Ingeniería Investigación y Tecnología, ISSN 2594-0732,
VII. 1. 45-55, 2006 (artículo arbitrado)
DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

Diseño de un programa en Logo para calcular flujos máximos en redes

M.A. Murray-Lasso
Unidad de Enseñanza Auxiliada por Computadora
Departamento de Ingeniería de Sistemas, División de Ingeniería Mecánica e Industrial
Facultad de Ingeniería, UNAM
E-mail: mamurray@servidor.unam.mx

(recibido: agosto de 2004; aceptado: diciembre de 2004)

Resumen

Se muestran los pormenores del diseño de un programa Logo para calcular flujos máximos entre pares de puntos sobre redes orientadas, cuyas ramas tienen un límite superior al flujo que puede circular por ellas. Para el diseño del programa se utilizan Listas, que es la única estructura de datos soportada por Logo, pero que es muy adecuada para representar y manipular redes que no están muy densamente conectadas, que es el caso más frecuente en redes de flujo. Dado que Logo no soporta arreglos, y éstos resultan convenientes en el programa, se muestra la manera de implementarlos utilizando la flexibilidad de Logo en el manejo de los nombres de las variables. También se proporcionan listas de programas en Logo Writer que implementan los algoritmos de etiquetado de Ford y Fulkerson. Como apoyo a la docencia se incluyen rutinas que se pueden utilizar como instrumentos que le permiten al maestro y alumno seguir las variaciones de las estructuras durante el proceso. Finalmente, se resuelven en detalle dos ejemplos ilustrativos.

Descriptores: Logo, flujos máximos, algoritmos de etiquetado, listas, arreglos.

Abstract

The details of the design of a Logo program to calculate maximum flows between pairs of points in oriented networks with branches having upper limits to flow are presented. For the design Lists, which are the only data structures supported by Logo, but which are quite appropriate to represent and handle networks not densely connected, the most common case in flow networks are utilised. Since Logo does not support arrays and they are convenient in the program, it is shown how to implement them taking advantage of the flexibility of Logo in the handling of the names of variables. Listings of Logo Writer programs implementing the labelling algorithms of Ford and Fulkerson are provided. As teaching aids, routines that can be used as instruments to allow teachers and students to follow the variations in the structures during the process, are also provided. Two illustrative examples are solved in detail.

Keywords: *Logo, maximum flows, labelling algorithms, lists, arrays.*

Introducción

El lenguaje Logo es uno de los lenguajes educativos diseñados para la enseñanza, incluyendo la enseñanza de la programación. Se ha popularizado entre los maestros de educación básica y media por tener entradas fáciles al lenguaje que permiten estar programando desde la primera sesión. El lenguaje generalmente funciona como un intérprete, no tiene instrucción GOTO, es altamente estructurado, favorece la creación de procedimientos que se comunican entre sí y el uso de la recursión. El lenguaje fue creado entre la empresa Bolt, Beranek y Newman y el Grupo de Inteligencia Artificial del MIT (Instituto Tecnológico de Massachusetts)y es un derivado del LISP. Tiene suficientes diferencias

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

comparado con lenguajes como BASIC, Pascal y C que orilla al programador a pensar con un estilo diferente. Su prin ci pal (y en muchas versiones, su única) estructura de datos, es la Lista, a la cual se espera se le agreguen y eliminen los elementos exclusivamente en su principio y en su final. El lenguaje Logo ha adquirido la reputación de ser un lenguaje de programación para niños; sin embargo, el poderío del lenguaje es igual al de cualquier otro lenguaje de programación; lo que pasa es que muchos libros se concentran en (y en algunos casos se limitan a) las instrucciones elementales para dibujar con la tortuga y casi no tratan las porciones "avanzadas" del lenguaje como el manejo no trivial de listas y la recursión no triv ial.

