Algoritmo Wave de Tarjan

Who?

Daniel Penazzi

When?

5 de mayo de 2021

Tabla de Contenidos

- Vimos que Dinic encuentra flujos maximales por medio de los networks auxiliares:
  - construir un network auxiliar, encontrar un flujo bloqueante en el, cambiar el flujo mediantes ese flujo bloqueante, repetir hasta que el flujo quede maximal.
- Y mencionamos que muchos algoritmos luego de Dinic usaron esa estrategia, cambiando la forma de encontrar flujos bloqueantes.
- El primero que hizo eso fue Karzanov pero su algoritmo era complicado y otros fueron simplificando sus ideas.
- Para ejemplificar esta estrategia de una forma distinta daremos el algoritmo Wave de Tarjan y mencionaremos brevemente el algoritmo MKM.

MKM

- En Dinitz, Edmonds-Karp, Ford-Fulkerson construiamos nuestro flujo bloqueante o maximal, según el caso, de a un camino aumentante por vez.
- Pero ¿que tal si pudieramos construir varios caminos aumentantes en paralelo?
- Un algoritmo anterior a Wave, llamado MKM por las iniciales de sus autores, hace esto, mediante dos operaciones “push” y “pull”
- Una operación de push en un vértice lo que hace es empujar cierta cantidad de flujo desde un vértice x hasta TODOS sus vécinos de $\Gamma^+(x)$.
- Pull envia flujo DESDE todos los vécinos de $\Gamma^-(x)$ hacia x.

MKM

- La cantidad de flujo enviada desde x hacia $\Gamma^+(x)$ y desde $\Gamma^-(x)$ hacia x son iguales, para que x quede con la propiedad $in(x) = out(x)$.
- Pero esta operación doble push-pull en x va a “desbalancear ”los vértices de $\Gamma^+(x)$ pues tendrán $in > out$ y a los de $\Gamma^-(x)$ pues tendrán $in < out$.
- Entonces lo que se hace es, a partir de los vértices de $\Gamma^+(x)$, se los va “balanceando” haciendo sólo push en ellos.
  - Esto desbalancearia a sus vecinos de $\Gamma^+(x)$, con in>out, asi que aplicamos push a ellos, etc, hasta llegar a t.
- Y dualmente a los vértices de $\Gamma^-(x)$ se les aplica solamente operacion de pull para balancearlos, lo cual va a desbalancear a sus vecinos con $out > in$, asi que se les aplica pull a ellos, etc, hasta llegar a s.

MKM

- Lo que queda sigue siendo flujo, y esto es equivalente a haber construido muchos posibles caminos aumentantes, todos pasando por x.
- Seguimos haciendo esto hasta obtener un flujo bloqueante.
- Pero ¿cual vértice x inicial elegir y cuanto flujo mandar por el?
- Cuanto flujo, es simplemente lo máximo que podamos mandar, que será el minimo de (lo máximo que podamos mandar hacia $\Gamma^+(x)$ con pull y lo máximo que podamos mandar desde $\Gamma^-(x)$ a x)
- porque queremos que x quede balanceado de entrada.

MKM

- Para cual x elegir, observemos que queremos que la serie de pushes desde x pueda llegar a t, y la serie de pulls desde x hacia s lleguen a s, es decir, no se “traben’‘en el medio.
- Pej, si x puede mandar 5 unidades de flujo a cada uno de y, z pero y solo puede mandar 2 unidades de flujo a sus vecinos u, v, entonces no podria balancear y con pulls y tendria un problema.
- MKM revisa todos los vértices para ver cual es el vértices que menos “capacidad” tiene de enviar flujo tanto hacia adelante como hacia atras.

