Teoremas Para el Final
Matemática Discreta II
FaMAF - UNC 2024

Complejidad de Edmonds-Karp

Teorema:

Edmonds-Karp termina siempre con complejidad O(m^2n)

Demostración:

Edmonds-Karp construye una sucesión de caminos aumentantes (de s a t) utilizando BFS a través de los cuales aumenta el flujo inicial, hasta que no existan más caminos. Por lo tanto su complejidad está dada por:

$$compl(Edmonds\ Karp) = O(Compl(Construir\ CamAum + Aumentar\ Flujo) * \#Caminos\ Aumentantes)$$

Notar que $compl(Aumentar\ Flujo) = O(n)$ pues la longitud máxima de un camino formado por BFS es de n-1 lados y aumentar el flujo es $O(1)$ por cada lado. Mientras que $compl(Construir\ CamAum) = O(m)$ pues la construcción se realiza con BFS. Así,

$$compl(Edmonds\ Karp) = O(m * \#Caminos\ Aumentantes)$$

Veremos que $\#Caminos\ Aumentantes = m * n$ y $\therefore$ $compl(Edmonds\ Karp) = O(m * m * n)$

Para ello, **definimos**:

Un lado se vuelve **crítico** en un paso dado si es utilizado para construir el siguiente flujo en modo forwards y se satura, o en modo backwards y se vacía. De esta forma, cada camino aumentante vuelve crítico a al menos un lado.

Definimos:

Dado un flujo f en N y dos vértices x, z:

$d(x, z) = 0,\ si\ x = z$

$d(x, z) = \infty,\ si\ no\ existe\ un\ f\ camimo\ aumentante\ entre\ x\ y\ z$

$d(x, z) = min\{longitud\ de\ f\ camino\ aumentante\ entre\ x\ y\ z\},\ si\ existe$

Si $f_0, f_1,$ ... son los flujos generados por una corrida de $Edmonds\ Karp$, **definimos**:

$$d_k(x) = d_{f_k}(s, x) \text{ y } b_k(x) = b_{f_k}(x, t)$$

Proposición (no hace falta demostrarla):

$$d_k(x) \leq d_{k+1}(x) \wedge b_k(x) \leq b_{k+1}(x)\ \forall x$$

Observación: $d_k(t) = d_k(x) + b_k(x)\ si\ ambas\ son\ finitas.$

Si acotamos la cantidad de veces que un lado puede volverse crítico, acotaremos entonces la cantidad de caminos aumentantes posibles.

Sea $x \rightarrow y$ un lado que se vuelve crítico en el paso k. Esto puede ocurrir porque se saturó o se vació.

Si $x \rightarrow y$ se saturó:

Entonces se utilizó un fk- camino aumentante de la forma s…xy…t y este es de longitud mínima por utilizar Edmonds-Karp. Si el lado xy se vuelve a volver crítico en un paso j>k, entonces es por que se vacía (y se utiliza backwards en el paso j) o porque se vuelve a saturar en el paso j, lo que implica que se utilizó backwards devolviendo flujo en algún paso i con k < i < j. Es decir que para que un lado que se saturó en el paso k se vuelva a volver crítico en un paso j, debe ser utilizado backwards en un paso i con k < i <= j

Por lo tanto se utilizó un fi-camino aumentante de la forma s…yx…t que también es de longitud mínima por utilizar Edmonds-Karp. Pero ahora

$$dj(t) >= di(t) = di(x) + bi(x) = di(y) + 1 + bi(x)$$
$$>= dk(y) + 1 + bk(x) = dk(x) + 1 + 1 + bk(x) = dk(t) + 2$$

Esto significa que la distancia de s a t debe aumentar en al menos 2 para que un lado se pueda volver a volver crítico una vez que se satura.

Si $x \rightarrow y$ se vació:

Entonces se utilizó un fk- camino aumentante de la forma s…yx…t y este es de longitud mínima por utilizar Edmonds-Karp. Si el lado $x \rightarrow y$ se vuelve a volver crítico en un paso j>k, entonces es por que se satura (y se utiliza forwards en el paso j) o porque se vuelve a vaciar en el paso j, lo que implica que se utilizó forwards mandando algo de flujo en algún paso i con k < i < j. Es decir que para que un lado que se vació en el paso k se vuelva a volver crítico en un paso j, debe ser utilizado forwards en un paso i con k < i <= j
Por lo tanto se utilizó un fi-camino aumentante de la forma s…xy…t que también es de longitud mínima por utilizar Edmonds-Karp. Pero ahora

$$dj(t) >= di(t) = di(y) + bi(y) = di(x) + 1 + bi(y)$$
$$>= dk(x) + 1 + bk(y) = dk(y) + 1 + 1 + bk(y) = dk(t) + 2$$

Esto significa que la distancia de s a t debe aumentar en al menos 2 para que un lado se pueda volver a volver crítico una vez que se vacía.

Por lo tanto la cantidad de veces que un lado puede volverse crítico es O(n), pues la longitud de los caminos varía entre 1 y n-1 y cada vez que se vuelve crítico debe aumentar en 2.

Como cada camino aumentante de Edmonds-Karp vuelve crítico al menos un lado, hay m lados y cada lado puede volverse crítico O(n) veces, entonces $\#Caminos\ Aumentantes = O(m * n)$.

Las distancias de Edmonds-Karp no disminuyen

Teorema:
Si dados **vértices x, z y flujo f** definimos a 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 $\mathbf{d_f(x, z)}$,y definimos $\mathbf{d_k(x) = d_{f_k}(s, x)}$ donde $\mathbf{f_k}$ es el k-ésimo flujo en una corrida de Edmonds-Karp, entonces $\mathbf{d_k(x) \leq d_{k+1}(x)}$.

