Teóricos examen final discreta 2, 29 o 31 de julio:

3-COLOR es NP-completo

Estrategia, ver que 3-SAT $<_p$ 3-COLOR

Entonces, queremos a partir de una instancia B de 3-SAT transformarla en forma polinomial a una instancia G de 3-COLOR tal que B satisfacible sii X(G) = 3

Como decidir si B es satisfacible no puede lograrse en forma polinomial, decir si X(G) = 3 tampoco lo sera.

Sean x1,...,xn las variables de B, $l_{i,j}$ los literales, $D_j = l_{1,j} + l_{2,j} + l_{3,j}$ y $B = D_1$ && … && D_j

Creamos G, los vértices:

s, t vértices especiales

2n vértices v_l uno por cada literal (es decir, por variable y negación)

6m vertices $\{a_{k,j}\}$ $\{e_{k,j}\}$con k =1,2,3 y j =1,...,m

⇒ Los vértices se crean en 2+2n+6m, es polinomial

Sean los lados:

El lado st

n lados $V_x V_{\bar{x}}$

2n lados tv_l por cada literal

3m lados $a_{1j}a_{2j}, a_{1j}a_{3j}, a_{3j}a_{2j}$

3m lados $a_{kj}e_{kj}$

3m lados se_{kj}

3m lados $e_{kj}v_{lkj}$

⇒ Lo construimos en 12m + 3n +1 ⇒ es polinomial

Denotamos, a partir de $b \in \{0,1\}^n$ llamamos:

x(b) el valor de la variable en b

l(b) el valor del literal en b

D(b) el valor de la disjunción en b

B(b) el valor de la expresion en b

Veamos entonces, que si B es satisfacible ⇒ X(G) = 3

Como B es satisfacible, existe $b \in \{0,1\}^n$ tq B(b) = 1. Debemos usarlo para hallar un coloreo propio con 3 colores

Iremos coloreando y chequeando

Coloreamos: $c(v_x) = x(b)$. Como $v_x v_{x'}$ forma lado, chequeamos:

$c(v_{x'}) = x'(b) = 1 - x(b) != x(b) = c(v_x)$, no forma problema

Coloreamos ahora, c(s) = 1 y c(t) = 2

⇒ el lado st, no forma problema y como c(t) = 2 y $c(v_l) \in \{0,1\}$ => tv_l no forma problema

Para colorear as y es usamos b, pues como $B(b) = 1 \Rightarrow$ para todo $j = 1,\dots,m$ existe k_j tq $l_{k_j}(b) = 1$. Y si hay más de uno, elegimos el primero
$\Rightarrow$ Coloreamos para cada j: $c(a_{k_j}) = 2$ y a los demás a_{rj} coloreamos uno con cada color disponible
Entonces, cada triángulo de as tienen los 3 colores y no forma problema
Coloreamos $c(e_{rj}) = 2$ $r != k_j$. $c(e_{kj}) = 0 \Rightarrow a_{rj}e_{rj}$ no forma problema y $e_{kj}a_{kj}$ tampoco
Como $c(s) = 1$ y $c(e_{ij}) \in \{0,2\} \Rightarrow se_{ij}$ no forma problema
Para $r != k_j$ tenemos $c(e_{rj}) = 2$ y $c(v_{l\ en\ rj}) \in \{0,1\}$, no forma problema los lados $e_{rj}v_{l\ en\ rj}$
Para el caso de k_j tenemos que $c(e_{kj}) = 0$ y $c(v_{l\ en\ kj}) = l_{kj}(b) = 1$ no forma problema
Entonces, coloreamos con 3 colores a G y es un coloreo propio, por lo tanto $X(G) <= 3$. Pero como G tiene ciclos impares dentro $\Rightarrow X(G) >= 3 \Rightarrow X(G) = 3$

