FaMAF

Universidad Nacional de Córdoba

Matemática Discreta II

Apuntes de clases

Profesor: Daniel Penazzi

2016

Por favor, mejorá este documento en github
https://github.com/mmodenesi/discreta_ii_famaf

Generalidades de notación

Salvo que se diga explícitamente otra cosa, al referirnos a grafos usamos las letras

- n para referirnos al número de vértices o nodos
- m para referirnos al número de lados o aristas

Coloreo

Un coloreo de $G = (V, E)$ es una función $C : V \to S$ donde S es algún conjunto. Este conjunto de llegada S generalmente es

$$\{1, 2, \ldots, n\} \text{ ó } \{0, 1, \ldots, n-1\}$$

Coloreo propio

Un coloreo C de un grafo $G = (V, E)$ se dice propio si

$$(v, w) \in E \Rightarrow C(v) \neq C(w)$$

Es decir, un coloreo que asigna elementos distintos a vértices adyacentes.

Número cromático

El número cromático de un grafo $G = (V, E)$, denotado $\chi(G)$ el mínimo $k \in \mathbb{N}$ tal que existe un coloreo propio de G con k colores.

El número cromático tiene algunas cotas obvias:

$$1 \leq \chi(G) \leq n$$

Tenemos interés por aquellos algoritmos que nos permiten calcular el número cromático de un grafo. El algoritmo más obvio, sería aquel que prueba los n^n coloreos (cada uno de los n vértices puede recibir n colores distintos), separar los coloreos propios y buscar el mínimo número de colores necesarios. Este algoritmo no es eficiente.

Problema de decisión

Un problema de desición es un problema cuyas únicas respuestas posibles son "síz "no"

P

P es una clase de problemas de decisión para los cuales existe un algoritmo determinístico que los resuelve en tiempo polinomial (tomando algún parámetro relevante de la entrada).

k-color

Dado un grafo G, k-color es un problema de decisión que consiste en responder a la pregunta "¿Es $\chi(G) \leq k$?"

Observaciones:

1-color $\in P$, ya que un algoritmo polinomial que lo resuelve consiste en chequear si $m = 0$. Si G tiene al menos un vértice, la respuesta a 1-color es "NO".

No sabemos si **3-color** $\in P$.

2-color $\in P$. Para probar esto, debemos dar un algoritmo polinomial que resuelva el problema, mostrar que es correcto y que es polinomial.

```
procedure TWO_COLOR
    j = 0 // número de vértices coloreados

    while j < n: // cantidad de vértices en el grafo
        x = v ∈ V tal que v no está coloreado
        C(x) = 1
        j++
        Q = cola con x como su único elemento
        while Q ≠ ∅:
            p = remover primer elemento de Q
            foreach w ∈ Γ(p):
                if w no está coloreado:
                    meter w en Q
                    C(w) = 3 − C(p)
                    j++
    // controlar que el coloreo es propio
    foreach {v, w} ∈ E:
        if C(v) == C(w):
            return False
    return True
```

Prueba de complejidad polinomial. El **while** interno recorre todos los vértices de la componente conexa de x por lo tanto el **while** externo se repite una vez por cada componente conexa. Lo que está apenas después del **while** externo es $O(1)$. El **while** interno se repite 1 vez por cada componente conexa de x (que abreviaremos $CC(x)$). Dentro de este **while**, tenemos un **for** que se repite tantas veces como el grado del vértice p, es decir que es $O(d(p))$. Luego, la complejidad del **while** interno es

$$O(\sum_{p \in CC(x)} d(p)) = O(2 \text{ \# aristas en} CC(x)) = O(\text{\# de aristas en } CC(x))$$

Entonces, la complejidad de la primera parte del algoritmo está en $O(m)$. En cuanto a la segunda parte, el **foreach** es $O(m)$. Por lo tanto, este algoritmo es polinomial.

Prueba de correctitud

Si la respuesta del algoritmo es que efectivamente el grafo es 2-coloreable, claramente es correcto, porque sólo la da si revisó todos los lados y chequeó que todos los vértices adyacentes tienen diferente color.

Si en cambio dice que no es 2-coloreable, entonces es por que existe un vértice $\{v, w\}$ tal que $C(v) = C(w)$.

Como v, w están en la misma componente conexa, sea x la raíz"de esta componente y Q la cola que se construye a partir de x. Supongamos que v entró primero a la cola (sin pérdida de generalidad). Cuando v pasó a ser el primer elemento de la cola, w debe estar ya en la cola Q. Si no estuviera, como $\{v, w\}$ es un vértice, v debería agregar a w con el color $3 - C(v)$ y por lo tanto $C(v) \neq C(w)$, lo cual contradice nuestra hipótesis. Entonces sabemos que cuando v es el primer elemento de la cola, w ya ha sido agregado por otro vértice. Imaginemos las cadenas

$$x = v_r \to \text{agregó a } \to v_{r-1} \to \text{agregó a } \to \cdots \to \text{agregó a } \to v_1 \to \text{agregó a } \to v_0 = v$$

y

$$x = w_t \to \text{agregó a } \to w_{t-1} \to \text{agregó a } \to \cdots \to \text{agregó a } \to w_1 \to \text{agregó a } \to w_0 = w$$

Cuyos colores deben ser

$$\underset{(1)}{x = v_r} \quad \to \quad \underset{(2)}{v_{r-1}} \quad \to \cdots \to \quad v_1 \quad \to \quad v_0 = v$$

y

$$\underset{(1)}{x = w_t} \quad \rightarrow \quad \underset{(2)}{w_{t-1}} \quad \rightarrow \cdots \rightarrow \quad w_1 \quad \rightarrow \quad w_0 = w$$

Como v entró antes, debe ser $r \leq t$, pero como w ya está en Q cuando v es el primer elemento de Q, debe ser $t \leq r + 1$. Por otro lado, como el color depende de la paridad de r y t, el hecho de que $C(v) = C(w)$ implica que t y r tienen la misma paridad, es decir $t \equiv r(2)$. Ahora, por

$$r \leq t, \quad t \leq r + 1 \quad \text{y} \quad t \equiv r(2)$$

debe ser $t = r$.

Ahora, sea k el primer ínidice tal que $v_k = w_k$, entonces tenemos un camino

$$v \quad v_1 \quad \ldots \quad v_k = w_k \quad w_{k-1} \quad w_{k-2} \quad \ldots \quad w$$

con $2k + 1$ vértices. Pero como v y w forman un lado, el grafo contiene a C_{2k+1}, por lo tanto $\chi(G) \geq 3$. La respuesta de que el grafo no es 2-coloreable es correcta.

Cororlario:

$\chi(G) \geq 3 \Leftrightarrow G$ tiene un ciclo impar

Algoritmo greedy de coloreo

Requiere un **orden** de los vértices, que influye en el resultado. Si el órden es $v_1, v_2, \ldots, v_n$, entonces

```
C(v_1) = 1
for k = 1,2,...,n:
    C(v_k) = min(j ∈ {1,2,...,n} : C(v_i) ≠ j ∀i ≤ k−1) tal que (v_i, v_k) ∈ E)
```

$\Gamma(v)$

El conjunto de vértices adyacentes de v se denota $\Gamma(v)$.

Cota inferior para $\chi(G)$

Si H es un subgrafo de G, entonces $\chi(H) \leq \chi(G)$.

K_n

K_n denota al grafo completo de n vértices. $\chi(K_n) = n$

C_n

C_n denota al grafo cíclico en n vértices.

$$\chi(C_n) = \begin{cases} 2 & n \text{ es par} \\ 3 & n \text{ es impar} \end{cases}$$

Prueba

Si n es par, coloreemos $C(x) = \begin{cases} 1 & x \text{ par} \\ 2 & x \text{ impar} \end{cases}$, de esta manera, tenemos que:

- los vecinos de $x \neq n, x \neq 1$ son $x - 1$ y $x + 1$, ambos de paridad distinta a la de x, por lo tanto reciben distinto color que x

- los vecinos de $x = n$ son $n - 1$ y 1. Como estamos considerando el caso n par , podemos asegurar que $n - 1$ es impar, al igual que 1. Por lo tanto, $x = n$ recibe distinto color que sus vecinos.
- Los vecinos de $x = 1$ son n y 2, ambos pares, por lo tanto, $x = 1$ recibe color distinto que el de sus vecinos.

Si n es impar, coloreemos $C(x) = \begin{cases} 1 & x \text{ impar}, \ x \neq n \\ 2 & x \text{ par} \\ 3 & x = n \end{cases}$, entonces

- como antes, los vecinos de $x \neq n, x \neq 1$ son $x - 1$ y $x + 1$, ambos de paridad distinta a la de x, por lo tanto reciben distinto color que x
- para $x = n$ que recibe el color 3, sus vecinos $n-1$ y 1 reciben los colores 2 y 1, respectivamente
- para $x = 1$, que recibe el color 1, sus vecinos n y 2 reciben los colores 3 y 2, respectivamente

Como este coloreo es propio, tenemos $\chi(C_n) \leq 3$. Ahora probaremos que no puede ser $\chi(C_n) < 3$. Supongamos, para llegar a una contradicción, que sí existe un coloreo de C_n de 2 colores cuando n es impar. Sea $color_1$ el color de 1 y sea $color_2$ el color de 2.

- Como $\{1, 2\} \in E$, $color_1 \neq color_2$,
- como $\{2, 3\} \in E$, $color_2 \neq C(3)$, luego debe ser $C(3) = color_1$
- como $\{3, 4\} \in E$, $color_1 \neq C(4)$, luego debe ser $C(4) = color_2$
- ...

Se ve que $C(x)$ es $color_1$ si x es impar, y $C(x)$ es $color_2$ si x es par. Pero entonces $\{n, 1\} \in E$ y $C(n) = C(1) = color_1$, y el coloreo no es propio. Absurdo.

$\Delta(G)$ y $\delta(g)$, grafos regulares

Sea $G = (V, E)$ un grafo, entonces

$$\Delta(G) = \text{Máx}_{v \in V}\{d(v)\}$$

$$\delta(G) = \text{Mín}_{v \in V}\{d(v)\}$$

Un grafo G se dice regular si $\Delta(G) = \delta(G)$ (todos los vértices tienen el mismo grado.

$\Delta(G) + 1$ es cota de $\chi(G)$

1. $\chi(G) \leq \Delta + 1$
2. El algoritmo greedy usa a lo sumo $\Delta + 1$ colores

Prueba
Como es evidente que $2 \Rightarrow 1$, basta con probar 2. Recordemos que en el algoritmo greedy,

$$C(v_k) = \min\{j \in \{1, 2, \dots, \} : C(v_i) \neq j \ \forall \ i \leq k - 1 \text{ tal que } (v_i, v_k) \in E\}$$

En el peor de los casos, ya todos los vecinos de v_k han sido coloreados y con colores distintos. Además, lo "peor"que podría pasar es que $d(v_k) = \Delta(G)$. Aún en esa situación, se deberían descartar $\Delta(G)$ colores y elegimos el color $\Delta(G) + 1$. Siempre habrá en el conjunto dado por $\{1, 2, \dots, \Delta(G), \Delta(G) + 1\}$ al menos un color que sí voy a poder utilizar.

Teorema de Brooks

Dado que $\Delta(G)+1$ es una cota para $\chi(G)$, podemos preguntarnos, ¿qué tan buena es esa cota? Para algunos grafos, vemos que el número cromático alcanza dicha cota:

$$\chi(C_{2k+1}) = 3 = 2+1 \quad \text{y} \quad \Delta(C_{2k+1}) = 2$$
$$\chi(K_n) = n = (n-1)+1 \quad \text{y} \quad \Delta(K_n) = n-1$$

El teorema de Brooks (1941) baja la cota para grafos conexos que no sean ciclos impares C_{2k+1} ni completos K_n.

Enunciado

Sea G conexo, $G \neq C_{2k+1}$ y $G \neq K_n$, entonces $\chi(G) \leq \Delta(G)$.

Esquema de la prueba

Daremos un algoritmo con input G conexo no completo ni ciclo impar que retorna un coloreo propio con a lo sumo $\Delta(G)$ colores.

Primero definimos dos funciones auxiliares:

---

procedure RECOLOREAR(v: **vértice**, $G = (V, E)$)
$c_0 := C(v)$
$C(v) = \min\{c \mid C(z) \neq c \; \forall \; z \in \Gamma(v) \wedge c \neq c_0\}$

Esta función cambia el color del vértice v tomando el mínimo color disponible (teniendo en cuenta los colores asignados a los vecinos de v).

---

procedure INTERCAMBIAR_COLORES($i, j, G = (V, E)$)
for $v \in V$ **do**
if $C(v) = i$ **then**
$C(v) := j$
if $C(v) = j$ **then**
$C(v) := i$

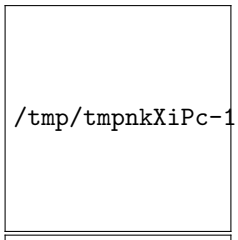
Esta función cambia la asignación de todos los vértices que tienen el color i por el color j y viceversa.

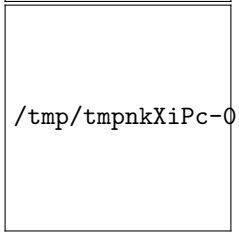
Notación

Si $W \subseteq V$ (es decir, un subconjunto de los vértices), el subgrafo de G que es generado por W, denotado $G[W]$ es

$$G[W] = (W, \{\{x, y\} \in E \mid x, y \in W\})$$

Por ejemplo, si tomamos el grafo $G = (\{a, b, c, d\}, \{\{a, b\}, \{a, c\}, \{a, d\}, \{b, c\}, \{b, d\}\})$ y definimos $W = \{a, d, b\}$, entonces $G[W] = (\{a, b, d\}, \{\{a, b\}, \{a, d\}, \{b, d\}\})$.





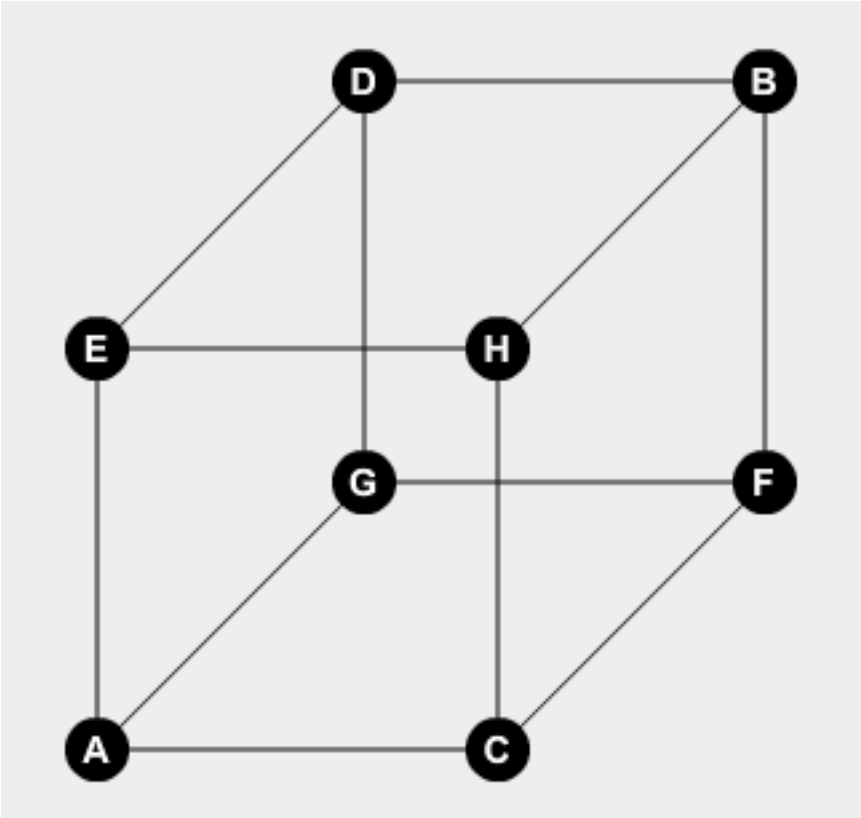
También utilizaremos la notación $H_{i,j}$ para referirnos al subgrafo de G generado por los vértices de color i, j (cadena de Kempe). La idea es utilizar INTERCAMBIAR˙COLORES(i, j, K) sólo cuando K sea una componente conexa de $H_{i,j}$.

---

procedure BROOKS($G = (V, E)$)
  $x = v \in V$ tal que $d(v) = \delta(G)$ ▷ x tiene el menor grado posible
  orden = REVERSE(orden BFS a partir de x) ▷ x es el último
  C = COLOREO_GREEDY(G, orden)
  if $|C| \leq \Delta$ **then**
    return C ▷ Fin del algoritmo!
    ▷ ———————————————————
    ▷ OBSERVACIÓN: Si $|C| = \Delta + 1$, entonces G es regular
    ▷ ———————————————————
    ▷ Ocurrió el peor caso de COLOREO_GREEDY,
    ▷ entonces sabemos que x tiene exactamente Δ vecinos
  for $i \in \{1, 2, \ldots, \Delta\}$ **do**
    $x_i = i$-ésimo vecino de x
    $i = C(x_i)$
    for j tal que x_j es vecino de x_i **do**
      ▷ Cadena de Kempe formada por los colores i, j, partiendo de x_i
      $K_{i,j}$ = la componente conexa de $H_{i,j}$ que contiene a x_i
      if x_i tiene al menos dos vecinos en $K_{i,j}$ **then**
        RECOLOREAR(x_i, G)
        C(x) = i
        return C
        ▷ OBS: Si llegó acá, se debe a que
        ▷ x_i tiene un solo vecino en $K_{i,j}$
      if $\exists\, i \neq j$ con $K_{i,j} \neq K_{j,i}$ **then**
        INTERCAMBIAR_COLORES($i, j, K_{i,j}$)
        C(x) = i
        return C
    ▷ Falló todo lo anterior
  Sean i, j tales que $K_{i,j} \neq \{x_i, x_j\}$
  if $\exists\, u \in K_{i,j}$ con al menos tres vecinos **then**
    u = primer vecino de x_i
    RECOLOREAR(u)
    Calcular $K_{i,j}$ otra vez
    INTERCAMBIAR_COLORES($i, j, K_{i,j}$)
    C(x) = i
    return C
  Sea $k \neq i, j$ ▷ como $G \neq$ ciclo impar, $k \leq \Delta$
  Sea u = único vecino de x_i en $K_{i,j}$
  if ningún vecino de u tiene el color k **then**
    C(u) = k
    C(x_i) = j
    C(x_j) = i
    return C
  if componente conexa de $H_{j,k}$ que contiene a u interseca a $K_{i,j}$ **then**
    sea $w \in$ tal intersección
    RECOLOREAR(w)
    Calcular $K_{i,j}$ otra vez
    INTERCAMBIAR_COLORES($i, j, K_{i,j}$)
    C(x) = i
    return C
  K = componente conexa de $H_{j,k}$ que tiene a u
  INTERCAMBIAR_COLORES(j, k, K)
  Calcular $K_{i,j}$ otra vez
  INTERCAMBIAR_COLORES($i, j, K_{i,j}$)
  C(x) = i
  return C

Greedy no siempre obtiene $\chi(G)$

Si consideramos el grafo G



y corremos el algoritmo greedy en órden alfabético, obtenemos

- Color de A: 1
- Color de B: 1
- Color de C: 2
- Color de D: 2
- Color de E: 3
- Color de F: 3
- Color de G: 4
- Color de H: 4

Si en cabmio corremos el algoritmo de coloreo greedy con el orden $\{A, C, H, E, D, G, F, B\}$ (es decir, los cuatro vértices de la çara frontal"seguidos de los cuatro vértices de la çara trasera", obtenemos:

- Color de A: 1
- Color de C: 2
- Color de H: 1
- Color de E: 2

- Color de D: 1
- Color de G: 2
- Color de F: 1
- Color de B: 2

No podemos probar los $n!$ órdenes posibles de los grafos para correr el algoritmo voraz, pero existen ciertas heurísticas para ordenar los vértices de modo de (quizás) obtener un coloreo propio con menos colores que uno que ya se tiene.

Greedy con un orden seleccionado

Sea G un grafo y C un coloreo propio cualquiera de G con k colores $\{1, 2, \ldots, k\}$.

- Reordenemos los colores con un orden **arbitrario** $(j_1, j_2, \ldots, j_k)$.
- Ahora reordenemos los vértices poniendo primero todos los vértices del color j_1, luego todos los vértices del color j_2, así sucesivamente, hasta poner por último todos los vértices del color j_k.
- Entonces el algoritmo de coloreo greedy con ese orden colorea G con a lo sumo k colores.

Prueba

Supongamos que el teorema es verdadero para k colores y probémoslo para $k + 1$.

Sea W el conjunto formado por los vértices de color $j_1, j_2, \ldots, j_k$, y sea U el conjunto formado por los vértices de color j_{k+1}.

Sea $H = G[W]$ (el subgrafo de G generado por W). Entonces C/H ("C restringido H") colorea H con k colores. Por hipótesis inductiva, greedy no usará más de k colores para colorear $G[W]$. Digamos que usa l colores, con $l \leq k$.

Ahora bien, al correr el algoritmo greedy sobre el grafo G (ahora sí el grafo entero, no sólo $G[W]$) va a hacer lo mismo hasta terminar de colorear todos los vértices de H (es decir, W): va a colorear todos los vértices que están en W usando sólo l colores. Nos preguntamos entonces, ¿cuántos colores extra necesitará greedy para los vértices de U?

Si $x \in U$:

1. si $\exists\, j \leq l : x$ "no es vecino de ningún vértices de color j", greedy no requiere ningún color extra
2. si no, greedy le dará el color extra $l + 1$

Es decir, el algoritmo greedy no e puede ver forzado a dar un color $l + 2$, porque para que pase eso, necesariamente x tiene que ser vecino de otro vértice de color $l + 1$. Los únicos vértices que tienen color $l + 1$ son los de U, que no forman vértices por definición (todos tienen el color j_{k+1}.

Por lo tanto, greedy va a utilizar a los sumo $l + 1$ colores.

Corolario

Existe un orden $\leq$ de los vértices tal que greedy $(G, \leq) = \chi(G)$.

Prueba

Sea C un coloreo propio con $\chi(G)$ colores. Entonces greedy en el orden de los vértices dado por este coloreo produce un coloreo C' con a lo sumo $\chi(G)$ colores. Como $\chi(G)$ es el mínimo, greedy on ese orden utiliza $\chi(G)$ colores.

Baby Brooks

Sea G un grafo conexo no regular, entonces $\chi(G) \leq \Delta(G)$

Notemos que este teorema es más débil que el Teorema general de Brooks, ya que este no afirma nada en relación a los ciclos pares, mientras que aquel sí.

Prueba

1. Elegir x con $d(x) = \delta(G)$ (un vértice de menor grado posible).
2. Ordenar los vértices con el orden inverso al dado por BFS(x), es decir, $x_1, x_2, \ldots, x_{n-1}, x_n = x$
3. Correr el algoritmo greedy en ese orden

Afirmación

Si $i \leq n-1$ greedy colorea x_i con algún color en $\{1, 2, \ldots, \Delta(G)\}$

Subprueba

El vértice x_i fue "puesto.[en] el orden BFS por algún vecino. El único que no fue puesto por ningún vecino fue el propio $x_n = x$, que por la definición de i queda excluido de esta afirmación. Por lo tanto este vecino (el que incluyó a x_i) estaba antes en el orden BFS. Sin embargo, como estamos usando el orden BFS invertido, ahora está después.

Pero por la forma en que funciona BFS, ese .[a]lguien.[es] sí o sí vecino de x_i. En consecuencia, podemos asegurar que para todo x_i hay un x_j (con $j > i$) que es vecino de x_i en el grafo.

Ahora bien:

1. $\forall\, i \leq n-1, \exists\, j > i : x_j \in \Gamma(x_i)$
2. $\forall\, i \leq n-1$, cuando vamos a colorear x_i, existe algún vecino de x_i sin colorear (porque x_j está después y todavía no llegué a colorearlo).

Por lo tanto, el número de vecinos coloreados de x_i es a lo sumo $d(x_i) - 1$ y $d(x_i) \leq \Delta(G)$ (aclaración: en los apuntes tengo $d(x_i) \leq \Delta(G) - 1$, pero me parece incorrecto), por lo tanto greedy elimina a los sumo $\Delta(G) - 1$ colores, y seguro que le da a x_i algún color en $\{1, 2, \ldots, \Delta(G)\}$.

(Fin supbrueba)

Hemos probado la afirmación sin usar que G no es regular. Queda aún por colorear x_n. Para $x_n = x$ todos sus vecinos están ya coloreados. Greedy puede eliminar a lo sumo $d(x)$ colores. Pero $d(x) = \delta(G)$ y como G no es regular, se tiene $\delta(G) < \Delta(G)$. Luego, greedy elimina $\delta(G) < \Delta(G)$ colores y colorea x con algún color en $\{1, 2, \ldots, \Delta(G)\}$.

Grafo dirigido

Un grafo dirigido es un par (V, E) con $E \subseteq V \times V$ (son pares ordenados).

Network

Un Network es una 5-tupla $N = (V, E, C, s, t)$ tal que

1. (V, E) es un grafo dirigido
2. C es una función $E \to \mathbb{R}_{\geq 0} \cup \infty$ (función çapacidad")
3. $s, t \in V$ tales que no exista $v \in V$ con $(v, s) \in E$ o $(t, v) \in E$.

s (source) sólo es vértice de partida de cualquier lado ("no consume"), mientras que t (sink) es sólo vértice de llegada ("no produce").

Notación

- $\overrightarrow{xy}$ denota al lado (x, y)
- $\Gamma^+(x) = \{y \in V : \overrightarrow{xy} \in E\}$ (el conjunto de vértices a los cuales se puede ir desde x)
- $\Gamma^-(x) = \{y \in V : \overrightarrow{yx} \in E\}$ (el conjunto de vértices desde los cuales se puede llegar a x)

Flujo

Un flujo en un network N es una función $f : E \to R_{\geq 0}$ tal que

1. $0 \leq f(\overrightarrow{xy}) \leq C(\overrightarrow{xy})$ ("Feasability")
2. Para todo $x \neq s, t$, $\sum_{y \in \Gamma^+(x)} f(\overrightarrow{xy}) = \sum_{y \in \Gamma^-(x)} f(\overrightarrow{xy})$ (todo lo que entra debe salir)

Valor de un flujo

El valor de un flujo f en un network N, denotado $v(f)$ es

$$v(f) = \sum_{z \in \Gamma(s)} f(\overrightarrow{sz})$$

Flujo entero

Un flujo f en un network N se dice **entero** si

$$f(\overrightarrow{xy}) \in \mathbb{Z} \; \forall \; \overrightarrow{xy} \in E$$

Flujo entero maximal

Un flujo entero maximal es un flujo entero tal que $v(f)$ es máximo entre todos los flujos enteros.

Operadores cuantificados sobre subconjuntos de V

Dado un grafo G, sean $A, B \subseteq V$ y sea $\phi : E \to \mathbb{R}$ una función que asocia un número real a un lado del grafo G. Entonces adoptamos la notación $\phi(A, B)$ para denotar la sumatoria de los valores que toma ϕ para todos aquellos lados del grafo que comienzan en A y terminan en B:

$$\Phi(A, B) = \sum_{\substack{x \in A \\ y \in B \\ \overrightarrow{xy} \in E}} \phi(\overrightarrow{xy})$$

Además, si $x \in V$, como una extensión a esta notación, podemos escribir:

$$\phi(x, B) = \phi(\{x\}, B)$$
$$\phi(A, x) = \phi(A, \{x\})$$

Como ejemplos:

- $f(A, B)$ es la sumatoria del flujo f en lados que empiezan en A y terminan en B.
- $CAP(A, B)$ es la capacidad de los lados que empiezan en A y terminan en B
- $OUT_f(x) = f(x, V)$
- $IN_f(x) = f(V, x)$

Relacion entre $v(f)$, el flujo saliente de s y el flujo entrante a t

$$v(f) = OUT_f(s) = IN_f(t)$$

Prueba

Por definición, $v(f) = \sum_{z \in \Gamma(s)} f(\overrightarrow{xy})$, es decir, $v(f) = OUT_f(s)$. Veamos entonces la segunda igualdad.

Si tomamos la fórmula:

$$\begin{aligned} f(V,V) &= \sum_{\substack{x,y \in V \\ \overrightarrow{xy} \in E}} f(\overrightarrow{xy}) \\ &= \sum_{x \in V} \left(\sum_{\substack{y \in V \\ \overrightarrow{xy} \in E}} f(\overrightarrow{xy}) \right) \\ &= \sum_{x \in V} f(x,V) \\ &= \sum_{x \in V} OUT_f(x) \end{aligned}$$

Pero también podemos escribir

$$\begin{aligned} f(V,V) &= \sum_{\substack{x,y \in V \\ \overrightarrow{yx} \in E}} f(\overrightarrow{yx}) \\ &= \sum_{x \in V} \left(\sum_{\substack{y \in V \\ \overrightarrow{yx} \in E}} f(\overrightarrow{yx}) \right) \\ &= \sum_{x \in V} f(V,x) \\ &= \sum_{x \in V} IN_f(x) \end{aligned}$$

Entonces,

$$\sum_{x \in V} OUT_f(x) = \sum_{x \in V} IN_f(x)$$

Si reescribimos esta fórmula haciendo un poco más explícitos sus términos:

$$\begin{aligned} \sum_{x \in V} OUT_f(x) &= OUT_f(x_1) + OUT_f(x_2) + \cdots + OUT_f(x_n) \\ &= IN_f(x_1) + IN_f(x_2) + \cdots + IN_f(x_n) \\ &= \sum_{x \in V} IN_f(x) \end{aligned}$$

Pero si $OUT_f(x) = IN_f(x)$ para todo $x \neq s, t$, sólo quedan los términos:

$$OUT_f(s) + OUT_f(t) = IN_f(s) + IN_f(t)$$

Sin embargo, no existen lados $\overrightarrow{xs}$ ni $\overrightarrow{tx}$, por lo tanto

$$IN_f(s) = OUT_f(t) = 0$$

En consecuencia:

$$OUT_f(s) = IN_f(s)$$

Corte

Dado un network $N = (V, E, C, s, t)$, un subconjunto $S \subset V$ se dice corte si

1. $s \in S$
2. $t \notin S$

Ejemplos de cortes:

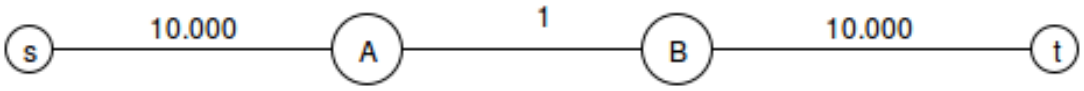
- $S_1 = \{s\}$
- $S_2 = V - \{t\}$

Capacidad de un corte

Dado un network N, la capacidad de un corte S es

$$\begin{aligned} CAP(S) &= c(S, V - S) \\ &= \sum_{\substack{x \in S \\ y \notin S \\ \overrightarrow{xy} \in E}} c(\overrightarrow{xy}) \end{aligned}$$

Ejemplo:



- $CAP(\{s\}) = c(\overrightarrow{sA}) = 10{,}000$
- $CAP(\{s, A\}) = c(\overrightarrow{AB}) = 1$

Camino aumentante

Dado un network $N = (V, E, C, s, t)$, un çamino aumentante"(augmenting path) es una sucesión de vértices $x_0 = s, x_1, x_2, \ldots, x_r = t$ tales que para todo $0 \leq i < r$ se cumple que una de estas dos cosas:

- $\overrightarrow{x_i x_{i+1}} \in E$ (es lado "forward") y $f(\overrightarrow{x_i x_{i+1}}) < c(\overrightarrow{x_i x_{i+1}})$ (se puede aún mandar más flujo), o bien
- $\overrightarrow{x_{i+1} x_i} \in E$ (es lado "backwards") y $f(\overrightarrow{x_{i+1} x_i}) > 0$ (se puede aún "devolver"flujo al anterior.

Generalización de $v(f)$

Sea S un corte y f un flujo, entonces

$$v(f) = f(S, V - S) - f(V - S, S)$$

Esta idea generaliza la noción de $v(f)$ para conjuntos.
Prueba
Sea $x \in S$. En particular $x \neq t$ (ya que S es corte). Entonces

$$f(x, V) - f(V, x) = \begin{cases} 0 & x \neq s \\ v(f) & x = s \end{cases}$$

Entonces

$$\begin{aligned} v(f) &= v(f) + 0 + 0 + \dots \\ &= \sum_{x \in S} (f(x, V) - f(V, x)) \\ &= f(S, V) - f(V, S) \end{aligned}$$

Si ahora partimos V en dos subconjuntos disjuntos: $V - S$ y S, podemos continuar

$$\begin{aligned} f(S, V) - f(V, S) &= f(S, S) + f(S, V - S) - f(S, S) - f(V - S, S) \\ &= f(S, V - S) - f(V - S, S) \end{aligned}$$

Flujo a partir de un camino aumentante

Dado un network $N = (V, E, C, s, t)$ y un flujo f sobre N. Sea además $x_0 = s, x_1, x_2, \dots, x_r = t$ un camino aumentante entre s y t. Sea

$$\epsilon_i = \begin{cases} c(\overrightarrow{x_i x_{i+1}}) - f(\overrightarrow{x_i x_{i+1}}) & \text{si } \overrightarrow{x_i x_{i+1}} \text{ es lado forward} \\ f(\overrightarrow{x_{i+1} x_i}) & \text{si } \overrightarrow{x_i x_{i+1}} \text{ es lado backwards} \end{cases}$$

Notemos que ϵ_i es "lo que podría aumentar"si estoy considerando un lado forward, o lo que me podrían devolver, si estoy considerando un lado backwards.
Sea $\varepsilon = \text{mín}(\epsilon_i)$ (que debe ser positivo por tener un camino aumentante).
Sea f^* una función sobre los lados del network

$$f^*(\overrightarrow{x_i x_{i+1}}) = \begin{cases} f(\overrightarrow{x_i x_{i+1}}) + \varepsilon & \text{si } \overrightarrow{x_i x_{i+1}} \text{ es un lado forward} \\ f(\overrightarrow{x_i x_{i+1}}) - \varepsilon & \text{si } \overrightarrow{x_i x_{i+1}} \text{ es un lado backward} \end{cases}$$

y además

$$f^*(\overrightarrow{xy}) = f(\overrightarrow{xy})$$

para todos los otros lados.
Entonces f^* es flujo y $v(f^*) = v(f) + \varepsilon$
Prueba
La prueba tiene la siguiente estructura

1. Probar que $0 \leq f^*(v)$
2. Probar que $f^* \leq c$
3. Conservación: $f^*(x, V) = f^*(V, x) \quad \forall\, x \neq s, t$
4. Probar que $v(f^*) = v(f) + \varepsilon$

1) $0 \leq f^*(v)$

En los lados "forward"del camino aumentante (aquellos que pertenecen a E en los que aún puedo aumentar la capacidad), f^* está definido como

$$f^*(\overrightarrow{x_i x_{i+1}}) = f(\overrightarrow{x_i x_{i+1}}) + \varepsilon \geq \varepsilon \geq 0$$

Mientras que en los lados "backwards"(aquellos que invertidos pertenecen a E, pero aún puedo devolver flujo) está definido como

$$f^*(\overrightarrow{x_{i+1} x_i}) = f(\overrightarrow{x_{i+1} x_i}) - \varepsilon \geq f(\overrightarrow{x_{i+1} x_i}) - \epsilon_i = 0$$

2) $f^* \leq c$

En los lados "backwards", esto es obvio, porque f es un flujo y estoy restando del mismo, por lo que sigue siendo $f^* = f - \varepsilon < f \leq c$

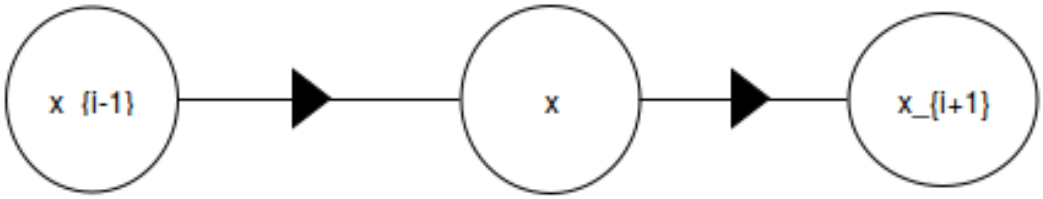
En los lados "forward"

$$\begin{aligned} f^*(\overrightarrow{x_i x_{i+1}}) &= f(\overrightarrow{x_i x_{i+1}}) + \varepsilon \\ &\leq f(\overrightarrow{x_i x_{i+1}}) + \epsilon_i \\ &= f(\overrightarrow{x_i x_{i+1}}) + c(\overrightarrow{x_i x_{i+1}}) - f(\overrightarrow{x_i x_{i+1}}) \\ &= c(\overrightarrow{x_i x_{i+1}}) \end{aligned}$$

3) $f^*(x, V) = f^*(V, x) \quad \forall\, x \neq s, t$

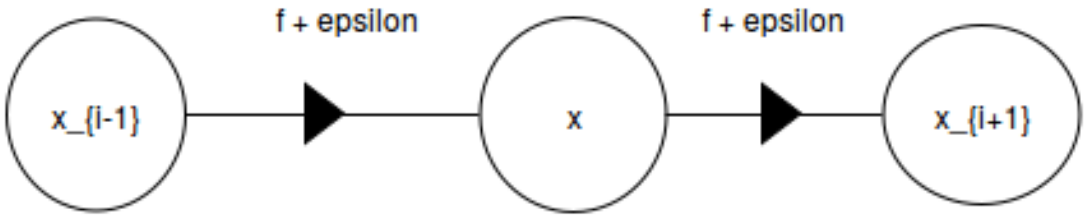
Si x no es uno de los x_i (esto es, no forma parte del camino aumentante) el valor del flujo entrante y saliente viene dado por f, por lo que la conservación está asegurada.