Demostración:
Conjunto A = vértices que no cumplen la regla
Sup A != vacio
Elijo el x con menor d_{k+1} en A
Existe f_{k+1}-camino aumentante s…zx en el paso k+1 ($d_{k+1}(x) < d_k(x) <=$ inf) y s no está en A =>
hay un z anterior a x y además $d_{k+1}(z) = d_{k+1}(x) - 1$

Como $d_{k+1}(z) < d_{k+1}(x)$ => z no está en A entonces y $d_k(z) <= d_{k+1}(z)$

$d_k(z) <= d_{k+1}(z) <$ inf => hay un f_k-camino aumentante p_k: s…z en el paso k

Si zx es un lado entonces lo podriamos agregar en modo forwards al camino s…z en el paso k pero esto cumpliria
$d_k(x) = d_k(z) + 1 <= d_{k+1}(z) + 1 <= d_{k+1}(x)$ absurdo pues x está en A

Luego el lado zx debe estar saturado, pero en el paso k+1 ya no lo está pues hay un camino aumentante s…zx entonces se debe haber utilizado backwards entre f_k y f_{k+1} (en el paso k) con un f_k-camino aumentante de la forma s…xz...t
pero esto implica $d_k(z) = d_k(x) + 1 > d_{k+1}(x) + 1 = d_{k+1}(z) + 1 + 1 >= d_k(z) + 2$ **absurdo**

Si xz es un lado podriamos agregar x backwards al camino s….z en el paso k pero esto cumpliria
$d_k(x) = d_k(z) + 1 <= d_{k+1}(z) + 1 <= d_{k+1}(x)$ absurdo pues x está en A

Luego el lado xz debe estar vacio pero no lo está más en el paso k+1
=> se usó forwards en el paso k con un camino de la forma s….xz…..t
lo cual cumple $d_k(z) = d_k(x) + 1 > d_{k+1}(x) + 1 = d_{k+1}(z) + 1 + 1 >= dk(z) + 2$ **absurdo**

Luego A es vacio.

Complejidad de Dinic (ambas versiones)

Teorema:
La complejidad del algoritmo de dinic, tanto "original" como "occidental" es $O(n^2m)$
Demostración:
O(cant NA (construir + hallar flujo bloqueante)
O(N * (m + ?))

version original
? = DFS en O(n) y cada DFS bloquea un lado => O(m) veces => O(mn)
PODAR => hay que podar luego de cada camino (O(m) podar donde se ven en O(1) los O(n) vertices)
podar es O(mn) + eliminar los lados de los vertices = O(m) pues sum{grado(v)} = O(m)

versión occidental con pseudo código del paso de hallar flujo bloqueante:

```
g = 0, flag = 1;
while (flag)
  p = [s]; x = s;
  if (hay vecinos de adelante de x)
       y = un vecino
       agrego y al camino p A
       x = y
  else
      if (x != s)
          y = anterior a x en p R
          elimino x de P e yx del NA
          x = y
      else
          flag = 0
  if (x == t)
       aumentar flujo g y eliminar los lados saturados C
```

Una corrida de este algoritmo será de la forma A….ARA…ACA…AR…
el paso A es O(1) pues simplemente agrega a una lista
el paso R es O(1) pues simplemente elimina de una lista
el paso C es O(n) pues aumentar flujo es O(n) y no puede haber mas de n lados en p[]

Cada R elimina un lado y cada C elimina al menos un lado => sólo puede haber O(m) A…AX en la corrida del algoritmo, con X = C o R

Cuántas veces ocurre A en cada paso? Cada A avanza un nivel en el NA, y hay O(n) niveles por lo tanto hay O(n) A's por cada paso

Así, hallar flujo bloqueante = compl(O(m) * compl(A…AX))

compl(A…AX) = compl(A….A) + compl(X) = O(n)*O(1) + O(1)*O(n) en el peor caso donde X=C

finalmente compl(hallar flujo bloqueante) = O(m * n)

Complejidad de Wave (tarjan)

Teorema:
El algoritmo wave de tarjan tiene complejidad $O(n^3)$
Demostración:
compl(wave) = O(cant NA * (construirlos + hallar bloqueante))
= O(N * (m + ?))
notar que $O(m) + O(n^2) = O(n^2)$ pues $m < nC2 = n^2 - n / 2$

conjuntos
ola forwards, enviando flujo cada lado se puede saturar o no
S = {casos donde los lados se saturan}
P = {casos donde no se saturan}
ola backwards, devolviendo flujo cada lado se puede vaciar o no
V = {casos donde los lados se vacían}
Q = {casos donde no se vacían}

Análisis de S
Hay O(m) lados, cuántas veces se podrían saturar? supongamos que un lado se saturó, para que se vuelva a saturar, entonces tendrá que haber devuelto flujo (para poder enviar y saturar de nuevo), pero si devolvió flujo entonces está bloqueado y no podrá volver a mandar, ni saturarse
S = O(m)

Análisis de V
Hay O(m) lados, cuántas veces se podrían vaciar? supongamos que un lado se vació, para que se vuelva a vaciar, entonces tendrá que haber enviado flujo (para poder devolver y vaciarse de nuevo), pero si devolvió flujo la primera vez entonces está bloqueado y no podrá volver a mandar, ni vaciarse otra vez
V = O(m)

Análisis de P
En cada ola hacia adelante, para cada vértice, se saturan todos sus lados salvo quizás el último, y enviar flujo por el es O(1)
P = O(n) * #olas hacia adelante

Analisis de Q
En cada ola hacia atrás, para cada vértice, se vacían todos sus lados salvo quizás el último, y devolver flujo por el es O(1)
Q = O(n) * #olas hacia atrás

olas hacia adelante = # olas hacia atrás y en cada ola hacia adelante, salvo quizás en la última, se desbalancea y por lo tanto se bloquea un vértice que no se vuelve a desbloquear, por lo tanto hay O(n) olas hacia adelante.

compl(P) = compl(Q) = O(n*n)

Luego compl(hallar flujo bloqueante en cada NA) = $O(2*m + 2*n^2) = O(n^2)$

Max flow, Min cut

Teorema:
El valor de todo flujo es menor o igual a la capacidad de todo corte
flujo f => f es maximal ⇔ v(f) = cap(S) para algún corte S (y el corte es minimal)
Demostración:
Lema importante: v(f) = f(S, complemento(S)) - f(complemento(S), S)
Con ese lema es facil ver la parte 1

parte 2, f flujo

f maximal -> construyo S = {s} U {x : existe un f-camino aumentante entre s y x}
s e S
t no está en S porque podría aumentar el flujo y no sería maximal

usando lema, v(f) = f(S, complemento(S)) - f(complemento(S), S)

f(S, complemento(S)) = suma f(xy) y existe un camino aumentante hasta x, pero no hasta y, pero existe el lado xy entonces xy debe estar saturado => f(xy) = c(xy) para todo xy => f(S, complemento(S)) = cap(S)

f(complemento(S), S) = suma f(xy) donde existe un camino aumentante hasta y pero no hasta x, pero existe el lado xy que se podria usar backwards para hacer un camino aumentante hasta x es decir xy debe estar vacío => f(xy) = 0 para todo xy => f(complemento(S), S) = 0

luego v(f) = cap(S)

S corte y f flujo con v(f) = cap(S)

en la parte 1 vimos que v(g) <= cap(S) = v(f) por lo tanto f es maximal

Notar que S es minimal pues también por 1 cap(T) >= v(f) = cap(S) para todo T y f

Baby Brooks

Teorema:
Si G es un grafo conexo no regular, X(G) <= Delta(G)
Demostración:
Colorear con Greedy utilizando el orden DFS(x) inverso donde x es el de menor grado d < D y todos van a tener un vecino arriba hasta que llegas a x pero x tiene < D vecinos.
Utilizar greedy garantiza que el coloreo es propio, veamos que siempre habrá un color disponible entre 1 y D para todos los vértices.

