aristas entre un mismo par de nodos distintos. Es decir, tiene un *multiconjunto* de aristas en lugar de un conjunto.

Definición 1.1.3. Un *seudografo* es un multigrafo que puede tener aristas que inciden sobre el mismo nodo, llamadas *loops*. Es decir, aristas de la forma (v, v), donde v es un nodo del seudografo.

Definición 1.1.4. El *grado* de un nodo v es la cantidad de aristas incidentes a v. Se nota $d(v)$ o $d_G(v)$ para aclarar que es el grado en el grafo G.

Proposición 1.1.1. *Sea G un grafo y $v \in V$. Entonces $0 \leq d(v) < n$.*

Proposición 1.1.2. *Sea G un grafo. Entonces*

$$\sum_{v \in V} d(v) = 2m$$

Demostración. Inducción en la cantidad de aristas.

Caso base. $m = 0$. Como G no tiene aristas, $d(v) = 0$ para todo $v \in V$. Luego $\sum_{v \in V} d(v) = 0 = 2m$.

Paso inductivo. Supongamos que la proposición vale para $m-1 \geq 0$ y veamos que vale para m. Sea (u, w) una arista de G y $G' = G - (u, w)$ de m aristas. Como el grado de todo vértice de G es igual tanto en ambos grafos salvo por u y w, cuyos grados en G' son uno menor que en G, $\sum_{v \in V} d_G(v) = 2 + \sum_{v \in V} d_{G'}(v)$. Luego, por hipótesis inductiva, $\sum_{v \in V} d_G(v) = 2 + 2(m-1) = 2m$. □

Definición 1.1.5. Un grafo es *completo* si todos sus nodos son adyacentes entre sí.

Notación. K_n es el grafo completo de n vértices.

Proposición 1.1.3. *La cantidad de aristas K_n es*

$$\binom{n}{2} = \frac{n(n-1)}{2}$$

Demostración. En K_n hay una arista entre todo par de nodos distintos de V. Luego, la cantidad de aristas de K_n es exactamente la cantidad de pares de nodos distintos de V, que es $\binom{n}{2} = \frac{n!}{2!\,(n-2)!} = \frac{n(n-1)}{2}$. □

Corolario 1.1.4. *Todo grafo de n vértices tiene a lo sumo $n(n-1)/2$ aristas.*

Definición 1.1.6. El *grafo complemento* de G, notado $\bar{G}$, es aquel que tiene el mismo conjunto de nodos pero en el cual dos nodos son adyacentes si y sólo si no son adyacentes en G. Es decir, $\bar{G} = (V, K \setminus E)$ donde $K = E(K_n)$.

Proposición 1.1.5. *Sea G un grafo. Entonces*

1. *Para todo nodo v, $d_{\bar{G}}(v) = n - 1 - d_G(v)$.*
2. *$\bar{G}$ tiene $n(n-1)/2 - m$ aristas.*

1.1.1. Caminos y circuitos

Definición 1.1.7. Un *camino* es una secuencia de aristas $e_1 e_2 \cdots e_k$ tal que un extremo de e_i coincide con uno de e_{i-1} y el otro con uno de e_{i+1} para $i = 2, 3, \ldots, k-1$. Un *camino simple* es un camino que no repite nodos.

Proposición 1.1.6. *Si P es un camino entre dos nodos distintos u y v, entonces existe un camino simple entre u y v en P.*

Demostración. Si $P = e_1 e_2 \cdots e_k$ es un camino simple, no hay nada que probar. Si no, debe haber algún nodo v por el cual P pasa más de una vez. Es decir, deben existir i, j tal que $i+2 \leq j$, $e_i = (u, v)$ y $e_j = (v, w)$. Si $e_1 e_2 \cdots e_i e_j e_{j+1} \cdots e_k$ es un camino simple, termina la demotración. En caso contrario, podemos hacer el mismo razonamiento hasta obtener un camino simple. □

Definición 1.1.8. Un *circuito* es un camino que empieza y termina en el mismo nodo. Un *circuito simple* es un circuito de tres o más aristas que no repite nodos.

Definición 1.1.9. La *longitud* de un camino es la cantidad de aristas que tiene. La *distancia* entre dos nodos u y v en un grafo, notada $d(u, v)$, es la longitud mínima de los caminos entre u y v si existe alguno. Si $u = v$ se dice que $d(u, v) = 0$ y si no existen caminos entre u y v se dice que $d(u, v) = \infty$.

Proposición 1.1.7. *Si un camino P entre u y v tiene longitud $d(u, v)$ entonces P es un camino simple.*

Demostración. Si P no fuera un camino simple, existiría un camino simple P' en P entre u y v de menor longitud. Pero entonces $d(u, v)$ no es la longitud mínima de los caminos entre u y v. □

Proposición 1.1.8. *La función de distancia cumple las siguientes propiedades para todo nodo u, v, w del grafo.*

1. $d(u,v) \geq 0$ *y* $d(u,v) = 0 \iff u = v$.
2. $d(u,v) = d(v,u)$
3. $d(u,v) \leq d(u,w) + d(w,v)$

Demostración.

1. Por definición de la función de distancia.
2. Porque un camino de u a v es un camino de v a u.
3. Porque $d(u,w) + d(w,v)$ es la longitud de un camino entre u a v (uno que pasa por w) y $d(u,v)$ es la longitud mínima de los caminos entre u y v.

□

Definición 1.1.10. Un grafo es *conexo* si existe un camino entre todo par de nodos.

1.1.2. Subgrafos

Definición 1.1.11. Un *subgrafo* de un grafo $G = (V, E)$ es un grafo $G' = (V', E')$ tal que $V' \subseteq V$ y $E' \subseteq E \cap (V' \times V')$.

Definición 1.1.12. Un *subgrafo inducido* de un grafo $G = (V, E)$ es un subgrafo $G' = (V', E')$ tal que $E' = E \cap (V' \times V')$.

Definición 1.1.13. Un *subgrafo generador* de un grafo G es un subgrafo de G que tiene el mismo conjunto de vértices.

Definición 1.1.14. Una *componente conexa* de un grafo G es un subgrafo conexo maximal de G.

1.1.3. Grafos bipartitos

Definición 1.1.15. Un grafo $G = (V, E)$ es *bipartito* si existe una partición $P = \{V_1, V_2\}$ de V (i.e. $V_1 \cup V_2 = V$, $V_1 \cap V_2 = \emptyset$ y $V_1, V_2 \neq \emptyset$) tal que toda arista de G tiene un extremo en V_1 y otro en V_2. P es una *bipartición* de G.

Definición 1.1.16. Un grafo bipartito con bipartición $\{V_1, V_2\}$ es *completo* si todo nodo de V_1 es adyacente a todo nodo de V_2.

Notación. Un grafo bipartito completo con un conjunto de la bipartición de tamaño p y el otro de tamaño q se nota $K_{p,q}$.

Teorema 1.1.9. *Un grafo G con dos o más nodos es bipartito si y sólo si no tiene circuitos simples de longitud impar.*

Demostración.

$\Longrightarrow$. Supongamos que existe un circuito simple C de longitud impar en G. Es decir, $C = v_1 v_2 \cdots v_k v_1$ donde $v_i \in V$ y k es impar. Toda bipartición de G debe tener a los nodos de índice par de C en un conjunto y los de índice impar en el otro. Pero entonces v_1 y v_k están en el mismo conjunto de la bipartición y son adyacentes. Esto es absurdo.

$\Longleftarrow$. Para toda componente conexa C_i de G tomamos un vértice cualquiera w_i y definimos V_1 y V_2 como el conjunto de los vértices que están a distancia par e impar, respectivamente, de algún w_i. Afirmamos que $\{V_1, V_2\}$ es una bipartición de G. Si no lo fuera, existirían dos vértices adyacentes u y v, ambos a distancia par o impar de algún w_i. Sea P un camino entre w_i y u de longitud $d(w_i, u)$ y Q un camino entre w_i y v de longitud $d(w_i, v)$. Llamemos z al primer nodo en común entre P y Q yendo de u y v hacia w_i. La longitud de $P_{w_i z}$ (el camino de w_i a z en P) debe ser igual a la longitud de $Q_{w_i z}$ (el camino de w_i a z en Q). De lo contrario, ir desde w_i hasta z por el más corto de los dos y después ir desde z hasta u o v sería un camino más corto que P o Q. Luego, u y v están ambos a distancia par o impar de z. Pero entonces $P_{zu} + (u, v) + Q_{vz}$ es un circuito simple de longitud impar, lo cual es absurdo. □

1.1.4. Isomorfismo

Definición 1.1.17. Dos grafos $G = (V, E)$ y $G' = (V', E')$ son *isomorfos* si existe una función biyectiva $f : V \to V'$ tal que para todo $u, v \in V$, $(u, v) \in E \iff (f(u), f(v)) \in E'$.

Proposición 1.1.10. *Si dos grafos son isomorfos entonces tienen la misma cantidad de nodos, aristas, componentes conexas, nodos de grado $d \geq 0$, caminos y circuitos simples de longitud $l \geq 1$.*

1.1.5. Grafos orientados

Definición 1.1.18. Un *grafo orientado*, *grafo dirigido* o *digrafo* $G = (V, E)$ es un par de conjuntos donde E es un conjunto de pares *ordenados* de elementos distintos de V. Los elementos de E se llaman *arcos*.

Definición 1.1.19. Sea v un nodo de un grafo orientado. El *grado de entrada* de v es la cantidad de arcos que entran a v y el *grado de salida* la cantidad de arcos que salen. Es decir, la cantidad de arcos que tienen a v como segundo elemento, en el primer caso, o como primer elemento, en el último.

Definición 1.1.20. Un *camino orientado* en un grafo orientado es una secuencia de arcos $e_1 e_2 \cdots e_k$ tal que el primer elemento de e_i coincide con el segundo elemento de e_{i-1} y el segundo elemento de e_i con el primero de e_{i+1} para $i = 2, 3, \ldots, k-1$.

Definición 1.1.21. Un *circuito orientado* en un grafo orientado es un camino orientado que empieza y termina en el mismo nodo.

Definición 1.1.22. Un grafo orientado es *fuertemente conexo* si para todo par de nodos u y v existe un camino orientado de u a v y otro de v a u.

Proposición 1.1.11. *Sea G un grafo orientado. Entonces*

$$\sum_{v \in V} d_{in}(v) = \sum_{v \in V} d_{out}(v).$$

Demostración. Inducción en la cantidad de arcos.

Caso base. $m = 0$. Como no hay arcos, $\sum_{v \in V} d_{in}(v) = 0 = \sum_{v \in V} d_{out}(v)$.

Paso inductivo. Supongamos que la proposición vale para $m - 1 \geq 0$ y veamos que vale para m. Sea e un arco de G y sea $G' = G - e$. La cantidad de arcos que salen de los nodos en G' es uno menos que en G, y la cantidad de arcos que entran a los nodos en G' también. Luego por hipótesis inductiva $\sum_{v \in V} d_{in}^{G}(v) = 1 + \sum_{v \in V} d_{in}^{G'}(v) = 1 + \sum_{v \in V} d_{out}^{G'}(v) = \sum_{v \in V} d_{in}^{G}(v)$. □

1.1.6. Representación de grafos

Definición 1.1.23. La *matriz de adyacencia* de un grafo G es una matriz $M \in \mathbb{R}^{n \times n}$ definida como:

$$m_{ij} = \begin{cases} 1 & \text{si los nodos } i \text{ y } j \text{ son adyacentes} \\ 0 & \text{si no} \end{cases}$$

Definición 1.1.24. La *matriz de incidencia* de un grafo G es una matriz $M \in \mathbb{R}^{m \times n}$ definida como:

$$m_{ij} = \begin{cases} 1 & \text{si la arista } i \text{ es incidente al nodo } j \\ 0 & \text{si no} \end{cases}$$

Definición 1.1.25. Las *listas de adyacencia* de un grafo G son listas $L_1, L_2, \ldots, L_n$ donde L_i es una lista de los nodos adyacentes al nodo i.

Teorema 1.1.12. *Si M es la matriz de adyacencia del grafo G, el elemento m_{ij}^k de M^k es la cantidad de caminos de longitud k entre los nodos i y j.*

Demostración. Inducción en k.

Caso base. $k = 1$. Los caminos de longitud 1 son las aristas. Luego m_{ij} es exactamente la cantidad de caminos de longitud 1 entre i y j.

Paso inductivo. Supongamos que el teorema vale para $k \geq 1$ y veamos que vale para $k + 1$. Todo camino de longitud $k + 1$ de i a j está formado por un camino de longitud k de i a algún nodo p y un camino de longitud 1 de p a j. Por hipótesis inductiva, hay $m_{ip}^k \cdot m_{pj}$ de estos caminos para cada nodo p. Luego, como $M^{k+1} = M^k \cdot M$, $m_{ij}^{k+1} = \sum_{p=1}^{n} m_{ip}^k \cdot m_{pj}$ es exactamente la cantidad de caminos de longitud $k + 1$ de i a j. □

1.2. Árboles

Definición 1.2.1. Un *árbol* es un grafo conexo sin circuitos simples.

Lema 1.2.1. *Sea G un grafo conexo y $e \in E$. $G - e$ es conexo si y sólo si e pertenece a un circuito simple de G.*

Demostración.

$\Longrightarrow$. Sea $e = (u, v)$. Como $G - e$ es conexo existe en él un camino simple P entre u y v que no contiene a e. Luego, $P + e$ es un circuito simple en G que contiene a e.

$\Longleftarrow$. Sea C un circuito simple de G que contiene a e. Veamos que existe un camino en $G - e$ entre todo para de nodos u y v. Como G es conexo, existe en él un camino P entre u y v. Si P no pasa por e, entonces es un camino en $G - e$. Si pasa por e, podemos reemplazar e por $C - e$ y obtener un camino entre u y v en $G - e$. Por lo tanto, $G - e$ es conexo. □

Lema 1.2.2. *La unión entre dos caminos simples distintos entre un par de vértices contiene un circuito simple.*

Demostración. Sean P y Q los dos caminos simples distintos y u, v el par de vértices. Llamemos w al último vértice en común entre P y Q yendo de u a v antes de que se separen los caminos. Este debe existir ya que si no los caminos serían iguales. Sea $z \neq w$ el primer vértice en común entre P y Q que se encuentra yendo desde w hacia v (existe porque si no los caminos no llegarían a v). Entonces $P_{wz} + Q_{zw}$ es un circuito simple. □

Teorema 1.2.3. *Sea G un grafo. Son equivalentes:*

1. *G es un árbol.*
2. *G es un grafo sin circuitos simples, pero si se agrega una arista e a G resulta un grafo con exactamente un circuito simple y este contiene a e.*
3. *Existe exactamente un camino simple entre todo par de nodos.*
4. *G es conexo, pero si se le quita cualquier arista a G queda un grafo no conexo.*

Demostración.

1 $\Longrightarrow$ *2.* Como G es conexo (pues es un árbol), $G + e$ también lo es. Luego por, el lema 1.2.1, e pertenece a un circuito simple de G. Afirmo que $G + e$ tiene exactamente un circuito simple. Para eso supongamos lo contrario: $G + e$ tiene dos circuitos simples distintos C y C'. e debe pertenecer a ambos porque G no tiene circuitos simples. Luego, si $e = (u, v)$, $C - e$ y $C' - e$ son caminos simples entre u y v en G. Pero entonces, por el lema 1.2.2, su unión contiene un circuito simple y este está en G. Esto es absurdo.

2 $\Longrightarrow$ *3.* Sean u y v nodos de G. Si son adyacentes, $e = (u, v)$ es un camino entre ellos. Si no, $G + e$ tiene un circuito simple C que contiene a e y, luego, $C - e$ es un camino simple entre u y v en G. Entonces existe un camino simple entre todo par de nodos de G. Más aún, no puede haber más de un camino simple entre dos nodos. De lo contrario, por el lema 1.2.2, la unión de dos de esos caminos contendría un circuito simple, pero G no tiene circuitos simples.

3 $\Longrightarrow$ *4.* Como existe un camino entre todo par de nodos, G es conexo. Si al sacar una arista $e = (u, v)$ de G quedara un grafo conexo, por el lema 1.2.1, e pertenecería a un circuito simple C de G. Pero entonces existiría más de un camino simple entre u y v (e y $C - e$).

4 $\Longrightarrow$ *1.* Si G tuviera un circuito simple, sacar una arista de ese circuito a G resultaría en un grafo que sigue siendo conexo. Esto es absurdo. □

Definición 1.2.2. Una *hoja* en un árbol es un nodo de grado uno.

Lema 1.2.4. *Todo árbol no trivial tiene al menos dos hojas.*

Demostración. Sea T el árbol y P un camino simple de longitud máxima de T. P existe porque T tiene al menos una arista y no tiene circuitos simples. Afirmamos que los nodos extremos de P son hojas. Si no lo fueran, tendrían algún otro nodo adyacente que no está en P y entonces podríamos expandir P con este nodo. Pero entonces P no sería de longitud máxima. □

Lema 1.2.5. *Sea G un árbol. Entonces $m = n - 1$.*

Demostración. Inducción en la cantidad de nodos.

Caso base. $n = 1$. G es el grafo trivial. Luego $m = 0 = n - 1$.

Paso inductivo. Supongamos que el lema vale para $n - 1 \geq 1$ y veamos que vale para n. Sea v una hoja de G (existe por el lema 1.2.4). Como $G - v$ es un árbol de $n-1$ nodos, la hipótesis inductiva indica que tiene $n-2$ aristas. Luego, dado que $G - v$ tiene una arista menos que G (la que es incidente a la hoja v), $n - 2 = m - 1$, lo cual es equivalente a decir que $m = n - 1$. □

Definición 1.2.3. Un *bosque* es un grafo sin circuitos simples.

Proposición 1.2.6. *Las componentes conexas de un bosque son árboles.*

Corolario 1.2.7. *Un bosque con c componentes conexas tiene $m = n-c$ aristas.*

Corolario 1.2.8. *Un grafo con c componentes conexas tiene $m \geq n-c$ aristas.*

Teorema 1.2.9. *Sea G un grafo. Son equivalentes:*

1. *G es un árbol.*
2. *G es un grafo sin circuitos simples y $m = n - 1$.*
3. *G es conexo y $m = n - 1$.*

Demostración.

1 $\implies$ 2. Por el lema 1.2.5.

2 $\implies$ 3. Como G no tiene circuitos simples, es un bosque. Luego, por el corolario 1.2.7, $m = n - c$ donde c es la cantidad de componentes conexas de G. Entonces, como $m = n - 1$, c debe ser igual a 1 y luego G es conexo.

3 $\implies$ 1. Supongamos que G tiene circuitos simples. Podemos sacar aristas de G hasta obtener un árbol T de n nodos y $m' < m$ aristas. Por el lema 1.2.5, T tiene $n - 1$ aristas, pero entonces $n - 1 = m' < m = n - 1$ es absurdo. □

Definición 1.2.4. Un *árbol orientado* es un grafo orientado G tal que:

1. G no tiene circuitos orientados.
2. El grafo no orientado subyacente de G es un árbol.
3. Existe un nodo r tal que todo nodo de G es alcanzable desde él por un camino orientado.

1.2.1. Árboles enraizados

Definición 1.2.5. Un *árbol enraizado* T es un árbol con un nodo distinguido llamado *raíz* desde el cual son alcanzables todos los demás nodos. El *nivel* de un nodo de T es su distancia a la raíz y la *altura* de T es la distancia de la raíz a los nodos más lejanos, es decir, el máximo nivel de sus nodos. Los nodos de T que no son hojas ni raíz se llaman *nodos internos*.

Observación. En un árbol no orientado cualquier nodo puede ser la raíz.

Definición 1.2.6. Un árbol enraizado es *m-ario* si todos sus nodos internos tienen grado a lo sumo $m+1$ y la raíz a lo sumo m. Es *exactamente m-ario* si los nodos internos tienen grado $m+1$ y la raíz m.

Definición 1.2.7. Un árbol enraizado de altura h es *balanceado* si todas sus hojas están a nivel h o $h-1$. Es *balanceado completo* si están todas a nivel h.

Proposición 1.2.10. *Un árbol m-ario de altura h tiene a lo sumo m^h hojas.*

Demostración. Inducción en la altura del árbol.

Caso base. $h = 0$. El árbol es el grafo trivial. Luego tiene $0 \leq m^0 = 1$ hojas.

Paso inductivo. Supongamos que la proposición vale para h y veamos que vale para $h+1$. Sea T' el árbol que resulta de sacar todas las hojas del nivel $h+1$ de T. Como la altura de T' es h, por hipótesis inductiva tiene a lo sumo m^h hojas. Dado que podemos obtener T a partir de T' agregándole las hojas que habíamos sacado y que por cada hoja de T' se pueden agregar a lo sumo m hojas, T tendrá a lo sumo $m^h \cdot m = m^{h+1}$ hojas. □

Corolario 1.2.11. *Sea un árbol m-ario de altura h y l hojas. Entonces $\lceil \log_m l \rceil \leq h$.*

Proposición 1.2.12. *Sea T un árbol exactamente m-ario balanceado de altura h y l hojas. Entonces $\lceil \log_m l \rceil = h$*

1.2.2. Representación de árboles

Definición 1.2.8. Sea $T = (V, E)$ un árbol orientado y r una raíz de T. Un *mapa de predecesores* de T es un diccionario $\pi : V \setminus \{r\} \to V$ tal que para todo $v \in V \setminus \{r\}$, $\pi(v) = u \iff (u, v) \in E$. Llamamos a T el árbol *inducido* por π con raíz r.

1.2.3. Recorrido de grafos

Los algoritmos *breadth-first search* (BFS) y *depth-first search* (DFS) sirven para recorrer grafos conexos (orientados o no). El recorrido que hacen define un subárbol del grafo. Ambos siguen el equema presentado en el algoritmo 1.2.1. La diferencia es que BFS implementa el conjunto S como una cola mientras que DFS usa una pila.

