Algoritmos que usan la estrategia de Dinic (incluyendo Dinic, obvio)

Hemos visto que Edmonds-Karp es un algoritmo polinomial, de complejidad $O(nm^2)$. Para networks ralos ($n \simeq m$), esto da una complejidad de $O(n^3)$, lo cual es la complejidad de los algoritmos mas rapidos, pero para networks densos ($m \simeq n(n-1)/2$, esto da un acomplejidad de $O(n^5)$, lo cual ya no es tan bueno.

El problema lo podemos ver en la corrida del ejemplo que hicimo, o en varios de los ejercicios que uds. deben hacer: muchas veces, se recaba bastante información al cosntruir el BFS tree, pero una vez que se encuentra el camino entre s y t, toda esa información se descarta. A menudo en el paso siguiente, la mayor parte del trabajo que se hizo se repite.

A Dinic (de la ex-URSS) se le ocurrió una estrategia para evitar reperit todo este trabajo: eluso deun netwrok auxiliar.

Veamos algunas definiciones:

Definición:

Un *layered network* (network por niveles) es un network (V, E, c) tal que V se descompone en conjuntos disjuntos V_i: $V = \cup_{i=0}^{r} V_i$ con la propiedad de que:

1) $V_0 = \{s\}$

2) $V_r = \{t\}$

3) $E \subseteq \cup_{i=0}^{r-1} V_i \times V_{i+1}$.

Es decir, tenemos los niveles V_i, y solo hay lados desde un vertice de un nivel a un vertice del nivel inmediatamente superior.

Definición Dado un network N y un flujo f, el network *residual* de N relativo a f ($R_f(N)$ es el network cuyos vertices son los mismos de N, y cuyos lados y capacidades son $\overrightarrow{\overrightarrow{xy}}$ tales que:

1) $\overrightarrow{xy}$ tambien es un lado en N y $f(\overrightarrow{xy}) < c(\overrightarrow{xy})$. En este caso la capacidad del lado $\overrightarrow{\overrightarrow{xy}}$ (en $R_f(N)$) se define como $c(\overrightarrow{xy}) - f(\overrightarrow{xy})$.

o bien:

2) $\overrightarrow{yx}$ es un lado en N con $f(\overrightarrow{yx}) > 0$. En este caso la capacidad del lado $\overrightarrow{\overrightarrow{xy}}$ (en $R_f(N)$) se define como $f(\overrightarrow{yx})$.

Definición Dado un network N y flujo f en N, definimos el network auxiliar (NA=NA(f, N)) de N con respecto a f como el layered network construido de la siguiente forma: correr BFS(s) en el network residual. Definir V_i como los vertices de nivel i del BFS. Si $t \in V_r$, eliminar del network todos los niveles superiores a r, y redefinir V_r como $\{t\}$. (i.e., elimino todos los otros vertices en el mismo nivel que t). Los lados del network seran todos los lados del network residual que conecten un vertice de V_i con uno de V_{i+1}.

Una forma de construir el NA es directamente correr BFS como si estuvieramos haciendo EK. Solo que vamos llevando un registro de los niveles, y que, si un vertice x agrega un nuevo vertice y, entonces agregamos el lado $\overrightarrow{xy}$, pero ademas, si x fuese a agregar un vertice y que ya esta en la cola, entonces nos fijamos si el nivel de y es uno mas que el nivel de x. Si lo es, tambien agregamos el lado $\overrightarrow{xy}$. (es decir, NO agregamos si y esta en un nivel anterior, o en el mismo nivel que x).

De esta forma, se construye el NA en tiempo $O(m)$.

Observar que t puede no estar en NA, pues puede no haber un camino en el network residual de s a t.

Definición: Dado un network N, un flujo *bloqueante* o *saturante* ("blocking flow", "saturating flow") es un flujo f tal que no hay caminos desde s a t cuyos lados sean todos de tipo forward, es decir, tal que todo camino dirigido de s a t tiene al menos un lado $\overrightarrow{xy}$ con $f(\overrightarrow{xy}) = c(\overrightarrow{xy})$.

Obviamente, una forma de encontrar un flujo bloqueante es simplemente correr Greedy.

Supongamos que tenemos un flujo f en un network N, y que tenemos un flujo g en el network residual. Definiremos la suma de f con g ($f + g$ o $f \oplus g$) como el flujo que es igual a:

1) en los lados forward: $(f \oplus g)(\overrightarrow{xy}) = f(\overrightarrow{xy}) + g(\overrightarrow{\overrightarrow{xy}})$

2) en los lados backward $f \oplus g)(\overrightarrow{yx}) = f(\overrightarrow{yx}) - g(\overrightarrow{\overrightarrow{xy}})$.

Podemos describir ahora los algoritmos basados en el modelo de Dinic.

```
f:=0;
Construir NA de f;
While (t este en NA)
       Hallar un blocking flow g en NA;
       f:=f+g;
       Construir NA de f;
Endwhile;
Output(f);
```

Los diversos algoritmos difieren solo en el paso "Hallar un blocking flow g en NA;". El algoritmo original de Dinic era basicamente usando Greedy, pero implementado en una forma mas eficiente que la usual, tomando en cuenta que el network en que se esta corriendo es una layered network. Veremos Dinic, y luego veremos otros dos algoritmos que tienen una estrategia distinta de la Greedy para hallar un blocking flow.

Pero primero: ¿cual es la complejidad de estos algoritmos?

Construir el NA es $O(m)$, por lo tanto la complejidad sera:

($O(m)$+complejidad de hallar blocking flow).numero de veces que debemos construir NAs.