Resumen Teórico Matemática Discreta II

Ferré Valderrama, Eduardo

22 de junio de 2024

Matemática Discreta II

Índice

1. ¿Cuál es la complejidad de Edmons-Karp?

La complejidad del algoritmo de Edmons-Karp es $O(nm^2)$

Demostración. Para facilitar la prueba, supondremos que sí en el network está el lado $\overrightarrow{xy}$, entonces no está el lado $\overrightarrow{yx}$.

Si $f_0, f_1, f_2, \ldots$ son los flujos parciales producidos al correr EK, entonces queremos ver que hay una cantidad finita de ellos y dar una cota para ese número. Como la búsqueda y construcción de cada camino aumentante se hace con **BFS**, cada incremento del flujo tiene complejidad $O(m)$.

Así, que si probamos que sólo puede haber $O(nm)$ flujos aumentantes, tendremos la prueba de que la complejidad de EK es $O(nm^2)$.

Primero, decimos que un lado se vuelve crítico al pasar de f_k a f_{k+1} si se **satura** o **vacía**.

En cada construcción de un camino aumentante al menos un lado se vuelve crítico, el problema es que un lado puede volverse crítico muchas veces. Veamos cuantas veces puede pasar.

Supongamos que $\overrightarrow{xy}$ se vuelve crítico en el paso k, y luego en el paso r, con $r > k$. Tenemos que analizar dos casos: se vuelve crítico en el paso k

porque se saturó (llamemosle **Caso 1**), o se vuelve crítico en el paso k porque se vació, (a este caso lo llamamos **Caso 2**).

- **Caso 1. En el que se satura (forward)**: Como se saturó en el paso k entonces:
  - Para construir f_{k+1} se debe usar un f_k camino aumentante de la forma $s \dots \overrightarrow{xy} \dots t$.
  - Como estamos usando EK ese camino es de longitud mínima, por lo tanto:

$$d_k(y) = d_k(x) + 1. \textbf{ (1)}$$

  Para que se vuelva crítico en el paso r, debe pasar una de estas dos cosas:

  - Se vuelve crítico en el paso r porque se vacía.
  - Se vuelve crítico en el paso r porque vuelve a saturarse.

  Para que ocurra lo segundo, debe haberse vaciado antes aunque sea un poco, sino es imposible que vuelva a saturarse.

  Entonces en cualquiera de estos dos casos, deducimos que existe l tal que $k < l \leq r$ tal que el flujo en $\overrightarrow{xy}$ disminuye al pasar de f_l a f_{l+1} ya sea vaciándose completamente o un poco.

  Esto implica que para construir f_{l+1} se usa un f_l-camino aumentante de la forma $s \dots \overleftarrow{yx} \dots t$. Como estamos usando EK, ese camino es de longitud mínima, por lo tanto:

$$d_l(x) = d_l(y) + 1. \textbf{ (2)}$$

  Entonces:

$$\begin{aligned} d_l(t) &= d_l(x) + b_l(x). \\ &= d_l(y) + 1 + b_l(x) \quad \text{Por } \textbf{(2)} \\ &\geq d_k(y) + 1 + b_k(x) \quad \text{Porque las distancias no disminuyen} \\ &= d_k(x) + 1 + 1 + b_k(x) \quad \text{Por } \textbf{(1)} \\ &= d_k(t) + 2 \end{aligned}$$

  Este fue el análisis de si el lado $\overrightarrow{xy}$ se volvía crítico en el paso k porque se saturaba. Ahora vamos con el **Caso 2.**

- **Caso 2. En el que se vacía (backward)**: Como se vacía, existe un camino (de longitud mínima) de la forma $s \dots \overleftarrow{yx} \dots t$ que se usa para pasar de f_k a f_{k+1}. Por lo tanto:

$$d_k(x) = d_k(y) + 1. \textbf{ (3)}$$

Para que se vuelva crítico en el paso r, debe o bien volverse a vaciar, o bien saturarse, en el segundo de esos casos estamos obviamente mandando más flujo en el paso r, pero en el primero si estamos diciendo que debe vaciarse de vuelta, entonces antes debe haberse llenado aunque sea un poco.

Así que en cualquier caso debe haber un l tal que $k < l \leq r$ para el cual se manda flujo a través de él. Entonces, existe un camino (de longitud mínima) de la forma $s \dots \overrightarrow{xy} \dots t$ que se usa para pasar de f_l a f_{l+1}. Por lo tanto:

$$d_l(y) = d_l(x) + 1. \textbf{ (4)}$$

Notemos que **(3)** y **(4)** son análogos a **(1)** y **(2)**, solo que con x e y intercambiados. Entonces podemos deducir que $d_l(t) \geq d_k(t) + 2$ de la misma forma que en el **Caso 1**, simplemente intercambiando x e y en la prueba. Así que también en este caso deducimos que $d_r \geq d_l(t) \geq d_k(t) + 2$.

Entonces tanto como para el **Caso 1** como para el **Caso 2**, luego de que un lado se vuelva crítico, para que se vuelva crítico otra vez, la distancia entre s y t debe aumentar al menos en 2.