Si, en cambio, x es parte del camino aumentante, es decir $x = x_i$ con $1 \leq i \leq r - 1$ (pues $x \neq s, t$), entones existen x_{i-1} y x_{i+1} (es decir, hay un lado entrante y un lado saliente de x_i).



Como no sabemos si se trata de lados forward o lados backwards, debemos analizar cuatro casos.

CASO 1: tanto $\overrightarrow{x_{i-1} x_i}$ como $\overrightarrow{x_i x_{i+1}}$ son lados forward:



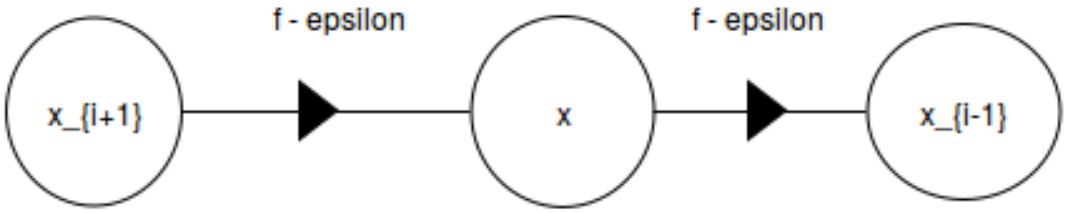
Entonces

$$f^*(x_i, V) = f(x_i, V) + \varepsilon \text{ (a causa del lado } \overrightarrow{x_i x_{i+1}})$$

$$f^*(V, x_i) = f(V, x_i) + \varepsilon \text{ (a causa del lado } \overrightarrow{x_{i-1} x_i})$$

Como f es flujo, se tiene que $f(x_i, V) = f(V, x_i)$, por lo tanto f^* mantiene la conservación en este caso.

CASO 2: tanto $\overleftarrow{x_{i-1} x_i}$ como $\overleftarrow{x_i x_{i+1}}$ son lados backwards, esto quiere decir que verán decrementado el flujo en ε.

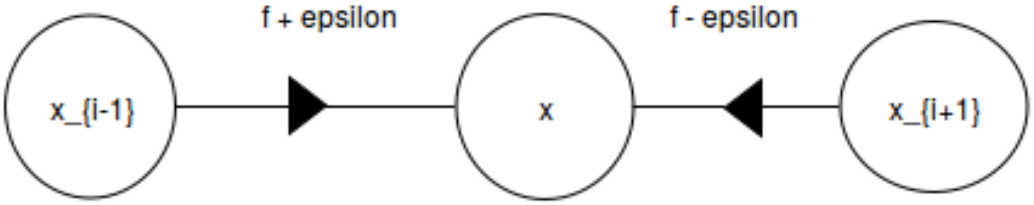


$$f^*(x_i, V) = f(x_i, V) - \varepsilon \text{ (a causa del lado } \overrightarrow{x_i x_{i-1}})$$

$$f^*(V, x_i) = f(V, x_i) - \varepsilon \text{ (a causa del lado } \overrightarrow{x_{i+1} x_i})$$

Por un razonamiento análogo al caso anterior, se ve que f^* también cumple la conservación en este caso.

CASO 3: $\overrightarrow{x_{i-1} x_i}$ es un lado forward (es un lado del network que verá incrementado su flujo en ε), pero $\overleftarrow{x_i x_{i+1}}$ es un lado backwards que verá decrementado su flujo en ε. Es decir, en el grafo se producirá:



Notemos que en el camino aumentante, x_{i+1} es posterior a x_i pero es un lado backwards, o sea que en el grafo, existe el lado $\overrightarrow{x_{i+1} x_i}$ (tanto x_{i-1} como x_{i+1} están en $\Gamma^-(x_i)$). En este caso

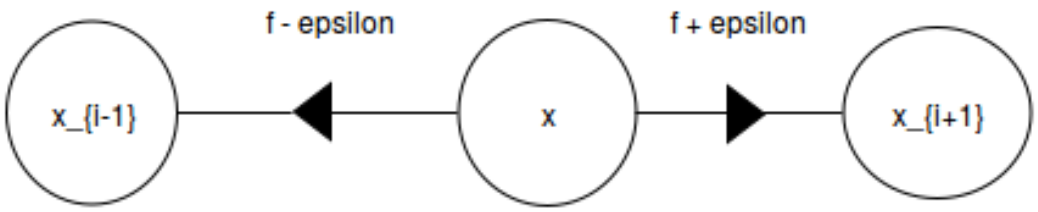
$$f^*(x_i, V) = f(x_i, V)$$

pues no hay lados que salgan de x_i que hayan cambiado su flujo. En cuanto al flujo entrante:

$$f^*(V, x_i) = f(V, x_i) + \varepsilon - \varepsilon = f(V, x_i)$$

Luego, se mantiene la conservación en este caso.

CASO 4: $\overleftarrow{x_{i-1} x_i}$ es un lado backwards (es decir, $\overrightarrow{x_i x_{i-1}}$ un lado del network que verá decrementado su flujo en ε), pero $\overrightarrow{x_i x_{i+1}}$ es un lado forward que verá incrementado su flujo en ε. Es decir, en el grafo se producirá:



Notemos que en el camino aumentante, x_{i-1} es anterior a x_i pero es un lado backwards, o sea que en el grafo, existe el lado $\overrightarrow{x_i x_{i-1}}$ (tanto x_{i-1} como x_{i+1} están en $\Gamma^+(x_i)$). En este caso

$$f^*(V, x_i) = f(V, x_i)$$

pues no hay lados que entren a x_i que hayan cambiado su flujo. En cuanto al flujo saliente:

$$f^*(x_i, V) = f(x_i, V) + \varepsilon - \varepsilon = f(x_i, V)$$

Luego, se mantiene la conservación en este caso.
Hemos probado que f^* es flujo. Falta ver la segunda afirmación
4) $v(f^*) = v(f) + \varepsilon$
Para todos los lados vértices vecinos de s que no formaban parte del camino aumentante, el valor de f^* es igual al valor de f. Sin embargo, hay un único vértice x_1 que formaba parte del camino aumentante, por lo tanto:

$$v(f^*) = f^*(s, V) = f(s, V) + \varepsilon$$

Max flow, min cut

Sea un network $N = (V, E, c, s, t)$.

1. Si f es flujo y S es corte, entonces $v(f) \leq CAP(S)$
2. Si $v(f) = CAP(S)$, entonces f es maximal y S, minimal
3. Si f es maximal, entonces existe un corte S con $v(f) = CAP(S)$

Prueba
1)

$$\begin{aligned} v(f) &= f(S, V-S) - f(V-S, S) \\ &\leq f(S, V-S) \\ &\leq c(S, V-S) \\ &= CAP(S) \end{aligned}$$

2)
Spongamos $v(f) = CAP(S)$. Sea g cualquier flujo, por 1) se tiene que

$$\begin{aligned} v(g) &\leq CAP(S) \\ &= v(f) \end{aligned}$$

es decir, f es maximal. Por otro lado, sea T un corte. Entonces por 1) se tiene que

$$\begin{aligned} CAP(T) &\geq v(f) \\ &= CAP(S) \end{aligned}$$

es decir, S es minimal.

3)

Sea $S = \{s\} \cup \{x : \text{ existe un camino aumentante (relativo a } f) \text{ entre } s \text{ y } x\}$ un candidato a corte. Nos hacemos la pregunta:

¿Está t en S? Si suponemos que sí, entonces existe un camino aumentante entre s y t, por la propiedad demostrada anteriormente (ver aquí) debemos aceptar que existe un flujo f^* y $\varepsilon > 0$ con $v(f^*) = v(f) + \varepsilon$, lo cual es absurdo. Por lo tanto, t no está en S. Esto implica que S es corte. Calculemos su capacidad.

Tenemos que

$$v(f) = f(S, V - S) - f(V - S, S)$$

Analicemos cada término:

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

Como $x \in S$ entonces existe un camino aumentante entre s y x y como $z \notin S$, entonces no existe un camino aumentante entre s y z. Entonces si $\overrightarrow{xz} \in E$, $f(\overrightarrow{xz}) = c(\overrightarrow{xz})$ (el flujo enviado iguala a la capacidad, ya no se puede enviar más). Por lo tanto,

$$\begin{aligned} f(S, V - S) &= \sum_{\substack{x \in S \\ z \in V - S \\ \overrightarrow{xz} \in E}} f(\overrightarrow{xz}) \\ &= \sum_{\substack{x \in S \\ z \in V - S \\ \overrightarrow{xz} \in E}} c(\overrightarrow{xz}) \\ &= c(S, V - S) \\ &= CAP(S) \end{aligned}$$

Por otro lado,

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

Si $x \in V - S$ entonces no existe camino aumentante entre s y x. Si $z \in S$ entonces existe un camino aumentante $s = z_0, z_1, \ldots, z_r = z$. En particular, $s = z_0, z_1, z_2, \ldots, z_r = z, x$ no es un camino aumentante, entonces $f(\overrightarrow{xz}) = 0$. Luego,

$$\begin{aligned} f(V, S - V) &= \sum_{\substack{x \in V - S \\ z \in S \\ \overrightarrow{xz} \in E}} f(\overrightarrow{xz}) \\ &= \sum_{\substack{x \in V - S \\ z \in S \\ \overrightarrow{xz} \in E}} 0 \\ &= 0 \end{aligned}$$

Entonces, retomando el incio del inciso 3)

$$v(f) = f(S, V - S) - f(V - S, S) = CAP(S) - 0 = CAP(S)$$

Ford-Fulkerson

```
procedure FORD_FULKERSON
    f = 0
    flag = True
    while flag:
        if ∃ camino aumentante entre s y t:
            construir f* a partir de f como en el lema
        else:
            flag = False
```

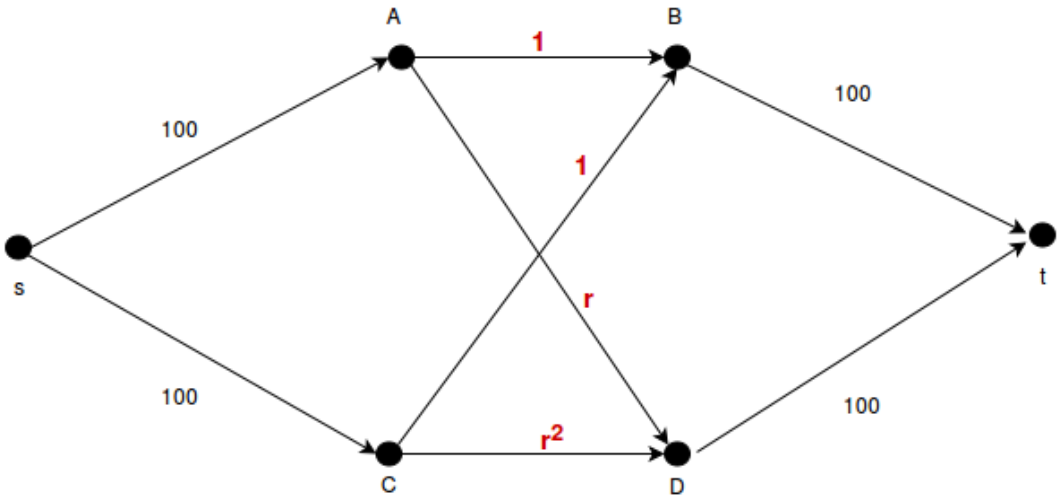
Como consecuencia del teorema "Max flow, min cut", podemos afirmar que si el algoritmo de Ford-Fulkerson termina es porque ya no hay caminos aumentantes entre s y t, por lo tanto existe un corte S con $v(f) = CAP(S)$ y por lo tanto, el flujo es maximal.

El algoritmo, sin embargo, tiene dos problemas:

1. No siempre termina
2. Su complejidad puede ser muy mala

Ejemplo de que Ford-Fulkerson puede no terminar

Consideremos el siguiente network



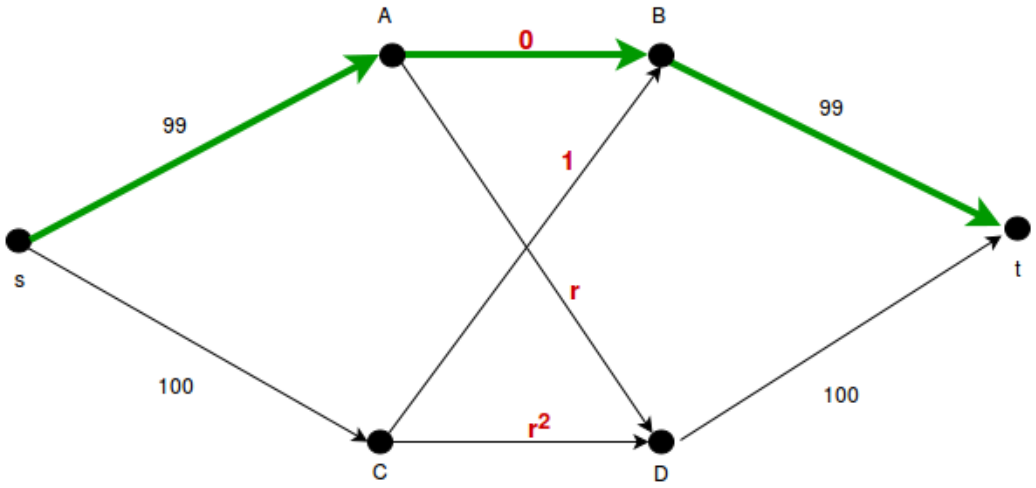
donde r es la única raíz positiva de $p(x) = x^3 + x - 1$.

Observación: $p(x) = x^3 + x - 1$ tiene una sola raíz positiva porque $p(0) = -1 < 0$, $p(1) = 1 > 0$ y la primera derivada de la función, $p'(x) = 3x^2 + 1$ es siempre positiva (es decir, la función es creciente, y por el toerema del valor medio toma todos los valores entre -1 y 1 en el intervalo $[0, 1]$).

Observación: Como $p(0) < 0 < p(1)$, se tiene que $r \in (0, 1)$ y por lo tanto, $1 > r > r^2 > r^3 > \dots$

El algoritmo de Ford-Fulkerson podría tomar los siguientes caminos

Camino 0: $sABt$ con $\varepsilon = 1$, $v(f_0) = 1$

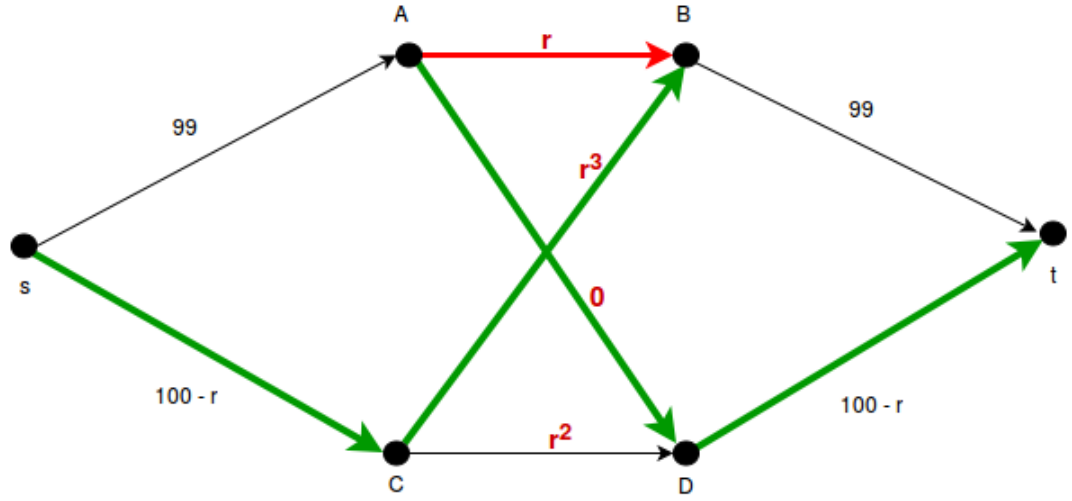


- $(c_0 - f_0)(\overrightarrow{AB}) = 0$
- $(c_0 - f_0)(\overrightarrow{CB}) = 1 = r^0$
- $(c_0 - f_0)(\overrightarrow{AD}) = r = r^1$
- $(c_0 - f_0)(\overrightarrow{CD}) = r^2$

Llamemos ψ_0 al conjunto formado por estos cuatro valores:

$$\psi_0 = \{(c_0 - f_0)(\overrightarrow{AB}), (c_0 - f_0)(\overrightarrow{CB}), (c_0 - f_0)(\overrightarrow{AD}), (c_0 - f_0)(\overrightarrow{CD})\} = \{0, 1, r, r^2\}$$

Camino 1: $sC\overleftarrow{BA}Dt$ con $\varepsilon = r$, y $v(f_1) = 1 + r$



- $(c_1 - f_1)(\overrightarrow{AB}) = 1 - (1 - r) = r$
- $(c_1 - f_1)(\overrightarrow{CB}) = 1 - r = r^3$
- $(c_1 - f_1)(\overrightarrow{AD}) = r - r = 0$
- $(c_1 - f_1)(\overrightarrow{CD}) = r^2 - 0 = r^2$

El conjunto formado por estos cuatro valores es

$$\psi_1 = \{(c_1 - f_1)(\overrightarrow{AB}), (c_1 - f_1)(\overrightarrow{CB}), (c_1 - f_1)(\overrightarrow{AD}), (c_1 - f_1)(\overrightarrow{CD})\} = \{r, r^3, 0, r^2\}$$

Parece haber un patrón, que intentaremos probar por inducción.

Hipótesis Inductiva:

Camino j:
Sin tener en cuenta el orden de los elementos del conjunto, para $j \in \mathbb{N}$ se cumple que $\psi_j = \{0, r^j, r^{j+1}, r^{j+2}\}$
y
si $\overrightarrow{h_j e_j}$ es el lado que cumple $(c_j - f_j)(\overrightarrow{h_j e_j}) = 0$
si $\overrightarrow{g_j d_j}$ es el lado que cumple $(c_j - f_j)(\overrightarrow{g_j d_j}) = r^{j+2}$
entonces $\{h_j, e_j\} \cap \{g_j, d_j\} = \emptyset$

Prueba:

Notemos que los vértices nomenclados h_j y g_j son .ªnteriores.en el grafo a los nomenclados e_j y d_j (en el gráfico, h_j y g_j pueden ser A ó C, mientras que e_j y d_j corresponden a B ó D).
Camino $j+1$: $sg_j\overleftarrow{e_j h_j} d_j t$ con $\varepsilon = r^{j+1}$ y $v(f_{j+1}) = v(f_j) + r^{j+1}$
Sabemos que

- por los lados $\overrightarrow{g_j e_j}$ y $\overrightarrow{h_j d_j}$ el flujo se incrementa en r^{j+1}, por lo tanto en el ciclo anterior, tenían una capacidad residual de al menos r^{j+1}. De ellos, uno era el lado cuya capacidad residual era r^j (y ahora es $r^j - r^{j+1} = r^j(1-r) = r^j r^3 = r^{j+3}$), mientras que el otro era el lado cuya capacidad residual era r^{j+1} (y ahora es $r^{j+1} - r^{j+1} = 0$).
- por el lado backwards $\overleftarrow{e_j h_j}$ el flujo es, en cambio, decrementado en r^{j+1}, por lo tanto, en el ciclo anterior mandaba al menos r^{j+1} y su capacidad residual era menor que $1 - r^j + 1$
- si tomamos $(c_{j+1} - f_{j+1})(\overrightarrow{h_j e_j}) = 0 + r^{j+1} = r^{j+1}$ y $(c_{j+1} - f_{j+1})(\overrightarrow{g_j d_j}) = r^{j+2}$ (este último, el lado que no usamos en este camino), tenemos que

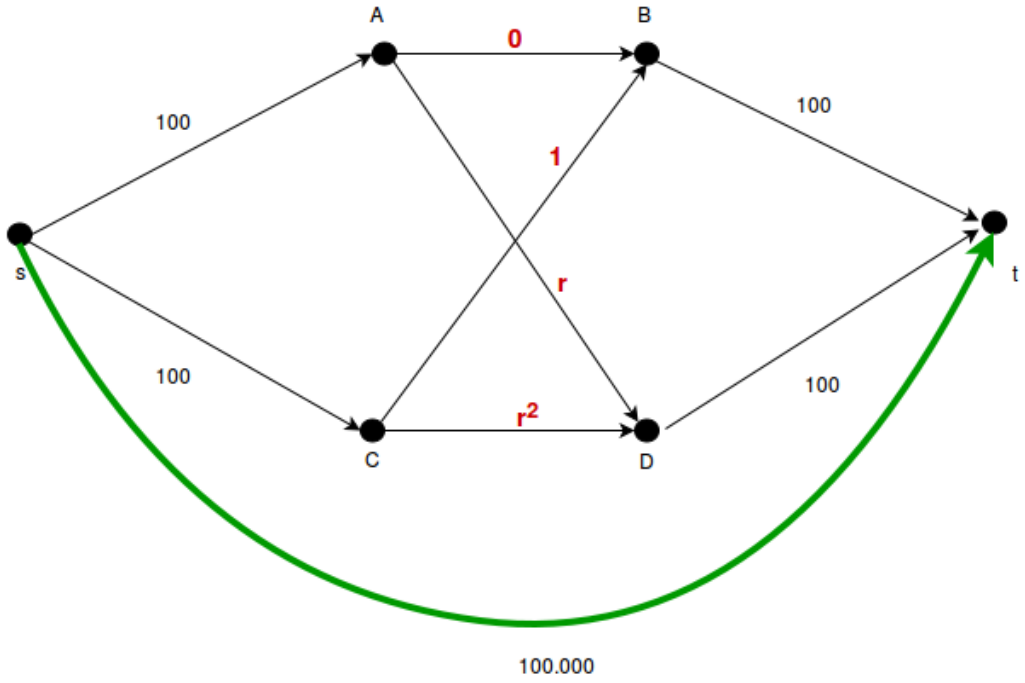
$$\psi_{j+1} = \{0, r^{j+1}, r^{j+2}, r^{j+3}\}$$

y además se cumple que los lados cuya capacidad residual es o bien 0 o bien $r^{(j+1)+2} = r^{j+3}$ están dispuestos de modo que no comparten ningún vértice.
Tenemos que

- el lado con capacidad residual 0 es $\overrightarrow{g_j e_j}$ y el lado con capacidad residual r^{j+3} es $\overrightarrow{h_j d_j}$, o bien
- el lado con capacidad resiudal 0 es $\overrightarrow{h_j d_j}$ y el lado con capacidad residual r^{j+3} es $\overrightarrow{g_j e_j}$.

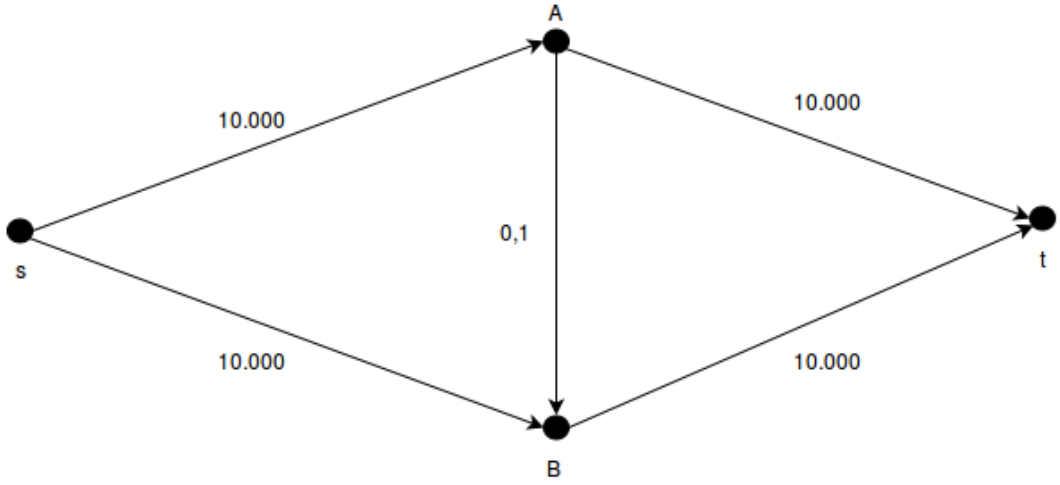
Esto prueba que el algoritmo de Ford-Fulkerson puede no terminar nunca, ya que eligiendo caminos de este modo, siempre se podrá elegir un camino aumentante más.

Podría pensarse que el valor del flujo tiende a alguna cota real F cuando j tiende a infinito y que F es el valor del flujo maximal, sin embargo esto es incorrecto, ya que basta con agregar el lado $\overrightarrow{st}$ con capacidad $F+1$ que el algoritmo de Ford-Fulkerson podría nunca elegirlo como camino aumentante y, por lo tanto, aproximarse infinitamente al valor F cuando en realidad existe un flujo que es mayor y nunca será "visto"por el algoritmo. Ese caso se podría presentar en el siguiente network:



Ejemplo de que la complejidad de Ford-Fulkerson puede ser muy mala

Consideremos el siguiente network



Una corrida de FF podría encontrar los siguientes caminos:

- $sABt$, incrementando f en $0,1$
- $s\overleftarrow{BA}t$, incrementando f en $0,1$

Repetir los dos pasos anteriores 200,000 veces.

Teorema de la integralidad

Sea $N = (V, E, C, s, t)$ un network. Si las capacidades de los lados son todas enteras, entonces Ford-Fulkerson termina y produce un flujo entero. En particular, si las capacidades son todas

enteras, entonces existe un flujo entero maximal.

Prueba

Por inducción en los flujos producidos por las sucesivas iteraciones de Ford-Fulkerson. Sean $f_0, f_1, f_2, \ldots, f_n$ los sucesivos flujos producidos por Ford-Fulkeroson.

Caso base

$f_0 = 0$ es entero.

Hipótesis inductiva

f_j es un flujo entero

Caso inductivo: Si f_j es entero, entonces f_{j+1} es entero

f_{j+1} se construye a partir de f_j, cambiando en algunos lados el valor f_j por $f_j \pm \varepsilon$. Por lo tanto, si $\varepsilon \in \mathbb{Z}$, también se tendrá $f_{j+1} \in \mathbb{Z}$. Pero

$$\varepsilon = \text{mín}\{\epsilon_i\}$$

y

$$\epsilon_i = \begin{cases} c(\overrightarrow{x_i x_{i+1}}) - f(\overrightarrow{x_i x_{i+1}}) & \text{si } \overrightarrow{x_i x_{i+1}} \text{ es lado forward} \\ f(\overrightarrow{x_{i+1} x_i}) & \text{si } \overrightarrow{x_i x_{i+1}} \text{ es lado backwards} \end{cases}$$

Como las capacidades son todas enteras, ϵ_i es entero, ε es entero y por lo tanto f_{j+1} es entero.

Esto prueba que si Ford-Fulkerson termina, obtendrá un flujo maximal entero. Falta probar que efectivamente termina.

Pero $v(f_{j+1}) - v(f_j) = \varepsilon \in \mathbb{Z} > 0$, por lo tanto $\varepsilon \geq 1$. El "salto"que se produce entre un flujo y el siguiente es discreto.

Como hay una cota superior para el conjunto de valores de flujos (por ejemplo, $CAP(\{s\}) = c(s)$), esta sucesión de flujos que se incrementa en por lo menos una unidad debe acabar.

Edmonds-Karp

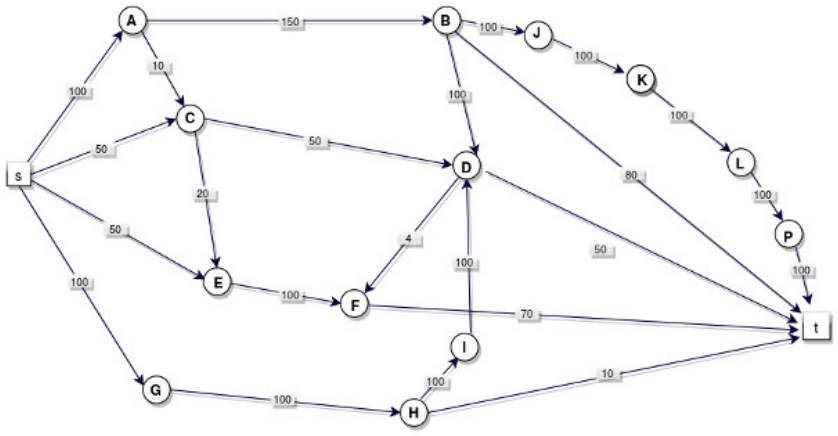
Consiste en correr Ford-Fulkerson con una pequeña modificación para garantizar que el algoritmo termina. Se trata de elegir los caminos aumentantes utilizando BFS en el grafo, a partir de s.

```
procedure EK
    // estas tres serán las variables de retorno del algoritmo

    f = 0    // El flujo maximal, inicialmente vale 0 para todos los
       lados
    v = 0    // El valor del flujo maximal
    S = s // El corte minimal

    done = false
    while not done:
        Q = (s) // cola para BFS
        ε(x) = ∞ ∀ x ∈ V
        S = {s}
        while Q ≠ ∅:
            // Por motivos de eficiencia, conviene terminar
            // este ciclo si en la cola ya está t,
            // pero para calcular la complejidad de Edmond–Karp
            // no nos interesa ese aspecto
            x = desencolar primer elemento de Q
            for z ∈ Γ*(x) − S
                if f(x→z) < c(x→z)
```

```
                encolar z en Q
                S = S ∪{z}
                A(z) = x      // El "ancestro" de z es x
                B(z) = 1      // Es un lado forward
                ε(z) = mín{ε(z), c(x⃗z⃗) − f(x⃗z⃗)}
        for z ∈ Γ⁻(x) − S
            if f(z⃗x⃗) > 0
                encolar z en Q
                S = S ∪{z}
                A(z) = x      // El "ancestro" de z es x
                B(z) = −1     // Es un lado backward
                ε(z) = mín{ε(z), f(z⃗x⃗)}
    // fin BFS
    if t ∉ S
        done = true
    else
        // reconstruir camino de s a t
        ε = ε(t)
        v = v + ε
        q = t
        while q ≠ s
            p = A(q)  // quién lo incluyó en el camino aumentante
            if B(q) = 1
                // p incluyó a q como lado forward
                f(p⃗q⃗) = f(p⃗q⃗) + ε
            else
                // p incluyó a q como lado backward
                f(q⃗p⃗) = f(q⃗p⃗) + ε
            q = p
```

Max Flow usando Edmonds-Karp

Consideremos el siguiente network:



Que se puede expresar como una lista de capacidades:

$\overrightarrow{sA}$: 100	$\overrightarrow{AB}$: 150	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: 4	$\overrightarrow{Ht}$: 10	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: 50	$\overrightarrow{AC}$: 10	$\overrightarrow{CD}$: 50	$\overrightarrow{EF}$: 100	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: 50	$\overrightarrow{Bt}$: 80	$\overrightarrow{CE}$: 20	$\overrightarrow{Ft}$: 70	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: 100	$\overrightarrow{BD}$: 100	$\overrightarrow{Dt}$: 50	$\overrightarrow{GH}$: 100	$\overrightarrow{JK}$: 100	

En cada paso, hacemos lo siguiente:

1. Corremos BFS desde s hasta que en la cola aparezca t, señalando para cada vértice que encolamos

   a) quién lo incluyó en la cola

   b) cuál es el flujo máximo que puede enviar (o devolver) ese lado

2. Cuando llegamos a t, calculamos ε a partir del camino aumentante obtenido

3. Actualizamos el valor de f.

Paso 1

Por ser el primer paso, se hace con mayor detalle. Después se expresa la corrida de BFS como una gran tabla.

Comenzamos por s, que puede incluir a A (enviando 100), a C (enviando 50), a E (enviando 50) y a G (enviando 100).

s	A	C	E	G
	s	s	s	s
	100	50	50	100

Como estos son todos los vértices que puede incluir s, lo tachamos y continuamos con el siguiente elemento de la cola, es decir, con A. A podría incluir a B (enviando 100) y a C (enviando 10). Sin embargo, no incluimos nuevamente a C porque este ya fue agregado por s. Recordar que así funciona BFS. La cola queda así:

~~s~~	A	C	E	G	B
	s	s	s	s	A
	100	50	50	100	100

A continuación, tachamos A y comenzamos a incluir a los vértices que puede agregar C:

~~s~~	~~A~~	C	E	G	B	D
	s	s	s	s	A	C
	100	50	50	100	100	50

Notar que C no agrega a E, ya que este ya estaba en la cola, pues fue agregado por s. A continuación los vértices que puede agregar E:

~~s~~	~~A~~	~~C~~	E	G	B	D	F
	s	s	s	s	A	C	E
	100	50	50	100	100	50	50

Aquí, es importante notar que $\overrightarrow{EF}$ tiene en este momento una capacidad disponible de 100, sin embargo, ponemos que E agrega a F con una capacidad de 50. Esto es así porque en esta corrida de BFS, E fue agregado por s y s sólo le podría mandar 50. A continuación, tachamos E y encolamos los vértices que agrega G:

~~s~~	~~A~~	~~C~~	~~E~~	G	B	D	F	H
	s	s	s	s	A	C	E	G
	100	50	50	100	100	50	50	100

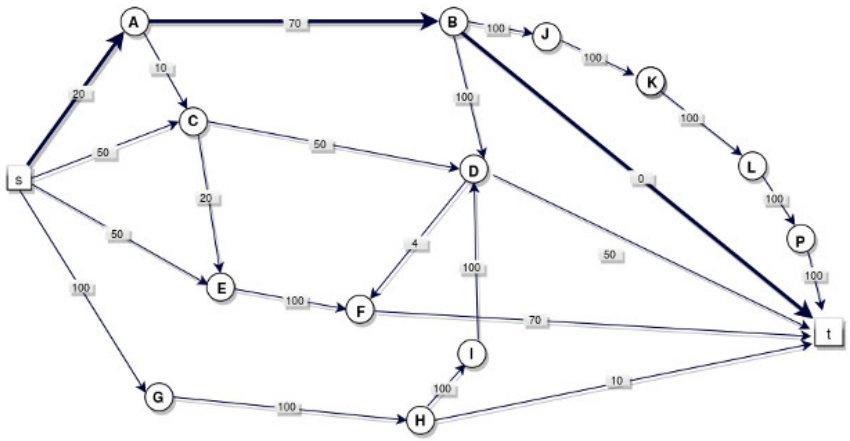
A continuación, tachamos a G y encolamos los vértices que agrega B:

~~s~~	~~A~~	~~C~~	~~E~~	~~G~~	B	D	F	H	t
	s	s	s	s	A	C	E	G	B
	100	50	50	100	100	50	50	100	80

En verdad, B podría haber agregado también a J, pero no nos interesa continuar, porque al llegar al vértice t, tenemos un camino aumentante: como t fue agregado por B que fue agregado por A, que a su vez fue agregado por s, tenemos el camino $sABt$. El máximo flujo que se puede enviar (o devolver) siguiendo este camino aumentante es 80. Por lo tanto, el resultado del paso 1 es:

$$sABt : 80$$

Acualizando el valor del flujo y las capacidades residuales, nuestro network ahora luce así:



Las etiquetas en los vértices muestran la capacidad remanente. La tabla de capacidades actualzada queda así:

$\overrightarrow{sA}$: ~~100~~20	$\overrightarrow{AB}$: ~~150~~70	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: 4	$\overrightarrow{Ht}$: 10	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: 50	$\overrightarrow{AC}$: 10	$\overrightarrow{CD}$: 50	$\overrightarrow{EF}$: 100	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: 50	$\overrightarrow{Bt}$: ~~80~~0	$\overrightarrow{CE}$: 20	$\overrightarrow{Ft}$: 70	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: 100	$\overrightarrow{BD}$: 100	$\overrightarrow{Dt}$: 50	$\overrightarrow{GH}$: 100	$\overrightarrow{JK}$: 100	

Es a partir de esta representación en forma de tabla que podemos deducir los valores que toma f_1 (el flujo correspondiente al paso 1):

$$f_1(\overrightarrow{xy}) = \begin{cases} 80 & \text{en los lados } \overrightarrow{sA}, \overrightarrow{AB}, \overrightarrow{Bt} \\ 0 & \text{en los otros lados} \end{cases}$$

Paso 2

A partir de este paso, los cómputos se abrevian y se muestra la cola BFS, y el camino aumentante y ε así como la tabla actualizada.

