

Complejidad del algoritmo de Dinitz

Daniel Penazzi

5 de mayo de 2021

Tabla de Contenidos

1 Cota superior del número de networks auxiliares

- Enunciado del Teorema
- Primera parte de la prueba
- Primer Caso
- Segundo Caso
- Corolarios

2 Complejidad del paso bloqueante en Dinitz

- Complejidad de Dinitz original
- Pseudocódigo de Dinic-Even
- Complejidad de Dinic-Even

3 Detalles sobre el network auxiliar

Complejidad general de los algoritmos “tipo” Dinic

- En la primera parte veremos como es la complejidad general de los algoritmos que usan la idea de Dinitz de networks auxiliares.
- Luego probaremos las complejidades concretas de la versión original de Dinitz y la versión “occidental” de Dinitz hecha por Even.
- Para probar la complejidad general de los algoritmos de este tipo, debemos acotar el número posible de networks auxiliares.

Propiedad fundamental.

- Recordemos que en el ejemplo vimos que la cantidad de niveles de cada network auxiliar aumentaba entre un network auxiliar y el siguiente. (salgo el último en el cual no se llega a t).
 - Esto pasa siempre:

Teorema

La distancia entre s y t **aumenta** entre networks auxiliares consecutivos.

Es decir, salvo en el último network auxiliar, donde no llegamos a t y la distancia entre s y t es infinita, la cantidad de niveles de un network auxiliar es menor que la cantidad de niveles de cualquier network auxiliar posterior.

Prueba

- Prueba: Sea NA un network auxiliar y NA' el network auxiliar siguiente.
 - Sea f el flujo (del network original) inmediatamente anterior a NA y f' el inmediatamente anterior a NA' .
 - Es decir, NA se construye usando f , y NA' se construye usando f' .
 - Sea $d(x) = d_f(s, x)$ y $d'(x) = d_{f'}(s, x)$.
 - Sabemos que $d(t) \leq d'(t)$ por la prueba de Edmonds-Karp.
 - Queremos probar que vale el $<$ ahí.

Prueba (cont.)

- Si NA' no tiene a t entonces $d'(t) = \infty > d(t)$ y ya está.
 - Asumamos entonces que $t \in NA'$.
 - Entonces existe un camino dirigido $x_0x_1\dots x_{r-1}x_r$ entre s y t en NA'
 - Pero no puede ser un camino en NA :
 - Si lo fuera, al construir f' habriamos saturado ese camino. (es decir, saturar al menos un lado).
 - Pues para terminar con NA y pasar de f a f' debemos saturar todos los caminos de NA .
 - Pero si quedó saturado, no podría ser un camino en NA' .

Prueba (cont.)

- Como $x_0x_1..x_{r-1}x_r$ no es un camino en NA , entonces pasa una de las dos cosas:
 - 1 Algún vértice x_i no está en NA .
 - 2 Están todos los x_i en NA pero falta algún lado $\overrightarrow{x_ix_{i+1}}$.
 - Veamos estos dos casos.
 - Supongamos primero que algún vértice x_i no está en NA .

Primer caso

- Como t esta en NA , entonces $x_i \neq t$.
 - Recordemos que todos los vértices $\neq t$ que esten a distancia mayor o igual que t no se incluyen.
 - Pero todos los que tengan distancia menor a t estan pues construimos NA con BFS.
 - Entonces la única forma en que x_i no este en NA es que $d(t) \leq d(x_i)$ (1).
 - Como probamos en Edmonds-Karp que $d \leq d'$, tenemos:
 - $d(x_i) \leq d'(x_i)$ (2)

Primer Caso

Primer caso

- Ahora bien, como $x_0x_1..x_{r-1}x_r$ es un camino en NA' y NA' es un network por niveles, concluimos que:
- $d'(x_i) = i$ para todo i . (3)
- Ademas vimos que $x_i \neq t$, asi que $i < r$. (4)
- Entonces: $d(t) \stackrel{(1)}{\leq} d(x_i) \stackrel{(2)}{\leq} d'(x_i) \stackrel{(3)}{=} i \stackrel{(4)}{<} r = d'(t)$
- Y hemos probado lo que queriamos en este caso.

