Teoremas para el final de matemática discreta II 2024

Tomás Maraschio

24 de junio de 2024

Índice

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

Para esta demostración usaré las definiciones y resultados de este teorema. **La complejidad de EK es** $O(n \times m^2)$. EK encuentra un camino aumentante de s a t y aumenta el flujo a lo largo de este, esto lo hace hasta que no encuentra más caminos aumentantes. Por lo tanto, su complejidad viene dada por:

$$(Compl(\text{aumentar flujo}) + Compl(\text{encontrar camino aumentante})) \times \#\text{caminos aumentantes}$$

La complejidad de aumentar el flujo en un camino es O(n), pues el camino tiene a lo sumo n vértices y tengo que recorrerlo dos veces. La complejidad de encontrar un camino aumentante es O(m) porque uso BFS. Luego, como $n \leq m$ (en componentes conexas al menos):

$$Compl(EK) = O(m) \times \#\text{caminos aumentantes}$$

Entonces me basta probar que hay $O(m \times n)$ caminos aumentantes. Para ello veré cuántas veces puede un lado volverse crítico, teniendo en cuenta que un lado se vuelve crítico en el paso k cuando para pasar al paso k+1 este se satura o vacía completamente. Esto lo hago porque por cada camino aumentante que encuentre, cuando aumente el flujo, voy a saturar o llenar al menos un lado en ese camino. Entonces si puedo acotar la cantidad de veces que un lado se vuelve crítico, habré acotado la cantidad de caminos aumentantes. Entonces veré los diferentes casos en donde se satura un lado.

1.1. El lado $\overrightarrow{xz}$ se satura en el paso k

Esto significa que para pasar de f_k a f_{k+1} usé un f_k-camino aumentante de la forma $s \cdots xz \cdots t$. Como estoy corriendo EK, este camino es el de menor longitud entre s y t, lo que implica que también es el de menor longitud entre s y x y z, entonces

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

Ahora voy a ver qué pasa cuando $\overrightarrow{xz}$ vuelve a hacerse crítico en un paso j. Esto puede ocurrir por dos razones:

- Si se usa backward en j y se vacía.
- Si en un paso i entre k y j se usa backward y se vacía un poco, para luego en j usarse forward y saturarse.

De todas formas, $\exists i : k < i \leq j$ tal que $\overrightarrow{xz}$ se usa backward en el paso i. Esto significa que para pasar de f_i a f_{i+1} uso un f_i-camino aumentante de la forma $s \cdots \overleftarrow{zx} \cdots t$, y por la misma razón que antes tengo que

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

Juntando todo llego a

$$\begin{aligned} d_j(t) &\geq d_i(t) \\ &= d_i(x) + b_i(x) \\ &= d_i(z) + 1 + b_i(x) \\ &\geq d_k(z) + 1 + b_k(x) \\ &= d_k(x) + 1 + 1 + b_k(x) \\ &= d_k(t) + 2 \end{aligned}$$

1.2. El lado $\overrightarrow{xz}$ se vacía en el paso k

Esto significa que para pasar de f_k a f_{k+1} usé un f_k-camino aumentante de la forma $s \cdots \overleftarrow{zx} \cdots t$. Como estoy corriendo EK, este camino es el de menor longitud entre s y t, lo que implica que

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

Ahora voy a ver qué pasa cuando $\overrightarrow{xz}$ vuelve a hacerse crítico en un paso j. Esto puede ocurrir por dos razones:

- Si se usa forward en j y se satura.
- Si en un paso i entre k y j se usa forward y se llena un poco, para luego en j usarse backward y vaciarse.

De todas formas, $\exists i : k < i \leq j$ tal que $\overrightarrow{xz}$ se usa forward en el paso i. Esto significa que para pasar de f_i a f_{i+1} uso un f_i-camino aumentante de la forma $s \cdots xz \cdots t$, y por la misma razón que antes tengo que

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

Juntando todo llego a

$$\begin{aligned} d_j(t) &\geq d_i(t) \\ &= d_i(z) + b_i(z) \\ &= d_i(x) + 1 + b_i(z) \\ &\geq d_k(x) + 1 + b_k(z) \\ &= d_k(z) + 1 + 1 + b_k(z) \\ &= d_k(t) + 2 \end{aligned}$$

1.3. Conclusión

En ambos casos llego a que si un lado se vuelve crítico, para que pueda volver a hacerlo la distancia entre s y t debe aumentar en al menos 2. Y como esta distancia está siempre entre 1 y n-1, esto puede pasar a lo sumo n/2 = O(n) veces. Como es para cada lado, tengo que en toda la corrida de EK, un lado cualquiera se vuelve crítico $O(n \times m)$ veces, lo que significa que tengo $O(n \times m)$ caminos aumentantes.

2. Probar que $d_k(x) \leq d_{k+1}(x)$

2.1. Definiciones

Primero, recuerdo las definiciones. Para un flujo f y vértices x,z:

$$d_f(x,z) = \begin{cases} 0 & \text{si } x = z \\ \infty & \text{si no existe f-camino aumentante entre x y z} \\ & \text{longitud del menor f-camino aumentante entre x y z} \end{cases}$$

A partir de esta puedo definir para $f_0, f_1, f_2, \cdots$ la sucesión de flujos producidos por Edmonds-Karp:

$$\begin{aligned} d_k(x) &= d_{f_k}(s,x) \\ b_k(x) &= d_{f_k}(x,t) \end{aligned}$$

Y además $d_k(t) = d_k(x) + b_k(x)$, si esos números son finitos.