Cola

~~s~~	~~A~~	~~C~~	~~E~~	~~G~~	~~B~~	D	F	H	J	t
	s	s	s	s	A	C	E	G	B	D
	20	50	50	100	20	50	50	100	20	50

Camino aumentante

$sCDt$: 50

Capacidades

$\overrightarrow{sA}$: ~~100~~20	$\overrightarrow{AB}$: ~~150~~70	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: 4	$\overrightarrow{Ht}$: 10	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: ~~50~~0	$\overrightarrow{AC}$: 10	$\overrightarrow{CD}$: ~~50~~0	$\overrightarrow{EF}$: 100	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: 50	$\overrightarrow{Bt}$: ~~80~~0	$\overrightarrow{CE}$: 20	$\overrightarrow{Ft}$: 70	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: 100	$\overrightarrow{BD}$: 100	$\overrightarrow{Dt}$: ~~50~~0	$\overrightarrow{GH}$: 100	$\overrightarrow{JK}$: 100	

Paso 3

Cola

~~s~~	~~A~~	~~E~~	~~G~~	~~B~~	~~C~~	F	H	D	J	t
	s	s	s	A	A	E	G	B	B	F
	20	50	100	20	10	50	100	20	20	50

Camino Aumentante

$sEFt$: 50

Capacidades

$\overrightarrow{sA}$: ~~100~~20	$\overrightarrow{AB}$: ~~150~~70	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: 4	$\overrightarrow{Ht}$: 10	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: ~~50~~	$\overrightarrow{AC}$: 10	$\overrightarrow{CD}$: ~~50~~	$\overrightarrow{EF}$: ~~100~~50	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: ~~50~~	$\overrightarrow{Bt}$: ~~80~~	$\overrightarrow{CE}$: 20	$\overrightarrow{Ft}$: ~~70~~20	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: 100	$\overrightarrow{BD}$: 100	$\overrightarrow{Dt}$: ~~50~~	$\overrightarrow{GH}$: 100	$\overrightarrow{JK}$: 100	

Paso 4

Cola

~~s~~	~~A~~	~~G~~	~~B~~	~~C~~	H	D	J	E	t
	s	s	A	A	G	B	B	C	H
	20	100	20	10	100	20	20	10	10

Camino aumentante

$sGHt$: 10

Capacidades

$\overrightarrow{sA}$: ~~100~~20	$\overrightarrow{AB}$: ~~150~~70	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: 4	$\overrightarrow{Ht}$: ~~10~~	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: ~~50~~	$\overrightarrow{AC}$: 10	$\overrightarrow{CD}$: ~~50~~	$\overrightarrow{EF}$: ~~100~~50	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: ~~50~~	$\overrightarrow{Bt}$: ~~80~~	$\overrightarrow{CE}$: 20	$\overrightarrow{Ft}$: ~~70~~20	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: ~~100~~90	$\overrightarrow{BD}$: 100	$\overrightarrow{Dt}$: ~~50~~	$\overrightarrow{GH}$: ~~100~~90	$\overrightarrow{JK}$: 100	

Paso 5

Cola

~~s~~	~~A~~	~~G~~	~~B~~	~~C~~	~~H~~	~~D~~	~~J~~	~~E~~	~~I~~	F	K	t
	s	s	A	A	G	B	B	C	H	D	J	F
	20	90	20	10	90	20	20	10	90	4	20	4

Camino aumentante

$sABDFt$: 4

Capacidades

$\overrightarrow{sA}$: ~~100~~~~20~~16	$\overrightarrow{AB}$: ~~150~~~~70~~66	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: ~~4~~	$\overrightarrow{Ht}$: ~~10~~	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: ~~50~~	$\overrightarrow{AC}$: 10	$\overrightarrow{CD}$: ~~50~~	$\overrightarrow{EF}$: ~~100~~50	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: ~~50~~	$\overrightarrow{Bt}$: ~~80~~	$\overrightarrow{CE}$: 20	$\overrightarrow{Ft}$: ~~70~~~~20~~16	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: ~~100~~90	$\overrightarrow{BD}$: ~~100~~96	$\overrightarrow{Dt}$: ~~50~~	$\overrightarrow{GH}$: ~~100~~90	$\overrightarrow{JK}$: 100	

Paso 6

Cola

~~s~~	~~A~~	~~G~~	~~B~~	~~C~~	~~H~~	~~D~~	~~J~~	~~E~~	~~I~~	~~K~~	F	t
	s	s	A	A	G	B	B	C	H	J	E	F
	16	90	16	10	90	16	16	10	90	16	10	10

Camino aumentante

$sACEFt$: 10

Capacidades

$\overrightarrow{sA}$: ~~100~~~~20~~~~16~~6	$\overrightarrow{AB}$: ~~150~~~~70~~66	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: ~~4~~	$\overrightarrow{Ht}$: ~~10~~	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: ~~50~~	$\overrightarrow{AC}$: ~~10~~	$\overrightarrow{CD}$: ~~50~~	$\overrightarrow{EF}$: ~~100~~~~50~~40	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: ~~50~~	$\overrightarrow{Bt}$: ~~80~~	$\overrightarrow{CE}$: ~~20~~10	$\overrightarrow{Ft}$: ~~70~~~~20~~~~16~~6	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: ~~100~~90	$\overrightarrow{BD}$: ~~100~~96	$\overrightarrow{Dt}$: ~~50~~	$\overrightarrow{GH}$: ~~100~~90	$\overrightarrow{JK}$: 100	

Paso 7

Cola

~~s~~	~~A~~	~~G~~	~~B~~	~~H~~	~~D~~	~~J~~	~~I~~	~~C~~	~~K~~	~~E~~	~~L~~	F	P	t
	s	s	A	G	B	B	H	D^-	J	C	K	E	L	F
	6	90	6	90	6	6	90	6	6	6	6	6	6	6

Hemos incluido el símbolo D^- para recordar el hecho de que en realidad D está devolviendo flujo a C. Recordemos que en el network existe el lado $\overrightarrow{CD}$, por lo que la sucesión $\overleftarrow{DC}$ es en realidad un lado backwards.

Camino aumentante

$s\overleftarrow{ABDC}EFt : 6$

Capacidades

$\overrightarrow{sA} : \cancel{100}\,\cancel{20}\,\cancel{16}\,\cancel{6}$	$\overrightarrow{AB} : \cancel{150}\,\cancel{70}\,\cancel{66}\,60$	$\overrightarrow{BJ} : 100$	$\overrightarrow{DF} : \cancel{4}$	$\overrightarrow{Ht} : \cancel{10}$	$\overrightarrow{KL} : 100$
$\overrightarrow{sC} : \cancel{50}$	$\overrightarrow{AC} : \cancel{10}$	$\overrightarrow{CD} : \cancel{50}\,6$	$\overrightarrow{EF} : \cancel{100}\,\cancel{50}\,\cancel{40}\,34$	$\overrightarrow{HI} : 100$	$\overrightarrow{LP} : 100$
$\overrightarrow{sE} : \cancel{50}$	$\overrightarrow{Bt} : \cancel{80}$	$\overrightarrow{CE} : \cancel{20}\,\cancel{10}\,4$	$\overrightarrow{Ft} : \cancel{70}\,\cancel{20}\,\cancel{16}\,\cancel{6}$	$\overrightarrow{ID} : 100$	$\overrightarrow{Pt} : 100$
$\overrightarrow{sG} : \cancel{100}\,90$	$\overrightarrow{BD} : \cancel{100}\,\cancel{96}\,90$	$\overrightarrow{Dt} : \cancel{50}$	$\overrightarrow{GH} : \cancel{100}\,90$	$\overrightarrow{JK} : 100$	

Paso 8

Cola

$\cancel{s}$	$\cancel{G}$	$\cancel{H}$	$\cancel{I}$	$\cancel{D}$	$\cancel{B}$	$\cancel{C}$	$\cancel{F}$	$\cancel{A}$	$\cancel{J}$	$\cancel{E}$	$\cancel{K}$	$\cancel{L}$	P	t
	s	G	H	I	D^-	D^-	D^-	B^-	B	F^-	J	K	L	P
	90	90	90	90	10	44	4	10	10	4	10	10	10	10

Camino aumentante

$sGHI\overleftarrow{DB}JKLPt : 10$

Capacidades

$\overrightarrow{sA} : \cancel{100}\,\cancel{20}\,\cancel{16}\,\cancel{6}$	$\overrightarrow{AB} : \cancel{150}\,\cancel{70}\,\cancel{66}\,60$	$\overrightarrow{BJ} : \cancel{100}\,90$	$\overrightarrow{DF} : \cancel{4}$	$\overrightarrow{Ht} : \cancel{10}$	$\overrightarrow{KL} : \cancel{100}\,90$
$\overrightarrow{sC} : \cancel{50}$	$\overrightarrow{AC} : \cancel{10}$	$\overrightarrow{CD} : \cancel{50}\,6$	$\overrightarrow{EF} : \cancel{100}\,\cancel{50}\,\cancel{40}\,34$	$\overrightarrow{HI} : \cancel{100}\,90$	$\overrightarrow{LP} : \cancel{100}\,90$
$\overrightarrow{sE} : \cancel{50}$	$\overrightarrow{Bt} : \cancel{80}$	$\overrightarrow{CE} : \cancel{20}\,\cancel{10}\,4$	$\overrightarrow{Ft} : \cancel{70}\,\cancel{20}\,\cancel{16}\,\cancel{6}$	$\overrightarrow{ID} : \cancel{100}\,90$	$\overrightarrow{Pt} : \cancel{100}\,\cancel{90}\,80$
$\overrightarrow{sG} : \cancel{100}\,\cancel{90}\,80$	$\overrightarrow{BD} : \cancel{100}\,\cancel{96}\,\cancel{90}\,100$	$\overrightarrow{Dt} : \cancel{50}$	$\overrightarrow{GH} : \cancel{100}\,\cancel{90}\,80$	$\overrightarrow{JK} : \cancel{100}\,90$	

Paso 9

Cola

$\cancel{s}$	$\cancel{G}$	$\cancel{H}$	$\cancel{I}$	$\cancel{D}$	$\cancel{C}$	$\cancel{F}$	$\cancel{A}$	$\cancel{E}$	$\cancel{B}$	$\cancel{J}$	$\cancel{K}$	$\cancel{L}$	P	t
	s	G	H	I	D^-	D^-	C^-	C	A	B	J	K	L	P
	80	80	80	80	44	4	10	4	10	10	10	10	10	10

Camino aumentante

$sGHI\overleftarrow{DC}ABJKLPt : 10$

Capacidades

$\overrightarrow{sA} : \cancel{100}\,\cancel{20}\,\cancel{16}\,\cancel{6}$	$\overrightarrow{AB} : \cancel{150}\,\cancel{70}\,\cancel{66}\,\cancel{60}\,50$	$\overrightarrow{BJ} : \cancel{100}\,90$	$\overrightarrow{DF} : \cancel{4}$	$\overrightarrow{Ht} : \cancel{10}$	$\overrightarrow{KL} : \cancel{100}\,\cancel{90}\,80$
$\overrightarrow{sC} : \cancel{50}$	$\overrightarrow{AC} : \cancel{10}\,\cancel{0}\,10$	$\overrightarrow{CD} : \cancel{50}\,\cancel{0}\,16$	$\overrightarrow{EF} : \cancel{100}\,\cancel{50}\,\cancel{40}\,34$	$\overrightarrow{HI} : \cancel{100}\,\cancel{90}\,80$	$\overrightarrow{LP} : \cancel{100}\,\cancel{90}\,80$
$\overrightarrow{sE} : \cancel{50}$	$\overrightarrow{Bt} : \cancel{80}$	$\overrightarrow{CE} : \cancel{20}\,\cancel{10}\,4$	$\overrightarrow{Ft} : \cancel{70}\,\cancel{20}\,\cancel{16}\,\cancel{6}$	$\overrightarrow{ID} : \cancel{100}\,\cancel{90}\,80$	$\overrightarrow{Pt} : \cancel{100}\,90$
$\overrightarrow{sG} : \cancel{100}\,\cancel{90}\,\cancel{80}\,70$	$\overrightarrow{BD} : \cancel{100}\,\cancel{96}\,\cancel{90}\,100$	$\overrightarrow{Dt} : \cancel{50}$	$\overrightarrow{GH} : \cancel{100}\,\cancel{90}\,\cancel{80}\,70$	$\overrightarrow{JK} : \cancel{100}\,\cancel{90}\,80$	

Paso 10

Cola

$\cancel{s}$	$\cancel{G}$	$\cancel{H}$	$\cancel{I}$	$\cancel{D}$	$\cancel{C}$	$\cancel{F}$	$\cancel{E}$
	s	G	H	I	D^-	D^-	C^-
	70	70	70	70	34	4	4

La cola se vació y nunca llegamos a t. Esto significa que hemos llegado al final del algoritmo, porque ya no se puede construir un camino aumentante entre s y t.

Valor del flujo $v(f_{MAX}) = f_{MAX}(\overrightarrow{sA}) + f_{MAX}(\overrightarrow{sC}) + f_{MAX}(\overrightarrow{sE}) + f_{MAX}(\overrightarrow{sG}) = 230$.

Además (y como comprobación de que obtuvimos un resultado correcto) la cola final (la que no completa un camino aumentante) es un corte minimal y su capacidad debe ser igual al valor del flujo maximal obtenido:

$S = \{s, G, H, I, D, C, E, F\}$

$CAP(S) = C(S, V - S) = c(\overrightarrow{sA}) + c(\overrightarrow{Ht}) + c(\overrightarrow{Dt}) + c(\overrightarrow{Ft}) = 100 + 10 + 50 + 70 = 230$

Complejidad y correctitud de Edmonds-Karp

La complejidad de Edmonds-Karp es $O(nm^2)$. En particular, siempre termina.

Observación: Para grafos ralos (es decir, con pocos lados) m se acerca a n y podemos decir que la complejidad de Edmonds-Karp es $O(n^3)$, mientras que en grafos no ralos (cercanos a grafos completos), m se aproxima a n^2 y podemos decir que la complejidad de Edmonds-Karp es $O(n^5)$.

Prueba

Probaremos primero un hecho por inducción. Para enunciarlo, sea $f_0 = 0, f_1, f_2, f_3, \ldots$ la sucesión de flujos producida por el algoritmo Edmonds-Karp. Hasta ahora no sabemos si esa sucesión termina, así que puede ser una sucesión infinita.

Para cada paso k del algoritmo y cada vértice x perteneciente al network, definimos dos funciones:

- $d_k(s) = 0$
- $d_k(x) =$ longitud (número de lados) del camino aumentante más corto entre s y x, si tal camino existe

y

- $b_k(t) = 0$
- $b_k(x) =$ longitud del camino aumentante más corto entre x y t, si tal camino existe.

Queremos probar que

$$d_k(x) \leq d_{k+1}(x) \text{ y } b_k(x) \leq b_{k+1}(x)$$

En la práctica, al correr el algoritmo, esto se traduce en que los caminos aumentantes encontrados nunca son más cortos que el anterior. Queremos probar que esto se cumple siempre. Lo probaremos para d_k, siendo la prueba para b_k análoga.

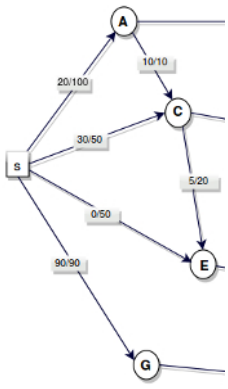
Hipótesis inductiva (inducción en i)

$H(i)$: Para todo vértice z tal que $d_{k+1}(z) \leq i$ vale que $d_k(z) \leq d_{k+1}(z)$

Esta es la propiedad que queremos probar que se cumple para todo número natural i. Dicho en palabras, si la longitud del camino aumentante más corto entre s y z en el paso $k+1$ (con respecto al flujo f_k) es menor o igual que i, entonces dicha longitud es mayor o igual a la longitud del camino aumentante más corto entre s y z con respecto al paso anterior (paso k, relativo al flujo f_{k-1}).

Caso base: $H(0)$.

Tenemos que ver que para todo vértice z tal que $d_{k+1}(z) \leq 0$ vale que $d_k(z) \leq d_{k+1}(z)$. Pero si $d_{k+1}(z) \leq 0$ debe ser $z = s$, entonces $d_k(z) = d_k(s) = 0 \leq d_{k+1}(s) = d_{k+1}(z)$. La siguiente imagen ayuda a comprender esta situación: $d_j(s) = 0$ para todo $j \in \mathbb{N}_0$.

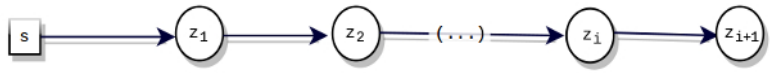


Caso inductivo: $H(i) \Rightarrow H(i+1)$

Sea z con $d_{k+1} \leq i+1$. Hay dos posibilidades, o bien $d_{k+1} \leq i$ o bien $d_{k+1}(z) = i+1$. En el primer caso, por hipótesis inductiva sabemos que $d_k(z) \leq d_{k+1}(z)$. Concentrémonos entonces en este último caso.

Si $d_{k+1}(z) = i+1$ entonces existe un camino aumentante (relativo a f_k) de la forma

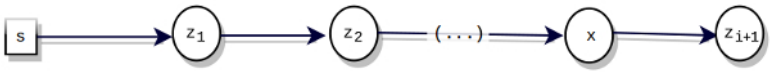
$$z_0 = s, z_1, z_2, \ldots, z_i, z_{i+1} = z$$



Sea $x = z_i$ (el vértice anterior en dicho camino aumentante).

CASO 1. Existe algún camino aumentante relativo a f_{k-1} de la forma

$$z_0 = s, z_1, \ldots, x, z$$



Notar que la situación en el paso anterior no necesariamente era igual, ya que no sabemos qué pasa entre z_2 y $z_i = x$ en ninguno de los dos paso (tanto en el paso k como en el paso $k+1$). Sin embargo, podemos afirmar que $d_k(z) \leq d_k(x) + 1$, pues al haber un camino $s, \ldots, x, z$ de longitud $d_k(x) + 1$ sabemos que el mínimo de las longitudes de los caminos aumentantes entre s y z debe ser menor o igual a eso (puede haber otros menores). La utilidad de este hecho se verá al concluir el caso 2.

CASO 2. No existe un tal camino relativo a f_{k-1}.Pero recordemos que estamos considerando que $d_{k+1}(z) = i+1$, o sea que por definición sí existe relativo a f_k(ej: $z_0 = s, z_1, \ldots, z_i, z_{i+1} = z$). El "pseudolado"$xz$ no está disponible en el paso k: o bien $\exists\ \overrightarrow{xz}$ y está saturado, o bien $\exists\ \overrightarrow{zx}$ y está vacío. Pero el paso k se hace relativo al f_{k-1}, y cuando llegamos la paso $k+1$ el lado xz está disponible de una de dos maneras:

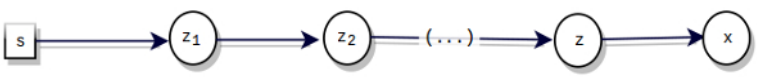
1. $f_{k-1}(\overrightarrow{xz}) = c(\overrightarrow{xz})$ pero $f_k(\overrightarrow{xz}) < c(\overrightarrow{xz})$, o bien
2. $f_{k+1}(\overrightarrow{zx}) = 0$ pero $f_k(\overrightarrow{zx}) > 0$

$1 \Rightarrow f_k$ devuelve flujo por $\overrightarrow{xz}$, por lo tanto para construir f_k utilizamos $\overrightarrow{zx}$ (lado backwards), usando un camino

$$z_0 = s, z_1, \ldots, z, x$$

$2 \Rightarrow f_k$ manda flujo por $\overrightarrow{zx}$ y se concluye lo mismo que antes: tiene que haber un camino (y lo usamos) de la forma

$$z_0 = s, z_1, \ldots, z, x$$



Como Edmonds-Karp funciona con BFS, este camino usado para construir f_k es de longitud mínima, por ende

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

Esto implica que $d_k(z) = d_k(x) - 1 \leq d_k(x) + 1$.
Notemos que tanto en el **CASO 1** como en el **CASO 2** llegamos a la conclusión de que

$$d_k(z) \leq d_k(x) + 1 \qquad (1)$$

Ahora bien, $d_{k+1}(x) = d_{k+1}(z_i) = i$, por lo tanto vale la hipótesis inductiva para x y entonces tenemos que

$$d_k(x) \leq d_{k+1}(x) \qquad (2)$$

Por (1) y (2) se concluye que $d_k(z) \leq d_k(x) + 1 \leq d_{k+1}(x) + 1 = i + 1 = d_{k+1}(z)$.
Así concluye la prueba de que la propiedad $H(i)$ vale para todo $i \in \mathbb{N}_0$.
Con esta herramienta, analicemos la complejidad del algoritmo.

El **while** interno (**while** $Q \neq \emptyset$) implementa

- BFS, $O(\sum_{x \in V} d(x)) = O(2m) = O(m)$
- Aumentar el flujo, $O(n)$, pues aumenta el flujo a lo largo de un camino aumentante, que tiene a lo sumo n vértices.

Luego, el **while** interno tiene complejidad $O(m) + O(n) = O(m)$
Por lo tanto, la complejidad de Edmonds-Karp es:

$$O(m) * \text{ (número de veces que se ejecuta el } \textbf{while} \text{ interno)}$$

Es decir, $O(m)$ por un número determinado de veces, hasta que deja de ser cierta la condición done $\Leftrightarrow t \notin S$. Es decir, hasta que no podamos encontrar un camino aumentante entre s y t.

Pero el número de caminos aumentantes que podemos encontrar es menor o igual al número de lados (m) por la cantidad de veces que un lado se puede saturar o vaciar. Necesitamos acotar este último valor.

Supongamos que al construir el flujo en el paso k, el lado $\overrightarrow{xz}$ se satura o el lado $\overrightarrow{zx}$ se vacía, tenemos camino

$$s \dots xz \dots t$$

y este camino se utiliza para construir f_k. Tenemos que

$$\begin{aligned} d_k(t) &= d_k(x) + b_k(x) \\ &= d_k(x) + 1 + b_k(z) \end{aligned}$$

Como $\overrightarrow{xz}$ se saturó o $\overrightarrow{zx}$ se vació, para volverlo a saturar (vaciar) lo tengo que vaciar (llenar al menos un poco). Por lo tanto, antes de saturarlo o vaciarlo dbe haber un j tal que para construir f_j debemos usar un camino aumentante de la forma

$$s \dots zx \dots t$$

Por lo tanto,

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

Por lo tanto, antes de poder saturarse o vaciarse otra vez, la longitud del menor camino aumentante de s a t debe aumentar en al menos 2. Concluimos que puede haber a lo sumo $n/2$ caminos aumentantes en los que se sature (o vacíe) un mismo lado.

Entonces, la complejidad de Edmonds-Karp es:

$$O(m.m.n) = O(nm^2)$$

Dinic

Se trata de un algoritmo para buscar el flujo maximal en un network, que constituye una mejora en la complejidad del algoritmo de Edmonds-Karp.

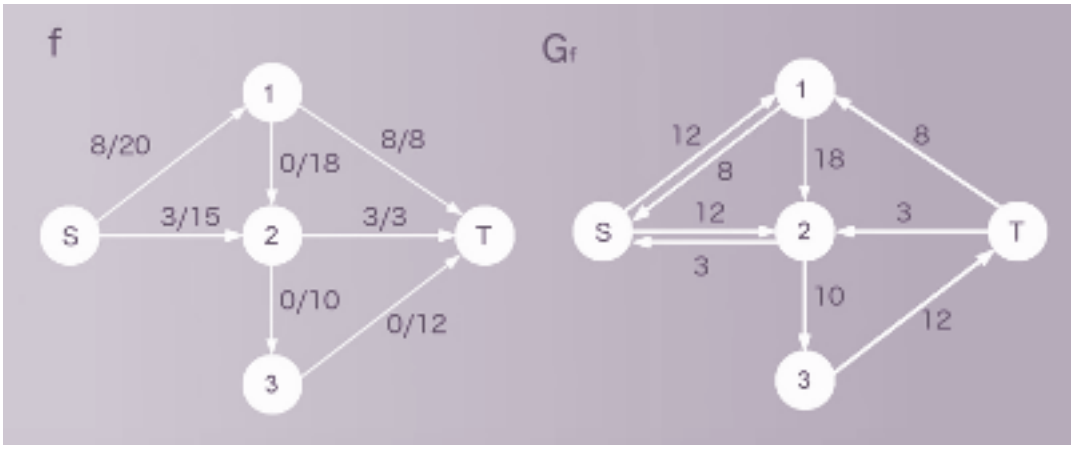
Idea general

Dado un network N y un flujo f, se construye un network auxiliar NA_f, donde se encuentra un flujo bloqueante g. Con f y g se construye f^*.

Network Residual

El Network Auxilar de Dinic constituye un refinamiento del Network Residual (NR). En un NR, están todos los vértices, y por cada lado $\overrightarrow{xy} \in E(N)$ se crean 0, 1 o 2 lados:

- Si $f(\overrightarrow{xy}) < c(\overrightarrow{xy})$ en N, se crea el lado $\overrightarrow{xy}$ en NR, con capacidad $c(\overrightarrow{xy}) - f(\overrightarrow{xy})$ (capacidad **residual**).
- Si $f(\overrightarrow{xy}) > 0$ en N, se crea el lado $\overrightarrow{yx}$ en NR, con capacidad $f(\overrightarrow{xy})$.

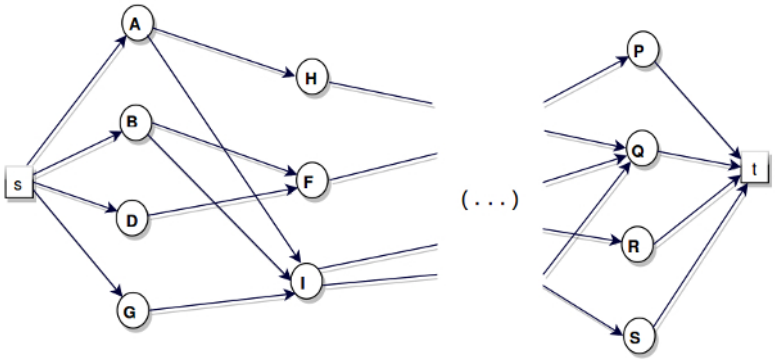


Network Auxiliar

El Network Auxiliar (NA) es un subnetwork del Network Residual, obtenido de la siguiente forma:

1. Se calculan las distancias de s a x para todo vértice x como en la prueba de Edmonds-Karp (longitud del menor camino aumentante entre s y x).
2. Sólo dejamos el lado $\overrightarrow{xy}$ si $d(y) = d(x) + 1$
3. Eliminamos todos los vértices x y los lados incidentes con ellos tales que $d(x) \geq d(t)$, a excepción del mismo t.

Por construcción, el Network auxiliar será un network por niveles:



Trabajar en este network es más fácil utilizando DFS.

Blocking Flow

Un Blocking Flow o flujo bloqueante en un NA es un flujo g tal que todo camino dirigido (no aumentante) de s a t tiene un lado $\overrightarrow{xy}$ con $g(\overrightarrow{xy}) = c(\overrightarrow{xy})$. Todos los caminos tienen al menos un lado saturado y por lo tanto ya no se puede mandar más por ningún lado.

Algoritmo

1. Construir un NA utilizando BFS.
2. Encontrar flujo bloqueante usando greedy con DFS.
3. Repetir hasta que no se pueda construir un NA porque s y t quedan inconexos.

Si f es el flujo actual en el network N, g es el flujo bloqueante obtenido sobre NA, entonces el flujo f^* se construye así:

- Si $\overrightarrow{xy} \in E(NA)$ viene de $\overrightarrow{xy} \in E(N)$, entonces $f^*(\overrightarrow{xy}) = f(\overrightarrow{xy}) + g(\overrightarrow{xy})$ donde f es el flujo del paso anterior en el network N y g es el flujo en el NA.
- Si $\overrightarrow{xy} \in E(NA)$ corresponde a $\overrightarrow{yx} \in E(N)$, entonces $f^*(\overrightarrow{yx}) = f(\overrightarrow{yx}) - g(\overrightarrow{xy})$

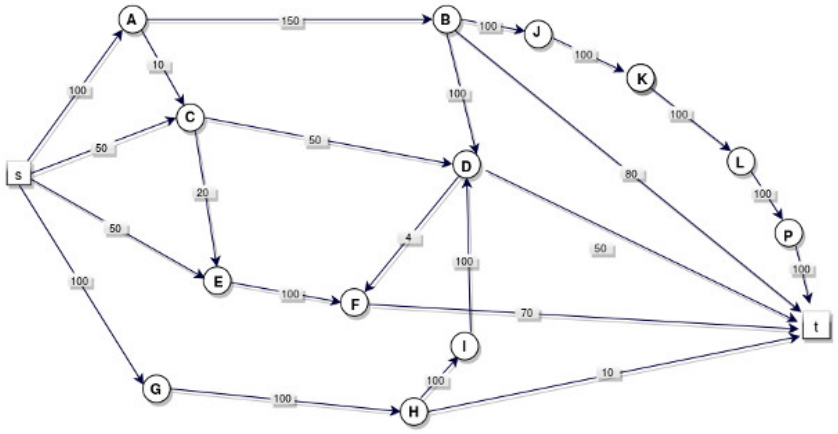
Pseudocódigo

```
procedure DINIC(Input: NA, Output: g bloqueante en NA)
// a partir de aquí, todos los símbolos hacen referencia al network
   auxiliar
g = 0
d = d(t)
P = array con [0, 1, ..., d(t)] elementos
done = False
while not done:
    i = 0
    while (i < d) and not done:
        // buscamos llegar hasta t

        if (Γ+(P[i]) ≠ ∅):
            // AVANZAR
            P[i+1] = algún vértice en Γ+(P[i])
            i++;
            // FIN AVANZAR
        else:
```

```
            if i ≠ 0:
                // borramos el lado para no volver a intentarlo
                // RETROCEDER
                Borrar P[i−1]P[i] del NA
                i−−;
                // FIN RETROCEDER
            else:
                 done = True
    if i = d:
        // tenemos camino aumentante entre s y t
        // INCREMENTAR
        ε = mín{c(P[i]P[i+1]) − g(P[i]P[i+1]) ∀ 0 ≤ i ≤ d}
        for i= 0 to d −1:
            g+= ε
            if g(P[i]P[i+1]) = c(P[i]P[i+1]):
                borrar lado P[i]P[i+1] del NA
        // FIN INCREMENTAR
```

Max-Flow usando Dinic

Calcularemos el flujo maximal del mismo network con el que se ejemplificó el algoritmo Edmonds-Karp:

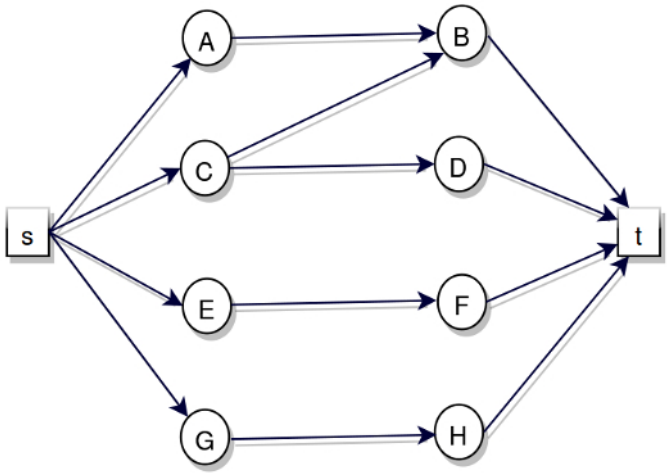


dado por la siguiente lista de capacidades:

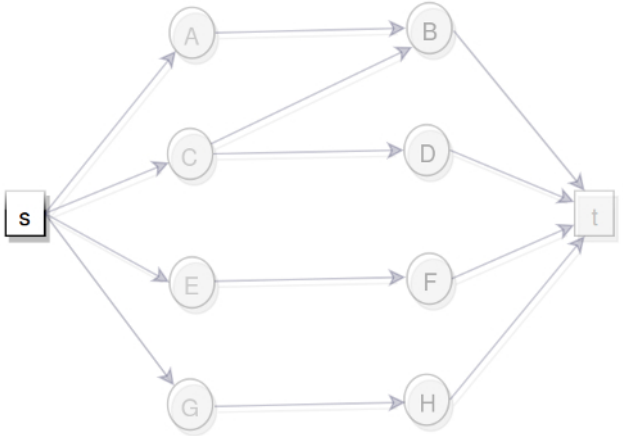
$\overrightarrow{sA}$: 100	$\overrightarrow{AB}$: 150	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: 4	$\overrightarrow{Ht}$: 10	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: 50	$\overrightarrow{AC}$: 10	$\overrightarrow{CD}$: 50	$\overrightarrow{EF}$: 100	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: 50	$\overrightarrow{Bt}$: 80	$\overrightarrow{CE}$: 20	$\overrightarrow{Ft}$: 70	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: 100	$\overrightarrow{BD}$: 100	$\overrightarrow{Dt}$: 50	$\overrightarrow{GH}$: 100	$\overrightarrow{JK}$: 100	

Paso 1

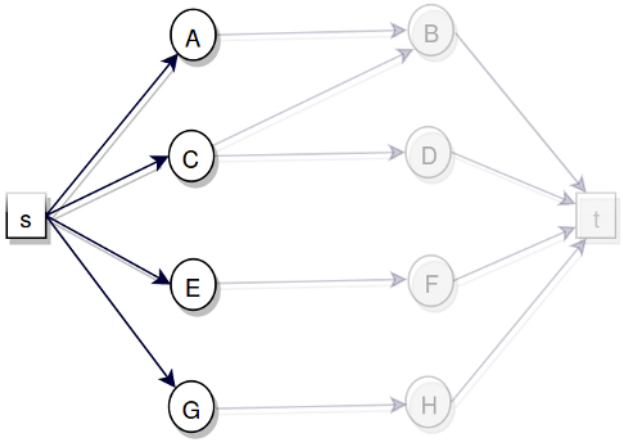
El Network auxiliar es el siguiente:



En la siguiente serie de imágenes se ve cómo se va fue construyendo dicho NA, utilizando BFS. Se empezó agregando a s:



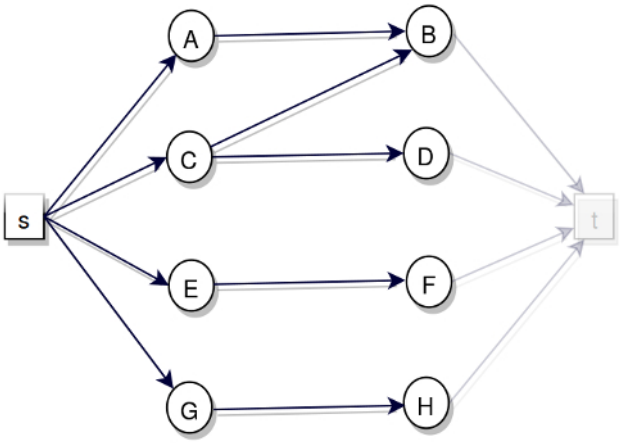
Como desde s tenemos caminos aumentantes de longitud 1 hasta A, C, E y G, estos vértices son agregados por s (y constituyen el primer nivel del NA).



Corriendo BFS, terminamos de agregar a los vecinos de s y continuamos con los vecinos de A. En el Network Residual, relativo al flujo $f_0 = 0$, los vecinos de A para los cuales se cumple que la longitud del menor camino aumentante desde s hasta ellos es 2 (1 más que la longitud del menor

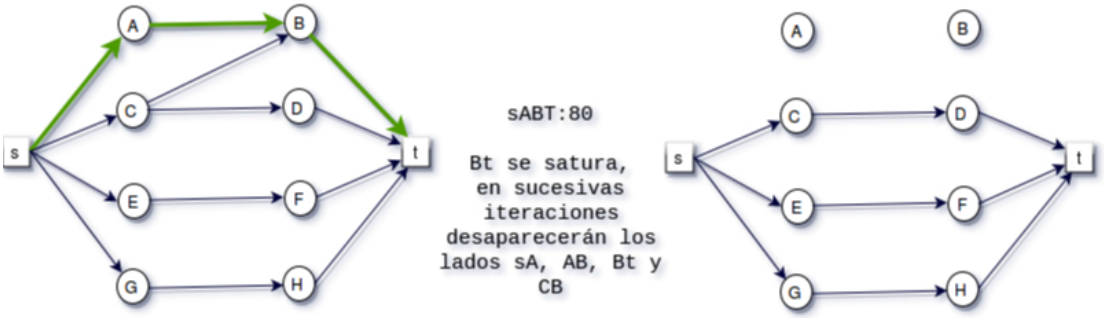
camino aumentante hasta A, es decir $d(A)+1$) son $\{B\}$, así que A agrega a B en el segundo nivel del NA (y lo encola para BFS).