Veamos la vuelta, si $X(G)=3 \Rightarrow$ B es satisfacible
Como tenemos que el nº cromático es 3 $\Rightarrow$ existe un coloreo c de 3 colores para G que es propio, debemos usarlo para definir un $b \in \{0,1\}^n$ tq $B(b) = 1$
$\Rightarrow$ Definimos $b_i = 1$ si $c(v_{xi}) = c(s)$ y $b_i = 0$ si $c(v_{xi}) != c(s)$
Ahora, debemos ver que b satisface a B
Sea $j \in \{1,\dots,m\}$. Como los a_{kj} formar un triángulos $\Rightarrow$ están los tres colores entre ellos, en particular, existe q tq $c(a_{qj}) = c(t)$
Veamos el color que puede tener e_{qj}:
Como $a_{qj}e_{qj}$ forma lado $\Rightarrow c(a_{qj}) != c(e_{qj})$
Como se_{qj} forma lado $\Rightarrow c(s) != c(e_{qj})$
Solo queda disponible ese tercer color que no tiene ni s ni t
Ahora veamos el color posible de $v_{l\ en\ qj}$
$e_{qj}v_{l\ en\ qj}$ forma lado $\Rightarrow c(e_{qj}) != c(v_{l\ en\ qj})$
$tv_{l\ en\ qj}$ forma lado $\Rightarrow c(t) != c(v_{l\ en\ qj})$
Solo le queda disponible el color de s $\Rightarrow c(v_{l\ en\ qj}) = c(s)$
Pero, el literal puede ser una variable o una negación
C1: existe i tq $l_{qj} = x_i$. Como $c(v_{l\ en\ qj}) = c(s) \Rightarrow c(v_{xi}) = c(s) \Rightarrow l_{qj}(b) = x_i(b) = b_i = 1$
C2: existe i tq $l_{qj} = x'_i$. Como $c(v_{l\ en\ qj}) = c(s) \Rightarrow c(v_{x'i}) = c(s)$
Cómo está el lado $v_{xi}v_{x'i} \Rightarrow c(v_{xi}) != c(v_{x'i}) = c(s) \Rightarrow c(v_{xi}) != c(s) \Rightarrow b_i = 0$
Entonces, $l_{qj}(b) = x'_i(b) = 1 - x_i(b) = 1 - b_i = 1-0 = 1 \Rightarrow l_{qj}(b) = 1$

En ambos casos $l_{qj}(b) = 1$, y como $j \in \{1,\dots,m\} \Rightarrow$ para todo j se cumple esto
Luego, como $D_j(b)= l_{1j}(b)|\ l_{2j}(b)|\ l_{3j}(b) = 1 \mid 1 \mid 1 = 1$ para todo j
$\Rightarrow B(b) = D_1(b)\ \&\&\ \dots\ \&\&\ D_m(b) = 1\ \&\&\ \dots\ \&\&\ 1 = 1$
Luego, B es satisfacible

Complejidad EK

Suponemos que si el lado xy está en el network ⇒ el lado yx no lo esta

Sabemos que complejidad EK = #Flujos intermedios * buscar y construir CA

Cómo buscamos y construimos los caminos utilizando BFS, entonces lo hacemos en $O(n)$

Sean $f_0,\dots,f_k,\dots$ los sucesivos flujos intermedios, queremos acotarlos

Definimos, para un lado xy en el network, decimos que el lado se vuelve crítico en el paso k+1 si ocurre alguno de los siguientes:

$f_k(xy) < c(xy)$ y $f_{k+1}(xy) = c(xy)$, es decir, se saturó en el paso k+1 ó

$f_k(xy) > 0$ y $f_{k+1}(xy) = 0$, es decir, se vació en la paso k+1

Ahora, un lado puede volverse critico muchas veces al correr el algoritmo, supongamos que la cantidad de veces que un lado puede volverse critico esta acotado por B. Como en cada construccion de un CA, al menos un lado se vuelve critico ⇒ cantidad total de flujos intermedios = $O(mB)$

Veamos que B = n

Supongamos que el lado xy se vuelve critico en el paso k y que vuelve a hacerse crítico en el paso r. Tenemos dos casos para el paso k

C1: Se satura en el paso k

Si en el paso k se satura ⇒ para el paso k+1 utilizo un f_k CA de longitud minima de forma s...xy...t ⇒ $d_k(y) = d_k(x) + 1$

Pero como se vuelve a saturar en el paso r, existe un $l > k$ tq se des-satura el flujo en el lado xy al pasar al flujo l+1

Entonces, para el paso l+1 uso un f_l CA de longitud minima de forma s...yx...y pues disminuye el flujo. ⇒ $d_l(x) = d_l(y) + 1$

Como las distancias no disminuyen:

$d_k(w) \le d_l(w)$ y $b_k(w) \le b_l(w)$ para todo vertice w del network

Entonces:

$d_k(t) = d_k(x) + b_k(x) = d_k(y) + b_k(x) +1 \le d_l(y) + b_l(x) + 1 = d_l(x) + b_l(y) + 2$

⇒ $d_k(t) \le d_l(y) + 2$

Por lo tanto, la distancia a t entre dos flujos que vuelven critico y dejan de hacer critico un lado aumenta en por lo menos 2

C2: Se vacía en el paso k

Si en el paso k se vacia ⇒ para el paso k+1 utilizo un f_k CA de longitud minima de forma s...yx...t ⇒ $d_k(x) = d_k(y) + 1$

Pero como se vuelve a saturar en el paso r, existe un $l > k$ tq se des-vacia el flujo en el lado xy al pasar al flujo l+1