Como la distancia entre s y t puede ir desde mínimo de 1 hasta un máximo de $n - 1$, entonces un lado puede volverse crítico un máximo de $O(\frac{n}{2}) = O(n)$ **veces**.

Como cada camino aumentante que se usa en EK tiene al menos un lado que se vuelve crítico, entonces el total de flujos intermedios está acotado por $O(nm)$, puesto que hay m lados y cada uno se puede volver crítico a lo sumo $O(n)$ veces.

Finalmente como cada construcción de un flujo tiene complejidad $O(m)$, entonces concluimos que la complejidad total de EK es

$$O(m) * O(nm) = O(nm^2) \blacksquare$$

2. Probar que, si dados vértices x, z y flujo f definimos la distancia entre x y z relativa a f como la longitud del menor f-camino aumentante entre x y z, si es que existe tal camino, o infinito si no existe o 0 si $x = z$, denotándola por $d_f(x, z)$, y definimos $d_k(x) = d_{f_k}(s, x)$, donde f_k es el k-ésimo flujo en una corrida de Edmonds-Karp, entonces $d_k(x) \leq d_{k+1}(x)$.

Demostración. Vamos a demostrar que $d_k(x) \leq d_{k+1}(x)$.

Sea $A = \{y : d_{k+1}(y) < d_k(y)\}$. Queremos ver que $A = \emptyset$, supongamos que no.

Sea $x \in A$ tal que

$$d_{k+1}(x) = min\{d_{k+1}(y) : y \in A\} \tag{1}$$

Observemos que $x \in A$, entonces $d_{k+1}(x) < d_k(x) \leq \infty \Rightarrow d_{k+1} < \infty$. Entonces existe un f_{k+1}-camino aumentante entre s y x.

Observación. $x \neq s$, pues $s \notin A$, pues $d_k(s) = d_{k+1}(s) = 0$.

Sea entonces $p_{k+1} := s \dots x$ un f_{k+1}-camino aumentante de longitud mínima entre s y x, es decir, de longitud $d_{k+1}(x)$. Como $x \neq s$ en ese camino existe un vertice inmediatamente anterior a x, llamemosle z. Entonces, $p_{k+1} = s \dots zx$.
Como p_{k+1} es de longitud mínima entre s y x, entonces es de longitud mínima entre s y cualquier otro vertice intermedio como pej z. Por lo tanto:

$$d_{k+1}(z) = d_{k+1}(x) - 1 \tag{2}$$

Como $p_{k+1} = s \dots zx$ es un f_{k+1}-camino aumentante, entonces existe el lado $\overrightarrow{zx}$ o existe el lado $\overrightarrow{xz}$. Analicemos los dos casos.

- **Caso 1.** Si $\overrightarrow{zx}$ es un lado, tenemos que $d_{k+1}(z) < d_{k+1}(x)$. Pero

  $$z \notin A \implies d_k(z) \leq d_{k+1}(z) < d_{k+1}(x) < \infty$$

  Como $d_k(z) < \infty$, existe un f_k-camino aumentante entre s y z. Lo denominamos p_k. Como $\overrightarrow{zx}$ es un lado, podría, en principio, agregar x al final de p_k, obteniendo un f_k-c.a. entre s y x que pasa por z. Pero entonces tendríamos $d_k(x) \leq d_k(z) + 1 \leq d_{k+1}(z) + 1 \leq d_{k+1}(x)$. Pero esto implica $x \notin A$.

Por lo tanto, no puedo agregar x al final de p_k. Entonces, o bien el lado $\overrightarrow{zx}$ está saturado en el paso k, o bien $f_k(\overrightarrow{zx}) = c(\overrightarrow{zx})$. Pero p_{k+1} era aumentante; es decir, sus lados no pueden estar saturados; lo cual implica que $f_{k+1}(\overrightarrow{zx}) < c(\overrightarrow{zx})$ en el paso $k+1$. Luego, para pasar de f_k a f_{k+1}, se usó el lado $\overrightarrow{xz}$ y se usó backwards en el camino

$$\tilde{p}_k : s \dots \tilde{z} \dots t$$

Como este camino se usa en Edmonds-Karp, es de longitud mínima. Por lo tanto,

$$d_k(z) = d_k(x) + 1$$

Completar, falta determinar el absurdo.

- **Caso 2.** Si $\overrightarrow{zx}$ es un lado, tenemos que p_{k+1} es de la forma $s \dots zx$. Como antes,

$$d_{k+1}(z) < d_{k+1}(x) \implies z \in A \implies d_k(z) \leq d_{k+1}(z)$$

lo cual a su vez implica que existe un f_k-camino aumentante entre s y z:

$$p_k : s \dots z$$

Podríamos agregar a x en el final de este camino usando $\overrightarrow{xz}$ backward: llegaríamos a la misma contradicción que antes, en que no podríamos agregar x. En este caso, si no lo podemos agregar, es porque $f_k(\overrightarrow{xz}) = 0$. Pero p_{k+1} es f_{k+1}-aumentante, lo cual implica que $f_{k+1}(\overrightarrow{xz}) > 0$. Luego debimos haber usado $\overrightarrow{xz}$ en modo forward al pasar de f_k a f_{k+1}. Por lo tanto existe un f-camino aumentante de la forma:

$$\tilde{p}_k : s \dots \overrightarrow{xz} \dots t$$

Y luego como usamos Edmonds-Karp, este camino aumentante es de longitud mínima.
Luego $d_k(z) = d_k(x) + 1$.
Completar, falta determinar el absurdo.