Segundo Caso

Segundo caso

- Asumimos ahora que estan todos los x_i en NA pero falta algún lado $\overrightarrow{x_i x_{i+1}}$.
- Y tomamos el primer i para el cual pasa eso.
- Por Edmonds-Karp, sabemos que $d(x_{i+1}) \leq d'(x_{i+1})$.
- Asi que tenemos dos subcasos: que ese \leq sea $<$, o que sea $=$.

Segundo Caso

Subcaso A del segundo caso

- Subcaso A: $d(x_{i+1}) < d'(x_{i+1})$ (5)
- Sea $b(x) = b_f(x, t)$, $b'(x) = b_{f'}(x, t)$.
- Por lo que vimos en Edmonds-Karp, tenemos que $b \leq b'$ y $d(t) = d(x) + b(x)$ para todo x y similar para d' , b' .

$$\begin{aligned}
 d(t) &= d(x_{i+1}) + b(x_{i+1}) \\
 &\leq d(x_{i+1}) + b'(x_{i+1}) \quad (\text{Edmonds - Karp}) \\
 &\stackrel{(5)}{<} d'(x_{i+1}) + b'(x_{i+1}) = d'(t)
 \end{aligned}$$

Hemos probado $d(t) < d'(t)$ en este subcaso.

Segundo Caso

Subcaso B

- Supongamos ahora que $d(x_{i+1}) = d'(x_{i+1}) = i + 1$
- Como i es el primer i para el cual $\overrightarrow{x_i x_{i+1}}$ no está en NA :
- Entonces la porción del camino $x_0 x_1 \dots x_i$ **SI está** en NA .
- Esto implica (al ser NA un network por niveles) que $d(x_i) = i$.
- Entonces, en NA , x_i está en el nivel i , y x_{i+1} en el nivel $i + 1$.
- En particular, concluimos que no solo no está en NA el lado $\overrightarrow{x_i x_{i+1}}$ sino que tampoco está el lado $\overrightarrow{x_{i+1} x_i}$ (pues los niveles no son legales para que ese lado esté).
- Este dato lo usaremos al final de la prueba.

Segundo Caso

Subcaso B

- Como $d(x_i) = i, d(x_{i+1}) = i + 1$, entonces x_i, x_{i+1} estan a distancia “legal” para que pueda existir el lado $\overrightarrow{x_i x_{i+1}}$.
- Pero ese lado no esta en NA . Por lo que concluimos que:
 - 1 $\overrightarrow{x_i x_{i+1}}$ es lado del network original pero esta saturado, o:
 - 2 $\overrightarrow{x_{i+1} x_i}$ es lado del network original pero esta vacio.
- Pero $\overrightarrow{x_i x_{i+1}}$ si es un lado en NA' .
- Asi que la situación es:
 - 1 $\overrightarrow{x_i x_{i+1}}$ es lado del network original, estaba saturado al construir NA pero des-saturado al construir NA' , o
 - 2 $\overrightarrow{x_{i+1} x_i}$ es lado del network original, estaba vacio al construir NA pero no vacio al construir NA' .

Segundo Caso

Subcaso B

- La única forma en que pase [1] es que al pasar de f a f' dessaturamos el lado $\overrightarrow{x_i x_{i+1}}$, es decir, devolvemos flujo, lo que dice que usamos en algún momento el lado como backward.
- Como pasamos de f a f' usando NA, esto significa que en NA debemos haber usado el $\overrightarrow{x_{i+1} x_i}$.
- Esto es un absurdo pues habíamos visto que ese lado no puede estar en NA, pues estariamos yendo del nivel $i + 1$ al i .
- La única forma en que pase [2] es que al pasar de f a f' mandamos algo de flujo por el lado $\overrightarrow{x_{i+1} x_i}$.
- Así que ese lado también debe ser un lado en NA, lo cual como dijimos es un absurdo.
- Fin prueba

Corolarios

Corolario 1

Corolario

En cualquier corrida de un algoritmo “tipo” Dinic, hay a lo sumo n networks auxiliares.