Entonces, para el paso l+1 uso un f_l CA de longitud minima de forma s...xy...y pues aumenta el flujo. ⇒ $d_l(y) = d_l(x) + 1$

Como las distancias no disminuyen:
$d_k(w) <= d_l(w)$ y $b_k(w) <= b_l(w)$ para todo vértice w del network

Entonces:
$d_k(t) = d_k(y) + b_k(y) = d_k(x) + b_k(y) +1 <= d_l(x) + b_l(y) + 1 = d_l(y) + b_l(y) + 2$
$\Rightarrow d_k(t) <= d_l(y) + 2$
Por lo tanto, la distancia a t entre dos flujos que vuelven crítico y dejan de hacer crítico un lado aumenta en por lo menos 2

En ambos casos, la distancia de s a t aumenta en por lo menos 2, ahora, como la distancia entre s y t puede ir desde 1 a n-1 $\Rightarrow$ Un lado puede volverse crítico a lo sumo n veces $\Rightarrow$ B=n
Y complejidad EK = $O(nm^2)$

Lema distancias
Ver que las distancias no disminuyen en los flujos sucesivos de EK
Prueba:
Definimos: Dado un flujo f y un vértice z, decimos que un vértice x es un vecino f-FF de z si pasa alguno de los siguientes:
        $xz \in E$ y $f(xz) < c(xz)$ ó $zx \in E$ y $f(zx) > 0$
[redacted]
Sean f_0,... etc flujos sucesivos al correr EK
Sean $A = \{x : d_k(x) > d_{k+1}(x) \}$
Como $d_k(s) = d_{k+1}(s) = 0 \Rightarrow$ s no pertenece a A
Suponemos A != Cjto vacío
Sea $x \in A$ tq $d_{k+1}(x) = Min \{d_{k+1}(y): y \in A\} \Rightarrow d_k(x) < d_{k+1}(x)$
Como $d_{k+1}(x)<$ INF $\Rightarrow$ existe un CA de s a x de long mínima
Como x!=s $\Rightarrow$ el camino no consta solo de x
Luego, existe z != x el vértice inmediato anterior a x en el CA de s a x de long mínima. Por lo tanto, z está en un CA de long mínima de s a z.
Como z es anterior al x $\Rightarrow d_{k+1}(z) = d_{k+1}(x) - 1$, por lo tanto $d_{k+1}(z) < d_{k+1}(x)$, entonces z no pertenece a A. Por lo tanto: $d_k(z) <= d_{k+1}(z)$
Entonces, $d_k(x) < d_{k+1}(x) = d_{k+1}(z) + 1 <= d_k(z) + 1 \Rightarrow d_k(x)> d_k(z) + 1$
Por lo tanto, x no es fk-FF vecino de z(1), pero x si es fk+1-FF vecino de z(2)
Tenemos dos casos, $xz \in E$ o $zx \in E$
C1: $xz \in E$
        por 1: $f_k(xz) = 0$
        por 2: $f_{k+1}(xz) >= 0 \Rightarrow f_{k+1}(xz) > f_k(xz) \Rightarrow$ enviamos flujo primero por x y luego por z
C2: $zx \in E$
        por 1: $f_k(zx) = c(zx)$

por 2: $f\_k+1(zx) <= c(zx)$ ⇒ $f\_k+1(zx) <= f\_k(zx)$ ⇒ devolvimos flujo, primero por x y luego por z

En ambos casos cuando paso del f_k flujo al f_k+1 flujo, primero pasó por x y luego pasó por z ⇒ el camino tiene la forma s...xz...t ⇒ d_k(z) = d_k(x) + 1 y teniamos d_k(x) > d_k(z) + 1

⇒ ahora tengo: d_k(x) > d_k(z) + 1 = d_k(x) + 1 + 1 ⇒ d_k(x) > d_k(x) + 2 ABSURDO

luego, A = VACÍO

Distancias en NA aumenta

Sean NA un network auxiliar y NA' el network auxiliar siguiente.

Sean d y d' las distancias FF de s a algún vértice en el network original al momento de construir NA y NA' respectivamente

Sabemos por EK que d(t) <= d'(t), veamos que solo vale <

Si t no esta en NA' ⇒ d'(t) = INF y vale

Asumimos que t esta en NA', por lo tanto, existe un camino dirigido s=x0,...,xr=t en NA'. Este camino no puede estar en NA pues si lo estuviera habría al menos un lado saturado en el network original al momento de construir NA' y no podría ser camino en NA'.