Finalmente concluimos $A = \emptyset$, es decir, que $d_k(x) \leq d_{k+1}(x)$. ■

3. Probar que, si dados vértices x, z y flujo f definimos la distancia entre x y z relativa a f como la longitud del menor f-camino aumentante entre x y z, si es que existe tal camino, o infinito si no existe o 0 si $x = z$, denotándola por $d_f(x, z)$, y definimos $b_k(x) = b_{f_k}(x, t)$, donde f_k es el k-ésimo flujo en una corrida de Edmonds-Karp, entonces $b_k(x) \leq b_{k+1}(x)$.

Demostración. Se evalúa en diciembre de 2024.

4. ¿Cual es la complejidad del algoritmo de Dinic? Probarla en ambas versiones: Dinitz original y Dinic-Even. (no hace falta probar que la distancia en networks auxiliares sucesivos aumenta).

La complejidad de Dinitz, en su versión original como occidental, es $O(n^2m)$.

Demostración. Como el nivel de t en los networks auxiliares aumenta, solo puede haber $O(n)$ networks auxiliares. ∴La complejidad de Dinitz es igual a la complejidad de hallar un flujo bloqueante en un network auxiliar, más la complejidad de construir un network auxiliar, por la cantidad de networks auxiliares—que es $O(n)$. Porque usamos DFS, la complejidad de construir un network auxiliar es $O(m)$. Entonces basta probar que la complejidad de hallar un flujo bloqueante es $O(nm)$, porque tendremos

$$O(n) \cdot (O(m) + O(nm)) = O(n^2m)$$

- **Versión Original:**

  En la versión original del algoritmo, el n.a. es tal que no hay vértices sin lado de salida. Esto implica que DFS siempre llega a t sin hacer backtracking. Entonces, DFS no es $O(m)$ sino O(cantidad de niveles) que podemos simplificar como $O(n)$. Como cada camino borra al menos un lado del n.a. hay $O(m)$ caminos. Luego la complejidad de hallar los caminos y aumentar el flujo por ellos es $O(nm)$. Pero debemos calcular el costo de que el n.a. satisfaga la propiedad de que no hay vértices sin lados de salida.

  Para esto, luego de cada camino hay que hacer algo para mantener esa propiedad como invariante. Dinitz llamó a la operación que logra esto **"podar"**. La operación consiste en recorrer los vértices desde los niveles más altos a los más bajos, chequea si el vértice tiene lados de salida; si tiene no hace nada, si no lo tiene lo borra.

  Chequear si el lado tiene lados de salida es $O(1)$; hay $O(n)$ vértices, y hay un "podar" por cada camino y hay $O(m)$ caminos. Esto entonces también es $O(nm)$.

  Nos queda analizar la complejidad de borrar el vértice y sus lados; esto se hace a lo sumo una vez por vértice, y una vez que podamos un vértice es probable que podar los siguientes requiera una complejidad menor (porque estamos quitando lados). Esto nos dice que podemos pensar en la complejidad promedio del problema, y no en la complejidad límite (es decir como cota).

  Borrar un vértice x y sus lados es $O(d(x))$; sobre todos los vértices, esto da $\sum O(d(x)) = O(m)$ (por la propiedad del apretón de manos). Luego la complejidad de hallar un flujo bloqueante es:

$$O(nm) + O(nm) + O(m) = O(nm) \blacksquare$$

- **Versión Dinic-Even:**
  Ahora consideremos la versión occidental. Lo más cómodo es dar el pseudocódigo. Estamos hablando del network auxiliar y el problema de hallar un flujo bloqueante en él.

  Para hallar flujo bloqueante g en un n.a:

Algorithm 1 Algoritmo

```
1: g := 0
2: bool flag := true
3: while flag do
4:     type path := [s]
5:     int x := s
6:     while x ≠ t do
7:         if Γ⁺(x) ≠ 0 then
8:             tomar y ∈ Γ⁺(x)
9:             agregar y a path
10:            x := y  ▷ Esta línea y la anterior son la parte (A) de avanzar
11:        else
12:            if x ≠ s then
13:                z := elemento anterior a x en path        ▷ (R) Retroceder
14:                borro x de path
15:                borro \overrightarrow{zx} de la n.a.
16:                x := z
17:            else
18:                flag := 0
19:            end if
20:        end if
21:    end while
22:    if x = t then
23:        aumentar flujo g a lo largo de path              ▷ (I) Incrementar
24:        borrar lados saturados
25:    end if
26: end while
27: return g
```

Entonces, una corrida de Dinitz occidental es una palabra $AAAIAAARARRAA\ldots$ Miraremos subpalabras de la forma $A\ldots AX$ con $X \in \{I, R\}$.