- Prueba
- La distancia entre s y t aumenta en al menos 1 por cada network auxiliar.
- Salvo por el último network auxiliar, donde la distancia es infinita, en los demás la distancia debe ser un número natural entre 1 y $n - 1$
- Por lo tanto hay a lo sumo n networks auxiliares.

Corolarios

Corolario 2

Corolario

Sea CFB la complejidad de hallar un flujo bloqueante en un network auxiliar con un algoritmo “tipo” Dinic. La complejidad del algoritmo es entonces $O(n * (m + CFB))$.

- Prueba:
- La complejidad es: (número de networks auxiliares)*(complejidad de crear un network auxiliar+complejidad de hallar un flujo bloqueante en ese network auxiliar).
- Por el corolario 1, número de networks auxiliares $\leq n$.
- La complejidad de crear un network auxiliar es $O(m)$, pues lo hacemos usando BFS.
- Entonces tenemos $n.(O(m) + CFB) = O(n * (m + CFB))$.
Fin prueba

Corolarios

Observación

- Observemos que CFB parecería que no puede ser menor que $O(m)$, pues deberíamos revisar todos los lados para saber si el flujo es bloqueante o no.
- En cuyo caso, la complejidad quedaría $O(n * (m + CFB)) = O(n * CFB)$.
- En todos los algoritmos que veamos pasará eso.
- Pero por las dudas alguien alguna vez descubra una forma mas eficiente de crear un flujo bloqueante que $O(m)$, escribimos el teorema de esta forma.
- Con este teorema general de la complejidad de algoritmos “tipo Dinic” podemos probar la complejidad del algoritmo de Dinitz:

Teorema de complejidad de Dinitz

Teorema

La complejidad de los algoritmos de Dinitz original y de la versión de Even es $O(mn^2)$.

- Antes de dar la prueba, observemos que si el network es ralo, es decir $m \sim n$ entonces tanto Edmonds-Karp como Dinic tienen complejidad $O(n^3)$.
- Pero si el network es denso, es decir, $m \sim n^2$, entonces Edmonds-Karp es $O(nm^2) = O(n^5)$ mientras que Dinitz o Dinic-Even son $O(mn^2) = O(n^4)$.
- Es decir, para networks densos, Dinitz o Dinic-Even son n veces mas rápidos que Edmonds-Karp.

Teorema de complejidad de Dinitz

- Si probamos que $CFB =$ la complejidad de hallar un flujo bloqueante en un network auxiliar es $O(mn)$ tanto en Dinitz como en la versión de Even, entonces por el teorema general tendremos que la complejidad es:

$$O(n*(m+CFB)) = O(n*(m+O(mn))) = O(n*O(mn)) = O(mn^2)$$

- Vamos a dividir la prueba en la versión original de Dinitz y la versión de Even.
- Empezaremos con la versión original de Dinitz.

Complejidad de Dinitz original

Complejidad versión original de Dinitz

- Recordemos que en la versión original de Dinitz, cada camino entre s y t se encuentra usando DFS, pero el network auxiliar tiene la propiedad extra que se garantiza que toda búsqueda DFS nunca va a tener que hacer backtracking, pues cada vértice con lado entrante tiene un lado saliente.
- Pero esta propiedad tiene el costo de tener que mantenerla entre camino y camino, revisando el network auxiliar.
- A esta operación Dinitz la llama “PODAR”
- Entonces no sólo tenemos que calcular la complejidad de encontrar todos los caminos, sino tambien la complejidad de hacer todos los “PODAR”

Complejidad de Dinitz original

Complejidad versión original de Dinitz

- Veamos primero la parte de la complejidad de encontrar todos los caminos.
- Construir un camino dirigido desde s a t es muy fácil:
 - Tomar $p = s$.
 - WHILE($p \neq t$)
 - Tomar q cualquier vecino de p . (*)
 - Agregar \overrightarrow{pq} al camino.
 - Tomar $p = q$
 - ENDWHILE
- (*) siempre hay un q por la propiedad que dijimos antes.