Pero si el camino no esta en NA puede ser por dos razones:

1: Falta algún vértice xi

2: Están todos los vértices pero falta un lado xixi+1

1) Como NA fue construido utilizando BFS ⇒ en NA están todos los vértices con distancia a s menor que la distancia de t a s

Asique si xi no esta en NA es pues d(xi) >= d(t)

Tengo en NA' el camino s...xi...t y como es un network por niveles ⇒ d'(xi)=i y d'(t)=r y cómo xi != t => i < r

Entonces: d(t) <= d(xi) <= d'(xi) = i < r = d'(t) ⇒ d(t) < d'(t)

2) Están todos los vértices pero falta algún lado XiXi+1

Sabemos por EK que d(Xi+1) <= d'(Xi+1)

Caso d(Xi+1) < d'(Xi+1) :

Sean b y b' las distancias de un vértice a t al momento de construir NA y NA' respectivamente tq d(t) = d(y) + b(y) para todo y que esté en el network auxiliar

⇒ d(t) = d(Xi+1) + b(Xi+1) <= d(Xi+1) + b'(Xi+1) < d'(Xi+1) + b'(Xi+1) = d'(t)

Caso d(Xi+1) = d'(Xi+1) = i+1:

Como i es el primer índice para el cual el lado no esta en NA ⇒ el camino s..Xi si esta en NA y es de long mínima y NA es network por niveles ⇒ d(Xi) = i

Tenemos que estan los vertices y las distancias también están bien, no esta el lado XiXi+1 y en particular tampoco está el lado Xi+1Xi y esto pasa solo si al construir NA ocurre alguna de las siguientes :

XiXi+1 está saturado al construir NA pero se des-satura al construir NA'

Xi+1Xi está vacío pero se des-vacía al construir NA'

En ambos casos, como NA’ es el network siguiente a NA para poder usar el lado en NA’ tengo que haber enviado flujo o devuelto flujo al momento de construir NA y eso es un absurdo.

Complejidad Dinic-Even y Dinitz

Sabemos que en ambos casos Complejidad = Cantidad NA * Compl paso bloqueante

Como la distancias en NA consecutivos aumenta y la distancia entre s y t puede ir desde 1 a n-1 ⇒ Cantidad NA = O(n)

Denotamos:

Avanzar(x) : A

Retroceder(x) : R

Incrementar + inicializar : I

Tanto A como R son O(1) pues A solo elije el primero de una lista y R borra un lado y retrocede al anterior de la lista. En cambio, I es O(n) pues busca el mínimo de una lista que puede tener a los sumo n elementos y luego los recorre para inicializar.

Veamos en el paso bloqueante

Dinic-Even : Correr el algoritmo nos da una sucesión de As,Rs,Is

Podemos agrupar la sucesión en palabras de la forma A...AX con X=R o X=I

Como A mueve el extremo de la cola por la que avanzó, puedo tener a lo sumo n avanzar. ⇒ Complejidad una A...AX = O(n+n) si X=R ó O(n+1) si X = R

Como tanto I como R borran lados puedo tener a lo sumo m palabras A...AX

⇒ Complejidad total A...AX = O(nm) = Complejidad paso bloqueante Dinic-Even

En cambio, para Dinitz no tenemos R, sino tenemos Podar: P, y separamos Incrementar: I ϵ inicializar : i

En este caso, podemos agrupar la sucesión en palabras de forma: A...AIPiA...AIPi…

Como en Dinic, complejidad de A...AI = O(nm) pues puedo tener a lo sumo n As y por cada I se borra al menos un lado

Como hay igual cantidad de i que de I y complejidad de una i es O(1)

⇒ Complejidad de is = O(m)

Para analizar los P veamos bien que hace, podar tiene dos partes, una en las que chequea que los vértices tienen líneas de salida (PV) y luego, si las tiene los deja, si no los borra B(X).

PV recorre todos los vértices y hay un PV por cada i ⇒ Compl. total PVs = O(nm)

Cada B(x) tiene complejidad δ(x). Pero una vez que se activa para algún vértice no se vuelve a activar más pues borra los lados del mismo, por lo tanto, la complejidad de todos los B(x) es la suma de los δ de los vértices ⇒ Complejidad total de B(x) = O(m)

Entonces, concluimos:

Complejidad total de A...AI : O(nm)

Complejidad total de is : O(m)

Complejidad total de PVs: O(nm)

Complejidad total de B(x) : O(m)
Complejidad total del paso bloqueante: O(nm)
Complejidad total de Dinitz: $O(n^2m)$