En este artículo se diseña un programa para el cálculo del flujo máximo en una red, el cual requiere las porciones avanzadas del lenguaje. En el programa diseñado se implementan lo que equivale a vectores de listas y arreglos bidimensionales (no obstante que el lenguaje no los soporta), los cuales permiten el manejo eficiente de la memoria para redes poco densamente conectadas, que son la regla, más que la excepción en las aplicaciones. El propósito principal de este trabajo es mostrar las partes avanzadas del lenguaje Logo en un contexto de aplicación a un caso específico, en vez de hacerlo con el estilo de un manual, ilustrando con ejemplos muy cortos el uso de cada instrucción y estructura de datos.

Algoritmo de etiquetado de Ford y Fulkerson para el cálculo del flujo máximo de una red

Entre los principales pioneros de la disciplina de flujos en redes están Ford y Fulkerson (1962), quienes dieron un algoritmo para calcular el flujo máximo que puede fluir entre un nodo origen y un nodo destino en una red orientada cuyas ramas tienen una capacidad máxima de flujo. Una típica red es la que se muestra en la figura 1, la cual utilizaremos como ejemplo du rante el artículo. En la red de la figura 1 se han numerado consecutivamente los nodos; las cantidades junto a las flechas denotan la capacidad máxima de flujo en la dirección de la flecha de la rama correspondiente. El nodo 1 es el nodo origen al cual se le inyecta un flujo que sale por el nodo destino 6. El algoritmo de Ford y Fulkerson es un algoritmo de etiquetado.

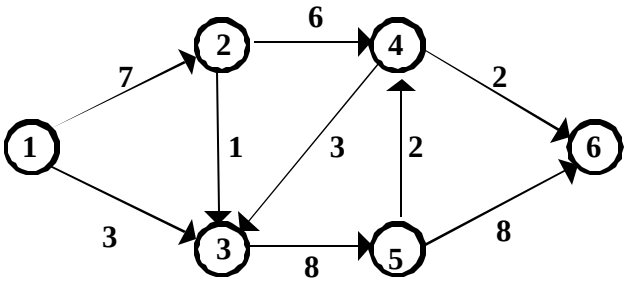

Figura 1.

Si llamamos a los nodos *i, j, k, ..., s , t ,...* (al nodo fuente de donde parte el flujo hacia el resto de la red le llamamos *s* y al nodo destino al que llega el flujo del resto de la red le llamamos *t*); al flujo en la rama *(i, j)* le llamamos x_{ij}, y a la capacidad de la rama *(i, j)*, u_{ij}, entonces el algoritmo es como sigue (Ford y Fulkerson, 1962):

Durante la rutina A, cada nodo está en uno de tres estados: no-etiquetado, etiquetado y no-rastreado, o etiquetado y rastreado. Al principio todos los nodos están en el estado no-etiquetado.

Rutina A (proceso de etiquetado)

Primero, la fuente *s* recibe la etiqueta $(*^+, \varepsilon(s) = \infty)$. (La fuente *s* está ahora etiquetada y no-rastreada; todos los demás nodos están no-etiquetados.) En el paso general seleccione cualquier nodo etiquetado y no-rastreado *i*. Supóngase que su etiqueta es $(\ell^{\pm}, \varepsilon(i))$. A todos los nodos *j* que están no-etiquetados y tales que el flujo $x_{ij} < u_{ij}$, asígnele la etiqueta $(i^+, \varepsilon(j))$, donde

$$\varepsilon(j) = min[\varepsilon(i), u_{ij} - x_{ij}] \qquad (1)$$

(Dichas nodos *j* ahora están etiquetados y no-rastreados). A todos los nodos *j* que ahora están no-etiquetados, y tales que $x_{ji} > 0$, asígneles la etiqueta $(i^-, \varepsilon(j))$, donde

$$\varepsilon(j) = min[\varepsilon(i), x_{ji}] \qquad (2)$$

(Los nodos *j* ahora están etiquetados y no-rastreados; *i* está etiquetado y rastreado).