Algoritmo 1.2.1: Esquema general de BFS/DFS

Entrada: Un grafo conexo G y una raíz r de G
Resultado: Recorre todos los nodos de G

```
1 marcar r como visitado
2 S ← {r}
3 mientras |S| ≠ ∅ hacer
4 |   sacar un nodo u de S
5 |   para cada v ∈ V adyacente a u no visitado hacer
6 |   |   marcar v como visitado
7 |   |   S ← S ∪ {v}
8 |   fin
9 fin
```

1.2.4. Árbol generador mínimo

Definición 1.2.9. Un *árbol generador* de un grafo G es un subgrafo generador de G que es árbol.

Definición 1.2.10. Dado un grafo $G = (V, E)$ y una función de peso $w : E \to \mathbb{R}$, un *árbol generador mínimo* (AGM) de G es un árbol generador de G de peso mínimo.

Definición 1.2.11. Dada una función de peso w de las aristas de un grafo G, el *peso* de G es $w(G) = \sum_{e \in E} w(e)$.

Lema 1.2.13. *Sea G un grafo y T un árbol generador de G. Si e es una arista de G que no está en T y f una arista de T que está en el circuito simple de $T + e$, entonces $T + e - f$ es un árbol generador de G.*

Demostración. $T' = T + e - f$ es un subgrafo generador de G porque tiene el mismo conjunto de nodos. Además, como $|E(T+e-f)| = |E(T)| = n-1$, T' es un árbol. Luego, sólo falta probar que T' es conexo. Sean u, v dos nodos de T y P un camino entre ellos en T. Si P no pasa por f entonces es un camino en T'. Si pasa por f la podemos reemplazar por $C - f$, donde C es el circuito simple de $T + e$ que contiene a f, y obtener un camino en T'. Luego T' es conexo. □

Algoritmo goloso

Definición 1.2.12. Un *corte* $C = (S, T)$ de un grafo G es una partición de V. Decimos que una arista *cruza* el corte C si tiene un extremo en S y otro en T. Un corte *respeta* a un conjunto de aristas A si ninguna arista de A cruza el corte.

Teorema 1.2.14. *Sea G un grafo conexo y $w : E \to \mathbb{R}$ una función de peso de las aristas de G. Si $A \subset E$ está en algún AGM de G, $C = (S, T)$ es un corte de G que respeta a A y $e \in E$ es una arista que cruza C de peso mínimo, entonces $A \cup \{e\}$ está en algún AGM de G.*

Demostración. Sea T un AGM de G que contiene a A. Supongamos que $e = (u, v)$ no está en T porque de lo contrario no habría nada que probar. Como T

es conexo, debe existir en él un camino P de u a v que cruza C, pues uno de estos nodos está en S y el otro está en T. Específicamente, una arista f de P es la que cruza C. Como f pertenece al circuito simple $P + e$ que se forma en $T + e$, el lema 1.2.13 dice que $T' = T + e - f$ es un árbol generador de G. Más aún, como e es una arista que cruza C de peso mínimo, $w(e) \leq w(f)$. Entonces $w(T') \leq w(T)$, lo cual implica que T' es un AGM de G. Como T y T' sólo difieren en aristas que cruzan C (e y f) y este corte respeta a A, T' también contiene a A. Por lo tanto $A \cup \{e\}$ está en T' que es un AGM de G. □

Algoritmo 1.2.2: Algoritmo de AGM goloso

Entrada: Un grafo conexo G y una función de peso w
Salida: Un árbol generador mínimo de G

```
1 E* ← ∅
2 mientras E* no determina un AGM de G hacer
3 |   C ← corte de G que respeta a E*
4 |   e ← arista de G de peso mínimo que cruza C
5 |   E* ← E* ∪ {e}
6 fin
7 devolver (V, E*)
```

Corolario 1.2.15. *El algoritmo 1.2.2 obtiene un AGM de G.*

El *algoritmo de Prim* (1.2.3) y el *algoritmo de Kruskal* (1.2.5) son casos particulares del algoritmo 1.2.2.

Algoritmo 1.2.3: Algoritmo de Prim

Entrada: Un grafo conexo G, una función de peso w y una raíz r
Salida: Un árbol generador mínimo de G

```
1 V* ← {r}
2 E* ← ∅
3 mientras E* no determina un AGM de G hacer
4 |   elegir (u, v) ∈ E de peso mínimo tal que u ∈ V* y v ∉ V*
5 |   E* ← E* ∪ {(u, v)}
6 |   V* ← V* ∪ {v}
7 fin
8 devolver (V*, E*)
```

Proposición 1.2.16 (Complejidad del algoritmo de Prim)**.** *El algoritmo de Prim con cola de prioridad (1.2.4) implementada con min-heap para grafos representados con listas de adyacencias tiene complejidad $O(m \log n)$.*

Demostración. El costo de inicializar los valores de π y ρ es de $O(n)$. Crear el conjunto vacío E^* puede hacerse en $O(1)$, mientras que el min-heap en función de ρ se puede construir en $O(n)$. El costo de cada una de las $O(n)$ llamadas a Extraer-Minimo es de $O(\log n)$. La operación Actualizar-Cola cuesta también $O(\log n)$ y se realiza a lo sumo $O(m)$ veces. Por lo tanto, la complejidad

Algoritmo 1.2.4: Algoritmo de Prim con cola de prioridad

Entrada: Un grafo conexo G, una función de peso w y una raíz r
Salida: Un árbol generador mínimo de G

```
1  ρ(r) ← 0
2  para cada v ∈ V \ {r} hacer ρ(v) ← ∞
3  E* ← ∅
4  Q ← cola de prioridad mínima de V en función de ρ
5  mientras Q ≠ ∅ hacer
6  |  u ← Extraer-Minimo(Q)
7  |  para cada v ∈ Q tal que (u, v) ∈ E hacer
8  |  |  π(v) ← u
9  |  |  ρ(v) ← mín{ρ(v), w(u, v)}
10 |  |  Actualizar-Cola(Q, v)
11 |  fin
12 fin
13 devolver π
```

del algoritmo es $O((n+m)\log n)$, que es igual a $O(m \log n)$ porque $O(n) = O(m)$ por ser G conexo. □

Algoritmo 1.2.5: Algoritmo de Kruskal

Entrada: Un grafo conexo G y una función de peso w
Salida: Un árbol generador mínimo de G

```
1  E* ← ∅
2  mientras E* no determina un AGM de G hacer
3  |  elegir e ∈ E \ E* de peso mínimo que no forme un circuito con
   |    aristas de E*
4  |  E* ← E* ∪ {e}
5  fin
6  devolver (V, E*)
```

Observación. Se puede terminar el ciclo del algoritmo 1.2.6 una vez que se hayan agregado $n-1$ aristas a E^* en lugar de recorrerlas todas.

Observación. Si en el algoritmo 1.2.6 el grafo G es genérico, el resultado es un *bosque generador mínimo* de G.

Proposición 1.2.17 (Complejidad del algoritmo de Kruskal)**.** *El algoritmo de Kruskal con disjoint-set (1.2.6) implementado con union-by-rank y path-compression para grafos representados con listas de adyacencias tiene complejidad* $O(m \log n)$.

Demostración. El costo de crear el conjunto de aristas E^* es $O(1)$ y ordenar E en función de w puede hacerse en $O(m \log m)$. Las $O(n)$ llamadas a Make-Set y Union junto con las $O(m)$ llamadas a Find-Set tienen un costo de $O((n+m)\,\alpha(n)) = O((n+m)\log n)$, donde α es la inversa de la función de Ackerman. Más aún, como G es conexo, esto es $O(m \log n)$. Luego, dado que $O(\log m) = O(\log n^2) = O(\log n)$, la complejidad del algoritmo es de $O(m \log n)$. □

Algoritmo 1.2.6: Algoritmo de Kruskal con disjoint-set

Entrada: Un grafo conexo G y una función de peso w
Salida: Un árbol generador mínimo de G

```
1  E* ← ∅
2  ordenar E en función de w
3  para cada v ∈ V hacer MAKE-SET(v)
4  para cada (u, v) ∈ E tomado en orden hacer
5  |  si FIND-SET(u) ≠ FIND-SET(v) entonces
6  |  |  E* ← E* ∪ {(u, v)}
7  |  |  UNION (u, v)
8  |  fin
9  fin
10 devolver (V, E*)
```

1.3. Camino mínimo

Definición 1.3.1. Sea G un grafo (orientado o no) y $w : E \to \mathbb{R}$ una función de peso de las aristas de G. Un *camino mínimo* P de un nodo u a otro nodo v de G es un camino de u a v de peso mínimo.

Notación. Si $v = u$ o v no es alcanzable desde u decimos que el peso de un camino mínimo de u a v es 0 e ∞ respectivamente y que tiene cero aristas.

Definición 1.3.2. Dado un grafo G y una función de peso $w : E \to \mathbb{R}$, el *problema de camino mínimo* consiste en determinar:

- *Único origen.* Un camino mínimo de un nodo v dado a cada nodo de G.
- *Único destino.* Un camino mínimo de cada nodo de G a un nodo v dado. Puede ser reducido al problema de único origen invirtiendo el sentido de las aristas.
- *Único par.* Un camino mínimo entre un par de nodos u y v dados.
- *Todo par.* Un camino mínimo entre todo par de nodos de G

El problema no está bien definido cuando G tiene circuitos de peso negativo alcanzables por un vértice de origen.

Notación. Cuando nos referimos al problema de camino mínimo asumimos que el grafo en cuestión no tiene circuitos de peso negativo alcanzables desde un vértice de origen.

Proposición 1.3.1. *Un camino mínimo no puede tener circuitos de peso positivo.*

Demostración. Supongamos que un camino mínimo P entre dos nodos u y v tiene un circuito de peso positivo C. Luego $P \setminus C$ es un camino entre u y v y $w(P \setminus C) = w(P) - w(C) < w(P)$. Pero entonces P no es un camino mínimo. □

Corolario 1.3.2. *Si un camino mínimo tiene circuitos, estos deben tener peso cero.*

Proposición 1.3.3. *Sea G un grafo y u, v nodos de G. Si existe un camino mínimo de u a v, entonces existe un camino mínimo simple de u a v.*

Demostración. Sea P un camino mínimo de u a v. Supongamos que P no es simple, ya que si lo fuera no habría nada que probar. Entonces debe haber algún circuito C en P, determinado por la primera y la última aparición de un nodo por el que P pasa más de una vez. Por el corolario 1.3.2, el peso de C debe ser cero. Luego $P' = P \setminus C$ es un camino de u a v que es mínimo, pues tiene el mismo peso que P. Podemos repetir este razonamiento hasta obtener un camino mínimo simple de u a v. □

Notación. Asumimos que los caminos mínimos son simples a menos que se aclare lo contrario.

Proposición 1.3.4 (Subestructura óptima)**.** *Si $P = v_1 v_2 \cdots v_k$ es un camino mínimo entre v_1 y v_k, entonces $P_{v_i v_j}$ es un camino mínimo entre v_i y v_j para todo i, j tal que $1 \leq i < j \leq k$.*

Demostración. Supongamos que $P_{v_i v_j}$ no es un camino mínimo entre v_i y v_j. Entonces debe existir otro camino Q de menor peso entre esos nodos. Pero si reemplazamos $P_{v_i v_j}$ por Q en P obtenemos un camino entre v_i y v_j de peso menor que P. Esto es absurdo ya que P es un camino mínimo. □

Proposición 1.3.5. *Un camino simple en un grafo G tiene a lo sumo $n - 1$ aristas.*

Demostración. Un camino simple en G tiene a lo sumo n nodos y, como un camino simple es un árbol, tiene a lo sumo $n - 1$ aristas. □

Definición 1.3.3. Sea G un grafo con función de peso w y u, v nodos de G. La *distancia mínima* de u a v es el peso de un camíno mínimo de u a v. Es decir, se define como

$$\delta(u, v) = \begin{cases} 0 & \text{si } u = v \\ w(P) & \text{si existe un camino mínimo } P \text{ de } u \text{ a } v \\ \infty & \text{si } v \text{ no es alcanzable desde } u \end{cases}$$

Proposición 1.3.6 (Desigualdad triangular)**.** *Sea G un grafo con función de peso w sin circuitos negativos, $s \in V$ y $(u, v) \in E$. Entonces*

$$\delta(s, v) \leq \delta(s, u) + w(u, v)$$

Demostración. Si existe un camino mínimo de s a v o $s = v$, ningún otro camino de s a v puede tener menor peso, por lo cual la desigualdad vale. Si no hay caminos de s y v entonces no puede existir un camino de s a u porque existiría uno de s a v. Luego $\delta(s, u) = \infty$ y la desigualdad también vale. □

1.3.1. Único origen

Definición 1.3.4. Sea G un grafo, $s \in V$ un nodo de origen y $v \in V$. Una *estimación de camino mínimo* a v, notada $\epsilon(v)$, es una cota superior del peso de un camino mínimo de s a v.

Definición 1.3.5. Sea G un grafo, $s \in V$ un nodo de origen. El proceso de *relajar* una arista $(u, v) \in E$ (algoritmo 1.3.1) consiste en considerar un camino que pasa por (u, v) y, si resulta ser posible, ajustar la estimación $\epsilon(v)$ y actualizar el mapa de predecesores que describe los caminos considerados para las estimaciones. Si una relajación logra ajustar la estimación se dice que es *exitosa*. Decimos que se relaja un nodo v si se relaja alguna arista que entra a él.

Algoritmo 1.3.1: Relajación de una arista

Entrada: Los nodos u, v de una arista a relajar
Resultado: Relaja la arista (u, v)

```
1 procedimiento RELAJAR(u, v):
2     si ε(v) > ε(u) + w(u, v) entonces
3         ε(v) ← ε(u) + w(u, v)
4         π(v) ← u
5     fin
6 fin
```

Proposición 1.3.7. *Sea G un grafo con función de peso w y $s \in V$ un nodo de origen. Si $(u, v) \in E$ y $\epsilon(u)$ y $\epsilon(v)$ son estimaciones de camino mínimo, entonces luego de relajar (u, v):*

1. *$\epsilon(v)$ es a lo sumo su valor anterior.*
2. *$\epsilon(v)$ sigue siendo una estimación de camino mínimo.*
3. *Si la relajación fue exitosa, $\pi(v) = u$.*

Demostración.

1. Por definición de relajación.
2. Si el valor de $\epsilon(v)$ no cambió, no hay nada que probar. Si cambió, entonces $\epsilon(v) = \epsilon(u) + w(u, v)$ por definición de relajación. Luego, como $\epsilon(u)$ es una estimación de camino mínimo, esto es mayor o igual a $\delta(s, u) + w(u, v)$, que por la desigualdad triangular es al menos tan grande como $\delta(s, v)$. Por lo tanto, $\epsilon(v)$ es una estimación de camino mínimo.
3. Por definición de relajación.

□

Corolario 1.3.8. *Sea G un grafo y $s \in V$ un nodo de origen. Si $\epsilon(v) = \delta(s, v)$ y $(u, v) \in E$ tal que $\epsilon(u)$ es una estimación de camino mínimo, entonces la relajación de (u, v) no es exitosa.*

Proposición 1.3.9. *Sea G un grafo y $s \in V$ un nodo de origen. El algoritmo 1.3.2 define ϵ como una estimación de camino mínimo de los nodos de G.*

Proposición 1.3.10. *Sea G un grafo con función de peso w, $s \in V$ un nodo de origen y ϵ inicializado con el algoritmo 1.3.2. Entonces, para todo $v \in V$, $\epsilon(v) = \delta(s, v)$ si y sólo si se hizo la secuencia de relajaciones exitosas P (con posibles relajaciones intermedias), donde P es algún camino mínimo de s a v.*

Algoritmo 1.3.2: Inicialización de estimaciones de camino mínimo

Entrada: Un grafo G y un nodo de origen s
Resultado: Inicializa las estimaciones de camino mínimo de s a todos los nodos de G

```
1 procedimiento INICILIZAR-ESTIMACIONES(G, s):
2 |  ϵ(s) ← 0
3 |  para cada v ∈ V hacer
4 |  |  ϵ(v) ← ∞
5 |  fin
6 fin
```

Demostración. Si v es igual a s o no es alcanzable desde s, entonces la proposición vale trivialmente por la definición del algoritmo 1.3.2 y la proposición 1.3.8. Supongamos entonces que v no es s y que es alcanzable desde s.

$\Longrightarrow$. Sea (u, v) la última relajación exitosa de v. Entonces, $\delta(s, v) = \epsilon(v) = \epsilon(u)+w(u, v)$. Como ϵ fue inicializado con el algoritmo 1.3.2 y sólo fue modificado mediante relajaciones, $\epsilon(u) \geq \delta(s, u)$. Luego, $\delta(s, v) \geq \delta(s, u)+w(u, v) \geq \delta(s, v)$. Esto implica que (u, v) es la última arista de un camino mínimo de s a v. Además, como $\delta(s, u) + w(u, v) = \delta(s, v) = \epsilon(u) + w(u, v)$ implica que $\epsilon(u) = \delta(s, u)$, podemos repetir este razonamiento para u. De esta forma, obtenemos una por una y en orden inverso las aristas de un camino mínimo de s a v.

$\Longleftarrow$. Sea $P = e_1e_2 \cdots e_k$ y demostremos por inducción en k. El caso base se da cuando $k = 1$. En este caso $P = (s, v)$ es un camino mínimo de s a v por hipótesis y, luego, la proposición vale trivialmente. Para el paso inductivo supongamos que vale para algún $k \geq 1$ y probemos que es cierto para $k + 1$. Sea $e_{k+1} = (u, v)$. Por hipótesis inductiva, como P_{su} es un camino mínimo de longitud k, $\epsilon(u) = \delta(s, u)$. Luego, cuando se relajó e_{k+1} se definió $\epsilon(v) = \epsilon(u) + w(u, v) = \delta(s, u) + w(u, v)$. Esto no es otra cosa que el peso de P, que es un camino mínimo de s a v. Por lo tanto, $\epsilon(v) = \delta(s, v)$. □

Corolario 1.3.11. *Sea G un grafo, $s \in V$ un nodo de origen, $v \in V$ y ϵ inicializado con el algoritmo 1.3.2. Si $\epsilon(v) = \delta(s, v)$, entonces π define un camino mínimo de s a v.*

Demostración. Si v es igual a s o no es alcanzable desde s el corolario vale trivialmente. Supongamos entonces que v es distinto de s y es alcanzable desde s. Por la proposición 1.3.10 sabemos que se hizo la secuencia de relajaciones exitosas $P = e_1e_2 \cdots e_k$, para algún $k \geq 1$. Llamemos $P_i = e_1e_2 \cdots e_i$ y $e_i = (x_i, y_i)$, para $1 \leq i \leq k$. Es fácil ver por el corolario 1.3.8 que al hacer la secuencia de relajaciones exitosas P_i, $\pi(y_j) = x_j$ para todo $j \leq i$. De esto se deduce directamente que $e_k = (\pi(v), v), e_{k-1} = (\pi^2(v), \pi(v)), \ldots, e_1 = (\pi^k(v), \pi^{k-1}(v)) = (s, \pi^{k-1}(v))$, donde $\pi^i(v)$ es el resultado de aplicar π a v de forma recursiva i veces. Por lo tanto, π define el camino mínimo P que va de s a v. □

Algoritmo de Dijkstra

El algoritmo de Dijkstra resuelve el problema de camino mínimo de único origen para grafos (orientados o no) con pesos no negativos en las aristas.

Algoritmo 1.3.3: Algoritmo de Dijkstra

Entrada: Un grafo G con pesos no negativos y un nodo de origen s
Salida: Caminos mínimos de s a cada nodo de G y sus pesos

```
1  INICILIZAR-ESTIMACIONES(G, s)
2  S ← ∅
3  Q ← V
4  mientras Q ≠ ∅ hacer
5  |  u ← argmín_{v∈Q} ϵ(v)
6  |  Q ← Q \ {u}
7  |  S ← S ∪ {v}
8  |  para cada v ∈ Q tal que (u, v) ∈ E hacer
9  |  |  RELAJAR(u, v)
10 |  fin
11 fin
12 devolver π, ϵ
```

Lema 1.3.12 (Invariante del algoritmo de Dijkstra)**.** *Sea G un grafo (orientado o no) con pesos no negativos y $s \in V$ un nodo de origen. Al comenzar la k-ésima iteración, el algoritmo de Dijkstra determina un camino mínimo de s a cada nodo de S.*

Demostración. Sean S_k, Q_k, ϵ_k y π_k los valores de S, Q, ϵ y π respectivamente al comenzar la k-ésima iteración. Demostramos por inducción en k.

Caso base. $k = 1$. Como $S_1 = \emptyset$, el lema vale trivialmente.

