Teoremas para el final de matemática discreta II 2024

Tomás Maraschio

19 de junio de 2024

Índice

1. ¿Cuál es la complejidad de Dinic en ambas versiones? No hace falta probar que la distancia en NAs sucesivos aumenta

Probaré que $Compl(Dinic) = O(n^2 \times m)$.

1.1. Complejidad de NAs

Como en cada NA la distancia entre s y t aumenta, tendré a lo sumo O(n) NAs. Además, como uso BFS, la complejidad de construir cada NA es

O(m). Entonces tengo que

$$\begin{aligned} Compl(Dinic) &= (Compl(\text{hallar flujo bloqueante}) + Compl(construirNA)) \times \#\text{cantidad de NAs} \\ &= (Compl(\text{hallar flujo bloqueante}) + O(m)) \times O(n) \end{aligned}$$

Me basta probar que $Compl(\text{hallar flujo bloqueante}) = O(n \times m)$.

1.2. Versión original

Los NAs en esta versión tienen la propiedad de que todo vértice tiene algún lado no saturado que va al siguiente nivel. Esto significa que DFS encuentra un camino de s a t sin tener que hacer backtracking. Entonces la complejidad de encontrar un camino en el NA es O(n° niveles) = O(n). Además, aumentar el flujo en un camino es O(n), pues hay que recorrer el camino dos veces. Ahora, como cada camino aumentante satura (al menos) un lado, tendré O(m) caminos aumentantes. Entonces por ahora tengo que la complejidad de encontrar los caminos y aumentar el flujo en ellos es $O(n \times m)$. Sin embargo, tengo que ver cuánto cuesta mantener la propiedad que mencioné al principio.

Para mantener esa propiedad, luego de cada camino ejecuto el proceso 'podar', que va revisando los vertices desde los niveles superiores para abajo:

- Si el vertice tiene lados que salen no hace nada
- Si no tiene lados que salen, borra el vértice y sus lados al siguiente nivel

Revisar si un vértice tiene lados que salen es O(1), y como luego de cada camino hago esto para cada vértice, esta parte me cuesta $O(n \times m)$.

Ahora, borrar un vértice y sus lados se hace a lo sumo una vez por vértice y su complejidad es O(d(x)). Entonces, la complejidad sobre todos los 'podar' de esta parte es $\sum_{x \in V} d(x) = O(m)$.

Finalmente, la complejidad de hallar flujo bloqueante es

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

1.3. Versión occidental

Daré el algoritmo que se usa para encontrar un flujo bloqueante g:

```
g := 0
while (1) {
    p := [s]
    x := s
    while (x != t) {
        if (VecinosAdelanteDe(x) != []) {      // Avanzar (A)
            y := tomar de VecinosAdelanteDe(x)
            agregar y al final de p
            x := y
        } else {
            if (x != s) {                      // Retroceder (R)
                z := vértice anterior a x en p
                borrar zx
                x := z
            } else {
                return g
            }
        }
    }
    aumentar g en p y borrar lados saturados   // Incrementar (I)
}
```

Notar que una corrida de este algoritmo es de la forma $A \cdots AIA \cdots AIA \cdots AR \cdots$ es decir, una sucesión de palabras de la forma $A \cdots AX$ donde X es I o R (excepto la última palabra de todas, que terminará con A). Además, $O(A) = O(R) = O(1)$ y $O(I) = O(n)$, pues para aumentar el flujo y borrar los lados tengo que recorrer el camino p dos veces y este tiene a lo sumo longitud n.

Tengo que ver cuantas As puede haber en una palabra. Como cada A me lleva al siguiente nivel del NA, hay O(n) As en cada palabra. De esta forma, el coste de las palabras me quedó:

- $Compl(A....AR) = O(n) + O(1) = O(n)$
- $Compl(A....AI) = O(n) + O(n) = O(n)$

Solo basta ver cuantas de estas palabras puede haber. Como cada R borra un lado y cada I borra al menos un lado, tengo que hay a lo sumo m palabras.