En el orden DFS inverso, todos los vértices, salvo x, tienen un vecino anterior que aún no fue coloreado (pues es quien los agregó al orden. Luego como máximo tienen D - 1 vecinos **ya coloreados** => habrá un color disponible en {1, …, D} para ellos. Cuando llegamos a X, no tiene vecinos anteriores, pero d(x) = x => tiene d<D vecinos => tendrá un color disponible que ninguno de sus vecinos utilice en {1, … ,D}

2-COLOR es polinomial

Teorema:
Se puede determinar que un grafo G es bipartito o “2-coloreable” deterministicamente en tiempo polinomial sobre el tamaño del grafo.
Demostración:
Corremos BFS(x) a partir de algún vértice x de G en tiempo O(m) y obtenemos los niveles de cada vértice en BFS.

Coloreamos G con el coloreo c tal que $c(v) = NIVEL_{BFS}(v) \bmod 2$

Chequeamos si es propio, esto es, para cada vértice, chequear que todos sus vecinos tengan colores distintos a él. Se hace en O(suma de grados) = O(m)

Si es propio => X(G) >= 2 fin.

Si no es propio, existe un lado vw con $NIVEL_{BFS}(v) \bmod 2 = NIVEL_{BFS}(w) \bmod 2$
Es decir que sus niveles tienen la misma paridad y por lo tanto la suma de sus niveles sería un número PAR.

Por lo obtenido en BFS, existen en el grafo caminos x…v y x…w los cuales se separan a partir de algún vértice z, es decir que son de la forma x…z…v y x…z…w lo cual implica la existencia de un ciclo

z….vw….z en G, ¿Cuántos lados tiene?
z…v tiene $NIVEL_{BFS}(v) - NIVEL_{BFS}(z)$ lados
vw es 1 lado
w…z tiene $NIVEL_{BFS}(w) - NIVEL_{BFS}(z)$ lados

En total, el ciclo tiene $NIVEL_{BFS}(v) - NIVEL_{BFS}(z) + 1 + NIVEL_{BFS}(w) - NIVEL_{BFS}(z)$

Pero la suma de los niveles de v y w es un valor par como dijimos antes, $2*NIVEL_{BFS}(z)$ es par, y al sumarle 1 significa que el ciclo de G tiene una cantidad IMPAR de lados por lo tanto X(G) >= 3.

Teorema de Hall

Teorema:
En un grafo G bipartito con partes X e Y, existe un matching “completo” de X a Y ⇔ para todo S contenido en X, |vecinos(S)| >= |S|

Demostración:
ida: el matching induce una biyección f entre los elementos de X con sus vecinos en Y, por lo que |f(S)| = |S| para todo S contenido en X y f(S) está contenido en vecinos(S) por lo que |vecinos(S)| >= |S| para todo S contenido en X

vuelta: suponemos la condicion de hall, para todo S contenido en X, |vecinos(S)| >= |S|

Supongamos que al correr el algoritmo para hallar matching maximal, hallamos un matching M con |E(M)| < |X|, es decir que no cubre a X

Corremos el algoritmo en su forma matricial de nuevo sobre M, obteniendo un conjunto S de filas etiquetadas y un conjunto T de columnas etiquetadas.
Definimos
S_0 = { filas etiquetadas con * en el primer paso }
Y para i = 1, …, k
T_i = { columnas etiquetadas por S_{i-1} }
S_i = { filas etiquetadas por T_i }

donde k es el paso en el que frena el algoritmo, de esta manera es claro que
S = unión (disjunta) de S_i
T = unión (disjunta) de T_i

Notar que el algoritmo nunca frena al pasar de un T_i a un S_i con i=1, …,k pues una columna etiquetada está libre y extiende el matching (lo cual no ocurre en este caso), o etiqueta a la fila con la que está matcheada. Por ello, existen la misma cantidad de estos conjuntos (k) sin contar el S_0.

Notar también que cada columna de T_i etiqueta únicamente a una fila (agrega únicamente una fila) al conjunto S_i y por lo tanto $|S_i| = |T_i|$ para todo i = 1, …, k

Luego como las uniones son disjuntas,
$|S| = |S_0| + |S_1| + \ldots + |S_k|$
$= |S_0| + |T_1| + \ldots + |T_k|$
$= |S_0| + |T|$
$> |T|$
Veamos ahora que T = vecinos(S)
T son las columnas etiquetadas, y las columnas son etiquetadas por filas de S, pero cada fila de S etiqueta únicamente a columnas vecinas, por lo que x en T => x es vecino de alguna fila de S => x en vecinos(S)

De esta forma, S es un subconjunto de X que cumple |S| > |vecinos(S)| (violando la condición de hall que supusimos cierta) por lo cual esto es un absurdo de asumir que |E(M)| < |X|.

Finalmente |E(M)| = |X| pues tampoco puede ser mayor.

Teorema del Matrimonio de König

Teorema:
Todo grafo bipartito regular tiene un matching perfecto.
Demostración:
Sea G un grafo bipartito regular con d = D = delta(G) con partes X e Y

Definimos, para un conjunto W contenido en E(G), E_w = { xy e E(G): x e W }

Luego, tomamos un subconjunto de vértices S contenido en X

xy en E_S => y en vecinos(x) => y en vecinos(S) => xy en $E_{\mathbf{vecinos(S)}}$

Por lo tanto E_s contenido en $E_{\mathbf{vecinos(S)}}$ => $|E_s|$ <= $|E_{\mathbf{vecinos(S)}}|$

¿Cuál es la cardinalidad de E_w?