Complejidad Wave
Suponemos que si xy es lado ⇒ yx no lo es
Sabemos Complejidad de Wave = #NA * Complejidad paso bloqueante
Como la cantidad de NA esta acotada por n ⇒ #NA = O(n)
Veamos que la complejidad del paso bloqueante es $O(n^2)$
En cada ola se hacen una balanceos que consta de revisar una series de vértices de $\Gamma^+(x)$ o $\Gamma^-(x)$ según sea el caso y hacer una serie de procesamientos que son O(1)
Veamos la cantidad de procesamientos que se realizan. Los separamos en categorías
Si procesamos el lado xy enviando flujo de x a y puede que pase:
1. Lo procesamos y se satura
2. No se satura
Si procesamos el lado yx devolviendo flujo de x a y puede que pase
3. Lo procesamos y se vacía
4. No se vacía
Sean:
S = procesamientos de tipo 1 sobre todas las olas hacia adelante
P = procesamientos de tipo 2 sobre todas las olas hacia adelante
V = procesamientos de tipo 3 sobre todas las olas hacia atrás
Q = procesamientos de tipo 4 sobre todas las olas hacia atrás
⇒ Complejidad del paso bloqueante= P+Q+V+S
Calculemos cada uno:
En cada BalanceoHaciaAdelante(x) enviamos hacia los vecinos de x todo lo que podemos, es decir, Min{D(x), g(xy)-c(xy)}. Si enviamos g(xy)-c(xy) el lado se satura
Entonces, al menos un vértice no se satura pues se le envía D(x).
Por lo tanto, en cada BalanceoHaciaAdelante(x) salvo a lo sumo un vértice, se satura => P acotado por la cantidad de BalanceoHaciaAdelante(x)
En cada ola hacia adelante realizo n-2 BalanceoHaciaAdelante(x), y si en una ola no se bloquea ningun vertice es porque fue la ultima ola y el algoritmo termina y todos los vértices quedaron balanceados.
Como, en cada ola hacia adelante, menos la ultima, al menos un vertice se bloquea y no se vuelve a desbloquear, por lo tanto tengo a lo sumo n olas
⇒ Tengo a lo sumo n olas y n-2 BalanceoHaciaAdelante(x) por ola
⇒ $P <= n^2$

Con un razonamiento similar, como tengo la misma cantidad de olas hacia adelante como hacia atras y en cada BalanceoHaciaAtras(x) salvo a lo sumo un lado no se vacía y tengo n-2 BalanceoHaciaAtras(x) por ola

$\Rightarrow Q <= n^2$

Calculemos S y V

Como si el lado xy se satura no se vuelve a saturar y tengo m lados $\Rightarrow S = O(m)$

Esto pues, para que el lado xy se vuelva a saturar 'y' le debe devolver flujo a 'x', y para que 'y' devuelva flujo debe estar bloqueado, pero si 'y' esta bloqueado, entonces 'x' no le puede enviar flujo, por lo tanto, no se vuelve a saturar.

Analogamente, si yx se vacia, no se vuelve a vaciar, $\Rightarrow V = O(m)$

Esto pues si yx esta vacio es porque 'x' le devolvio flujo a 'y', y solo le puede devolver flujo si 'x' esta bloqueado, pero para que 'x' este bloqueado 'y' le debe enviar flujo pero el lado esta vacio.

Entonces, concluimos:

Complejidad Paso bloqueante = S +P +V +Q = $O(m) + O(n^2) + O(m) + O(n^2) = O(n^2)$

Complejidad Wave = $O(n^3)$

MAX FLOW MIN CUT

Probar que si f es un flujo, las siguientes afirmaciones son equivalentes:

i) f es maximal.

ii) Existe un corte S tal que v(f) = cap(S). (y en este caso, S es minimal)

iii) No existen f-caminos aumentantes.

(puede usar sin necesidad de probarlo que el valor de todo flujo es menor o igual que la capacidad de todo corte)

Prueba:

Estrategia: 1⇒ 3⇒ 2⇒ 1

<u>1⇒ 3:</u>

Por absurdo, pues si existiese un f-CA ⇒ podríamos enviar ε a través de él y obtener un flujo f' tq v(f') = v(f) + ε y es un absurdo pues f es maximal

<u>2⇒ 1:</u>

Sea g un flujo sobre el network. Como el valor de todo flujo es menor igual a la capacidad de todo corte ⇒ v(g) <= cap(S), pero cap(S)=v(f)

⇒ v(g)<=v(f) ⇒ f es maximal

<u>3⇒ 2</u>

Sea S = {s} U {x: existe un CA de s a x}

Como no hay un f-CA ⇒ t no pertenece a S y S es un corte pues si esta s

Veamos que v(f) = cap(S)