Repítase el paso gen eral hasta que una de dos: el nodo destino *t* esté etiquetado y no-rastreado

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

o hasta que no sea posible asignar más etiquetas estando el nodo destino t no-etiquetado. En el primer caso ejecute la rutina B, en el segundo caso termina el algoritmo.

Rutina B (cambio de flujo)

El nodo destino t ha recibido la etiqueta $(j^{\pm}, \varepsilon(t))$. Si t está etiquetado $(j^{+}, \varepsilon(t))$, reemplace x_{jt} por $x_{jt} + \varepsilon(t)$; si t está etiquetada $(j^{-}, \varepsilon(t))$ reemplace x_{tj} por $x_{tj} - \varepsilon(t)$. En cualquier caso, en seguida hay que ponerle atención al nodo j. En general, si j está etiquetado $(i^{+}, \varepsilon(i))$ reemplace x_{ij} por $x_{ij} + \varepsilon(t)$; y si está etiquetado $(x^{-}, \varepsilon(j))$ reemplace x_{ji} por $x_{ji} - \varepsilon(t)$ y siga con el nodo i. Continúe de la misma manera hasta que alcance el nodo s. Elimine las etiquetas y regrese a la Rutina A.

El proceso de etiquetado es una búsqueda sistemática de una vereda aumentante del flujo del origen s al destino t. Se lleva información en las etiquetas de modo que cuando se etiqueta al destino t (al evento se le llama perforación), se pueden cambiar los flujos con facilidad. Si por otra parte, la Rutina A termina y el nodo destino t no ha sido etiquetado (no-perforación), el flujo es máximo y los arcos que van de los nodos etiquetados a los nodos no-etiquetados son un corte mínimo, cuyo valor es igual a la suma de las capacidades de las ramas que parten de un nodo etiquetado y llegan a un nodo no-etiquetado.

Descripción de la red para uso interno de la computadora

Hay varias maneras de describir una red para uso interno en la computadora. Para redes densamente conectadas una descripción eficiente involucra la *matriz de incidencia* (Ahuja *et al*., 1993). Dadas las características de Logo, se utilizarán las listas de *predecesores y sucesores** de cada nodo. La ventaja de esta descripción es que solamente se utiliza la memoria requerida para las ramas existentes y no para todas las posibles, además de que se adapta muy bien a la estructura de datos *lista* con la que cuenta el lenguaje Logo. En vez de explicar la descripción en abstracto, es más fácil explicarla ejemplificando para la red de la figura 1. La red de la misma figura queda completamente especificada si se da:

1. El número de nodos.
2. El número de ramas.
3. El número del nodo origen.
4. El número del nodo destino.
5. Para cada rama de la red se da el nodo inicial, el nodo final y la capacidad máxima de flujo de la rama. Estas cantidades se pueden meter a una lista y darse en el orden descrito.

Así, para la red de la figura 1 dicha lista es:

[6 9 1 6 1 3 3 1 2 7 2 3 1 2 4 6 3 5 8 4 3 3 4 6 2 5 4 2 5 6 8]

Estructuras de datos para los sucesores y predecesores

Para aplicar el algoritmo de Ford y Fulkerson se necesita recorrer uno a uno los sucesores y predecesores de un nodo, como paso preliminar se armarán listas de predecesores y sucesores para cada uno de los nodos. Para ello, se utilizará un vector (arreglo unidimensional) con una componente por nodo, cuyas componentes son listas de sucesores o predecesores. El lenguaje Logo no maneja arreglos; la manera como se implementarán los vectores es creando variables individuales con nombres que incluyan los índices. Así por ejemplo, llamamos *suc1* a la primera componente, *suc2* a la segunda, ... , *suc6* a la sexta, si queremos manejar un vector *suc(i), i = 1, ... , 6*. Cada una de estas componentes será una lista con los sucesores del nodo, cuyo índice es el número al final del nombre. Esta estratagema, que sería impráctica manejarla en lenguajes como BASIC, Pascal o C, es práctica en Logo porque se puede manipular el nombre de la variable *nombre* en la instrucción

da *nombre valor*

* Son *sucesores* de un nodo i los nodos j a los que se puede llegar partiendo de i, viajando por una sola rama en el sentido de su flecha. Son *predecesores* de un nodo i los nodos j a los que se puede llegar partiendo de i viajando por una sola rama en el sentido contrario a su flecha.