E_w = unión disjunta de lados de cada uno de los vértices de w
$|E_w|$ = suma de grados de vértices de W, pero todos tienen el mismo grado por lo que
= D * | W |

Luego utilizando $|E_s|$ <= $|E_{\mathbf{vecinos(S)}}|$ obtenemos D * |S| <= D * |vecinos(S)| y por lo tanto
|S| <= |vecinos(S)| para todo S contenido en X.

Luego el teorema de hall aplica indicando que existe un matching completo de X a Y.

Notar que |X| <= |vecinos(X)| <= |Y|, pero la elección de X sobre Y fue arbitraria por lo tanto también se cumple la condición de hall para todo S contenido en Y, lo cual implica que |Y| <= |X|

Finalmente |X| = |Y| y el matching “completo” es “perfecto”.

Cota de Hamming

Teorema:

Sea C un código binario en $\{0,1\}^n$, de longitud N, d = d(c) y t = floor((d-1)/2)
luego |C| <= 2^n / sum(nCr) r=1,...,t

Demostración:

Definimos A contenido en $\{0,1\}^n$ como la unión de los discos de radio t al rededor de las palabras de C, esto es:

A = union { $D_t(v)$ } para v en C

Como C corrige t errores, la unión es disjunta y

| A | = suma de | $D_t(v)$ | para todo v en C

Observamos que $D_t(v)$ = unión { $S_r(v)$ } para r = 0, …, t donde $S_r(v)$ = { x en $\{0,1\}^n$: $d_H(v, x)$ = r}

Y la unión es disjunta pues no hay dos palabras iguales que estén a distintas distancias de v para ningún v.

Luego

| $D_t(v)$ | = suma de |{ $S_r(v)$ }| para r = 0

Como $S_r(v)$ son las palabras que difieren en r bits de v, hay una biyección entre los conjuntos de subconjuntos de r bits y los elementos de $S_r(v)$

Así,

|$S_r(v)$| = | conjunto de subconjuntos de r bits de los n posibles| = nCr

Juntando todo,

| A | = suma de | $D_t(v)$ | para todo v en C
| $D_t(v)$ | = suma de |{ $S_r(v)$ }| para r = 0, …, t
|$S_r(v)$| = nCr ← independiente de v

luego

| $D_t(v)$ | = suma de |nCr| para r = 0, …, t ← independiente de v
| A | = suma de (suma de nCr para r=0,...,t) para todo v en C pero al ser independiente de v entonces
| A | = |C| * suma de nCr para r=0,...,t

Así,

|C| = |A| / suma de nCr para r=0,...,t y |A| <= 2^n
=>
|C| <= 2^n / suma de nCr para r=0,...,t

δ(G) = mín{#columnas LD de H}

Teorema:

Si H es la matriz de chequeo de un código C, entonces δ(G) = mín{#columnas LD de H}

Demostración:

m <= d

d => una palabra x distinta de 0 en C tiene d unos pues d = min{ |x| : x en C, x != 0 }

=> $Hx^t = H(e_{i1} + \dots + e_{id})^t = H(e_{i1})^t + \dots + H(e_{id})^t$ = suma de d columnas = 0

=> d columnas LD => m <= d

d <= m

m => suma de m columnas = 0 => $H(ei1 + .. + eim)^t = 0$

=> la palabra (ei1 + .. + eim) está en el código pues está en Nu(H) y es distinta de 0 pues m >= 1

=> como d = min{ |x| : x en C, x != 0 }, entonces d <= m

Propiedades de un código cíclico

Teorema:

Si C es un código cíclico de longitud n y dimensión k, y g es su polinomio generador, entonces:

I. $C = \{p(x) : gr(p) < n \ \& \ g(x) \mid p(x)\}$
II. $C = \{v(x) * g(x) \bmod (1+x^n) : v(x) \text{ es un polinomio cualquiera}\}$
III. $gr(g) = n - k$
IV. $g(x) \mid (1+x^n)$

Demostración:

Definimos:

$C_1 = \{p(x) : gr(p) < n \ \& \ g(x) \mid p(x)\}$
$C_2 = \{v(x) * g(x) \bmod (1+x^n) : v(x) \text{ es un polinomio cualquiera}\}$

Para demostrar i) e ii), veremos que $C_1 \subseteq C_2 \subseteq C \subseteq C_1$ y por lo tanto todos serán iguales.

$\mathbf{C_1 \subseteq C_2}$

Dada p en C_1, sabemos que $gr(p) < n$ por lo que $p(x) \bmod (1+x^n) = p(x)$
como $g(x) \mid p(x)$ entonces existe un polinomio $q(x)$ tal que
$p(x) = g(x)q(x)$
tomando módulo,
$p(x) \bmod (1+x^n) = g(x)q(x) \bmod (1+x^n)$
$p(x) = g(x)q(x) \bmod (1+x^n)$ está en C_2

$\mathbf{C_2 \subseteq C}$

Dada p en C_2, sabemos que $p(x) = v(x) * g(x) \bmod (1+x^n)$ para algún polinomio $v(x)$
luego
$p(x) = (v_0 + v_1x + v_2x^2 + \ldots + v_dx^d) * g(x) \bmod (1+x^n)$ para algún d
$= v_0 * g(x) + v_1x * g(x) + \ldots + v_dx^d * g(x) \bmod (1+x^n)$
$= v_0*g(x) + v_1 rot(g) + \ldots + v_d rot^d(g)$
y todos los términos son palabras de C por lo tanto $p(x)$ está en C.

$\mathbf{C \subseteq C_1}$

Dada p en C, sabemos que $gr(p) < n$ por lo que $p(x) \bmod (1+x^n) = p(x)$
dividir p por g nos da un $q(x)$ y $r(x)$, con $gr(r) < gr(g)$ tal que
$p(x) = g(x)q(x) + r(x)$
tomando módulo,
$p(x) \bmod (1+x^n) = g(x)q(x) + r(x) \bmod (1+x^n)$
$p(x) = g(x)q(x) \bmod (1+x^n) + r(x)$
$r(x) = p(x) + g(x)q(x) \bmod (1+x^n)$

pero $p(x)$ en C y $g(x)q(x) \bmod (1+x^n)$ en $\mathbf{C_2 \subseteq C}$ => $r(x)$ en C pero $gr(r) < gr(g)$
=> $r(x) = 0$ pues g es el único polinomio de menor grado en C
=> $gr(p) < n \ \& \ g(x) \mid p(x)$
=> $p(x)$ en C_1

Veamos ahora iii)