Complejidad de Dinitz original

Complejidad de encontrar todos los caminos

- Entonces la construcción de cada camino es $O(r)$, donde r es el número de niveles.
- Como $r < n$ podemos decir que esta parte es $O(n)$.
- Luego de cada camino, se borran del network auxiliar los lados saturados, pero esto es simplemente recorrer una vez mas el camino, asi que es otro $O(n)$.
- Como cada camino satura y por lo tanto borra al menos un lado, hay a lo sumo m caminos.
- Asi que el total de la complejidad de encontrar todos los caminos es $O(mn)$.

Complejidad de Dinitz original

Complejidad de PODAR

- Como cada PODAR viene luego de un camino (mas uno extra, al principio de todo, luego de la construcción del network) sabemos que hay a lo sumo $m + 1$ de ellos.
- Podriamos luego calcular la complejidad de cada PODAR y calcular la complejidad total como $O(m)$ veces la complejidad de cada PODAR.
- Pero este análisis **no es buena idea** porque la complejidad de cada PODAR es bastante variable, y si tomaramos siempre la complejidad del peor caso posible, nos daria una cota global igual o peor que la de Edmonds-Karp, dependiendo como la analizaramos.

Complejidad de Dinitz original

Clave para analizar los Ps

- Esa fue tambien una genialidad de Dinitz, darse cuenta que si bien la PEOR complejidad de los PODAR es alta, la complejidad **promedio** es menor, y por lo tanto al recorrer todos los PODAR obtenemos una complejidad mejor que la de Edmonds-Karp.
- ¿Cómo es, exactamente, PODAR?
- PODAR va recorriendo todos los vértices, desde niveles mas altos a mas bajos, chequeando si tienen lados de salida, y borrandolos si no tienen lados de salida.
- Ahi ya hay un problema, porque puede ser que no tenga que borrar ningún vértice, algunos o muchos.
- Esto en realidad lo podemos solucionar viendo que a cada vértice sólo se lo puede borrar una vez.

Complejidad de Dinitz original

Clave para analizar los Ps

- El problema es que ademas ese “borrar”el vértice implica borrar todos los lados de entrada al vértice.
- Y podria ser que haya que borrar uno o incluso ningún lado, o tener que borrar casi todos, asi que en el peor caso es $O(m)$ por cada vértice, y es lo que complica el análisis.
- La clave está en no contar la complejidad de cada PODAR y cuantos hay, sino la complejidad de TODOS los PODAR en CONJUNTO.

Complejidad de Dinitz original

PVs

- Para analizar la complejidad, llamemos PV a la parte de PODAR de simplemente recorrer los vértices mirando si tienen lados de salida o no.
- Y llamemos $B(x)$ a borrar todos los lados de entrada de un vértice x .
- Entonces PODAR es hacer PV, deteniéndonos en los vértices x para los cuales no tienen lados de salida, y activando $B(x)$ para esos.

Complejidad de Dinitz original

PVs

- La complejidad de cada PV es $O(n)$, pues chequea todos los vértices.
- Hay un PV en cada PODAR, así que hay a lo sumo $m + 1$ PV en total.
- La complejidad total de **todos** ellos es entonces $O(nm)$.
- Para los $B(x)$ es donde debemos hacer un análisis mas refinado pues no sabemos cuantos $B(x)$ s se hacen en un PV.