- Cada A es $O(1)$;
- Cada R es $O(1)$;
- Cada I es recorrer un camino dos veces (una para incrementar el flujo, otra para borrar los lados), y resulta ser $O(n)$.

La cantidad de A's en $A\ldots AX$ se calcula como sigue. Cada avanzar, mueve el pivote x de un nivel al siguiente. $\therefore$ Hay $O(n)$ letras A.

- Cada avanzar, mueve el pivote x de un nivel al siguiente. $\therefore$ Hay $O(n)$ letras A.
- $\therefore O(A\ldots AR) = O(n) + O(1) + O(n)$
- y $O(A\ldots AI) = O(n)(\#A) + O(n)(\#I) = O(n)$

Donde $\#Z$ es la cantidad de veces que ocurre la letra Z en la palabra en

cuestión. En resumen, $O(A \dots AX) = O(n)$. Y la cantidad de palabras $A \dots AX$ en una corrida se calcula así:

- Cada R borra un lado,
- y cada I borra *al menos* un lado;
- luego hay $O(m)$ palabras $A \dots AX$.

Luego la complejidad total es $O(nm)$. ■

5. ¿Cual es la complejidad del algoritmo de Wave? Probarla.

La complejidad de Wave es $O(n^3)$.

Demostración. Como es un algoritmo de tipo Dinitz, la distancia en n.a aumenta. Por lo tanto hay $O(n)$ networks auxiliares. Basta probar que la complejidad de hallar un flujo bloqueante en Wave es $O(n^2)$.
Dividiremos la complejidad en varias partes.

- Los **"Los balanceos hacia adelante"** de los vertices.
  Cuando un vertice x manda flujo a z, puede pasar que $\overrightarrow{xz}$ se sature o no. Sea entonces:
  - $S =$ Todos los pasos donde se satura un lado.
  - $P =$ Todos los pasos donde no se satura un lado.
- Los **"Los balanceos hacia atrás"** el razonamiento es análogo.
  Cuando devolvemos flujo, podemos devolverlo todo o solo un poco. Sea entonces:
  - $V =$ La parte donde se vacían los lados.
  - $Q =$ La parte donde se devuelve flujo parcialmente.

Entonces la complejidad de hallar un flujo bloqueante en Wave es

$$S + P + V + Q$$

Analicemos cada una en detalle.

- **Análisis de S**
  Como habíamos mencionado, cuando un vertice x manda flujo a z, puede pasar que $\overrightarrow{xz}$ se sature o no. Supongamos que $\overrightarrow{xz}$ se saturó. La pregunta es si puede volver a saturarse en alguna ola hacia adelante. Para volver a saturarse, debe primero des-saturarse.

  Por lo tanto, z debe devolverle flujo a $x \Rightarrow z$ debe estar "bloqueado".

Pero, en las olas hacia adelante, cuando un vértice chequea sus vecinos, solo se manda flujo a vertices no bloqueados. Entonces si z le devuelve flujo a x, x no le puede volver a mandar flujo a z, por lo tanto:

- $\overrightarrow{xz}$ no se puede volver a saturar otra vez.
- Entonces cada lado se satura a lo sumo una vez.

Entonces concluimos que la complejidad de S es $O(m)$.

- **Análisis de V**
  La pregunta es cuantas veces puede vaciarse un lado. La respuesta es muy similar al caso de S. Si x devuelve flujo a un vertice u y se vacía $\overrightarrow{ux}$, entonces x debe estar bloqueado, por lo tanto:

  - u nunca puede volver a enviar flujo.
  - Por lo tanto, $\overrightarrow{ux}$ no puede volver a vaciarse (porque no se puede volver a enviar nada).
  - Entonces cada lado se vacía a lo sumo una vez.

  Entonces concluimos que la complejidad de V es $O(m)$.

- **Análisis de P y Q**
  Cuando intentamos balancear hacia adelante desde un vértice, todos los lados excepto quizas el ultimo se saturan. Es decir, en cada vértice de cada ola hacia adelante, hay a lo sumo un lado que no se satura, y çuenta"para la complejidad de P.

  - Enviar flujo por ese lado es $O(1)$.
  - Luego la complejidad de P es $O(n)*$(#olas hacia adelante).

  Similar en Q.

  Al balancear hacia atras, en cada vértice se vacían todos los lados excepto a lo sumo uno.

  - Luego la complejidad de Q es $O(n)*$(#olas hacia atrás).

  Luego tenemos que:

  $$\#\text{olas hacia adelante} = \#\text{olas hacia atrás}$$

  Pues a cada ola hacia adelante corresponde una ola hacia atras. Ahora bien tenemos que calcular la cantidad de olas.

  En cada ola hacia adelante excepto la ultima (cuando se logra el balanceo), algun vertice queda desbalanceado. Pero entonces la ola debio de bloquearlo. Por lo tanto, en cada ola hacia adelante excepto tal vez la ultima, se bloquea al menos un vertice. Los vertices nunca se desbloquean.