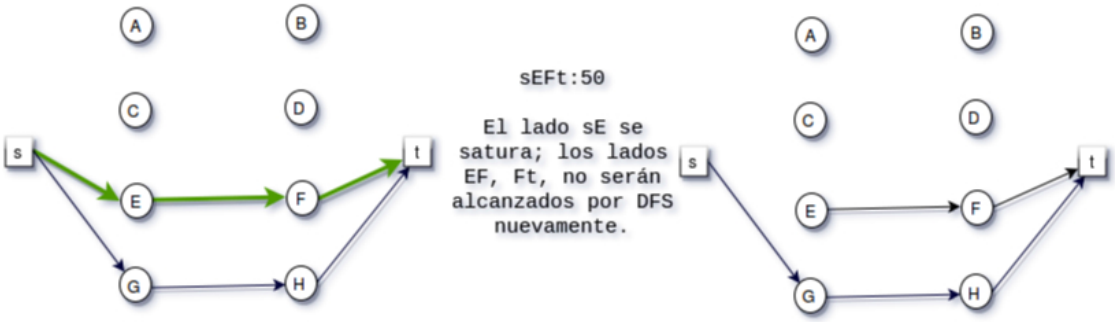
Luego consideramos los vértices que podría agregar C. Estos son $\{B, D\}$. Notemos que aunque B ya fue agregado por A, el lado $\overrightarrow{CB}$ cumple con las condiciones impuestas por el algoritmo: dejamos el lado $\overrightarrow{CB}$ porque este existe en el Network residual y además $d(B) = d(C) + 1$. Así agregamos los vecinos de E, de F, y se forma el segundo nivel del NA:

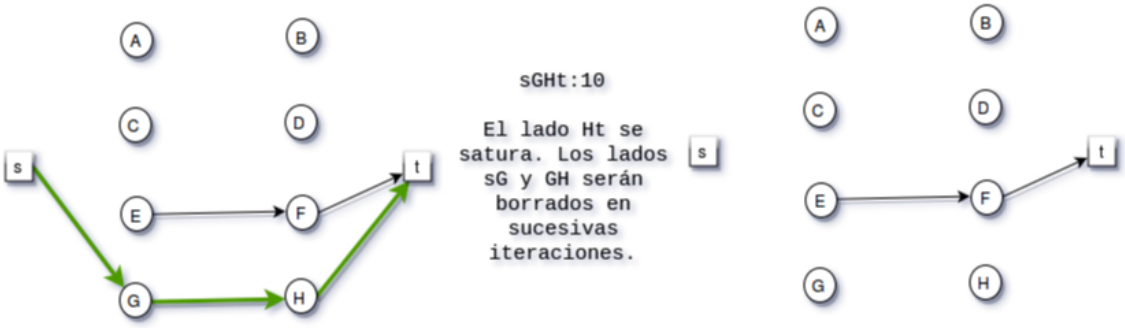


Como desde todos ellos se puede alcanzar t, el NA se concluye en el siguiente paso (por más que BFS podría seguir).

Ahora bien, utilizando DFS en el NA, buscamos caminos aumentantes. En cada camino aumentante, se satura o se vacía al menos un lado, de modo que el NA va "perdiendo alguno de sus lados".





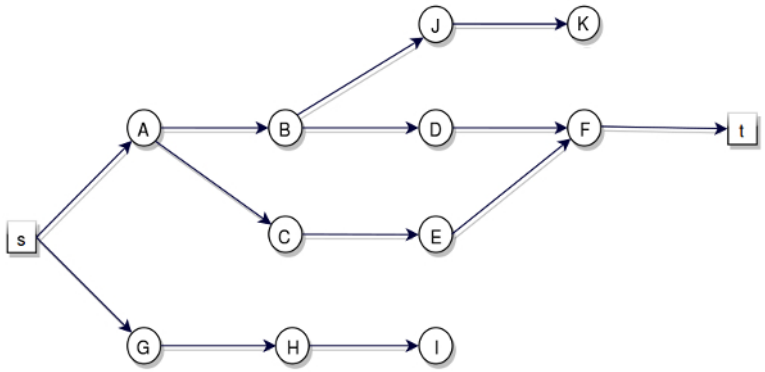


La lista de capacidades queda actualizada así:

$\overrightarrow{sA}$: ~~10~~20	$\overrightarrow{AB}$: ~~15~~70	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: 4	$\overrightarrow{Ht}$: ~~10~~0	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: ~~5~~0	$\overrightarrow{AC}$: 10	$\overrightarrow{CD}$: ~~5~~0	$\overrightarrow{EF}$: ~~10~~50	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: ~~5~~0	$\overrightarrow{Bt}$: ~~8~~0	$\overrightarrow{CE}$: 20	$\overrightarrow{Ft}$: ~~7~~20	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: ~~10~~90	$\overrightarrow{BD}$: 100	$\overrightarrow{Dt}$: ~~5~~0	$\overrightarrow{GH}$: ~~10~~90	$\overrightarrow{JK}$: 100	

Paso 2

Network Auxiliar:



Caminos aumentantes

$sABDFt : 4$

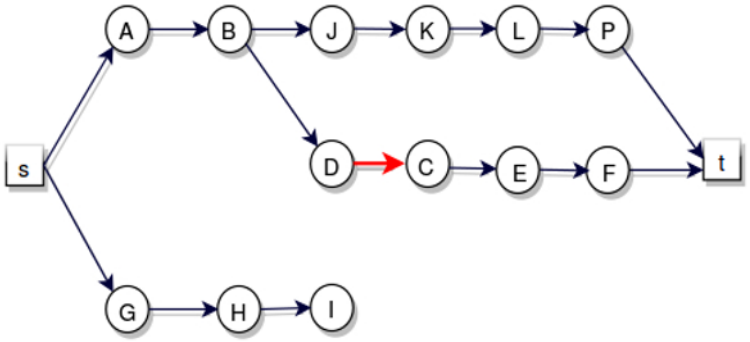
$sACEFt : 10$

Capacidades actualizadas

$\overrightarrow{sA}$: ~~100~~ ~~20~~ ~~16~~ 6	$\overrightarrow{AB}$: ~~15~~ ~~7~~ 0 66	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: ~~4~~ 0	$\overrightarrow{Ht}$: ~~1~~ 0	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: ~~5~~ 0	$\overrightarrow{AC}$: ~~1~~ 0	$\overrightarrow{CD}$: ~~5~~ 0	$\overrightarrow{EF}$: ~~100~~ ~~50~~ 40	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: ~~5~~ 0	$\overrightarrow{Bt}$: ~~8~~ 0	$\overrightarrow{CE}$: ~~2~~ 0 10	$\overrightarrow{Ft}$: ~~7~~ 0 ~~20~~ ~~16~~ 6	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: ~~10~~ 0 90	$\overrightarrow{BD}$: ~~10~~ 0 96	$\overrightarrow{Dt}$: ~~5~~ 0	$\overrightarrow{GH}$: ~~10~~ 0 90	$\overrightarrow{JK}$: 100	

Paso 3

Network auxiliar



Notar que el lado $\overrightarrow{DC}$ del network auxiliar corresponde a un lado backwards en el network original.

Caminos aumentantes

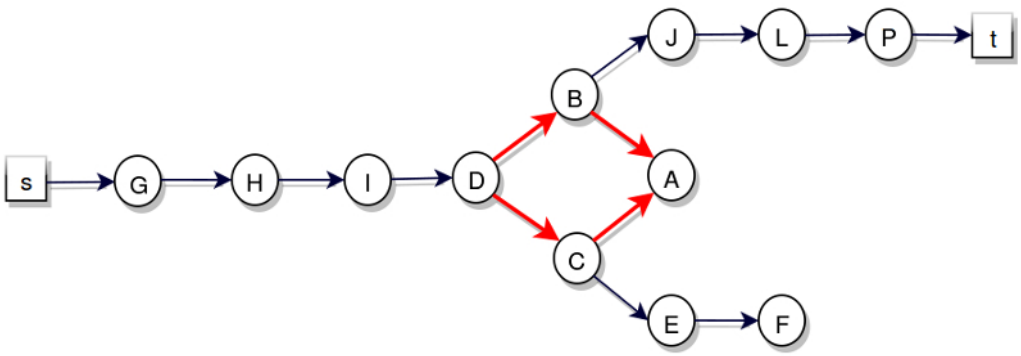
$$sAB\overleftarrow{DC}EFt : 6$$

Capacidades actualizadas

$\overrightarrow{sA}$: ~~100~~ ~~20~~ ~~16~~ ~~6~~ 0	$\overrightarrow{AB}$: ~~15~~ ~~7~~ 0 ~~6~~ ~~6~~ 60	$\overrightarrow{BJ}$: 100	$\overrightarrow{DF}$: ~~4~~ 0	$\overrightarrow{Ht}$: ~~1~~ 0	$\overrightarrow{KL}$: 100
$\overrightarrow{sC}$: ~~5~~ 0	$\overrightarrow{AC}$: ~~1~~ 0	$\overrightarrow{CD}$: ~~5~~ 0 ~~6~~	$\overrightarrow{EF}$: ~~100~~ ~~50~~ ~~4~~ 0 34	$\overrightarrow{HI}$: 100	$\overrightarrow{LP}$: 100
$\overrightarrow{sE}$: ~~5~~ 0	$\overrightarrow{Bt}$: ~~8~~ 0	$\overrightarrow{CE}$: ~~2~~ 0 ~~1~~ 4	$\overrightarrow{Ft}$: ~~7~~ 0 ~~20~~ ~~16~~ ~~6~~ 0	$\overrightarrow{ID}$: 100	$\overrightarrow{Pt}$: 100
$\overrightarrow{sG}$: ~~10~~ 0 90	$\overrightarrow{BD}$: ~~10~~ 0 ~~9~~ 6 90	$\overrightarrow{Dt}$: ~~5~~ 0 0	$\overrightarrow{GH}$: ~~10~~ 0 90	$\overrightarrow{JK}$: 100	

Paso 4

Network Auxiliar



Caminos Aumentantes

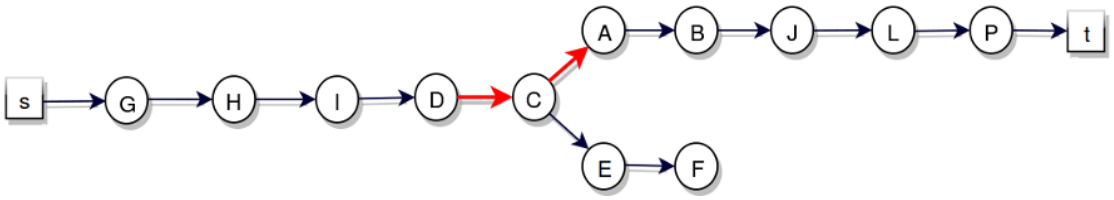
$sGHI\overleftarrow{DB}JKLPt : 10$

Capacidades actualizadas

$\overrightarrow{sA} : \cancel{100}\,\cancel{20}\,\cancel{16}\,\cancel{0}0$	$\overrightarrow{AB} : \cancel{150}\,\cancel{70}\,\cancel{66}60$	$\overrightarrow{BJ} : \cancel{100}90$	$\overrightarrow{DF} : \cancel{4}0$	$\overrightarrow{Ht} : \cancel{10}0$	$\overrightarrow{KL} : \cancel{100}90$
$\overrightarrow{sC} : \cancel{50}0$	$\overrightarrow{AC} : \cancel{10}0$	$\overrightarrow{CD} : \cancel{50}\cancel{0}6$	$\overrightarrow{EF} : \cancel{100}\,\cancel{50}\,\cancel{4}034$	$\overrightarrow{HI} : \cancel{100}90$	$\overrightarrow{LP} : \cancel{100}90$
$\overrightarrow{sE} : \cancel{50}0$	$\overrightarrow{Bt} : \cancel{80}$	$\overrightarrow{CE} : \cancel{20}\,\cancel{10}4$	$\overrightarrow{Ft} : \cancel{70}\,\cancel{20}\,\cancel{16}\,\cancel{0}0$	$\overrightarrow{ID} : \cancel{100}90$	$\overrightarrow{Pt} : \cancel{100}90$
$\overrightarrow{sG} : \cancel{100}\,\cancel{90}80$	$\overrightarrow{BD} : \cancel{100}\,\cancel{96}\,\cancel{90}100$	$\overrightarrow{Dt} : \cancel{50}0$	$\overrightarrow{GH} : \cancel{100}\,\cancel{90}80$	$\overrightarrow{JK} : \cancel{100}90$	

Paso 5

Network Auxiliar



Caminos aumentantes

$sGHI\overleftarrow{DC}\overleftarrow{A}BJKLPt : 10$

Capacidades actualizadas

$\overrightarrow{sA} : \cancel{100}\,\cancel{20}\,\cancel{16}\,\cancel{0}0$	$\overrightarrow{AB} : \cancel{150}\,\cancel{70}\,\cancel{66}\,\cancel{60}50$	$\overrightarrow{BJ} : \cancel{100}\,\cancel{90}80$	$\overrightarrow{DF} : \cancel{4}0$	$\overrightarrow{Ht} : \cancel{10}0$	$\overrightarrow{KL} : \cancel{100}\,\cancel{90}80$
$\overrightarrow{sC} : \cancel{50}0$	$\overrightarrow{AC} : \cancel{10}\cancel{0}10$	$\overrightarrow{CD} : \cancel{50}\cancel{0}\cancel{6}16$	$\overrightarrow{EF} : \cancel{100}\,\cancel{50}\,\cancel{4}034$	$\overrightarrow{HI} : \cancel{100}\,\cancel{90}80$	$\overrightarrow{LP} : \cancel{100}\,\cancel{90}80$
$\overrightarrow{sE} : \cancel{50}0$	$\overrightarrow{Bt} : \cancel{80}$	$\overrightarrow{CE} : \cancel{20}\,\cancel{10}4$	$\overrightarrow{Ft} : \cancel{70}\,\cancel{20}\,\cancel{16}\,\cancel{0}0$	$\overrightarrow{ID} : \cancel{100}\,\cancel{90}80$	$\overrightarrow{Pt} : \cancel{100}\,\cancel{90}80$
$\overrightarrow{sG} : \cancel{100}\,\cancel{90}\,\cancel{80}70$	$\overrightarrow{BD} : \cancel{100}\,\cancel{96}\,\cancel{90}100$	$\overrightarrow{Dt} : \cancel{50}0$	$\overrightarrow{GH} : \cancel{100}\,\cancel{90}\,\cancel{80}70$	$\overrightarrow{JK} : \cancel{100}\,\cancel{90}80$	

Paso 6

Network auxiliar

Como no llegamos a t, esto es, no existen caminos aumentantes entre s y t, podemos afirmar que el flujo obtenido es maximal.

Valor del flujo

$v(f_{MAX}) = f_{MAX}(\overrightarrow{sA}) + f_{MAX}(\overrightarrow{sC}) + f_{MAX}(\overrightarrow{sE}) + f_{MAX}(\overrightarrow{sG}) = 230$

Corte minimal

$S = \{s, G, H, I, D, C, E, F\}$

$CAP(S) = C(S, V - S) = c(\overrightarrow{sA}) + c(\overrightarrow{Ht}) + c(\overrightarrow{Dt}) + c(\overrightarrow{Ft}) = 100 + 10 + 50 + 70 = 230$

Correctitud Dinic

En una corrida del algoritmo, sea A un network auxiliar y sea A' el siguiente network auxiliar. Sea $d(x)$ la distancia de s a x en A y sea $d'(x)$ la distancia de s a x en A'. Entonces $d(t) < d'(t)$. Es decir, la distancia en networks auxiliares sucesivos aumenta.

Prueba

Por la prueba de Edmonds-Karp, sabemos que $d(t) \leq d'(t)$, pero queremos probar $d(t) < d'(t)$, o sea, que es estrictamente menor.

Sea $s = x_0, x_1, x_2, \ldots, x_r = t$ un camino dirigido en A'. Afirmamos que ese camino **no existe** en A, pues para pasar de A a A' debimos bloquear todos los caminos dirigidos de A. Por lo tanto, si ese camino hubiese estado en A, Dinic lo habría bloqueado y por lo tanto no estaría en A'. Veamos cuáles son las posibles razones de que ese camino no esté en A.

1. Falta un vértice, es decir, $\exists\, i$ tal que $x_i \notin V(A)$
2. Falta un lado, es decir, $\exists\, i$ tal que $\overrightarrow{x_i x_{i+1}} \notin E(A)$
   - *a*) $\overrightarrow{x_i x_{i+1}}$ no está porque corresponde a un lado vacío o saturado en N (ni siquiera está en el network residual)
   - *b*) $\overrightarrow{x_i x_{i+1}}$ está en el network residual, pero $d(x_{i+1}) \neq d(x_i) + 1$

Analicemos los tres casos
1) Si $x_i \notin V(A)$, entonces

$$\begin{aligned} d(t) &\leq d(x_i) && \text{(por eso lo excluimos de } A\text{)} \\ &\leq d'(x_i) && \text{(por prueba Edmonds-Karp)} \\ &< d'(t) && \text{(ya que } t \text{ esta despues en este camino)} \end{aligned}$$

2.a) $\overrightarrow{x_i x_{i+1}}$ no está siquiera en el network residual que da origen a A. Pero necesariamente debe estar en el que da origen a A'. Para que pase esto, debemos haber utilizado en A el lado $\overrightarrow{x_{i+1} x_i}$. Por la prueba de Edmonds-Karp, sabemos que entonces

$$d'(t) \geq d(t) + 2 > d(t)$$

2.b) $\overrightarrow{x_i x_{i+1}}$ está en el network residual que da origen a A, debe ser $d(x_{i+1}) \leq d(x_i) + 1$ (mayor no puede ser, ya que al estar conectado x_i con x_{i+1}, la distancia es mayor en a lo sumo una unidad) pero como no está en A, debe ser $d(x_{i+1}) \neq d(x_i) + 1$. Concluimos que $d(x_{i+1}) < d(x_i) + 1$ (*). Entonces:

$$\begin{aligned} d(t) &= d(x_{i+1}) + b(x_{i+1}) && \text{(ver prueba EK)} \\ &\leq d(x_{i+1}) + b'(x_{i+1}) && \text{(ver prueba EK)} \\ &< d(x_i) + 1 + b'(x_{i+1}) && \text{(por (*))} \\ &\leq d'(x_i) + 1 + b'(x_{i+1}) && \text{(ver prueba EK)} \\ &\leq d'(t) \end{aligned}$$

Esta prueba nos dice que los caminos aumentantes provistos por networks auxiliares sucesivos deben ser cada vez más largos. En consecuencia, alguna vez deja de haber caminos aumentantes (ya que el número de vértices en el grafo es finito).

Complejidad de Dinic

La complejidad de Dinic es $O(n^2 m)$, lo cual es un avance con respecto a Edmons-Karp que se nota para grafos densos (donde m es del orden de n^2).

Prueba

Vimos que la distancia en networks auxiliares sucesivos aumenta. Como la distancia entre s y t puede ir desde 1 hasta $n-1$, hay a lo sumo $O(n)$ networks auxiliares.

$$\text{Complejidad de Dinic} = O(n) \cdot \text{ complejidad de hallar flujo bloqueante en } NA$$

Por lo tanto, tenemos que probar que esto último está en $O(nm)$.
Para hallar un flujo bloqueante, debemos

1. Crear NA, que se hace con BFS, que es $O(m)$
2. Hallar flujo bloqueante

En el pseudocódigo presentado para este algoritmo, hay secciones indicadas como AVANZAR, RETROCEDER e INCREMENTAR. Llamaremos:

- AVANZAR $\to A$
- RETROCEDER $\to R$
- INCREMENTAR $\to I$

La búsqueda del flujo bloqueante de Dinic luce como una palabra con estos tres símbolos:

$$AAAIAAAAAIAR \dots AIARAAI \dots$$

Es una palabra cuyo alfabeto es $\{A, I, R\}$. Subdividamos las posibles palabras en

- A^+I
- A^+R
- R (puede haber R sin A antes).

Debemos calcular la complejidad de cada palabra, y la cantidad de cada una.
Recordemos:
A:

```
// AVANZAR
P[i+1] = algún vértice en Γ+(P[i])
i++;
// FIN AVANZAR
```

tiene complejidad $O(1)$
R:

```
// RETROCEDER
Borrar P[i-1]P[i] del NA
i--;
// FIN RETROCEDER
```

Tiene complejidad $O(1)$
I:

```
// INCREMENTAR
ε = mín{c(P[i]P[i+1]) − g(P[i]P[i+1]) ∀ 0 ≤ i ≤ d}
for i= 0 to d -1:
    g+= ε
    if g(P[i]P[i+1]) = c(P[i]P[i+1]):
        borrar lado P[i]P[i+1] del NA
// FIN INCREMENTAR
```

Recorre un camino de longitud $d(t)$ 2 veces, por lo tanto, tiene complejidad $O(d)$.

Por lo tanto, la complejidad de las palabras de tipo A^+R es $O(j)$, donde j es la cantidad de veces que se repite la letra A. Pero cada A incrementa i, y tenemos $0 \leq i \leq d$, por lo tanto $j \leq d$. Concluimos que la complejidad de este tipo de palabras está en $O(d)$.

Por un razonamiento análogo (no puede haber más de d letras A), la complejidad de las palabras de tipo A^+I es $O(d) + O(d) = O(d)$.

Ahora la cantidad de cada tipo de palabra. R tiene la instrucción "borrar lado", por lo que puede haber a lo sumo m palabras de tipo R y de tipo A^+R. También I tiene la instrucción oculta (que se ejecuta al menos una vez) de "borrar lado". Luego, el número de palabras de tipo A^+I debe ser como máximo m. Por lo tanto, hay a lo sumo m palabras de cada tipo.

La complejidad de "hallar camino bloqueante es"$O(m)+O(2 \cdot md) = O(md)$. Pero como $d \leq n$, podemos decir que $O(md) \subseteq O(mn)$ y la complejidad de Dinic es $O(mn^2)$.

Estrategias de resolución de problemas

Hemos visto problemas (2-color, max-flox/min cut) que se sesuelven con algoritmos polinomiales determinísticos y otros para los cuales no se conoce un algoritmo polinomial que los resuelva (3-color, calcular $\chi(G)$). ¿Qué hacemos con estos problemas?

Existen algoritmos que mediante algo de aleatoriedad intentan alcanzar una solución aceptable (no necesariamente la óptima).

Estrategias básicas

1. Exploration: búsqueda más o menos ciega del espacio de solución (ej: seleccionar órden de vértices al azar y correr coloreo greedy a ver si obtenemos un coloreo con menos colores que el que ya tenemos).

2. Explotation: búsqueda dirigida utilizando alguna propiedad del problema que se conoce (ej: greedy con un orden específico no empeora el coloreo que se tiene).

El problema con la exploración guiada puede ser que converja a un mínimo local (o máximo) y no encuentre el mínimo (o máximo).

Ejemplos de estrategias dirigidas:

HILL CLIMBING: modifica aleatoriamente la solución actual y se queda con la nueva sólo sí es menor o igual a la anterior.

SIMULATED ANNEALING: Igual que Hill Climbing, pero una solución peor se puede aceptar de acuerdo a determinada probabilidad.

Algoritmos genéticos

Características

1. trabajan sobre poblaciones (de soluciones)

2. Los individuos (soluciones) interactúan entre sí

3. Trata de imitar la selección natural

Tienen 3 operadores básicos:

Mutación

Cambia aleatoriamente a un individuo, con baja probabilidad y su función es .$^{\text{ex}}$plorar.$^{\text{el}}$ espacio de soluciones. Permite salir de mínimos (o máximos) locales. Se regula mediante probabilidades.

Crossover

Imita el crossover real de la reproducción de las especies. Básicamente, a partir de dos invididuos de la vieja población se obtienen dos individuos de la nueva generación (la población es estable, no aumenta ni decrece de una generación a la siguiente).

Selección

Es la decisión de qué parejas van a ser sometidas al crossover (simulando la reproducción, para avanzar a la siguiente generación). Algunos individuos serán seleccionados más de una vez; otros, por lo tanto, ninguna.

Esquema básico de algoritmos genéticos

```
t = 0
crear aleatoriamente una población inicial P(0)
while (no se den las condiciones para parar):
    selección (P(t))
    crossover
    mutación
    P(t+1) = Población así obtenida
    t++
```

Crossover

Es el mecanismo mediante el cual a partir de dos individuos de la población t se obtienen dos individuos de la población $t+1$.

Crossover simple

Imita el crossover de la vida real. El çromosoma"se corta en un sólo punto. Ej:

$$\begin{array}{rcl} P_1 & = & A\ C\ G\ T \mid F\ R\ G\ A\ D\ E \\ P_2 & = & B\ E\ F\ C \mid K\ E\ R\ A\ B\ C \end{array} \Rightarrow \begin{array}{rcl} H_1 & = & A\ C\ G\ T \mid K\ E\ R\ A\ B\ C \\ H_2 & = & B\ E\ F\ C \mid F\ R\ G\ A\ D\ E \end{array}$$

Two point crossover

El cromosoma se corta en dos puntos:

$$\begin{array}{rcl} P_1 & = & A \mid C\ G\ T\ F \mid R\ G\ A\ D\ E \\ P_2 & = & B \mid E\ F\ C\ K \mid E\ R\ A\ B\ C \end{array} \Rightarrow \begin{array}{rcl} P_1 & = & A \mid E\ F\ C\ K \mid R\ G\ A\ D\ E \\ P_2 & = & B \mid C\ G\ T\ F \mid E\ R\ A\ B\ C \end{array}$$

Order based crossover

Comenzamos trazando una línea como en el crossover simple

$$\begin{array}{rcl} P_1 & = & A\ B\ C\ D \mid E\ F\ G\ H\ I\ J \\ P_2 & = & B\ E\ F\ C \mid J\ I\ D\ G\ A\ H \end{array}$$

Luego para obtener el hijos H_1 se toma la primera parte del progenitor P_1 y la segunda se reordena según el orden que dicta P_2 y viceversa:

$$\begin{array}{rcl} H_1 & = & A\ B\ C\ D \mid E\ F\ J\ I\ G\ H \\ H_2 & = & B\ E\ F\ C \mid A\ D\ G\ H\ I\ J \end{array}$$

Cyclic Crossover

Dado dos progenitores de la forma

$$\begin{array}{lllllllllll} P_1 & = & B & I & G & A & E & C & J & F & D & H \\ P_2 & = & D & J & B & F & E & C & I & G & A & H \end{array}$$

obtenemos el hijo H_1 por pasos a partir de P_2 del siguiente modo:

1. $H_1 = P_2 = D \quad J \quad B \quad F \quad E \quad C \quad I \quad G \quad A \quad H$
2. Elegimos un elemento al azar, en este caso el primero, y lo cambiamos por el elemento en esa posición de P_1,
3. $\begin{array}{lllllllllll} H_1 & = & \cancel{D} & J & B & F & E & C & I & G & A & H \\ & & B & & & & & & & & & \end{array}$
4. Para evitar que haya dos B repetidas, cambiamos la otra B por el elemento en esa posición de P_1,
5. $\begin{array}{lllllllllll} H_1 & = & \cancel{D} & J & \cancel{B} & F & E & C & I & G & A & H \\ & & B & & G & & & & & & & \end{array}$
6. etc.

Si hay ciclos de 1 o de n elementos, no habrá cambios.

PMX

Partial mixing crossover, es similar a two-point crossover, con distinta técnica.
Primero se selecciona un bloque al azar.

$$\begin{array}{lllll|llll|l} P_1 & = & A & B & C & D & E & F & G \\ P_2 & = & E & D & F & G & B & A & C \end{array}$$

Se intercambian:

$$\begin{array}{lllll|llll|l} H_1 & = & A & B & F & G & B & A & G \\ H_2 & = & E & D & C & D & E & F & C \end{array}$$

Se crea una tabla de equivalencias a partir de los elementos del bloque central:

$$\begin{aligned} A &\leftrightarrow F \leftrightarrow C \\ D &\leftrightarrow G \\ B &\leftrightarrow E \end{aligned}$$

Se utiliza la tabla de equivalencias para obtener los reemplazos adecuados para los elementos fuera del bloque central:

$$\begin{array}{lllll|llll|l} & & C & E & & & & & D \\ H_1 & = & \cancel{A} & \cancel{B} & F & G & B & A & \cancel{G} \\ H_2 & = & \cancel{E} & \cancel{D} & C & D & E & F & \cancel{C} \\ & & B & G & & & & & A \end{array}$$

Selección

Se requiere una fitness function (función de adaptabilidad). A mayor valor de fitness, mayor será la probabilidad de un individuo de ser seleccionado para su reproducción. A partir del valor de esta función, se calcula la .^{es}peranza de reproducción", que transforma el valor de adaptabilidad en una probablidad de reproducción, o bien cuántes veces espera el individuo reproducirse.

$$E_i = \frac{F(i)}{\overline{F}},$$

donde $\overline{F} = \frac{\sum_i F(i)}{n}$ es el promedio de los valores que toma la función de fitness en el conjunto de todos los individuos.

Usando los valores de E_i seleccionamos los individuos para hacer crossover.

Por ejemplo:

$F(1)$	$17,118$
$F(2)$	$9,51$
$F(3)$	$36,138$
$F(4)$	$22,824$
$F(5)$	$91,926$
$F(6)$	$13,314$

Se tiene que $\sum_i F(i) = 190,2$ y $\overline{F} = 31,7$, por lo tanto:

$E(1)$	$0,54$
$E(2)$	$0,3$
$E(3)$	$1,14$
$E(4)$	$0,72$
$E(5)$	$2,88$
$E(6)$	$0,42$

Se puede decir que el individuo 1 tiene una esperanza de reproducirse entre 0 y 1 vez, mientras que el individuo 5 puede esperar reproducirse entre 2 y 3 veces.

A continuación se detallan algunos métodos para elegir los individuos (y las parejas respectivas) que se utilizarán para la reproducción, a partir de las esperanzas de reproducción calculadas.

Ruleta

Elegir 2 individuos al azar de entre los 6 (tres veces, en total) usando E_i como peso, os ea, el individo 5 tiene más chances de salir seleccionado. Un método común es como sigue:

1. Seleccionar n números al azar, entre 0 y n (por ejemplo, $4,2$; $0,9$; $1,32$; $2,76$; $3,78$; $4,98$)
2. Para cada número, elegir el primer i tal que $\sum_{j \leq i} E_j \geq$ número (la suma de las esperanzas acumuladas hasta ese individuo).

i	$E(i)$	$\sum_{j \leq i} E_j$
1	$0,56$	$0,54$
2	$0,3$	$0,84$
3	$1,14$	$1,98$
4	$0,72$	$2,7$
5	$2,88$	$5,58$
6	$0,42$	6

De este modo,

- $4,2$ nos dice que elijamos al individuo 5,

- $0,9$ nos dice que elijamos al individuo 3, (esta es la primer pareja)
- $1,32$ nos dice que elijamos al individuo 3,
- $2,76$ nos dice que elijamos al individuo 5 (esta es la segunda pareja)
- $3,78$ nos dice que elijamos al individuo 5
- $4,98$ nos dice que elijamos al individuo 5 (esta es la tercer pareja)

SUS (Stochastic Universal Sampling)

Elegir un número al azar entre 0 y 1. El resto es como el método **Ruleta**, pero los números que se usan para delimitar los intervalos son

$$x;\ x+1;\ x+2;\ \ldots;\ x+n-1$$

Ejemplo: número al azar $x = 0,3$,

$0,3$	$1,3$	$2,3$	$3,3$	$4,3$	$5,3$
I_1	I_3	I_4	I_5	I_5	I_5

De acuerdo a este método, tendríamos las parejas $(I_1; I_3), (I_4; I_5), (I_5; I_5)$

Métodos con remainder (resto)

Para cada i, asignamos al individuo I_i $\lfloor E_i \rfloor$ "tokens"de reproducción:

$$\begin{array}{lcccl} I_1 & \rightarrow & \lfloor E_i \rfloor = \lfloor 0,54 \rfloor & \rightarrow & 0 \\ I_2 & \rightarrow & \rightarrow & \rightarrow & 0 \\ I_3 & \rightarrow & \rightarrow & \rightarrow & 1 \\ I_4 & \rightarrow & \rightarrow & \rightarrow & 0 \\ I_5 & \rightarrow & \rightarrow & \rightarrow & 2 \\ I_6 & \rightarrow & \rightarrow & \rightarrow & 0 \end{array}$$

Según esto, el individuo I_3 tiene garantizada una "plaza"de reproducción, mientras que el individuo I_5, a su vez tiene dos. Quedan entonces tres plazas de reproducción por asignar, entonces, aplicamos **Ruleta** o **SUS** a sus restos:

		$\sum_i$
R_{I_1}	$0,54$	$0,54$
R_{I_2}	$0,3$	$0,84$
R_{I_3}	$0,14$	$0,98$
R_{I_4}	$0,72$	$1,71$
R_{I_5}	$0,88$	$2,58$
R_{I_6}	$0,42$	3

Seleccionamos 3 números entre 0 y 3, etc.

Métodos basados en el ranking

Sólo dependen de la posición del individuo en el ranking y no del valor concreto de la función fitness. En nuestro ejemplo, el ranking sería:

$$I_5, I_3, I_4, I_1, I_6, I_2$$

Una manera de determinar la esperanza de cada individuo a partir de su posición es

$$LP(P) = (2 - Sp) + 2(Sp - 1) \cdot \tfrac{P-1}{n-1},\ \text{con } 1 \leq Sp \leq 2,$$

Donde $P = n - \text{posicion} + 1$, Sp es un parámetro (Selection pressure).
Ejemplo, con $Sp = 2$

$$
\begin{aligned}
I_2 &\to LP(1) = 0 \\
I_6 &\to LP(2) = 0,4 \\
I_1 &\to LP(3) = 0,8 \\
I_4 &\to LP(4) = 1,2 \\
I_4 &\to LP(5) = 1,6 \\
I_5 &\to LP(6) = 2
\end{aligned}
$$

(Luego aplicar **Ruleta** o **SUS** con esos valores)
Forma 2:
Este método previene el problema de la early convergence. Cambia la forma de calcular E_i tomando E_i^* (sigma scaling).

$$E_i^* = 1 + \frac{F_i - \overline{F}}{2\sigma}$$

donde σ es la desviación estándar de los F_i.
En nuestro ejempplo, tenemos $\sigma = 30,637$ y $2\sigma = 61,27$, con lo que

$$
\begin{aligned}
E_1^* &= 1 + \frac{17,118 - 31,7}{61,27} = 0,762 \\
E_2^* &= 0,638 \\
E_3^* &= 1,0724 \\
E_4^* &= 0,855 \\
E_5^* &= 1,9726 \\
E_6^* &= 0,6939
\end{aligned}
$$

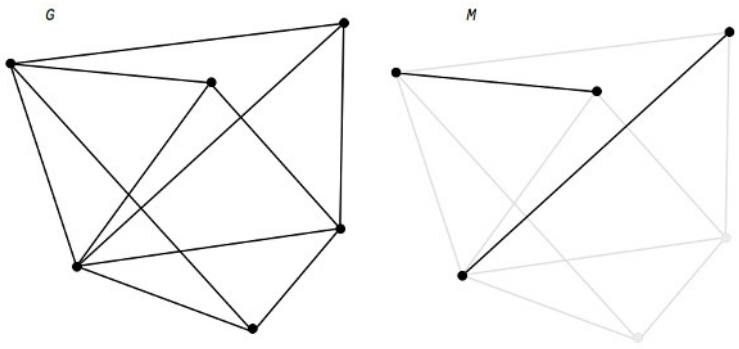
Esto tiene el efecto de achicar las diferencias entre las esperanzas. Luego, aplicar uno de los métodos anteriores con estos valores.

Matching

Dado un grafo G, un matching en G es un subgrafo M de G tal que

$$d_M(x) = 1 \quad \forall\, x \in V(M)$$

Es decir, el grado (relativo al subgrafo M) de x es 1 para todos los vértices de M.
Ejemplo:



Matching maximal

Un matching M es maximal en G si

$$|E(M)| \geq |E(M')| \quad \forall\ M' \text{ matching en } G$$

Es decir, la cardinalidad del conjunto de lados del matching M es la máxima posible.

De ahora en adelante, restringiremos nuestra atención al problema de encontrar matching maximal en grafos bipartitos.

Matching maximal como un problema de Max-Flow / Min-Cut

Daremos un algoritmo para encontrar un matching maximal en un grafo **bipartito**.

Notación

Cuando escribimos $G = (X \cup Y, E)$ queremos significar que X e Y son las dos partes de $V(G)$ tales que $X \cap Y = \emptyset$ y $X \cup Y = V(G)$.