Como C es lineal de dimension k, $|C| = 2^k$

Por definición, C_1 incluye las palabras de grado menor a n que sean múltiplos de g(x). Esto induce una biyección entre las palabras de C_1 y los polinomios de grado < n - gr(g), pues para cada una de las palabras de C_1 existe un q(x) con grado < n-gr(g) tal que g(x)q(x) = p(x) sea de grado < n

Luego $|C| = |C_1|$ = |polinomios de grado < n-gr(g)| = $2^{n - gr(g)}$
=> $2^k = 2^{n - gr(g)}$
=> k = n - gr(g)
=> gr(g) = n-k

Veamos, finalmente, iv)
Dividir $(1+x^n)$ por g(x) nos da dos polinomios
q(x) y r(x) con gr(r) < gr(g) tal que $1+x^n = g(x)q(x) + r(x)$
Tomando módulo,
$1+x^n \bmod (1+x^n) = g(x)q(x) + r(x) \bmod (1+x^n)$
$0 = g(x)q(x) \bmod (1+x^n) + r(x)$
$r(x) = g(x)q(x) \bmod (1+x^n)$
El término de la derecha es una palabra de C_2 i.e una palabra de C, por lo tanto r(x) en C y gr(r) < gr(g), como vimos anteriormente, significa que r(x) = 0 por las propiedades de g.
Finalmente concluimos que $g(x) \mid (1+x^n)$

3-SAT es NP-completo

Teorema:
3-SAT es NP-Completo (3-SAT = 3-CNF-SAT)
Demostración:
Para demostrar que 3-SAT es NP completo, mostraremos que CNF-SAT se reduce polinomialmente a 3-SAT, y como CNF-SAT es NP-completo, entonces 3-SAT también lo será.

Para ello, debemos construir una instancia A(B) de 3-SAT a partir de una instancia B de CNF-SAT, en tiempo polinomial sobre el tamaño de B.

Una instancia B con variables $x_1, \dots, x_n$ de CNF-SAT es de la forma $B = D_1 \& \dots \& D_m$, donde $D_j = l_{j1} \lor \dots \lor l_{j(rj)}$ donde l_{jk} son literales i.e. variables o negaciones de variables.

Queremos transformarla en $A(B) = E_1 \& \dots \& E_m$ donde cada E_j será una conjunción de disyunciones de 3 variables i.e. una instancia de 3-CNF-SAT.

Además queremos, dado $b = (b_1, \dots, b_n)$ vector de asignación de variables $x_1, \dots, x_n$

$$B(b) = 1 \Leftrightarrow \text{Existe } d \text{ (asignaciones de } y_{jk}) \text{ tal que } [A(B)](b)(d) = 1$$

i.e B es satisfacible ⇔ A(B) lo es

Notar que como B es una conjunción de D_j's, $B(b) = 1 \Leftrightarrow D_j(b) = 1$ para j=1, ..., m

Para obtener A(B), transformaremos cada D_j en un E_j de la siguiente manera:

Si $rj = 1 \rightarrow E_j = (l_{rj} \lor y_{j1} \lor y_{j2}) \& (l_{rj} \lor \neg y_{j1} \lor y_{j2}) \& (l_{rj} \lor y_{j1} \lor \neg y_{j2}) \& (l_{rj} \lor \neg y_{j1} \lor \neg y_{j2})$
Si $rj = 2 \rightarrow E_j = (l_{j1} \lor lj_2 \lor y_{j1}) \& (l_{j1} \lor lj_2 \lor \neg y_{j1})$
Si $rj = 3 \rightarrow E_j = D_j$

Hasta aquí, es claro que $E_j(b) = 1 \Leftrightarrow D_j(b) = 1$ pues las variables agregadas **no pueden** hacer todos los términos verdaderos a la vez i.e. lo deben hacer las variables de D_j

Si $r_j >= 4 \rightarrow E_j = (l_{j1} \lor l_{j2} \lor y_{j1}) \& (\neg y_{j1} \lor y_{j2} \lor l_{j3}) \& (\neg y_{j2} \lor y_{j3} \lor l_{j3}) \& \dots \& (\neg y_{j(rj-3)} \lor y_{j(rj-2)} \lor l_{j(rj-2)}) \& (\neg y_{j(rj-2)} \lor l_{j(rj-1)} \lor l_{j(rj)})$

Notar que la construcción de cada E_j es en tiempo polinomial sobre el tamaño de cada D_j ya que la cantidad de variables agregadas es lineal en la cantidad de literales en D_j. Luego la construcción de A(B) completa es en tiempo polinomial sobre el tamaño de B.

Veamos ahora que dado b vector de asignación de variables y $r_j >= 4$,

$$D_j(b) = 1 \Leftrightarrow E_j(b)(d) = 1$$

(=>)

Dado un j en {1, …, m} y b = $b_1, …, b_n$ asignación de variables

$D_j(b) = 1$ => Existe un literal $l_{j(kj)}$ tal que $l_{j(kj)}(b) = 1$, si hay más de uno tomamos el primero.

Construimos d = $(d_1, …, d_{rj})$ con $d_1 = … = d_{kj-2} = 1$ y $d_{kj-1} = … = d_{rj-2} = 0$, donde $y_{jh}(d) = d_h$

$E_j = (l_{j1} \vee l_{j2} \vee y_{j1})(b)(d)$
& $(\neg y_{j1} \vee y_{j2} \vee l_{j3})(b)(d)$
& …
& $(\neg y_{j1} \vee y_{j2} \vee l_{j3})(b)(d)$ ← Todos los términos anteriores a este son 1 pues tienen un d_h con 1 <= h <= kj-2
& $(\neg y_{j(kj-2)} \vee y_{j(kj-1)} \vee l_{jkj})(b)(d)$ ← Este término es =1 pues $l_{jkj}(b) = 1$
& $(\neg y_{j(kj)} \vee —)(b)(d)$ ← Desde aquí hasta el final, todos los términos son 1 pues comienzan con
& … un $\neg d_h = 1$ con jk <= h <= rj-2
& $(\neg y_{j(rj-3)} \vee y_{j(rj-2)} \vee l_{j(rj-2)})(b)(d)$
& $(\neg y_{j(rj-2)} \vee l_{j(rj-1)} \vee l_{j(rj)})(b)(d)$

Luego queda demostrado que dado b que hace verdadero a D_j existe un d tal que junto con el b dado hacen verdadero al E_j correspondiente.