- Luego hay $O(n)$ olas.
- Entonces la complejidad de P es igual a la de Q
- Esto es $O(n) * O(n) = O(n^2)$

Finalmente tenemos que la complejidad de hallar un flujo bloqueante en Wave es:

$$\begin{aligned} S + P + V + Q &= O(m) + O(n^2) + O(m) + O(n^2) \\ &= O(n^2) + O(n^2) + O(m) + O(m) \\ &= O(n^2) + O(m) \quad \text{y} \quad m = O(n^2) \\ &= O(n^2) \blacksquare \end{aligned}$$

6. Probar que la distancia en networks auxiliares sucesivos aumenta.

Se evalúa en diciembre de 2024.

7. Probar que el valor de todo flujo es menor o igual que la capacidad de todo corte y que si f es un flujo, entonces f es maximal si y solo si existe un corte S tal que $v(f) = cap(S)$. (y en este caso, S es minimal) (puede usar sin necesidad de probarlo que si f es flujo y S es corte entonces $v(f) = f(S, \overline{S}) - f(\overline{S}, S)$)

Demostración.

1. Probaremos que $f \leq Cap(S)$ para un corte S y un flujo f arbitrarios. Por lema sabemos que:

$$v(f) = f(S, \overline{S}) - f(\overline{S}, S)$$

Observemos que el término $f(\overline{S}, S)$ es de la forma $\sum f(\overrightarrow{xy})$. Por definición, cada término en esa sumatoria es mayor o igual a cero, y por lo tanto la sumatoria es positiva. Entonces tenemos que $-f(\overline{S}, S) \leq 0$. Luego

$$v(f) = f(S, \overline{S}) - f(\overline{S}, S) \leq f(S, \overline{S})$$

El primer término:$f(S,\overline{S})$, también es una sumatoria de términos positivos y es por lo tanto mayor a cero, pero cada término es a su vez menor a la capacidad de cada lado, es decir $f(S,\overline{S}) \leq c(S,\overline{S}) = Cap(S)$. Por lo tanto:

$$v(f) \leq f(S,\overline{S}) \leq Cap(S) \blacksquare$$

2. Ahora probemos que f es maximal $\iff$ existe un corte minimal S tal que $v(f) = Cap(S)$.

   - ($\Leftarrow$) Sea ahora f un flujo y S un corte con $v(f) = Cap(S)$. Sea g cualquier otro flujo. Entonces, por la propiedad recien demostrada, $v(g) \leq Cap(S)$. Pues $Cap(S) = v(f)$, tenemos $v(g) \leq v(f)$. Por lo tanto f es maximal. Como detalle si T es un corte, $Cap(T) \geq v(f) = Cap(S)$, lo cual implica que S es minimal.
   - ($\Rightarrow$) Asumimos que f es maximal. Probaremos que existe un corte S con $v(f) = Cap(S)$. Para esto, debemos construir S a partir f. Definiremos

     $$S = \{s\} \cup \{x \in V : \exists f\text{-camino aumentante entre } s \text{ y } x\}$$

     Que S es un corte se sigue por contradicción. Si S no es corte, debe contener a t. Luego existe un f-camino aumentante desde s a t. Esto implica que puedo aumentar el flujo f, lo cual contradice que f es maximal.
     Sabiendo que S es un corte, tenemos

     $$v(f) = f(S,\overline{S}) - f(\overline{S},S) \tag{1}$$

     Consideremos el primer término en la resta de (1):

     $$f(S,\overline{S}) = \sum_{\substack{x\in S \\ z\notin S \\ \overrightarrow{xz}\in E}} f(\overrightarrow{xz})$$

     Sea $\overrightarrow{xz}$ un par dentro del rango de la suma de arriba. Pues $x \in S$, existe un f-camino aumentante de s a x. Pues $z \notin S$, no existe un f-camino aumentante de s a z. Pero el lado $\overrightarrow{xz}$ sí existe. Así que $s \dots xz$ podría ser un f-camino aumentante; como tal camino no existe por hipótesis, no es aumentante. Esto implica que $f(\overrightarrow{xz}) = c(\overrightarrow{xz})$. La conclusión es que $f(\overrightarrow{xz}) = c(\overrightarrow{xz})$ para todo $x \in S, z \notin S, \overrightarrow{xz} \in E$. Entonces

     $$f(S,\overline{S}) = \sum_{\substack{x\in S \\ z\notin S \\ \overrightarrow{xz}\in E}} f(\overrightarrow{xz}) = \sum_{\substack{x\in S \\ z\notin S \\ \overrightarrow{xz}\in E}} c(\overrightarrow{xz}) = Cap(S)$$

Ahora consideremos el segundo término de la ecuación (1).

$$f(\overline{S}, S) = \sum_{\substack{w \notin S \\ x \in S \\ \overrightarrow{wx} \in E}} f(\overrightarrow{wx})$$