Complejidad de Dinitz original

Análisis de los $B(x)$

- La complejidad de un $B(x)$ es $O(d(x))$. (en realidad, del grado de entrada de x).

Complejidad de Dinitz original

Análisis de los $B(x)$

- La complejidad de un $B(x)$ es $O(d(x))$. (en realidad, del grado de entrada de x).
En realidad, esto depende de cómo sea la estructura que se use para guardar los lados.
Se podría programar $B(x)$ para que fuese $O(1)$, o tambien, podria demorar $O(n)$ o incluso $O(m)$. Dejamos esos casos como ejercicio.

Complejidad de Dinitz original

Análisis de los $B(x)$

- La complejidad de un $B(x)$ es $O(d(x))$. (en realidad, del grado de entrada de x).
- Tambien observemos que si se “activa” la llamada a $B(x)$, no se vuelve a hacer $B(x)$ para ese vértice x nunca mas, porque le borramos todos los lados y ademas borramos el vértice.
- Entonces, independientemente de cuantos $B(x)$ s hay en un PV, lo que sabemos es que al final de todo, habrá a lo sumo un $B(x)$ por cada vértice x .
- Como la complejidad de $B(x)$ es $O(d(x))$, entonces la complejidad del conjunto de $B(x)$ s es $O(\sum_x d(x)) = O(m)$. (lema del apretón de manos:
 $(\sum_x d(x) = 2m)$

Complejidad de Dinitz original

Complejidad de Dinitz original, parte final

■ Resumiendo

- 1 La complejidad total de buscar todos los caminos y borrar los lados saturados es $O(nm)$.
 - 2 La de los PVs es $O(nm)$.
 - 3 La de todos los $B(x)$ s es $O(m)$.
- Entonces la complejidad total del paso bloqueante es $O(nm) + O(nm) + O(m) = O(nm)$
 - Y la del algoritmo completo, $O(mn^2)$, como vimos.
 - Fin complejidad Dinitz original.

Pseudocódigo de Dinic-Even

Dinic-Ever

- La versión de Ever no tienen los PODAR, asi que por un lado “ganamos” en facilidad de análisis.
- Pero por otro lado, como ahora DFS puede tener que hacer backtracks, ya no sabemos que cada DFS sea $O(n)$ y el analisis se complica por ahi.
- De hecho, no vamos a poder simplemente calcular la complejidad de cada DFS y luego multiplicar por m porque eso nos daria $O(m^2)$.
- Asi que vamos a tener que ser mas cuidadosos.
- Para poder analizar la complejidad de la versión de Ever nos va a convenir dar un pseudocódigo.

Pseudocódigo de Dinic-Even

Dinic-Ever

- El pseudocódigo es sólo para la parte de encontrar un flujo bloqueante en el network auxiliar.
- En el pseudocódigo $\Gamma^+(x)$ es el $\Gamma^+(x)$ DEL NETWORK AUXILIAR, no del network original
- Y “borrar lados” tambien se refiere al network auxiliar.
- Recordemos que lo que hace Ever es un DFS normal, excepto que cuando se ve forzado a hacer un backtrack “guarda” la información de que debió hacer un backtrack ahí.
- La forma de hacerlo es borrar o bien el lado por el cual se hace backtrack o bien directamente el vértice desde el cual se hace backtrack.
- Aca veremos la primera opción y dejamos la 2da como ejercicio.

Pseudocódigo de Dinic-Even

A,R,I

- Para poder escribir el pseudocódigo en una página y luego analizar la complejidad, llamaremos a partes del código de la siguiente forma:
 - AVANZAR(x): Elegimos algún vecino y de $\Gamma^+(x)$, agregamos \overrightarrow{xy} al camino y cambiamos $x = y$.
 - Nota: Sólo ejecutaremos AVANZAR si $\Gamma^+(x)$ no es vacío.
 - RETROCEDER(x): Tomamos z el vértice anterior a x en la pila, borramos \overrightarrow{zx} del camino y del network auxiliar, y hacemos $x = z$.
 - Sólo ejecutaremos RETROCEDER si efectivamente podemos retroceder, es decir, si $x \neq s$.