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

En Logo *nombre* se puede construir por medio de otras instrucciones. Por ejemplo, en vez de *nombre* se puede escribir

(**palabra** "suc :indice)

y *palabra* se construye con las letras *suc* concatenadas con las cifras del valor (:) de *indice*. La estratagema que se describe es equivalente a que en los lenguajes BASIC, Pascal o C se tuvieran instrucciones de asignación del siguiente tipo:

"suc" + índice **:=** *expresión*

aquí del lado izquierdo de la asignación se está construyendo el nombre de la variable por concatenación. Esta instrucción no existe en los lenguajes mencionados, ya que en los tres se requiere que del lado izquierdo de una asignación aparezca literalmente el nombre legal de una variable.

Esta posibilidad existente en Logo (y en LISP) permite el estratagema para manejar arreglos sin que el lenguaje los soporte por medio de nombres adecuadamente seleccionados de variables simples.

El siguiente procedimiento **suc** utiliza la estratagema indicada para construir el arreglo de sucesores de los nodos.

```
para suc :datos
si vacia? :datos [alto]
da (palabra "suc pr :datos) pul pr mpr
:datos cosa (palabra "suc pr :datos)
da (palabra "f pr :datos ", pr mpr :datos) 0
da (palabra "u pr :datos ", pr mpr :datos)
pr mpr mpr :datos
suc mpr mpr mpr :datos
fin
```

Antes de ejecutar **suc** (en el procedimiento **datos** que se muestra más abajo) se le quitan a la lista *datos* (que se le pasa como parámetro al procedimiento **suc**), los cuatro primeros números, de modo que en *datos* se tiene para cada rama un trío con el nodo inicial, el nodo final y la capacidad de flujo máxima de la rama. La segunda línea de **suc** detiene el procedimiento (alto) cuando se vacía la lista *datos,* con lo cual se detiene la recursión de la última línea de **suc,** ya que el procedimiento se llama a sí mismo en la última línea. En la tercera línea de **suc** se le asigna (da) a la lista cuyo nombre (palabra) está formado por la concatenación de suc con el primer elemento (pr) de la lista *datos,* lo que se forma al agregar al final (pul) el segundo elemento (pr mpr) de *datos* al resultado (cosa) del objeto cuyo nombre (palabra) es "suc concatenada con el primer elemento de *datos.*

En las líneas 4 y 5 de **suc** se aprovecha que ya se está recorriendo la lista *datos* para formar e inicializar dos arreglos bidimensionales de flujos y capacidades de flujo máximo en las ramas. Se emplea la misma estratagema de ponerle a variables individuales nombres que incluyen los dos índices separados por una coma. A los flujos se le inicializa con ceros y a las capacidades se les ponen los valores tomados de la lista *datos.*

Todo lo que se especifica en el procedimiento **suc** se hace con los primeros tres elementos de la lista *datos.* Para hacer las operaciones con todos los demás se hace una llamada recursiva en la sexta línea cambiando el parámetro *datos* por la misma lista, a la cual se le eliminan sus primeros tres elementos. Esto se hace con la instrucción *mpr* que significa *menosprimero,* cuyo efecto sobre una lista es eliminar el primer elemento. Al aplicar la instrucción tres veces, el efecto es eliminar el primer trío de elementos.