Paso inductivo. Supongamos que el lema vale para algún $k \geq 1$ y veamos que vale para $k + 1$. Como no se relajan los nodos de S_k, $\epsilon_{k+1}(v) = \epsilon_k(v) = \delta(s, v)\ \forall\ v \in S_k$. Luego basta ver que $\epsilon_{k+1}(u) = \delta(s, u)$ porque u es el único nodo que se agrega a S_k. Sabemos que $u \neq s$ porque s es agregado a S en la primera iteración y estamos en la iteración $k + 1 > 1$. Por otro lado, si u no es alcanzable por s entonces $\delta(s, u) = \infty = \epsilon(u)$ y no hay nada que demostrar. Luego, supongamos que existe algún camino de s a u y sea P uno de ellos. Como $s \in S_k$ y $u \in Q_k$, debe haber alguna arista (x, y) de P que cruza (S_k, Q_k). Es decir, $x \in S_k$ e $y \in Q_k$. Entonces

$$
\begin{aligned}
w(P) &= w(P_{sx}) + w(x, y) + w(P_{yu}) \\
&\geq w(P_{sx}) + w(x, y) && \text{(pesos no negativos)} \\
&\geq \delta(s, x) + w(x, y) && \text{(definición de } \delta\text{)} \\
&= \epsilon(x) + w(x, y) && \text{(hipótesis inductiva)} \\
&= \epsilon(y) && \text{(relajación de } (x, y) \text{ al agregar } x \text{ a } S\text{)} \\
&\geq \epsilon(u) && (u = \text{argmín}_{v \in Q_k}\, \epsilon(v))
\end{aligned}
$$

Llegamos a que $w(P) \geq \epsilon_k(u)$ para cualquier camino P de s a u. Luego, $\epsilon_{k+1}(u) = \epsilon_k(u) = \delta(s, u)$ y entonces $\epsilon_{k+1}(v) = \delta(s, v)$ para todo $v \in S_{k+1}$. Por lo tanto, por el corolario 1.3.11, π_{k+1} determina un camino mínimo de s a cada nodo $v \in S_{k+1}$ y su peso es $\epsilon_{k+1}(v)$. □

Teorema 1.3.13 (Correctitud del algoritmo de Dijkstra)**.** *Sea G un grafo*

(orientado o no) con pesos no negativos y $s \in V$ un nodo de origen. El algoritmo de Dijkstra determina un camino mínimo de s a cada nodo de V.

Demostración. Como Q comienza siendo igual a V y en cada iteración se saca un nodo de Q y se lo agrega a S, cuando $Q = \emptyset$ al finalizar el ciclo S es igual a V. Luego, por el teorema del invariante, el lema 1.3.12 vale para $S = V$. □

Proposición 1.3.14 (Complejidad del algoritmo de Dijkstra)**.** *El algoritmo de Dijkstra para grafos representados por listas de adyacencia tiene complejidad $O(n^2 + m)$ si la búsqueda del mínimo en Q se hace de forma lineal y $O((n + m) \log n)$ si se usa una cola de prioridad implementada con min-heap.*

Demostración. En el algoritmo se hacen n búsquedas de mínimo en Q y, dado que por cada nodo se relajan a lo sumo todas las aristas incidentes a él, como mucho se hacen $2m$ relajaciones. Luego, para la búsqueda lineal la complejidad es de $O(n \cdot n + 2m) = O(n^2 + m)$. Para el min-heap, primero se lo debe crear, lo cual se puede hacer en $O(n)$, y para cada relajación se debe actualizar el min-heap. Entonces, el costo en este caso es $O(n \log n)$ para extraer los mínimos de Q y $O(m \log n)$ para relajar las aristas. Por lo tanto, la complejidad es $O((n + m) \log n)$. □

Observación.

1. Para grafos densos ($m = \Theta(n^2)$), la implementación con búsqueda lineal es asintóticamente mejor que la de min-heap ($\Theta(n^2)$ vs. $\Theta(n^2 \log n)$).

2. Si el grafo no orientado subyacente es conexo, la complejidad para la implementación con min-heap es $O(m \log n)$, pues $n = O(m)$.

3. En la implementación con búsqueda lineal se puede obtener la misma complejidad para matrices de adyacencia debido a que se las puede transformar a listas de adyacencia en $O(n^2)$.

Algoritmo de Bellman-Ford

El algoritmo de Bellman-Ford resuelve el problema de camino mínimo de único origen para grafos (orientados o no) sin circuitos negativos. Si el grafo tiene circuitos negativos, lo puede detectar.

Lema 1.3.15 (Invariante del algoritmo de Bellman-Ford)**.** *Sea G un grafo (orientado o no) sin circuitos negativos y $s \in V$ un nodo de origen. Al comenzar la k-ésima iteración, el algoritmo de Bellman-Ford determina un camino mínimo de a lo sumo $k - 1$ aristas de s a los nodos de G.*

Demostración. Inducción en k.

Caso base. $k = 1$. Los caminos mínimos de longitud cero desde s son los que van a s y a los nodos no alcanzables desde s. Luego el lema vale trivialmente por cómo se incializaron los valores de ϵ.

Paso inductivo. Supongamos que el lema vale para algún $k \geq 1$ y veamos que vale para $k+1$. Sea el comienzo de la k-ésima iteración y sea $v \in V$. Si existe un camino mínimo de a lo sumo $k - 1$ aristas a v, por hipótesis inductiva, el algoritmo ya lo determinó. Luego el corolario 1.3.8 afirma que no será modificado. Supongamos ahora que los caminos mínimos a v tienen exactamente k aristas y

Algoritmo 1.3.4: Algoritmo de Bellman-Ford

Entrada: Un grafo G y un nodo de origen s
Salida: Caminos mínimos de s a cada nodo de G y sus pesos

```
1  INICILIZAR-ESTIMACIONES(G, s)
2  i ← 0
3  repetir
4  |  i ← i + 1
5  |  para cada (u, v) ∈ E hacer
6  |  |  RELAJAR(u, v)
7  |  fin
8  hasta que no hubo cambios en ϵ o i = n
9  si hubo cambios en ϵ entonces
10 |  error hay ciclos negativos
11 fin
12 devolver π, ϵ
```

sea P uno de ellos y (u, v) la última arista de P. Por subestructura óptima de camino mínimo, $P' = P - (u, v)$ es un camino mínimo a u. Luego, por hipótesis inductiva, $\epsilon(u) = \delta(s, u)$. Entonces, después de relajar el eje (u, v) en la iteración actual, el valor de $\epsilon(v)$ será $\epsilon(u) + w(u, v) = \delta(s, u) + w(u, v) = w(P) = \delta(s, v)$. Por lo tanto, por el corolario 1.3.11, al finalizar la iteración el algoritmo habrá determinado un camino mínimo de a lo sumo $k-1$ aristas a cada nodo de G. □

Teorema 1.3.16 (Correctitud del algoritmo de Bellman-Ford)**.** *Sea G un grafo (orientado o no) y $s \in V$ un nodo de origen. Si G tiene circuitos negativos, el algoritmo de Bellman-Ford lo detecta; si no, determina un camino mínimo de s a cada nodo de G.*

Demostración. Sea G sin circuitos negativos. Como los caminos mínimos tienen a lo sumo $n - 1$ aristas, el lema 1.3.15 garantiza que al empezar a lo sumo la n-ésima iteración el algoritmo ya habrá determinado un camino mínimo a cada nodo de G. Luego, en esa iteración no habrán cambios en ϵ y el algoritmo terminará correctamente.

Veamos ahora que si no hubo cambios en alguna iteración $k \leq n$ entonces el algoritmo determinó un camino mínimo para cada nodo de G. Por el corolario 1.3.11, basta ver que $\epsilon(v) = \delta(s, v)$ para todo $v \in V$. Si $v = s$ o v no es alcanzable desde s la igualdad vale por cómo se inicializa ϵ. Supongamos entonces que existe un camino mínimo $P = v_0 v_1 \cdots v_r$ donde $v_0 = s$, $v_r = v$. Que no haya habido cambios en ϵ significa, por la definición de relajación, que para todo $(x, y) \in E$, $\epsilon(y) \leq \epsilon(x) + w(x, y)$. Equivalentemente, $\epsilon(y) - \epsilon(x) \leq w(x, y)$. Luego, si sumamos las aristas de P tenemos que $\sum_{i=1}^{r} (\epsilon(v_i) - \epsilon(v_{i-1})) \leq \sum_{i=1}^{r} w(v_{i-1}, v_i)$. Lo primero es igual a $\epsilon(v_r) - \epsilon(v_0) = \epsilon(v_r)$ y lo segundo es el peso de P, que es igual a $\delta(s, v_r)$. Por lo tanto $\epsilon(v_r) \leq \delta(s, v_r)$ y como $\delta(s, v_r) \leq \epsilon(v_r)$, queda demostrado lo que se quería.

Para ver que cuando G tiene circuitos negativos hay cambios en ϵ en todas las iteraciones del algoritmo podemos usar el resultado anterior. Para eso supongamos que esto no pasa: en alguna de las iteraciones no hay cambios en ϵ. Luego, si tomamos $P = v_0 v_1 \cdots v_r$ como un circuito negativo, tenemos que

$\epsilon(v_r) - \epsilon(v_0) \leq w(P)$. Como P es un circuito, v_0 y v_r son el mismo nodo. Pero entonces $w(P) \geq \epsilon(v_0) - \epsilon(v_0) = 0$, lo cual es absurdo porque P es un circuito negativo. □

Proposición 1.3.17 (Complejidad del algoritmo de Bellman-Ford)**.** *El algoritmo de Bellman-Ford para grafos representados con listas de adyacencia tiene complejidad $O(nm)$.*

Demostración. Se hacen a lo sumo n iteraciones y en cada una se relajan m aristas. □

1.3.2. Todo par

Notación. Dado un grafo $G = (V, E)$, nos referimos a su conjunto de vértices como $\{v_1, v_2, \ldots, v_n\}$ y definimos $V_k = \{v_1, v_2, \ldots, v_k\}$.

Definición 1.3.6. Sea G un grafo con función de peso $w : E \to \mathbb{R}$. La *matriz de pesos* de G es una matriz $W \in \mathbb{R}^{n \times n}$ definida como

$$w_{ij} = \begin{cases} 0 & \text{si } v_i = v_j \\ w(v_i, v_j) & \text{si } (v_i, v_j) \in E \\ \infty & \text{si } v_i \neq v_j \wedge (v_i, v_j) \notin E \end{cases}$$

Algoritmo de Floyd-Warshall

El algoritmo de Floyd-Warshall resuelve el problema de camino mínimo de todo par para grafos (orientados o no) sin circuitos negativos. Si el grafo tiene circuitos negativos, lo puede detectar. Es un algoritmo de programación dinámica

Definición 1.3.7. Sea P un camino. Un nodo de P es *intermedio* si no es el primero ni el último. Es decir, si P es la secuencia de vértices $v_1 v_2 \cdots v_k$, v_i es un vértice intermedio de P si $1 < i < k$.

Definición 1.3.8. Dado un grafo $G = (V, E)$ y dos nodos $u, v \in V$, decimos que P es un camino mínimo de u a v con nodos intermedios en $I \subseteq V$ si dentro de todos los caminos de u a v cuyos nodos intermedios están en I, P es uno de peso mínimo.

Lema 1.3.18. *Sea G un grafo, W su matriz de pesos y $D^{(k)} \in \mathbb{R}^{n \times n}$ tal que $d_{ij}^{(k)}$ es el peso de un camino mínimo de v_i a v_j con nodos intermedios en $\{v_1, v_2, \ldots, v_k\}$. Entonces $D^{(k)}$ puede definirse recursivamente como*

$$d_{ij}^{(k)} = \begin{cases} w_{ij} & \text{si } k = 0 \\ \text{mín}\{d_{ij}^{(k-1)},\ d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} & \text{si } k > 0 \end{cases}$$

Demostración. Inducción en k.

Caso base. $k = 0$. Un camino mínimo sin nodos intermedios, si existe, es una arista. Luego el peso de un camino mínimo de v_i a v_j es exactamente w_{ij}.

Paso inductivo. Supongamos que el lema vale para algún $k - 1 \geq 0$ y veamos que vale para k. Sea P un camino mínimo de v_i a v_j con nodos intermedios en V_k. Separemos en dos casos:

- *v_k es un nodo intermedio de P.* Entonces $P = P_{ik} + P_{kj}$, donde P_{ik} es un camino mínimo de v_i a v_k y P_{kj} uno de v_k a v_j, ambos con nodos intermedios en V_{k-1}. Luego, por hipótesis inductiva, $d_{ij}^{(k-1)} \geq w(P) = d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Por lo tanto, $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)} = w(P)$.
- *v_k no es un nodo intermedio de P.* Entonces P es un camino mínimo con nodos intermedios en V_{k-1}. Luego, por hipótesis inductiva, $d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \geq w(P) = d_{ij}^{(k-1)}$. Por lo tanto, $d_{ij}^{(k)} = d_{ij}^{(k-1)} = w(P)$.

□

Lema 1.3.19. *Sea G un grafo, W su matriz de pesos, $D^{(k)} \in \mathbb{R}^{n\times n}$ tal que $d_{ij}^{(k)}$ es el peso de un camino mínimo de v_i a v_j con nodos intermedios en $\{v_1, v_2, \ldots, v_k\}$ y $\Pi^{(k)} \in V^{n\times n}$ tal que $\pi_{ij}^{(k)}$ es el predecesor de v_j en un tal camino. Entonces $\Pi^{(k)}$ puede definirse recursivamente como*

$$\pi_{ij}^{(0)} = \begin{cases} nil & si\ v_i = v_j \ \vee \ w_{ij} = \infty \\ v_i & si\ v_i \neq v_j \ \wedge \ w_{ij} < \infty \end{cases}$$

$$\pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & si\ d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \\ \pi_{kj}^{(k-1)} & si\ d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)} \end{cases}$$

Demostración. Inducción en k.

Caso base. $k = 0$. Un camino mínimo sin nodos intermedios, si existe, es una arista. Luego el lema vale trivialmente.

Paso inductivo. Supongamos que el lema vale para algún $k-1 \geq 0$ y veamos que vale para k. Separemos en casos:

- $d_{ij}^{(k-1)} \leq d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Entonces $d_{ij}^{(k)} = d_{ij}^{(k-1)}$, por lo cual un camino mínimo de v_i a v_j con nodos intermedios en V_{k-1} también es uno con nodos intermedios en V_k. Luego, $\pi_{ij}^{(k-1)}$ es también el predecesor de v_j en un camino mínimo de v_i a v_j con nodos intermedios en V_k.
- $d_{ij}^{(k-1)} > d_{ik}^{(k-1)} + d_{kj}^{(k-1)}$. Entonces $d_{ij}^{(k)} = d_{ik}^{(k-1)} + d_{kj}^{(k-1)} < d_{ij}^{(k-1)}$, por lo cual un camino mínimo P de v_i a v_j con nodos intermedios en V_k tiene que pasar por v_k. Luego el predecesor de v_j en un tal camino es el mismo que en un camino mínimo de v_k a v_j con nodos intermedios en V_{k-1}.

□

Teorema 1.3.20 (Correctitud del algoritmo de Floyd-Warshall)**.** *Sea G un grafo (orientado o no). Si G tiene circuitos negativos el algoritmo de Floyd-Warshall lo detecta; si no, determina un camino mínimo entre todo par de nodos de G y sus pesos.*

Demostración. Sea G sin circuitos negativos. Si para cada valor de k el algoritmo cambiara los elementos de las matrices D y Π al final de la iteración en lugar de hacerlo inmediatamente cada vez que considera un par (i, j), sería trivial ver de los lemas 1.3.18 y 1.3.19 que este computa las matrices $D^{(n)}$ y $\Pi^{(n)}$, por lo cual sería correcto. Veamos que aplicar los cambios de forma inmediata tiene

Algoritmo 1.3.5: Algoritmo de Floyd-Warshall

Entrada: Un grafo G

Salida: Caminos mínimos entre todo par de nodos de G y sus pesos

```
1  D ← W
2  Π ← Π^(0)
3  para k = 1 hasta n hacer
4  |  para i = 1 hasta n hacer
5  |  |  para j = 1 hasta n hacer
6  |  |  |  si d_ij > d_ik + d_kj entonces
7  |  |  |  |  d_ij ← d_ik + d_kj
8  |  |  |  |  π_ij ← π_kj
9  |  |  |  fin
10 |  |  fin
11 |  |  si d_ii < 0 entonces
12 |  |  |  error hay circuitos negativos
13 |  |  fin
14 |  fin
15 fin
16 devolver D, Π
```

el mismo resultado. Cuando se compara d_{ij} se está tomando su valor correcto ya que siempre se cambia un elemento (i, j) distinto. Luego, basta ver que los valores de d_{ik}, d_{kj} y π_{kj} no cambian a lo largo de la iteración de un k dado. Esto es cierto porque un camino mínimo con nodos intermedios en V_k que tiene a v_k como origen o destino es también un camino mínimo con nodos intermedios en V_{k-1}. Como prueba formal podemos ver que si $i = k$ o $j = k$, entonces $d_{ij} > d_{ik} + d_{kj}$ es equivalente a $d_{kj} > d_{kj}$ o $d_{ik} > d_{ik}$ respectivamente, que es falso en ambos casos.

Sea ahora G con circuitos negativos y probemos por el absurdo que el algoritmo lo detecta. Para eso supongamos que $d_{ii} \geq 0$ para todo k y todo i. Sea k' el primer valor de k para el cual V_k contiene a todos los nodos intermedios de algún circuito negativo. Nótese que $v_{k'}$ tiene que pertenecer a estos circuitos negativos por cómo definimos k'. Como al final de la iteración de $k = k' - 1$ las distancias corresponden a caminos mínimos de un subgrafo de G sin circuitos negativos y por lo probado en el párrafo anterior, en ese punto del algoritmo $d_{ij} = d_{ij}^{(k'-1)}$. Luego, al terminar la iteración de $k = k'$, $0 \leq d_{ii} = \min\{d_{ii}^{(k'-1)}, d_{ik'}^{(k'-1)} + d_{k'i}^{(k'-1)}\} \leq d_{ik'}^{(k'-1)} + d_{k'i}^{(k'-1)}$ para todo i. En particular, esto vale para un i tal que v_i está en algún circuito negativo simple C con nodos intermedios en $V_{k'}$. Pero si $C_{ik'}$ y $C_{k'i}$ son los caminos simples de C que van de v_i a $v_{k'}$ y de $v_{k'}$ a v_i respectivamente, entonces $w(C) = w(C_{ik'}) + w(C_{k'i}) \geq d_{ik'}^{(k')} + d_{k'i}^{(k')} \geq 0$. Esto es absurdo porque C es un circuito negativo. □

Proposición 1.3.21 (Complejidad del algoritmo de Floyd-Warshall)**.** *El algoritmo de Floyd-Warshall tiene complejidad $O(n^3)$.*

Observación. Si sólo se desean obtener caminos mínimos cuyos nodos interme-

dios están en un determinado subconjunto de vértices de tamaño k, el algoritmo de Floyd-Warshall puede hacerlo en $O(k^3)$. Además, si posteriormente se desea agrandar el subconjunto (con otros nodos del grafo), sólo es necesario hacer una iteración más del algoritmo por cada nodo agregado a este para obtener los caminos mínimos que faltan.

Algoritmo de Dantzig

El algoritmo de Dantzig resuelve el problema de camino mínimo de todo par para grafos (orientados o no) sin circuitos negativos. Si el grafo tiene circuitos negativos, lo puede detectar. Es un algoritmo de programación dinámica.

Lema 1.3.22. *Sea G un grafo, W su matriz de pesos y $D^{(k)} \in \mathbb{R}^{k\times k}$ tal que $d_{ij}^{(k)}$ es el peso de un camino mínimo de v_i a v_j en el subgrafo de G inducido por $\{v_1, v_2, \ldots, v_k\}$. Entonces $D^{(k)}$ puede definirse recursivamente como*

$$d_{11}^{(1)} = 0$$

$$d_{ij}^{(k)} = \begin{cases} 0 & si\ i = j = k \\ \text{mín}_{1\le t<k}\left(d_{it}^{(k-1)} + w(v_t, v_k)\right) & si\ i \neq k\ \wedge\ j = k \\ \text{mín}_{1\le t<k}\left(w(v_k, v_t) + d_{tj}^{(k-1)}\right) & si\ i = k\ \wedge\ j \neq k \\ \text{mín}\left\{d_{ij}^{(k-1)},\ d_{ik}^{(k)} + d_{kj}^{(k)}\right\} & si\ i, j \neq k \end{cases}$$

Demostración. Inducción en k.

Caso base. $k = 1$. El subgrafo inducido es el nodo aislado v_1. Luego $d_{11}^{(1)} = w(1,1) = 0$.

Paso inductivo. Supongamos que el lema vale para algún $k-1 \geq 1$ y veamos que vale para k. Analicemos cada caso:

- $i = j = k$. El peso de un camino mínimo de v_k a sí mismo es 0.
- $i \neq k \wedge j = k$. Un camino mínimo P de v_i a v_k está compuesto por un camino mínimo P_{it} de v_i a algún nodo $v_t \neq v_k$ (posiblemente vacío) y la arista (v_t, v_k), donde $1 \leq t < k$ porque se trata del subgrafo inducido por V_k. En particular, P es aquel que tiene el menor peso. Como P_{it} no pasa por v_k, es un camino mínimo en el subgrafo inducido por V_{k-1}. Luego, por hipótesis inductiva, el peso de P es exactamente la expresión enunciada.
- $i = k \wedge j \neq k$. Es análogo al caso $i \neq k \wedge j = k$.
- $i, j \neq k$. Un camino mínimo P de v_i a v_j puede pasar o no por v_k. Si no pasa por v_k, entonces es un camino mínimo en el subgrafo inducido por V_{k-1}. Si pasa por v_k, está compuesto por un camino mínimo de v_i a v_k y otro de v_k a v_j. Luego, como ambas posibilidades corresponden un camino de v_i a v_j, el peso de P es el mínimo de las dos. Esto, por hipótesis inductiva, es la expresión enunciada.