Pseudocódigo de Dinic-Even

A,R,I

- Para poder escribir el pseudocódigo en una página y luego analizar la complejidad, llamaremos a partes del código de la siguiente forma:
 - AVANZAR(x): Elegimos algún vecino y de $\Gamma^+(x)$, agregamos \overrightarrow{xy} al camino y cambiamos $x = y$.
 - Nota: Sólo ejecutaremos AVANZAR si $\Gamma^+(x)$ no es vacío.
 - RETROCEDER(x): Tomamos z el vértice anterior a x en la pila, borramos \overrightarrow{zx} del camino y del network auxiliar, y hacemos $x = z$.
 - Sólo ejecutaremos RETROCEDER si efectivamente podemos retroceder, es decir, si $x \neq s$.
Como dijimos, analizaremos esta versión, que sólo borra el lado \overrightarrow{zx} . Otra posibilidad es borrar x , es decir, ejecutar un $B(x)$ y borrar todos los lados de entrada a x en vez de sólo \overrightarrow{zx} . Dejamos como ejercicio analizar esa versión.

- Para poder escribir el pseudocódigo en una página y luego analizar la complejidad, llamaremos a partes del código de la siguiente forma:
 - AVANZAR(x): Elegimos algún vecino y de $\Gamma^+(x)$, agregamos \overrightarrow{xy} al camino y cambiamos $x = y$.
 - Nota: Sólo ejecutaremos AVANZAR si $\Gamma^+(x)$ no es vacío.
 - RETROCEDER(x): Tomamos z el vértice anterior a x en la pila, borramos \overrightarrow{zx} del camino y del network auxiliar, y hacemos $x = z$.
 - Sólo ejecutaremos RETROCEDER si efectivamente podemos retroceder, es decir, si $x \neq s$.
 - INCREMENTAR Una vez construido el camino, calculamos cuánto se puede mandar por él, mandamos eso y borramos del network cualquier lado que haya sido saturado.

Pseudocódigo de Dinic-Even

Dinic-Even para hallar flujo bloqueante g

- $g = 0$
- STOPFLAG:=1//para saber cuando parar
- WHILE (STOPFLAG) //while externo
 - | $p = [s], x = s$ // Inicialización inicial de x y del camino p
 - | WHILE (($x \neq t$) AND (STOPFLAG))//while interno
 - | | IF $\Gamma^+(x) \neq \emptyset$ THEN AVANZAR(x)
 - | | ELSE IF ($x \neq s$) THEN RETROCEDER(x)
 - | | ELSE STOPFLAG=0
 - | IF ($x == t$) THEN INCREMENTAR
- RETURN(g)

Complejidad de Dinic-Even

- Una vez que se tiene creado el camino para aumentar el flujo, el INCREMENTAR es $O(n)$.
- pues la longitud del camino es a lo sumo n , y INCREMENTAR aumenta el flujo a lo largo de ese camino y borrar los lados saturados.
- Ademas, la complejidad de AVANZAR es $O(1)$, pues consiste en buscar al primer vértice en la lista de $\Gamma^+(x)$, agregar un lado al camino, y cambiar x .
- La complejidad de RETROCEDER tambien es $O(1)$ pues simplemente borramos un lado y cambiamos quien es x .

Complejidad de Dinic-Even

A,R,Is

- Si denotamos AVANZAR por A, RETROCEDER por R e INCREMENTAR+ inicializar por I, entonces vemos que Dinic-Even es una sucesión de As,Rs,Is.
- Pej, AAAAAIAARAARRAARARAI...
- Podemos dividir esta sucesión de letras en “palabra” de la forma AAA....AAX
- Donde X es R o I.
- Las preguntas que debemos responder son:
 - 1 ¿Cuantas palabras hay?
 - 2 ¿Cuál es la “complejidad” de cada palabra?