(<=) Veamos ahora que dado b = $(b_1, …, b_n)$, d = $(d_1, …, d_{rj})$ vectores de asignaciones de x_i e y_i respectivamente,
$E_j(b)(d) = 1 \Leftrightarrow D_j(b) = 1$

Supongamos que no, es decir que existen b = $(b_1, …, b_n)$ y d = $(d_1, …, d_{rj})$
tal que $E_j(b)(d) = 1$ pero $D_j(b) = 0$

$D_j(b) = 0 \Rightarrow l_{jh}(b) = 0$ para todo h=1, ..., rj

luego
$E_j(b) = 1 = (l_{j1} \vee l_{j2} \vee y_{j1})$ & $(\neg y_{j1} \vee y_{j2} \vee l_{j3})$ & $(\neg y_{j2} \vee y_{j3} \vee l_{j3})$ & … & $(\neg y_{j(rj-3)} \vee y_{j(rj-2)} \vee l_{j(rj-2)})$ & $(\neg y_{j(rj-2)} \vee l_{j(rj-1)} \vee l_{j(rj)})$
= $(0 \vee 0 \vee y_{j1})$ & $(\neg y_{j1} \vee y_{j2} \vee 0)$ & $(\neg y_{j2} \vee y_{j3} \vee 0)$ & … & $(\neg y_{j(rj-3)} \vee y_{j(rj-2)} \vee 0)$ & $(\neg y_{j(rj-2)} \vee 0 \vee 0)$
= y_{j1} & $(\neg y_{j1} \vee y_{j2})$ & … & $(\neg y_{j(rj-3)} \vee y_{j(rj-2)})$ & $(\neg y_{j(rj-2)})$
= 0 pues si fuese 1, $y_{j1} = 1 \Rightarrow y_{j(rj-2)} = 1$ pero $(\neg y_{j(rj-2)}) = 1$, absurdo

Concluimos entonces por (=>) y (<=) que D_j es satisfacible $\Leftrightarrow$ E_j lo es y por lo tanto B es satisfacible $\Leftrightarrow$ A(B) lo es.

3-COLOR es NP-completo

Teorema:
3-COLOR es NP-Completo
Demostración:
Para demostrar que 3-COLOR es NP-Completo, veremos que 3(-CNF)-SAT se reduce polinomialmente a 3-COLOR, y como 3-SAT es NP-Completo, entonces 3-COLOR también lo será.

Para ello, debemos dar un algoritmo A polinomial en el tamaño de una instancia B de 3-SAT de manera tal que A(B) = G sea una instancia de 3-COLOR y que B sea satisfacible ⇔ X(G) <= 3

Como B es una instancia de 3-SAT, es de la forma $B = D_1$ & … & D_m con $D_j = l_{j1} \lor l_{j2} \lor l_{j3}$ donde cada l_{jk} es una variable en $\{x_1, \dots, x_n\}$ o su negación.

Construimos A(B) = G:

Vértices:

$V(G) = \{u_i, w_i\} \cup \{a_{j1}, a_{j2}, a_{j3}, e_{j1}, e_{j2}, e_{j3}\} \cup \{s, t\}$ para $i = 1, \dots, n$; $j = 1, \dots, m$

Lados:

$E(G) = \{ a_{j1}a_{j2}, a_{j2}a_{j3}, a_{j3}a_{j1} \}$ ← Triangulos de a's
$\cup \{ a_{jr}e_{jr} \}$ ← "garras" de los triángulos de a's
$\cup \{ tu_i, tw_i, u_iw_i\}$ ← Triángulos de u's, w's y t
$\cup \{ st \}$
$\cup \{ se_{jr} \}$
$\cup \{ e_{jr}v(l_{jr})\}$
para todo $j = 1, \dots, m$; $i = 1, \dots, n$; $r = 1,2,3$;
donde $v(l_{jr}) = u_i$ si Existe i tal que $l_{jr} = x_i$
$= w_i$ si Existe i tal que $l_{jr} = \neg x_i$

Es claro que la construcción de G es en tiempo polinomial pues las cantidades de lados y vértices son lineales en n, m y r (tamaño de B).

Veamos ahora que B es satisfacible ⇔ X(G) <= 3

Notar primero, que
B es satisfacible
⇔
Existe $b = (b_1, \dots, b_n)$ asignaciones de variables tal que B(b) = 1
⇔
$D_j(b) = 1$ para todo $j = 1, \dots, m$

(=>) Asumimos que B es satisfacible i.e Existe $b = (b_1, \dots, b_n)$ tal que $D_j(b) = 1$ para todo $j = 1, \dots, m$
Queremos ver que X(G) <= 3, para ello, daremos un coloreo c propio de G con 3 colores.

Colorearemos el grafo G paso a paso, asegurándonos de que nunca deje de ser un coloreo propio.
Como st es un lado => c(s) != c(t)
Asignamos c(s) = SALMON y c(t) = TURQUESA y de esta manera el lado st mantiene lo propio del coloreo.

Dado un j en {1, …, m} Como $D_j(b) = 1$ entonces existe un kj tal que $l_{j(kj)}(b) = 1$. Si hay más de uno tomamos uno solo. Ahora coloreamos:

$c(a_{j(kj)})$ = TURQUESA
y para j != kj, $c(a_{jr})$ = uno SALMON y el otro NEGRO

De esta forma los “triángulos de a’s” utilizan los 3 colores i.e mantienen lo propio del coloreo.

Seguimos con
$c(e_{j(kj)})$ = NEGRO
y para j != kj, $c(e_{jr})$ = TURQUESA

de esta forma el lado $a_{j(kj)}e_{j(kj)}$ mantiene lo propio del coloreo (uno TURQUESA y otro NEGRO)
y los lados $e_{jr}a_{jr}$ para r != kj tambien mantienen lo propio del coloreo pues e es TURQUESA y los a serán SALMON o NEGRO.

Definimos ahora,

$c(u_i)$ = SALMON si $b_i = 1$ y NEGRO si $b_i = 0$
$c(w_i)$ = NEGRO si $b_i = 1$ y SALMON si $b_i = 0$

Revisemos los lados
$e_{jr}v(l_{jr})$ para r != kj mantienen lo propio del coloreo pues $c(e_{jr})$ = TURQUESA y $l_{jr} = w_{algo}$ o u_{algo} utiliza el color NEGRO o SALMON.