2.2. Primera parte

Sea el conjunto

$$A = \{x \in V : d_{k+1}(x) < d_k(x)\},$$

si pruebo que este conjunto es vacío ya está. Sup. que A no es vacío, entonces tomo $x \in A$ tal que $d_{k+1}(x) = min\{d_{k+1}(y) : y \in A\}$. Además tengo que

$$d_{k+1}(x) < d_k(x) \leq \infty \implies d_{k+1}(x) < \infty, \tag{0}$$

lo que significa que existe un f_{k+1}-camino aumentante entre s y x. Y además como

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

sé que x tiene un elemento anterior en ese camino, el cual denotaré

$$p_{k+1} : s \cdots zx$$

Ahora, como estoy usando E-K, ese camino es el de menor longitud entre s y x, lo que significa que también es el de menor longitud entre s y cualquier vértice anterior a x, porque si no podría encontrar un camino aún más corto a x. Entonces

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

Luego, como $d_{k+1}(z) < d_{k+1}(x)$ y x es el elemento de A con menor d_{k+1}, sé que $z \notin A$, lo que implica que

$$d_k(z) \leq d_{k+1}(z) \tag{2}$$

Juntando 0, 1 y 2 llego a $d_k(z) < \infty$, lo que significa que existe un f_k-camino aumentante de longitud mínima entre s y z de la forma $p_k : s \cdots z$.

2.3. Si $\overrightarrow{zx}$ es un lado

En principio podría agregar x al final de p_k para obtener un f_k-camino aumentante entre s y x. Pero si hago eso llego a

$$d_k(x) \leq d_k(z) + 1 \leq d_{k+1}(z) + 1 = d_{k+1}(x),$$

lo que es un absurdo porque x está en A. Este absurdo tiene que venir de que no puedo agregar x al camino, y esto tiene que ser porque $f_k(\overrightarrow{zx}) = c(\overrightarrow{zx})$. Pero en p_{k+1} está el lado $\overrightarrow{zx}$, lo que significa que para pasar de f_k a f_{k+1} tuve que usarlo en modo backward. Es decir, existe un camino de longitud mínima (por usar E-K)

$$p'_k : s \cdots \overleftarrow{xz} \cdots t,$$

entonces $d_k(z) = d_k(x) + 1$. Luego,

$$\begin{aligned} d_k(z) &= d_k(x) + 1 \\ &> d_{k+1}(x) + 1 \\ &= d_{k+1}(z) + 2 \\ &\geq d_k(z) + 2 \end{aligned}$$

Absurdo pues 0 <2.

2.4. Si $\overrightarrow{xz}$ es un lado

En principio podría agregar x al final de p_k para obtener un f_k-camino aumentante entre s y x. Pero si hago eso llego a

$$d_k(x) \leq d_k(z) + 1 \leq d_{k+1}(z) + 1 = d_{k+1}(x),$$

lo que es un absurdo porque x está en A. Este absurdo tiene que venir de que no puedo agregar x al camino, y esto tiene que ser porque $f_k(\overrightarrow{zx}) = 0$. Pero en p_{k+1} está el lado $\overleftarrow{zx}$, lo que significa que para pasar de f_k a f_{k+1} tuve que usarlo en modo forward. Es decir, existe un camino de longitud mínima (por usar E-K)

$$p'_k : s \cdots xz \cdots t,$$

entonces $d_k(z) = d_k(x) + 1$. Luego,

$$\begin{aligned} d_k(z) &= d_k(x) + 1 \\ &> d_{k+1}(x) + 1 \\ &= d_{k+1}(z) + 2 \\ &\geq d_k(z) + 2 \end{aligned}$$

Absurdo pues 0 <2.

2.5. Conclusión

En ambos casos llego a un absurdo, y este absurdo viene de suponer que $A \neq \emptyset$, entonces A es vacío y finalmente

$$d_k(x) \leq d_{k+1}(x)$$

3. ¿Cuál es la complejidad de Dinic en ambas versiones? No hace falta probar que la distancia en NAs sucesivos aumenta

Probaré que $Compl(Dinic) = O(n^2 \times m)$.

3.1. Complejidad de NAs

Como en cada NA la distancia entre s y t aumenta, tendré a lo sumo O(n) NAs. Además, como uso BFS, la complejidad de construir cada NA es O(m). Entonces tengo que

$$\begin{aligned} Compl(Dinic) &= (Compl(\text{hallar flujo bloqueante}) + Compl(construirNA)) \times \#\text{cantidad de NAs} \\ &= (Compl(\text{hallar flujo bloqueante}) + O(m)) \times O(n) \end{aligned}$$

Me basta probar que $Compl(\text{hallar flujo bloqueante}) = O(n \times m)$.

3.2. Versión original

Los NAs en esta versión tienen la propiedad de que todo vértice tiene algún lado no saturado que va al siguiente nivel. Esto significa que DFS encuentra un camino de s a t sin tener que hacer backtracking. Entonces la complejidad de encontrar un camino en el NA es O(n° niveles) = O(n). Además, aumentar el flujo en un camino es O(n), pues hay que recorrer el camino dos veces. Ahora, como cada camino aumentante satura (al menos) un lado, tendré O(m) caminos aumentantes. Entonces por ahora tengo que la complejidad de encontrar los caminos y aumentar el flujo en ellos es $O(n \times m)$. Sin embargo, tengo que ver cuánto cuesta mantener la propiedad que mencioné al principio.

Para mantener esa propiedad, luego de cada camino ejecuto el proceso 'podar', que va revisando los vertices desde los niveles superiores para abajo:

- Si el vertice tiene lados que salen no hace nada
- Si no tiene lados que salen, borra el vértice y sus lados al siguiente nivel

Revisar si un vértice tiene lados que salen es O(1), y como luego de cada camino hago esto para cada vértice, esta parte me cuesta $O(n \times m)$.

Ahora, borrar un vértice y sus lados se hace a lo sumo una vez por vértice y su complejidad es O(d(x)). Entonces, la complejidad sobre todos los 'podar' de esta parte es $\sum_{x \in V} d(x) = O(m)$.