¿Cuantas palabras hay?

- Cada palabra termina en un $X=(R \text{ o } I)$.
- R borra el lado por el cual retrocede.
- I manda flujo por un camino en el cual se satura al menos un lado, y borra todos los lados saturados.
- Concluimos que X , sea R o I , borra al menos un lado.
- Por lo tanto la cantidad de palabras es a lo sumo m .

Complejidad de Dinic-Even

Complejidad de cada palabra

- Vimos que R y A son $O(1)$.
- I es INCREMENTAR, que es $O(n)$ mas un inicializar, que es $O(1)$, asi que I es $O(n)$.
- Asi que si una palabra A....AX tiene k As, la complejidad de la palabra será $O(k + 1) = O(k)$ si X es R y $O(k + n)$ si X es I.
- Pero A mueve el extremo del camino donde estamos parados un nivel para adelante.
- Como hay r niveles, entonces $k \leq r$.
- Como puede haber a lo sumo n niveles, entonces $r \leq n$.
- Por lo tanto la complejidad de A...AX es $O(n)$ si X=R y $O(n + n) = O(n)$ si X=I.

Complejidad de Dinic-Even

Complejidad Total

- Es decir, $O(n)$ en cualquiera de los dos casos.
- Entonces hemos probado que:
 - 1 Hay a lo sumo m palabras.
 - 2 La complejidad de cada palabra es $O(n)$
- Por lo tanto la complejidad total del paso de encontrar un flujo bloqueante es (número de palabras)*(complejidad de c/palabra)= $O(mn)$
- Y por lo tanto, como vimos al principio de la prueba, la complejidad total de Dinic-Even es $O(n) * O(mn) = O(mn^2)$.

Construyendo un network auxiliar

- La idea original de Dinitz, como dije, era construir el network auxiliar.
- ¿Cómo hacemos esto?
- En principio habria que crear una estructura nueva, hacer copias de los vértices, hacer copia de algunos lados, y crear otros lados que no existen en el network original. (los lados “backward”).
- O bien para no tener que hacer todo eso se puede “crear”el network auxiliar dentro del network original.

Construyendo un network auxiliar

- Como en el network auxiliar nunca se agregan vértices nuevos, todo lo que es necesario hacer es indicar cuales son los lados del network auxiliar, que es equivalente a indicar para cada vértice quienes son sus vecinos en el network auxiliar.
- Es decir, en la estructura que representa el network original seguramente tendremos un array o algún otro tipo de estructura que nos indique para cada vértice x quienes son los vértices en $\Gamma^+(x)$ y otro array que indique quienes son los vértices en $\Gamma^-(x)$.
- Ademas de esos arrays podemos tener simplemente un par de arrays extras que indiquen quienes seran los vértices en los $\Gamma^+(x)$ y $\Gamma^-(x)$ **del network auxiliar**.

Otra forma de representar un network auxiliar

- Así que construir el network auxiliar sería simplemente llenar esos arrays.
- Incluso hay una alternativa más simple, que es quizás la más usada en la actualidad.
- Consiste en simplemente calcular los números de nivel para cada vértice, corriendo BFS.
- Y luego, al correr el algoritmo, se corre Ford-Fulkerson con DFS sobre el network original, pero “mirando” sólo lados que unan vértices de un nivel con vértices del nivel siguiente.

Otra forma de representar un network auxiliar

- Los vértices se “borran” del network auxiliar simplemente cambiandole el número de nivel al vértice a “infinito” (es decir, un número mas grande que cualquier posible nivel, pej, $n + 2$).
- Cuando no se pueden construir mas caminos entre s y t con esas restricciones, se recalculan los números de nivel, corriendo otra vez BFS.
- Esto es mucho eficiente en la computadora, pero mas engorroso de explicar y hacer a mano.
- Algunos detalles del calculo de complejidad cambian, los dejamos como ejercicio.