Nos queda ver $e_{j(kj)}v(l_{j(kj)})$, recordemos que $l_{j(kj)}(b) = 1$

Si $l_{j(kj)}$ es una variable, Existe i tal que $l_{j(kj)} = x_i$ y por lo tanto $v(l_{j(kj)}) = u_i$

Ahora, $1 = l_{j(kj)}(b) = x_i(b) = b_i$ => $c(u_i)$ = SALMON

como $c(e_{j(kj)})$ = NEGRO, este lado no crea problemas

Si $l_{j(kj)}$ es una negación de variable, Existe i tal que $l_{j(kj)} = \neg x_i$ y por lo tanto $v(l_{j(kj)}) = w_i$

Ahora, $1 = l_{j(kj)}(b) = \neg x_i(b) = 1 - b_i$ => $b_i = 0$ => $c(u_i)$ = NEGRO
Como w_iu_i es un lado, $c(w_i)$!= NEGRO y $c(e_{j(kj)})$ = NEGRO

por lo que el lado $e_{j(kj)}v(l_{j(kj)}) = e_{j(kj)}w_{j(kj)}$ mantiene lo propio del coloreo.

Luego B satisfacible => Existe un coloreo propio de A(B) = G con 3 colores => X(G) <= 3.

(<=) Asumimos X(G) <= 3, como G = A(B) tiene triángulos (K3) formados por a's, entonces X(G) >= 3. Esto nos indica que X(G) = 3

Sea c un coloreo propio de G con 3 colores, construiremos B = (b_1, …, b_n) tal que B(b) = 1

Definimos, para ello, b_i = 1 si c(u_i) = c(s) y b_i = 0 en caso contrario.

Veamos ahora que B(b) = 1, para ello, necesitamos que D_j(b) = 1 para todo j = 1, …, m y por lo tanto necesitamos que exista, para cada j, un kj tal que $l_{j(kj)}$(b) = 1

Fijamos un j en { 1, …, m }

Los triángulos formados por a's utilizan los 3 colores, por lo tanto
elegimos kj tal que **c($a_{j(kj)}$) = c(t)**

Como **$a_{j(kj)}e_{j(kj)}$** es un lado, **c($e_{j(kj)}$) != c(t)**
Como **s$e_{j(kj)}$** es un lado, **c($e_{j(kj)}$) != c(s)**
y como **ts** es un lado **c(t) != c(s)** => **c($e_{j(kj)}$) = 3er color**

$e_{j(kj)}$v($l_{j(kj)}$) un lado y **c($e_{j(kj)}$) = 3er color** => **c(v($l_{j(kj)}$)) != 3er color**
v($l_{j(kj)}$) = w_i o u_i para algún i => **tv($l_{j(kj)}$)** es un lado => **c(v($l_{j(kj)}$)) != c(t)**
Luego **c(v($l_{j(kj)}$)) = c(s)**

Si $l_{j(kj)}$ es una variable x_i entonces $l_{j(kj)}$(b) = x_i(b) = b_i y **v($l_{j(kj)}$) = u_i**
=> **c(v($l_{j(kj)}$)) = c(u_i) = c(s)**
=> $l_{j(kj)}$(b) = b_i = 1

Si $l_{j(kj)}$ es una negación de variable ¬x_i entonces $l_{j(kj)}$(b) = ¬x_i(b) = 1 - b_i
y **v($l_{j(kj)}$) = w_i**
=> **c(w_i) = c(v($l_{j(kj)}$)) = c(s)** y w_iu_i lado
=> c(u_i) != c(s)
=> b_i = 0
=> $l_{j(kj)}$(b) = 1 - b_i = 1

Por lo tanto X(G) => Existe b tal que B(b) = 1 => B es satisfacible.

Matrimonio3D es NP-completo

Teorema:
Matrimonio3D es NP-COMPLETO
Demostración:
Para demostrarlo, mostraremos que 3(-CNF)-SAT se reduce polinomialmente a Matrimonio3D y como 3-SAT es NP-Completo entonces Matrimonio3D también lo será.

Es decir, a partir de una instancia B con variables $x_1, \dots, x_n$ de la forma $B = D_1 \& \dots \& D_m$ con $D_j = l_{j1} \vee l_{j2} \vee l_{j3}$ construiremos en tiempo polinomial sobre n y m, un 3-hipergrafo A(B) = H tal que Existe un matching perfecto en H ⇔ B es satisfacible.

Construcción de H:

Vértices:
$X = \{a_{ij}\}_{ij} \cup \{s_j\}_j \cup \{g_k\}_k$
$Y = \{b_{ij}\}_{ij} \cup \{t_j\}_j \cup \{h_k\}_k$
$Z = \{u_{ij}, w_{ij}\}_{ij}$
para i = 1, …, n ; j = 1, …, m y k = 1, …, m(n-1)
De esta manera, $|X| = mn + m + m(n-1) = 2mn = |Y| = |Z|$

Lados: $E_0 \cup E_1 \cup E_2 \cup E_3$
$E_0 = \{ \{ a_{ij}, b_{ij}, u_{ij} \} \}_{ij}$
$E_1 = \{ \{ a_{i(j+1)}, b_{ij}, w_{ij} \} \}$
$E_2 = \{ \{ g_k, h_k, z \} : z \text{ en } Z \}$

y definimos $v(l_{jr}) = u_{ij}$ si existe i tal que $l_{jr} = x_i$
$= w_{ij}$ si existe i tal que $l_{jr} = \neg x_i$
con ello,
$E_3 = \{ \{ s_j, t_j, v(l_{jr}) \} \}$
para todos los lados, i = 1, …, n ; j = 1, …, m y k = 1, …, m(n-1) y r = 1,2,3

Veamos ahora si

Existe un matching perfecto en H ⇔ B es satisfacible

(=>) Asumimos que existe un matching M perfecto en H

Como M es perfecto, entonces cubre todos los vértices. En particular, debe cubrir a los vértices a_{ij} por lo tanto existe un lado L en E(M) tal que L en E_0 ó L en E_1

Si L en $E(M) \cap E_0$ => L es de la forma $\{ a_{ij}, b_{ij}, u_{ij} \}$. Como M es un matching (. : una biyección), el lado b_{ij} no puede aparecer en otro lado de E(M) => $\{ a_{i(j+1)}, b_{ij}, w_{ij} \}$ no está en E(M). Pero $a_{i(j+1)}$ **debe estar** en E(M) lo cual significa que $\{ a_{ij}, b_{ij}, u_{ij} \}$ para todo j = 1, …,m. Llamamos esta situacion el **“CASO 0”** para **i**.