MKM

- Y entonces mandan flujo desde ese vertice, porque de esa forma sabe que todos los demas tendrán esa capacidad de pull o push, o menos.
- Si bien se puede probar que MKM tiene la misma complejidad del algoritmo Wave, en la practica se demora mas porque por construcción usa el x por el cual puede mandar la menor cantidad de flujo.
- Por eso Tarjan lo modificó, para tratar de mandar mas flujo.
- La modificación de Tarjan no usa pulls (aunque si una variación de pull) y tampoco usa flujos sino "preflujos"

Wave

- En Dinitz el invariante de las funciones parciales g obtenidas durante la construcción del flujo bloqueante en un network auxiliar era “ser flujo” y nos deteniamos cuando era bloqueante.
- MKM si bien durante un ciclo push-pull lo que se tiene no es flujo, al termino de los mismos si, y la estrategia es la misma: mantener “flujosidad”y parar cuando es bloqueante.
- En Wave el invariante es que “g sea bloqueante” y nos detenemos cuando g sea flujo.

- El primer paso, de incrementar el valor de g inmediatamente bloqueandola, se hace simplemente mandando desde s a cada uno de sus vecinos todo lo que se pueda por los lados correspondientes.
- Es decir, es como si en MKM eligieramos $x = s$ sin importarnos si podemos llegar a t con los push o no.
- Esto claramente “bloquea" g, pero tambien claramente lo que queda no es flujo.
- Todos los vecinos de s, es decir, los vértices del nivel 1, ahora tenemos un montón de flujo “entrante" y nada saliente. Estan “desbalanceados".

Pseudocódigo

- Podemos dar el pseudocódigo de esta primera parte.
- Al igual que en Dinic, cuando ponemos $\Gamma^+(x)$, nos referimos al Γ^+ del network auxiliar, no al del network original, y c es la capacidad en el network auxiliar, no en el original.
- Tambien necesitaremos registrar no quien es $\Gamma^-(x)$ pero si un conjunto $M(x)$ que indique quienes son los vértices que le han Mandado flujo a x.

Wave: (pseudocódigo de la parte en que encuentra flujo bloqueante: Inicialización

- $g = 0$
- FORALL x
- $D(x) = 0$//D(x) es $in(x) - out(x)$, inicialmente cero.
- Aca va algo extra que explicamos luego
- ENDFOR
- FORALL $x \in \Gamma^+(s)$
- $g(\overrightarrow{sx}) = c(\overrightarrow{sx})$//mandamos todo lo que podemos
- $D(x) += c(\overrightarrow{sx})$, $D(s) -= c(\overrightarrow{sx})$

Wave: (pseudocódigo de la parte en que encuentra flujo bloqueante: Inicialización

- $g = 0$
- FORALL x
- $D(x) = 0$//D(x) es $in(x) - out(x)$, inicialmente cero.
- Aca va algo extra que explicamos luego
- ENDFOR
- FORALL $x \in \Gamma^+(s)$
- $g(\overrightarrow{sx}) = c(\overrightarrow{sx})$//mandamos todo lo que podemos
- $D(x) += c(\overrightarrow{sx})$, $D(s) -= c(\overrightarrow{sx})$
- Queremos dejar registrado quienes son los vértices que efectivamente le mandan flujo a un vértice

Wave: (pseudocódigo de la parte en que encuentra flujo bloqueante: Inicialización

- $g = 0$
- FORALL x
- $D(x) = 0$//D(x) es $in(x) - out(x)$, inicialmente cero.
- En realidad vamos a necesitar un booleano mas adelante, que inicializamos aca
- ENDFOR
- FORALL $x \in \Gamma^+(s)$
- $g(\overrightarrow{sx}) = c(\overrightarrow{sx})$//mandamos todo lo que podemos
- $D(x) + = c(\overrightarrow{sx})$, $D(s) - = c(\overrightarrow{sx})$
- $M(x) = \{s\}$//s le manda flujo a x.
- ENDFOR//termina la inicialización

Wave: (pseudocódigo de la parte en que encuentra flujo bloqueante: Inicialización

- $g = 0$
- FORALL x
- $D(x) = 0$//D(x) es $in(x) - out(x)$, inicialmente cero.
- $B(x) = 0$
- ENDFOR
- FORALL $x \in \Gamma^+(s)$
- $g(\overrightarrow{sx}) = c(\overrightarrow{sx})$//mandamos todo lo que podemos
- $D(x) += c(\overrightarrow{sx})$, $D(s) -= c(\overrightarrow{sx})$
- $M(x) = \{s\}$//s le manda flujo a x.
- ENDFOR//termina la inicialización