Finalmente, la complejidad de hallar flujo bloqueante es

$$O(n \times m) + O(n \times m) + O(m) = O(n \times m)$$

3.3. Versión occidental

Daré el algoritmo que se usa para encontrar un flujo bloqueante g:

```
g := 0
while (1) {
    p := [s]
    x := s
    while (x != t) {
        if (VecinosAdelanteDe(x) != []) {      // Avanzar (A)
            y := tomar de VecinosAdelanteDe(x)
            agregar y al final de p
            x := y
        } else {
            if (x != s) {                      // Retroceder (R)
                z := vértice anterior a x en p
                borrar zx
                x := z
            } else {
                return g
            }
        }
    }
    aumentar g en p y borrar lados saturados   // Incrementar (I)
}
```

Notar que una corrida de este algoritmo es de la forma $A \cdots AIA \cdots AIA \cdots AR \cdots$ es decir, una sucesión de palabras de la forma $A \cdots AX$ donde X es I o R (excepto la última palabra de todas, que terminará con A). Además, $O(A) = O(R) = O(1)$ y $O(I) = O(n)$, pues para aumentar el flujo y borrar los lados tengo que recorrer el camino p dos veces y este tiene a lo sumo longitud n.

Tengo que ver cuantas As puede haber en una palabra. Como cada A me lleva al siguiente nivel del NA, hay O(n) As en cada palabra. De esta forma, el coste de las palabras me quedó:

- $Compl(A....AR) = O(n) + O(1) = O(n)$

- $Compl(A....AI) = O(n) + O(n) = O(n)$

Solo basta ver cuantas de estas palabras puede haber. Como cada R borra un lado y cada I borra al menos un lado, tengo que hay a lo sumo m palabras.

Finalmente:

$$Compl(\text{hallar flujo bloqueante}) = O(n) \times O(m) = O(n \times m)$$

4. ¿Cuál es la complejidad del algoritmo de Wave? No hace falta probar que la distancia en NAs sucesivos aumenta

Probaré que $Compl(Wave) = O(n^3)$.

Al igual que en Dinic, como la distancia entre s y t aumenta en NAs sucesivos, tengo O(n) NAs, pues t no puede estar a una distancia mayor a n de s. Ahora, como

$Compl(Wave) = (Compl(\text{hallar flujo bloqueante}) + Compl(\text{construir NA})) \times O(n)$

y $Compl(\text{construir NA}) = O(m)$ pues uso BFS, bastaría probar que

$$Compl(\text{hallar flujo bloqueante}) = O(n^2).$$

Notar que $O(n^2) + O(m) = O(n^2)$ pues $m \leq \binom{n}{2} = O(n^2)$.

Voy a dividir el proceso de hallar un flujo bloqueante en casos.

Cuando estoy balanceando vértices hacia adelante:

- V: los pasos donde saturo un lado
- P: los pasos donde no saturo un lado

Cuando estoy balanceando los vértices hacia atrás:

- S: los pasos donde vacío un lado
- Q: los pasos donde no vacío un lado

V: Sup. que x le manda flujo a z y $\overrightarrow{xz}$ se satura. Para que $\overrightarrow{xz}$ vuelva a saturarse, primero tiene que vaciarse un poco, pero si se vacía es porque z se bloqueó y entonces le devolvió flujo a x. Pero como z está bloqueado (y nunca se desbloqueará), x no le va a volver a mandar flujo a z. Entonces, cada lado puede saturarse a lo sumo una vez. $\therefore Compl(V) = O(m)$.

S: Sup. que x le devuelve flujo a z y $\overrightarrow{zx}$ se vacía. Como x devolvió flujo significa que está bloqueado, entonces z no volverá a mandarle flujo, entonces $\overrightarrow{zx}$ no se podrá llenar para volver a vaciarse. Esto implica que un lado se puede vaciar a lo sumo una vez. $\therefore Compl(S) = O(m)$.

P: Cuando un vértice manda flujo a sus vecinos para balancearse, satura todos los lados, excepto quizás uno. Esto significa que para cada vértice en cada ola hacia adelante se satura parcialmente a lo sumo un lado. Entonces $Compl(P) = O(n) \times$ #olas hacia adelante.

Ahora, en cada ola hacia adelante (excepto la última) queda al menos un vértice desbalanceado, es decir que cada ola hacia adelante hace que se bloquee al menos un vértice. Y como no se desbloquean una vez bloqueados, tengo que #olas hacia adelante $= O(n)$. $\therefore Compl(P) = O(n^2)$.

Q: Cuando un vértice devuelve flujo a los vecinos, vacía todos los lados excepto quizás uno. Esto significa que para cada vértice en cada ola hacia atrás se vacía parcialmente a lo sumo un lado. Entonces $Compl(Q) = O(n) \times$ #olas hacia atrás $= O(n) \times$ #olas hacia adelante $= O(n^2)$.

Finalmente,

$Compl($**hallar flujo bloqueante**$) = O(m)+O(n^2)+O(m)+O(n^2) = O(n^2)$.

5. Probar que el valor de todo flujo es menor o igual a la capacidad de todo corte. También que si f es flujo, es maximal si y solo si existe S corte con V(f) = Cap(S) y S es minimal. No hace falta probar que $V(f) = f(S, \overline{S}) - f(\overline{S}, S)$

5.1. Valor de todo flujo menor o igual a capacidad de todo corte

$$f(\overline{S}, S) = \sum_{\substack{x \notin S \\ y \in S \\ \overrightarrow{xy} \in E}} f(\overrightarrow{xy}) \geq 0 \text{ pues } f(\overrightarrow{xy}), \forall \overrightarrow{xy} \in E$$

Entonces tengo $V(f) = f(S, \overline{S}) - f(\overline{S}, S) \leq f(S, \overline{S}) \leq c(S, \overline{S}) = Cap(S)$

5.2. $\Longleftarrow$ vuelta

Sea f flujo y S corte con $V(f) = Cap(S)$. Para todo flujo g, por lo que probé arriba tengo $V(g) \leq Cap(S) = V(f) \implies$ f es maximal. Además,

para todo corte T tengo $Cap(T) \geq V(f) = Cap(S) \implies$ S es minimal.

5.3. $\implies$ ida

Sea f flujo maximal. Voy a construir un S y probaré que es un corte minimal con V(f) = Cap(S).

$$S = \{s\} \cup \{x\text{: existe f-camino aumentante de s a x}\}$$

S es corte

Sup. que no lo es. La única forma que pase esto es porque $t \notin S$, pues $s \in S$. Pero esto significa que existe un f-camino aumentante de s a t, por lo que puedo aumentar f en ese camino. Absurdo porque f es maximal, luego S es corte.