Sea $\overrightarrow{wx}$ un par arbitrario en el rango de la suma. Como antes, $x \in S \Rightarrow$ que hay un camino aumentante de s a x; pero no existe camino aumentante entre s y w; pero $\overrightarrow{wx}$ es un lado. Es decir, $s \dots \overleftarrow{wx}$ podría ser un f-camino aumentante; y como no lo es sucede que $f(\overrightarrow{wx}) = 0$. Luego

$$f(\overline{S}, S) = \sum_{\substack{w \notin S \\ x \in S \\ \overrightarrow{wx} \in E}} f(\overrightarrow{wx}) = 0$$

Luego $v(f) = Cap(S) - 0 = Cap(S)$. ■

8. Probar que si G es conexo no regular, entonces $\chi(G) \leq \Delta(G)$.

Sea x tal que $d(x) = \delta(G)$. Corro BFS empezando por x y guardo el orden inverso en que los vértices se visitaron. Ahora voy a correr greedy en este orden que acabo de guardar. Notar que en BFS todo vértice es incluido por un vértice que ya ha sido visitado, entonces en el orden inverso todo vértice tiene al menos un vecino por delante (excepto el x). Esto significa que para cada vértice y, en el peor caso va a tener $d(y) - 1 < \Delta$ vecinos coloreados todos con un color distinto, entonces voy a poder elegir un color en $\{1, \cdots, \Delta\}$. Finalmente, cuando llego a x, como $d(x) = \delta < \Delta$, podré elegir algún color de $\{1, \cdots, \Delta\}$ que no está usado por ningún vecino de x para colorearlo. De esta forma tengo un coloreo propio de G que usa (a lo sumo) Δ colores, entonces $\chi(G) \leq \Delta(G)$.

9. Probar que 2-COLOR es polinomial

Demostración. Problema: "k-color":

Dado G ¿Es $\chi(G) \leq k$?, en particular $k = 2$.

- 1-color es trivial (no es polinomial).
- 2-color es polinomial (hay algún algoritmo que corre en tiempo polinomial en un rango de la entrada).
- Entonces la idea de la demostración es correr un algoritmo en tiempo polinomial, en particular solo nos basta con correrlo en grafos conexos

Tomamos un $x \in V$, corremos BFS empezando en x. Si:

$$\begin{aligned} N(z) &= \text{nivel } z \text{ en el árbol BFS} \\ &= \text{distancia entre } z \text{ y } x \quad \text{en el árbol BFS} \\ &= \text{distancia entre } z \text{ y } x \quad \text{en } G \end{aligned}$$

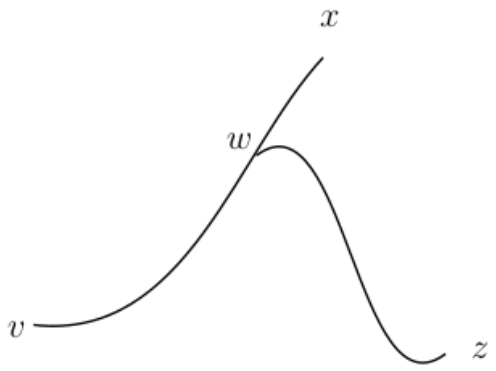
Sea:

- $C(z) = (N(z) \mod 2)$
- **If** (C es propio, return si, es 2-colorable).
- **Else** (return no, no es 2-colorable).

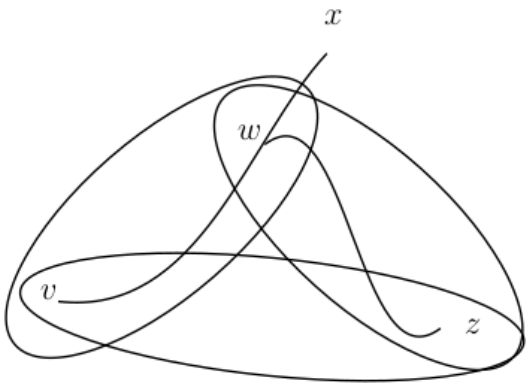
El algoritmo es polinomial porque BFS es $O(m)$ y chequear que es propio es $O(m)$, lo que hay que probar es si es correcto. Entonces supongamos que la respuesta es **"no es 2-colorable"**.

$$\Rightarrow \exists v, z : c(v) = c(z) \wedge vz \in E$$

Entonces $d(x, v) = d(x, z) \mod 2$
Tomamos un camino entre x y v en BFS y un camino entre x y z en BFS y sea w el único vértice en común (como lo muestra la siguiente figura).



Miramos el ciclo en G: $w \ldots \underbrace{vz}_{\text{cruzo a } z} \ldots \underbrace{w}_{\text{vuelvo a } x}$ (como lo muestra la siguiente imagen).