- ¿Que hacemos con todos esos vértices que quedaron desbalanceados?
- Intentamos balancearlos, haciendo push en cada uno de ellos, y luego con sus vecinos, etc, hasta llegar a t.
- Sólo que Tarjan, al menos en el algoritmo original suyo (luego hubo otras versiones) no le llama push sino “balanceo hacia adelante” (forward balance).
- Pero podria suceder que NO PODAMOS balancear un vértice, pues elegimos s para empezar todo, no el vértice de menor “capacidad”
- Pero entonces ¿Que hacemos en el caso de que se llegue a un vértice que no puede ser balanceado porque no puede mandar mas flujo hacia adelante?

- Para empezar, los “marcamos" para indicar que esos vértices no pueden mandar mas flujo hacia adelante y que hay que resolver la situación.
- A los vértices marcados de esta forma especial Tarjan los llama “bloqueados".
- Y ese es el rol del $B(x)$ que pusimos antes en la inicialización.
- Entonces, para empezar, el ForwardBalance de Tarjan, que abreviaremos FB, seria asi:

FB

- $FB(x)$:
- WHILE ($(D(x) > 0)$ AND $(\Gamma^+(x) \neq \emptyset)$)
- Tomar $y \in \Gamma^+(x)$
- IF $(1 == B(y))$ THEN $\Gamma^+(x) = \Gamma^+(x) - y$
- //si y esta bloqueado no tiene sentido mandarle incluso mas flujo.
- ELSE $A = Min\{D(x), c(\overrightarrow{xy}) - g(\overrightarrow{xy})\}$
- $g(\overrightarrow{xy}) += A$
- $D(x) -= A$, $D(y) += A$
- $M(y) = M(y) \cup \{x\}$
- IF $(g(\overrightarrow{xy}) == c(\overrightarrow{xy}))$ THEN $\Gamma^+(x) = \Gamma^+(x) - y$
- ENDWHILE
- IF $(D(x) > 0)$ THEN $B(x) = 1$

- Bueno, pero ¿qué hacemos con esos vertices bloqueados y desbalanceados?
- Obviamente, lo único que queda por hacer es devolver flujo a uno o mas de los vértices que le mandaron flujo.
- Por eso necesitamos el conjunto $M(x)$ del cual hablabamos antes.
- Esto es lo que Tarjan llama “backward balance” (balanceo hacia atrás), que abreviaremos *BB*.
- BackwardBalance siempre podrá balancear un vértice, pues en el peor de los casos, simplemente devuelve todo el flujo que le hayan mandado.
- Su pseudocódigo seria el siguiente:

BB

- $BB(x)$:
- WHILE $(D(x) > 0)$://el balanceo hacia atras siempre tiene exito
- Tomar $y \in M(x)$//mientras $D(x) > 0$, existirá alguien en $M(x)$.
- $A = Min\{D(x), g(\overrightarrow{yx})\}$//lo máximo que x le puede devolver a y
- $g(\overrightarrow{yx}) -= A$ // restamos A, es decir, devolvemos flujo
- $D(x) -= A$, $D(y) += A$.
- IF $(0 == g(\overrightarrow{yx}))$ THEN $M(x) = M(x) - \{y\}$
- ENDWHILE

- Si es un vértice de nivel 1, le estaria devolviendo flujo a s y no hay problema.
- Pero ¿que pasa si no es un vértice del nivel 1?
- Al devolver las unidades de flujo a uno o mas vértices, esos vértices, que estaban balanceados, quedarán desbalanceados otra vez.
- Podriamos decir “bueno, esos vértices devuelven flujo a sus vecinos de $\Gamma^{-}(x)$, etc, hasta llegar a s.”
- Pero eso es un ERROR, porque “rompe”la propiedad de ser bloqueante.