V(f) = Cap(S)

Sean $x \in S, y \notin S, \overrightarrow{xy} \in E$.

$$x \in S \implies \text{existe f-camino aumentante de s a x}$$

$$y \notin S \implies \text{no existe f-camino aumentante de s a y}$$

Entonces el camino $s \cdots xy$ podría ser aumentante, pero no lo es, y esto solo puede ser si $f(\overrightarrow{xy}) = c(\overrightarrow{xy})$. Entonces

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

Ahora sean $x \notin S, y \in S, \overrightarrow{xy} \in E$.

$$x \notin S \implies \text{no existe f-camino aumentante de s a x}$$

$$y \in S \implies \text{existe f-camino aumentante de s a y}$$

Entonces el camino $s \cdots \overleftarrow{yx}$ podría ser aumentante, pero no lo es, y esto solo puede ser si $f(\overrightarrow{xy}) = 0$. Entonces

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

Finalmente: $V(f) = f(S, \overline{S}) - f(\overline{S}, S) = Cap(S) - 0 = Cap(S)$

6. Probar que si G es un grafo 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 $\chi(G) \leq \Delta(G)$.

7. Probar que 2-COLOR es polinomial

Para ello voy a dar un algoritmo polinomial que colorea un grafo con dos colores. Luego probaré que si el coloreo que da mi algoritmo no es propio es porque en el grafo hay algún ciclo impar, lo que implicaría $\chi(G) \geq 3$. Notar que voy a asumir que el grafo es conexo, sin embargo si no lo es, simplemente hay que correr el algoritmo en todas sus componentes conexas y ver que todas sean bipartitas.

7.1. Algoritmo

Agarro un vértice x de G y corro BFS empezando en él. Luego a cada vértice y le asigno el color $Nivel_{BFS}(y)$ mod 2. Esto es lo mismo que pintar a x del color 1, y luego cada vez que un vértice z agrega a un vértice y en BFS, asigno $c(y) = 1 - c(z)$. Es claro que esta parte es O(m) por usar BFS.

Ahora tengo que revisar que el coloreo sea en efecto propio, lo que cuesta $\sum_{x \in V} O(d(x)) = O(m)$. Entonces llego a que mi algoritmo es polinomial.

7.2. ¿Qué pasa cuando el coloreo que di no es propio?

Eso significa que $\exists u, v \in V : c(u) = c(v) \wedge uv \in E$. Entonces

$$\begin{aligned} & Nivel_{BFS}(u) \bmod 2 = Nivel_{BFS}(v) \bmod 2 \\ \implies & Nivel_{BFS}(u) + Nivel_{BFS}(v) \text{ es par} \end{aligned} \tag{1}$$

Como u y v fueron agregados por un BFS desde x, sé que existe un camino de x a u y un camino de x a v. Ahora sea w el vértice tal que estos dos caminos se separan (notar que w puede ser x), voy a calcular la cantidad de lados en el ciclo $w \cdots uv \cdots w$.

- En $w \cdots u$ hay $Nivel_{BFS}(u) - Nivel_{BFS}(w)$ lados.
- En uv hay 1 lado.
- En $v \cdots w$ hay $Nivel_{BFS}(v) - Nivel_{BFS}(w)$ lados.

Luego en el ciclo hay

$$\begin{aligned} &Nivel_{BFS}(u) - Nivel_{BFS}(w) + 1 + Nivel_{BFS}(v) - Nivel_{BFS}(w) \\ &\quad = Nivel_{BFS}(u) + Nivel_{BFS}(v) + 2 \times Nivel_{BFS}(w) + 1 \end{aligned}$$

lados, que es un número impar por (1).

8. Enunciar y probar el teorema de Hall

El teorema de Hall dice que en un grafo bipartito con partes X e Y, existe un matching completo de X a Y si y solo si $|\Gamma(S)| \geq |S|, \forall S \subseteq X$. Notar que un matching M de X a Y es completo si |E(M)| = |X|.

8.1. $\implies$ ida

Sea M un matching completo de X a Y. Este me induce una función inyectiva

$$f : X \to Y$$

tal que

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

Luego, como f es inyectiva, tengo que

$$|f(S)| = |S|, \forall S \subseteq X.$$

Y además

$$f(S) \subseteq \Gamma(S).$$

Finalmente, tengo que $|S| \leq |\Gamma(S)|, \forall S \subseteq X$.

8.2. $\Longleftarrow$ vuelta

Sup. que se cumple $|\Gamma(S)| \geq |S|, \forall S \subseteq X$ (condición de Hall), pero que al correr el algoritmo de encontrar matching maximal llego a un M con $|E(M)| < |X|$. Trabajaré sobre la forma matricial del algoritmo. Sean

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

Y en general:

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

Notar que como M no es completo, tengo algunas filas sin matchear, es decir que

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

Además, es claro que

$$S = S_0 \dot\cup S_1 \dot\cup S_2 \dot\cup \cdots \tag{1a}$$
$$T = T_1 \dot\cup T_2 \dot\cup T_3 \dot\cup \cdots \tag{1b}$$

Ahora, cuando estoy viendo una columna pueden pasar dos cosas:

- La columna está libre, entonces la matcheo y extiendo el matching. Esto no pasa porque M es maximal.
- La columna está matcheada con una fila, entonces etiqueto <u>únicamente</u> esa fila.

Así, es claro que

$$|T_i| = |S_i| \tag{2}$$

y que el algoritmo se detiene al pasar de un S_k a un $T_{k+1} = \emptyset$. Entonces llego a que

$$\begin{aligned} S &= S_0 \dot\cup \cdots \dot\cup S_k \\ T &= T_1 \dot\cup \cdots \dot\cup T_k \end{aligned}$$

Juntando todo:

$$\begin{aligned} |S| &= |S_0| + |S_1| + \cdots + |S_k| \quad \text{por 1a} \\ &= |S_0| + |T_1| + \cdots + |T_k| \quad \text{por 2} \\ &= |S_0| + |T| \quad \text{por 1b} \\ &> |T| \quad \text{por 0} \end{aligned}$$

Ahora voy a ver 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 $y \in \Gamma(S)$.
- $\Gamma(S) \subseteq T$: sup. que existe un $y \in \Gamma(S)$ que no está en T. Existe un $x \in S$ que es vecino de y. Pero cuando revisé x, habría visto que y era vecino suyo, y por lo tanto lo habría etiquetado. Absurdo pues supuse $y \notin T$, luego, $\Gamma(S) \subseteq T$.