Algoritmo

Dado $G = (X \cup Y, E)$ bipartito, entonces construimos $N = N(G)$ un network con

$$V(N) = V(G) \cup \{s, t\}$$
$$E(N) = \{\overrightarrow{xy} : x \in X, y \in Y, \overrightarrow{xy} \in E(G)\} \cup \{\overrightarrow{sx} : x \in X\} \cup \{\overrightarrow{yt} : y \in Y\}$$

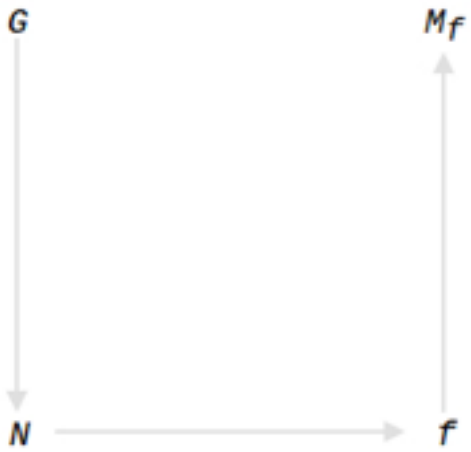
Con capacidades en todos los lados igual a 1.

Con G construido de esta manera, buscamos un flujo maximal f.

A partir de f, construimos el matching M_f en G de la siguiente manera:

$$V(M_f) = \{x \in X : \exists\ y \in Y : f(\overrightarrow{xy}) = 1\} \cup \{y \in Y : \exists\ x \in X : f(\overrightarrow{xy}) = 1\}$$
$$E(M_f) = \{xy : x \in X, y \in Y, f(\overrightarrow{xy}) = 1\}$$

Entonces, como se ve en el diagrama a continuación:



recibimos un grafo G, lo utilizamos para obtener un network N, calculamos un flujo maximal f en N, utilizamos f y N para construir un matching M_f en G. Debemos probar ahora que M_f es matching y que es maximal.

M_f es matching

Si no lo fuera, existiría $z \in V(M_f)$ tal que $d_{M_f}(z) = 0$ o $d_{M_f}(z) \geq 2$. Lo primero es imposible, porque por construcción, sólo se agregan a M_f aquellos vértices que están conectados con algo.

Veamos lo segundo.

Si $z \in X$: como $d_{M_f}(z) \geq 2$, entonces $\exists\ y_1, y_2 \in Y$ distintos tales que $zy_1, zy_2 \in E(M_f)$. Pero entonces

$$f(\overrightarrow{zy_1}) = f(\overrightarrow{zy_2}) = 1 \Rightarrow OUT_f(z) \geq 2,$$

mientras que

$$\Gamma^-(z) = \{s\} \Rightarrow IN_f(z) \leq 1,$$

lo cual es imposible.

Si $z \in Y$: mediante un razonamiento análogo pero con t en lugar de s, veríamos que no se cumple la conservación y f no sería flujo, lo cual es absurdo.

Biyección entre flujos f y matchings M_f

Observermos que

$$\begin{aligned} v(f) &= \sum_{x \in X} f(\overrightarrow{sx}) \\ &= |\{x \in X : f(\overrightarrow{sx}) = 1\}| \\ &= |V(M_f) \cap X| \quad \text{pues } M_f \text{ es matching y } G \text{ es bipartito} \\ &= |E(M_f)| \end{aligned}$$

es decir que el valor del flujo f (que es maximal) coincide con el número de lados en el matching.

Viceversa, dado un matching en G, el flujo f_M dado por

$$f_M(\overrightarrow{xy}) = \begin{cases} 1 & xy \in E(M) \\ 0 & xy \notin E(M) \end{cases}$$

$$f_M(\overrightarrow{sx}) = \begin{cases} 1 & x \in V(M) \\ 0 & x \notin V(M) \end{cases}$$

$$f_M(\overrightarrow{yt}) = \begin{cases} 1 & y \in V(M) \\ 0 & y \notin V(M) \end{cases}$$

efectivamente es flujo, pues

$$\begin{aligned} x \notin V(M) &\Rightarrow f_M(\overrightarrow{sx}) = 0, f_M(\overrightarrow{xy}) = 0 \quad \forall\ y \\ &\Rightarrow IN_f(x) = 0, OUT_f(x) = 0 \end{aligned}$$

$$\begin{aligned} x \in V(M) &\Rightarrow f_M(\overrightarrow{sx}) = 1 \\ &\Rightarrow IN_f(x) = 1 \\ &\Rightarrow \exists\ y \text{ unico tal que } f(\overrightarrow{xy}) = 1 \\ &\Rightarrow OUT_f(x) = 1 = IN_f(x) \end{aligned}$$

y de forma análoga, si $y \in Y$ se puede demostrar que $IN_f(y) = OUT_f(y)$.

Además está claro por la definición, $M_{f_M} = M$. Entonces $v(f_M) = |E(M_{f_M})| = |E(M_f)|$.

Todo esto sirvió para probar que tenemos una biyección entre los flujos maximales y los matchings construidos a partir de ellos.

M_f es maximal

Sea f_{MAX} flujo maximal entero y $M_{f_{MAX}}$ matching construido a partir de este. Sea M otro matching cualquiera. Entonces

$$|E(M)| = v(f_M) \leq v(f_{MAX}) = |E(M_{f_{MAX}})|$$

es decir, $M_{f_{MAX}}$ es maximal.

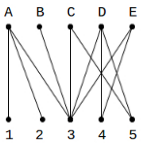
Algoritmo para encontrar matching maximal

Overview

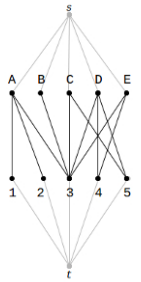
1. Dinic con un network auxiliar (lo cual va a dar caminos de la forma $sx_1y_1t : 1 \quad sx_2y_2t : 1 \quad \dots \quad sx_ny_nt : 1$, que se corresponde a agregar estos lados al matching).
2. Tras encontrar el primer flujo bloqueante con Dinic, continuar con Edmonds-Karp.

Detalles

Si tenemos que encontrar un matching maximal en el grafo G dado por



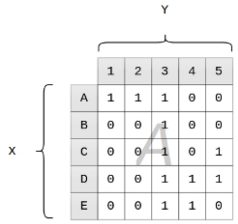
a partir del cual construimos el network N dado por



entonces podemos pensar en la matriz de adyacencia de G, que es de la forma

		A	B	C	D	E	1	2	3	4	5
X	A	0	0	0	0	0	1	1	1	0	0
	B	0	0	0	0	0	0	0	1	0	0
	C	0	0	0	0	0	0	0	1	0	1
	D	0	0	0	0	0	0	0	1	1	1
	E	0	0	0	0	0	0	0	1	1	0
Y	1	1	0	0	0	0	0	0	0	0	0
	2	1	0	0	0	0	0	0	0	0	0
	3	1	1	1	1	1	0	0	0	0	0
	4	0	0	0	1	1	0	0	0	0	0
	5	0	0	1	1	0	0	0	0	0	0

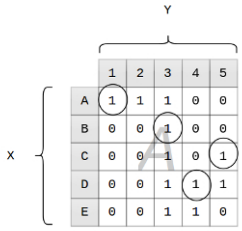
Observamos que por ser $G = (X \cup Y, E)$ un grafo bipartito, no hay unos en dos regiones importantes de la matriz. Estamos interesados entonces en la matriz A que nos dice qué vértcies de X se pueden unir con qué vértices de Y:



Es aquí donde corremos Dinic hasta encontrar un flujo bloqueante. Así, obtenemos los caminos:

$$sA1t : 1$$
$$sB3t : 1$$
$$sC5t : 1$$
$$sD4t : 1$$

Lo cual equivale a agregar los lados $A1, B3, C5$ y $D4$ al matching:



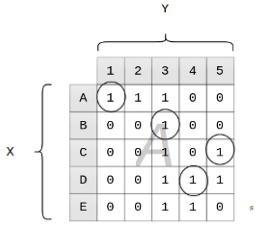
Luego aplicamos Edmonds-Karp, donde las colas van a ser de la forma:

$$s \;\Big|\; \begin{matrix} x_{j_1} \\ s \end{matrix} \quad \begin{matrix} x_{j_2} \\ s \end{matrix} \quad \dots \quad \begin{matrix} x_{j_q} \\ s \end{matrix} \;\Big|\; \begin{matrix} y_{k_{11}} \\ x_{j_1} \end{matrix} \quad \begin{matrix} y_{k_{12}} \\ x_{j_1} \end{matrix} \quad \dots \quad \begin{matrix} y_{k_{1r}} \\ x_{j_1} \end{matrix} \;\Big|\; \dots \;\Big|\; \begin{matrix} y_{k_{q1}} \\ x_{j_q} \end{matrix} \quad \begin{matrix} y_{k_{q2}} \\ x_{j_q} \end{matrix} \quad \dots \quad \begin{matrix} y_{k_{qr}} \\ x_{j_q} \end{matrix} \;\Big|\; \dots \quad \text{(lados backwards)}$$

Es decir, donde

- comenzamos con un grupo de vértices $x \in X$ agregados por s
- continuamos con vértices $y \in Y$ agregados por estos últimos
- los vértices de Y comienzan a agregar vértices de X, nuevamente, como lados backwards,
- etc.

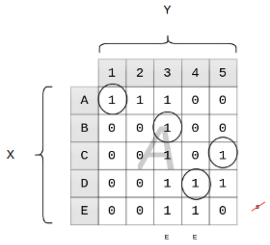
Esta cola la corremos manuamente colocando etiquetas al costado de la matriz A:
s agrega a las filas no matcheadas, luego, etiquetamos dichas filas con s:



Lo cual sería como comenzar la cola de Edmonds-Karp con:

$$\begin{array}{cc} \not{s} & E \\ & s \end{array}$$

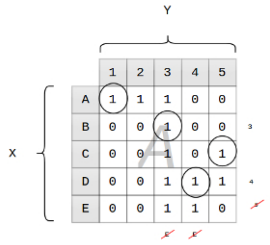
Ahora es el turno de que E incluya a los vértices de Y que puede alcanzar y que no están en la cola, es decir, etiquetamos con E las columnas correspondientes a vértices adyacentes a E (allí donde hay unos) siempre y cuando no estén etiquetadas:



que es como haber continuado la cola de Edmonds-Karp del siguiente modo:

$$\begin{array}{cccc} \not{s} & \not{E} & 3 & 4 \\ & s & E & E \end{array}$$

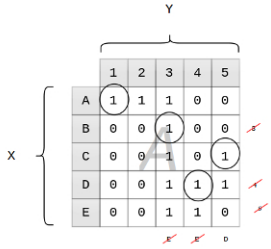
Como los elementos 3 y 4 (o, equivalentemente, las columnas 3 y 4) fueron previamente matcheadas, entonces pueden devolver flujo (lados backwards) a aquellos vértices de X que se los enviaron que no están en la cola aún (o, equivalentemente, a las filas cuyos unos están marcados siempre que esas filas no hayan sido etiquetadas previamente):



que es como haber continuado la cola de Edmonds-Karp así:

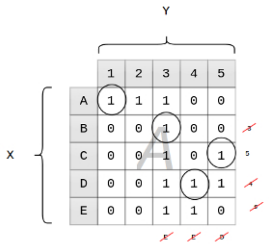
$$\begin{array}{ccccccc} \not{s} & \not{E} & \not{3} & \not{4} & B & D \\ & s & E & E & 3^- & 4^- \end{array}$$

Como B no puede agregar elementos nuevos, simplemente lo tachamos. En cambio, la fila D puede agregar a 5:



$$\begin{array}{ccccccc} \not{s} & \not{E} & \not{3} & \not{4} & \not{B} & \not{D} & 5 \\ & s & E & E & 3^- & 4^- & D \end{array}$$

Luego 5 puede agregar C (lado backwards):



$$\begin{array}{cccccccc} \not{s} & \not{E} & \not{3} & \not{4} & \not{B} & \not{D} & \not{5} & C \\ & s & E & E & 3^- & 4^- & D & 5^- \end{array}$$

Como C ya no puede agregar a nadie y aún no pudimos llegar a t (no etiquetamos ninguna columna sin matching) el matching no se puede exteneder. Luego, tenemos un matching maximal.

Matching Perfecto y Matching Completo

Si $G = (X \cup Y, E)$ es bipartito diremos que un matching M en G es **perfecto** si $V(M) = V(G)$. Diremos que es **completo** de X a Y si $X \subseteq V(M)$ (o completo de Y a X si $Y \subseteq V(M)$).

Condiciones obvias

Para que exista un matching perfecto, $|X|$ debe ser igual a $|Y|$ (de modo que pueda existir una biyección entre los elementos). Aún si $|X| = |Y|$ puede no haber matching perfecto (recordar ejemplo anterior).

$\Gamma(S)$ en un grafo

Dado un grafo G y un subconjunto $S \subseteq V(G)$,

$$\begin{aligned} \Gamma(S) &= \cup_{z \in S} \Gamma(z) \\ &= \{w \in V(G) : \exists\ z \in S : zw \in E(G)\} \end{aligned}$$

Esta es la definición general, sin embargo en adelante nos vamos a restringir a casos con grafos bipartitos y con $S \subseteq X$ ó $S \subseteq Y$.

Teorema de Hall

Sea $G = (X \cup Y, E)$ un grafo bipartito, entonces existe un matching completo de X a Y si y sólo si $|S| \leq |\Gamma(S)|$ para todo $S \subseteq X$.

Prueba

$\Rightarrow$ Si M es matching completo de X a Y, entonces $\forall\ x \in X\ \exists!\ y \in Y : xy \in E(M)$. Sea

$$f(x) = \text{ese único } y$$

Notemos que entonces f es una función inyectiva de X a Y, ya que $xy \in M \wedge zy \in M \Rightarrow x = z$ (por ser M un matching). Además, por definición $f(x) \in \Gamma(x)$. Esto implica que $f(S) \subseteq \Gamma(S)\ \forall\ S \subseteq X$ y por lo tanto $|f(S)| = |S| \leq |\Gamma(S)|$.

$\Leftarrow$ Supongamos que no es cierto. Entonces existe G grafo bipartito con $|S| \leq |\Gamma(S)|\ \forall\ S \subseteq X$ pero no tiene matching completo de X a Y. Esto quiere decir que cuando corremos el algoritmo, quedan filas sin matchear.

Sea S_0 el conjunto de tales filas y $T_1 = \Gamma(S_0)$. Todas las columnas de T_1 están matcheadas (pues si hubiera una libre, se matchearía con algo de S_0).

Sea $S_1 \subseteq X$ las filas matcheadas con las columnas de T_1 y sea $T_2 = \Gamma(S_1) - T_1$, etc.

En general:

$$S_i = \text{Filas matcheadas por } T_i$$

$$T_{i+1} = \Gamma(S_i) - (T_0 \cup T_1 \cup \cdots \cup T_i)$$

Esto puede continuar hasta que T_n sea vacío, ya que S_n no va a ser vacío nunca.

Como el algoritmo para sin hallar matching perfecto, y $\forall\ i$, $T_i \neq \emptyset$ produce un S_i, la única forma de parar es un k tal que $T_{k+1} = \emptyset$.

Sea $S = S_0 \cup S_1 \cup \cdots \cup S_k$ (las filas que tienen etiqueta).

Observaciones:

1. Como S_i son las filas matcheadas con T_i, entonces $|S_i| = |T_i|$ (por ser matching parcial).
2. Por construcción, los T_i son disjuntos
3. $T_1 \cup T_2 \cup \cdots \cup T_{i+1} = \Gamma(S_0 \cup S_1 \cup \cdots \cup S_i)$

Prueba de 3: para $i = 1$ vale $(T_1 = \Gamma(S_0))$, y si vale para i, entonces vale para $i+1$, ya que:

$$\begin{aligned}
T_1 \cup \cdots \cup T_{i+2} &= T_1 \cup T_2 \cup \cdots \cup T_{i+1} \cup T_{i+2} \\
&= T_1 \cup T_2 \cup \cdots \cup T_{i+1} \cup (\Gamma(S_{i+1}) - (T_1 \cup \cdots \cup T_{i+1})) \\
&= T_1 \cup T_2 \cup \cdots \cup T_{i+1} \cup \Gamma(S_{i+1}) \\
&= \Gamma(S_0 \cup S_1 \cup \cdots \cup S_i) \cup \Gamma(S_{i+1}) \\
&= \Gamma(S_0 \cup S_1 \cup \cdots \cup S_i \cup S_{i+1})
\end{aligned}$$

Entonces

$$\begin{aligned}
|\Gamma(S)| &= |\Gamma(S_0 \cup S_1 \cup \cdots \cup S_k)| \\
&= |T_1 \cup T_2 \cup \cdots \cup T_k \cup T_{k+1}| \text{ (por 3)} \\
&= |T_1 \cup T_2 \cup \cdots \cup T_k| \text{ (por } T_{k+1} = \emptyset) \\
&= |T_1| + |T_2| + \cdots + |T_k| \text{ (por 2)} \\
&= |S_1| + |S_2| + \cdots + |S_k| \text{ (por 1)} \\
&= |S| - |S_0| \text{ (por 2)} \\
&< |S| \text{ (por ser } S_0 \neq \emptyset)
\end{aligned}$$

Esto contradice la hipótesis, es absurdo.

Aplicación

A la hora de buscar un matching perfecto, si no lo encontramos, el teorema de Hall provee una manera de certificar que no existe:

$$S = \text{Todas las filas marcadas}$$

$$\Gamma(S) = \text{todas las columnas marcadas}$$

Teorema del matrimonio

Todo grafo bipartito regular tiene un matching perfecto.

Estructura de la prueba

1. Probar que en un grafo bipartito regular todo matching completo es perfecto.
2. Probar que en un grafo bipartito regular se cumple la condición de Hall.

Prueba

Sea $G = (X \cup Y, E)$ bipartito regular. Para todo $W \subseteq V(G)$ definamos

$$E_W = \{xy \in E(G) : x \in W \vee y \in W\}$$

Supongamos ahora $W \subseteq X$ o $W \subseteq Y$. Como $\chi(G) = 2$, entonces hay lados y como G es regular, existe $\delta = \Delta > 0$ tal que $d(z) = \Delta \forall\ z \in V(G)$. Como para todo $w \in W$ existe al menos un lado wv y si $W \subseteq X$ o $W \subseteq Y$, a vértices distintos le corresponden lados distintos, tenemos que a cada w le corresponden Δ lados distintos:

$$|E_W| = \Delta \cdot |W|, \text{ si } W \subseteq X \text{ o } W \subseteq Y$$

En particular, $|E_X| = \Delta \cdot |X|$. Pero $E_X = E = E_Y$, pues G es bipartito, por lo tanto, $\Delta \cdot |X| = \Delta \cdot |Y|$, por lo tanto $|X| = |Y|$. Esto significa que todo matching completo es perfecto. Basta entonces con probar la condición de Hall.

Sea $S \subseteq X$, sea $xy \in E_S$ con $x \in X$, $y \in Y$, o sea, $y \in \Gamma(x)$ y por lo tanto, $y \in \Gamma(S)$ y $xy \in E_{\Gamma(S)}$. Es decir, hemos probado que

$$\begin{aligned} E_S \subseteq E_{\Gamma(S)} &\Rightarrow |E_S| \leq |E_{\Gamma(S)}| \\ &\Rightarrow \Delta \cdot |S| \leq \Delta \cdot |\Gamma(S)| \\ &\Rightarrow |S| \leq |\Gamma(S)| \end{aligned}$$

Índice cromático

Un coloreo propio de los lados de un grafo es una función $C : E \to S$ (donde S es cualquier conjunto) tal que

$$xy \cap zw \neq \emptyset \Rightarrow C(xy) \neq C(zw)$$

$\chi'(G)$, denominado índice cromático de G (o también edge chromatic number) es la menor cardinalidad de un coloreo propio de los lados.

Existe una cota obvia para $\chi'(G)$: como todos los lados que parten de un mismo vértice precisan colores diferentes, $\chi'(G) \geq \Delta(G)$. Por otro lado, el teorema de Vizing (que no probaremos) nos dice que $\chi'(G) \leq \Delta(G) + 1$. Es decir que sólo hay dos posibilidades para $\chi'(G)$. En general, probar que es $\chi'(G) = \Delta(G)$ o $\chi'(G) = \Delta(G) + 1$ es difícil.

Índice cromático de grafos bipartitos

Mediante los conceptos de matching se puede probar que si G es bipartito, entonces $\chi'(G) = \Delta(G)$. Se prueba utilizando el teorema del matrimonio.

Sea H bipartito regular, con $\Delta(H) = \Delta(G)$ tal que $G \subseteq H$. Como H es bipartito regular, tiene un matching perfecto M.

Sea $H_2 = (V(H), E(H) - E(M))$. Como M es matching perfecto, H_2 sigue siendo regular y $|H_1| = \Delta - 1$. Como H_2 es bipartitto regular, tiene un matching perfecto M_2. Si lo removemos, nos vuelve a quedar un grafo bipartito regular, con $\Delta = \Delta - 2$, etc.: podemos continuar removiendo lados Δ veces.

Tenemos entonces matchings perfectos $M, M_2, \ldots, M_\Delta$ con

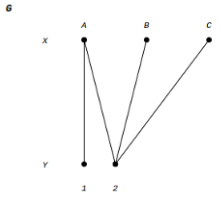
$$E(H) = E(M_1) \cup E(M_2) \cup \cdots \cup E(M_\Delta)$$

Entonces damos el coloreo

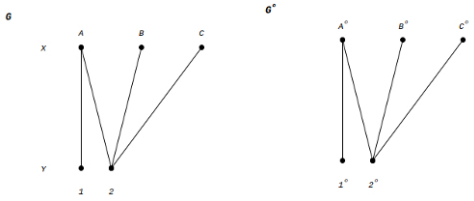
$$C(xy) = i \text{ si } xy \in E(M_i)$$

Este coloreo es propio (pues M_i es matching) y utiliza sólo Δ colores. Falta probar que H existe. Es decir, que para todo G bipartito existe un H bipartito regular tal que $\Delta(H) = \Delta(G)$ y $G \subseteq H$.

Sea $G = (V = X \cup Y, E)$ bipartito

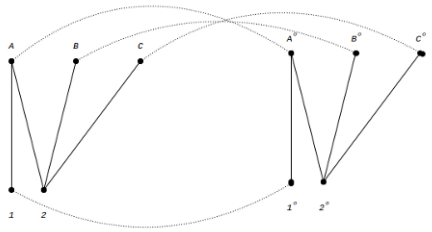


y sea $G^o = (V^o, E^o)$ una copia de G



Sea $E^* = \{vv^o : d_G(v) < \Delta(G)\}$ y tomemos $\tilde{G} = (\tilde{V}, \tilde{E})$, donde

$$\tilde{V} = V \cup V^o$$
$$\tilde{E} = E \cup E^o \cup E^*$$



Propiedades de $\tilde{G}$:

1. $\tilde{G}$ es bipartito, sus partes son $X \cup Y^o$ y $X^o \cup Y$ (no hay lados de X a X, ni lados de Y^o a Y^o, ni lados de X a Y^o, etc.)
2. Sea $v \in V$ tal que $d_G(V) = \Delta$, entonces v no es parte de ningún lado de E^* (tampoco v^o), por lo tanto $d_{\tilde{G}}(v) = d_{\tilde{G}}(v^o) = \Delta$.
3. Sea $v \in V$ tal que $d_G(v) < \Delta$, entonces v forma parte de un lado de E^* (también v^o), por lo tanto $d_{\tilde{G}}(v) = d_{\tilde{G}}(v^o) = d_G(v) + 1$.

Entonces podemos concluir que

$$\Delta(\tilde{G}) = \Delta(G)$$
$$\delta(\tilde{G}) = \delta(G) + 1$$

Si iteramos el procedimiento,

$$G \to \tilde{G} \to \tilde{\tilde{G}} \to \tilde{\tilde{\tilde{G}}} \to \dots$$

llegaremos a un grafo H bipartito tal que $\delta(H) = \Delta(H) = \Delta$.

Matching con pesos

Situación

Tenemos los conjuntos X e Y disjuntos (supondremos $|X| = |Y|$ para facilitar las cosas). Por ejemplo

- X: conjunto de trabajadores
- Y: conjunto de trabajos

Contamos además con una matriz de pesos con filas indexadas por X y columnas indexadas por Y, donde $A_{x,y}$ es el costo de asignar el trabajo y al trabajador x. De entre todos los matchings perfectos posibles, queremos hallar el óptimo (relativo a la matriz de pesos A) donde óptimo puede significar varias cosas:

- Los pesos son costos y queremos minimizar el mayor costo.
- Los pesos son costos y queremos minimizar el costo total.
- Los pesos son ganancias y queremos maximizar la mayor ganancia.
- Los pesos son ganancias y queremos maximizar la ganancia total.

Minimizar el máximo costo (Bottleneck problem)

Dados conjuntos X, Y disjuntos y la matriz $A_{n\times n}$ de costos, queremos hallar un matching perfecto que minimice el mayor costo.

Fuerza bruta

Buscar los $n!$ matchings perfectos y encontrar el que minimice el mayor costo. No es viable.

Estrategia dummy

Tomar la diagonal principal de la matriz A. En ese matching el máximo costo es $m = \text{máx}\{A_{i,i} : 1 \leq i \leq n\}$. Nos preguntamos ¿Existirá un matching perfecto cuyo máximo costo sea menor que m? Entonces armamos una matriz A' con

$$A'_{i,j} = \begin{cases} 1 & A_{i,j} < m \\ 0 & A_{i,j} \geq m \end{cases}$$

y buscamos un matching perfecto excluyendo los ceros. Si lo encontramos, tendremos un nuevo matching con máximo costo $m' < m$ e iteramos el proceso hasta que ya no encontremos matching perfecto.

Posible mejora

Utilizar binary search con los posibles m. Por ejemplo, si nuestra matriz de costos es

	I	II	III	IV	V	VI	VII	$VIII$
A	1	7	17	19	10	24	8	33
B	34	32	30	35	29	34	4	9
C	12	2	20	29	34	23	19	32
D	33	4	25	2^{1000}	12	30	31	18
E	31	33	7	15	23	17	34	33
F	35	32	12	29	29	33	32	1
G	20	15	32	33	4	32	33	32
H	30	30	33	2	32	12	1	2

Los costos están entre 1 y 2^{1000}. El valor del medio es 19. Entonces generamos una matriz donde hay un 1 si el valor es ≤ 19 y un 0 de lo contrario:

	I	II	III	IV	V	VI	VII	$VIII$
A	1	1	1	1	1	0	1	0
B	0	0	0	0	0	0	1	1
C	1	1	0	0	0	0	1	0
D	0	1	0	0	1	0	0	1
E	0	0	1	1	0	1	0	0
F	0	0	1	0	0	0	0	1
G	0	1	0	0	1	0	0	0
H	0	0	0	1	0	1	1	1

Buscamos matching perfecto y encontramos uno con máximo costo 17. Los posibles máximos costos menores que 17 son

$$1, 2, 4, 7, 8, 9, 10, 12, 15, 17$$

Entonces, para continuar haciendo búsqueda binaria repetimos el proceso pero esta vez con 9 como umbral (el valor del medio). El proceso se repite hasta determinar que uno tiene un matching perfecto con costo máximo m y que no existe matching perfecto con un costo máximo menor.

Complejidad del algoritmo

El algoritmo tiene complejidad $O(n^3 \log(n))$. Veamos la justificación.

Tenemos que decidir si hay matching perfecto o no un cierto número de veces. ¿Cuál es la complejidad de decidir si hay matching perfecto?

1. Primero hay que hallar un matching inicial para lo cual hay que revisar n filas, y cada revisión está en $O(n)$, luego esta parte está en $O(n^2)$.

2. Luego hay que extender el matching. Para extenderlo en un lado, hay que revisar todas las filas y todas las columnas ($O(n^2)$). Para extenderlo hasta un matching perfecto (o certificar que no hay) hay que repetir lo anterior $O(n)$ veces. Por lo tanto, esta parte tiene su complejidad en $O(n^3)$.

Entonces, decidir si existe matching perfecto tiene complejidad en $O(n^2) + O(n^3) = O(n^2)$.

¿Cuántas veces se realiza esta operación? Si implementamos la estrategia de hacer búsqueda binaria en los $O(n)$ posibles valores distintos, tendremos que hacerlo $O(\log(n^2)) = O(2\log(n)) = O(\log(n))$ veces.

Por lo tanto, la complejidad total del algoritmo es $O(n^3)O(\log(n)) = O(n^3 \log(n))$.

Falta un detalle: para hacer la búsqueda binaria hay que ordenar los n^2 elementos. Este paso, hecho con un buen algoritmo de ordenación, estaría en $O(n^2 \log(n))$, por lo tanto no empeora la cota obtenida anteriormente.

Algoritmo húngaro

Objetivo

Dado una matriz de costos, queremos hallar un matching que minimice la suma de costos.

Notación

$$\begin{aligned}\sum_{C,M} &= \sum_{\substack{i,j \\ \overrightarrow{ij}\in E(M)}} C_{i,j} \\ &= \sum_{i=1}^{n}\left(\sum_{\substack{j \\ \overrightarrow{ij}\in E(M)}} C_{i,j}\right) \text{ (sumar por filas)} \\ &= \sum_{j=1}^{n}\left(\sum_{\substack{i \\ \overrightarrow{ij}\in E(M)}} C_{i,j}\right) \text{ (sumar por columnas)}\end{aligned}$$

Notar que como M es matching, la expresión $\sum_{\substack{j \\ \overrightarrow{ij}\in E(M)}} C_{i,j}$ es la suma de un único término.

Observación clave #1

Si $C_{i,j} \geq 0 \quad \forall i,j$ y M es matching tal que $ij \in E(M) \Rightarrow C_{i,j} = 0$, entonces claramente M es un matching de suma mínima.

Observación clave #2

Sea C una matriz de costos, tomemos una constante y sea $\tilde{C}$ la matriz que se obtiene al restar a toda una fila la constante. Entonces M minimiza la suma para C si y sólo si la minimiza para $\tilde{C}$. Si C es una matriz cuadrada, esto también es cierto para las columnas.

Veamos una prueba. Sea A un matching cualquiera, entonces

$$\sum_{\tilde{C},A} = \sum_{i=1}^{n}\left(\sum_{\substack{j \\ \overrightarrow{ij}\in E(A)}} \tilde{C}_{i,j}\right)$$

Pero existe k tal que

$$\tilde{C}_{i,j} = \begin{cases} C_{i,j} & i \neq k \\ C_{i,j} - \text{constante} & i = k \end{cases}$$

por lo tanto,

$$\sum_{\tilde{C},A} = \sum_{\substack{i=1 \\ i\neq k}}^{n}\left(\sum_{\substack{j \\ \overrightarrow{ij}\in E(A)}} \tilde{C}_{i,j}\right) + \sum_{\substack{j \\ \overrightarrow{ij}\in E(A)}} \left(\tilde{C}_{k,j} - \text{constante}\right)$$

Entonces, M minimiza la suma respecto a $\tilde{C}$ si y sólo si

$$\sum_{C,M} \leq \sum_{\tilde{C},A} \quad \forall A \Leftrightarrow \sum_{C,M} - \text{constante} \leq \sum_{C,A} - \text{constante} \quad \forall A$$
$$\Leftrightarrow \sum_{C,M} \leq \sum_{C,A} \quad \forall A$$

o sea, si y sólo si M minimiza la suma respecto a C.

Algoritmo, primer intento

1. Restar a cada fila su mínimo (esto introduce al menos un cero).
2. Restar a cada columna su mínimo.
3. Buscar un matching de ceros M_0.
4. Si encontramos M_0 fin del algoritmo. Si no... esperar.

Ejemplo

Sea la matriz de costos

	I	II	III	IV	V
A	7	6	5	10	7
B	2	7	4	2	10
C	4	5	5	15	11
D	10	12	1	10	2
E	12	15	13	9	10

Entonces:

- Mínimo de la fila 1: 5
- Mínimo de la fila 2: 2
- Mínimo de la fila 3: 4
- Mínimo de la fila 4: 1
- Mínimo de la fila 5: 9

Al restar este número a cada fila, obtenemos

	I	II	III	IV	V
A	2	1	0	5	2
B	0	5	2	0	8
C	0	1	1	11	7
D	9	11	0	9	1
E	3	6	4	0	1

Ahora:

- Mínimo de la columna 1: 0
- Mínimo de la columna 2: 1
- Mínimo de la columna 3: 0
- Mínimo de la columna 4: 0

- Mínimo de la columna 5: 1

Al restar estos números a cada columna obtenemos

	I	II	III	IV	V
A	2	0	0	5	1
B	0	4	2	0	7
C	0	0	1	11	6
D	9	10	0	9	0
E	3	5	4	0	0

Sobre esta matriz buscamos un matching de ceros:

$$A\ III, B\ I, C\ II, D\ V, E\ IV$$

Como lo obtuvimos, este matching minimiza la suma y el algoritmo se acabó.

¿Qué sucede si no hallamos matching perfecto?

Recurrimos al Teorema de Hall. Notemos que al buscar el matching de ceros (y fallar) quedamos en presencia de un conjunto de filas S (las filas etiquetadas) y un conjunto de columnas $\Gamma(S)$ (las columnas etiquetadas) donde $|S| > |\Gamma(S)|$. Por ejemplo, en grafo bipartito dado por la matriz de adyacencia:

	I	II	III	IV	V
A	0	0	0	2	3
B	9	12	0	9	17
C	9	6	0	13	0
D	4	2	0	0	0
E	4	3	0	0	6

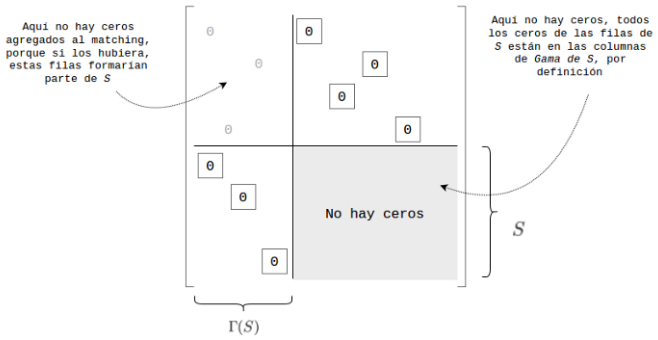
Buscamos un matching de ceros con Dinic y llegamos a la situación:

	I	II	III	IV	V
A	$\boxed{0}$	0	0	2	3
B	9	12	$\boxed{0}$	9	17
C	9	6	0	13	$\boxed{0}$
D	4	2	0	$\boxed{0}$	0
E	4	3	0	0	6

y comenzamos a correr Edmonds-Karp, etiquetando las filas y las columnas, hasta llegar a

	I	II	III	IV	V	
A	$\boxed{0}$	0	0	2	3	
B	9	12	$\boxed{0}$	9	17	~~III~~
C	9	6	0	13	$\boxed{0}$	~~V~~
D	4	2	0	$\boxed{0}$	0	~~IV~~
E	4	3	0	0	6	~~$\emptyset$~~
			~~E~~	~~E~~	~~D~~	

Entonces tenemos $S = \{B, C, D, E\}$ y $\Gamma(S) = \{III, IV, V\}$. Por el teorema de Hall sabemos que no hay matching perfecto de ceros. En general, llegaremos a una situación que se puede representar mediante el siguiente esquema:



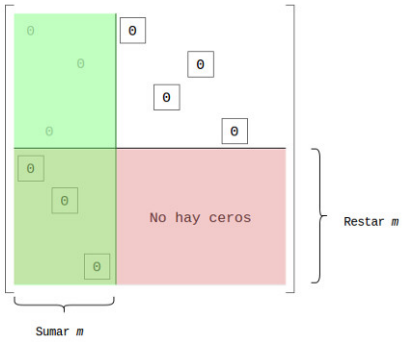
Notemos que para simplificar, las filas de S están todas juntas, así como las columnas de $\Gamma(S)$. En la realidad no necesariamente ocurre esto, pero podemos reordenar la matriz de adyacencia para que aparezcan así, si lo deseamos:

	III	IV	V	I	II
A	0	2	3	$\boxed{0}$	0
B	$\boxed{0}$	9	17	9	12
C	0	13	$\boxed{0}$	9	6
D	0	$\boxed{0}$	0	4	2
E	0	0	6	4	3

Entonces necesitamos agregar ceros en la región que no los tiene, $S \times \overline{\Gamma(S)}$. Por lo tanto, lo que debemos hacer es:

- Buscar $m = \min S \times \overline{\Gamma(S)}$,
- restar m a las filas de S (para introducir al menos un nuevo cero) y sumar m a las columnas de $\Gamma(S)$ (para evitar posibles números negativos),
- continuar buscando matching.

Notar que hay 4 regiones en la matriz de adyacencia:



1. $\overline{S} \times \overline{\Gamma(S)}$: entradas donde no se suma ni se resta nada,
2. $\overline{S} \times \Gamma(S)$: entradas donde se suma m,
3. $S \times \Gamma(S)$: entradas donde se suma m y se resta m, es decir, es como si no se sumara ni se restara nada, y
4. $S \times \overline{\Gamma(S)}$: entradas donde se resta m.

Para hacerlo manualmente más simple, se propone el siguiente método:

- Tachar las columnas de $\overline{S}$ y $\Gamma(S)$,

- Buscamos el mínimo m de las entradas no tachadas, es decir, de $S \times \overline{\Gamma(S)}$,
- Restar m de las entradas no tachadas, y
- Sumar m de las entradas taachadas dos veces, es decir, de $\overline{S} \times \Gamma(S)$.