- Lo que hay que hacer con esos vertices es ver si no pueden redirigir las unidades que les fueron devueltas a otros vertices hacia adelante.
- Si puede, se los manda.
- Solamente si no puede mandar mas unidades de flujo hacia adelante, entonces y sólo entonces devolveria flujo.
- Es decir, devolvemos flujo desde un vértice sólo si sabemos con certeza que no puede mandar flujo hacia adelante.
- Es decir, si está bloqueado.
- $BB(x)$ se ejecuta entonces sólo si $B(x) = 1$.

- Ahora bien, al devolver flujo desde un vértice bloqueado a uno no bloqueado, puede suceder que el no bloqueado quede desbalanceado, es decir, le entre mas flujo del que salga.
- Como dijimos, en ese caso NO lo balanceamos pues sólo balanceamos vértices bloqueados.
- Y entonces ¿qué pasa con los desbalanceados no bloqueados?
- Una posibilidad sería inmediatemente tratar de hacer un $FB(x)$ y si no lo balancea, bloquearlo y hacer un $BB(x)$.
- Pero ignoro cual seria la complejidad de esa implementación, y no es lo que hace Tarjan.

- Para poder implementar estas ideas correctamente, Tarjan diseña el algoritmo Wave para operar, como su nombre en inglés lo indica, en “olas".
- Primero se manda una ola hacia adelante, partiendo de s, que lleva todo lo que se puede llevar desde s, como dijimos, haciendo *FB*s.
- Esta “ola hacia adelante" va llevando todo lo que se pueda desde cada vértice hacia adelante, hasta llegar a t.
- Luego se usa una ola que devuelva flujo: se lanza una ola retrograda desde t, que va devolviendo flujo desde los vertices hacia sus vecinos hacia atras.
- Como explicamos antes, sólo devolveremos desde un vértice si sabemos con seguridad que el vértice no puede mandar mas flujo hacia adelante, es decir si esta bloqueado

- Los vertices desbalanceados pero no bloqueados simplemente esperan a la SIGUIENTE ola hacia adelante para ver si los puede balancear o no.
- En resumen:
  1. La ola hacia atras recorre los vértices desde t hacia s, devolviendo flujo desde los vértices bloqueados solamente, balanceandolos.
  2. La ola hacia adelante recorre los vértices desde s a t, mandando flujo hacia los vértices del siguiente nivel, balanceando los vértices que pueda.
     - La ola hacia adelante puede no ser capaz de balancear un vértice.
     - en ese caso lo bloquea.

- Por lo tanto Wave consistirá en una serie de ciclos ola hacia adelante-ola hacia atras, donde las olas hacia adelante tratan de balancear vértices enviando flujo hacia niveles superiores y bloqueando los vértices que no pueden balancear, mientras las olas hacia atras devolverán flujo hacia niveles inferiores, balanceando todos los vértices desbalanceados y bloqueados que encuentre.
- Veremos que la cantidad de ciclos es finita asi que la serie de olas eventualmente termina.

Pseudocódigo

- Ahora podemos completar el pseudocódigo del paso bloqueante de Wave.
- Como estaremos recorriendo los vértices de los niveles en el orden de los niveles, y los niveles se construyen con BFS, en realidad estamos recorriendo los vértices en el orden BFS cuando hacemos la ola hacia adelante, y en orden BFS inverso cuando hacemos la ola hacia atras.
- Para indicar esto diremos que x va desde "$s + 1$", entendiendo como tal el primer vértice en el orden BFS posterior a s, hasta "$t - 1$", entendiendo como tal el último vértice en el orden BFS antes de t.

Pseudocódigo

- Un detalle: ¿cuando paramos?
- Paramos cuando lo que tengamos sea flujo, es decir, cuando $D(x) = 0 \forall x \neq s, t$.
- Pero si en cada ciclo ola hacia adelante/ola hacia atras tenemos que hacer un chequeo sobre todos los vértices, agregariamos una demora extra.
- Observemos que cuando x le manda A unidades de flujo a z, $D(x)$ disminuye en A y $D(z)$ aumenta en A.
- Asi que la suma de todos los $D(x)$, sobre todos los vértices, siempre es cero.
- Como $D(s)$ es el único vértice con desbalanceo negativo, (y queda asi desde la inicialización) entonces $D(x) = 0 \forall x \neq s, t$ si y solo si $D(s) + D(t) = 0$.

