DINIC

Solo indicaremos como es encontrar un flujo saturante o bloqueante g en el network auxiliar NA. DINIC opera en el NA usando DFS, por lo tanto tendremos un camino parcial p, por el cual avanzaremos cuando podamos, retrocederemos cuando no podamos, y finalmente, si llegamos a t, aumentaremos el flujo a lo largo de ese camino. Se usaran las subrutinas AVANZAR, RETROCEDER y AUMENTAR, que son mas o menos obvias, y las escribimos luego. Estas subrutinas se colocan en un DFS. La version original de Dinic(1970) usaba, por supuesto, GOTOs:

DINIC-version GOTO

```
    g := 0
1.  p := [s], x := s // Inicialización
2.  IF Γ+(x) ≠ ∅   THEN { AVANZAR
                          IF (x ≠ t) THEN GOTO 2
                                     ELSE AUMENTAR, GOTO 1
                          ENDIF
                        }
                   ELSE IF (x ≠ s)   THEN RETROCEDER GOTO 2
                                     ELSE RETURN(g) STOP
```

Esta version es facilmente entendible, pero para no ir contra la religion oficial, he aqui una version sin GOTOs:

DINIC-version GOTOless

```
g := 0
STOPFLAG:=0//para saber cuando parar
WHILE (STOPFLAG=0) DO://while externo
        p := [s], x := s // Inicialización
        WHILE ((x ≠ t) AND (STOPFLAG=0)) DO://while interno
                IF Γ+(x) ≠ ∅ THEN AVANZAR
                             ELSE IF (x ≠ s) THEN RETROCEDER
                                             ELSE STOPFLAG:=1
                                   ENDIF
                ENDIF
        ENDWHILE//while interno
IF (x = t) THEN AUMENTAR//puedo haber salido del while por dos razones
ENDWHILE//while externo
RETURN(g)
```

Finalmente, las subrutinas:

AVANZAR:
Tomar $y \in \Gamma^+(x)$ // AVANZAR solo se efectua si esto no es vacio.
$p := p + [y]$
$x := y$

RETROCEDER:
$y :=$ predecesor de x en p
$p := p - x$
borrar $\overrightarrow{yx}$ de NA
$x := y$

(en esta version, RETROCEDER es $O(1)$. En otra version en vez de borrar solo $\overrightarrow{yx}$ se borran todos los lados incidentes con x. Esa es $O(m)$. Sin embargo, la complejidad final del algoritmo resulta ser la misma. Esta version es, sin embargo, mas facil de programar.)

AUMENTAR:

$\Delta := Min\{c(\overrightarrow{uv}) - g(\overrightarrow{uv}) | \overrightarrow{uv} \in p\}$
FORALL $\overrightarrow{uv} \in p$ DO:
  $g(\overrightarrow{uv}) := g(\overrightarrow{uv}) + \Delta$
  IF $g(\overrightarrow{uv}) = c(\overrightarrow{uv})$ THEN borrar $\overrightarrow{uv}$ de NA.