Similarmente, si L en $E(M) \cap E_1$ => L es de la forma $\{ a_{i(j+1)}, b_{ij}, w_{ij} \}$. Como M es un matching (. : una biyección), el lado b_{ij} no puede aparecer en otro lado de E(M) => $\{ a_{ij}, b_{ij}, u_{ij} \}$ no está en E(M). Pero a_{ij} **debe estar** en E(M) lo cual significa que $\{ a_{i(j+1)}, b_{ij}, w_{ij} \}$ para todo j = 1, …,m. Llamamos esta situacion el **“CASO 1”** para **i**.

Definimos entonces un vector de asignación de variables $b = (b_1, \ldots, b_n)$ de la siguiente manera:
$b_i = 0$ si se da el **CASO 0** para i
$= 1$ si se da el **CASO 1** para i

Veamos ahora que $B(b) = 1$ y por lo tanto que B es satisfacible.

Notar que $B(b) = 1 \Leftrightarrow D_j(b) = 1$ para todo j=1,...,m
Fijando un j, $D_j(b) = 1 \Leftrightarrow$ Existe un r tal que $l_{jr}(b) = 1$

Como M es un matching perfecto, **debe** cubrir a los lados $\{ s_j, t_j, v(l_{jr}) \}$ para algún valor de $r = r_j$ pues son los únicos en donde aparecen los vértices s_j y t_j. Tomamos este valor r_j y veremos que $l_{j(rj)}(b) = 1$

Si $l_{j(rj)}$ es una variable x_i entonces $l_{j(rj)}(b) = x_i(b) = b_i$
entonces $v(l_{j(rj)}) = u_{ij} \Rightarrow \{ s_j, t_j, v(l_{j(rj)}) \} = \{ s_j, t_j, u_{ij} \}$ por lo que u_{ij} no puede estar en otro lado del matching => $\{ a_{ij}, b_{ij}, u_{ij} \}$ no está en el matching => **CASO 1** para **i** => $b_i = 1$
luego $l_{jr}(b) = x_i(b) = b_i = 1$.

Si $l_{j(rj)}$ es una negación de variable $\neg x_i$ entonces $l_{j(rj)}(b) = \neg x_i(b) = 1 - b_i$
entonces $v(l_{j(rj)}) = u_{ij} \Rightarrow \{ s_j, t_j, v(l_{j(rj)}) \} = \{ s_j, t_j, w_{ij} \}$ por lo que w_{ij} no puede estar en otro lado del matching => $\{ a_{i(j+1)}, b_{ij}, w_{ij} \}$ no está en el matching => **CASO 0** para **i** => $b_i = 0$
luego $l_{jr}(b) = x_i(b) = 1 - b_i = 1$.

(<=) Asumimos que B es satisfacible, debemos ver que existe un matching perfecto en H. Para ello, construiremos un matching M con $E(M) = F_0$ u F_1 u F_2 u F_3 y mostraremos que **cubre** a todos los vértices de H.

$F_0 = \{ \{ a_{ij}, b_{ij}, u_{ij} \} : j = 1, \ldots, m \text{ y } b_i = 0 \}$
$F_1 = \{ \{ a_{i(j+1)}, b_{ij}, w_{ij} \} : j = 1, \ldots, m \text{ y } b_i = 1 \}$
Es claro que todos los lados de F_0 y F_1 son disjuntos pues i no puede ser 0 y 1 a la vez.

Como $B(b) = 1 \Rightarrow D_j(b) = 1$ para todo j=1,...,m
=> Para todo j=1,...,m Existe un r_j tal que $l_{j(rj)}(b) = 1$
$F_3 = \{ \{ s_j, t_j, v(l_{j(rj)}) \} \}$
Es claro que son disjuntos entre sí pues hay un único r_j por cada j.
Veamos que los lados de F_3 son disjuntos con todos los lados anteriores:

Si no lo fueran, el problema lo causaría $v(l_{j(rj)})$ pues s y t no forman parte de ningún lado anterior.

Si $l_{j(rj)}$ es una variable x_i =>
$1 = l_{j(rj)}(b) = x_i(b) = b_i = 1$
=> el vértice u_{ij} no está en F_0 para ningún j
pero $v(l_{j(rj)}) = u_{ij}$
=> Todos los lados de F_3 donde $l_{j(rj)}$ es una variable son disjuntos con los de con F_0 y (obviamente) con los de F_1 pues no tienen posibilidad de coincidir.

Si $l_{j(rj)}$ es una negación de variable $\neg x_i$ =>
$1 = l_{j(rj)}(b) = \neg x_i(b) = 1 - b_i \Rightarrow b_i = 0$
=> el vértice w_{ij} no está en F_1 para ningún j (para ese i)
pero $v(l_{j(rj)}) = w_{ij}$
=> Todos los lados de F_3 donde $l_{j(rj)}$ es una negación variable son disjuntos con los de con F_1 y (obviamente) con los de F_0 pues no tienen posibilidad de coincidir.

Finalmente, definimos N = { z en Z : z **no está** cubierto por M }

| N | ? Veamos | Z - N | = | { z en Z : z **está** cubierto por M } |

F_3 cubre **m** vértices en Z, pues cubre a uno por cada j = 1, …, m
Si p = #{i : b_i = 0} y q = #{i : b_i = 1} entonces
F_0 cubre p*m vértices en Z
F_1 cubre q*m vértices en Z
Obs: p + q = n

Luego | Z - N | = m + p*m + q*m = (1 + p + q)*m = m*(n+1)
Por lo que |N| = |Z| - | Z - N | = 2mn - m*(n+1) = m * (2n - n - 1) = m(n-1)

Lo cual nos indica que existe una biyección f entre { 1, …, m(n-1) } y los lados de N

Definimos entonces F_2 = { {g_k, h_k, f(k)} } los cuales son **claramente** disjuntos de todos los lados anteriores pues g y h no estaban cubiertos, y f(k) en N => tampoco estaba cubierto. También son disjuntos entre sí pues f es una biyección.

De esta forma y como todos los conjuntos dados son disjuntos,
|E(M)| = $|F_0|$ + $|F_1|$ + $|F_2|$ +$|F_3|$
= pm + qm + m(n-1) + m
= nm + m(n-1) + m
= 2mn = |X| = |Y| = |Z|
Y por lo tanto M es un matching perfecto.