□

Lema 1.3.23. *Sea G un grafo, W su matriz de pesos, $D^{(k)} \in \mathbb{R}^{k\times k}$ tal que $d_{ij}^{(k)}$ es el peso de un camino mínimo de v_i a v_j en el subgrafo de G inducido*

por $\{v_1, v_2, \ldots, v_k\}$ *y* $\Pi^{(k)} \in V^{k\times k}$ *tal que* $\pi_{ij}^{(k)}$ *es el predecesor de* v_j *en un tal camino. Entonces* $\Pi^{(k)}$ *puede definirse recursivamente como*

$$\pi_{11}^{(1)} = nil$$

$$\pi_{ij}^{(k)} = \begin{cases} nil & si\ d_{ij}^{(k)} = \infty \ \vee\ i = j = k \\ v_t & si\ d_{ij}^{(k)} = \infty \ \wedge\ i \neq k \ \wedge\ j = k \\ v_k & si\ d_{ij}^{(k)} = \infty \ \wedge\ i = k \ \wedge\ j \neq k \ \wedge\ t = j \\ \pi_{tj}^{(k-1)} & si\ d_{ij}^{(k)} = \infty \ \wedge\ i = k \ \wedge\ j \neq k \ \wedge\ t \neq j \\ \pi_{ij}^{(k-1)} & si\ d_{ij}^{(k)} = \infty \ \wedge\ i, j \neq k \ \wedge\ d_{ij}^{(k)} = d_{ij}^{(k-1)} \\ \pi_{kj}^{(k-1)} & si\ d_{ij}^{(k)} = \infty \ \wedge\ i, j \neq k \ \wedge\ d_{ij}^{(k)} \neq d_{ij}^{(k-1)} \end{cases}$$

donde t *es el parámetro que realiza el mínimo en la definición de* $d_{ij}^{(k)}$ *para los casos* $i \neq k \wedge j = k$ *e* $i = k \wedge j \neq k$.

Demostración. La demostración se deduce directamente de tomar el predecesor que corresponde según cada caso de la definición de $d_{ij}^{(k)}$. □

Algoritmo 1.3.6: Algoritmo de Dantzig

Entrada: Un grafo G
Salida: Caminos mínimos entre todo par de nodos de G y sus pesos

```
1  D ← W
2  no se muestra el cómputo de Π por simplicidad
3  para k = 2 hasta n hacer
4  |  para i = 1 hasta k − 1 hacer
5  |  |  d_ik ← mín_{1≤t<k}(d_it + d_tk)
6  |  |  d_ki ← mín_{1≤t<k}(d_kt + d_ti)
7  |  fin
8  |  para i = 1 hasta k − 1 hacer
9  |  |  para j = 1 hasta k − 1 hacer
10 |  |  |  d_ij ← mín{d_ij, d_ik + d_ki}
11 |  |  fin
12 |  |  si d_ii < 0 entonces
13 |  |  |  error hay circuitos negativos
14 |  |  fin
15 |  fin
16 fin
17 devolver D, Π
```

Observación. Por simplicidad, en el algoritmo 1.3.6 se muestra sólo el cómputo de las distancias mínimas. Para obtener los caminos mínimos sólo se debe implementar la definición recursiva de Π.

Teorema 1.3.24 (Correctitud del algoritmo de Dantzig)**.** *Sea* G *un grafo (orientado o no). Si* G *tiene circuitos negativos el algoritmo de Dantzig lo detecta; si no, determina un camino mínimo entre todo par de nodos de* G *y sus pesos.*

Demostración. Cuando G no tiene circuitos negativos el teorema se deduce directamente de los lemas 1.3.22 y 1.3.23. Sólo notamos que como D es inicializado con la matriz de pesos de G, $w(v_t, v_k) = d_{tk}$ y $w(v_k, v_t) = d_{kt}$ en cada iteración de los valores de k.

Sea G con circuitos negativos y veamos por el absurdo que el algoritmo lo detecta. Para eso supongamos que $d_{ii} \geq 0$ para todo k y todo i. Sea k' el primer valor de k tal que el subgrafo inducido por V_k (llamémoslo G_k) contiene algún circuito negativo. Nótese que $v_{k'}$ tiene que estar en estos circuitos por cómo definimos k'. Al terminar la iteración de $k = k'$, por el lema 1.3.22, $0 \leq d_{ij} = \text{mín}\{d_{ij}^{(k'-1)},\ d_{ik'}^{(k')} + d_{k'i}^{(k')}\} \leq d_{ik'}^{(k')} + d_{k'i}^{(k')}$ para todo i. En particular, esto vale para un i tal que v_i está en algún circuito negativo simple C de $G_{k'}$. Pero si $C_{ik'}$ y $C_{k'i}$ son los caminos simples del circuito que van de v_i a $v_{k'}$ y de $v_{k'}$ a v_i respectivamente, entonces $w(C) = w(C_{ik'}) + w(C_{k'i}) \geq d_{ik'}^{(k')} + d_{k'i}^{(k')} \geq 0$. Esto es absurdo porque C es un circuito negativo. □

Proposición 1.3.25 (Complejidad del algoritmo de Dantzig)**.** *El algoritmo de Dantzig tiene complejidad $O(n^3)$.*

Observación. Si se tienen los caminos mínimos entre todo par de vértices de un grafo, basta hacer una iteración del algoritmo de Dantzig para determinar los caminos mínimos de un nuevo nodo.

1.4. Grafos eulerianos y hamiltonianos

1.4.1. Grafos eulerianos

Definición 1.4.1. Un circuito en un grafo G es *euleriano* si pasa por cada arista de G exactamente una vez.

Definición 1.4.2. Un grafo es *euleriano* si tiene un circuito euleriano.

Teorema 1.4.1. *Un grafo conexo es euleriano si y sólo si todos sus nodos tienen grado par.*

Demostración. Sea G el grafo conexo.

$\Longrightarrow$. Sea C un circuito euleriano de G. Como C pasa por cada arista de G exactamente una vez, el grado de un nodo de G es igual a la cantidad de aristas incidentes a él por las que pasa C. Luego, dado que en un circuito cada vez que aparece un nodo hay una arista por la que se entra a él y una por la que se sale, el grado de los nodos de G debe ser par.

$\Longleftarrow$. Demostramos por inducción en la cantidad de aristas de G.

Caso base. $m = 3$. G es un circuito simple y, por lo tanto, es euleriano.

Paso inductivo. Supongamos que la implicación vale para todo m' tal que $3 \leq m' < m$ y veamos que vale para m. Sea G un grafo conexo de m aristas y v_0 un nodo cualquiera de G. Podemos construir un circuito C_0 comenzando desde v_0 yendo por una arista no explorada cada vez que llegamos a un nodo hasta volver a v_0. Esto es posible porque G es conexo y sus nodos tienen grado par, por lo cual siempre que entramos a un nodo por una arista vamos a poder salir de él por otra. Si C_0 tiene todas las aristas de G, termina la demostración. Si no, definimos $G' = G \setminus C_0$. Los nodos de G' tienen grado par porque al sacar C_0 de G sacamos una cantidad par de aristas por cada nodo. Luego,

cada componente conexa de G' es un grafo conexo con nodos de grado par y con menos de m aristas y entonces, por hipótesis inductiva, son eulerianos. Sea $C_1, C_2, \ldots, C_k$ un circuito euleriano de cada componente conexa de G'. Como G es conexo, C_0 debe tener vértices $v_1, v_2, \cdots, v_k$, cada uno perteneciente a uno de estos circuitos. Podemos suponer sin pérdida de generalidad que estos vértices aparecen en ese orden en C_0. Luego, podemos definir un circuito euleriano de la siguiente forma. Comenzando en v_0 vamos por C_0 hasta llegar a v_1 y luego recorremos C_1 hasta volver a v_1, de v_1 vamos por C_0 hasta v_2 y luego recorremos C_2 hasta volver a v_2, y así sucesivamente. Finalmente, luego de recorrer C_k y volver a v_k, terminamos el circuito recorriendo lo que falta de C_0 hasta regresar a v_0. □

Observación. El teorema 1.4.1 vale también para multigrafos.

Algoritmo 1.4.1: Algoritmo de Hierholzer

Entrada: Un grafo G conexo con nodos de grado par
Salida: Un circuito euleriano de G

```
1 elegir un nodo v_0 cualquiera
2 construir un circuito C desde v_0 sin repetir aristas
3 mientras E \ C ≠ ∅ hacer
4 |   elegir un nodo u de C tal que (u, v) ∈ E \ C
5 |   construir un circuito P desde u sin repetir aristas y tal que
  |     P ∩ C = ∅
6 |   unir P a C por medio de u
7 fin
8 devolver C
```

Proposición 1.4.2 (Correctitud del algoritmo de Hierholzer)**.** *Sea G un grafo conexo con nodos de grado par. El algoritmo de Hierholzer determina un circuito euleriano de G.*

Demostración. La correctitud del algoritmo se deduce de la demostración del teorema 1.4.1. El primer circuito obtenido en el algoritmo corresponde al circuito C_0 de la demostración y la construcción de los subsiguientes circuitos es equivalente a la aplicación del paso inductivo sobre las componentes conexas de $G \setminus C_0$. Reemplazar u por C' en cada iteración corresponde a armar el recorrido enunciado al final de la demostración. □

Teorema 1.4.3. *Sea $G = (V, E)$ un grafo conexo. G es euleriano si y sólo si E se puede particionar en circuitos simples tal que cada eje de E pertenece a exactamente uno de estos circuitos.*

Demostración.

$\Longrightarrow$. Probemos por inducción en m. En el caso base $m = 3$, por lo que G es un circuito simple y luego E es la partición. Supongamos ahora que la implicación vale para todo m' tal que $3 \leq m' < m$ y veamos que vale para m. Sea G con m aristas y C un circuito euleriano de G. Si C es simple, la partición es E. Si no, C debe contener algún circuito simple C'. El grafo $G' = G \setminus C'$ es euleriano porque $C \setminus C'$ es un circuito euleriano de este. Luego, por hipótesis

inductiva, $E(G') = E \setminus C$ tiene una partción P en circuitos simples. Entonces, $P \cup \{C\}$ es una partición en circuitos simples de E.

$\Longleftarrow$. Sea P la partición de E y probemos por inducción en la cantidad de circuitos simples de P. En el caso base P tiene sólo un circuito simple y entonces G es trivialmente euleriano. Supongamos ahora que la implicación vale para alguna cantidad k de circuitos simples y probemos que vale para $k+1$. Sean $C_1, C_2, \ldots, C_{k+1}$ los circuitos simples de P. La partición $P' = P \setminus \{C_1\}$, por hipótesis inductiva, es una partición de las aristas de un grafo euleriano. Sea C un circuito euleriano de dicho grafo. Como G es conexo, debe haber un nodo $v \in C_1$ que está en un circuito de P' y por lo tanto en C. Luego, el circuito que resulta de unir C_1 con C por medio de v es un circuito euleriano de G porque $C_1 \cap C = \emptyset$ y $C_1 \cup C = E$. □

Definición 1.4.3. Un camino en un grafo G es *euleriano* si pasa por cada arista de G exactamente una vez y no es un circuito.

Teorema 1.4.4. *Un grafo conexo tiene un camino euleriano si y sólo si tiene exactamente dos nodos de grado impar.*

Demostración. Sea G el grafo conexo.

$\Longrightarrow$. Sea C un camino euleriano de G. Como C pasa por cada arista de G exactamente una vez, el grado de un nodo de G es igual a la cantidad de aristas incidentes a él por las que pasa C. En un camino que no es circuito cada vez que aparece un nodo hay una arista por la que se entra a él y una por la que se sale, salvo para el primer nodo que tiene una arista más que sale de él y el último nodo que tiene una arista más que entra a él. Luego, el grado del primer y el último nodo de C debe ser impar y el del resto par. Por lo tanto, G tiene exactamente dos nodos de grado impar.

$\Longleftarrow$. Si G tiene sólo dos nodos el camino euleriano es su única arista. Supongamos entonces que tiene más nodos. Sean u y v dos nodos distintos de G que no son adyacentes. Estos tienen que existir porque si no G sería un grafo completo y todos sus nodos, que son más de dos, tendrían el mismo grado par o impar. Definamos un nuevo grafo $G' = G + (u, v)$. Todos los nodos de G' tienen grado par ya que u y v eran los únicos con grado impar y agregamos una arista entre ellos. Luego, por el teorema 1.4.1, G' tiene un circuito euleriano C, el cual pasa por cada arista de G y la arista (u, v) exactamente una vez. Entonces, el camino $C' = C - (u, v)$ entre u y v no es un circuito y pasa por cada arista de G exactamente una vez. Es decir, es un camino euleriano de G. □

Observación. El teorema 1.4.4 vale también para multigrafos.

Proposición 1.4.5 (Algoritmo de camino euleriano)**.** *Sea G un grafo conexo con exactamente dos nodos de grado impar. Si en el algoritmo de Hierholzer (1.4.1) se elige como nodo inicial v_0 (línea 1) a uno de los nodos de grado impar u y en lugar de construir un circuito desde v_0 (línea 2) se construye un camino desde $v_0 = u$ al otro nodo de grado impar sin repetir aristas, entonces el algoritmo obtiene un camino euleriano de G.*

Demostración. Para la demostración vamos a dar una prueba alternativa de la implicación $\Longleftarrow$ del teorema 1.4.4. Es trivial ver que esta se corresponde con los pasos que realiza el algoritmo enunciado por lo visto en la demostración de correctitud del algoritmo de Hierholzer.

Sean u y v los nodos de grado impar de G y sea P un camino entre ellos que no repite aristas. Si P tiene todas las aristas de G entonces es un camino euleriano y no hay nada que probar. Si no, definimos un grafo $G' = G \setminus P$. Los nodos de G' tienen grado par porque sacamos de G una cantidad impar de aristas incidentes a u y v y una cantidad par de aristas incidentes al resto de los nodos. Luego, por el teorema 1.4.1, las componentes conexas de G' son circuitos eulerianos $C_1, C_2, \ldots, C_k$. Como G es conexo, P debe tener vértices $v_1, v_2, \ldots, v_k$, cada uno perteneciente a uno de estos circuitos. Podemos suponer sin pérdida de generalidad que estos vértices aparecen en ese orden en P. Luego, podemos definir el siguiente camino euleriano. Comenzando en u vamos por P hasta v_1 y luego recorremos C_1 hasta volver a v_1, vamos por P hasta v_2 y luego recorremos C_2 hasta volver a v_2, y así sucesivamente. Finalmente, luego de recorrer C_k y volver a v_k, vamos por P hasta terminar en u. □

Algoritmo 1.4.2: Algoritmo recursivo de circuito y camino euleriano

Entrada: Un grafo G conexo con todos sus nodos de grado par o exactamente dos nodos de grado impar
Salida: Un circuito euleriano o camino euleriano de G

```
1  procedimiento EULERIANO(G, u, S):
2      para cada (u, v) ∈ E no visitado hacer
3          marcar (u, v) como visitado
4          EULERIANO(G, v, S)
5      fin
6      agregar u al principio de S
7  fin
8  S ← lista vacía de nodos
9  elegir un nodo v_0 de grado impar o cualquiera si no hay
10 EULERIANO(G, v_0, S)
11 devolver S
```

Proposición 1.4.6. *Los algoritmos 1.4.2 (versión recursiva) y 1.4.3 (versión iterativa) son implementaciones del algoritmo de Hierholzer que hacen el paso de construcción del circuito inicial de forma que produzca un circuito si el grafo tiene todos sus nodos de grado par o si tiene exactamente dos nodos de grado impar un camino entre estos nodos.*

Demostración. El algoritmo 1.4.2 elige un nodo inicial v_0 como en el agoritmo de Hierholzer. Luego llama a un procedimiento recurisvo con v_0 que para cada arista sin visitar incidente al nodo recibido se llama recursivamente con el otro extremo de la arista. Esto genera un árbol de ejecución en el cual empezando desde el nodo v_0 se construye un camino yendo siempre por una arista no visitada hasta que no queda ninguna tal arista incidente. De esta forma, como vimos en demostraciones anteriores, se produce desde v_0 un circuito si todos los nodos tienen grado par o si hay exactamente dos nodos de grado impar y v_0 es uno de ellos un camino entre estos nodos. Este paso es equivalente a la construcción del circuito inicial en el algoritmo de Hierholzer. Una vez hecho esto, cuando el algoritmo llega al final de la rama del árbol de ejecución, "vuelve hacia atrás"

Algoritmo 1.4.3: Algoritmo iterativo de circuito y camino euleriano

Entrada: Un grafo G conexo con todos sus nodos de grado par o exactamente dos nodos de grado impar

Salida: Un circuito euleriano o camino euleriano de G

```
1  S ← lista vacía de nodos
2  elegir un nodo v_0 de grado impar o cualquiera si no hay
3  agregar v_0 al final de S
4  p ← puntero al último elemento de S
5  mientras quedan aristas sin visitar hacer
6  |  u ← elemento apuntado por p
7  |  buscar una arista (u, v) ∈ E que no fue visitada
8  |  si existe (u, v) entonces
9  |  |  marcar (u, v) como visitada
10 |  |  agregar v a S después del elemento apuntado por p
11 |  |  avanzar p
12 |  si no
13 |  |  retroceder p
14 |  fin
15 fin
```

y construye de la misma forma los circuitos restantes que no fueron visitados, insertándolos en el camino inicial. Esto se corresponde con los pasos realizaods por el ciclo del algoritmo de Hierholzer.

El algoritmo 1.4.3 es simplemente una implementación iterativa del algoritmo 1.4.2. En él se usa un puntero p que apunta al nodo correspondiente al nodo u del procedimiento recursivo y se lo avanza y retrocede de acuerdo al árbol de ejecución de dicho procedimiento. □

Corolario 1.4.7. *Los algoritmos 1.4.2 y 1.4.3 determinan un circuito euleriano del grafo si tiene todos sus nodos de grado par o un camino euleriano si tiene exactamente dos nodos de grado impar.*

Demostración. En la demostración de la proposición 1.4.6 vimos que si G tiene todos sus nodos de grado par el camino inicial construido es un circuito, mientras que si tiene exactamente dos nodos de grado impar es un camino entre ellos. En el primer caso, el algoritmo hace lo que enuncia el algoritmo de Hierholzer y, entonces, determina un circuito euleriano de G. En el otro, implementa el algoritmo definido en la proposición 1.4.5, por lo cual determina un camino euleriano de G. □

Proposición 1.4.8. *La complejidad de los algoritmos 1.4.2 y 1.4.3 para grafos representados con listas de adyacencia es $O(m)$.*

Demostración. La elección de v_0 puede hacerse en $O(m)$ con listas de adyacencia. Para cada nodo de G se consideran sus aristas incidentes a lo sumo una vez. Luego, se consideran como mucho $2m$. En el algoritmo recursivo se recorren las aristas incidentes en tiempo lineal al grado del nodo. En el algoritmom iterativo se puede obtener el mismo costo manteniendo para cada nodo un puntero al últi-

mo elemento recorrido de la lista de adyacencias. Por lo tanto, la complejidad de ambos algoritmos es $O(m)$. □

Definición 1.4.4. Un circuito orientado en un digrafo G es *euleriano* si pasa por cada arista de G exactamente una vez.

Definición 1.4.5. Un grafo orientado es *euleriano* si tiene un circuito orientado euleriano.

Teorema 1.4.9. *Un digrafo fuertemente conexo G es euleriano si y sólo si para todos sus nodos el grado de salida es igual al de entrada.*

Demostración. La demostración es análoga al caso de grafos no orientados. □

Definición 1.4.6. Un camino orientado en un digrafo G es *euleriano* si pasa por cada arista de G exactamente una vez y no es un circuito.

Teorema 1.4.10. *Un digrafo fuertemente conexo G tiene un camino euleriano orientado si y sólo si para todos sus nodos el grado de salida es igual al de entrada salvo para un nodo cuyo grado de salida es uno más que el entrada y otro nodo cuyo grado de entrada es uno más que el de salida.*

Demostración. La demostración es análoga al caso de grafos no orientados. □

Definición 1.4.7. Dado un grafo $G = (V, E)$ con función de peso $w : E \to \mathbb{R}_{\geq 0}$ de sus aristas, el *problema del cartero chino* consiste en encontrar un circuito que pase por cada arista de G al menos una vez de peso mínimo.

Observación.

- La solución del problema del cartero chino para grafos eulerianos es un circuito euleriano.
- Existen algoritmos polinomiales para el problema del cartero chino cuando el grafo es orientado o no orientado, pero no se conocen algoritmos polinomiales si el grafo tiene algunas aristas orientadas y otras no.

1.4.2. Grafos hamiltonianos

Definición 1.4.8. Un circuito en un grafo G es *hamiltoniano* si pasa por cada nodo de G exactamente una vez.

Definición 1.4.9. Un grafo es *hamiltoniano* si tiene un circuito hamiltoniano.

Observación. No se conocen buenas caracterizaciones de grafos hamiltonianos ni algoritmos polinomiales para decidir si un grafo es hamiltoniano o no.

Teorema 1.4.11 (Condición necesaria)**.** *Sea G un grafo hamiltoniano. Si S es un subconjunto no vacío de nodos de G y c la cantidad de componentes conexas de $G \setminus S$, entonces $c \leq |S|$.*

Demostración. Sea C un circuito hamiltoniano de G. Entonces el grafo $C \setminus S$ es un subgrafo generador del grafo $G \setminus S$, es decir, se puede obtener el segundo agregando aristas al primero. Como estas aristas pueden conectar al menos 0 componentes conexas de $C \setminus S$, $G \setminus S$ tendrá a lo sumo tantas componentes

conexas como $C \setminus S$. Luego, basta probar el teorema para el grafo C. Hagamos esto por inducción en el tamaño de S.

Caso base. $|S| = 1$. Sacar cualquier nodo de C resultará en un camino, que tiene una sola componente conexa. Luego $c = 1 \leq 1 = |S|$.