Por propiedades de flujo tenemos:

v(f) = c(S,S') - c(S',S)

c(S,S') = $\sum_{x,y} f(x,y)[x \in S][y \in S'][xy \in E]$

Sean, x,y de la suma:

como x ϵ S ⇒ hay un camino aumentante de la forma s…x

como y ϵ S’ ⇒ no hay un camino de s a y, en particular no esta el camino s...xy
Pero xy si es un lado del network ⇒ Si no esta el camino hasta y solo puede ser si f(xy) = c(xy), es decir el lado está saturado

⇒ c(S,S’) = $\sum_{x,y} f(x,y)[x \in S][y \in S'][xy \in E]$

c(S,S’) = $\sum_{x,y} c(x,y)[x \in S][y \in S'][xy \in E]$ = cap(S)
Ahora, el otro sumando

c(S’,S) = $\sum_{x,y} f(x,y)[x \in S'][y \in S][xy \in E]$
Sean, x,y de la suma:
como y ϵ S ⇒ hay un camino aumentante de la forma s…y
como x ϵ S’ ⇒ no hay un camino de s a y, en particular no está el camino s...yx
Pero xy si es un lado del network ⇒ Si no está el camino hasta y solo puede ser si f(xy) = 0, es decir el lado está vacío.

⇒ c(S’,S) = $\sum_{x,y} f(x,y)[x \in S'][y \in S][xy \in E]$ = 0

⇒ v(f) = cap(S) - 0 = cap(S)

Además, si T es otro corte, como la capacidad de todo flujo es menor o igual a la capacidad de todo corte ⇒ cap(s) = v(f) <= cap(T) ⇒ S es minimal

Cota de hamming
Sea C un código de longitud n, y δ(C)=δ y t = floor(δ-1/2)

⇒ #C <= 2^n / $\sum_{r=0}^{t} (n\ en\ r)$
Prueba:

Sea A = $\bigcup_{v \in C}$ $D_t(v)$. Como vimos en el teorema fundamental de códigos, el código corrige t errores, entonces la unión es disjunta ⇒ #A = $\sum_{v \in C}$ $\#D_t(v)$
Para calcular la cardinalidad del conjunto definimos:

$S_r(v) = \{w \in \{0,1\}^n : d_H(w,v) = r\}$ ⇒ $D_t(v) = \bigcup_{r=0}^{t} S_r(v)$
Como las distancias entre dos palabras no puede al mismo tiempo ser distinta, ⇒ la unión es disjunta, por lo tanto $\#D_t(v) = \sum_{r=0}^{t} \#S_r(v)$
Para calcular $\#S_r(v)$ notemos que si una palabra pertenece a ese conjunto es pues difiere de v en exactamente r de los n bits posibles, por lo tanto podemos definir un subconjunto de {1,...,n} con los índices donde w_i != v_i

Entonces, definimos I_r = {z ⊆ {1,...,n} tq #z = r}
⇒ definimos las funciones:
psi : $S_r(v)$ -> I_r tq psi(w) --> {i : w_i != v_i}
y
fi : I_r -> $S_r(v)$ tq fi(z) --> el único w tq $w_i = v_i$ si i ϵ z y $w_i = 1 + v_i$ si i no pertenece a z
Entonces, fi y psi son la inversa una de otra
⇒ $\#S_r(v) = \#I_r$ = (n en r) ⇒ $\#D_t(v) = \sum_{r=0}^{t}$ (n en r)
Como ⇒ #A = $\sum_{v \epsilon C} \sum_{r=0}^{t}$ (n en r) = #C * $\sum_{r=0}^{t}$ (n en r)
⇒ # C = $\frac{\#A}{\sum_{r=0}^{t} (n\ en\ r)}$ Pero, #A <= 2^n ⇒ #C <= $\frac{2\hat{}n}{\sum_{r=0}^{t} (n\ en\ r)}$

Matriz de chequeo y columnas LD
Probar que si H es matriz de chequeo de C, entonces;
δ(C) = Min{j : ∃ un conjunto de j columnas LD de H}
Prueba:
Denotamos H^i la i-ésima columna de H
Sean {H^{i1} ,H^{i2} … ,H^{is}} cjto de columnas LD de H
Entonces, existen c_1,...,c_s no todos nulos tq la combinacion lineal con las columnas del conjunta me da cero.
Sea w = $c_1e^{i1}+...+c_se^{is}$ con e^i el vector todo nulo menos la i-ésima posición
Hacemos $H*w^t = H (c_1e^{i1}+...+c_se^{is})^t = c_1H^{i1}+...+c_sH^{is}$ = 0, por lo tanto w ϵ C
Como w != 0 pues no todos los cs eras nulos y δ(C) = Min{|v| : v != 0 y v ϵ C}
⇒ δ(C) <= |w| <= s. Sea m = Min{j : ∃ un conjunto de j columnas LD de H} ⇒ δ(C) <= m
Veamos la otra desigualdad, sea v ϵ C tq |v| = δ(C)
⇒ existen $e^{i1},...,e^{i\delta(C)}$ tq v = $e^{i1}+...+e^{i\delta(C)}$ y como v ϵ C ⇒ H v^t = 0, pero
H $(e^{i1}+...+e^{i\delta(C)})^t$ = $H^{i1}+...+H^{i\delta(C)}$ = 0, por lo tanto el cjto de esas columnas es LD
⇒ m <= δ(C)
Luego m = δ(C)

