Como siempre, solo indicaremos como es encontrar un flujo saturante o bloqueante g en el network auxiliar NA.

MKM funciona a nivel de vertices. Se define la capacidad (residual) de un vertice como el minimo de las capacidades de entrada y de salida, es decir:

$$c^-(x) = \sum_{y \in \Gamma^-(x)} (c(\overrightarrow{yx}) - g(\overrightarrow{yx}))$$

$$c^+(x) = \sum_{y \in \Gamma^+(x)} (c(\overrightarrow{xy}) - g(\overrightarrow{xy}))$$

$$c(x) = \begin{cases} Min\{c^-(x), c^+(x)\} & \text{si } x \neq s, t \\ c^+(s) & \text{si } x = s \\ c^-(t) & \text{si } x = t \end{cases}$$

MKM busca el vertice de menor capacidad. Si esa capacidad es cero, se borra el vertice del network, se actualizan las capacidades y se sigue buscando. Cuando se encuentra un vertice x con capacidad minima M distinta de cero, se "empujan" ("push") M unidades de flujo desde x hacia sus vecinos hacia t. Esto desbalancea estos vertices, por lo tanto, se van recorriendo ellos, empujando el flujo que les haya llegado en cada momento hacia sus vecinos. Mientras se hace esto, lo que se va obteniendo no es un flujo, pues hay vertices desbalanceados, pero al final de todo los vertices quedan balanceados y se tiene un flujo. El recorrido se hace en modo BFS, de modo de acumular la mayor cantidad posible de flujo en un vertice antes de tener que balancearlo (de esta forma se procesa cada vertice solo una vez) Nunca va a haber un problema de que se "trabe" el proceso en algun vertice pues empezamos en el vertice de menor capacidad.

Una vez que se empujo todo hacia t, como x esta ahora desbalanceado, "jalamos" ("pull") M unidades de flujo desde x hacia sus vecinos hacia s, repitiendo el proceso, en el orden BFS inverso.

Una vez terminado esto, se borra x, se actualizan las capacidades, y se sigue. Se para cuando la capacidad o bien de s o bien de t sea cero, pues indica que no se puede sacar o recibir mas flujo.

En pseudo codigo, seria:

g:=0
Calcular los c^-, c^+, c
FORALL $x \in V$ DO: $D(x) := 0$//etiqueta que mide el desbalanceo del vertice
WHILE (($c(s) \neq 0$) AND ($c(t) \neq 0$)) DO
    $x := Argmin\{c(v) | v \in V\}$
    IF ($c(x) \neq 0$) THEN PP(x)
        ELSE REMOVER(x)
    ENDIF
ENDWHILE
RETURN(g)

Donde las subrutinas REMOVER y PP son las siguientes:

REMOVER(x):
FORALL $y \in \Gamma^+(x)$ DO:
    $c^-(y) := c^-(y) - (c(\overrightarrow{xy}) - g(\overrightarrow{xy}))$
    $c(x) = Min\{c^-(x), c^+(x)\}$
    Borrar $\overrightarrow{xy}$ de NA.
ENDFOR
FORALL $y \in \Gamma^-(x)$ DO:
    $c^+(y) := c^+(y) - (c(\overrightarrow{yx}) - g(\overrightarrow{yx}))$
    $c(x) = Min\{c^-(x), c^+(x)\}$
    Borrar $\overrightarrow{yx}$ de NA.
ENDFOR
Borrar x de NA.

PP(x):

PUSH(x)

PULL(x)

$c(x) := 0$//durante PULL y PUSH las capacidades de todos los vertices cambian menos la de x

Describimos PUSH abajo. PULL se deja como ejercicio.

PUSH(x):

D(x):=c(x)//desbalanceamos x
SUBPUSH(x)//esto cambiara otros $D(z)$'s y otros $c^-(z)$'s
FOR $z =$"$x+1$" TO "$t-1$" DO: //en el orden BFS
$c^+(z) := c^+(z) - D(z)$//no hace falta cambiar c^- pues fue cambiada en subpush
$c(z) := c(z) - D(z)$//las capacidades disminuyen en el desbalance de z.
SUBPUSH(z)
ENDFOR
$c(t) := c^-(t)$// $c^-(t)$ se habra ido cambiando durante los SUBPUSHes de sus vecinos

y tenemos:

SUBPUSH(z):

WHILE ($D(z) > 0$) DO:
$y :=$"Primer" elemento de $\Gamma^+(z)$.//en el orden BFS o no, no importa mucho aca
$A := Min\{D(z), c(\overrightarrow{zy}) - g(\overrightarrow{zy})\}$// esto es lo que voy a poder mandar por $\overrightarrow{zy}$.
$g(\overrightarrow{zy}) := g(\overrightarrow{zy}) + A$//incremento el flujo
$D(y) := D(y) + A$//se incrementa el desbalanceo de y pues mandamos flujo
$D(z) := D(z) - A$//se alivio el desbalanceo de z al mandar flujo
$c^-(y) := c^-(y) - A$// al mandar flujo, hay menos capacidad entrante
IF $g(\overrightarrow{zy}) = c(\overrightarrow{zy})$ THEN borrar $\overrightarrow{zy}$ de NA, borrar y de $\Gamma^+(z)$.
ENDWHILE

Observaciones:

1) Observar que si se hace PP, entonces x NO se remueve. Sin embargo, en la siguiente iteracion, como luego de un PP, el x tendra capacidad 0, sera removido. Podriamos entonces ahorrarnos una iteracion, y removerlo directamente dentro del PP. Esto sin embargo seria un error, puesto que si en alguna iteracion el vertice de menor capacidad es t o s, entonces lo removeriamos y no estoy seguro que pasaria con la guarda entonces. (supongo que estaria bien, pues "$c(s)$" y "$c(t)$" serian registros que seguirian existiendo aunque s y t se borraran de NA, pero por las dudas...

De la forma en que esta, s y t no se remueven.

2) La capacidad c^- que se cambia en SUBPUSH podria no ser cambiada ahi, y cambiarla directamente en bloque cuando cambiamos las otras capacidades c^+ y c. Esto es exactamente igual para los vertices intermedios, pero no lo es para t: si no cambiamos $c-$ para t en los subpush de sus vecinos anteriores, entonces como nunca hacemos SUBPUSH(t), no cambiariamos nunca la capacidad de t. Sin embargo, otra posibilidad seria usar el hecho de que $D(t)$ si habra cambiado, y rebajar $c(t)$ por ese cambio. El problema es que, si bien los desbalanceos de los otros vertices si se modifican, hasta que vuelven a ser cero, con lo cual $D(z)$ realmente guarda el desbalanceo producido en el paso actual, $D(t)$ guardara el desbalanceo total, pues t no se balancea. Con lo cual, tendriamos que tener un registro extra, o bien hacer $D(t) = 0$ al final del push (algo parecido vale para s y el pull).

3) A pesar de lo dicho anteriormente, conviene observar que al final de todo el algoritmo $D(t)$ no necesariamente guarda el desbalanceo total de t. (ni $D(s)$ el de s) Esto es porque hay una situacion en la cual se resetea $D(t)$: cuando t es el vertice de menor capacida, en la linea "$D(x) := c(x)$" si $x = t$, se cambiara $D(t)$ a un valor nuevo que no necesariamente reflejara el desbalanceo total de t, sino solo el desbalanceo del ultimo paso. (luego de este paso, el algoritmo terminara)

4) Observar que esto requiere programar bien el "FOR $z =$"$x+1$" TO "$t-1$" DO: " para que no haga nada si $x = t$. En la notacion esta implicito esto.