En forma similar a como se procedió para formar el arreglo de sucesores, se forma un arreglo de predecesores con el procedimiento **pred.**

```
para pred :datos
si vacia? :datos [alto]
da (palabra "pred pr mpr :datos) pul pr
:datos cosa (palabra "pred pr mpr :datos)
pred mpr mpr mpr :datos
fin
```

Procedimientos preliminares para introducir datos e inicializar arreglos

Antes de llamar a **suc** y **pred** se ejecuta el procedimiento **datos**, cuya función es inicializar la lista *datos* que describe la red, luego tomar los

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

primeros cuatro elementos de la lista original *datos,* asignarle los primeros cuatro valores a las variables globales *nn, nr, ni, nf* que representan el número de nodos, número de ramas, nodo inicial (origen) y nodo final (destino) respectivamente, y eliminar estos cuatro elementos de la lista para que en la lista *datos* (que es un objeto global) queden exclusivamente los tríos de las ramas.

```
para datos
da "datos [6 9 1 6 1 3 3 1 2 7 2 3 1 2 4 6 3 5
8 4 3 3 4 6 2  5 4 2 5 6 8]
da "nn pr :datos
da "nr pr mpr :datos
da "ni pr mpr mpr :datos
da "nf pr mpr mpr mpr :datos
da "datos mpr mpr mpr mpr :datos
fin
```

A continuación con **initetiq** se inicializan las etiquetas, todas con (0, 0), excepto el nodo origen que recibirá la etiqueta (* $^{+}$, 9999). El número 9999 es el número más grande que utiliza LogoWriter y lo usamos en vez de infinito (∞).

```
para initetiq :n
si :n = 0 [da (palabra "e :ni) (lista "*+
9999) alto]
da (palabra "e :n) [0 0]
initetiq :n - 1
fin
```

Nótese como en **initetiq** el cambio de índices se logra con una llamada recursiva. El número de nodos se le proporciona al procedimiento por medio del parámetro *n.* Para inicializar los "arreglos" de listas de predecesores y sucesores se ejecuta el procedimiento **initpredsuc**, en el cual todas las listas iniciales están vacías.

```
para initpredsuc
da "n 1 repite :nn [da (palabra "suc :n) []
da (palabra "pred   :n) [] da "n :n + 1]
fin
```

Como se indicó antes, los "arreglos" *f* y *u* para los flujos en las ramas y las capacidades de flujo máximas, se inicializan en el procedimiento **suc.**

Procedimientos principales para calcular el flujo máximo

Ahora se diseñarán los procedimientos principales para implementar el algoritmo de Ford y Fulkerson para el cálculo del flujo máximo. Se utilizan cuatro procedimientos: **flujo, etiqsuc, etiqpred, incflujo.** El primero hace la prueba para determinar si ya se ha alcanzado el flujo máximo, borra las etiquetas al final de cada perforación y reinicializa un nuevo intento para encontrar una vereda que aumente el flujo y lleva el control de la lista de los nodos etiquetados y rastreados. Los dos siguientes procedimientos (**etiqsuc** y **etiqpred**) realizan el proceso de etiquetar a todos los sucesores y predecesores del nodo al que se le está dando servicio en la lista de nodos etiquetados y no rastreados. El procedimiento **incflujo** hace las operaciones necesarias para incrementar el flujo en las ramas adecuadas cuando hay una perforación.

Para llevar cuenta en la computadora de cuáles nodos están etiquetados y no rastreados y cuáles están etiquetados y rastreados, se utilizan dos listas: la que almacena para cada perforación los nodos etiquetados y rastreados se llama *ee* y es un objeto global. La que almacena en cada perforación los nodos etiquetados y no rastreados se llama *lista*, se le pasa como parámetro al procedimiento **flujo** y está vigente en la ejecución de los tres procedimientos **etiqsuc, etiqpred** e **incflujo.** A los elementos almacenados en *lista* se les va dando servicio para entonces, desde ellos etiquetar sucesores y predecesores. Para que un nodo se etiquete es necesario que no esté etiquetado, es decir, que no sea miembro ni de *ee* ni de *lista.* Una vez que desde un nodo se etiquetan sucesores y predecesores, al nodo se le elimina de *lista* y se le agrega a *ee.* Para eliminarlo se llama recursivamente a los procedimientos **etiqsuc** y **etiqpred** con el argumento *mpr lista* en vez de *lista.* Para agregar un nodo al final de una lista se utiliza la instrucción de Logo *pul elemento lista.*

```
para flujo :lista
si (y vacia? :lista no (ul cosa (palabra "e
:nf)) >  0)   [resultados altotodo]
si vacia? :lista [initetiq :nn da "ee []
flujo (lista :ni)]
```