Paso inductivo. Supongamos que el teorema vale para un conjunto de tamaño $k \geq 1$ y veamos que vale para uno de tamaño $k+1$. Sea $S \subset V$ de tamaño $k+1$, $v \in S$ y $S' = S \setminus \{v\}$. Por hipótesis inductiva, la cantidad de componentes conexas c' de $C' = C \setminus S'$ es a lo sumo k. Dado que C es un circuito simple, cada componente conexa de C' debe ser un camino simple o un nodo aislado. Luego, sacar v de C' (obteniendo $C \setminus S$) puede resultar en a lo sumo una componente conexa más. Entonces $c \leq c' + 1 \leq k + 1 = |S|$. □

Teorema 1.4.12 (Dirac)**.** *Sea G un grafo. Si $n \geq 3$ y $d(v) \geq n/2$ para todo $v \in V$, entonces G es hamiltoniano.*

Demostración. Sea G tal que $n \geq 3$ y $d(v) \geq n/2 \ \forall \ v \in V$. G debe ser conexo, ya que si no lo fuera tendría más de una componente conexa, cada una con al menos $n/2 + 1$ nodos, y entonces G tendría al menos $n + 2$ nodos. Sea $P = v_0 v_1 \cdots v_k$ un camino simple de longitud máxima en G, definido como secuencia de nodos. Todos los nodos adyacentes a v_0 y v_k deben estar en P, si no P no sería de longitud máxima. Entonces, al menos $n/2$ nodos de $\{v_1, v_2, \ldots, v_k\}$ son adyacentes a v_0 y al menos $n/2$ nodos de $\{v_0, v_1, \ldots, v_{k-1}\}$ son adyacentes a v_k. Dicho de otra forma, existen al menos $n/2$ nodos $v_i \in \{v_1, v_2, \ldots, v_k\}$ tal que v_i es adyacente a v_0 y existen al menos $n/2$ nodos $v_i \in \{v_1, v_2, \ldots, v_k\}$ tal que v_{i-1} es adyacente a v_k. Luego, como $\{v_1, v_2, \ldots, v_k\}$ tiene a lo sumo $n - 1$ nodos, debe existir algún v_i en él tal que v_i es adyacente a v_0 y v_{i-1} es adyacente a v_k. Podemos definir entonces el circuito $C = v_0 v_i v_{i+1} \cdots v_k v_{i-1} v_{i-2} \cdots v_0$. Afirmamos que C es un circuito hamiltoniano. Si no lo fuera, existiría un nodo $u \in V$ que no está en C. Como G es conexo, u debe ser adyacente a algún nodo x de C. Pero entonces, si y es otro nodo de C adyacente a x, $C - (x, y) + (u, x)$ sería un camino simple de u a y de longitud $k + 1$, que es mayor a la longitud de P. Esto es absurdo. □

Proposición 1.4.13. *Un grafo bipartito con número impar de vértices no puede ser hamiltoniano.*

Demostración. Un circuito hamiltoniano en un grafo con número impar de vértices es un circuito simple de longitud impar. Pero un grafo es bipartito si y sólo si no tiene circuitos simples de longitud impar. □

Problema del viajante de comercio

Definición 1.4.10. Dado un grafo $G = (V, E)$ con función de peso $w : E \to \mathbb{R}_{\geq 0}$ de sus aristas, el *problema del viajante de comercio* (TSP por sus siglas en inglés) consiste en encontrar un circuito hamiltoniano de peso mínimo en G.

Observación. No se conocen algoritmos polinomiales para el problema del viajante de comercio.

Heurística del vecino más cercano Comienza desde un nodo cualquiera y elige siempre la arista de menor peso que va a un nodo no visitado hasta completar el circuito hamiltoniano si es posible. Es un algoritmo goloso y su complejidad es $O(m)$ para listas de adyacencia.

Algoritmo 1.4.4: Heurística del vecino más cercano

Entrada: Un grafo hamiltoniano G con función de peso w
Salida: Un circuito hamiltoniano de G si se encuentra

elegir un nodo v_0 cualquiera
$S \leftarrow \{v_0\}$
$u \leftarrow v_0$
mientras existe algún nodo $v \notin S$ adyacente a u **hacer**
    elegir una arista (u, v) con $v \notin S$ de peso mínimo
    $S \leftarrow S \cup \{v\}$
    $\pi(v) \leftarrow u$
    $u \leftarrow v$
fin
si $S = V \ \wedge \ (u, v_0) \in E$ **entonces**
    $\pi(v_0) \leftarrow u$
    devolver π
fin
si no error no se encontró una solución

Heurística de inserción Comienza con un circuito simple de longitud tres y va insertando nodos en él hasta tener un circuito hamiltoniano si es posible.

Algoritmo 1.4.5: Heurística de inserción

Entrada: Un grafo hamiltoniano G con función de peso w
Salida: Un circuito hamiltoniano de G si se encuentra

$C \leftarrow$ circuito simple de longitud tres
$S \leftarrow$ nodos de C
mientras $S \neq V$ **y** se pueden insertar nodos en C **hacer**
    elegir un nodo $v \notin S$ que se puede insertar en C
    insertar v en C
    $S \leftarrow S \cup \{v\}$
fin
si $S = V$ **entonces devolver** C
si no error no se encontró una solución

El nodo elegido se inserta en el circuito entre un par de nodos consecutivos que minimiza el incremento del peso. Existen varios criterios de elección del nodo a insertar:

- Un nodo al azar.
- El nodo más cercano a un nodo del circuito.
- El nodo más lejano a un nodo del circuito.
- El nodo que hace crecer menos el peso del circuito.

Para grafos euclidianos se puede tomar como circuito inicial la cápsula convexa de los nodos y en cada paso insertar el nodo que forme el mayor ángulo con dos nodos consecutivos del circuito construido hasta el momento.

Heurística del árbol generador mínimo Es usado para grafos completos. Consiste en obtener un árbol generador mínimo del grafo y construir un circuito hamiltoniano usando en lo posible las aristas de este árbol. Para hacer esto define el circuito como la secuencia de nodos en el orden dado por el recorrido DFS de los nodos del AGM. De esta forma el circuito usará las aristas de las ramas del AGM y sólo irá por una arista que no está en él cuando el recorrido llegue al final de una rama y tenga que pasar a otra. La complejidad es de $O(m \log n)$ para listas de adyacencia por el algoritmo de AGM.

Algoritmo 1.4.6: Heurística del árbol generador mínimo

Entrada: Un grafo completo G con $n \geq 3$ y función de peso w
Salida: Un circuito hamiltoniano de G

```
1 T ← árbol generador mínimo de G
2 H ← lista de nodos
3 recorrer T con DFS agregando los nodos a H a medida que son visitados
4 devolver H
```

Heurística 2-opt Es una heurística de mejoramiento que a partir de una solución inicial intercambia las aristas entre dos pares de nodos consecutivos del circuito en cada iteración para mejorar el valor de la solución.

Algoritmo 1.4.7: Heurística 2-opt

Entrada: Un grafo hamiltoniano G con función de peso w
Salida: Un circuito hamiltoniano de G

```
1 obtener una solución inicial H
2 mientras sea posible hacer
3 |   elegir aristas (v_i, v_{i+1}) y (v_j, v_{j+1}) de H tal que
  |     w(v_i, v_{i+1}) + w(v_j, v_{j+1}) > w(v_i, v_j) + w(v_{i+1}, v_{j+1})
4 |   H ← H \ {(v_i, v_{i+1}), w(v_j, v_{j+1})}
5 |   H ← H ∪ {(v_i, v_j), w(v_{i+1}, v_{j+1})}
6 fin
7 devolver H
```

Una generalización de este algoritmo es la heurística k-opt que en cada iteración toma k pares de nodos consecutivos y hace entre ellos un intercambio de aristas que mejore en mayor medida el valor de la solución. En la práctica se usan sólo 2-opt y 3-opt.

1.5. Planaridad

Definición 1.5.1. Una *representación planar* de un grafo G es un conjunto de puntos en el plano que se corresponden con los nodos de G unidos por curvas que se corresponden con las aristas de G sin que estas se crucen entre sí.

Definición 1.5.2. Una *región* en una representación planar es el conjunto de todos los puntos alcanzables sin atravesar nodos ni aristas desde un mismo

punto que no es un nodo ni parte de una arista. Toda representación planar de un grafo tiene exactamente una región de área infinita llamada la *región exteior*. La *frontera* de una región es el circuito que la rodea (puede tener nodos y aristas repetidos) y el *grado* o *tamaño* de una región es el número de aristas que tiene su frontera.

Definición 1.5.3. Un grafo es *planar* si admite una representación planar.

Proposición 1.5.1. *Sea T un árbol. Entonces T es planar.*

Demostración. Supongamos que T no es planar. Entonces, en toda representación de G en el plano debe haber una arista (u, v) que tiene que cruzar necesariamente otro nodo o arista para ir de un extremo al otro. Para que esto suceda al menos uno de los extremos debe estar rodeado por nodos y aristas. Pero esto implicaría que T tiene un circuito, lo cual es absurdo. □

Proposición 1.5.2. *K_5 es el grafo no planar con el menor número de nodos y $K_{3,3}$ es el grafo no planar con el menor número de aristas.*

Proposición 1.5.3. *Si un grafo contiene un subgrafo no planar, entonces es no planar.*

Demostración. Si el grafo G fuera planar se podría obtener una representación planar del subgrafo H eliminando de una representación planar de G los puntos y las curvas que corresponden a los nodos y las aristas que no están en H. □

Definición 1.5.4. *Subdividir* un arista $e = (u, v)$ de un grafo G consiste en agregar un nodo $w \notin V$ a G y reemplazar e por las aristas (u, w) y (w, v).

Definición 1.5.5. Un grafo G' es una *subdivisión* de un grafo G si puede se obtener a partir de G mediante sucesivas operaciones de subdivisión. En ese caso se dice que G es *subdivisible* a G'.

Definición 1.5.6. Dos grafos G y G' son *homeomorfos* si hay un isomorfismo entre una subdivisión de G y una de G'.

Proposición 1.5.4. *Sea G un grafo y G' una subdivisión de G. Entonces G es planar si y sólo si G' es planar.*

Demostración. Dada una representación planar de G, subdividir una arista de G equivale a agregar un punto sobre la curva que corresponde a la arista subdividida, y deshacer esa subdivisión es equivalente borrar el punto agregado. En ambos casos la planaridad de la representación se mantiene. Luego, si G es planar se puede obtener una representación planar de G' agregando puntos de esta forma, y si G' es planar se puede obtener una representación planar de G borrando puntos de la manera descrita. □

Proposición 1.5.5. *Sean G y G' dos grafos homeomorfos. Entonces G es planar si y sólo si G' es planar.*

Demostración. Sea H una subdivisión de G y H' una subdivisión de G' tal que H y H' son isomorfos. Entonces G es planar $\iff$ H es planar $\iff$ H' es planar $\iff$ G' es planar. □

Teorema 1.5.6 (Kuratowski)**.** *Un grafo es planar si y sólo si no contiene ningún subgrafo homeomorfo a K_5 o $K_{3,3}$*

Definición 1.5.7. *Contraer* un arista $e = (u, v)$ de un grafo G consiste en eliminar e de G y considerar a u y v como un solo nodo w de forma que todas las aristas que eran incidentes a u o a v son ahora incidentes a w.

Definición 1.5.8. Un grafo G' es una *contracción* de un grafo G si puede se obtener a partir de G mediante sucesivas operaciones de contracción. En ese caso se dice que G es *contraíble* a G'.

Teorema 1.5.7 (Wagner)**.** *Un grafo es planar si y sólo no contiene ningún subgrafo contraíble a K_5 o $K_{3,3}$.*

Teorema 1.5.8 (Fórmula de Euler)**.** *Sea G un grafo conexo planar. Entonces la cantidad de regiones de cualquier representación planar de G es*

$$r = m - n + 2$$

Demostración. Inducción en m.

Caso base. $m = 0$. Entonces G es el grafo trivial y toda representación planar tiene una región, la región exterior. Luego $m - n + 2 = 0 - 1 + 2 = 1 = r$.

Paso inductivo. Supongamos que el teorema vale para algún $m - 1 \geq 0$ y veamos que vale para m. Si G no tiene circuitos, entonces G es un árbol y toda representación planar tiene sólo a la región exterior. Luego $m - n + 2 = (n - 1) - n + 2 = 1 = r$. Supongamos ahora que G tiene algún circuito y sea e una de sus aristas. Entonces, en toda representación planar de G eliminar la arista e reduce en uno la cantidad de regiones y resulta en la representación planar de un grafo conexo de $m - 1$ aristas. Luego, por hipótesis inductiva, $r - 1 = (m - 1) - n + 2$, lo cual es equivalente a decir que $r = m - n + 2$. □

Proposición 1.5.9. *Sea G un grafo planar con k componentes conexas. Entonces la cantidad de regiones de cualquier representación planar de G es*

$$r = m - n + k + 1$$

Demostración. Sean $C_1, C_2, \ldots C_k$ las componentes conexas de G y sean n_i y m_i la cantidad de nodos y aristas respectivamente de C_i. Cada componente conexa de G debe ser planar porque de lo contrario G no lo sería. Luego, la cantidad de regiones de toda representación planar de C_i es $r_i = m_i - n_i + 2$. Una representación planar de G consiste de una representación planar de cada una de sus componentes conexas. En ella, todas las componentes conexas comparten la región exterior. Luego, la cantidad de regiones de toda representación planar de G es $r = 1 + \sum_{i=1}^{k}(r_i - 1) = 1 + \sum_{i=1}^{k}(m_i - n_i + 1) = 1 + m - n + k$. □

Proposición 1.5.10. *Sea G un grafo conexo planar con $n \geq 3$. Entonces $m \leq 3n - 6$.*

Demostración. La frontera de toda región interior de una representación planar de G debe tener al menos 3 aristas. Además, como G es conexo y $n \geq 3$, G tiene al menos 2 aristas. Luego, la frontera de la región exterior también tiene al menos 3 aristas. Entonces, dado que cada arista puede estar en la frontera de a lo sumo 2 regiones distintas, la cantidad de regiones r es menor o igual a $\frac{2}{3}m$. Luego, $m - n + 2 = r \leq \frac{2}{3}m \iff \frac{1}{3}m \leq n - 2 \iff m \leq 3n - 6$. □

Corolario 1.5.11. *K_5 es no planar.*

Proposición 1.5.12. *Sea G un grafo planar con $n \geq 3$. Entonces*

$$m \leq 3n - 6$$

Demostración. Sea k la cantidad de componentes conexas de G y sean n_i y m_i la cantidad de nodos y de aristas respectivamente de cada una ($i = 1, 2, \ldots, k$). Entonces $m = \sum_{i=1}^{k} m_i \leq \sum_{i=1}^{k}(3n_i - 6) = 3n - 6k \leq 3n - 6$. □

Proposición 1.5.13. *Sea G un grafo conexo planar con $n \geq 3$. Si G no tiene circuitos simples de tres aristas, entonces $m \leq 2n - 4$.*

Demostración. Como G no tiene circuitos simples de tres aristas, la frontera de toda región interior de una representación planar de G debe tener al menos 4 aristas. Dado que además G es conexo con $n \geq 3$, la frontera de la región exterior también debe tener al menos 4 aristas. Entonces, como cada arista puede estar en la frontera de a lo sumo 2 regiones distintas, la cantidad de regiones r es a lo sumo $\frac{2}{4}m = \frac{1}{2}m$. Luego, $m - n + 2 = r \leq \frac{1}{2}m \iff \frac{1}{2}m \leq n - 2 \iff m \leq 2n - 4$. □

Corolario 1.5.14. *Sea G un grafo conexo planar bipartito. Entonces $m \leq 2n - 4$.*

Corolario 1.5.15. *$K_{3,3}$ es no planar.*

Proposición 1.5.16. *Sea G un grafo planar con $n \geq 3$. Si G no tiene circuitos simples de tres aristas, entonces*

$$m \leq 2n - 4$$

Demostración. Análoga a la de la proposición 1.5.12. □

1.5.1. Algoritmos de planaridad

Algoritmo de Demoucron, Malgrange y Pertuiset

Definición 1.5.9. Sea G un grafo y R una representación planar de un subgrafo H de G.

- Una *parte de G relativa a R* es una componente conexa de $G \setminus H$ junto con las aristas que la conectan a los nodos de H (llamadas *aristas colgantes*) o una arista (u, v) de $G \setminus H$ tal que $u, v \in V(H)$.
- Dada una parte p de G relativa a R, un *nodo de contacto* es un nodo de H incidente a una arista colgante de p.
- Una *extensión planar* de R es una representación planar de un subgrafo de G que se obtiene a partir de R.
- Una parte p es *dibujable* en una región f de R si existe una extensión planar de R en la que p queda en f.
- Una parte p es *potencialmente dibujable* en f si todo nodo de contacto de p pertenece a la frontera de f. Llamamos $F(p)$ al conjunto de regiones de R en donde p es potencialmente dibujable.

Algoritmo 1.5.1: Algoritmo de Demoucron, Malgrange y Pertuiset

Entrada: Un grafo G
Salida: Un representación planar de G si existe

```
1  R ← representación planar de cualquier circuito simple de G
2  mientras R no es una representación planar de G hacer
3  |  para cada parte p de G relativa a R obtener F(p)
4  |  si existe una parte p tal que F(p) = ∅ entonces
5  |  |  devolver FALSO
6  |  si no, si existe una parte p tal que |F(p)| = 1 entonces
7  |  |  elegir p y f ∈ F(p)
8  |  si no
9  |  |  elegir cualquier parte p y cualquier región f ∈ F(p)
10 |  fin
11 |  obtener un camino C en p entre dos nodos de contacto de p
12 |  agregar C a R en la región f
13 fin
```

Definición 1.5.10. Sea G un grafo y R una representación planar de un subgrafo de G. R es *extensible* a una representación planar de G si se puede obtener una representación planar de G a partir de R.

Lema 1.5.17. *Si G es planar, la representación planar R el comienzo de cada iteración del ciclo del algoritmo 1.5.1 es extensible a una representación planar de G.*

Teorema 1.5.18 (Correctitud del algoritmo de DMP)**.** *Dado un grafo G, si G es el planar el algoritmo de Demoucron, Malgrange y Pertuiset determina una representación planar de G; si no, indica que no lo es.*

Demostración. Si G es planar, entonces por el lema 1.5.17 y el teorema del invariante el ciclo del algoritmo termina con R siendo una representación planar de G. Si no es planar, en alguna iteración debe existir una parte p de G relativa a R tal que $F(p) = \emptyset$. De lo contrario, se podría extender R en todas las iteraciones y se llegaría eventualmente a una representación planar de G. Luego, cuando el algoritmo se encuentra con un tal p, devuelve falso indicando que G no es planar. □

Proposición 1.5.19 (Complejidad del algoritmo de DMP)**.** *El algoritmo de Demoucron, Malgrange y Pertuiset tiene complejidad $O(n^2)$.*

Algoritmo de Hopcroft y Tarjan

Hopcroft y Tarjan propusieron un algoritmo de planaridad de complejidad $O(n)$ en *John Hopcroft and Robert E. Tarjan. Efficient planarity testing. Journal of the Association for Computing Machinery, 1974, pp. 549–568.*

1.6. Coloreo de grafos

1.6.1. Coloreo de nodos

Definición 1.6.1. Un *coloreo de los nodos* de un grafo $G = (V, E)$ es una asignación $f : V \to C$ tal que $f(u) \neq f(v)$ para todo $(u, v) \in E$. Los elementos de C se llaman *colores*.

Definición 1.6.2. Dado un entero positivo k, un *k-coloreo* de un grafo G es un coloreo de los nodos de G que usa k colores. Si existe un tal coloreo, G se dice *k-coloreable*.

Definición 1.6.3. El *número cromático* de un grafo G, notado $\chi(G)$, es el menor número de colores necesario para colorear los nodos de G. Si $\chi(G) = k$ se dice que G es *k-cromático*.

Proposición 1.6.1. *Las siguientes afirmaciones son verdaderas.*

1. $\chi(K_n) = n$.
2. *Si G es un grafo bipartito con aristas entonces* $\chi(G) = 2$.
3. *Si H_{2k} es un circuito simple par entonces* $\chi(H_{2k}) = 2$.
4. *Si H_{2k+1} es un circuito simple impar entonces* $\chi(H_{2k+1}) = 3$.
5. *Si T es un árbol no trivial entonces* $\chi(T) = 2$.

Demostración.

1. Como todos los nodos son adyacentes entre sí, se necesita un color diferente para cada nodo.
2. Se necesitan al menos dos colores porque G tiene aristas, y como es bipartito basta usar un color para cada uno de los dos conjuntos de una bipartición.
3. Un circuito simple par es un grafo bipartito con aristas. Luego su número cromático es 2.
4. El número cromático no puede ser 1 porque el circuito tiene aristas. Tampoco puede ser 2 porque para eso se deberían alternar los colores a lo largo del circuito pero el último nodo, como se trata de un circuito simple impar, sería adyacente a dos nodos de distinto color y entonces debería tener un tercer color. Luego, como dimos un 3-coloreo del circuito y este necesita al menos tres colores, el número cromático es 3.
5. Se puede probar por inducción en la cantidad de vértices. Un árbol de dos nodos es trivialmente 2-cromático. Si T es un árbol con $n > 2$ nodos y v una de sus hojas, $T - v$ es un árbol no trivial y entonces, por hipótesis inductiva, es 2-cromático. Luego, dado un 2-coloreo de $T - v$, de los dos colores usados se puede colorear v con aquel que no fue usado para su único nodo adyacente.

□

Cotas para el número cromático

Proposición 1.6.2. *Sea G un grafo y H un subgrafo de G. Entonces $\chi(H) \leq \chi(G)$.*

Demostración. Si $f : V(G) \to C$ es un $\chi(G)$-coloreo de G entonces f restringido a $V(H)$ es un coloreo válido de H, pues $(u, v) \in E(H) \subseteq E(G) \implies f(u) \neq f(v)$. Luego, como f restringido a $V(H)$ usa a lo sumo $\chi(G)$ colores, $\chi(H) \leq \chi(G)$. □

Definición 1.6.4. Una *clique* en un grafo es un subgrafo completo maximal. El *número clique* de un grafo G, notado $\omega(G)$, es la cantidad de nodos de una clique máxima de dicho grafo.