Wave: (pseudocódigo de las olas)

- WHILE $(D(s) + D(t) \neq 0)$
- FOR $x =$"$s + 1$" TO "$t - 1$": //BFS orden. (Ola hacia adelante)
- IF $(D(x) > 0)$ THEN FB(x)
- ENDFOR//termina ola hacia adelante
- FOR $x =$"$t - 1$" TO "$s + 1$": //BFS inverso. (Ola hacia atras)
- IF $((1 == B(x))$ AND $(D(x) > 0)$) THEN BB(x)
- ENDFOR//fin ola hacia atras.
- ENDWHILE
- RETURN(g)

- TEOREMA: La complejidad de Wave es $O(n^3)$.
- Prueba:
- Como Wave es un algoritmo que usa la técnica de los networks auxiliares y vimos que puede haber a lo sumo n networks auxiliares antes de encontrar un flujo maximal, vemos que la complejidad de Wave es n veces la complejidad de encontrar un flujo bloqueante en un network auxiliar.
- Asi que basta con demostrar que la complejidad de encontrar un flujo bloqueante en un network auxiliar es $O(n^2)$.

- Para encontrar un flujo bloqueante, Wave hace una serie de ciclos de olas hacia adelante-olas hacia atras.
- En cada ola hacia adelante hacemos una serie de $FB(x)$ y en cada ola hacia atras una serie de $BB(x)$.
- En cada uno de esos, revisamos una serie de ys, ya sea en $\Gamma^+(x)$ o en $M(x)$.
- Para cada uno de esos y que miramos, tardamos O(1) (pues hay que calcular cuanto mandar/devolver, restarlo de $D(x)$, sumarlo a $D(y)$, cambiar cuanto vale g en el lado correspondiente, y hacer algún IF).
- Por lo tanto la complejidad de encontrar un flujo bloqueante con Wave es simplemente la cantidad total de veces que hacemos estas cosas.

- Para poder calcular cuantos de estos hacemos, podemos dividirlos en categorias.
- Cuando estamos mirando un $y \in \Gamma^+(x)$ para ver cuanto mandar desde x a y,pueden pasar dos cosas:
  1. Luego de terminar, el lado $\overrightarrow{xy}$ queda saturado.
  2. No queda saturado.
- Sea S la cantidad total sobre todas las olas hacia adelante de la categoria [1] arriba, y P la cantidad total, sobre todas las olas hacia adelante, de de la categoria [2] arriba.

- Similarmente, cuando estamos mirando $y \in M(x)$ para que x le devuelva flujo a y, pueden pasar dos cosas:
  - Luego de procesarlo, el lado $\overrightarrow{yx}$ queda vacio
  - No queda vacio
- Sea V la cantidad total sobre todas las olas hacia atras, de procesamientos de la categoria [I] arriba, y Q la cantidad total de procesamientos sobre todas las olas hacia atras, de lados de la categoria [II] arriba.
- Entonces la complejidad del paso bloqueante de Wave es $S + P + V + Q$.
- Calculemos esos números.

S

- Veamos primero S y V.
- Supongamos que un lado $\overrightarrow{xy}$ se satura.
- En el pseudocódigo que dimos, borrabamos y de $\Gamma^+(x)$.
- asi que efectivamente borramos el lado $\overrightarrow{xy}$ del network auxiliar.
- Y como nunca mas lo agregamos, nunca mas podriamos saturar el lado $\overrightarrow{xy}$, los lados se saturan una sola vez.
- Asi que S está acotado por m.

