Complejidad de Dinic

¿Cual es la complejidad del algoritmo de Dinic? Probarla en ambas versiones: **Dinitz original y Dinic-Even**.

Nota: no hace falta probar que la distancia en networks auxiliares sucesivos aumenta.

Enunciado

La complejidad del algoritmo de Dinic (sea la versión *Dinitz original* como *Dinic-Even*) es $O(n^2m)$

Demostración

- Se considera que $[3.2.1]$ significa *"se define esta propiedad o definición bajo este número o tag (para ser usado después)"*, mientras que $(3.2.1)$ hace referencia a que se usa tal propiedad o definición.
- Además, se considera que $(3.2.\{1,4\})$ es lo mismo que decir $(3.2.1),(3.2.4)$

---

0. Estructura

- 1 Complejidad general de los algoritmos *tipo* Dinic
- 2 Acotar $\#cntNA's$
  - 2.1. La distancia entre s y t aumenta entre NA's consecutivos
  - 2.2. Acotación a $O(n)$
- 3 $CCNA$ es $O(m)$
- 4 CFB es $O(mn)$
  - 4.1. CFB en *Dinitz original*
    - 4.1.1. Idea de su funcionamiento
    - 4.1.2. Complejidad de encontrar todos los caminos $O(mn)$
    - 4.1.3. Complejidad de hacer TODOS los $PODAR$ $O(mn)$
      - Se divide el cálculo en PV (recorrer los vértices) y $B(x)$ (borrar aristas de entrada de x)
    - 4.1.4. Conclusión para CFB de *Dinitz original* $O(mn)$
  - 4.2. CFB en *Dinic-Even*

- 4.2.1. Idea de su funcionamiento
- 4.2.2. Complejidad de $AVANZAR$, $RETROCEDER$ e $INCREMENTAR$
  - Son $O(1), O(1), O(n)$ respectivamente
- 4.2.3. Cálculo de CFB usando *"palabras"*
  - Consideramos palabras $AAA..AAX$ con $X \in \{R, I\}$
  - Hay m palabras y cada una es $O(n)$. Luego, CFB es $O(mn)$

- 5 Conclusión de los resultados
  - En ambos casos, como CFB es $O(mn)$, llegamos a $O(n^2m)$

1. Complejidad general de los algoritmos *tipo* Dinic

- En base a su funcionamiento, la complejidad de un *algoritmo tipo Dinic* puede considerarse como

$$O(\#cntNA's) \times O(CCNA + CFB) \ [1.1]$$

donde:
- $\#cntNA's$ es la cantidad de NA's en la corrida de Dinic
- $CCNA$ es la complejidad de crear un NA
- CFB es la complejidad del algoritmo para hallar un flujo bloqueante en un NA dado

2. Acotar $\#cntNA's$

2.1. La distancia entre s y t aumenta entre NA's consecutivos

- Esto es equivalente a decir que, salvo para el último NA (donde no se llega a t), la cantidad de niveles de un NA es menor que la cantidad de cualquier NA posterior.
- Lo consideramos *demostrado* en base a la *Nota* del problema.

2.2. Acotación a $O(n)$

- Por (2.1), la distancia entre s y t aumenta en al menos 1 por cada NA.
- Luego, salvo por el último NA, donde la distancia es ∞, la distancia debe ser un número natural entre 1 y $n-1$
- Por ello, entonces, hay a lo sumo n NA's $\Rightarrow \#cntNA's < n \Rightarrow O(\#cntNA's) = O(n)$

3. $CCNA$ es $O(m)$

- La complejidad de crear un NA es $O(m)$ ya que Dinic lo crea usando BFS

4. CFB es $O(mn)$

- Se va a dividir la prueba en dos casos: *versión de Dinitz* y *versión de Even*. Luego, los resultados se van a juntar en la sección (5)