Fundamental de cíclicos
Sea C un código cíclico de dimensión k y longitud n y sea g(x) su polinomio generador. Probar que:
i) C está formado por los múltiplos de g(x) de grado menor que n: C = {p(x) : gr(p) < n&g(x)|p(x)}
ii) C = {v(x) ⊙ g(x) : v es un polinomio cualquiera}
iii) gr(g(x)) = n − k.
iv) g(x) divide a 1 + x^n

Prueba:
1 y 2:
Sean $C1 = \{p(x) : gr(p) < n \& g(x)|p(x)\}$
$C2 = \{v(x) \odot g(x) : v \text{ es un polinomio cualquiera}\}$
Por propiedades de rotación sabemos que $C2 \subseteq C$
Sea $p(x) \in C \Rightarrow gr(p) < n$
Dividimos p(x) por g(x) $\Rightarrow p(x) = q(x)g(x) + r(x), gr(r) < gr(g) < n$
Cómo $gr(r) < n \Rightarrow r(x) = r(x) \bmod 1+x^n$ y $r(x) = q(x)g(x) + p(x)$
$\Rightarrow r(x) = q(x)g(x) + p(x) \bmod 1+x^n$
$= q(x)g(x) \bmod 1+x^n + p(x) \bmod 1+x^n$
$= q(x)\odot g(x) + p(x)$ y ambos sumandos pertenecen al código, como es lineal $\Rightarrow$ r(x) está en el código. Pero cómo $gr(r) < gr(g) \Rightarrow r(x) = 0$.
$\Rightarrow p(x) = q(x)g(x) = q(x)\odot g(x) \Rightarrow$ p(x) está en C1 y p(x) está en C2
$\Rightarrow C1 = C = C2$
3: Sean $t = gr(g)$. Sea $p(x) \in C \Rightarrow gr(p) < n$
Por 1, $p(x) = q(x)g(x) \Rightarrow gr(q*g) < n$
$\Rightarrow gr(q) < n-t$
Por lo tanto, para cada elemento del código se corresponde un polinomio de grado menor a n-t.
$\Rightarrow \#C = 2^k$ pues es un código lineal, y al mismo tiempo, por la afirmación de arriba $\#C = 2^{(n-t)}$
$\Rightarrow k = n-t \Rightarrow t = gr(g) = n-k$

4: Dividimos $1+x^n$ por g(x) $\Rightarrow 1+x^n = q(x)g(x) + r(x), gr(r) < gr(g) < n$
Cómo $gr(r) < n \Rightarrow r(x) = r(x) \bmod 1+x^n$ y $r(x) = q(x)g(x) + 1+x^n$
$\Rightarrow r(x) = (q(x)g(x) + 1+x^n) \bmod 1+x^n$
$= (q(x)g(x) \bmod 1+x^n) + (1+x^n \bmod 1+x^n)$
$= q(x)\odot g(x) + 0$ y $\in C \Rightarrow r(x) \in C$. Pero como $gr(r) < gr(g) \Rightarrow r(x) = 0. \Rightarrow 1+x^n = q(x)g(x)$, por lo tanto, $g(x)|(1+x^n)$

Teorema Hall
Sea G un grafo bipartito, con partes X e Y.
Si todo subconjunto S de X cumple $|S| <= |\Gamma(S)| \Rightarrow$ hay un matching completo sobre X
Prueba:
Por absurdo, supongamos $|S| <= |\Gamma(S)|$ pero no hay matching completo
Sea S_0 subcjto de X con los vértices de X que no fueron alcanzados por el matching
Si $x \in S_0$ sii no existe y en Y tq xy sea lado del matching $\Rightarrow f(xy)=0$
$\Rightarrow out_f(x) = 0 = in_f(x) \Rightarrow f(sx)=0$
Viceversa, si $f(sx)=0 \Rightarrow in_f(x)=0=out_f(x) \Rightarrow$ no hay y en Y tq xy sea lado del matching
$\Rightarrow S_0=\{ x \in X : out_f(x)=0=in_f(x) \}$

Sea C el corte obtenido en la ultima iteración al correr EK
Como C corte ⇒ s e C y los demás elementos son de X o de Y
Sean S subcjto de X y T subcjto de Y tq C = {s} U T U S
Ahora, como los elementos de S_o no estan el matching ⇒ están en la cola, por lo tanto S_0 incluido S. Veamos los demás elementos de S que no son de S_0
No los pudo haber agregado s, entonces si o si fueron agregados por elementos de T. Pero las salidas de los elementos de T son solo {t}, asi que solo pueden haber agregado por lado backward.
Si f(yt) > 0 ⇒ f(yt) = 1 ⇒ existe x en X tq f(xy) = 1.
Por lo tanto cada elemento de T puede agregar a lo sumo un elemento de X a la cola.