Finalmente:

$$Compl(\text{hallar flujo bloqueante}) = O(n) \times O(m) = O(n \times m)$$

2. ¿Cuál es la complejidad del algoritmo de Wave? No hace falta probar que la distancia en NAs sucesivos aumenta

Probaré que $Compl(Wave) = O(n^3)$.

Al igual que en Dinic, como la distancia entre s y t aumenta en NAs sucesivos, tengo O(n) NAs, pues t no puede estar a una distancia mayor a n de s. Ahora, como

$$Compl(Wave) = (Compl(\text{hallar flujo bloqueante}) + Compl(\text{construir NA})) \times O(n)$$

y $Compl(\text{construir NA}) = O(m)$ pues uso BFS, bastaría probar que

$$Compl(\text{hallar flujo bloqueante}) = O(n^2).$$

Notar que $O(n^2) + O(m) = O(n^2)$ pues $m \leq \binom{n}{2} = O(n^2)$.

Voy a dividir el proceso de hallar un flujo bloqueante en casos.

Cuando estoy balanceando vértices hacia adelante:

- V: los pasos donde saturo un lado
- P: los pasos donde no saturo un lado

Cuando estoy balanceando los vértices hacia atrás:

- S: los pasos donde vacío un lado
- Q: los pasos donde no vacío un lado

V: Sup. que x le manda flujo a z y $\overrightarrow{xz}$ se satura. Para que $\overrightarrow{xz}$ vuelva a saturarse, primero tiene que vaciarse un poco, pero si se vacía es porque z se bloqueó y entonces le devolvió flujo a x. Pero como z está bloqueado (y nunca se desbloqueará), x no le va a volver a mandar flujo a z. Entonces, cada lado puede saturarse a lo sumo una vez. $\therefore Compl(V) = O(m)$.

S: Sup. que x le devuelve flujo a z y $\overrightarrow{zx}$ se vacía. Como x devolvió flujo significa que está bloqueado, entonces z no volverá a mandarle flujo, entonces $\overrightarrow{zx}$ no se podrá llenar para volver a vaciarse. Esto implica que un lado se puede vaciar a lo sumo una vez. $\therefore Compl(S) = O(m)$.

P: Cuando un vértice manda flujo a sus vecinos para balancearse, satura todos los lados, excepto quizás uno. Esto significa que para cada vértice en cada ola hacia adelante se satura parcialmente a lo sumo un lado. Entonces $Compl(P) = O(n) \times \#\text{olas hacia adelante}$.

Ahora, en cada ola hacia adelante (excepto la última) queda al menos un vértice desbalanceado, es decir que cada ola hacia adelante hace que se bloquee al menos un vértice. Y como no se desbloquean una vez bloqueados, tengo que #olas hacia adelante $= O(n)$. $\therefore Compl(P) = O(n^2)$.

Q: Cuando un vértice devuelve flujo a los vecinos, vacía todos los lados excepto quizás uno. Esto significa que para cada vértice en cada ola hacia atrás se vacía parcialmente a lo sumo un lado. Entonces $Compl(Q) = O(n) \times$ #olas hacia atrás $= O(n) \times$ #olas hacia adelante $= O(n^2)$.

Finalmente,

$Compl($**hallar flujo bloqueante**$) = O(m)+O(n^2)+O(m)+O(n^2) = O(n^2)$.