4.1. CFB en *Dinitz original*

4.1.1. Idea de su funcionamiento

- En *Dinitz original*, cada camino entre s y t se encuentra usando DFS
  - El NA tiene la propiedad de que se garantiza que toda búsqueda nunca va a tener que hacer *backtracking* ya que cada vértice con lado entrante tiene un lado saliente (excepto t, claro)
  - Esta propiedad tiene el costo de matenimiento entre camino y camino, revisando el NA y actualizándolo (operación $PODAR$)
- Luego, la complejidad de CFB en este caso se va a ver como el costo de encontrar todos los caminos $+$ costo global de aplicar $PODAR$ $[4.1.1.1]$
  - Es decir, no se va a considerar el peor costo de $PODAR$ por cada camino realizado, sino su costo global *promedio*.

4.1.2. Complejidad de encontrar todos los caminos

- Sea r el número de niveles del NA, sabemos que como DFS no realiza backtracking, la construcción de cada camino es $O(r)$.
  Luego, como $r < n$, entonces la construcción de un camino es $O(n)$
- Luego de cada camino, se borran del NA los lados saturados. Como esto puede hacerse recorriendo el camino, es $O(n)$
- Como cada camino satura y, por lo tanto, borra al menos 1 lado, hay a lo sumo m caminos.
- Por ello, entonces, juntando la información anterior, sabemos que la complejidad de encontrar todos los caminos es

  $$O(\#cntCaminos) \times O(construirCamino + borrarLadosSaturados)$$

  lo cual es igual a

  $$O(m) \times O(n + n) = O(mn)$$

4.1.3. Complejidad de hacer TODOS los $PODAR$

- Como cada $PODAR$ viene luego de un camino ($+$ uno extra al principio del todo luego de la construcción del NA), entonces hay a lo sumo $m + 1$ de ellos
  - Esto implica, por ende, que la cantidad total de operaciones $PODAR$ es $O(m)$
- Cada $PODAR$ funciona del siguiente modo:
  - Va recorriendo todos los vértices, desde niveles más altos a más bajos, chequeando si tienen lados de salida y, en caso que no, lo borra junto con todos sus lados de entrada
  - Definamos cada sección del algoritmo como:
    - PV al recorrido de vértices y chequeo de lados de salida

    - $B(x)$ a borrar todos los lados de entrada del vértice x
  - Luego, cada $PODAR$ es hacer PV deteniéndonos en los vértices x que no tienen líneas de salida y ejecutando, allí, $B(x)$
- Veamos la complejidad de cada parte:
  - **Complejidad global de PV**
    - Como PV solo hace un chequeo de los vértices, entonces cada uno es $O(n)$
    - Luego, como hay $O(m)$ $PODAR$, tenemos que es $O(mn)$
  - **Complejidad global de $B(x)$**
    - Para cada $B(x)$ sabemos que su complejidad es $O(d(x))$ (en realidad, de la cantidad de líneas de entrada a x, pero están bajo el mismo O)
    - Para cada vértice, sabemos que $B(x)$ se aplica una sola vez (ya que que después lo sacamos del NA).
    - Luego, entonces, sabemos que hay $B(x)$ con costo $O(d(x))$ para cada vértice del NA. Por ello, entonces, su complejidad global es $O(\sum_x d(x)) = O(2m) = m$ por el *lema del apretón de manos*.
- Por ello, entonces, la *complejidad global de PODAR* es

$$O(PVs) + O(Bs) = O(mn) + O(m) = O(mn)$$

4.1.4. Conclusión para CFB de *Dinitz original*

- Dados los resultados obtenidos y en base a $(4.1.1.1)$, tenemos que

$$\begin{aligned} O(CFB) &= O(encontrarCaminos) + O(costoGlobalPODAR) \\ &= O(mn) + O(mn) \text{ por } (4.1.\{2,3\}) \\ &= O(mn) \end{aligned}$$