Calculamos la longitud de este ciclo:

$$\begin{aligned}
\text{longitud} &= 1 + d(v,w) + d(z,w) \\
\text{longitud mod } 2 &= (1 + d(v,w) + \ldots + d(z,w)) \bmod 2 \\
&= (1 + d(v,w) + d(z,w) + 2d(x,w)) \bmod 2 \\
&= (1 + d(x,v) + d(x,z)) \bmod 2 = (1 + \overbrace{c(x) + c(z)}^{=0}) \bmod 2 \\
&= 1
\end{aligned}$$

Es un ciclo impar, entonces no se puede colorear con 2 colores $\Rightarrow \chi(G) \geq 3$.■

10. Enunciar y probar el Teorema de Hall.

Sea $G = (V, E)$ bipartito con partes X e Y. Existe un matching completo de X a Y si y solo si

$$|\Gamma(S)| \geq |S| \quad \forall S \subseteq X$$

Notemos que un matching M de X a Y es completo si y solo si $|M| = |X|$.
Demostración.
(⇒)
Sea M un matching completo de X a Y, este matching nos induce a una función inyectiva

$$f : X \to Y$$

tal que

$$f(x) \in \Gamma(x)$$

Como f es inyectiva tenemos que:

$$|f(S)| = |S| \quad \forall S \subseteq X$$

Finalmente

$$f(S) \subseteq \Gamma(S) \Rightarrow |S| \leq |\Gamma(S)| \quad \forall S \subseteq X$$

($\Leftarrow$)
Supongamos que se cumple la condición de Hall, es decir:

$$|\Gamma(S)| \geq |S| \quad \forall S \subseteq X$$

Y también asumamos que al correr el algoritmo para hallar un matching maximal, llegamos a un matching maximal M tal que $|E(M)| < |X|$. Es decir, un matching incompleto en X (podría ser en Y). Construiremos un $S \subseteq X$ que viola la condición de Hall, es decir tal que

$$|\Gamma(S)| < |S|$$

llegando a un absurdo.
Corramos el algoritmo de extensión de matching sobre el ultimo matching M. Como esta matching no cubre a X, existen filas sin matchear, más otras filas etiquetadas. Entonces sean:

$$\begin{aligned} S &= \{\text{filas etiquetadas}\} \\ T &= \{\text{columnas etiquetadas}\} \\ S_0 &= \{\text{filas etiquetadas con } *\} \\ T_1 &= \{\text{columnas etiquetadas por } S_0\} \\ S_1 &= \{\text{filas etiquetadas por } T_1\} \end{aligned}$$

Y en general:

$$\begin{aligned} T_{i+1} &= \{\text{columnas etiquetadas por } S_i\} \\ S_i &= \{\text{filas etiquetadas por } T_i\}. \end{aligned}$$

Notemos que como M no es completo, tenemos algunas filas sin matchear, es decir que

$$S_0 \neq \emptyset \tag{0}$$

Las sucesiones son finitas y $S = S_0 \cup S_1 \cup \ldots$ con uniones disjuntas, y $T = T_1 \cup T_2 \cup \ldots$ con uniones disjuntas.
Cuando revisamos una columna pueden pasar dos cosas:

- La columna está libre, entonces la matcheamos y extendemos el matching. Esto no pasa porque M es maximal.
- La columna está matcheada con una fila, entonces etiquetamos únicamente esa fila.

Por lo tanto, el algoritmo nunca se detiene cuando pasa de algún T_i a S_i. Es decir que el algoritmo se detiene solamente al pasar de un S_k a un $T_{k+1} = \emptyset$ Observemos que, como cada columna etiqueta a la fila con la cual está matcheada, cada columna etiquetada etiqueta a una sola fila, entonces tenemos que:

$$|T_i| = |S_i| \tag{1}$$

Porque el T_i crea al S_i
Y tambíen
$S = S_0 \cup S_1 \cup \dots S_k$ con uniones disjuntas, y $T = T_1 \cup T_2 \cup \dots T_k$ con uniones disjuntas.
Entonces finalmente tenemos que:

$$\begin{aligned} |S| &= |S_0| + |S_1| + \cdots + |S_k| \\ &= |S_0| + |T_1| + \cdots + |T_k| \quad \text{por (1)} \\ &= |S_0| + |T| > |T| \quad \text{por (0)} \end{aligned}$$

Por lo tanto $|S| > |T|$. Ahora demostremos que $T = \Gamma(S)$.

- $T \subseteq \Gamma(S)$: sea $y \in T$, y tuvo que ser etiquetado por una fila de S, y como cada fila etiqueta a sus columnas vecinas es claro que $y \in \Gamma(S)$.
- $\Gamma(S) \subseteq T$: Supongamos que existe un $y \in \Gamma(S)$ que no está en T. Existe un $x \in S$ que es vecino de y. Pero cuando revisamos x, habríamos visto que y era vecino suyo, y por lo tanto lo habríamos etiquetado. Absurdo pues habíamos supueto $y \notin T$, luego, $\Gamma(S) \subseteq T$.

Finalmente, construimos un $S \subseteq X$ tal que no se cumple la condición de Hall. Lo que es un absurdo pues habiamos supuesto que era cierta. Entonces este absurdo viene de suponer que el matching maximal no es completo.■

11. Enunciar y probar el teorema del matrimonio de Koenig

Todo grafo bipartito regular tiene matching perfecto.