S

- Nos podriamos preguntar si es correcto borrar un lado.
- Dado que en Wave devolvemos flujo por lados (cosa que no haciamos en Dinitz) ¿Cómo sabemos que en realidad nunca lo necesitariamos de vuelta?
- Para poder ser usado otra vez, al estar saturado primero deberia des-saturarse, es decir, primero y deberia DEVOLVERLE flujo a x.
- La única forma en que y le puede devolver flujo a x es si y esta bloqueado.
- Pero si está bloqueado, x no puede mandarle mas flujo a y.
- En particular, $\overrightarrow{xy}$ no puede volver a ser usado, asi que está bien borrarlo.

V

- Similarmente, supongamos que $\overrightarrow{yx}$ se vacia.
- Aca no puedo argumentar como con S, porque borro a y de $M(x)$, pero mas adelante podria ser que lo vuelva a poner en $M(x)$, según el pseudocódigo que dimos.
- Pero $\overrightarrow{yx}$ sólo se puede vaciar si x esta bloqueado pues de otra forma no podria devolverle flujo a y.
- Pero si x esta bloqueado, el vértice "y" nunca mas puede mandarle flujo.
- Si $\overrightarrow{yx}$ nunca mas puede recibir flujo, entonces menos aún va a poder volver a vaciarse.
- Asi que V también está acotado por el número total de lados, m.

P

- Veamos P.
- En cada $FB(x)$ buscamos vecinos de x y les mandamos todo el flujo que podamos: $Min\{D(x), c(\overrightarrow{xy}) - g(\overrightarrow{xy})\}$.
- Si ese mínimo es igual a $c(\overrightarrow{xy}) - g(\overrightarrow{xy})$ el lado $\overrightarrow{xy}$ queda saturado, lo retiramos de $\Gamma^+(x)$ y continuamos con otro.
- Entonces, de entre todos los vécinos de x hay A LO SUMO uno sólo tal que ese mínimo es $D(x)$.
- Es decir, al hacer $FB(x)$, todos los lados $\overrightarrow{xy}$ que miramos, salvo a lo sumo uno, se saturan.
- Concluimos entonces que en cada FB hay a lo sumo UN lado procesado como parte de P.

P

- Con lo cual P está acotado superiormente por la cantidad total de FB .
- En cada ola hacia adelante, hay a lo sumo $n - 2$ FB asi que sólo necesitariamos calcular cuantas olas hacia adelante hay.
- En cada ola hacia adelante, salvo la última, AL MENOS UN vértice es bloqueado.
- ¿Por qué? Pues porque si una ola hacia adelante no bloquea ningún vértice, esto significa que los balanceó a todos, y si los balancea a todos, el algoritmo termina, asi que debe ser la última ola.
- Como los vértices NUNCA SE DESBLOQUEAN, puede haber a lo sumo n olas hacia adelante. ($n - 2 + 1 = n - 1$ si nos queremos poner estrictos, pero es irrelevante)

P, Q

- Por lo tanto, tenemos a lo sumo $n-2$ FB por cada ola hacia adelante, y tenemos $O(n)$ olas hacia adelante, significa que tenemos en total $O(n^2)$ FB y por lo tanto, la cantidad de procesamientos de P es $O(n^2)$.
- Similarmente, al hacer un BB, a lo sumo un lado no se vacia, asi que Q es la cantidad total de BB
- que son $n-2$ por cada ola hacia atras.
- y como el número de olas hacia atras es igual al número de olas hacia adelante,
- y vimos que estas estan acotadas por n,
- tenemos que Q tambien es $O(n^2)$.

- En resumen, la complejidad de hallar un flujo bloqueante en un network auxiliar con Wave es igual a:
- $S+P+V+Q = O(m)+O(n^2)+O(m)+O(n^2) = O(n^2)$.
- Como dijimos al comienzo, esto implica que la complejidad total de Wave es $O(n^3)$. Fin

- Una variación de estas ideas en donde en vez de usar network auxiliares se usan las etiquetas de niveles como habiamos explicado en Dinitz, es conocida como el push-relabel algorithm.
- Aunque es incluso un poco mas general, dependiendo como se etiqueteen los vértices.
- La complejidad general del push-relabel es $O(n^2 m)$ pero ciertas especificaciones que son equivalentes a Wave logran la complejidad $O(n^3)$.