Corolario 1.6.3. *Sea G un grafo. Entonces*

$$\omega(G) \leq \chi(G)$$

Observación. La cota del corolario 1.6.3 no es ajustada necesariamente. Se pueden definir grafos de forma que esta cota sea tan mala como se quiera.

Definición 1.6.5. Los *grafos de Mycielski* M_k para enteros positivos k se define recursivamente de la siguiente forma:

1. $M_1 = K_1$
2. $M_2 = K_2$
3. Si M_k tiene p nodos $u_1, u_2, \ldots, u_p$, entonces M_{k+1} tiene $2p+1$ nodos $u_1, u_2, \ldots, u_p, v_1, v_2, \ldots, v_p, w$. M_{k+1} tiene todas las aristas de M_k, aristas uniendo v_i con los vecinos de u_i en M_i y aristas uniendo w con cada v_i.

Proposición 1.6.4. *Sea M_k un grafo de Mycielski con $k \geq 2$. Entonces $\chi(M_k) = k$ y $\omega(M_k) = 2$.*

Notación. Dado un grafo G, notamos $\Delta(G)$ al máximo grado de sus nodos.

Proposición 1.6.5. *Sea G un grafo. Entonces*

$$\chi(G) \leq \Delta(G) + 1$$

Demostración. Se deduce de la proposición 1.6.11. □

Teorema 1.6.6 (Brooks)**.** *Sea G un grafo conexo que no es un circuito impar ni un grafo completo. Entonces*

$$\chi(G) \leq \Delta(G)$$

Teorema 1.6.7 (Teorema de los cinco colores)**.** *Sea G un grafo planar. Entonces $\chi(G) \leq 5$.*

Demostración. Inducción en n.

Caso base. $n \leq 5$. Podemos asignar a codo nodo un color distinto, obteniendo así un coloreo de a lo sumo 5 colores.

Paso inductivo. Supongamos que el teorema vale para algún $n' \geq 5$ y veamos que vale para $n = n' + 1 \geq 6$. G debe tener algún nodo v de grado menor o igual

a 5, si no $2m = \sum_{u \in V} d(u) \geq \sum_{u \in V} 6 = 6n$ implicaría que $m \geq 3n \geq 3n - 6$, lo cual es absurdo por la proposición 1.5.12. Por hipótesis inductiva $G - v$ tiene un 5-coloreo. Si $d(v) < 5$ o bien $d(v) = 5$ y los nodos adyacentes a v sólo usan 4 colores, entonces de los 5 colores podemos asignarle a v uno que no es usado y obtener un 5-coloreo de G. Supongamos entonces que $d(v) = 5$ y sean $u_1, u_2, \ldots, u_5$ los 5 nodos adyacentes a v ordenados en sentido horario según una representación planar de G. Podemos asumir sin pérdida de generalidad que el color de u_i es i para $i = 1, 2, \ldots, 5$. Sea $G_{i,j}$ el subgrafo de G inducido por los nodos de color i y j. Si u_1 y u_3 no pertenecen a una misma componente conexa en $G_{1,3}$ entonces podemos intercambiar los colores 1 y 3 en la componente conexa de u_1 obteniendo así otro coloreo válido y liberando el color 1, el cual podemos asignarle ahora a v. Lo mismo se puede decir para los nodos u_2 y u_4. Supongamos ahora que u_1, u_3 y u_2, u_4 pertenecen a la misma componente conexa en $G_{1,3}$ y $G_{2,4}$ respectivamente. Esto implica que en G existe un camino simple $P_{1,3}$ entre u_1 y u_3 y otro $P_{2,4}$ entre u_2 y u_4. Pero como u_2 está entre u_1 y u_3 en sentido horario en la representación planar de G, u_2 está encerrado en el circuito $(v, u_1) + P_{1,3} + (u_3, v)$. Entonces $P_{2,4}$ debe cruzar algún nodo o arista para ir de u_2 a u_4, lo cual es absurdo porque la representación en cuestión es planar. □

Teorema 1.6.8 (Teorema de los cuatro colores)**.** *Sea G un grafo planar. Entonces $\chi(G) \leq 4$.*

Algoritmos de coloreo de nodos

No se conocen algoritmos polinomiales para calcular el número cromático de un grafo genérico.

Algoritmos heurístcos

Algoritmo 1.6.1: Algoritmo secuencial (heurístico)

Entrada: Un grafo G con nodos en algún orden $v_1, v_2, \ldots, v_n$
Salida: Un coloreo de los nodos de G

para $i = 1, 2, \ldots, n$ **hacer**
  $f(v_i) \leftarrow \text{mín}\{c \mid c \in \mathbb{N} \wedge c \neq f(v_j)\ \forall\ (v_i, v_j) \in E,\ 1 \leq j < i\}$
fin
devolver f

Proposición 1.6.9 (Correctitud del algoritmo secuencial)**.** *Dado un grafo G, el algoritmo secuencial determina un coloreo válido de los nodos de G.*

Demostración. En cada iteración el algoritmo asigna a v_i un color que no fue usado para ninguno de los nodos anteriores adyacentes a él. Luego, determina un coloreo válido de G. □

Proposición 1.6.10 (Complejidad del algoritmo secuencial)**.** *El algoritmo secuencial para grafos representados por listas de adyacencia tiene complejidad $O(n + m)$.*

Notación. Dado un conjunto S de k elementos, $\langle s_1, s_2, \ldots, s_k \rangle_S$ denota un ordenamiento de los elementos de S.

Definición 1.6.6. Sea $G = (V, E)$ un grafo y $\langle v_1, v_2, \ldots, v_n \rangle_V$. Definimos

$$u_S(G, v_1, v_2, \ldots, v_n) = \max_{1 \leq i \leq n} \min\{i,\, d(v_i) + 1\}$$

Proposición 1.6.11. *Sea G un grafo y $\chi_S(G)$ el número de colores usado por el algoritmo secuencial para colorear los nodos de G considerados en el orden $v_1, v_2, \ldots, v_n$. Entonces*

$$\chi_S(G) \leq u_S(G, v_1, v_2, \ldots, v_n)$$

Demostración. Basta probar que $f(v_i) \leq \min\{i,\, d(v_i) + 1\}\ \forall\ 1 \leq i \leq n$. Para todo valor de i el algoritmo asigna el menor color $c \in \mathbb{N}$ que no es usado por ninguno de los nodos adyacentes a v_i de los $i-1$ nodos anteriores. Luego, como hay a lo sumo $k = \min\{i - 1,\, d(v_i)\}$ tales nodos y a estos se les pudo haber asignado a lo sumo k colores distintos, $f(v_i)$ puede ser a lo sumo $k + 1 = \min\{i,\, d(v_i) + 1\}$. □

Definición 1.6.7. Sea G un grafo. Un ordenamiento $\langle v_1, v_2, \ldots, v_n \rangle$ de los nodos de G está en orden *largest first* si $d(v_1) \geq d(v_2) \geq \cdots \geq d(v_n)$. Dado un tal ordenamiento, definimos $u_{LF}(G) = u_S(G, v_1, v_2, \ldots, v_n)$.

Definición 1.6.8. Sea $G = (V, E)$ un grafo. Definimos

$$u_{LF}(G) = u_S(G, v_1, v_2, \ldots, v_n) \quad \text{para } \langle v_1, v_2, \ldots, v_n \rangle_V \text{ largest first.}$$

Proposición 1.6.12. *Sea $G = (V, E)$ un grafo. Entonces*

$$u_{LF}(G) \leq u_S(G, v_1, v_2, \ldots, v_n) \quad \forall\ \langle v_1, v_2, \ldots, v_n \rangle_V$$

Observación. La proposición 1.6.12 no implica necesariamente que el algoritmo secuencial largest first producirá un coloreo con menos cantidad de colores.

Proposición 1.6.13. *Sea G un grafo y $\chi_S(G)$ el número de colores usado por el algoritmo secuencial para colorear los nodos de G considerados en el orden $v_1, v_2, \ldots, v_n$. Si G_i es el subgrafo de G inducido por $v_1, v_2, \ldots, v_i$ entonces*

$$\chi_S(G) \leq 1 + \max_{1 \leq i \leq n} d_{G_i}(v_i)$$

Demostración. Basta probar que $f(v_i) \leq 1 + d_{G_i}(v_i)\ \forall\ 1 \leq i \leq n$. En la demostración de la proposición 1.6.11 en realidad podemos decir que $k = d_{G_i}(v_i)$, porque la cantidad de nodos adyacentes a v_i de los nodos $v_1, v_2, \ldots, v_{i-1}$ es exactamente la cantidad de nodos adyacentes a v_i en el subgrafo inducido por $v_1, v_2, \ldots, v_i$. Luego, como vimos ahí, $f(v_i) \leq k + 1 = d_{G_i}(v_i) + 1$. □

Definición 1.6.9. Sea G un grafo. Un ordenamiento $\langle v_1, v_2, \ldots, v_n \rangle$ de los nodos de G está en orden *smallest last* si v_i es el nodo de menor grado en el subgrafo inducido por $v_1, v_2, \ldots, v_i$, para $i = 1, 2, \ldots, n$.

Definición 1.6.10. Sea $G = (V, E)$ un grafo y $\langle v_1, v_2, \ldots, v_n \rangle_V$. Definimos

$$u_{SL}(G) = 1 + \max_{1 \leq i \leq n} \min_{1 \leq j \leq i} d_{G_i}(v_j) \quad \text{para } \langle v_1, v_2, \ldots, v_n \rangle_V \text{ smallest last}$$

Proposición 1.6.14. *Sea G un grafo y $\chi_{SL}(G)$ el número de colores usado por el algoritmo secuencial para colorear los nodos de G considerados en el orden $v_1, v_2, \ldots, v_n$ smallest last. Entonces*

$$\chi_{SL}(G) \leq u_{SL}(G)$$

Demostración. Basta probar que $f(v_i) \leq 1 + \text{mín}_{1 \leq j \leq i}\, d_{G_i}(v_j)\ \forall\ 1 \leq i \leq n$. Como vimos en la demostración de la proposición 1.6.13, $f(v_i) \leq 1 + d_{G_i}(v_i)$ para todo i. Además, dado que el algoritmo considera a los nodos en orden smallest last, $d_{G_i}(v_j) \leq d_{G_i}(v_i)$ para todo $1 \leq i < j$. Luego $f(v_i) \leq 1 + d_{G_i}(v_j) = 1 + \text{mín}_{1 \leq j \leq i}\, d_{G_i}(v_j)$. □

Proposición 1.6.15. *Sea G un grafo. Entonces $u_{SL}(G) \leq u_{LF}(G)$.*

Proposición 1.6.16. *El algoritmo secuencial smallest last colorea un grafo planar con a lo sumo 6 colores.*

Definición 1.6.11. Sea G un grafo, $f : V \to C$ un coloreo de G y $p, q \in C$. Hacer un p, q-intercambio en f consiste en intercambiar los colores p y q en este coloreo.

Definición 1.6.12. Sea G un grafo, $f : V \to C$ un coloreo de G y $p, q \in C$. Si $v \in G$ es un nodo adyacente a nodos de color p, un p, q-intercambio de f *libera* al color p con respecto a v si luego de la operación v ya no es adyacente a nodos de color p.

Proposición 1.6.17. *Sea G un grafo, $f : V \to C$ un coloreo de G y G_{pq} el subgrafo de G inducido por los nodos con color $p, q \in C$. Dado un nodo $v \in V$, un p, q-intercambio de f libera a p con respecto a v si y sólo si todos los nodos de G_{pq} adyacentes a v tienen color p y el intercambio se hace al menos sobre todas las componentes conexas de G_{pq} que tienen nodos adyacentes a v.*

Demostración.

$\Longrightarrow$. Supongamos que existe un nodo de G_{pq} con color q adyacente a v. Después de hacer el intercambio el color de ese nodo será p, luego el p, q-intercambio no pudo haber liberado a p. Entonces todos los nodos de G_{pq} adyacentes a v deben tener color p. Si hay una componente conexa G_{pq} con nodos adyacentes a v sobre la cual no se hace el intercambio, esta seguirá teniendo al color p en los nodos adyacentes a v. Luego, no se pudo haber liberado a p.

$\Longleftarrow$. Luego del p, q-intercambio todos los nodos adyacentes a v que tienen color p pasarán a tener color q. Entonces, como ningún nodo adyacente a v tiene color q, no quedarán nodos con color p adyacentes a v después del intercambio. □

Observación. El algoritmo secuencial con intercambio no es siempre mejor que el algoritmo secuencial.

Proposición 1.6.18. *El algoritmo secuencial con intercambio colorea un grafo bipartito con 2 colores.*

Proposición 1.6.19. *El algoritmo secuencial smallest last con intercambio colorea un grafo planar con a lo sumo 5 colores.*

Algoritmo 1.6.2: Algoritmo secuencial con intercambio

Entrada: Un grafo G con nodos en algún orden $v_1, v_2, \ldots, v_n$
Salida: Un coloreo de los nodos de G

```
1  k ← 0
2  para i = 1, 2, ..., n hacer
3  |  s ← mín{c | c ∈ ℕ ∧ c ≠ f(v_j) ∀ (v_i, v_j) ∈ E, 1 ≤ j < i}
4  |  si s ≤ k entonces
5  |  |  f(v_i) ← s
6  |  si no
7  |  |  buscar colores p, q ≤ k tal que un p,q-intercambio de f libera a p
   |  |   con respecto a v
8  |  |  si existen p, q entonces
9  |  |  |  hacer el p,q-intercambio
10 |  |  |  f(v_i) ← p
11 |  |  si no
12 |  |  |  f(v_i) ← s ,  k ← k + 1
13 |  |  fin
14 |  fin
15 fin
16 devolver f
```

Algoritmo exacto El algoritmo 1.6.3 es un algoritmo de backtracking exacto. Para cada nodo obtiene un conjunto de colores válidos y considera un coloreo con cada uno de ellos. Implementa una poda por optimalidad con la cual descarta los coloreos que usan más colores que el mejor encontrado hasta el momento. Las variables que usa el algoritmo son:

- i: índice del nodo considerado en la iteración actual
- f: coloreo que se está considerando
- f^*: mejor coloreo encontrado hasta el momento
- q^*: cantidad de colores de f^*
- q_i: cantidad de colores usados para los nodos anteriores al i-ésimo en f
- S_i: conjunto de colores posibles del i-ésimo nodo
- s: color elegido para el i-ésimo nodo en f
- q: cantidad de colores usados luego de elegir el color para el i-ésimo nodo

1.6.2. Coloreo de aristas

Definición 1.6.13. Un *coloreo de las aristas* de un grafo $G = (V, E)$ es una asignación $f : E \to C$ tal que $f(u, w) \neq f(v, w)$ para todo $u, v, w \in V$, es decir, que aristas con un mismo extremo no tengan el mismo color.

Definición 1.6.14. El *índice cromático* de un grafo G, notado $\chi'(G)$, es el menor número de colores necesario para colorear las aristas de G.

Algoritmo 1.6.3: Algoritmo de backtracking

Entrada: Un grafo G y una cota inferior χ de $\chi(G)$
Salida: Un coloreo mínimo de G y $\chi(G)$

```
1  para i = 1, 2, ..., n hacer S_i ← NIL
2  i ← 1 , q_1 ← 0 , q* ← ∞
3  mientras i > 0 ∧ q* > χ hacer
4  |  si S_i = NIL entonces
5  |  |  S_i ← {s | s ∈ ℕ ∧ s ≤ q_i+1 ∧ s ≠ f(v_j) ∀ (v_i, v_j) ∈ E, 1 ≤ j < i}
6  |  fin
7  |  si S_i = ∅ entonces
8  |  |  S_i ← NIL
9  |  |  i ← i − 1
10 |  si no
11 |  |  s ← mín S_i , S_i ← S_i \ {s}
12 |  |  q ← máx{q_i, s}
13 |  |  si q < q* entonces
14 |  |  |  f(v_i) ← s
15 |  |  |  si i < n entonces q_{i+1} ← q , i ← i + 1
16 |  |  |  si no f* ← f , q* ← q
17 |  |  fin
18 |  fin
19 fin
20 devolver f*, q*
```

Teorema 1.6.20 (Vizing)**.** *Sea G un grafo. Entonces*

$$\Delta(G) \leq \chi'(G) \leq \Delta(G) + 1$$

1.7. Matching, conjunto independiente y recubrimientos

Definición 1.7.1. Un *matching* o *correspondencia* entre los nodos de un grafo G es un conjunto M de aristas de G tal que para todo nodo $v \in V$, v es incidente a lo sumo a una arista de M. Es decir, no hay aristas de M que compartan algún extremo.

Definición 1.7.2. Un *conjunto independiente* de un grafo G es un conjunto I de nodos de G tal que para toda arista $e \in E$, e es incidente a lo sumo a un nodo de I. Es decir, los nodos de I no son adyacentes entre sí.

Definición 1.7.3. Un *recubrimiento de aristas* de G es un conjunto R_n de nodos de G tal que para toda arista $e \in E$, e es incidente al menos a un nodo de R_n.

Definición 1.7.4. Un *recubrimiento de nodos* de G es un conjunto R_e de aristas de G tal que para todo nodo $v \in V$, v es incidente al menos a una arista de R_e.

Proposición 1.7.1. *Sea $G = (V, E)$ un grafo y $S \subseteq V$. Entonces S es un conjunto independiente de G si y sólo si $V \setminus S$ es un recubrimiento de aristas de G.*

Demostración. S es un conjunto independiente de $G \iff$ toda arista $e \in E$ es incidente a lo sumo a un nodo de $S \iff$ toda arista $e \in E$ es incidente al menos a un nodo de $V \setminus S \iff V \setminus S$ es un recubrimiento de aristas de G. □

Corolario 1.7.2. *Sea $G = (V, E)$ un grafo y $S \subseteq V$. Entonces S es un conjunto independiente máximo de G si y sólo si $V \setminus S$ es un recubrimiento de aristas mínimo de G.*

Observación. La relación de la proposición 1.7.1 no se mantiene para matchings y recubrimientos de nodos. Por ejemplo, si e es la única arista de K_2, $\{e\}$ es un matching pero $E \setminus \{e\} = \emptyset$ no es un recubrimiento de nodos.

Definición 1.7.5. Sea M un matching en un grafo G. Un nodo $v \in V$ se dice *saturado* por el matching M si hay una arista de M que es incidente a v.

Definición 1.7.6. Sea M un matching en un grafo G. Un *camino alternado* en G con respecto a M es un camino simple en el cual se alternan aristas que están M y aristas que no están en M.

Definición 1.7.7. Sea M un matching en un grafo G. Un *camino de aumento* en G con respecto a M es un camino alternado entre nodos no saturados por M.

Lema 1.7.3. *Sea M_0 y M_1 dos matchings en un grafo $G = (V, E)$ y sea $G' = (V, E')$ con $E' = M_0 \triangle M_1$ (diferencia simétrica entre M_0 y M_1). Entonces las componentes conexas de G' pueden ser:*

1. *Un nodo aislado.*
2. *Un camino simple con aristas alternadamente en M_0 y M_1.*
3. *Un circuito simple con aristas alternadamente en M_0 y M_1.*

Demostración. $E' = M_0 \triangle M_1 = (M_0 \setminus M_1) \cup (M_1 \setminus M_0)$ es la unión de dos conjuntos que son matchings de G, pues sacar aristas de un matching resulta en otro matching. Como los nodos de G son incidentes a lo sumo a una arista de un matching, en la unión de dos matchings son incidentes a lo sumo a dos aristas. Es decir, en el subgrafo de G inducido por esta unión todos los nodos tienen grado menor o igual a 2, lo cual implica que las componentes conexas pueden ser nodos aislados, caminos simples o circuitos simples. Más aún, como en un matching no puede haber caminos ni circuitos, si una componente conexa es un camino o circuito simple debe tener sus aristas alternadamente en cada matching de la unión. □

Teorema 1.7.4. *Sea G un grafo. M es un matching máximo de G si y sólo si no existe un camino de aumento en G con respecto a M.*

Demostración.

$\implies$. Supongamos que existe un camino de aumento P con respecto a M. Como P es un camino alternado entre nodos no saturados por M, su cantidad

de aristas que no están en M es uno más que su cantidad de aristas que sí lo están. Además, dado que P es alternado, $P \setminus M$ es un matching de los nodos de P. Si sacamos de M sus aristas que están en P, obtenemos un matching M' que no satura ningún nodo de P. Debido a esto, como $P \setminus M$ sólo satura nodos de P, $M' \cup (P \setminus M)$ es un matching válido, que por lo visto antes tendrá una arista más que M. Esto es absurdo porque M es máximo.