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004


```
etiqsuc cosa (palabra "suc pr :lista) pr
:lista
etiqpred cosa (palabra "pred pr :lista) pr
:lista
da "ee pul pr :lista :ee
flujo mpr :lista
fin

para etiqsuc :x :n
si vacia? :x [alto]
si (y ((cosa (palabra "u :n ", pr :x)) -
(cosa (palabra "f :n ", pr :x))) > 0 no
miembro? pr :x :lista no miembro? pr     :x
:ee) [da (palabra "e pr :x) (lista (palabra
:n "+) (min ul cosa (palabra "e :n) (cosa
(palabra "u :n ", pr :x)) - (cosa (palabra
"f :n ", pr :x)))) da "lista pul   pr :x
:lista]
si (ul cosa (palabra "e :nf)) > 0 [incflujo
:nf ul cosa   (palabra "e :nf) initetiq :nn
da "ee [] flujo (lista :ni) alto]
etiqsuc mpr :x :n
fin

para etiqpred :y :m
si vacia? :y [alto]
si (y (cosa (palabra "f pr :y ", :m)) > 0 no
miembro? pr :y :lista no miembro? pr :y :ee)
[da (palabra "e pr :y) (lista (palabra :m
"-) (min ul cosa (palabra "e :m) (cosa
(palabra "f pr :y ", :m)))) da "lista pul pr
:y :lista]
si (ul cosa (palabra "e :nf)) > 0 [incflujo
:nf ul cosa   (palabra "e :nf) initetiq :nn
da "ee [] flujo (lista :ni) alto]
etiqpred mpr :y :m
fin

para incflujo :nnn :ff
si :nnn = :ni [alto]
si (ul pr cosa (palabra "e :nnn)) = "+ [da
(palabra "f mul pr cosa (palabra "e :nnn) ",
:nnn) :ff + cosa (palabra "f mul pr cosa
(palabra "e :nnn) ", :nnn)]
si (ul pr cosa (palabra "e :nnn)) = "- [da
(palabra "f :nnn ", mul pr cosa (palabra "e
:nnn))  cosa (palabra "f :nnn     ", mul pr
cosa (palabra "e :nnn)) - :ff]
incflujo mul pr cosa (palabra "e :nnn) :ff
fin
```

En los procedimientos **etiqsuc** y **etiqpred** se utiliza la función **min,** ya que el algoritmo de Ford y Fulkerson lo requiere. La función **min** reporta (o regresa) el mínimo de sus dos argumentos numéricos. Como LogoWriter no cuenta con esta función, ésta se ha implementado como sigue:

```
para min :a :b
si :a < :b [re :a]
re :b
fin
```

El procedimiento **etiqsuc** etiqueta a los sucesores del nodo al que se está dando servicio en la *lista*; **etiqpred** hace lo correspondiente para los predecesores. Detecta cuando el nodo destino recibe una etiqueta y llama al procedimiento **incflujo** cuando esto sucede. Controla que al regreso de esta llamada se borren y reinicialicen la lista *ee,* asi como las etiquetas, y se llame de nuevo al procedimiento **flujo** para dar servicio al primer elemento en la *lista.* El procedimiento **incflujo** se encarga de incrementar (o decrementar cuando la primera parte de la etiqueta tiene signo menos) los flujos en todas las ramas que forman una vereda del origen al destino. Para encontrar la vereda en sentido contrario, comenzando con el destino, se va fijando en las primeras partes de los nodos, cada una de las cuales apuntan al nodo etiquetador.