De este modo, tenemos una forma de hacer aparecer ceros en la región en la que lo necesitamos para extender el matching. Ahora bien, como el número de filas sin matchear es finito, tarde o temprano encontraremos un matching perfecto de ceros y este matching será el que minimice la suma de los costos.

Complejidad del algoritmo húngaro

La complejidad del algoritmo húngaro es

1. $O(n^5)$ si se implementa de forma naif
2. $O(n^4)$ si se implementa correctamente
3. $O(n^3)$ si se implementa de forma eficiente

Prueba

1. **Complejidad del matching inicial** es $O(n^2)$ (por cada una de las n filas, debemos revisar $O(n)$ columnas).
2. Llamemos **Extender** el matching a incrementar el número de lados en 1, es decir, agregar una fila más. El número de veces que esta operación debe realizarse es $O(n)$. Veamos la complejidad de cada extensión. Cuando lanzamos el algoritmo para extender el matching (Edmonds-Karp etiquetando filas y columnas) pueden pasar dos cosas:
   a) O bien para con matching pefecto en $O(n^3)$ (por qué?)
   b) O bien para con $|S| \geq |\Gamma(S)|$, y entonces necesitamos hacer un cambio de matriz (agregar ceros en $S \times \overline{\Gamma(S)}$).
   c) Para hacer un **cambio de matriz** necesitamos
      1) buscar el mínimo elemento m de $S \times \overline{\Gamma(S)}$, lo cual es $O(n^2)$,
      2) sumar m a S, lo cual es $O(n^2)$
      3) restar m a $\Gamma(S)$, lo cual es $O(n^2)$

Si lo hacemos de forma naif, habría que relanzar el algoritmo, una y otra vez hasta obtener matching perfercto de ceros. Esto implicaría una complejidad total de

$$(O(n^3) + O(n^2)) \cdot \text{ (veces que se hace el cambio de matriz)}$$

Por lo tanto, tenemos que acotar el número de veces que se hace el cambio de matriz. Sin embargo, no es necesario relanzar el algoritmo desde cero, ya que podemos continuar con el matching parcial que teníamos. Debemos notar que la operación de hacer un cambio de matriz no nos hace perder el matching que tenemos hasta el momento, ya que en $\overline{S} \times \overline{\Gamma(S)}$ y en $\Gamma(S) \times S$ no sumamos ni restamos nada. Si lo hacemos así, entonces, ¿cuántas veces hay que extender el matching mediante un cambio de matriz?

Propiedad: Luego de un cambio de matriz, o bien se extiende el matching, o bien $|S|$ aumenta.
Prueba: Luego de un cambio de matriz, tenemos (al menos) un nuevo cero en $S \times \overline{\Gamma(S)}$, es decir, en alguna fila i de S intersección alguna columna j de $\overline{\Gamma(S)}$. Pero entonces la columna j se etiqueta con i y se revisará. Tenemos dos casos posibles:

1. la columna j está libre y extendemos el matching

2. la columna j forma parte del matching, es decir, existe una fila k matcheada con j. Entonces la fila k se etiqueta con j y tenemos un nuevo S más grande.

Entonces terminamos con una extensión del matching o se produce un nuevo S de cardinalidad mayor.

Como $|S|$ sólo puede crecer $O(n)$ veces, tenemos que hay a lo sumo n cambios de matriz antes de extender el matching. Por lo tanto, la complejida de la operación **Extender** el matching es

$$\begin{aligned}\text{compl(1 extension)} &= \text{cantidad de (Cambios de Matriz)} \cdot (\text{compl(Cambio de Matriz)}) + (\text{buscar } m) \\ &= O(n) \cdot O(n^2) + O(n^2) \\ &= O(n^3)\end{aligned}$$

y la complejidad total del algoritmo es

$$\begin{aligned}\text{compl(hungaro)} &= \text{compl(matching inicial)} + \text{compl(extension)} \cdot \text{ (cantidad de extensiones)} \\ &= O(n^2) + O(n^3) \cdot O(n) \\ &= O(n^4)\end{aligned}$$

Para lograr una mayor eficiencia y llevar el algoritmo a $O(n^3)$ nos concentramos en bajar la complejidad de la **Extensión**. En lugar de recorrer la matriz sumando o restando m (lo cual es $O(n^2)$) se construye un registro auxilar donde se indica para cada entrada de la matriz cuánto hay que sumarle o restarle cada vez que se lo lea.

Código

Un código sobre un alfabeto A es simplemente una colección de palabras cuyas letras están todas en A. Por ejemplo, el código morse, el código ASCII.

Código de bloque

Un **código de bloque** de longitud n es un subconjunto de A^n (i.e., las palabras tienen todas la misma longitud). Por ejemplo, el código ASCII.

Código binario

Un código es **binario** si su alfabeto es $A = \{0, 1\}$.

Distancia de Hamming

Dadas dos palabras

$$\begin{aligned}x &= x_1 x_2 \dots x_n \\ y &= y_1 y_2 \dots y_n\end{aligned}$$

la distancia de Hamming entre x e y es

$$d_H(x, y) = d_H(y, x) = |\{i : x_i \neq y_i\}|$$

es decir, la cantidad de posiciones en las que sus letras difieren.

Propiedades de la distancia de Hamming

La distancia de Hamming cumple con las siguientes propiedades:

1. $d_H(x,x) = 0$
2. $d_H(x,y) = 0 \Leftrightarrow x = y$
3. $d_H(x,y) \geq 0$
4. $d_H(x,y) = d_H(y,x)$
5. $d_H(x,y) + d_H(y,z) \geq d_H(x,z)$ (desigualdad del triángulo).

Prueba

Las pruebas 1 a 4 son demasiado evidentes. Veamos 5.
Sean

$$A = \{i : x_i = y_i\}$$
$$B = \{i : y_i = z_i\}$$
$$C = \{i : x_i = z_i\}$$

Sea $i \in A \cap B$. Como $x_i = y_i = z_i$, tenemos que $i \in C$. Es decir, se cumple que $A \cap B \subseteq C$ (1). Pero entonces $\overline{C} \subseteq \overline{A \cap B} = \overline{A} \cup \overline{B}$ (ya que al complementar ambos lados de (1), la relación de inclusión se invierte).

Ahora, veamos que $|\overline{C}| = |\overline{A} \cup \overline{B}|$ (ya que si $x_i \neq z_i$ debe ser $x_i \neq y_i \vee y_i \neq z_i$). Pero $|\overline{A} \cup \overline{B}| \leq |\overline{A}| + |\overline{B}|$.

Pero

$$|\overline{C}| = d(x,z)$$
$$|\overline{B}| = d(y,z)$$
$$|\overline{A}| = d(x,y)$$

y por lo tanto

$$|\overline{C}| \leq |\overline{A}| + |\overline{B}| \Leftrightarrow d(x,z) \leq d(x,y) + d(y,z)$$

$B_r(x)$

Dado un código binario de bloque de longitud n, definimos la bola de radio r como

$$B_r = \{y \in \{0,1\}^n : d(x,y) \leq r\}$$

Es decir, todas aquellas palabras en el código que están a distancia menor o igual a r a partir de x.

δ_C

Dado un código C, definimos $\delta_C = \text{mín}\{d(x,y) : x, y \in C, x \neq y\}$ (la mínima distancia no nula entre palabras de C.

Ejemplos:

$C_1 = \{00, 01, 10, 11\}$

$d(x,y)$	00	01	10	11	
00	0	1	1	2	
01	1	0	2	1	$\Rightarrow \delta_{C_1} = 1$
10	1	2	0	1	
11	2	1	1	0	

$C_2 = \{000, 101, 011, 111\}$

$d(x,y)$	000	101	011	111	
000	0	2	2	2	
101	2	0	2	2	$\Rightarrow \delta_{C_2} = 2$
011	2	2	0	2	
111	2	2	2	0	

Peso de Hamming

El peso de Hamming de una palabra x, denotado por $|x|$ es

$$\begin{aligned} |x| &= d(x,0) \\ &= |\{i : x_i \neq 0\}| \\ &= |\{i : x_i = 1\}| \end{aligned}$$

$\{0,1\}^n$ como espacio vectorial

Vamos a pensar a $\mathbb{Z}_2^n = \{0,1\}^n$ como un espacio vectorial, tomando $\mathbb{Z}_2 = \{0,1\}$ con la suma módulo 2 (también llamada suma XOR) y el producto módulo 2 como un cuerpo.

Suma XOR

$$\begin{aligned} 0 \oplus 0 &= 0 \\ 0 \oplus 1 &= 1 \\ 1 \oplus 0 &= 1 \\ 1 \oplus 1 &= 0 \end{aligned}$$

Producto

$$\begin{aligned} 0 \odot 0 &= 0 \\ 0 \odot 1 &= 0 \\ 1 \odot 0 &= 0 \\ 1 \odot 1 &= 1 \end{aligned}$$

Observación: $d(x, y) = |x \oplus y|$, pues

$$\begin{aligned} d(x, y) &= |\{i : x_i \neq y_i\}| \\ &= |\{i : x_i \oplus y_i = 1\}| \\ &= |x \oplus y| \end{aligned}$$

Por ejemplo, sea $x = 10010100$ y $y = 00100101$, entonces

$$d(x, y) = d(10010100, 00100101) = 4$$

ya que difieren en 4 posiciones, y también

$$|x \oplus y| = |10010100 \oplus 00100101| = |10110001| = 4$$

Errores detectados y errores corregidos por un código

Sea C un código y sea $\delta = \delta_C$, entonces decimos que

1. C detecta r errores si y sólo si $B_r(x) \cap C = \{x\}\ \forall\ x \in C$, o sea, para toda palabra x en C se tiene que x es la única palabra en $B_r(x)$ (de esta forma, al producirse r errores, la palabra recibida no puede pasar por otra palabra del código).
2. C corrige t errores si y sólo si $B_t(x) \cap B_t(y) = \emptyset\ \forall\ x, y \in C$, o sea, para todo par de palabras x e y en C se tiene que sus bolas de radio t no se tocan (de esta forma, al producirse t errores, la palabra recibida, aunque no pertenece al código, está más cerca de una palabra w del código que de cualquier otra).

Teorema

1. C detecta $\delta - 1$ errores,
2. C no detecta δ errores,
3. C corrige $\lfloor \frac{\delta-1}{2} \rfloor$ errores, y
4. C no corrige $\lfloor \frac{\delta-1}{2} \rfloor + 1$ errores.

Prueba

1) Supongamos que C no detecta $\delta - 1$ errores. Entonces existe $v \in C$ tal que $B_{\delta-1}(v) \cap C \neq \{v\}$, es decir, existe $w \in C$, con $v \neq w$ tal que $d(v, w) \leq \delta - 1$. Esto es absurdo, pues δ es la mínima distancia no nula entre elementos de C.

2) Sean $v, w \in C$ tales que $d(v, w) = \delta$. Entonces $B_\delta(v) \cap C \neq \{v\}$ pues dicha intersección contiene también al menos a w.

3) Sea $t = \lfloor \frac{\delta-1}{2} \rfloor$. Supongamos $B_t(x) \cap B_t(y) \neq \emptyset$, entonces existe $z \in$ esa intersección y tenemos que

$$\begin{aligned} d(x, z) &\leq t \\ d(z, y) &\leq t \end{aligned}$$

y por lo tanto

$$d(x, z) + d(z, y) \leq 2t$$

Pero por desigualdad triangular tendríamos que

$$d(x,y) \leq 2t = 2\lfloor \tfrac{\delta-1}{2} \rfloor \leq \delta - 1 < \delta$$

lo cual es absurdo.

4) Sea $t = \lfloor \frac{\delta-1}{2} \rfloor$ y sean $x \neq y \in C$ con $d(x,y) = \delta$. Mostraremos que existe un elemento en $B_{t+1}(x) \cap B_{t+1}(y)$. Sea $e = x \oplus y$. Se tiene que $|e| = |x \oplus y| = d(x,y) = \delta$. Sea $e' =$ tomar e y quedarse con $t+1$ de sus unos (tiene exactamente δ, así que tiene también $t+1$). Por último, sea $z = x \oplus e'$. Este es nuestro candidato a existir en la intersección.

Por un lado, tenemos que

$$\begin{aligned} d(x,z) &= |x \oplus z| \\ &= |x \oplus x \oplus e'| \\ &= |e'| \\ &= t+1 \end{aligned}$$

y por lo tanto $z \in B_{t+1}(x)$. Análogamente:

$$\begin{aligned} d(y,z) &= |y \oplus z| \\ &= |y \oplus x \oplus e'| \\ &= |e + e'| \\ &= \delta - (t+1) \\ &\leq t+1 \end{aligned}$$

es decir, $z \in B_{t+1}(y)$. Como $z \in B_{t+1}(x)$ y $z \in B_{t+1}(y)$, debemos aceptar que $B_{t+1}(x) \cap B_{t+1}(x) \neq \emptyset$ y por ende, C no corrige $\lfloor \frac{\delta-1}{2} \rfloor + 1$ errores.

Cota de Hamming

En general, para un código binario de longitud n uno desea un δ alto, así corrige y detecta muchos errores, sin embargo, esto viene a un precio. Mientras más alto es δ, menos palabras podemos incluir en el código. Análogamente, para incluir más palabras manteniendo el δ, debemos aumentar n (transmitir mayor cantidad de bits para la misma cantidad de palabras). La cota de Hamming es un indicador de cómo se relaciona δ con la cantidad de palabras del código.

Teorema

Sea C un código binario de longitud n y sea $t = \lfloor \frac{\delta-1}{2} \rfloor$. Entonces

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

Prueba

Sea $A = \bigcup_{x \in C} B_t(x)$, es decir, la unión de todas las bolas de radio t que se pueden formar a partir de todas y cada una de las palabras x de C. Como $t = \lfloor \frac{\delta-1}{2} \rfloor$, entonces C corrige t errores, lo cual implica que $B_t(x) \cap B_t(y) = \emptyset \; \forall \; x \neq y \in C$, es decir, la unión en A es una unión disjunta. Tenemos entonces que

$$|A| = \sum_{x \in C} |B_t(x)|$$

Para estimar cuántos elementos hay en cada bola de radio t, vamos a dividirlas en capas como una cebolla. Así, definimos $S_r(x) = \{y : d(x, y) = r\}$ (las superficies esféricas de radio r y centro x). Entonces podemos expresar a $B_t(x)$ como

$$B_t(x) = \bigcup_{r=0}^{t} S_r(x)$$

y, nuevamente, la unión es disjunta. Por lo tanto vale que

$$|B_t(x)| = \sum_{r=0}^{t} |S_r(x)|$$

¿Cuánto vale $|S_r(x)|$?

$$\begin{aligned} y \in S_r(x) &\Leftrightarrow d(x, y) = r \\ &\Leftrightarrow |x \oplus y| = r \\ &\Leftrightarrow \exists\, e : |e| = r : y = x \oplus e \end{aligned}$$

Esto nos dice que si y pertenece a $S_r(x)$, entonces existe una palabra e cuyo peso de Hamming es r tal que podemos expresar a y como la suma $x \oplus r$. Pero entonces, haciendo un pequeño abuso de notación, podemos escribir

$$S_r(x) = x \oplus S_r(0)$$

Es decir, la superficie de radio r y centro x se puede pensar como sumar x a cada elemento de la superficie de radio r y centro 0. Algo así como trasladadar $S_r(0)$ hasta estar centrada en x. Pero entonces

$$|S_r(x)| = |S_r(0)|$$

y la cardinalidad de este conjunto no depende en absoluto de x, si no que es el número de vecinos de la palabra $\mathbf{0}$ con r unos, o lo que es igual, $\binom{n}{r}$. Entonces

$$\begin{aligned} |A| &= \sum_{x \in C} |B_t(x)| \\ &= \sum_{x \in C} \left(\sum_{r=0}^{t} |S_r(x)| \right) \\ &= \sum_{x \in C} \left(\sum_{r=0}^{t} \binom{n}{r} \right) \\ &= |C| \cdot \sum_{r=0}^{t} \binom{n}{r} \end{aligned}$$

Despejando, se obtiene que

$$|C| = \frac{|A|}{\sum_{r=0}^{t} \binom{n}{r}}$$

Sólo resta observar que $A \subseteq \{0, 1\}^n$, por lo tanto $|A| \leq 2^n$.

Códigos perfectos

Un código se dice perfecto si

$$|C| = \frac{2^n}{\sum_{r=0}^{t} \binom{n}{r}}$$

Códigos lineales

Un código se de lineal si es un subespacio vectorial de $\{0,1\}^n$. En nuestro caso, el cuerpo es $\mathbb{Z}_2 = \{0,1\}$, luego un código es lineal si y sólo si

1. $x, y \in C \Rightarrow x \oplus y \in C$ (el código es cerrado bajo la suma XOR)
2. $0 \in C$

(la multiplicación por escalar es trivial).

Así, si vale la condición 1, la condición 2 se puede reemplazar por $C \neq \emptyset$, ya que si existe $x \in C$, entonces $\mathbf{0} = x \oplus x \in C$.

Dimensión de un código lineal

Como sabemos de álgebra lineal, la dimensión de un subespacio vectorial es la cardinalidad de cualquier base. Es común denotar por k a la dimensión de un código lineal.

Si un código lineal tiene dimensión k, entonces contiene 2^k elementos.

Prueba

Si $x_1, x_2, \ldots, x_k$ forman una base para C, entonces los elementos de C serán de la forma

$$b_1 x_1 \oplus b_2 x_2 \oplus \cdots \oplus b_k x_k$$

y hay una biyección entre los elementos de C y $\{0,1\}^k$.

Corolario

Todos los códigos lineales tienen cardinalidad que es potencia de 2. En particular, no hay códigos lineales con 6 elementos.

δ de un código lineal

Si C es un código lineal, entonces $\delta = \text{mín}\{|x| : x \in C, x \neq \mathbf{0}\}$.

Prueba

Sea $w = \text{mín}\{|x| : x \in C, x \neq \mathbf{0}\}$. Veamos que

1. $w \leq \delta$
2. $\delta \leq w$

1. Sean $x, y \in C, x \neq y$ con $\delta = d(x,y)$. Entonces $\delta = |x \oplus y|$ y $x \oplus y \in C$ porque C es lineal. Entonces w es el peso de alguien, luego, es mayor o igual al peso mínimo.

2. Tomemos $x \neq \mathbf{0}$ con $|x| = w$. Entonces $w = d(x, \mathbf{0}) \leq \delta$.

Conceptos de Álgebra

Para continuar el estudio de los códigos lineales, se hará uso de los siguientes conceptos vistos en Álgebra:

Espacio fila y espacio columna

Dada una matriz A, el espacio fila de A (denotado por $EF(A)$) es el espacio vectorial generado por las filas de A. Asimismo, el espacio columna de A (denotado por $EC(A)$) es el espacio vectorial generado por las columnas de A.

Observación: en Álgebra, a estos espacios los denominábamos espacio renglón y espacio columna, respectivamente.

Rango fila y rango columna

Rango - Fila = $\dim(EF(A))$

Rango - Columna = $\dim(EC(A))$

Observación: en Álgebra, la dimensión común del espacio columna y del espacio renglón se denominaba simplemente rango de la matriz A.

Teorema del rango

Rango - Fila = Rango - Columna

Observación: en Álgebra, esta propiedad aparece, por ejemplo, como teorema 5.6.1 de Anton, y no tiene este nombre (de hecho, el teorema del rango es otra cosa).

Núcleo de una transformación lineal

Si $T : V \to W$ es una transformación lineal, el núcleo de T es el conjunto de vectores que mapean a $\mathbf{0}$ bajo T, es decir:

$$\text{Nu}(T) = \{\mathbf{x} \in V : T(\mathbf{x}) = \mathbf{0}\}$$

Núcleo de una matriz

Definimos el núcleo de una matriz A como el núclio de la transformación lineal asociada a dicha matriz, es decir:

$$\text{Nu}(A) = \{\mathbf{x} : A\mathbf{x} = \mathbf{0}\}$$

Observación: en Álgebra, este es era el espacio nulo de A.

Relación entre la dimensión del núcleo y la dimensión de la imagen de una TL

Si $T : V \to W$ es una transformación lineal, entonces

$$\dim(\text{Nu}(T)) + \dim(\text{Im}(T)) = \dim(V)$$

para matrices, esto se traduce en

$$\dim(\text{Nu}(A)) + \text{rango}(A) = \text{cantidad de columnas de } A$$

Observación: en Álgebra, aproximadamente esto mismo se vio como Teorema del Rango, que decía:

> Si $T : V \to W$ es una transformación lineal de un espacio vectorial V de dimensión n a un espacio vectorial W de dimensión m, entonces
>
> $$\text{rango}(T) + \text{nulidad}(T) = n$$
>
> En otras palabras, para transformaciones lineales la suma del rango y la nulidad es igual a la dimensión del dominio.

Además, teníamos el Teorema de la dimensión para matrices (5.6.3), que decía:

Si A es una matriz con n columnas, entonces

$$\text{rango}(A) + \text{nulidad}(A) = n$$

Construcción de códigos lineales

Como los códigos lineales son espacios vectoriales, tenemos dos formas sencillas de construirlos:

1. Como espacio fila de una matriz
2. Como espacio nulo de una matriz

Códigos lineales como espacio fila de una matriz

$$C = EF(G)$$

donde G es la matriz generadora. Si $\dim(C) = k$, G debe tener k filas. Si n es la longitud del código, G debe ser $k \times n$.
Ejemplo:

$$G = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \end{bmatrix}$$

Si $\alpha = \langle 1,0,0,1,0,1,1\rangle$, $\beta = \langle 0,1,0,1,0,0,1\rangle$ y $\gamma = \langle 0,0,1,1,1,1,0\rangle$ son las filas de G, entonces C es el espacio generado por $\{\alpha, \beta, \gamma\}$. Es decir, el espacio formado por todas las combinaciones lineales de dichos vectores. Todo vector en C es una combinación lineal de $\{\alpha, \beta, \gamma\}$, o sea:

$$\mathbf{x} \in C \Rightarrow \mathbf{x} = c_1\alpha + c_2\beta + c_3\gamma$$

Es fácil ver que C debe tener $2^3 = 8$ elementos: $\{\mathbf{0}, \alpha, \beta, \gamma, \alpha+\beta, \alpha+\gamma, \beta+\gamma, \alpha+\beta+\gamma\}$, luego:

$$C = \begin{bmatrix} \mathbf{0} \\ \alpha \\ \beta \\ \gamma \\ \alpha+\beta \\ \alpha+\gamma \\ \beta+\gamma \\ \alpha+\beta+\gamma \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 \end{bmatrix}$$

En general:

$$C = EF(G) = \{\text{comb lineal de las filas de } G\} = \{\mathbf{v} : \exists\, \mathbf{u} : \mathbf{v} = \mathbf{u}G\}$$

donde $\mathbf{u}$ es el mensaje que queremos enviar.

Códigos lineales como núcleos de matrices

$$C = \text{Nu}(H)$$

donde H se denomina matriz de chequeo de C.
Ejemplo:

$$H = \begin{bmatrix} 1 & 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

Si $\mathbf{v} \in C$, debe ser $\mathbf{v} = \langle v_1, v_2, v_3, v_4, v_5, v_6 \rangle \in \mathrm{Nu}(H)$, es decir $H\mathbf{v} = \mathbf{0}$:

$$\begin{array}{ccccccccccc} v_1 & & & & \oplus & v_4 & & & \oplus & v_6 & = & 0 \\ & v_2 & \oplus & & & v_4 & \oplus & v_5 & & & = & 0 \\ & & & v_3 & & & & & \oplus & v_6 & = & 0 \end{array}$$

Resolviendo el sistema de ecuaciones, tenemos que debe ser

$$\begin{aligned} v_1 &= v_4 \oplus v_6 \\ v_2 &= v_4 \oplus v_5 \\ v_3 &= v_6 \end{aligned}$$

Es decir que asignando valores arbitrarios en $\{0, 1\}$ a las variables libres v_4, v_5, v_6 obtenemos todo el espacio vectorial C:

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

Relación entre matriz generadora y matriz de chequeo de un código lineal

Sea C un código lineal, entonces $[I|A]$ es generadora de C si y sólo si $[A^t|I]$ es matriz de chequeo de C (de forma similar, $[A|I]$ es de chequeo si y sólo si $[I|A^t]$ es generadora).

Prueba

Debemos probar que $EF([I|A]) = \mathrm{Nu}([A^t|I])$. Para ello,

1. probaremos que $EF([I|A]) \subseteq \mathrm{Nu}([A^t|I])$
2. probaremos que $\dim(EF([I|A])) = \dim(\mathrm{Nu}([A^t|I]))$

1) sea $\mathbf{v} \in EF([I|A])$, entonces $\exists\ \mathbf{u} : \mathbf{v} = \mathbf{u}\,[I|A]$, por lo tanto, $\mathbf{v} = \mathbf{u}||\mathbf{u}A$ (esto es, $\mathbf{u}$ concatenado con el producto $\mathbf{u}A$). Entnces

$$\begin{aligned} \left[A^t|I\right]\mathbf{v}^t &= \left[A^t|I\right]\begin{bmatrix} \mathbf{u}^t \\ A^t\mathbf{u}^t \end{bmatrix} \\ &= A^t\mathbf{u}^t \oplus IA^t\mathbf{u}^t \\ &= A^t\mathbf{u}^t \oplus A^t\mathbf{u}^t \\ &= \mathbf{0} \text{ (ya que estamos en } \mathbb{Z}_2) \end{aligned}$$

2) Sea $k = \dim(EF([I|A])$, entonces A es una matriz $k \times (n-k)$, por lo tanto A^t es $(n-k) \times k$, por lo tanto,

$$\begin{aligned} \dim(\mathrm{Nu}(\left[A^t|I\right])) &= n - \mathrm{rango}(\left[A^t|I\right]) \\ &= n - (n-k) \\ &= k \end{aligned}$$

Ejemplo

Sea

$$H = [A|I] = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

la matriz de chequeo del código lineal C, entonces la matriz generadora es:

$$G = [I|A^t] = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}\right]$$

δ de un código lineal a partir de sus matrices

Sea C cualquier código lineal con matriz de chequeo H, entonces $\delta(C)$ = mínimo número de columnas linealmente dependientes de H. Esto es, si en H hay 2 columnas LD, entonces $\delta(C) = 2$.

Prueba

Recordemos de Álgebra:

$$\mathbf{e}_i = \langle 0, 0, \ldots, 1, \ldots, 0 \rangle$$

denotaba al vector que sólo tiene un uno en la coordenada i-ésima y cero en las demás.
Notar que

$$H\mathbf{e}_i^t = H^{(i)}$$

es la i-ésima columna de H.
Sea $MINLD$ = mínimo número de columnas linealmente dependientes de H. Sea $\mathbf{v} \neq \mathbf{0} \in C$ con $|\mathbf{v}| = \delta(C)$. Entonces $\mathbf{v}$ tiene δ unos, y es la combinación lineal de $\delta \mathbf{e}_i$, es dcir, existen $j_1, j_2, \ldots, j_\delta$ con $\mathbf{v} = \mathbf{e}_{j_1} + \mathbf{e}_{j_2} + \cdots + \mathbf{e}_{j_\delta}$. Como $\mathbf{v} \in C$, debe ser $H\mathbf{v}^t = \mathbf{0}$, y tenemos que

$$\begin{aligned} \mathbf{0} &= H\mathbf{v}^t \\ &= H\left(\mathbf{e}_{j_1} \oplus \mathbf{e}_{j_2} \oplus \cdots \oplus \mathbf{e}_{j_\delta}\right) \\ &= H\mathbf{e}_{j_1} \oplus H\mathbf{e}_{j_2} \oplus \cdots \oplus H\mathbf{e}_{j_\delta} \\ &= H^{(j_1)} \oplus H^{(j_2)} \oplus \cdots \oplus H^{(j_\delta)} \end{aligned}$$

Es decir, las columnas $j_1, j_2, \ldots j_\delta$ de H forman un conjunto linealmente dependiente. Entonces $MINLD \leq \delta$.

Por otro lado, sea $\{H^{(j_1)}, H^{(j_2)}, \ldots, H^{(j_{MINLD})}\}$ un conjunto LD de columnas de H de tamaño mínimo. Entonces existen constantes $c_r \in \{0, 1\}$ no todas iguales a 0 tales que

$$\mathbf{0} = c_1 H^{j_1} \oplus c_2 H^{j_2} \oplus \cdots \oplus c_{MINLD} H^{j_{MINLD}}$$

Entonces tenemos el vector no nulo $\mathbf{v} = c_1\mathbf{e}_{j_1} \oplus c_2\mathbf{e}_{j_2} \oplus \cdots \oplus c_{MINLD}\mathbf{e}_{j_{MINLD}}$ que pertenece al código, ya que

$$\begin{aligned} H\mathbf{v}^t &= H(c_1\mathbf{e}_{j_1}^t) \oplus c_2\mathbf{e}_{j_2}^t \oplus \cdots \oplus c_{MINLD}\mathbf{e}_{j_{MINLD}}^t \\ &= c_1 H^{(j_1)} \oplus c_2 H^{(j_2)} \oplus \cdots \oplus c_{MINLD} H^{(j_{MINLD})} \\ &= \mathbf{0} \end{aligned}$$

(esto es, $\mathbf{v} \in \mathrm{Nu}(H) = C$). Concluimos que $\delta(C) \leq |\mathbf{v}| = MINLD$

Corolario

Si H no tiene la columna $\mathbf{0}$ ni columnas repetidas, entonces $C = \text{Nu}(H)$ corrige al menos un error.

Prueba

Como H no contiene la columna $\mathbf{0}$ no tiene conjuntos LD de sólo un elemento, es decir, $MINLD \geq 2$.

Como H no contiene columnas repetidas, $\nexists\, i, j \quad i \neq j : H^{(i)} = H^{(j)}$, por lo tanto $H^{(i)} \oplus H^{(j)} \neq \mathbf{0} \forall\ i, j \quad i \neq j$ (recordar que estamos en $\mathbb{Z}_2$). Entonces H no tiene conjuntos LD de sólo dos elementos, es decir, $MINLD \geq 3$.

Concluimos que $\delta \geq 3$ y por lo tanto $\lfloor \frac{\delta-1}{2} \rfloor \geq 1$ y C corrige al menos un error.

Códigos de Hamming

Un código de Hamming es un código lineal que tiene una matriz de chequeo en la cual figuran todas las columnas no nulas posibles para un tamaño dado. Esto es, C es un código de Hamming si existen r y H $r \times n$ con $\{H^{(1)}, H^{(2)}, \ldots, H^{(n)}\}$ todas distintas. En general, a cualquier código de Hamming con H $r \times n$ lo denotamos $\mathcal{H}_r$. Ejemplos:

$$\mathcal{H}_2 \to H = \left[\begin{array}{cc|c} 1 & 0 & 1 \\ 0 & 1 & 1 \end{array}\right], G = \left[\begin{array}{cc|c} 1 & 1 & 1 \end{array}\right], \mathcal{H}_2 = \{000, 111\}$$

$$\mathcal{H}_3 \to H = \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right], G = \left[\begin{array}{ccc|cccc} 1 & 1 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right], \mathcal{H}_3 = \{0000000, \ldots\}$$

Propiedad

Los códigos de Hamming son perfectos.

Prueba

Debemos probar que

$$|\mathcal{H}_r| = \frac{2^n}{\sum_{k=0}^{n} \binom{n}{k}} \text{ con } t = \lfloor \tfrac{\delta-1}{2} \rfloor$$

Como en H no hay columnas repetidas ni nulas, entonces $\delta \geq 3$. Pero las columnas

$$\begin{bmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ \vdots \\ 0 \end{bmatrix} \text{ y } \begin{bmatrix} 1 \\ 1 \\ \vdots \\ 0 \end{bmatrix}$$

están en H y conforman un conjunto LD de tres elementos, por lo tanto $\delta \leq 3$. Tenemos entonces que $\delta = 3$ y $t = 1$.

Entonces lo que debemos probar es $|\mathcal{H}_r| = \frac{2^n}{1+n}$. Surge la pregunta: ¿Cuánto vale n en los códigos de Hamming?

$$\begin{aligned} n &= \#\text{ de columnas de } H \\ &= \#\text{ total de columnas no nulas con } r \text{ filas} \\ &= 2^r - 1 \end{aligned}$$

Por lo tanto, $n + 1 = 2^r$ y $\frac{2^n}{1+n} = \frac{2^n}{2^r} = 2^{n-r} = 2^{2^r-1-r}$.

Por otro lado,

$$\begin{aligned}|\mathcal{H}_r| &= 2^k \\ &= 2^{n-\text{rango(H)}} \\ &= 2^{n-r} \\ &= 2^{2^r-1-r}\end{aligned}$$

Uso de un código que detecta un error

Supongamos que el emisor quiere transmitir la palabra $\mathbf{v} \in C$ y supongamos que se producen en el camino r errores, de forma tal que el receptor recibe $\mathbf{w}$. Sea entonces $\mathbf{e} = \mathbf{v} \oplus \mathbf{w}$ y sea H matriz de chequeo de C.

$$\begin{aligned}H\mathbf{w}^t &= H(\mathbf{v} \oplus \mathbf{e})^t \\ &= H\mathbf{v}^t \oplus H\mathbf{e}^t \\ &= \mathbf{0} \oplus H\mathbf{e}^t \text{ (pues } \mathbf{v} \in C) \\ &= H\mathbf{e}^t\end{aligned}$$

Si $\mathbf{e} = \mathbf{0}$ (es decir, si $\mathbf{v} \oplus \mathbf{w} = \mathbf{0}$ y no hubo errores), entonces $H\mathbf{w}^t = \mathbf{0}$.
Si $|\mathbf{e}| = 1$, existe j tal que $\mathbf{e} = \mathbf{e}_j$ y $H\mathbf{w}^t = H\mathbf{e}_j^t = H^{(j)}$ es igual a la columna j de H.
Luego, si asumimos que hubo a lo sumo un error, al recibir la palabra $\mathbf{w}$, se debe efectuar el chequeo $H\mathbf{w}^t$ y

- si se obtiene $\mathbf{0}$ es porque no hubo errores.
- si se obtiene la columna j-esima de H, entonces la palabra enviada y la palabra recibida difieren en el bit j-ésimo.

Uso de un código de Hamming

En el caso particular en que C es un código de Hamming y las columnas de H están ordenadas de modo que la columna i-ésima es la representación binaria del número i como en

$$H = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$

entonces al recibir la palabra $\mathbf{w}$ simplemente se calcula $H\mathbf{w}^t$ y se interpeta el resultado como la representación binaria de la posición que hay que cambiar para obtener $\mathbf{v}$.
Ejemplo: recibimos $\mathbf{w} = \langle 1,1,1,1,0,0,0\rangle$. Realizamos la multiplicación

$$H\mathbf{w}^t = \begin{bmatrix} 1 & 0 & 1 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix} \oplus \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Como $\langle 0,0,1\rangle = 4_{\text{base } 2}$, la palabra enviada fue con mayor probabilidad $\mathbf{v} = \langle 1,1,1,0,0,0,0\rangle$, es decir, $\mathbf{w}$ con el cuarto bit cambiado.

Códigos cíclicos

Un código es cíclico si es lineal e **invariante por rotaciones**. Ejemplo

$$C = \{000, 101, 110, 011\}$$

es cíclico porque

- es lineal (demostración a cargo del lector) y
- para cualquier palabra del código, la que se obtiene de **rotar** sus bits, también está en el código.

Notación

Vamos a especificar mejor qué significa **rotar**. Para ello, conviene adoptar la convención de numerar los bits de la palabra desde cero, así:

$$\mathbf{x} = x_0 x_1 \dots x_{n-1}$$

y definimos

$$\text{rot}(\mathbf{x}) = x_{n-1} x_0 x_1 \dots x_{n-2}$$

Ahora usando esta notación, podemos decir que C es cíclico si y sólo si:

1. $C \neq \emptyset$
2. $\alpha, \beta \in C \Rightarrow \alpha \oplus \beta \in C$
3. $\alpha \in C \Rightarrow \text{rot}(\alpha) \in C$

Identificación de Códigos y Polinomios

Si $\mathbf{v} = v_0 v_1 \dots v_{n-1}$, es una palabra den un código, nos interesa poder pensarla como el polinomio $v_0 + v_1 x + \dots v_{n-1} x^{n-1}$.
Ej:

$$\langle 1, 0, 1, 1 \rangle = 1 + x^2 + x^3$$

Así, podríamos pensar en la multiplicaciones de palabras del codigo como multiplicación de polinomios:

$$\begin{aligned} \langle 1, 0, 1, 1 \rangle \otimes \langle 0, 1, 0, 0 \rangle &= (1 + x^2 + x^3)x \\ &= x + x^3 + x^4 \\ &= \langle 0, 1, 0, 1, 1 \rangle \end{aligned}$$

El problema con esta multiplicación es que me devolvió una palabra más larga. Entonces vamos a tomar

$$\mathbb{Z}_2[x] = \{ \text{ los polinomios con coeficientes en } \mathbb{Z}_2 \}$$

y si $m(x)$ es un polinomio, entonces

$$\mathbb{Z}_2[x]/m(x) = \{ \text{ los restos de dividir los polinomos de } \mathbb{Z}_2[x] \text{ por } m(x) \text{ con la suma } \oplus \}$$

- SUMA: $p(x) \oplus q(x)$ sería la suma polinomial normal, pero en $\mathbb{Z}_2$.