$\Longleftarrow$. Supongamos que $M = M_0$ no es un matching máximo. Luego debe existir un matching M_1 con más aristas. Sea $G' = (V, E')$ con $E' = M_0^* \cup M_1^*$ donde $M_0^* = M_0 \setminus M_1$ y $M_1^* = M_1 \setminus M_0$. Como M_0^* y M_1^* son disjuntos, G' debe tener más aristas de M_1^* que de M_0^*. Luego, tiene alguna componente conexa C con más aristas de M_1^*. Por lo enunciado en el lema 1.7.3, C puede ser un camino o circuito simple con aristas alternadamente en M_0^* y M_1^*. C no puede ser un circuito simple porque tendría la misma cantidad de aristas de ambos conjuntos. Luego, es un camino simple, que para tener más aristas de M_1^*, debe empezar y terminar con una arista de M_1^*. Pero entonces C es un camino alternado entre dos nodos no saturados por M_0, es decir, un camino de aumento con respecto a $M_0 = M$. Esto es absurdo. □

Algoritmo 1.7.1: Algoritmo de caminos de aumento

Entrada: Un grafo G
Salida: Un matching máximo de G

```
1 M ← ∅
2 repetir
3 |   P ← camino de aumento de G con respecto a M
4 |   M ← (M \ P) ∪ (P \ M)
5 hasta que P = ∅
6 devolver M
```

Proposición 1.7.5 (Correctitud del algoritmo de camino de aumento)**.** *Dado un grafo G, el algoritmo de caminos de aumento determina un matching máximo de los nodos de G.*

Demostración. La correctitud del algoritmo se deduce de la demostración del teorema 1.7.4. Para la implicación $\Longrightarrow$ vimos que si $P \neq \emptyset$ es un camino de aumento con respecto a M, entonces $(M \setminus P) \cup (P \setminus M)$ es un matching válido de mayor tamaño que M. Luego, el algoritmo obtiene un matching de más grande en cada iteración hasta que $P = \emptyset$, es decir, hasta que no existan caminos de aumento con respecto a M. Para la implicación $\Longleftarrow$ del teorema vimos que si esto sucede entonces M debe ser un matching máximo. Por lo tanto, el algoritmo es correcto. □

Observación. Micali y Vazirani propusieron un algoritmo de matching máximo de complejidad $O(\sqrt{n} \cdot m)$ en *S. Micali and V. V. Vazirani. An $O(\sqrt{|V|} \cdot |E|)$ algorithm for finding maximum matching in general graphs. Proceedings of IEEE 21st Annual Symposium on Foundations of Computer Science, 1980, pp. 17–27.*

Teorema 1.7.6. *Sea G un grafo sin nodos aislados. Si M es un matching máximo en G y R_n un recubrimiento de nodos mínimo de G, entonces*

$$|M| + |R_e| = n$$

Demostración.

$\leq$. Podemos obtener un recubrimiento de nodos a partir de M agregando una arista por cada nodo que no satura M. Luego, dado que la cantidad de nodos no saturados por M es $n - 2|M|$, este recubrimiento tendrá tamaño $|M| + (n - 2|M|) = n - |M|$. Entonces, como R_e es un recubrimiento mínimo, $|R_e| \leq n - |M|$. Por lo tanto $|M| + |R_e| \leq n$.

$\geq$. Podemos obtener un matching a partir de R_e sacando las aristas que recubren un nodo que ya fue recubierto por otra. La cantidad de recubrimientos de R_e (número de veces que un nodo es recubierto por una arista) es $2|R_e|$. Luego, la cantidad de recubrimientos "innecesarios" (recubrimientos de nodos ya recubiertos) es $2|R_e| - n$. Como R_e es un recubrimiento mínimo, cada arista de R_e puede hacer a lo sumo un recubrimiento innecesario, si no esa arista no pertenecería a R_e. Luego, para obtener un matching es necesario sacar $2|R_e| - n$ aristas de R_e. Esto resulta en un matching de tamaño $|R_e| - (2|R_e| - n) = n - |R_e|$. Entonces, como M es un matching máximo, $|M| \geq n - |R_e|$. Por lo tanto $|M| + |R_e| \geq n$. □

Teorema 1.7.7. *Sea G un grafo. Si I es un conjunto independiente máximo de G y R_n un recubrimiento de aristas mínimo de G, entonces*

$$|I| + |R_n| = n$$

Demostración. Por el corolario 1.7.2, $V \setminus I$ es un recubrimiento de aristas mínimo de G, por lo cual $|R_n| = |V \setminus I|$. Entonces $|I| + |R_n| = |I| + |V \setminus I| = |V| = n$. □

Proposición 1.7.8. *Sea G un grafo. Un recubrimiento de nodos de G es minimal si y sólo si no contiene caminos ni circuitos simples de longitud mayor o igual a 3.*

Demostración. Sea R_e el recubrimiento de nodos.

$\Longrightarrow$. Supongamos que R_e contiene un camino o circuito simple $P = e_1 e_2 \cdots e_k$ de longitud mayor o igual a 3. Un extremo de e_2 es recubierto por e_1 y el otro por e_3. Pero entonces podemos sacar e_2 de R_e y seguir teniendo un recubrimiento de nodos. Esto es absurdo porque R_e es minimal.

$\Longleftarrow$. Sea G' el grafo inducido por R_e. Dado que G' no tiene circuitos simples, este es un bosque, y como sólo puede tener caminos simples de longitud menor a 3, sus componentes conexas son árboles de 2 o 3 nodos. Luego, sacar una arista de R_e desconectaría a uno de estos árboles en dos nodos aislados o en un nodo aislado y 2 nodos adyacentes. En cualquier caso queda algún nodo aislado y R_e deja de ser un recubrimiento de nodos. Por lo tanto, R_e debe ser minimal. □

Proposición 1.7.9. *Sea G un grafo sin vértices aislados, I un conjunto independiente de G y R_e un recubrimiendo de nodos de G. Entonces $|I| \leq |R_e|$.*

Demostración. Como I es un conjunto independiente, toda arista de G puede recubrir a lo sumo a un nodo de I. Luego, R_e debe tener al menos una arista distinta por cada nodo de I. Por lo tanto, $|I| \leq |R_e|$. □

1.8. Flujo en redes

Definición 1.8.1. Una *red* N es un grafo orientado conexo que tiene dos nodos distinguidos: una *fuente* s, con grado de salida positivo, y un *sumidero* t, con grado de entrada positivo.

Definición 1.8.2. Una *función de capacidad* en una red $N = (V, E)$ es una función $c : E \to \mathbb{R}_{\geq 0}$.

Definición 1.8.3. Un *flujo* en una red $N = (V, E)$ con función de capacidad c es una función $f : E \to \mathbb{R}_{\geq 0}$ que verifica:

1. $0 \leq f(e) \leq c(e)$ para todo arco $e \in E$.
2. *Ley de conservación de flujo*

$$\sum_{(u,v)\in E} f(u,v) = \sum_{(v,u)\in E} f(v,u) \qquad \forall\ v \in V \setminus \{s,t\}$$

El *valor del flujo* f es $|f| = \sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s)$.

Definición 1.8.4. El *problema de flujo máximo* consiste en determinar el flujo de valor máximo que se puede definir en una red con cierta función de capacidad.

Definición 1.8.5. Un *corte* en la red $N = (V, E)$ es un subconjunto $S \subseteq V \setminus \{t\}$ tal que $s \in S$.

Notación. Dada una red $N = (V, E)$ y $S, T \subseteq V$ llamamos ST al conjunto de arcos de N que van de un nodo de S a uno de T. Es decir, $ST = \{(u, v) \in E \mid u \in S \wedge v \in T\}$. Para denotar al conjunto $V \setminus S$ escribimos $\bar{S}$.

Proposición 1.8.1. *Sea f un flujo definido en una red N. Si S es un corte de N, entonces*

$$|f| = \sum_{e\in S\bar{S}} f(e) - \sum_{e\in \bar{S}S} f(e)$$

Demostración. Inducción en el tamaño de S.

Caso base. $|S| = 1$. Entonces $S = \{s\}$ y la proposición vale trivialmente por la definición de valor del flujo.

Paso inductivo. $|S| > 1$. Sea u un nodo de S y por claridad definimos $U = \{u\}$. Si pasamos u de S a $\bar{S}$ obtenemos un nuevo corte $T = S \setminus U$ y al hacer esto las aristas que entran y salen del corte ($\bar{S}S$ y $S\bar{S}$) cambian (a $\bar{T}T$ y $T\bar{T}$). A los arcos que salen del corte se les suman los que van de $S \setminus U$ a u, pero desaparecen los arcos que van de u a $\bar{S}$. Para los arcos que entran se agregan los que van de u a $S \setminus U$, pero los que van de $\bar{S}$ a u no están más. Como la red no tiene loops podemos decir que los arcos entre u y $S \setminus U$ son arcos entre u y S. Luego, lo dicho recién se traduce a que $T\bar{T} = S\bar{S} \cup SU \setminus U\bar{S}$ y $\bar{T}T = \bar{S}S \cup US \setminus \bar{S}U$. Luego, como $|T| = |S| - 1$, por hipótesis inductiva

$$\begin{aligned}
|f| &= \sum_{e\in T\bar{T}} f(e) - \sum_{e\in \bar{T}T} f(e) \\
&= \Big(\sum_{e\in S\bar{S}} f(e) + \sum_{e\in SU} f(e) - \sum_{e\in U\bar{S}} f(e)\Big) - \Big(\sum_{e\in \bar{S}S} f(e) + \sum_{e\in US} f(e) - \sum_{e\in \bar{S}U} f(e)\Big)
\end{aligned}$$

Si juntamos las sumatorias de SU con US y $U\bar{S}$ con $\bar{S}U$ y usamos la ley de conservación de flujo:

$$\begin{aligned}
|f| &= \sum_{e\in S\bar{S}} f(e) - \sum_{e\in \bar{S}S} f(e) + \sum_{(u,v)\in E} f(u,v) - \sum_{(v,u)\in E} f(v,u) \\
&= \sum_{e\in S\bar{S}} f(e) - \sum_{e\in \bar{S}S} f(e)
\end{aligned}$$

□

Definición 1.8.6. La *capacidad de un corte* S en una red N con función de capacidad c se define como

$$c(S) = \sum_{e \in S\bar{S}} c(e)$$

Proposición 1.8.2. *Sea N una red con función de capacidad c. Si S es un corte y F el valor de un flujo en N, entonces*

$$F \leq c(S)$$

Demostración. $|f| = \sum_{e \in S\bar{S}} f(e) \ - \sum_{e \in \bar{S}S} f(e) \leq \sum_{e \in S\bar{S}} f(e) \leq \sum_{e \in S\bar{S}} c(e) = c(S)$ □

Corolario 1.8.3 (Certificado de optimalidad)**.** *Sea N una red con función de capacidad c. Si S es un corte en N y f un flujo en N tal que $|f| = c(S)$, entonces f define un flujo máximo y S un corte de capacidad mínima.*

Redes con arcos en ambas direcciones Sea $N = (V, E)$ una red con función de capacidad c que tiene pares de nodos para los cuales existen ambos arcos entre ellos. El problema de flujo máximo en la red N se puede reducir a uno equivalente en una red N' con arcos en una sola dirección. Para ello se define N' como una red igual a N con función de capacidad c' igual a c salvo que para todo $u, v \in V$ tal que $(u, v), (v, u) \in E$ se agrega un nodo w y se reemplaza el arco (u, v) (o el otro) por los arcos (u, w) y (w, v), y se define $c'(u, w) = c'(w, u) = c(u, w)$. Luego, dado un flujo máximo f' en N', el flujo máximo f en N equivalente tendrá $f(u, v) = f'(u, w) = f'(w, v)$.

Notación. De acá en adelante asumimos que las redes tienen arcos en una sola dirección a menos que se aclare lo contrario.

Redes con múltiples fuentes y sumideros Si se quiere encontrar un flujo máximo en una red con varias fuentes $s_1, s_2, \ldots, s_p$ y varios sumideros $t_1, t_2, \ldots, t_q$, se puede agregar una fuente s con arcos (s, s_i) y capacidad $c(s, s_i) = \infty$ para $i = 1, 2, \ldots, p$ y un sumidero t con arcos (t_i, t) y capacidad $c(t_i, t) = \infty$ para $i = 1, 2, \ldots, q$. De esta forma, un flujo máximo en esta nueva red se corresponde con un flujo máximo en la red original.

1.8.1. Método de Ford-Fulkerson

Definición 1.8.7. Sea $N = (V, E)$ una red con función de capacidad c y sea f un flujo en N. La *red residual* de N es una red $R(N, f) = (V, E_R)$ tal que

$$\begin{array}{ll} (u, v) \in E_R & \text{si} \quad f(u, v) < c(u, v) \\ (v, u) \in E_R & \text{si} \quad f(u, v) > 0 \end{array} \qquad \forall\ (u, v) \in E$$

Un *camino de aumento* es un camino orientado de s a t en $R(N, f)$.

Proposición 1.8.4 (Correctitud del algoritmo de camino de aumento)**.** *Dada una red residual $R = (V, E_R)$, el algoritmo de camino de aumento determina un camino de aumento en R si existe y devuelve un camino vacío si no.*

Algoritmo 1.8.1: Algoritmo de camino de aumento

Entrada: Una red residual $R = (V, E_R)$
Salida: Un camino de aumento en R

```
1  S ← {s}
2  mientras t ∉ S y existe (u, v) ∈ E_R tal que u ∈ S, v ∉ S hacer
3  |  π(v) ← u
4  |  S ← S ∪ {v}
5  fin
6  si t ∈ S entonces
7  |  reconstruir un camino P entre s y t con π
8  |  devolver P
9  si no
10 |  devolver ∅
11 fin
```

Demostración. Primero probamos el siguiente invariante para el ciclo del algoritmo: al comienzo de cada iteración los nodos de S son alcanzables desde s y π determina un camino de s a ellos. En la primera iteración esto vale trivialmente porque $S = \{s\}$. Supongamos ahora que el invariante vale en cierta iteración y veamos que vale en la siguiente. El algoritmo toma un arco (u, v) con $u \in S$ y $v \notin S$. Como u es alcanzable desde s por hipótesis, podemos ir de s a u por algún camino y luego llegar a v por la arista (u, v). Además, como π determina un camino P de s a u, definir $\pi(v) = u$ determina el camino $P + (u, v)$ de s a v. Por lo tanto el invariante se preserva para la siguiente iteración.

Con esto podemos afirmar que si $t \in S$ al terminar el ciclo entonces π determina un camino de s a t. Si $t \notin S$, el ciclo debe haber terminado por no existir aristas que cruzan S. Esto implica que no hay caminos de aumento en R, porque si hubiera uno alguna de sus arista debería cruzar S. □

Definición 1.8.8. Sea N una red con función de capacidad c y P un camino de aumento de N con respecto al flujo f. Definimos

$$\Delta(u, v) = \begin{cases} c(u, v) - f(u, v) & \text{si } (u, v) \in E \\ f(u, v) & \text{si } (u, v) \notin E \end{cases} \qquad \forall\ (u, v) \in P$$

$$\Delta(P) = \min_{e \in P} \Delta(e)$$

Proposición 1.8.5. *Sea $N = (V, E)$ una red con función de capacidad c, f un flujo en N y P un camino de aumento en $R(N, f)$. Entonces el flujo definido por*

$$\bar{f}(u, v) = \begin{cases} f(u, v) & si\ (u, v), (v, u) \notin P \\ f(u, v) + \Delta(P) & si\ (u, v) \in P \\ f(u, v) - \Delta(P) & si\ (v, u) \in P \end{cases} \qquad \forall (u, v) \in E$$

es un flujo factible en N y $|\bar{f}| = |f| + \Delta(P) > |f|$.

Demostración. Primero veamos que $0 \leq \bar{f}(u, v) \leq c(u, v)$. Cuando $(u, v), (v, u) \notin P$ el valor de $\bar{f}$ es igual al de f, por lo que la desigualdad vale. Para los otros dos casos, como $\Delta(P) > 0$ por definición, tenemos que

- $(u,v) \in P \implies 0 \leq f(u,v) + \Delta(P) \leq f(u,v) + \Delta(u,v) \leq c(u,v)$
- $(v,u) \in P \implies c(u,v) \geq f(u,v) - \Delta(P) \geq f(u,v) - \Delta(u,v) \geq 0$

Falta ver que se cumple la ley de conservación de flujo para todo $v \in V \setminus \{s,t\}$. En todos los casos de la definición de $\bar{f}(e)$ se encuentra el término $f(e)$; lo único que cambia es la presencia del término $\pm\Delta(P)$. Luego, podemos decir que

$$\sum_{(u,v)\in E} \bar{f}(u,v) = \sum_{(u,v)\in E} f(u,v) + \sum_{\substack{(u,v)\in E \\ (u,v)\in P}} \Delta(P) - \sum_{\substack{(u,v)\in E \\ (v,u)\in P}} \Delta(P)$$

$$\sum_{(v,u)\in E} \bar{f}(v,u) = \sum_{(v,u)\in E} f(v,u) - \sum_{\substack{(v,u)\in E \\ (u,v)\in P}} \Delta(P) + \sum_{\substack{(v,u)\in E \\ (v,u)\in P}} \Delta(P)$$

Lo que queremos ver es que estas dos expresiones son iguales o, equivalentemente, que su resta es cero. La resta entre los primeros términos de la expresión de arriba y la de abajo es cero por la conservación de flujo de f. La resta entre los segundos términos no es más que la sumatoria de $\Delta(P)$ sobre las aristas de P que entran a v y, similarmente, la de los terceros términos es el opuesto de la sumatoria de $\Delta(P)$ sobre las aristas de P que salen de v. Si $v \notin P$ estas dos sumatorias son cero. Si $v \in P$, como P es un camino de s a t y $v \neq s,t$, v es un nodo intermedio en este camino. Luego, cada vez que se entra a él por un arco se debe salir por otro. Esto quiere decir que en P entran y salen la misma cantidad de arcos de v, por lo cual la resta de estas sumatorias es cero. Entonces

$$\sum_{(u,v)\in E} \bar{f}(u,v) - \sum_{(v,u)\in E} \bar{f}(v,u) = \sum_{(u,v)\in P} \Delta(P) - \sum_{(v,u)\in P} \Delta(P) = 0$$

□

Teorema 1.8.6. *Sea f un flujo en una red N. Son equivalentes*

1. *f es un flujo máximo.*
2. *No existe camino de aumento en $R(N,f)$.*
3. *$|f| = c(S)$ para algún corte S de capacidad mínima.*

Demostración.

1 $\implies$ 2. Si existiera un camino de aumento P podría definir el flujo $\bar{f}$ de la proposición 1.8.5 que tendría valor $|f| + \Delta(P) > |f|$. Esto es absurdo porque f es un flujo máximo.

2 $\implies$ 3. Como no existe camino de aumento en $R(N,f)$, debe haber un corte S tal que no hay arcos que salen de él en $R(N,f)$. Luego, por definición de red residual, $f(e) = c(e)$ para todo $e \in S\bar{S}$ y $f(e) = 0$ para todo $e \in \bar{S}S$. Entonces, por la proposición 1.8.1

$$|f| = \sum_{e\in S\bar{S}} f(e) - \sum_{e\in \bar{S}S} f(e) = \sum_{e\in S\bar{S}} c(e) = c(S)$$

3 $\implies$ 1. El corolario 1.8.3 indica que F debe ser un flujo máximo. □

Observación. El flujo inicial puede ser $f(e) = 0$ para todo arco e.

Algoritmo 1.8.2: Método de Ford-Fulkerson

Entrada: Una red $N = (V, E)$ con y una función de capacidad c
Salida: Un flujo máximo en N

```
1  f ← cualquier flujo factible en N
2  mientras existe camino de aumento P en R(N, f) hacer
3  |  para cada (u, v) ∈ P hacer
4  |  |  si (u, v) ∈ E entonces
5  |  |  |  f(u, v) ← f(u, v) + Δ(P)
6  |  |  si no
7  |  |  |  f(v, u) ← f(v, u) − Δ(P)
8  |  |  fin
9  |  fin
10 fin
```

Proposición 1.8.7. *El valor de un flujo máximo en una red con capacidades enteras es entero.*

Demostración. El valor de un flujo máximo es la capacidad de un corte mínimo, y esta es entera porque los arcos tienen capacidades enteras. □

Proposición 1.8.8. *Si las capacidades de la red y los valores del flujo inicial son números enteros, el método de Ford-Fulkerson hace a lo sumo F iteraciones, donde F es el valor de un flujo máximo.*

Demostración. En cada iteración el método incrementa el valor del flujo en $\Delta(P)$, donde P es un camino de aumento. Como las capacidades y el flujo inicial de los arcos es entero, para todo arco e de la red residual $\Delta(e)$ será entero. Luego $\Delta(P) > 0$ será al menos 1 y la cantidad de iteraciones necesaria será a lo sumo el valor de un flujo máximo. □

Observación. Existen redes para las cuales la cota de la proposición 1.8.8 pueden alcanzarse. Por ejemplo, para la red de la figura 1.1 en la red residual hay siempre un solo arco entre u y v que cambia de dirección en cada iteración del algoritmo. Luego, si siempre se elige un camino de aumento que pasa por este arco, sólo se podrá incrementar en 1 el valor del flujo.

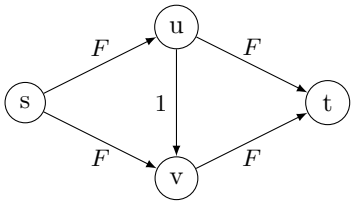

Figura 1.1: Red en la cual Ford-Fulkerson puede hacer F iteraciones

Observación. Si las capacidades de la red son número racionales no enteros se las puede multiplicar por un valor suficientemente grande para convertirlas en enteros.

Observación. Si las capacidades de la red o los valores del flujo inicial son números irracionales el método de Ford-Fulkerson puede no terminar.

Proposición 1.8.9 (Complejidad del método de Ford-Fulkerson)**.** *El método de Ford-Fulkerson para redes con capacidades enteras representadas por listas de adyacencia tiene complejidad $O(nF)$ donde F es el valor de un flujo máximo.*

Demostración. Podemos definir flujo inicial de cada arco como 0 con costo $O(m)$ y la red residual se puede computar en $O(m)$ antes de iniciar el ciclo. En cada iteración el camino de aumento puede obtenerse con BFS o DFS en $O(m)$. Luego, por la proposición 1.8.8, la complejidad del algoritmo será $O(nF)$. □

Algoritmo de Edmonds-Karp

El algoritmo de Edmonds-Karp es una implementación del método de Ford-Fulkerson que utiliza BFS para determinar un camino de aumento en cada iteración.