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

9. Enunciar y probar el teorema del matrimonio de Koenig

Este teorema dice que todo grafo bipartito regular tiene un matchig perfecto. Un matching es perfecto si es completo en ambos sentidos.

Defino

$$E_W = \{wu \in E : w \in W\} \quad \text{para } W \subseteq V$$

Sean X e Y las partes del grafo. Sea $S \subseteq X$ y $l \in E_S$. Entonces existen $x \in S$ y $y \in Y$ tales que $l = xy$. Entonces $y \in \Gamma(x) \subseteq \Gamma(S)$, por lo que $l \in E_{\Gamma(S)}$. Finalmente llego a

$$E_S \subseteq E_{\Gamma(S)} \implies |E_S| \leq |E_{\Gamma(S)}|, \forall S \subseteq X$$

Ahora calcularé cuanto vale $|E_W|$ para $W \subseteq X$ o $W \subseteq Y$. Sea $wu \in E_W$, es claro que $u \notin W$, pues $w \in X$ o $w \in Y$, entonces

$$E_W = \dot{\bigcup}_{w \in W} \{wu : u \in \Gamma(w)\}$$

Entonces

$$\begin{aligned}|E_W| &= \sum_{w\in W} |\{wu : u \in \Gamma(w)\}| \\ &= \sum_{w\in W} d(w) \\ &= \sum_{w\in W} \Delta \quad \text{pues el grafo es regular} \\ &= |W| \times \Delta\end{aligned}$$

Luego

$$\begin{aligned}|E_S| \leq |E_\Gamma(S)| &\Leftrightarrow |S| \times \Delta \leq |\Gamma(S)| \times \Delta \\ &\Leftrightarrow |S| \leq |\Gamma(S)|\end{aligned}$$

Así, por teorema de Hall, sé que hay matching completo de X a Y. Pero la elección de X sobre Y en esta demostración fue arbitraria, por lo que la podría repetir para un $S \subseteq Y$ y llegar a que hay un matching completo de Y a X. Esto significa que $|X| \leq |Y|$ y $|X| \geq |Y|$, por lo que $|X| = |Y|$, entonces el matching es perfecto.

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

10.1. Teorema

Para todo código $C \in \{0,1\}^n$ con $t = \lfloor \frac{\delta-1}{2} \rfloor$:

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

10.2. Demostración

Sea

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

buscaré $|A|$.

Como C corrige t errores, tengo que

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

Luego, es claro que A es unión disjunta.

Ahora, defino

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

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. Con esto en mente:

- Dado $w \in S_r(v)$, puedo obtener r bits en los que v y w difieren.
- Dado un conjunto de r bits, puedo 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. Finalmente:

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

Juntando todo tengo:

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

11. Probar que si $C = Nu(H)$, entonces $\delta(C)$ = tamaño del mínimo conjunto de columnas de H que es LD

Sean $m = \text{mín}\{j : \exists \text{ conjunto LD de j columnas de H}\}$ y $\delta = \delta(C)$.

11.1. $m \leq \delta$

Como C es lineal,

$$\delta = \text{mín}\{|x| : x \in C \wedge x \neq 0\}$$

Sea x tal que $|x| = \delta$. $\exists i_1, \cdots i_\delta$ tales que x tiene un 1 en esas posiciones y un 0 en las demás. Entonces tengo que

$$x = e_{i_1} + \cdots + e_{i_\delta}$$

Además, como $x \in C = Nu(H)$, $H \cdot x^T = 0$.

Luego,

$$\begin{aligned} 0 &= H \cdot x^T \\ &= H \cdot (e_{i_1} + \cdots + e_{i_\delta})^T \\ &= H \cdot (e_{i_1}^T + \cdots + e_{i_\delta}^T) \\ &= H \cdot e_{i_1}^T + \cdots + H \cdot e_{i_\delta}^T) \\ &= H^{(i_1)} + \cdots + H^{(i_\delta)} \end{aligned}$$

Entonces, como existe un subconjunto de δ columnas que es LD, $m \leq \delta$.

11.2. $\delta \leq m$

Por como definí m, sé que hay un conjunto de m columnas que es LD. Sean $H^{(j_1)}, \cdots, H^{(j_m)}$ esas columnas. Luego, sea $x = e_{j_1} + \cdots + e_{j_m}$.

$$\begin{aligned} H \cdot x^T &= H \cdot (e_{j_1} + \cdots + e_{j_m})^T \\ &= H \cdot (e_{j_1}^T + \cdots + e_{j_m}^T) \\ &= H \cdot e_{j_1}^T + \cdots + H \cdot e_{j_m}^T \\ &= H^{(j_1)} + \cdots + H^{(j_m)} \\ &= 0 \end{aligned}$$

Entonces $x \in Nu(H) = C$, y como $x \neq 0$, tengo que $\delta \leq |x| = m$.

12. Sea un código C de longitud n, dimensión k y polinomio generador g(x), probar que:

1. $C = \{p(x) : gr(p) < n \wedge g(x) | p(x)\} = C_1$

2. $C = \{v(x) \odot g(x) : v(x) \text{ es un polinomio cualquiera}\} = C_2$
3. $gr(g) = n - k$
4. $g(x)|(1 + x^n)$

12.1. 1 y 2

Para probar esto, probaré que $C_1 \subseteq C_2 \subseteq C \subseteq C_1$.