Arranque del programa y generación de resultados

En los procedimientos **ff** y **calc** se hacen las llamadas a los procedimientos que arrancan todo el proceso para calcular los flujos máximos. El procedimiento **ff** finalmente llama a los procedimientos **datos, initpredsuc, suc, pred** y **calc.** Todos, menos **calc,** han sido explicados anteriormente. El procedimiento **calc** inicializa la lista *ee,* llama al procedimiento que inicializa los "arreglos" de etiquetas y para arrancar los cálculos de flujo máximo, finalmente llama al procedimiento **flujo** para realizar los cálculos. En la llamada a **flujo** utiliza como parámetro *lista,* la misma que cuenta con un solo elemento consistente en el nodo origen de la red, *ni.* Cuando la

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

llamada a **flujo** se detiene, se llama al procedimiento **resultados** , que imprime los flujos en las ramas que producen el flujo máximo, dando para cada rama de la red su nodo inicial, final y el valor del flujo en la rama, así como el valor del flujo máximo entre el nodo origen, destino y los nodos que especifican el corte mínimo. Para calcular los flujos máximos entre un par de nodos en una red, el usuario solamente debe, desde el editor de procedimientos de LogoWriter, colocar en el procedimiento **datos** la lista que de scribe la red y desde el centro de mandos teclear **ff** seguido de la tecla de retorno. A continuación, se muestran los procedimientos **ff**, **calc** y **resultados** .

```
para ff
datos
initpredsuc
suc :datos
pred :datos
calc
fin

para calc
da “ee []
initetiq :nn
flujo (lista :ni)
fin

para resultados
datos
(es “RESULTADOS) (es ” ) (inserta “I) tab
(inserta ”F) tab    (es “flujo) (es ” )
da “fff 0 repite :nr [(inserta pr :datos)
tab (inserta pr mpr  :datos) tab (es cosa
(palabra ”f pr :datos “, pr mpr    :datos))
si (pr :datos) = :ni [da ”fff :fff + cosa
(palabra “f pr :datos ”, pr mpr :datos)] da
“datos mpr    mpr mpr :datos]
(es “ ) (es ”|FLUJO MAXIMO DE| :ni “A :nf ”=
:fff)
(es “ ) (es ”|CORTE MINIMO {S} =| :ee)
fin
```

Ahora se muestra lo que producen los programas con la información del listado de **datos** que se anotó arriba, el cual corresponde a la red de la figura 1. El resultado se logra tecleando en el centro de mandos **ff** seguido de la tecla de re torno.

```
RESULTADOS

I          F       flujo

1          3       3
1          2       6
2          3       1
2          4       5
3          5       7
4          3       3
4          6       2
5          4       0
5          6       7

FLUJO MAXIMO DE 1 A 6 = 7
CORTE MINIMO {S} = 1 2 4
```

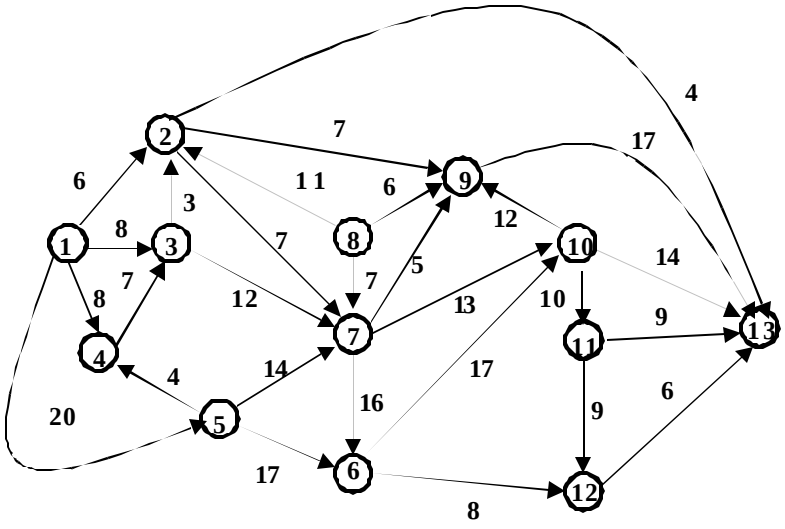

Figura 2.