4.2. CFB en *Dinic-Even*

4.2.1. Idea de su funcionamiento

- En *Dinic-Even*, los DFS ahora pueden realizar backtracking ya que no mantiene la propiedad de los NA que considera *Dinitz original*.
- La idea de su funcionamiento es la siguiente:
  - Se busca el camino usando backtracking en caso que con DFS hayamos llegado a un nodo sin líneas de salida (que no es t)
    - En caso que no haya camino, cortamos la ejecución.
    - Caso contrario, con el camino que encontramos, se calcula cuánto se puede mandar, incrementamos el flujo y borramos del NA cualquier lado que haya sido saturado
  - Por ello mismo, podemos dividir las partes más *importantes del código* en las siguientes operaciones (que son las que se van a usar para el cálculo de complejidad)

    - $AVANZAR(x)$: elegimos algún vecino y de $\Gamma^+(x)$, agregamos $\overrightarrow{xy}$ al camino y cambiamos $x = y$
        - Solo se ejecuta si $\Gamma^+(x) \neq \emptyset$
    - $RETROCEDER(x)$: se toma el nodo z anterior a x en la pila, se borra $\overrightarrow{zx}$ del camino y del NA, y hacemos $x = z$
        - Solo se ejecuta si se puede hacer, es decir, si $x \neq s$
          Caso contrario, significa que no hay otro camino
    - $INCREMENTAR(x)$: una vez construido el camino, se calcula cuánto se puede mandar, se incrementa el flujo y se borran del NA los lados saturados

4.2.2. Complejidad de $AVANZAR$, $RETROCEDER$ e $INCREMENTAR$

- $AVANZAR(x)$: Es $O(1)$ ya que solo busca un nodo cualquiera de $\Gamma^+(x)$, se agrega un lado y se cambia x
- $RETROCEDER(x)$: Es $O(1)$ ya que solo se borra un lado y se cambia x
- $INCREMENTAR(x)$: Es $O(n)$ ya que la longitud es a lo sumo n y, por ende, las tres operaciones (calcular cuánto mandar, incrementar el flujo y borrar lados saturados) son $O(n)$

4.2.3. Cálculo de CFB usando *"palabras"*

- Si denotamos $AVANZAR$ por A, $RETROCEDER$ por R e $INCREMENTAR$ por I, entonces se puede ver que *Dinic-Even* es una sucesión de A, R, I
- Dado esto, consideremos que dividimos la sucesión en palabras $AAA..AAX$ donde $X \in \{R, I\}$
    - **¿Cuántas palabras hay?**
        - Cada palabra termina en $X \in \{R, I\}$.
        - Como R borra el lado por el que retrocede e I borra todos los lados saturados (al menos uno ya que manda flujo), entonces X borra al menos un lado
        - Luego, la cantidad de palabras es a lo sumo m
    - **¿Cuál es la complejidad de cada palabra?**
        - R, A son $O(1)$ mientras que I es $O(n)$
        - Luego, una palabra $AAA..AAX$ con k $A's$ tiene complejidad $O(k+1) = O(k)$ si $X = R$ y $O(k+n)$ si $X = I$
        - Pero A mueve el extremo del camino un nivel hacia adelante.
          Como hay r niveles (y a lo sumo puede haber n), entonces $k \leq r \leq n$.
        - Luego, entonces, la complejidad es $O(n)$ si $X = R$ y $O(n+n) = O(n)$ si $X = I$
          Es decir, $O(n)$ en cualquier caso
- Luego, como hay a lo sumo m palabras y cada una es $O(n)$, entonces CFB es $O(mn)$

5. Conclusión de los resultados

- Como puede verse por $(4.1.5)$ y $(4.2.3)$, CFB tiene complejidad $O(mn)$
- Luego, por (1.1) sabemos que

$$O(\#cntNA's) \times O(CCNA + CFB)$$

- Como $O(\#cntNA's) = O(n), O(CCNA) = O(m), O(CFB) = O(mn)$, si reemplazamos tenemos que $O(Dinic) = O(n \times (m + mn)) = O(n^2m)$ $\blacksquare$