3. Probar que el valor de todo flujo es menor o igual a la capacidad de todo corte. También que si f es flujo, es maximal si y solo si existe S corte con V(f) = Cap(S) y S es minimal. No hace falta probar que $V(f) = f(S, \overline{S}) - f(\overline{S}, S)$

3.1. Valor de todo flujo menor o igual a capacidad de todo corte

$$f(\overline{S}, S) = \sum_{\substack{x \notin S \\ y \in S \\ \overrightarrow{xy} \in E}} f(\overrightarrow{xy}) \geq 0 \text{ pues } f(\overrightarrow{xy}), \forall \overrightarrow{xy} \in E$$

Entonces tengo $V(f) = f(S, \overline{S}) - f(\overline{S}, S) \leq f(S, \overline{S}) \leq c(S, \overline{S}) = Cap(S)$

3.2. $\Longleftarrow$ vuelta

Sea f flujo y S corte con $V(f) = Cap(S)$. Para todo flujo g, por lo que probé arriba tengo $V(g) \leq Cap(S) = V(f) \implies$ f es maximal. Además, para todo corte T tengo $Cap(T) \geq V(f) = Cap(S) \implies$ S es minimal.

3.3. $\Longrightarrow$ ida

Sea f flujo maximal. Voy a construir un S y probaré que es un corte minimal con V(f) = Cap(S).

$$S = \{s\} \cup \{x\text{: existe f-camino aumentante de s a x}\}$$

S es corte

Sup. que no lo es. La única forma que pase esto es porque $t \notin S$, pues $s \in S$. Pero esto significa que existe un f-camino aumentante de s a t, por lo que puedo aumentar f en ese camino. Absurdo porque f es maximal, luego S es corte.

V(f) = Cap(S)

Sean $x \in S, y \notin S, \overrightarrow{xy} \in E$.

$$x \in S \implies \text{existe f-camino aumentante de s a x}$$

$$y \notin S \implies \text{no existe f-camino aumentante de s a y}$$

Entonces el camino $s \cdots xy$ podría ser aumentante, pero no lo es, y esto solo puede ser si $f(\overrightarrow{xy}) = c(\overrightarrow{xy})$. Entonces

$$f(S, \overline{S}) = \sum_{\substack{x \in S \\ y \notin S \\ \overrightarrow{xy} \in E}} f(\overrightarrow{xy}) = \sum_{\substack{x \in S \\ y \notin S \\ \overrightarrow{xy} \in E}} c(\overrightarrow{xy}) = c(S, \overline{S}) = Cap(S)$$

Ahora sean $x \notin S, y \in S, \overrightarrow{xy} \in E$.

$$x \notin S \implies \text{no existe f-camino aumentante de s a x}$$

$$y \in S \implies \text{existe f-camino aumentante de s a y}$$

Entonces el camino $s \cdots \overleftarrow{yx}$ podría ser aumentante, pero no lo es, y esto solo puede ser si $f(\overrightarrow{xy}) = 0$. Entonces

$$f(\overline{S}, S) = \sum_{\substack{x \notin S \\ y \in S \\ \overrightarrow{xy} \in E}} f(\overrightarrow{xy}) = \sum_{\substack{x \notin S \\ y \in S \\ \overrightarrow{xy} \in E}} 0 = 0$$

Finalmente: $V(f) = f(S, \overline{S}) - f(\overline{S}, S) = Cap(S) - 0 = Cap(S)$

4. Probar que si G es un grafo conexo no regular entonces $\chi(G) \leq \Delta(G)$

Sea x tal que $d(x) = \delta(G)$. Corro BFS empezando por x y guardo el orden inverso en que los vértices se visitaron. Ahora voy a correr greedy en

este orden que acabo de guardar. Notar que en BFS todo vértice es incluido por un vértice que ya ha sido visitado, entonces en el orden inverso todo vértice tiene al menos un vecino por delante (excepto el x). Esto significa que para cada vértice y, en el peor caso va a tener $d(y) - 1 < \Delta$ vecinos coloreados todos con un color distinto, entonces voy a poder elegir un color en $\{1, \cdots, \Delta\}$. Finalmente, cuando llego a x, como $d(x) = \delta < \Delta$, podré elegir algún color de $\{1, \cdots, \Delta\}$ que no está usado por ningún vecino de x para colorearlo. De esta forma tengo un coloreo propio de G que usa (a lo sumo) Δ colores, entonces $\chi(G) \leq \Delta(G)$.