- PROD: $p(x) \otimes q(x)$ tomar el resto del producto usual dividido por $m(x)$.

Tomaremos $m(x) = 1 + x^n$, de modo que

$$p(x) \otimes q(x) = \text{resto de dividir } p(x)q(x) \text{ por } 1 + x^n$$

esto es lo mismo que realizar el producto usual $p(x)q(x)$ e identificar:

$$\begin{gathered} x^n \text{con } 1 \\ x^{n+1} \text{con } x \\ x^{n+2} \text{con } x^2 \\ \vdots \end{gathered}$$

Notar que en $\mathbb{Z}_2[x]/1+x^n$ tenemos que $1+x^n \equiv 0 \mod (1+x^n)$ (análogamente a lo que pasa, por ejemplo, en $\mathbb{Z}_k$, donde k es 0), y $x^n \equiv 1 \mod (1+x^n)$, pues $x^n = 1 \cdot (1+x^n) + 1$ (recordar que los coeficientes están en $\mathbb{Z}_2$).

Retomando el ejemplo anterior con esta nueva herramienta:

$$\begin{aligned} \langle 1,0,1,1\rangle \otimes \langle 0,1,0,0\rangle &= \left[(1+x^2+x^3)x\right] \mod (1+x^4) \\ &= \left[x+x^3+x^4\right] \mod (1+x^4) \\ &= x+x^3+1 \\ &= 1+x+x^3 \\ &= \langle 1,1,0,1\rangle \end{aligned}$$

Observación

$$\begin{aligned} \mathbf{v} \otimes x &= (v_0 + v_1 x + \cdots + v_{n-1}x^{n-1}) \otimes x \\ &= \left[v_0 x + v_1 x^2 + \cdots + v_{n-1}x^n\right] \mod (1+x^n) \\ &= v_0 x + v_1 x^2 + \cdots + v_{n-1} \\ &= v_{n-1} + v_0 x + v_1 x^2 + \cdots + v_{n-2} \\ &= \text{rot}(\mathbf{v}) \end{aligned}$$

Podemos redefinir a los códigos cíclicos como aquellos códigos lineales que son invariantes bajo la (multiplicación módulo $1 + x^n$) por x.

Corolario

Si C es cíclico, $\mathbf{v} \in C$ y p es cualquier palabra (polinomio), entonces

$$p \otimes v \in C$$

donde v denota a la palabra $\mathbf{v}$ vista como polinomio.

Prueba

$$\begin{aligned} p \otimes v &= \left(\sum_{i=0}^{n} p_i x^i \right) \otimes v \\ &= \sum_{i=0}^{n} p_i (x^i \otimes v) \end{aligned}$$

y $x^i \otimes v$ es la rotación sucesiva (i veces) de la palabra v, por lo tanto pertenece al código.

Generador de un código cíclico

Dado un código cíclico C, el **generador** de C es un polinomio no nulo $g(x)$ tal que $gr(g(x)) \leq gr(v(x)) \forall\ v \in C, v \neq 0$.

El generador de un código cíclico es único

Si C es un código cícligo, y $g(x)$ es el generador de C, entonces $g(x)$ es único.

Prueba

Supongamos que hubieran dos polinomios generadores de C: g_1 y g_2. Entonces $gr(g_1) \leq gr(g_2)$ y $gr(g_2) \leq gr(g_1)$ implica que $gr(g_1) = gr(g_2)$. Sea entonces $t = gr(g_1) = gr(g_2)$.

$$g_1 = a_0 + a_1 x + \cdots + a_{t-1} x^{t-1} + x^t$$

$$g_2 = b_0 + b_1 x + \cdots + b_{t-1} x^{t-1} + x^t$$

Como C es lineal, $g_1 \oplus g_2 \in C$, pero

$$g_1 \oplus g_2 = (a_0 \oplus b_0) + (a_1 \oplus b_1)x + \ldots (a_{t-1} \oplus b_{t-1}) x^{t-1}$$

donde el t-ésimo término se cancela y $gr(g_1 \oplus g_2) \leq t - 1 < gr(g_1)$.
Como g_1 es generador, debe ser $gr(g_1 \oplus g_2) = 0$, de lo cual se desprende que $g_1 = g_2$.

Múltiplos del polinomio generador

Sea C un código cíclico de longitud n y sea $g(x)$ su polinomio generador. Sea $v \in C$, entonces $\exists\ u \in \mathbb{Z}_2\,[x]$ con

$$v = u \otimes g$$

es decir, que v se puede expresar como múltiplo del polinomio generador. De hecho, podemos elegir u de forma tal que $v = u \cdot g$ (producto usual). Esta propiedad explica el nombre que recibe el polinomio generador.

Prueba

Sea $v \in C$, dividamos v entre g. Entonces existen q, r con $gr(r) < gr(g)$ tales que

$$v = qg \oplus r$$

Esto es equivalente a

$$qg = v \oplus r$$

Pero $v \in C \Rightarrow gr(v) < n$ y $gr(r) < gr(g) < n$. Luego, $gr(v \oplus r) < n$. Ahora bien, $qg = v \oplus r$, así que concluimos que $gr(qg) < n$.

Pero en general, si $gr(p) < n$, se tiene que $p \mod (1+x^n) = p$, entonces $qg \mod (1+x^n) = qg$, o sea que $q \otimes g = qg$.

Como $g \in C$, tenemos que $qg \in C$. Pero qg era $v \oplus r$, entonces $v \oplus r \in C$ y como $v \in C$ debemos aceptar que $r \in C$.

Pero $gr(r) < gr(g)$ y g generador implican que $r = 0$ y $v = gq$.

3 propiedaded del generador

Sea C un código cíclico de longitud n y dimensión k y sea g su polinomio generador, entonces

1. $gr(g) = n - k$
2. $g \mid 1 + x^n$
3. Si $g = g_0 + g_1 x + \cdots + g_{n-1} x^{n-1}$, entonces $g_0 = 1$

Prueba

1) Recordemos que vimos que

$$v \in C \Rightarrow \exists\, u : v = ug \wedge gr(u) < n$$

Para que $gr(ug) < n$, debe ser $gr(u) + gr(g) < n$, esto es

$$gr(u) < n - gr(g)$$

Por lo tanto, podemos caracterizar al código como

$$C = \{v : \exists\, u, gr(u) < n - gr(g) : v = ug\}$$

Entonces hay una biyección entre C y $\{u : gr(u) < n - gr(g)\}$, con lo cual las cardinalidades son iguales. Entonces tenemos que

$$\begin{aligned} |C| &= |\{u : gr(u) < n - gr(g)\}| \\ 2^k &= 2^{n-gr(g)} \\ k &= n - gr(g) \end{aligned}$$

2) Dividamos $1 + x^n$ por g:

$$1 + x^n = qg \oplus r \qquad gr(r) < gr(g)$$

Es decir que

$$r = qg \oplus (1 + x^n)$$

y tomando módulo $1 + x^n$ en ambos lados:

$$\begin{aligned} r \mod (1+x^n) &= (qg \oplus (1+x^n)) \mod (1+x^n) \\ &= q \otimes g \end{aligned}$$

Pero $gr(g) < n \Rightarrow r \mod (1+x^n) = r$ y por lo tanto $r = q \otimes g$. Pero $q \otimes g \in C$ y como $gr(r) < gr(g)$ concluimos que $gr(r) = 0$, es decir, $r = 0$.

3) Por 2) sabemos que $g \mid 1 + x^n$. Sea q tal que $qg = 1 + x^n$, debe ser $q_0 = g_0 = 1$.

Uso del polinomio generador para codificar palabras

Sea $g(x) = 1 + x^2 + x^3$ el generador de un código de bloque con $n = 7$ (notar que el código tiene dimensión $n - gr(g) = 7 - 3 = 4$, es decir, tiene $2^4 = 32$ palabras).

Método 1

Dado u con $gr(u) < n - gr(g) = k$, lo más fácil es mandar $v = ug$ y recibida esta palabra y corregida sus errores, decodificarla como v/g.

CODIFICACIÓN: Queremos codificar la palabra 1101, que puede pensarse como el polinomio $1 + x + x^3$. Entonces

$$\begin{aligned} (1+x+x^3) \otimes g(x) &= (1+x+x^3) \otimes (1+x^2+x^3) \\ &= \left[(1+x+x^3)(1+x^2+x^3)\right] \mod (1+x^7) \\ &= 1 + x^2 + x^3 + x + x^3 + x^4 + x^3 + x^5 + x^6 \\ &= 1 + x^2 + \cancel{x^3} + x + \cancel{x^3} + x^4 + x^3 + x^5 + x^6 \\ &= 1 + x + x^2 + x^3 + x^4 + x^5 + x^6 \\ &= \langle 1,1,1,1,1,1,1 \rangle \end{aligned}$$

Si en cambio queremos codificar la palabra 1010 equivalente al poliniomio $1 + x^2$, entonces

$$\begin{aligned} (1+x^2) \otimes g(x) &= (1+x^2) \otimes (1+x^2+x^3) \\ &= \left[(1+x^2)(1+x^2+x^3)\right] \mod (1+x^7) \\ &= 1 + x^2 + x^3 + x^2 + x^4 + x^5 \\ &= 1 + \cancel{x^2} + x^3 + \cancel{x^2} + x^4 + x^5 \\ &= 1 + x^3 + x^4 + x^5 \\ &= \langle 1,0,0,1,1,1,0 \rangle \end{aligned}$$

La matriz generadora asociada a este método es de la forma

$$G = \begin{bmatrix} g \\ xg \\ \vdots \\ x^{k-1}g \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 1 & 1 \end{bmatrix}$$

Notar que la matriz identidad $I_{4\times 4}$ no aparece en ningun lugar de G así obtenida.

Método 2

Si $gr(p) < n$, entonces $p \oplus (p \mod g(x)) \in C$, pues $[p \oplus (p \mod g(x))] \mod g(x) = 0$, entonces codificamos u con $gr(u) < k$ como $ux^{n-k} \oplus (ux^{n-k} \mod g(x))$

CODIFICACIÓN: Para codificar la palabra 1101, tenemos que enviar:

$$(1 + x + x^3)x^{n-k} \oplus \left[(1 + x + x^3)x^{n-k} \mod g(x)\right] = (x^3 + x^4 + x^6) \oplus \left[(x^3 + x^4 + x^6) \mod (g(x))\right]$$

El problema es que no tenemos una idea clara de cuánto es $(x^3 + x^4 + x^6) \mod (g(x))$, para lo cual generamos una tabla con $x^i \mod (g(x))$:

Para $i < gr(g)$ el cálculo es simple:

$$1 \mod (g(x)) = 1 \mod (1 + x^2 + x^3) = 1$$
$$x \mod (g(x)) = x \mod (1 + x^2 + x^3) = x$$
$$x^2 \mod (g(x)) = x^2 \mod (1 + x^2 + x^3) = x^2$$

Para $i = gr(g)$, razonamos del siguiente modo:

$$\begin{aligned} g(x) &\mod (g(x)) = 0 \\ (1 + x^2 + x^3) &\mod (g(x)) = 0 \\ x^3 &\mod (g(x)) = 1 + x^2 \end{aligned}$$

Luego

$$x^3 \mod (g(x)) = x^3 \mod (1 + x^2 + x^3) = 1 + x^2$$

Para $i > gr(g)$, utilizamos los valores previamente calculados:

$$\begin{aligned} x^4 \mod (g(x)) &= xx^3 \mod (g(x)) \\ &= x(1 + x^2) \mod (g(x)) \\ &= (x + x^3) \mod (g(x)) \\ &= x + 1 + x^2 \\ &= 1 + x + x^2 \end{aligned}$$

$$\begin{aligned} x^5 \mod (g(x)) &= xx^4 \mod (g(x)) \\ &= x(1 + x + x^2) \mod (g(x)) \\ &= (x + x^2 + x^3) \mod (g(x)) \\ &= x + x^2 + 1 + x^2 \\ &= 1 + x \end{aligned}$$

$$\begin{aligned} x^6 \mod (g(x)) &= xx^5 \mod (g(x)) \\ &= x(1 + x) \mod (g(x)) \\ &= (x + x^2) \mod (g(x)) \\ &= x + x^2 \end{aligned}$$

En resumen, tenemos las siguientes equivalencias:

i	x^i	$x^i \mod (g(x))$
0	1	1
1	x	x
2	x^2	x^2
3	x^3	$1+x^2$
4	x^4	$1+x+x^2$
5	x^5	$1+x$
6	x^6	$x+x^2$

Como chequeo de lo obtenido hasta el momento, debería ser cierto que $x^7 \mod (g(x)) = 1$:

$$\begin{aligned} x^7 \mod (g(x)) &= xx^6 \mod (g(x)) \\ &= x(x+x^2) \mod (g(x)) \\ &= (x^2+x^3) \mod (g(x)) \\ &= x^2+1+x^2 \\ &= 1 \end{aligned}$$

Volviendo a nuestra codificación de la palabra 1101, ahora podemos reemplazar los valores utilizando la tabla y tenemos que

$$\begin{aligned} (1+x+x^3)x^{n-k} \oplus \left[(1+x+x^3)x^{n-k} \mod (g(x))\right] &= (x^3+x^4+x^6) \oplus \left[(x^3+x^4+x^6) \mod (g(x))\right] \\ &= (x^3+x^4+x^6) \oplus \left[(1+x^2) \oplus (1+x+x^2) \oplus (x+x^2)\right] \\ &= x^3+x^4+x^6+1+x^2+1+x+x^2+x+x^2 \\ &= 1+1+x+x+x^2+x^2+x^2+x^3+x^4+x^6 \\ &= \cancel{1}+\cancel{1}+\cancel{x}+\cancel{x}+\cancel{x^2}+\cancel{x^2}+x^2+x^3+x^4+x^6 \\ &= x^2+x^3+x^4+x^6 \\ &= \langle 0,0,1,1,1,0,1\rangle \end{aligned}$$

Notar que el final de la palabra codificada coincide con la palabra original (lo cual facilitaría su decodificación). Esto es así porque en la matriz generadora sí aparece la matriz identidad.

La matriz generadora asociada a este método es

$$G = \begin{bmatrix} x^{n-k} \mod (g(x)) \oplus x^{n-k} \\ x^{n-k+1} \mod (g(x)) \oplus x^{n-k+1} \\ \vdots \\ x^{n-1} \mod (g(x)) \oplus x^{n-1} \end{bmatrix} = \begin{bmatrix} \text{codificacion de } 1 \\ \text{codificacion de } x \\ \vdots \\ \text{codificacion de } x^{n-1} \end{bmatrix} =$$

$$\begin{bmatrix} x^3 \mod (g(x)) \oplus x^3 \\ x^4 \mod (g(x)) \oplus x^4 \\ \vdots \\ x^6 \mod (g(x)) \oplus x^6 \end{bmatrix} = \left[\begin{array}{ccc|cccc} 1 & 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{array}\right]$$

Error Trapping

Este método no siempre funciona, pero es útil cuando se producen errores en ráfaga (varios errores todos juntos). Es decir, es favorable cuando los errores están juntos y es problemático cuando los errores están muy dispersos dentro de la palabra.

Supongamos que se manda v y llega w con a lo sumo t errores, con $t = \lfloor \frac{\delta-1}{2} \rfloor$. Sea

$$s_0 = w \mod (g)$$
$$s_i = x^i w \mod (g) \quad \text{para } i = 0, 1, \ldots, n-1$$

en particular, s_0 se denomina síndrome.

Teorema

Sea $\tilde{e} = x^{n-i} s_i \mod (1 + x^n)$. Si $|\tilde{e}| \leq t$, entonces $v = w + \tilde{e}$.

Ejemplo

$g = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} + x^{11}$, $n = 23$, $t = \lfloor \frac{\delta - 1}{2} \rfloor = 3$.

Recibimos $w = 00100101001111011000000$, que es equivalente al polinomio $x^2 + x^5 + x^7 + x^{10} + x^{11} + x^{12} + x^{13} + x^{15} + x^{16}$. Para calcular $s_0 = w \mod (g)$, hacemos la tabla de equivalencias:

$$\begin{aligned}
x^{11} &\mod g = 1 + x^2 + x^4 + x^5 + x^6 + x^{10} \\
x^{12} &\mod g = 1 + x + x^2 + x^3 + x^4 + x^7 + x^{10} \\
x^{13} &\mod g = 1 + x + x^3 + x^6 + x^8 + x^{10} \\
x^{14} &\mod g = 1 + x + x^5 + x^6 + x^7 + x^9 + x^{10} \\
x^{15} &\mod g = 1 + x + x^4 + x^5 + x^7 + x^8 \\
x^{16} &\mod g = x + x^2 + x^5 + x^6 + x^8 + x^9
\end{aligned}$$

En principio haría falta seguir hasta x^{23}, pero como w sólo llega hasta x^{16}, las demás equivalencias no se van a utilizar. Ahora bien,

$$s_0 = w \mod g = x^4 + x^7 + x^6 + x^8 + x^9$$

Como $|s_0| = 5 > 3$, no puedo hacer nada con esto y busco s_1:

$$s_1 = x s_0 \mod g = x^5 + x^7 + x^8 + x^9 + x^{10}$$

Como $|s_1| = 5 > 3$, no puedo hacer nada con esto y busco s_2:

$$s_2 = x s_1 \mod g = 1 + x^2 + x^4 + x^5 + x^8 + x^9$$

Como $|s_2| = 6 > 3$, no puedo hacer nada con esto y busco s_3:

$$s_3 = x s_2 \mod g = x + x^3 + x^5 + x^6 + x^9 + x^{10}$$

Como $|s_3| = 6 > 3$, no puedo hacer nada con esto y busco s_4:

$$s_4 = x s_3 \mod g = 1 + x^5 + x^7$$

Como $|s_4| = 3 \leq 3$, podemos parar y tenemos que

$$\begin{aligned}
\tilde{e} &= x^{23-4} s_4 \mod (1 + x^{23}) \\
&= x^{19}(1 + x^5 + x^7) \mod (1 + x^{23}) \\
&= (x^{19} + x^{24} + x^{26}) \mod (1 + x^{23}) \\
&= x^{19} + x + x^3 \\
&= x + x^3 + x^{19}
\end{aligned}$$

y $v = w \oplus \tilde{e}$, es decir, la palabra recibida, cambiando los bits 1, 3 y 19 (contando desde 0).

Prueba

Sea $\tilde{v} = w + \tilde{e}$, queremos ver que $\tilde{v} = v$. Como $gr(w) < n$ y $gr(\tilde{e}) < n$, se tiene que $gr(\tilde{v}) < n$, por lo tanto, si vemos que $g \mid \tilde{v}$ habremos probado que $\tilde{v} \in C$ y tendremos que:

- $d(v, w) \leq t$ (por hipótesis, $w \in B_t(v)$)
- $d(\tilde{v}, w) = |\tilde{e}| \leq t$, por lo tanto $w \in B_t(\tilde{v})$

con lo cual $B_t(v) \cap B_t(\tilde{v}) \neq \emptyset$ y deberíamos aceptar entonces que $v = \tilde{v}$. Veamos entonces $g \mid \tilde{v}$. Para ello, vamos a calcular $\tilde{v} \mod (g)$:

$$\begin{aligned} \tilde{v} \mod (g) &= (w \oplus \tilde{e}) \mod (g) \\ &= w \mod (g) \oplus \tilde{e} \mod (g) \\ &= s_0 \oplus \tilde{e} \mod (g) \end{aligned}$$

Como $\tilde{e} = x^{n-i} s_i \mod (1 + x^n)$, entonces existe q tal que $x^{n-1} s_i = \tilde{e} \oplus q(1 + x^n)$.
Pero $s_i = x^i w \mod (g)$, entonces existe p tal que $x^i w = s_i \oplus pg$, es decir, $s_i = x^i w \oplus pg$.
Uniendo estos dos últimos resultados, tenemos que

$$\begin{aligned} \tilde{e} &= x^{n-i} s_i \oplus q(1 + x^n) \\ &= x^{n-i}(x^i w \oplus pg) \oplus q(1 + x^n) \\ &= x^n w \oplus x^{n-i} pg \oplus q(1 + x^n) \\ &= (1 + x^n + 1) w \oplus x^{n-i} pg \oplus q(1 + x^n) \\ &= (1 + x^n) w \oplus w \oplus x^{n-i} pg \oplus q(1 + x^n) \\ &= w \oplus x^{n-i} pg \oplus (1 + x^n)(q \oplus w) \end{aligned}$$

Por lo tanto

$$\begin{aligned} \tilde{e} \mod (g) &= \left[w \oplus x^{n-i} pg \oplus (1 + x^n)(q \oplus w)\right] \mod (g) \\ &= w \mod (g) \oplus \left[x^{n-i} pg\right] \mod (g) \oplus \left[(1 + x^n)(q \oplus w)\right] \mod (g) \\ &= w \mod (g) \oplus \left[(1 + x^n)(q \oplus w)\right] \mod (g) \qquad (\text{porque } g \mid g) \\ &= w \mod (g) \qquad (\text{porque } g \mid 1 + x^n) \end{aligned}$$

Entonces $\tilde{e} = w \mod (g) = s_0$ y por lo tanto $s_0 \oplus \tilde{e} \mod (g) = s_0 \oplus s_0 = 0$.

Algoritmo Wave

El algoritmo Wave (de Tarjan) resuelve el problema de encontrar un flujo maximal en un Network. Elimina la filosofía de caminos aumentantes que está presente en los algoritmos estudiados previamente (Ford-Fulkerson, Edmonds-Karp, Dinic). La idea es, sobre un Network Auxiliar, mantener un **pre-flujo** bloqueante (mandar el máximo que se puede) que aún no es flujo porque no cumple la propiedad de la conservación, y trabajar hasta convertirlo en flujo. Pero el invariante es que es bloqueante.

Idea general

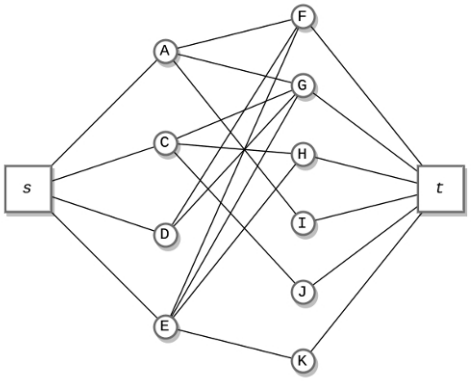
1. Crear Network Auxiliar
2. Mandar de s a sus vecinos todo lo que se pueda
3. Para todos los vértices en orden $BFS(s)$, si el vértice está desbalanceado (es decir, el flujo entrante es mayor que el flujo saliente) mandar hacia adelante todo lo que se pueda. Si el vértice no se logra balancear, **bloquearlo**.
4. Para todos los vértices en orden $BFS(s)$ invertido, si el vértice está bloqueado y no balanceado y devolver flujo hasta que se balancee.
5. Repetir 3) y 4) hasta que todos los vértices estén balanceados.
6. Volver a comenzar desde 1), hasta que no se pueda llegar de s a t.

Ejemplo

Dado el siguiente Network:

$sA : 8$	$AG : 3$	$CH : 3$	$EG : 6$	$GB : 4$
$sC : 7$	$AI : 8$	$CJ : 5$	$EH : 7$	$Ht : 4$
$sD : 10$	$BI : 2$	$DF : 4$	$EK : 3$	$It : 9$
$sE : 15$	$BJ : 2$	$DG : 2$	$Ft : 7$	$Jt : 15$
$AF : 4$	$CG : 2$	$EF : 3$	$Gt : 6$	$Kt : 3$

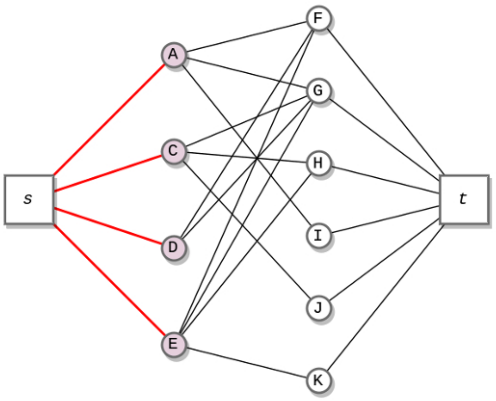
Se mostrará cómo proceder con el primer Network Auxilar:



Orden $BFS(s)$:

s | A C D E | F G I H J K | t

Comenzamos mandando desde s todo lo que la capacidad de los lados permite:



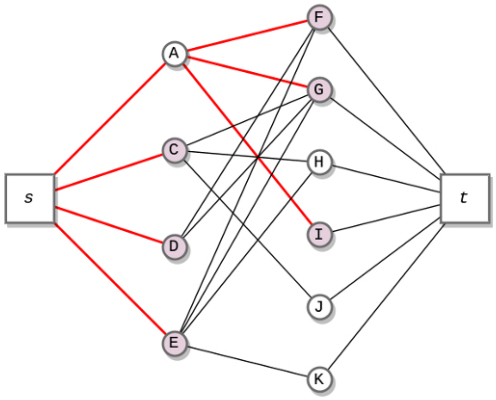
Notar que ahora los vértices A, C, D y E están **desbalanceados**. Estos desbalanceos los vamos registrando:

s	A	C	D	E	F	G	I	H	J	K	t
-40	8	7	10	15							

Las capacidades actualizadas son:

$sA : \not{8}0$	$AG : 3$	$CH : 3$	$EG : 6$	$GB : 4$
$sC : \not{7}0$	$AI : 8$	$CJ : 5$	$EH7$	$Ht : 4$
$sD : \not{1}\not{0}0$	$BI : 2$	$DF : 4$	$EK3$	$It : 9$
$sE : \not{1}\not{5}0$	$BJ : 2$	$DG : 2$	$Ft : 7$	$Jt : 15$
$AF : 4$	$CG : 2$	$EF : 3$	$Gt : 6$	$Kt : 3$

A continuación, se trata de mandar de cada vértice desbalanceado hacia adelante todo lo que se pueda hasta balancearlo. Si no se lo puede balancear, se lo marca como bloqueado. Comenzamos por el vértice A, que para balancearse manda 4 a F, 3 a G y 1 a I:



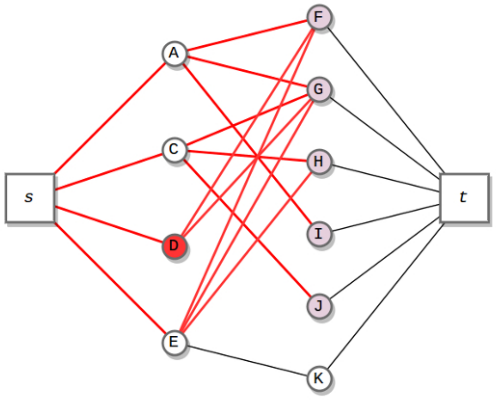
Ahora los desbalanceos son:

s	A	C	D	E	F	G	I	H	J	K	t
-40	~~8~~	7	10	15	4	3	1				
	~~4~~										
	~~1~~										
	0										

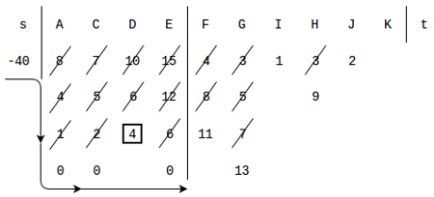
y las capacidades, actualizadas:

$sA : \not{8}0$	$AG : \not{3}0$	$CH : 3$	$EG : 6$	$GB : 4$
$sC : \not{7}0$	$AI : \not{8}7$	$CJ : 5$	$EH : 7$	$Ht : 4$
$sD : \not{1}\not{0}0$	$BI : 2$	$DF : 4$	$EK : 3$	$It : 9$
$sE : \not{1}\not{5}0$	$BJ : 2$	$DG : 2$	$Ft : 7$	$Jt : 15$
$AF : \not{4}0$	$CG : 2$	$EF : 3$	$Gt : 6$	$Kt : 3$

De este modo logramos balancear al vértice A. El proceso continúa hacia la derecha, intentando balancear cada vértice desbalanceado. Seguimos con C, D, E:

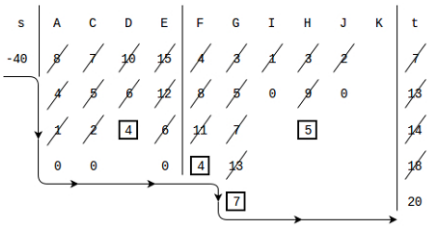


Notar que el vértice D no se pudo balancear. Habiendo recibido 10 desde s, sólo pudo enviar hacia adelante un total de 6. Lo recuadramos para recordar que está bloqueado:



$sA : \not{8}0$	$AG : \not{3}0$	$CH : \not{3}0$	$EG : \not{6}0$	$GB : 4$
$sC : \not{7}0$	$AI : \not{8}7$	$CJ : \not{5}3$	$EH : \not{7}1$	$Ht : 4$
$sD : \not{1}\not{0}0$	$BI : 2$	$DF : \not{4}0$	$EK : 3$	$It : 9$
$sE : \not{1}\not{5}0$	$BJ : 2$	$DG : \not{2}0$	$Ft : 7$	$Jt : 15$
$AF : \not{4}0$	$CG : \not{2}0$	$EF : \not{3}0$	$Gt : 6$	$Kt : 3$

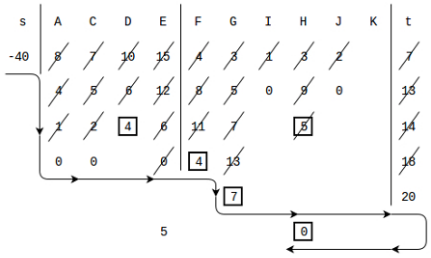
Continuamos con la ola hacia la derecha, intentando balancear ahora los vértices F, G, I, H, J, K que finalmente depositarán su flujo en t.



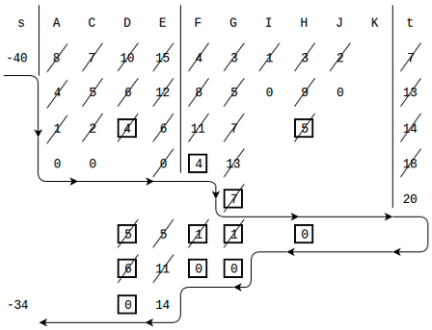
quedando las capacidades actualizadas así:

$sA : \not{8}0$	$AG : \not{3}0$	$CH : \not{3}0$	$EG : \not{6}0$	$GB : 4$
$sC : \not{7}0$	$AI : \not{8}7$	$CJ : \not{5}3$	$EH : \not{7}1$	$Ht : \not{4}0$
$sD : \not{1}\not{0}0$	$BI : 2$	$DF : \not{4}0$	$EK : 3$	$It : \not{9}8$
$sE : \not{1}\not{5}0$	$BJ : 2$	$DG : \not{2}0$	$Ft : \not{7}0$	$Jt : 1\not{5}3$
$AF : \not{4}0$	$CG : \not{2}0$	$EF : \not{3}0$	$Gt : \not{6}0$	$Kt : 3$

Llegados a este punto, el algoritmo indica que debemos recorrer los vértices en el órden $BFS(s)$ inverso, revisando los vértices bloqueados y no balanceados y devolver desde ellos flujo hasta que se balanceen. Así, el primer vértice que nos encontramos para tratar es H. Vemos que tiene un sobrante de 5, del cual se tiene que deshacer devolviendo. Observando la tabla con las capacidades notamos que H está recibiendo 3 desde C y 6 desde E. La idea es devolver primero al último que le envió. De ese modo, le puede devolver 5 a E y quedar así balanceado:

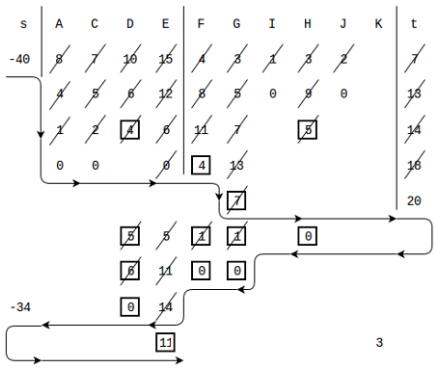


Notemos que ahora E perdió su estado de balanceo. Continuamos, devolviendo el excedente de G, F y D (en ese orden):

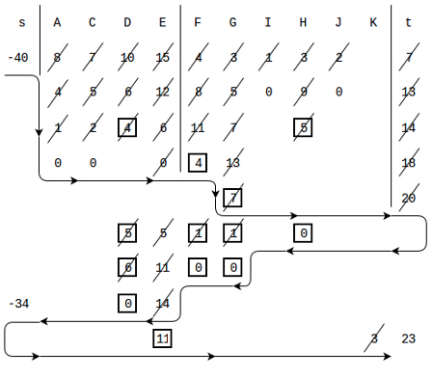


$sA : 80$	$AG : 30$	$CH : 30$	$EG : 006$	$GB : 4$
$sC : 70$	$AI : 87$	$CJ : 53$	$EH : 716$	$Ht : 40$
$sD : 1006$	$BI : 2$	$DF : 401$	$EK : 3$	$It : 98$
$sE : 150$	$BJ : 2$	$DG : 201$	$Ft : 70$	$Jt : 153$
$AF : 40$	$CG : 20$	$EF : 303$	$Gt : 60$	$Kt : 3$

Notar que en todo momento la suma de los desbalanceos de todos los vértices tiene que ser 0. Además, en este momento el único vértice que está desbalanceado es E. La ola vuelve, de izquierda a derecha:

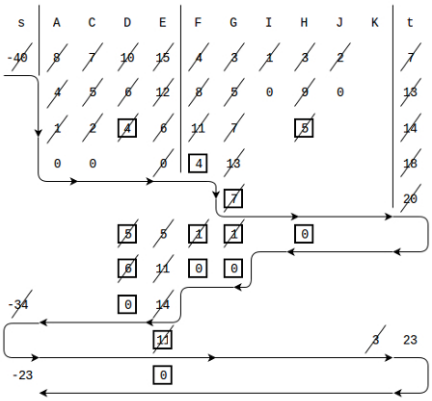


Como E sólo puede liberarse de 3 del total de su desbalanceo, lo bloqueamos. Continuamos:



$sA : 80$	$AG : 30$	$CH : 30$	$EG : 606$	$GB : 4$
$sC : 70$	$AI : 87$	$CJ : 53$	$EH : 716$	$Ht : 40$
$sD : 1006$	$BI : 2$	$DF : 401$	$EK : 3$	$It : 98$
$sE : 150$	$BJ : 2$	$DG : 201$	$Ft : 70$	$Jt : 153$
$AF : 40$	$CG : 20$	$EF : 303$	$Gt : 60$	$Kt : 30$

Por último, la ola vuelve, balanceando al único vértice bloqueado que queda, E y se obtiene finalmente un flujo:



$sA : 80$	$AG : 30$	$CH : 30$	$EG : 606$	$GB : 4$
$sC : 70$	$AI : 87$	$CJ : 53$	$EH : 716$	$Ht : 40$
$sD : 1006$	$BI : 2$	$DF : 401$	$EK : 3$	$It : 98$
$sE : 15011$	$BJ : 2$	$DG : 201$	$Ft : 70$	$Jt : 153$
$AF : 40$	$CG : 20$	$EF : 303$	$Gt : 60$	$Kt : 30$

Concluido así el trabajo con el primer Network Auxiliar, se debe continuar construyendo Networks Auxiliares hasta que no se pueda. Entonces el flujo así obtenido será maximal.

Pseudocódigo

```
// para el blocking step, es decir, para trabajar sobre un Network
    Auxiliar
g = 0 // flujo intermedio, no el maximal.
forall(x):
    B(x) = 0 // no está bloqueado
    D(x) = 0 // desbalanceo
    P(x) = ∅ // vérties que envían flujo a x

forall x ∈ Γ+(s):
    g(sx⃗) = c(sx⃗) // saturar todos los vértices que parten de s
```

```
    D(x) = c(\vec{sx})
    D(s) = D(s) - c(\vec{sx})
    P(x) = {s}

while D(s) + D(t) ≠ 0:

    // ola hacia la derecha
    for s in [s+1..t-1]: // orden BFS(s)
        if D(x) > 0:
            forward_balance(x)

    // ola hacia la izquierda
    for s in [t-1..s+]: // orden BFS(s)
        if D(x) > 0 and B(x):
            backward_balance(x)

return g
```

Donde las funciones auxilaries son:

```
forward_balance(x):
    while D(x) > 0 and Γ+(x) ≠ ∅:
        tomar y ∈ Γ+(x)
        if B(y) = 1:
            // removerlo de los candidatos, para no considerarlo más
            Γ+(x) = Γ+(x) - {y}
        else:
            A = min(D(x), c(\vec{xy}) - g(\vec{xy}))
            g(\vec{xy}) = g(\vec{xy}) + A
            D(x) = D(x) - A
            D(y) = D(y) + A
            P(y) = P(y) ∪{x}
            if g(\vec{xy}) = c(\vec{xy}):
                // removerlo de los candidatos, para no considerarlo m
                    ás
                Γ+(x) = Γ+(x) - {y}
    if D(x) > 0:
        B(x) = 1 // bloquearlo si todavía está debalanceado
```

```
backward_balance(x):
    while D(x) > 0:
        tomar y ∈ P(x)
        A = min(D(x), g(\vec{yx}))
        g(\vec{yx}) = g(\vec{xx}) - A
        D(x) = D(x) - A
        D(y) = D(y) + A
        if g(\vec{yx}) = 0:
            P(x) = P(x) - {y}
```

Complejidad algoritmo Wave

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

Prueba

Wave es un algoritmo que trabaja en networks auxiliares y como la distancia entre s y t en networks auxiliares sucesivos aumenta, puede haber a lo sumo n networks auxiliares, así que:

$$\text{compl(Wave)} = n \cdot \text{compl(paso bloqueante)}$$

El paso bloqueante de Wave consiste en una serie de olas hacia adelante y hacia atrás. Las olas hacia adelante son una sucesión de `forward_balance` (FB) y las olas hacia atrás, una sucesión de `backward_balance` (BB). Cada FB y BB, a su vez, es una sucesión de

- buscar vecinos (hacia adelante o hacia atrás)
- "procesar" el lado resultante

donde esos procesamientos son complicados, pero $O(1)$. Por lo tanto, la complejidad del paso bloqueante es

$$\text{Compl(paso bloqueante)} = \text{número total de "procesamientos" de un lado}$$

que no es igual al número de lados porque un lado puede ser procesado más de una vez.