Veamos que si o si agrega uno:
Si y ϵ T y no agrega a nadie a C
Como f(yt)=1, pero no puede agregar a t, lo cual esta bien. Pero si f(yt)=1⇒ $out_f(y)=1=in_f(y)$ ⇒ existe x ϵ X tq f(xy)=1. Ahora, x es candidato a ser agregado a la cola. Pero sin embargo, y no lo agrega, por lo tanto lo agrega alguien mas. No lo puede haber agregado ningun otro elemento de T pues no puede que f(xy)=1 y f(xz)=1 al mismo tiempo.
Entonces, si no lo agrega nadie de T, solo lo puede agregar s ⇒ x ϵ S_0 y $out_f(x) = 0$ ABSURDO

Luego, los elementos de T si o si agregan un elemento a la cola, entonces
los elementos que agrega T son los de $S\text{-}S_0$ ⇒ $|T| = |S\text{-}S_0|$

Pero veamos un poco más a los elementos de T
Estos elementos fueron agregados por vértices de S ⇒ T incluido Γ(S)
Veamos que son iguales
Sea y e Γ(S), pero y no esta en T ⇒ y no esta en C
Pero como y ϵ Γ(S) ⇒ existe x ϵ S tq xs es un lado del grafo. Pero x no agrego a y a la cola ⇒ f(xy)=1
Veamos quien agrego a x, pues y no lo pudo haber agregado porque no estaba en la cola, entonces lo tiene que haber agregado s, pero si lo agrego s ⇒ x ϵ S_0 y $out_f(x)=0$ ABSURDO
Luego T = Γ(S)
⇒ $|T| = |\Gamma(S)| = |S - S_0| < |S|$ pues S_0 no es vacio ⇒ $|\Gamma(S)| < |S|$ ABSURDO por hipotesis.
Luego, S_0 es vacío y el matching es completo sobre X

Matrimonio de König
Todo grafo bipartito regular tiene un matching perfecto
Prueba:
Sean X,Y las partes del grafo

Sea W subcjto de vértices del grafo y E_W={wz ϵ E: w ϵ W}

Si W ⊆ X ⇒ $|E_w|$ = |{wz ϵ E: w ϵ W}| = $\sum_{w \in W}$ |{z : zw ϵ E}| = $\sum_{w \in W} \delta(w)$

Pero como el grafo es regular ⇒ δ(z) = Δ ⇒ $|E_w|$ = $\sum_{w \in W} \Delta$ = |W| * Δ

Si W ⊆ Y, es análogo.
Hace falta que sea un subcjto de alguna de las partes, asi no cuento dos veces el mismo lado.
Pero ahora, si W = X ⇒ $|E_X|$ =|{xz ϵ E: x ϵ X}| = |E| = Δ * |X|
Y si W = Y ⇒ $|E_Y|$ =|{zy ϵ E: y ϵ Y}| = |E| = Δ * |Y|
⇒ |Y| = |X|
Ahora, como las partes tienen la misma cantidad de vértices ⇒ solo me basta encontrar un matching completo para alguna de las partes para tener un matching perfecto. Lo hacemos mediante el Teorema de Hall.
Sea S⊆ X subcjto cualquiera de X y sea l ϵ E_S , l de la forma xy con x ϵ S, y ϵ Y
Como y ϵ Γ(x)⊆Γ(S) ⇒ l de la forma xy con x ϵ S, y ϵ Γ(S) ⇒ l ϵ E_Γ(S) ⇒ E_S⊆E_Γ(S)
⇒ |E_S|<=|E_Γ(S)|
Aplicamos la propiedad de arriba, pues S ⊆ X y Γ(S) ⊆ Y
⇒ |E_Γ(S)| = |Γ(S)| * Δ y |E_S| = |S| * Δ
⇒ |S| * Δ <= |Γ(S)| * Δ ⇒ |S| <= |Γ(S)| y por teo de hall, como S era un subcjto de X cualquiera, hay matching completo para X, luego hay un matching perfecto para el grafo.