$\underline{C_1 \subseteq C_2}$

Sea $p(x) \in C_1$, entonces existe $q(x)$ tal que $p(x) = g(x) \cdot q(x)$. Además

$$n > gr(p) = gr(g(x) \cdot q(x))$$

Entonces es claro que

$$p(x) = g(x) \cdot q(x) = g(x) \odot q(x) \in C_2$$

$\underline{C_2 \subseteq C}$

Sea $p(x) = v(x) \odot g(x) \in C_2$, con $v(x)$ un polinomio cualquiera de la forma

$$v(x) = v_0 + v_1 \cdot x + v_2 \cdot x^2 + \cdots + v_{gr(v)} \cdot x^{gr(v)}$$

Entonces:

$$\begin{aligned} p(x) &= v(x) \odot g(x) \\ &= (v_0 + v_1 \cdot x + v_2 \cdot x^2 + \cdots + v_{gr(v)} \cdot x^{gr(v)}) \odot g(x) \\ &= v_0 \odot g(x) + v_1 \cdot (x \odot g(x)) + v_2 \cdot (x^2 \odot g(x)) + \cdots + v_{gr(v)} \cdot (x^{gr(v)} \odot g(x)) \\ &= v_0 \cdot g(x) + v_1 \cdot Rot(g(x)) + v_2 \cdot Rot^2(g(x)) + \cdots + v_{gr(v)} \cdot Rot^{gr(v)}(g(x)) \\ &\in C \end{aligned}$$

Pues todas las rotaciones de $g(x)$ están en C.

$\underline{C \subseteq C_1}$

Sea $p(x) \in C$, es claro que $gr(p) < n$, falta ver que $g(x)|p(x)$. Entonces voy a dividir p por g:

$$\exists q(x), r(x) : p(x) = g(x) \cdot q(x) + r(x) \wedge gr(r) < gr(g)$$

Ahora tomo módulo:

$$\begin{aligned} p(x) &= p(x) \bmod (1+x^n) \\ &= (g(x)\cdot q(x) + r(x)) \bmod (1+x^n) \\ &= g(x) \odot q(x) + (r(x) \bmod (1+x^n)) \\ &= g(x) \odot q(x) + r(x) \quad \text{pues } gr(r) < gr(g) < n \end{aligned}$$

Entonces tengo que

$$r(x) = p(x) + g(x) \odot q(x)$$

Y como $p \in C$ y $g(x) \odot q(x) \in C_2 \subseteq C$, entonces $r \in C$. Pero como g, es el generador, este es el único polinomio no nulo de menor grado en C, y como $gr(r) < gr(g)$, entonces $r(x) = 0$.

Finalmente, como $g(x)|p(x)$, $p(x) \in C_1$.

12.2. 3

Sea $p(x) \in C$, entonces existe $q(x)$ tal que $p(x) = g(x)\cdot q(x)$. Además $n > gr(p) = gr(g) + gr(q)$, entonces $gr(q) < n - gr(g)$. Ahora, sea un $q(x)$ tal que $gr(q) < n - gr(g)$, tengo que $g(x)\cdot q(x) \in C$.

Es decir, hay una biyección entre C y el conjunto de polinomios de grado menor a $n - gr(g)$. Entonces:

$$\begin{aligned} &|C| = |\text{conjunto de polinomios de grado menor a } n - gr(g)| \\ \iff & \\ &2^k = 2^{n-gr(g)} \\ \iff & \\ &k = n - gr(g) \\ \iff & \\ &gr(g) = n - k \end{aligned}$$

12.3. 4

Divido $1 + x^n$ por $g(x)$:

$$\exists q(x), r(x) : 1 + x^n = g(x)\cdot q(x) + r(x) \wedge gr(r) < gr(g)$$

Ahora, si tomo módulo:

$$\begin{aligned} 0 &= (1+x^n) \bmod (1+x^n) \\ &= g(x)\cdot q(x) + r(x) \bmod (1+x^n) \\ &= g(x) \odot q(x) + (r(x) \bmod (1+x^n)) \\ &= g(x) \odot q(x) + r(x) \quad \text{pues } gr(r) < gr(g) < n \\ \Longrightarrow & \\ r(x) &= g(x) \odot q(x) \in C \end{aligned}$$

Pero como g, es el polinomio de C no nulo de menor grado y $gr(r) < gr(g)$, entonces $r(x) = 0$ y $g(x)|(1+x^n)$.

13. Probar que 3SAT es NP-completo

Como sé que SAT es NP-completo, probaré que SAT $\leq_P$ 3SAT. Primero tengo que dar un algoritmo polinomial que transforme instancias de SAT en instancias de 3SAT.

13.1. El algoritmo

Sea B una expresión booleana con variables $x_1, \cdots, x_i$ tal que

$$B = D_1 \wedge \cdots \wedge D_m$$

Donde cada D_j es una disjunción de literales $l_{j1}, \cdots, l_{jr_j}$. Entonces voy a transformar a cada D_j en un E_j que será una conjunción de disjunciones de 3 literales:

Si $r_j = 1$:

Introduzco las variables y_{j1}, y_{j2}:

$$E_j = (l_{j1} \vee y_{j1} \vee y_{j2}) \wedge (l_{j1} \vee \overline{y_{j1}} \vee y_{j2}) \wedge (l_{j1} \vee y_{j1} \vee \overline{y_{j2}}) \wedge (l_{j1} \vee \overline{y_{j1}} \vee \overline{y_{j2}})$$

Si $r_j = 2$:

Introduzco y_j:

$$E_j = (l_{j1} \vee l_{j2} \vee y_j) \wedge (l_{j1} \vee l_{j2} \vee \overline{y_j})$$

Si $r_j = 3$:

Entonces es claro que $E_j = D_j$.

Si $r_j \geq 4$:

Voy a introducir las variables $y_{j1}, \cdots, y_{j(r_j-3)}$:

$$\begin{aligned} E_j = & (l_{j1} \vee l_{j2} \vee y_{j1}) \\ & \wedge (\overline{y_{j1}} \vee y_{j2} \vee l_{j3}) \\ & \wedge (\overline{y_{j2}} \vee y_{j3} \vee l_{j4}) \\ & \wedge \cdots \\ & \wedge (\overline{y_{j(r_j-4)}} \vee y_{j(r_j-3)} \vee l_{j(r_j-2)}) \\ & \wedge (\overline{y_{j(r_j-3)}} \vee l_{j(r_j-1)} \vee l_{jr_j}) \end{aligned}$$

Es claro que este algoritmo es polinomial, porque si B tiene k literales, agrego a lo sumo k literales más (en realidad es menos que esto).