Estos procesamientos de lados, podemos dividirlos en dos categorías:

1. aquellos que saturan o vacían el lado
2. aquellos que no

Sea T = número de procesamientos del tipo 1, y Q = número de procesamientos del tipo 2. Lo que queremos es acotar $T + Q$.

Veamos primero T.

Supongamos que el lado $\overrightarrow{xy}$ se satura. ¿Puede volver a saturarse? Para volver a saturarse, primero tiene que vaciarse (aunque sea, un poco). Es decir, primero y debe devolver algo de flujo a x. Pero para que en Wave y le devuelva flujo a x debe ocurrir que y esté bloqueado, porque sólo se ejecuta BB(y) cuando y está bloqueado. Pero si y está bloqueado, x no puede mandarle flujo nunca más. Como consecuencia, **los lados sólo pueden saturarse una vez**.

Supongamos ahora que el lado $\overrightarrow{xy}$ se vacía completamente. ¿Puede volver a vaciarse? Mediante un razonamiento análogo al anterior, podemos ver que para volver a vaciarse, primero hay que enviar algo de flujo a través suyo. Pero si ya lo vaciamos una vez, está bloqueado, con lo cual x no puede volver a mandar flujo a y, y el lado $\overrightarrow{xy}$ no puede volver a vaciarse. Como consecuencia, **los lados sólo pueden vaciarse una vez**.

Entonces, $T \leq 2m$.

Veamos ahora Q.

El siguiente punto es clave:

- en cada FB, a los sumo un lado no se satura (**Explicación**: supongamos que x puede mandar a $y_1, y_2, \ldots, y_n$. Primero se manda todo lo que se puede a y_1. Si aún queda flujo, se manda todo lo que se puede a y_2. Así sucesivamente, llegado el caso de que el flujo saliente de x se agote por completo en el paso i, a lo sumo el último vértice y_i puede estar no saturado. Los anteriores -si es que hubo- se llenaron. Los siguientes -si es que hay- aún no se usaron. El actual -y_i- quizás se sature, o quizás no. Por eso, a los sumo un lado no se satura).
- en cada BB, a lo sumo un lado no se vacía (**Explicación**: razonamiento análogo al caso FB).

Por lo tanto, $Q \leq$ número total de FB y BB.

El número total de FB en cada ola hacia la derecha es a lo sumo n. El número total de BB en cada ola hacia la izquierda es a lo sumo n. Entonces el número total de FB + BB es $\leq 2n \cdot$ (número de ciclos "ola hacia la derecha / ola hacia la izquierda").

Ahora bien, en cada ola hacia adelante, pueden o no bloquearse algunos vértices. Si no se bloquea ninguno, están todos balanceados (excepto por s y t) y se trata de la última iteración. Por ende, en toda iteración que no sea la última, se bloquea **al menos** un vértice. Por lo tanto, el número total de ciclos "ola hacia la derecha / ola hacia la izquierda") es $\leq (n-2)+1 = n-1$.

Entonces, $Q \leq 2n(n-1) = O(n^2)$.

Reuniendo las cotas obtenidas para T y para Q,

$$T + Q \leq 2m + O(n^2) = O(m) + O(n^2) = O(n^2)$$

y la complejidad de Wave es $O(n^3)$ como queríamos probar.

Problema de decisión

Un problema de decisión es una función $\pi : I \to \{\text{si, no}\}$, donde I es el conjunto de instancias del problema. Por ejemplo: dado un grafo, podemos colorearlo con dos colores? donde una instancia sería un grafo en particular.

NP

Decimos que un problema $\in NP$ si es un problema de decisión para el cual existe un algoritmo no determinístico polinomial que lo resuelva si la respuesta es SI. Consideramos que la fuente de aleatoriedad y es $O(1)$, es decir, no incrementa la complejidad del problema. Si la respuesta es NO, no podemos tomar la respuesta del algoritmo como algo concluyente.

Formalización

$\pi : I \to \{\text{si, no}\} \in NP$ si y sólo si $\exists\ J$ y un polinomio q y $\rho : I \times J \to \{\text{si, no}\}$ con

1. $\rho \in P$
2. $\forall x \in I \quad \pi(x) = \text{SI} \Leftrightarrow\ \exists\ y \in J : \begin{cases} \rho(x, y) = \text{SI} \\ |y| \leq q(|x|) \end{cases}$

VP

Decimos que un problema de decisión está en la clase VP si existe un certificado para la respuesta "SI"que se puede verificar en tiempo polinomial. Ejemplos:

- Inv: dada una matriz cuadrada, ¿es esta invertible?. Un certificado para una matriz A podría consistir en una matriz B tal que $AB = I$. La comprobación en tiempo polinomial consitiría en realizar la multiplicación.
- k-color: el certificado es un coloreo con k colores, y la comprobación sería chequear en tiempo polinomial que el mismo es propio, que trivialmente es $O(m)$.

NP = VP

Prueba

- $VP \subseteq NP$

Supongamos que un problema π está en VP (es decir, que existe un certificado para la respuesta "SI"que puede ser chequeado en tiempo polinomial). Entonces, podemos elegir en forma no determinística el certificado y verificarlo en tiempo polinomial, por lo tanto $\pi \in NP$.

- $NP \subseteq VP$

Supongamos que un problema π está en NP (es decir, que si la respuesta es "SI", existe un algoritmo no determinístico que lo resuelve en tiempo polinomial). Entonces existe un çertificado"de dicha solución (el conjunto de elecciones no determinísticas) y la verificación consiste en correr el algoritmo polinomial con esas elecciones (esto es, quitando el componente no determinístico).

CO-NP

La categoría de problemas conocida como $CO - NP$ es similar a NP pero con la respuesta "NO". Ejemplos:

- Primos: ¿es n primo? Un certificado para el "NO"lo constituye un entero b tal que $b|n$.
- Matching bipartito: por teorema de Hall
- Inv: $\mathbf{x} \neq \mathbf{0}$ tal que $A\mathbf{x} = \mathbf{0}$.

Preguntas abiertas

Vimos que

- Inv $\in NP \cap CO - NP$ y que $Inv \in P$.
- matching bipartitito es un problema que está en P y pertenece a $NP \cap CO - NP$.
- Primo $\in CO - NP \cap NP$ y primo $\in P$.

Preguntas abiertas:

$$¿P = NP \cap CO - NP?$$

$$¿P = NP?$$

SAT

Terminología

- Expresiones booleanas: alguna función de varialbes booleanas
- Variable booleana: variable que toma valores en $\{0, 1\}$
- signos usuales: $\wedge, \vee, \overline{x}$
- Disyunción: $\vee$ de cosas
- Conjunción: $\wedge$ de cosas
- literal: una variabla o negación de variable

Definición:

SAT (por satisfactibilidad) es un problema de decisión central para el área:

> Dada una expresión booleana: ¿existe una asignación que la vuelva verdadera?

Casos particulares

CNF-Sat: Es como SAT, pero se requiere que la expresión booleana esté en Çonjunctive Normal Form", es decir, que sea una conjunción de disyunciones, por ejemplo: $(x_1 \vee x_2) \wedge (\overline{x_1} \vee x_2)$.

DNF-Sat: Es como SAT, pero se requiere que la expresión booleana de entrada esté en forma disjuntiva normal, es decir, que sea una disyunción de conjunciones, por ejemplo: $(x_1 \wedge x_2) \vee (\overline{x_1} \wedge x_2 \wedge x_3)$.

Teorema de Cook (informal)

CNF-Sat es el más "difícil"de todos los problemas de NP. En particular, si CNF-Sat $\in P$, entonces $P = NP$.

Reducción polinomial

Sean $\pi_1 : I_1 \to \{\text{si, no}\}$ y $\pi_2 : I_2 \to \{\text{si, no}\}$ dos problemas de decisión. Decimos que π_1 se reduce polinomialmente a π_2, y escribimos $\pi_1 \leq_p \pi_2$ si existe un algoritmo polinomial $A : I_1 \to I_2$. Es decir, contamos con un algoritmo que en tiempo polinomial transforma instancias de π_1 en instancias de π_2, tal que para todo $I \in I_1$ se cumple que $\pi_1(I) = \pi_2(A(I))$.

4-color $\leq_p$ SAT

Sea $G = (V, E)$ un grafo (una instancia de 4-color), con vértices $v_1, v_2, \ldots, v_n$. Vamos a construir (polinomialmente) una expresión booleana en forma CNF.

Variables

$4n$ variables: $x_{i,j}$, con $i = 1, 2, \ldots, n$ y $j = 1, 2, 3, 4$

Expresiones

Sea $A_i = x_{i,1} \vee x_{i,2} \vee x_{i,3} \vee x_{i,4}$ (alguno de los cuatro colores fue asignado al vértice v_i)

Sea $A = A_1 \wedge A_2 \wedge \cdots \wedge A_n$ (y esto ocurre para todos los vértices)

Sea $Q_{i,h,j,r} = \overline{x_{i,j}} \vee \overline{x_{h,r}}$, con $h, i = 1, 2, \ldots, n$ y $j, r = 1, 2, 3, 4$ (el vértice v_i no recibió el color j o el vértice v_h no recibió el color r).

Sea $D_i = Q_{i,i,1,2} \wedge Q_{i,i,1,3} \wedge Q_{i,i,1,4} \wedge Q_{i,i,2,3} \wedge Q_{i,i,2,4} \wedge Q_{i,i,3,4}$ (el vértice v_i no recibió 2 o más colores)

Sea $D = D_1 \wedge D_2 \wedge \cdots \wedge D_n$ (y esto ocurre para todos los vértices)

Sea $F_{i,h} = Q_{i,h,1,1} \wedge Q_{i,h,2,2} \wedge Q_{i,h,3,3} \wedge Q_{i,h,4,4}$(el vértice v_i y el vértice v_h no recibieron el mismo color)

Sea $F = \langle \forall\, i, h : F_{i,h} : (v_i, v_h) \in E \rangle$ (y esto ocurre para todos los vértices que forman lado)

Sea $B = A \wedge D \wedge F$

Estructura de la prueba

Debemos ver que $\chi(G) \leq 4 \Leftrightarrow B$ es satisfactible. Primeros veremos $\Rightarrow$, esto es, supondremos que $\chi(G) \leq 4$ y daremos una asignación a las variables $x_{i,j}$ que satisfaga B. Luego veremos $\Leftarrow$, esto es, supondremos que exite una asignación de las variables que satisfaga B y construiremos un coloreo propio de G con a lo sumo 4 colores.

Prueba

$\Rightarrow$

Supongamos que $\chi(G) \leq 4$. Entonces existe un coloreo propio de G con 4 o menos colores. Sea $\overrightarrow{b} \in \mathbb{Z}_2^{4n}$ definido por

$$b_{i,j} = \begin{cases} 1 & \text{si } c(v_i) = j \\ 0 & \text{si no} \end{cases}$$

Veamos que $B(\overrightarrow{b}) = 1$, es decir, $A(\overrightarrow{b}) = 1$, $D(\overrightarrow{b}) = 1$ y $F(\overrightarrow{b}) = 1$.

- $A(\overrightarrow{b}) = 1$. Debemos probar que para todo $i \in \{1, 2, \ldots, n\}$ se cumple $b_{i,1} \vee b_{i,2} \vee b_{i,3} \vee b_{i,4}$. Pero $c(v_i) \in \{1, 2, 3, 4\}$, es decir, el vértice v_i recibió alguno de los 4 colores. Por esto, podemos afirmar que $\exists\ j \in \{1, 2, 3, 4\} : c(v_i) = j$, y para ese j, $b_{i,j} = 1$. Esto hace la expresión $A_i(\overrightarrow{b}) = 1$ para todo i.

- $D(\overrightarrow{b}) = 1$. Debemos probar que para todo $i \in \{1, 2, \ldots, n\}$ se cumple $Q_{i,i,j,r}(\overrightarrow{b}) = 1$, es decir, $\overline{b_{i,j}} \vee \overline{b_{i,r}} = 1$ para todo i, j, r con $r \in 1, 2, 3, 4$ y $j < r$. Supongamos que no. Entonces, existen i, j, r con $j < r$ tales que $\overline{b_{i,j}} \vee \overline{b_{i,r}} = 0$. Esto implica que $\overline{b_{i,j}} = \overline{b_{i,r}} = 0$, es decir, $b_{i,j} = b_{i,r} = 1$. O sea que $c(v_i) = j = r$. Esto es absurdo, pues $j < r$, el vértice no puede haber recibido dos colores distintos.
- $F(\overrightarrow{b}) = 1$. Debemos probar que para todo i, h tales que $(v_i, v_h) \in E$ se cumple $F_{i,h}(\overrightarrow{b}) = 1$. Supongamos que no es cierto. Entonces, existen i, h con $(v_i, v_h) \in E$ y $F_{i,h}(\overrightarrow{b}) = 0$. Esto es, $\exists\ j \in \{1, 2, 3, 4\}$ tal que $Q_{i,h,j,j}(\overrightarrow{b}) = 0$. Esto significa que $\overline{b_{i,j}} \vee \overline{b_{h,j}} = 0$, es decir que $b_{i,j} = b_{h,j} = 1$. Pero esto es lo mismo que decir que $c(v_i) = c(v_h) = j$, lo cual es absurdo, pues c es un coloreo propio y $(v_i, v_h) \in E$.

$\Leftarrow$

Spongamos $B(\overrightarrow{b}) = 1$, es decir, $A(\overrightarrow{b}) = 1$, $D(\overrightarrow{b}) = 1$ y $F(\overrightarrow{b}) = 1$.

1.

$$A(\overrightarrow{b}) = 1 \Rightarrow A_i(\overrightarrow{b}) = 1\ \forall\ i \in \{1, 2, 3, 4\}$$
$$\Rightarrow \forall\ i\ \exists\ j : b_{i,j} = 1$$

2.

$$D(\overrightarrow{b}) = 1 \Rightarrow\ \forall\ i\ \ \nexists\ j \neq r : b_{i,j} = b_{i,r} = 1$$

Por 1 y 2, se cumple que $\forall\ i\ \exists!\ j : b_{i,j} = 1$. Definimos $c(v_i) = j$.

Debemos probar que ese coloreo es propio, es decir, que si $(v_i, v_h) \in E$, entonces $c(v_i) \neq c(v_h)$. Pero como $F(\overrightarrow{b}) = 1$, debe ser $F_{i,h}(\overrightarrow{b}) = 1$, es decir, $\nexists\ j$ tal que $b_{i,j} = b_{h,j} = 1$, esto es el vértice v_i y el vértice v_h no recibieron el mismo color.

k-SAT

Es como **CNF-Sat**, pero se requiere que haya exactamente k literales en cada disyunción.

NP-HARD

Si $\pi \leq_p \rho\ \forall\ \pi \in NP$, ρ se dice $NP - HARD$.

NP-Completo

Un problema de decisión ρ se dice $NP-COMPLETO$ si ρ es $NP-HARD$ y además $\rho \in NP$. Para ver que ρ es $NP - COMPLETO$ basta ver que SAT $\leq_p \rho$.

3-SAT es NP-COMPLETO

Veamos que SAT $\leq_p$ 3−SAT.

Sea $B = D_1 \wedge D_2 \wedge \cdots \wedge D_m$ una instancia de CNF-SAT, con variables $x_1, x_2, \ldots, x_n$, con $D_i = l_{i,1} \vee l_{i,2} \vee \cdots \vee l_{i,k_i}$, donde $l_{i,j}$ son literales (una variable o su negación). Para cada D_i construiremos, en tiempo polinomial, un E_i que sea conjunción de 3 literales (con variables extra). De modo que $\tilde{B} = E_1 \wedge E_2 \wedge \cdots \wedge E_m$ será nuestra instancia de 3-SAT. Para realizar esta construcción, vamos a proceder de diferente manera según la cantidad de literales k_i presentes en D_i:

Si $k_i = 3$

Definimos

$$E_i = D_i.$$

Si $k_i = 2$

Tomamos una variable extra, y_{i1}, y definimos

$$E_i = (l_{i1} \vee l_{i2} \vee y_{i1}) \wedge (l_{i1} \vee l_{i2} \vee \overline{y_{i1}}).$$

Si $k_i = 1$

Tomamos dos variables extra, y_{i1} y y_{i2}, y definimos

$$E_i = (l_{i1} \vee y_{i1} \vee y_{i2}) \wedge (l_{i1} \vee y_{i1} \vee \overline{y_{i2}}) \wedge (l_{i1} \vee \overline{y_{i1}} \vee y_{i2}) \wedge (l_{i1} \vee \overline{y_{i1}} \vee \overline{y_{i2}}).$$

Si $k_i = 4$

Agregamos $k_i - 3$ variables nuevas. Defninimos

$$E_i = (l_{i1} \vee l_{i2} \vee y_{i1}) \wedge (l_{i3} \vee \overline{y_{i1}} \vee y_{i2}) \wedge (l_{i4} \vee \overline{y_{i2}} \vee y_{i3}) \wedge \cdots \wedge (l_{ik_i-2} \vee \overline{y_{ik_i-4}} \vee y_{ik_i-3}) \wedge (l_{ik_i-1} \vee l_{ik_i} \vee \overline{y_{ik_i-3}})$$

Debemos ver que

$$\exists\ \overrightarrow{b} : B(\overrightarrow{b}) = 1 \Leftrightarrow \exists\ \overrightarrow{a} : \tilde{B}(\overrightarrow{b}, \overrightarrow{a}) = 1$$

Es decir, que si B es satisfactible mediante una asignación $\overrightarrow{b}$, entonces $\tilde{B}$ es satisfactible mediante una ampliación de dicha asignación (para abarcar a las nuevas variables).

$\Leftarrow$

Supongamos que $\tilde{B}(\overrightarrow{b}, \overrightarrow{a}) = 1$, debemos ver que $B(\overrightarrow{b}) = 1$. Supongamos, para llegar a un absurdo, que $B(\overrightarrow{b}) = 0$. Entonces existe un i tal que $D_i = 0$. Pero sabemos que $E_i(\overrightarrow{b}, \overrightarrow{a}) = 1$. Entonces analicemos cómo se construyó E_i.

- Si $k = 3$, $D_i(\overrightarrow{b}) = 0 \Rightarrow E_i = 0 \Rightarrow \tilde{B}(\overrightarrow{b}, \overrightarrow{a}) = 0$, lo cual es absurdo.
- Si $k = 2$, $D_i(\overrightarrow{b}) = l_{i1}(\overrightarrow{b}) \vee l_{i2}(\overrightarrow{b}) = 0 \Rightarrow E_i = y_{i1} \wedge \overline{y_{i1}} = 1$, lo cual es absurdo.
- Si $k = 1$, $D_i(\overrightarrow{b}) = l_{i1} = 0 \Rightarrow (a_{i1} \vee a_{i2}) \wedge (a_{i1} \vee \overline{a_{i2}}) \wedge (\overline{a_{i1}} \vee a_{i2}) \wedge (\overline{a_{i1}} \vee \overline{a_{i2}}) = 1$, lo cual es absurdo.
- Si $k > 3$, $D_i(\overrightarrow{b}) = 0 \Rightarrow l_{ij}(\overrightarrow{b}) = 0\ \forall\ j$. Es decir, podemos ignorar los literales l_{ij}. Pero como $\tilde{B}(\overrightarrow{b}, \overrightarrow{a}) = 1$, debe ser cierto que $1 = a_{i1} \wedge (\overline{a_{i1}} \vee a_{i2}) \wedge (\overline{a_{i2}} \vee a_{i3}) \wedge \cdots \wedge (\overline{a_{ik_i-4}} \vee a_{ik_i-3}) \wedge \overline{a_{ik_i-3}}$, o dicho de otro modo, $1 = a_{i1} \wedge (a_{i1} \Rightarrow a_{i2}) \wedge (a_{i2} \Rightarrow a_{i3}) \wedge \cdots \wedge (a_{ik-4} \Rightarrow a_{ik-3}) \wedge \overline{a_{ik-3}}$, lo cual es absurdo.

En cualquier caso, llegamos a una contradicción. Luego, debe ser $B(\overrightarrow{b}) = 1$.

$\Rightarrow$

Supongamos $B(\overrightarrow{b}) = 1$. Para los $k_i = 1$ y $k_i = 2$, es trivial ver que definiendo $a_{ij} = 0$, $D_i(\overrightarrow{b}) = 1 \Rightarrow E_i(\overrightarrow{b}, \overrightarrow{a}) = 1$. Veamos para $k > 3$. $D_i(\overrightarrow{b}) = 1 \Rightarrow \exists\ j : l_{ij}(\overrightarrow{b}) = 1$. Entonces, definimos

$$a_{i1} = a_{i2} = \cdots = a_{ij-2} = 1$$
$$a_{ij-1} = a_{ij} = \cdots = a_{ik_i-3} = 0$$

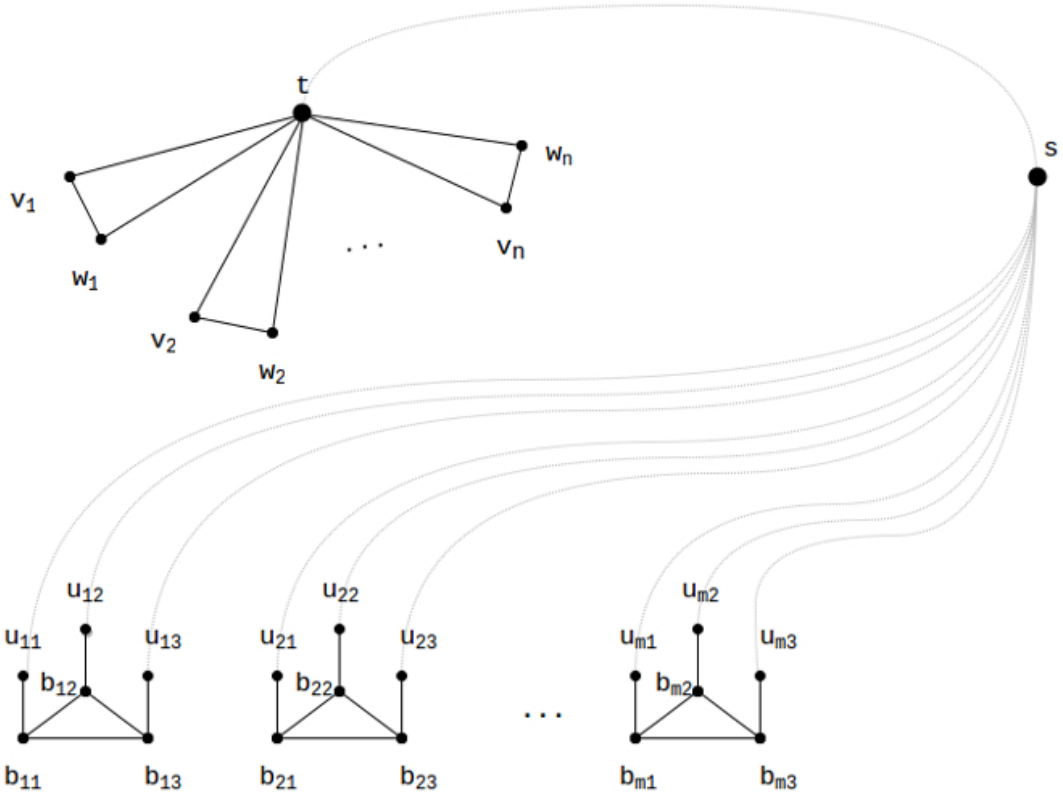
Así, obtenemos:

$$\begin{aligned}
E(\overrightarrow{b}, \overrightarrow{a}) = (l_{i1}(\overrightarrow{b}) \vee l_{i2}(\overrightarrow{b}) \vee a_{i1}) &\quad \text{notar que este termino es 1 por } a_{i1} \\
\wedge\ (l_{i3}(\overrightarrow{b}) \vee \overline{a_{i1}} \vee a_{i2}) &\quad \text{notar que este termino es 1 por } a_{i2} \\
\vdots & \\
\wedge\ (l_{ij-1}(\overrightarrow{b}) \vee \overline{a_{ij-3}} \vee a_{ij-2}) &\quad \text{notar que este termino es 1por } a_{ij-2} \\
\wedge\ (l_{ij}(\overrightarrow{b}) \vee \overline{a_{ij-2}} \vee a_{ij-1}) &\quad \text{notar que este termino es 1por } l_{ij} \\
\wedge\ (l_{ij+1}(\overrightarrow{b}) \vee \overline{a_{ij-1}} \vee a_{ij}) &\quad \text{notar que este termino es 1por } \overline{a_{ij-1}} \\
\vdots & \\
\wedge\ (l_{ik-1}(\overrightarrow{b}) \vee l_{ik} \vee \overline{a_{ik_i}}) &\quad \text{notar que este termino es 1por } \overline{a_{ik_i}} \\
= 1 &
\end{aligned}$$

3-COLOR es NP-COMPLETO

Ya hemos visto que 3-COLOR $\in$ NP, veamos que 3-SAT $\leq_p$ 3-COLOR, es decir, dada B una expresión booleana en CNF con 3 literales por disjunción (una instancia de 3-SAT), debemos crear un grafo G tal que B es satisfactible $\Leftrightarrow \chi(G) \leq 3$.

Sea $B = D_1 \wedge D_2 \wedge \cdots \wedge D_m$ con variables $x_1, x_2, \ldots, x_n$, donde cada $D_i = l_{i1} \vee l_{i2} \vee l_{i3}$. Nuestro G será un $G_1 = (V, E \cup F)$, es decir, $G = (V, E)$ con lados extra F, determinados según B. Este es G:



Tenemos n triángulos t, v_j, w_j (uno por cada variable presente en la expresión booleana) y m garras $u_{i1}, u_{i2}, u_{i3}, b_{i1}, b_{i2}, b_{i3}$ (una por cada disjunción).

Además, hay que agregar F. Para definir F, recordemos que todo literal es una variable o negación de una variable. Para cada literal l, definimos

$$\psi(l) = \begin{cases} v_k & \text{si } l = x_k \\ w_k & \text{si } l = \overline{x_k} \end{cases}$$

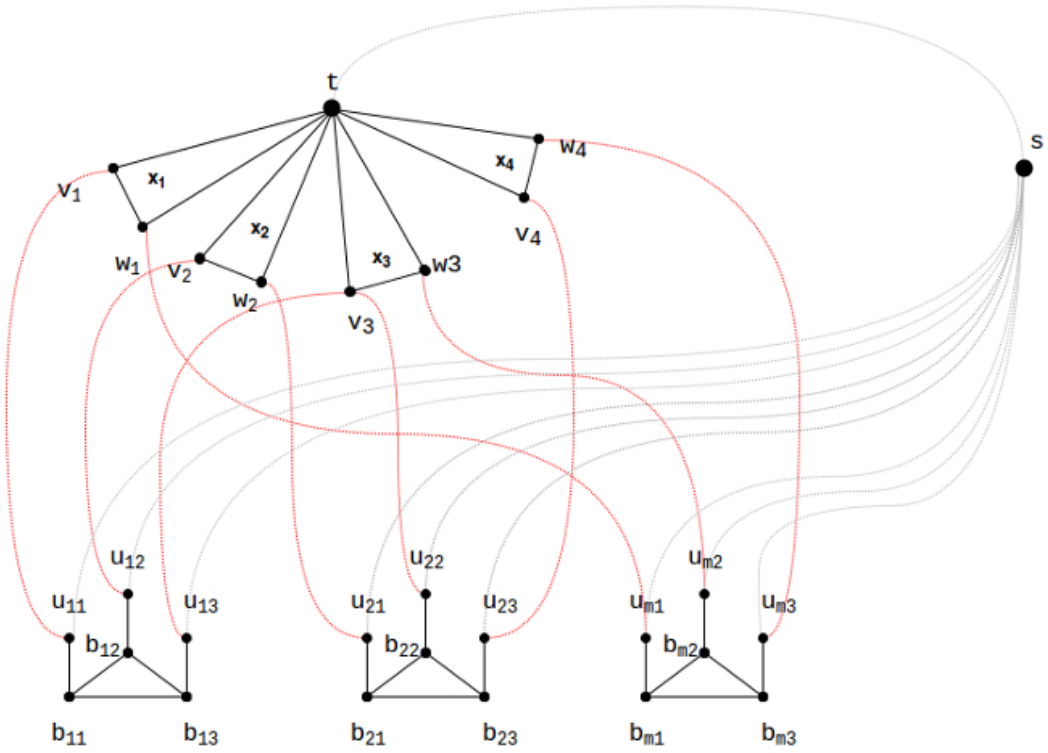
Entonces

$$F = \{u_{ij}\psi(l_{ij}) : i \in \{1, 2, \ldots, m\}, j \in \{1, 2, 3\}\}$$

Dicho en palabras, desde los vértices u_{ij} de cada garra (recordemos que estas representan la disjunción $D_i = l_{i1} \vee l_{i2} \vee l_{i3} : i \in \{1, 2, \ldots, m\}$) salen lados hacia los triángulos (recordemos que estos representan a las variables) que se unen con el vértice v_k (si l_{ij} es la variable x_k) o con el vértice w_k (si l_{ij} es en cambio su negación).

Un ejemplo:

$$B = (x_1 \vee x_2 \vee x_3) \wedge (\overline{x_2} \vee x_3 \vee x_4) \wedge (\overline{x_1} \vee \overline{x_3} \vee \overline{x_4})$$



Ahora debemos probar que $\exists\ \overrightarrow{b} : B(\overrightarrow{b}) = 1 \Leftrightarrow \chi(G) \leq 3$

$\Leftarrow$

Suponemos $\chi(G) \leq 3$ y construiremos un $\overrightarrow{b}$ tal que $B(\overrightarrow{b}) = 1$. Como G contiene triángulos, debe ser $\chi(G) = 3$. Es decir, existe un coloreo $c(G)$ con 3 colores. Definimos $\overrightarrow{b}$:

$$\overrightarrow{b}_k = \begin{cases} 1 & \text{si } c(v_k) = c(s) \\ 0 & \text{si no} \end{cases}$$

Para probar que $B(\overrightarrow{b}) = 1$ debemos probar que $D_i(\overrightarrow{b}) = l_{i1} \vee l_{i2} \vee l_{i3} = 1\ \forall\ \in \{1, 2, \ldots, n\}$. Como $\{b_{i1}, b_{i2}, b_{i3}\}$ forman un triángulo, entonces los 3 colores deben aparecer entre ellos. Es decir, $\exists\, j : c(b_{ij}) = c(t)$.

Ahora bien,

- $\{u_{ij}, b_{ij}\}$ forman lado, por lo tanto, $c(u_{ij}) \neq c(b_{ij})$ y por lo tanto, $c(u_{ij}) \neq c(t)$.
- $\{u_{ij}, s\}$ forman lado, por lo tanto $c(u_{ij}) \neq c(s)$.
- $\{s, t\}$ forman lado, por lo tanto, $c(s) \neq c(t)$

Por estas tres razones, el color de u_{ij} es el **tercer** color. Pero además:

- $\{u_{ij}, \psi(l_{ij})\}$ forman lado, por lo tanto, $c(\psi(l_{ij}))$ $\neq$ el "tercerçolor.
- $\{\psi(l_{ij}), t\}$ forman lado, por lo tanto, $c(\psi(l_{ij})) \neq c(t)$

En consecuencia, $c(\psi(l_{ij})) = c(s)$.

Si $\psi(l_{ij}) = v_k$ (porque $l_{ij} = x_k$) entonces tenemos que $c(v_k) = c(s) \Rightarrow b_k = 1$ (por definición de $\overrightarrow{b}$), y $l_{ij}(\overrightarrow{b}) = 1 \Rightarrow D_i(\overrightarrow{b}) = 1$.

Si, en cambio, $\psi(l_{ij}) = w_k$ (porque $l_{ij} = \overline{x_k}$) entonces, como $\{v_k, w_k\}$ forman lado y por lo tanto $c(v_k) \neq c(w_k)$, debe ser que $c(v_k) \neq c(s) \Rightarrow b_k = 0$. Pero $l_{ij} = \overline{x_k} \Rightarrow l_{ij}(\overrightarrow{b}) = \overline{b_k} = 1$ y $D_i(\overrightarrow{b}) = 1$.

En cualquiera de los dos casos, $D_i(\overrightarrow{b}) = 1$. Como esto ocurre para todo $i \in \{1, 2, \ldots, n\}$, debe ser $B(\overrightarrow{b}) = 1$.

$\Rightarrow$

Asumimos que $\exists\ \overrightarrow{b}\ : B(\overrightarrow{b}) = 1$ y construimos un coloreo propio para G utilizando sólo 3 colores.

Definimos el coloreo:

$$c(s) = \text{blanco}$$
$$c(t) = \text{celeste}$$
$$c(v_k) = \begin{cases} \text{blanco} & \text{si } b_k = 1 \\ \text{negro} & \text{si } b_k = 0 \end{cases}$$
$$c(w_k) = \begin{cases} \text{negro} & \text{si } b_k = 1 \\ \text{blanco} & \text{si } b_k = 0 \end{cases}$$

Oservar que el lado $\{s, t\}$ del grafo no crea problemas. Además, $\{v_k, t\}$ y $\{w_k, t\}$ no crean problemas.

Ahora falta colorear las garras. Veamos primero cómo colorear los u_{ir}. Como $\exists\ \overrightarrow{b}\ : B(\overrightarrow{b}) = 1$, $\forall\ i \in \{1, 2, \ldots, n\} D_i(\overrightarrow{b}) = 1$, es decir, para cada i dado, existe un j tal que $l_{ij}(\overrightarrow{b}) = 1$. Tomamos ese j (si hay más de 1, elijo 1) y definimos:

$$c(u_{ir}) = \begin{cases} \text{negro} & \text{si } r = j \\ \text{celeste} & \text{si } r \neq j \end{cases}$$

Observemos que con este coloreo, el lado $\{u_{ir}, s\}$ no crea problemas. Además:

- para $r \neq j$, tenemos que $\{u_{ir}, \psi(u_{ir})\}$ no crea problemas, ya que $c(u_i r) =$ celeste y $c(\psi(u_{ir})) =$ blanco o negro.
- para $r = j$ tenemos que $c(u_{ir}) =$ negro y para ver que $\{u_{ir}, \psi(u_{ir})\}$ no crea problemas, debemos analizar dos casos:
  - si $l_{ij} = x_k$, entonces $\psi(u_{ir}) = v_k$ y como $l_{ij}(\overrightarrow{b}) = 1$, debe ser $x_k = 1$ y por lo tanto $c(v_k) =$ blanco. En este caso, el lado $\{u_{ir}, \psi(u_{ir})\}$ no crea problemas
  - si $l_{ij} = \overline{x_k}$, entonces $\psi(u_{ir}) = w_k$ y como $l_{ij}(\overrightarrow{b}) = 1$, debe ser $x_k = 0$ y por lo tanto $c(w_k) =$ blanco. En este caso también, el lado $\{u_{ir}, \psi(u_{ir})\}$ no crea problemas.

Finalmente, Sólo resta colorear las bases b_{ir}.

$$c(b_{ir}) = \begin{cases} \text{celeste} & \text{si } r = j \\ \text{negro o blanco} & \\ \text{blanco o negro} & \end{cases}$$

Es decir, le damos el color celeste al b_{ij} y los otros dos los coloreamos uno negro y uno blanco, de modo que

- el triángulo $\{b_{i1}, b_{i2}, b_{i3}\}$ no crea problemas.
- el lado $\{b_{ij}, u_{ij}\}$ no crea problemas, ya que $c(b_{ij}) =$ celeste y $c(u_{ij}) =$ negro
- el lado $\{b_{ir}, u_{ir}\}$ con $r \neq j$, no crea problemas, ya que $c(b_{ir}) =$ blanco o negro y $c(u_{ir}) =$ celeste.