Proposición 1.8.10. *El algoritmo de Edmonds-Karp hace a lo sumo $O(nm)$ iteraciones.*

Proposición 1.8.11 (Complejidad del algoritmo de Edmonds-Karp)**.** *El algoritmo de Edmonds-Karp para redes representadas por listas de adyacencia tiene complejidad $O(nm^2)$.*

Demostración. El flujo inicial y la red residual se pueden definir en $O(m)$. La proposición 1.8.10 dice que el algoritmo hace $O(nm)$ iteraciones, y en cada una el costo de hacer BFS y aumentar el flujo es $O(m)$. Entonces la complejidad es $O(nm^2)$. □

Capítulo 2

Complejidad

2.1. Conceptos básicos

Definición 2.1.1. Un *algoritmo eficiente* es un algoritmo de complejidad polinomial. Se dice que un problema está *bien resuelto* si se conocen algoritmos eficientes para resolverlo. Un problema es *intratable* si no puede ser resuelto por ningún algoritmo eficiente.

Observación. Un problema puede ser intratable por distintos motivos. Por ejemplo, puede ser porque requiere una respuesta de longitud no polinomial (pedir todas las permutaciones de los elementos de la entrada) o porque el problema es indecidible (problema de la parada).

Definición 2.1.2. Una *instancia* de un problema es una especificación de sus parámetros. El conjunto de instancias de un problema Π es su *dominio* y se nota D_Π.

Definición 2.1.3. Un *problema de decisión* es un problema Π cuya respuesta es SI o NO. Tiene asociado un subconjunto $Y_\Pi \subseteq D_\Pi$ de instancias cuya respuesta es SI y un subconjunto $N_\Pi \subseteq D_\Pi$ de instancias cuya respuesta es NO.

Observación. El estudio de la teoría de complejidad se aplica a problemas de decisión.

Definición 2.1.4. Dada I una instancia de un problema de optimización Π, se definen las siguientes *versiones* del problema:

- *Optimización*: encontrar una *solución óptima* de Π para I (de valor mínimo o máximo).
- *Evaluación*: determinar el *valor* de una solución óptima de Π para I.
- *Decisión*: dado un número k determinar si existe una solución factible de Π para I cuyo valor esté acotado por k (por arriba si el problema es de minimización y por abajo si es de maximización).
- *Localización*: dado un número k determinar una solución factible de Π para I cuyo valor esté acotado por k.

Observación. Si se resuelve el problema de decisión de Π se puede:

- Resolver el problema de evaluación haciendo búsqueda binaria sobre el parámetro k.
- Resolver el problema de localización resolviendo el problema de decisión para el parámetro k sobre versiones reducidas de I.
- Resolver el problema de optimización resolviendo el problema de decisión para el valor óptimo sobre versiones reducidas de I.

Definición 2.1.5. Un algoritmo para resolver un problema es *pseudopolinomial* si la complejidad del mismo es polinomial en función del *valor* (y posiblemente el tamaño) de la entrada.

2.2. Modelos de cómputo

Los modelos de cómputo son modelos formales que describen cómo un conjunto de datos de salida se computa en una máquina abstracta a partir de un conjunto de datos de entrada. Sirven para expresar cualquier algoritmo. La *máquina de Turing* (Alan Turing, 1937) y la *máquina de acceso aleatorio* (Aho, Hopcroft y Ullman, 1974) son dos modelos polinomialmente equivalentes. Es decir, uno puede simular las operaciones del otro con un costo polinomial.

2.2.1. Máquina de Turing

Una máquina de Turing consiste de una cinta de memoria infinita en ambas direcciones que se encuentra dividida en celdas discretas y una cabeza lecto-escritora capaz de leer y escribir símbolos en las celdas de la cinta. La cinta posee una celda numerada con 0. Hacia la derecha de esta se enumeran las celdas de forma creciente con los enteros positivos y hacia su izquierda de forma decreciente con los enteros negativos.

La máquina tiene un conjunto finito de símbolos Σ llamado *alfabeto* que junto con un símbolo distinguido $*$ llamado *blanco* forma el conjunto $\Gamma = \Sigma \cup \{*\}$ de símbolos que pueden tener las celdas de la cinta. Cuenta también con un conjunto finito de *estados* Q de los cuales hay un *estado inicial* q_0 y un conjunto de *estados finales* Q_f. La cabeza tiene un conjunto de movimientos $M = \{+1, -1\}$ que puede hacer (moverse una celda hacia la derecha o izquierda respectivamente).

Sobre la cinta de memoria se encuentra escrita la entrada que es una cadena de símbolos de Σ, y el resto de las celdas tiene $*$ (blancos). Un programa S se define como un conjunto de quíntuplas $S \subseteq Q \times \Gamma \times Q \times \Gamma \times M$.

La máquina comienza en el estado inicial q_0 y con la cabeza lecto-escritora posicionada sobre la celda inicial 0 de la cinta. La máquina realiza acciones de acuerdo al programa S definido. La quíntupla $(q_i, s_h, q_j, s_k, m) \in S$ se interpreta como: *si la máquina está en el estado q_i y la cabeza lee s_h entonces escribir s_k, hacer el movimiento m y pasar al estado q_j*. El programa termina cuando no se pueden inferir nuevas acciones para seguir o cuando se alcanza un estado final.

Definición 2.2.1. Una máquina M *resuelve* el problema Π si para toda instancia de Π alcanza un estado final y responde de forma correcta (i.e. termina en un estado final correcto).

Definición 2.2.2. Una *máquina de Turing determinística* (MTD) es una máquina de Turing que tiene un programa S tal que para todo par $(q, s) \in Q \times \Gamma$ existe a lo sumo una quíntupla en S que comienza con ese par.

Definición 2.2.3. La *complejidad temporal* de una MTD M para un problema Π es la cantidad máxima de movimientos que hace la cabeza desde el estado inicial hasta alcanzar un estado final en función del tamaño de la entrada. Es decir

$$T_M(n) = \text{máx}\{m_M(I) \mid I \in D_\Pi \ \wedge \ |I| = n\}$$

donde $m(I)$ es la cantidad de movimientos que hace la cabeza de M para la instancia I.

Definición 2.2.4. Una *máquina de turing no determinística* (MTND) es una máquina de Turing que tiene un programa S tal que para algún par $(q, s) \in Q \times \Gamma$ existe más de una quíntupla en S que comienza con ese par.

Observación. Una MTND es una tabla que mapea un par $(q_i, s_h) \in Q \times \Gamma$ a un conjunto de ternas $(q_j, s_k, m) \in Q \times \Gamma \times M$. Esto admite dos interpretaciones equivalentes:

1. En cada paso se selecciona una de las alternativas posibles.

2. En cada paso se continúa la ejecición en paralelo de todas las alternativas, generando una copia de la MTND por cada una.

Definición 2.2.5. Una MTND *resuelve* un problema de decisión Π si sucede lo siguiente (según la interpretación) para toda instancia del problema:

1. existe una secuencia de alternativas que lleva a un estado de aceptación si y sólo si la respuesta es SI.

2. alguna de las copias se detiene de la MTND se detiene en un estado de aceptación si y sólo si la respuesta es SI.

Esto es equivalente a decir que para toda instancia de Y_Π existe una rama del árbol de ejecución de la MTND que llega a un estado final de aceptación y para toda instancia de N_Π ninguna rama llega a un estado final de aceptación.

Definición 2.2.6. La *complejidad temporal* de una MTND M para un problema Π es el máximo número de pasos que toma como mínimo reconocer una instancia de Y_Π en función de su tamaño. Es decir

$$T_M(n) = \text{máx}\{p_M(I) \mid I \in Y_\Pi \ \wedge \ |I| = n\}$$

donde $p_M(I)$ es la mínima cantidad de pasos que toma llegar a un estado de aceptación para la instancia I en el árbol de ejecución de M.

Definición 2.2.7. Una MTND es *polinomial* para el problema Π si existe una función polinomial $T(n)$ tal que para toda instancia de Y_Π de tamaño n alguna de las ramas termina en un estado de aceptación en a lo sumo $T(n)$ pasos.

2.3. Clases de complejidad

2.3.1. Clases P y NP

Definición 2.3.1. La clase de complejidad P (polinomial) es el conjunto de problemas de decisión para los cuales existe una máquina de Turing determinística de complejidad polinomial que lo resuelve.

Definición 2.3.2. La clase de complejidad *NP* (polinomial no determinístico) es el conjunto de problemas de decisión para los cuales existe una máquina de Turing no determinística de complejidad polinomial que resuelve las instancias con respuesta SI. Equivalentemente, dada una instancia del problema con respuesta SI se puede dar un *certificado* que garantiza que la respuesta es SI y esta garantía puede ser verificada en tiempo polinomial.

Observación. $\mathsf{P} \subseteq \mathsf{NP}$. ¿$\mathsf{P} = \mathsf{NP}$? es un problema abierto.

Proposición 2.3.1. *Un problema de decisión que pertenece a la clase NP puede ser resuelto por un algoritmo determinístico de complejidad exponencial.*

Demostración. Sea Π el problema de decisión. Como $\Pi \in \mathsf{NP}$ existe una MTND que llega a un estado de aceptación en alguna rama de su árbol de ejecución en una cantidad de pasos de a lo sumo $p(n)$, donde $p(n)$ es un polinomio y n es el tamaño de la instancia del problema. Para resolver Π de forma determinística basta recorrer el árbol de ejecución de la MTND hasta llegar a un estado de aceptación, en cuyo caso se responde SI, o hasta alcanzar la cantidad de pasos $p(n)$ en todas las ramas (i.e. llegar a la altura $p(n)$ del árbol), en cuyo caso se responde NO. Como la MTND tiene un alfabeto y un conjunto de estados iniciales finitos de tamaño σ y q respectivamente, en cada paso existen $c = \sigma q$ alternativas posibles. Luego, el algoritmo recorrerá a lo sumo $O(c^{p(n)})$ nodos del árbol de ejecución, es decir, hará a lo sumo $O(c^{p(n)}) = O(2^{p(n)})$ pasos. Por lo tanto, es un algoritmo de complejidad exponencial que resuelve Π de forma determinística. □

Definición 2.3.3. Un *transformación* o *reducción polinomial* de un problema de decisión Π_1 a uno Π_2 es una función de complejidad polinomial $f : D_{\Pi_1} \to D_{\Pi_2}$ que transforma una instancia I_1 de Π_1 en una instancia $f(I_1) = I_2$ de Π_2 tal que

$$I_1 \in Y_{\Pi_1} \iff I_2 \in Y_{\Pi_2}$$

Si existe tal función se dice que Π_1 se *reduce polinomialmente* a Π_2 y se nota $\Pi_1 \leq_p \Pi_2$.

Proposición 2.3.2. *La reducción polinomial es una relación transitiva.*

Demostración. Sean Π_1, Π_2, Π_3 problemas de decisión, f una reducción polinomial de Π_1 a Π_2 y g una reducción polinomial de Π_2 a Π_3. Para toda instancia I de Π_1, $I \in Y_{\Pi_1} \iff f(I) \in Y_{\Pi_2} \iff g(f(I)) \in Y_{\Pi_3}$. Además, la complejidad de aplicar $g \circ f$ a I es polinomial porque es la suma de las complejidades de aplicar f y g, que son ambas polinomiales. Luego $g \circ f$ es una reducción polinomial de Π_1 a Π_3. Probamos entonces que $\Pi_1 \leq_p \Pi_2 \wedge \Pi_2 \leq_p \Pi_3 \implies \Pi_1 \leq_p \Pi_3$, es decir, que la reducción polinomial es transitiva. □

Definición 2.3.4. El *problema complemento* de un problema de decisión Π, notado Π^{c}, es el problema de decisión que responde al complemento de la decisión de Π. Es decir, Π^{c} responde SI si y sólo si Π responde NO.

Definición 2.3.5. La clase de complejidad *Co-NP* es el conjunto de problemas complemento de los problemas de la clase NP. Es decir, es el conjunto de problemas de decisión para los cuales existe una máquina de Turing no determinística de complejidad polinomial que resuelve las instancias con respuesta NO.

Observación. P $\subseteq$ Co-NP.

2.3.2. Clase NP-completo

Definición 2.3.6. Un problema Π es *NP-hard* si todo problema en NP es reducible polinomialmente a él. Es decir, $\Pi' \leq_p \Pi \ \forall \ \Pi' \in NP$.

Definición 2.3.7. La clase de complejidad *NP-completo* es el conjunto de problemas de decisión que están en NP y son NP-hard.

Proposición 2.3.3. *Si un problema* Π *es NP y existe algún problema NP-completo que es reducible polinomialmente a* Π*, entonces* Π *es NP-completo.*

Demostración. Si existe algún Π^* NP-completo tal que $\Pi^* \leq_p \Pi$ entonces para todo $\Pi' \in NP$, $\Pi' \leq_p \Pi^* \leq_p \Pi$ por transitividad. Luego Π es NP-hard y como está en NP es NP-completo. □

Definición 2.3.8. El *problema de satisfacibilidad booleana* (SAT) consiste en decidir para una fórmula booleana si existe alguna asignación de valores de sus variables que la haga verdadera.

Teorema 2.3.4 (Cook-Levin)**.** *El problema de satisfactibilidad booleana es NP-completo.*

Definición 2.3.9. Un problema Π es una *restricción* de un problema Π' si el dominio de Π está incluído en el de Π'. Es decir, $D_\Pi \subset D_{\Pi'}$. En este caso se dice que Π' es una *extensión* de Π.

Observación. Si un problema Π es NP, una resitricción de Π también lo es.

Definición 2.3.10. Dada un fórmula booleana, un *literal* es un variable (literal positivo) o la negación de una variable (literal negativo) y una *cláusula* es una disjunción de literales (o un solo literal). Una fórmula está en *forma normal conjuntiva* si es una conjunción de cláusulas (o una sola cláusula).

Definición 2.3.11. El problema CNFSAT es una restricción de SAT a fórmulas en forma normal conjuntiva.

Proposición 2.3.5. *SAT es reducible polinomialmente a CNFSAT.*

Corolario 2.3.6. *CNFSAT es NP-completo.*

Definición 2.3.12. El problema 3-SAT es una restricción de CNFSAT en la cual cada cláusula dela fórmula tiene exactamente tres literales.

Proposición 2.3.7. *CNFSAT es reducible polinomialmente a 3-SAT.*

Demostración. Como 3-SAT es una restricción de CNFSAT y este es NP, 3-SAT también es NP. Dada una instancia de CNFSAT $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_n$, la reducimos polinomialmente a una instancia $\phi' = \phi'_1 \wedge \phi'_2 \wedge \cdots \wedge \phi'_n$ de 3-SAT donde ϕ'_i es una conjunción de cláusulas de tres literales. Para cada cláusula $C_i = x_1 \vee x_2 \vee \cdots \vee x_k$ separamos en casos según la cantidad k de literales que tiene:

- $k = 1$. Definimos $\phi'_i = x_1 \vee x_1 \vee x_1$.
- $k = 2$. Definimos $\phi'_i = x_1 \vee x_2 \vee x_2$.
- $k = 3$. Definimos $\phi'_i = C_i$.
- $k \geq 4$. Definimos $\psi_j = \neg y_{j-2} \vee x_j \vee y_{j-1}$ y

  $$\begin{aligned}\phi'_i =&(x_1 \vee x_2 \vee y_1) \wedge \psi_3 \wedge \psi_4 \wedge \cdots \wedge \psi_{k-2} \wedge (y_{k-3} \vee x_{k-1} \vee x_k)\\ =&(x_1 \vee x_2 \vee y_1) \wedge \cdots \wedge (\neg y_{j-2} \vee x_j \vee y_{j-1}) \wedge \cdots \wedge (\neg y_{k-3} \vee x_{k-1} \vee x_k)\end{aligned}$$

  Cuando C_i es verdadero, es decir algún $x_j = 1$, podemos hacer ϕ'_i verdadero definiendo $y_1 = y_2 = \cdots = y_{k-3}$ verdadero si $x_1 = x_2 = 0$, falso si $x_{k-1} = x_k = 0$ o cualquiera si no es ninguno de esos dos casos. En cambio si C_i es falso, todos sus literales deben ser falsos y entonces

  $$\phi' = y_1 \wedge (\neg y_1 \vee y_2) \wedge (\neg y_2 \vee y_3) \wedge \cdots \wedge (\neg y_{k-4} \vee y_{k-3}) \wedge \neg y_{k-3}.$$

  Distribuyendo los literales se puede ver que esta fórmula es una contradicción. Porbamos entonces que C_i y ϕ'_i son equisatisfactibles.

Como cada fórmula ϕ'_i es satisfactible si y sólo C_i lo es, ϕ y ϕ' son equisatisfactibles. Luego, dado que el segundo puede obtenerse a partir del primero en tiempo polinomial, ϕ' es una reducción polinomial de ϕ. □

Corolario 2.3.8. *3-SAT es NP-completo.*

Proposición 2.3.9. *El problema de decisión de coloreo mínimo de los nodos de un grafo es NP-completo.*

Demostración.

NP. Tomando como certificado un coloreo de los nodos del grafo podemos verificar que sea válido viendo para cada nodo si su color es distinto al de todos sus vecinos y podemos contar la cantidad de colores para ver si cumple con la cota pedida. Esto se puede hacer en tiempo polinomial en función de la cantidad de nodos y de aristas del grafo.

NP-hard. Proponemos una reducción polinomial de una instancia ϕ de CNF-SAT, que es una fórmula, a una instancia (G, k) del problema de coloreo, donde G es un grafo y k una cota superior de la cantidad de colores. Definimos k igual a dos veces la cantidad de variables de ϕ y G tal que tenga:

- Dos nodos por cada variable de ϕ, uno que corresponde al literal que la afirma y otro al literal que la niega. Todos estos nodos que llamamos V_L son adyacentes entre sí.
- Un nodo por cada cláusula que es adyacente a los literales de V_L que no aparecen en dicha cláusula. Los llamamos V_C.

- Un nodo por cada variable que es adyacente a todas las cláusulas de V_C y a los literales de V_L que corresponden a las demás variables. Los llamamos V_V.

Los nodos de V_L tendrán todos un color distinto en un coloreo de a lo sumo k colores porque todos son adyacentes entre sí y $|V| = k$. Sirven para representar a cada valor posible de cada variable (literal negado o afirmado) con un color distinto. Como ya se usan k colores para V_L, cada cláusula de V_C deberá tener el color de un nodo de V_L no adyacente a él, es decir, que corresponde a un literal que sí está en la cláusula. Esto representa la elección del literal que hará verdadera a la cláusula. Los nodos de V_V pueden tener sólo los colores de los literales de V_L de la variable a la que corresponden. Representan los valores opuestos a asignar a las variables de ϕ ya que si alguna de las cláusulas elige un literal de la variable (i.e. tiene el color de ese literal) el nodo correspondiente sólo podrá tener el color del otro literal. Luego, si existe un k-coloreo de G, para cada nodo de V_V podemos asignarle a la variable correspondiente de ϕ el valor opuesto al que indica el literal de V_L de mismo color.

Si ϕ es satisfactible entonces podemos tomar de cada cláusula el literal que la hace verdadera y colorear el nodo correspondiente en V_C con el color de dicho literal en V_L. Los nodos de V_V que corresponden a los literales tomados quedarán con un solo color posible, mientras que los demás pueden tener cualquiera de los dos que disponen. Si hubieran conflictos en este paso sería porque hay un nodo $v \in V_V$ que es adyacente a una cláusula C_i de V_C que tiene el color de x y otra C_j que tiene el color de $\neg x$, donde x es la variable correspondiente a v. Pero esto significaría que $C_i = \neg C_j$, por lo que ϕ no sería satisfactible.

Como la definición del grafo puede hacerse en tiempo polinomial con respecto a la cantidad de variables de ϕ, la reducción es polinomial. □

Proposición 2.3.10. *El problema de decisión de conjunto independiente máximo en un grafo es NP-completo.*

Demostración.

NP. Tomando como certificado los nodos de un conjunto independiente podemos verificar que sean nodos del grafo sin repeticiones, que no sean adyacentes entre sí y que la cantidad de nodos es mayor o igual a la cota en tiempo polinomial.

NP-hard. Proponemos una reducción polinomial de 3-SAT al problema de conjunto independiente mínimo, cuyas instancias son un par (G, k) donde G es un grafo y k una cota inferior del tamaño del conjunto independiente. Sea $\phi = C_1 \wedge C_2 \wedge \cdots \wedge C_s$ una instancia de 3-SAT, donde cada C_i es una cláusula con tres literales, y sean $x_1, x_2, \ldots, x_t$ las variables de ϕ. Definimos k como la cantidad de cláusulas s y G de la siguiente manera:

1. Para cada cláusula C_i ponemos por cada una de sus variables x_j un nodo v_{ij} si tiene el literal x_j o $v_{ij}^{\neg}$ si tiene el literal $\neg x_j$. Hacemos adyacentes a todos estos nodos de forma que haya un grafo K_r por cláusula donde r es la cantidad de literales distintos que tiene.

2. Para cada variable x_k agregamos una arista entre los nodos que representan su afirmación y su negación. Es decir, ponemos la arista $(v_{ik}, v_{jk}^{\neg})$ para todo i, j.

Un conjunto independiente en G tendrá a lo sumo un nodo por cláusula, ya que no podrá tener más de un nodo por cada K_r. Esto es equivalente a elegir un literal de la cláusula que la haga verdadera. Además, dos literales de cláusulas distintas pueden estar en el conjunto independiente sólo si uno no es la negación del otro. Esto evita que se elijan literales que produzcan una contradicción en ϕ. Luego, si existe un conjunto independiente de al menos $k = s$ nodos este determinará una asignación de valores de las variables de ϕ que la hacen verdadera.

Por otro lado, si ϕ no es satisfactible siempre habrá una contradicción entre dos cláusulas. Luego, por lo explicado antes, no se podrá elegir un nodo de los K_r de estas cláusulas y el conjunto independiente tendrá menos de k nodos. □