13.2. Los D son satisfacibles sii los E lo son

Ahora que puedo transformar mi problema SAT en 3SAT, falta ver que mi B original es satisfacible si y solo si el nuevo lo es. Y como B es una conjunción de disjunciones, esto es lo mismo que ver que cada D_j es satisfacible $\iff$ E_j lo es. Esto es lo mismo que ver que

$$D_j(\overrightarrow{b}) = 1 \iff \exists \overrightarrow{d} : E_j(\overrightarrow{b}, \overrightarrow{d}) = 1$$

donde $\overrightarrow{b}$ es el vector de asignación de las variables x, mientras que $\overrightarrow{d}$ es el de las y.

Es trivial ver que esto se cumple para los D_j donde $r_j \leq 3$. Me quedan los otros casos.

$\Longrightarrow$ ida

Sup. que D_j es satisfacible. Es decir, existe un vector de bits $\overrightarrow{b}$ tal que $D_j(\overrightarrow{b}) = 1$. Como D_j es una disjunción, hay un l_{jk} tal que $l_{jk}(\overrightarrow{b}) = 1$. Tomo tal l_{jk} que cumpla eso (si hay más de uno, tomo el primero) y defino $\overrightarrow{d}$:

$$d_1, \cdots, d_{k-2} = 1 \quad \text{y} \quad d_{k-1}, \cdots, d_{r_j-3} = 0$$

Ahora si evalúo E_j, teniendo en cuenta que $y_{ji}(\overrightarrow{d}) = d_i$ y $\overline{y_{ji}(\overrightarrow{d})} = 1 - d_i$:

$$\begin{aligned}
E_j(\overrightarrow{b}, \overrightarrow{d}) &= (l_{j1} \vee l_{j2} \vee y_{j1})(\overrightarrow{b}, \overrightarrow{d}) \quad (\text{" } = 1\text{" pues } d_1 = 1) \\
&\wedge (\overline{y_{j1}} \vee y_{j2} \vee l_{j3})(\overrightarrow{b}, \overrightarrow{d}) \quad (\text{" } = 1\text{" pues } d_2 = 1) \\
&\wedge \cdots \\
&\wedge (\overline{y_{j(k-3)}} \vee y_{j(k-2)} \vee l_{j(k-1)})(\overrightarrow{b}, \overrightarrow{d}) \quad (\text{" } = 1\text{" pues } d_{k-2} = 1) \\
&\wedge (\overline{y_{j(k-2)}} \vee y_{j(k-1)} \vee l_{jk})(\overrightarrow{b}, \overrightarrow{d}) \quad (\text{" } = 1\text{" pues } l_{jk}(\overrightarrow{b}) = 1) \\
&\wedge (\overline{y_{j(k-1)}} \vee y_{jk} \vee l_{j(k+1)})(\overrightarrow{b}, \overrightarrow{d}) \quad (\text{" } = 1\text{" pues } d_{k-1} = 0) \\
&\wedge \cdots \\
&\wedge (\overline{y_{j(r_j-3)}} \vee l_{j(r_j-1)} \vee l_{jr_j})(\overrightarrow{b}, \overrightarrow{d}) \quad (\text{" } = 1\text{" pues } d_{r_j-3} = 0) \\
&= 1
\end{aligned}$$

$\Longleftarrow$ vuelta

Sup. que tengo $\overrightarrow{b}$ y $\overrightarrow{d}$ tales que $E_j(\overrightarrow{b}, \overrightarrow{d}) = 1$. Ahora sup. que $D_j(\overrightarrow{b}) = 0$. Esto implica que todas las l_{jq} se evalúan a 0 con $\overrightarrow{b}$, y como $E_j(\overrightarrow{b}, \overrightarrow{d}) = 1$ esto implica que

$$\begin{aligned}
1 = (&y_{j1} \\
&\wedge (\overline{y_{j1}} \vee y_{j2}) \\
&\cdots \\
&\wedge (\overline{y_{j(r_j-4)}} \vee y_{j(r_j-3)}) \\
&\wedge \overline{y_{j(r_j-3)}})(\overrightarrow{d})
\end{aligned}$$

lo cual es claramente un absurdo. Luego, $D_j(\overrightarrow{b}) = 1$.

14. Probar que 3COLOR es NP-completo

Para ello probaré que 3SAT $\leq$ 3COLOR.

14.1. Algoritmo

Voy a crear un grafo G a partir de una expresión booleana $B = D_1 \wedge \cdots \wedge D_m$ con variables $x_1, \cdots, x_i$, donde $D_j = l_{j1} \vee l_{j2} \vee l_{j3}$.

Vértices

$$\begin{aligned} V(G) =& \{u_1, \cdots, u_i, w_1, \cdots, w_i\} \\ & \cup \{a_{j1}, a_{j2}, a_{j3}\}_{j=1,\cdots,m} \\ & \cup \{e_{j1}, e_{j2}, e_{j3}\}_{j=1,\cdots,m} \\ & \cup \{CAPI, t\} \end{aligned}$$

Lados

- Triángulos
$$\{u_i t, t w_i, w_i u_i\}_{i=1,\cdots,n}$$
- Garras
$$\{a_{j1}a_{j2}, a_{j2}a_{j3}, a_{j3}a_{j1}\}_{j=1,\cdots,m} \cup \{a_{j1}e_{j1}, a_{j2}e_{j2}, a_{j3}e_{j3}\}_{j=1,\cdots,m}$$
- Lados
$$\{(CAPI)e_{jr}\}_{\substack{j=1,\cdots,m \\ r=1,2,3}}$$
- Lados
$$\{(CAPI)t\}$$
- Lados

  Para estos voy a definir
$$v(l_{jr}) = \begin{cases} u_q & \text{si } \exists q : l_{jr} = x_q \\ w_q & \text{si } \exists q : l_{jr} = \overline{x_q} \end{cases}$$
  Y luego los lados serán
$$\{e_{jr}(v(l_{jr}))\}_{\substack{j=1,\cdots,m \\ r=1,2,3}}$$