Demostración. Dado $W \subseteq V$, definimos

$$E_W = \{wu : w \in W\}$$

Sean X, Y las partes de G. Sea $S \subseteq X$. Sea $l \in E_S$. Se sigue que

$$\exists x \in S, y \in Y : l = xy = yx$$

Es decir que $y \in \Gamma(x)$. Pero $x \in S$. Entonces $y \in \Gamma(S)$. Entonces $l \in E_{\Gamma(S)}$.
$\therefore E_S \subseteq E_{\Gamma(S)}$. Por lo tanto

$$|E_S| \leq |E_{\Gamma(S)}| \tag{1}$$

Calculemos en general $|E_W|$ cuando $W \subseteq X$ o $W \subseteq Y$. Si $wu \in E_W$, entonces $v \notin W$, pues $W \subseteq X \Rightarrow v \in Y$, $W \subseteq Y \Rightarrow v \in X$.
De lo anterior se sigue que

$$E_W = \bigcup_{w \in W} \{wv : v \in \Gamma(w)\} \tag{2}$$

donde la unión es disjunta. Luego

$$|E_W| = \sum_{w \in W} |\Gamma(w)| = \sum_{w \in W} d(w) \tag{3}$$

Como G es regular, $d(w) = \delta = \Delta$. Luego

$$|E_W| = \Delta |W| \tag{4}$$

Usando (1) y (4) tenemos que

$$|S|\Delta \leq |\Gamma(S)|\Delta \Rightarrow |S| \leq |\Gamma(S)| \tag{5}$$

Como esto vale para todo $S \subseteq X$, el teorema de Hall implica que existe un matching completo de X a Y. Si demostramos que $|X| = |Y|$, se seguirá que ese matching será perfecto.

La primera forma de probar esto es ver que como X, Y son las partes de $G, E = E_X = E_Y$. Entonces $|E_X| = |E_Y|$, de lo cual se sigue que $\Delta|X| = |Y|\delta \Rightarrow |X| = |Y|$.

Otra forma de verlo es observar que, dado que existe un matching completo de X a $Y, |X| \leq |Y|$. Pero la elección de X sobre Y en la prueba fue arbitraria. Por lo tanto vale lo mismo para Y. Luego $|X| = |Y|$.

En ambos casos, el matching es perfecto. ■

12. Probar que si G es bipartito entonces $\chi'(G) = \Delta(G)$.

Se evalua en diciemre de 2024

13. Probar la complejidad $O(n^4)$ del algoritmo Hungaro y dar una idea de como se la puede reducir a $O(n^3)$

Se evalua en diciemre de 2024

14. Enunciar y probar el teorema de la cota de Hamming

Para todo código $C \in \{0,1\}^n$ con $\delta(c) = \delta$ y $t = \left\lfloor \frac{\delta-1}{2} \right\rfloor$ entonces:

$$|C| \leq \frac{2^n}{1 + n + \binom{n}{2} + \cdots + \binom{n}{t}}$$

Demostración. Sea

$$A = \bigcup_{v \in C} D_t(v)$$

Buscamos $|A|$.

Como C corrige t errores tenemos que

$$D_t(v) \cap D_t(w) = \emptyset, \forall v, w \in C \text{ tales que } v \neq w$$

Es claro que A es unión disjunta
Ahora definimos

$$S_r(v) = \{w \in C : d_H(v,w) = r\}$$

De esta forma es claro que:

$$D_t(v) = \bigcup_{r=0}^{t} S_r(v) \quad \text{unión disjunta}$$

Sea $w \in S_r(v)$, hay un subconjunto de los n bits de las palabras que tiene r elementos tal que w difiere de v en esos r bits. Entonces tenemos:

- Dado $w \in S_r(v)$, podemos obtener r bits en los que v y w difieren.
- Dado un conjunto de r bits, podemos obtener un w tal que $d_H(v,w) = r$

Así, existe una biyección entre $S_r(v)$ y el conjunto de subconjuntos de r bits. Entonces la cardinalidad de estos conjuntos es la misma. Luego

$$|S_r(v)| = \binom{n}{r} \implies |D_t(v)| = \sum_{r=0}^{t} \binom{n}{r}$$

Finalmente tenemos:

$$\begin{aligned}
|A| &= \sum_{v \in C} |D_t(v)| \\
&= \sum_{v \in C} \sum_{r=0}^{t} \binom{n}{r} \\
&= |C| \sum_{r=0}^{t} \binom{n}{r} \\
&\leq 2^n \quad \text{pues } A \subseteq \{0,1\}^n \implies |C| \leq \frac{2^n}{\sum_{r=0}^{t} \binom{n}{r}} \blacksquare
\end{aligned}$$

15. Sea C un código lineal con matriz de chequeo H. Entonces $\delta(C)$ es el mínimo número de columnas LD de H

Demostración.
Sea $m = \text{mín}\{r : \exists r \text{ columnas LD de } H\}$. Probaremos $m = \delta(C)$.
Para ello probaremos que $m \geq \delta(C)$ y $m \leq \delta(C)$.

$m \leq \delta$

$m \geq \delta$