14.2. B satisfacible sii $\chi(G) \leq 3$

$\Longleftarrow$ vuelta

Sup. $\chi(G) \leq 3$, como en G hay triángulos entonces $\chi(G) = 3$. Sea C un coloreo propio de G que usa 3 colores, como $(CAPI)t$ es un lado, tengo que

los colores posibles son $\{C(CAPI), C(t), Z\}$. Ahora defino el vector de bits $\overrightarrow{b}$:

$$b_i = \begin{cases} 1 & \text{si } C(u_i) = C(CAPI) \\ 0 & \text{si } C(u_i) \neq C(CAPI) \end{cases}$$

Para probar que $B(\overrightarrow{b})$ tengo que ver que $\forall j, \exists r : l_{jr}(\overrightarrow{b}) = 1$. Sea $j \in \{1, \cdots, m\}$. Como los a_{jr} forman un triángulo, hay un r tal que $C(a_{jr}) = C(t)$.

$$\begin{aligned} e_{jr}a_{jr} \in E(G) &\implies C(e_{jr}) \neq C(a_{jr}) = C(t) \\ e_{jr}(CAPI) \in E(G) &\implies C(e_{jr}) \neq C(CAPI) \end{aligned}$$

Lo que implica que $C(e_{jr}) = Z$. Ahora

$$\begin{aligned} e_{jr}v(l_{jr}) \in E(G) &\implies C(v(l_{jr})) \neq C(e_{jr}) = Z \\ tv(l_{jr}) \in E(G) &\implies C(v(l_{jr})) \neq C(t) \end{aligned}$$

Entonces tengo que $C(v(l_{jr})) = C(CAPI)$.

- Si $v(l_{jr})$ es una variable

Existe k tal que $l_{jr} = x_k$. Entonces $v(l_{jr}) = u_k$, y por lo tanto $C(u_k) = C(CAPI)$. Por cómo definí $\overrightarrow{b}$, tengo que $b_k = 1$, entonces

$$l_{jr}(\overrightarrow{b}) = x_k(\overrightarrow{b}) = b_k = 1$$

- Si $v(l_{jr})$ es una negación de variable

Existe k tal que $l_{jr} = \overline{x_k}$. Entonces $v(l_{jr}) = w_k$, y por lo tanto $C(w_k) = C(CAPI)$. Como $u_k w_k$ es un lado, tengo que $C(u_k) \neq C(CAPI)$, entonces $b_k = 0$. Finalmente

$$l_{jr}(\overrightarrow{b}) = \overline{x_k(\overrightarrow{b})} = 1 - b_k = 1$$

Finalmente, como para cualquier j existe un r tal que $l_{jr}(\overrightarrow{b}) = 1$, tengo que $B(\overrightarrow{b}) = 1$.

$\Longrightarrow$ **ida**

Sup. que $B(\overrightarrow{b}) = 1$, voy a dar un coloreo C con 3 colores y probaré que es propio en G. Primero, defino que $C(CAPI) = ROJO$ y $C(t) = VERDE$, por lo que es claro que no hay problema con el lado $t(CAPI)$. Para los u y los w defino:

$$C(u_i) = \begin{cases} ROJO & \text{si } b_i = 1 \\ NEGRO & \text{si } b_i = 0 \end{cases} \quad C(w_i) = \begin{cases} NEGRO & \text{si } b_i = 1 \\ ROJO & \text{si } b_i = 0 \end{cases}$$

Entonces los triángulos formados por u_i, w_i, t no tienen problema ya que están pintados de (NEGRO, ROJO, VERDE) en algún orden.

Ahora tengo que ver cómo pintar los a. Como $B(\overrightarrow{b}) = 1$, sé que

$$\forall j, \exists k_j : l_{jk_j}(\overrightarrow{b}) = 1$$

Entonces tomo ese k_j para cada j (si hay más de uno, tomo el más chico). Luego, para cada $j \in \{1, \cdots, m\}$ defino:

$C(a_{jk_j}) = VERDE$ y pinto los otros dos uno de ROJO y otro de NEGRO

De esta forma, los triángulos de a no tienen problema.

Ahora, para pintar los e:

$$C(e_{jk_j}) = NEGRO \quad C(e_{jr}) = VERDE \text{ para } r \neq k_j$$

Falta ver que los lados con e no tengan problema.

- $e_{jk_j}a_{jk_j}$

  No genera problema pues $C(e_{jk_j}) = NEGRO \neq VERDE = C(a_{jk_j})$

- $e_{jr}a_{jr}$ con $r \neq k_j$

  No hay problema pues $C(e_{jr}) = VERDE$ y $C(a_{jr}) = ROJO$ o $NEGRO$

- $e_{jr}CAPI$

  No hay problema pues $C(e_{jr}) = NEGRO$ o $VERDE$ y $C(CAPI) = ROJO$

- $e_{jr}v(l_{jr})$ con $r \neq k_j$

  No hay problema pues $C(e_{jr}) = VERDE$ y $C(v(l_{jr})) = ROJO$ o $NEGRO$ porque $v(l_{jr}) = u_i$ o w_i para algún i

- $e_{jk_j}v(l_{jk_j})$

- Si l_{jk_j} es una variable
  Entonces existe q tal que $l_{jk_j} = x_q$. Entonces

$$1 = l_{jk_j}(\overrightarrow{b}) = x_q(\overrightarrow{b}) = b_q$$

  Por lo que $C(v(l_{jk_j})) = C(u_q) = ROJO$, entonces no hay problema.

- Si l_{jk_j} es una negación de una variable
  Entonces existe q tal que $l_{jk_j} = \overline{x_q}$. Entonces

$$1 = l_{jk_j}(\overrightarrow{b}) = \overline{x_q}(\overrightarrow{b}) = 1 - b_q \implies b_q = 0$$

  Por lo que $C(v(l_{jk_j})) = C(w_q) = ROJO$, entonces no hay problema.