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

Ejemplo adicional

Para probar el programa con una red un poco mayor, se resolverá un ejemplo tomado de (Jensen y Barnes, 1980). La red se muestra en la figura 2. Junto a cada rama aparecen sus capacidades de flujo máximas. Si se toma como nodo origen el nodo 1 y como nodo destino el 13, notamos que la red tiene 13 nodos y 28 ramas. La representación de la red en forma de lista es la siguiente:

```
[13 28 1 13 1 2 6 1 3 8 1 4 8 1 5 20 2 7 7 2 9
7 2 13 4 3 2 3 3 7 12 4 3 7 5 4 4 5 6 17 5 7
14 6 10 17 6 12 8 7 6 16 7 9 5 7 10 13 8 2 11
8 7 7 8 9 6 9 13 17 10 9 12 10 11 10 10 13 14
11 12 9 11 13 9 12 13 6]
```

Esta lista se introduce en el procedimiento **datos** con la instrucción **da "datos *lista,*** ya sea en vez de la que actualmente aparece o a continuación (pues en este último caso la asignación simplemente sobreescribe en *lista* lo dejado por la anterior instrucción). Luego se teclea **ff** seguido de la tecla de retorno. Los resultados que se obtienen se muestran a continuación.

```
RESULTADOS

I          F       flujo

1          2       6
1          3       8
1          4       7
1          5       20
2          7       0
2          9       5
2          13      4
3          2       3
3          7       12
4          3       7
5          4       0
5          6       17
5          7       3
6          10      14
6          12      6
7          6       3
7          9       5
7          10      7
8          2       0
8          7       0
8          9       0
9          13      17
10         9       7
10         11      0
10         13      14
11         12      0
11         13      0
12         13      6

FLUJO MAXIMO DE 1 A 13 = 41

CORTE MINIMO {S} = 1 4
```

Revisando los resultados constatamos:

1) que se satisfacen los límites máximos sobre todos los flujos,

2) que se satisface la Ley de Conservación de Flujos de Kirchhoff en cada uno de los nodos habiendo un exceso de flujo que sale del nodo origen igual al exceso de flujo que llega al nodo destino, ambos con valor igual a la capacidad o valor del corte mínimo, el cual corresponde a las ramas que atraviesan la superficie gaussiana que encierra los nodos 1 y 4. La teoría de flujos nos dice que basta que se satisfagan las restricciones máximas de flujo en todas las ramas, la ley de conservación de flujo en todos los nodos y que el flujo entre origen y destino sea igual al valor de algún corte con respecto a origen y destino para que dicha distribución de flujos corresponda a un flujo máximo entre origen y destino. De esta manera, se tiene la certeza de que la solución producida por el programa es correcta.

Instrumentación adicional para el programa

El programa desarrollado en este artículo tiene propósitos educativos, por ello, conviene agregarle instrumentos adicionales que sean útiles en la de teoría de flujos en redes, en general y de la enseñanza del algoritmo de Ford y Fulkerson para el cálculo de flujos máximos, en particular. Ante todo, conviene que el alumno pueda ver las estructuras de datos en lugares estratégicos del programa para comparar con lo que el alumno haría manualmente. Algunas estructuras de datos

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

no cambian du rante todo el proceso, por lo que se pueden imprimir una vez. Este es el caso de las listas de predecesores y sucesores y las variables nn, nr, ni, nf con la cantidad de nodos, ramas, el nodo origen y el nodo destino, así como las capacidades máximas de flujo de las ramas y sus nodos inicial y final.

Las cantidades que varían durante el curso de la aplicación del algoritmo son: las etiquetas de los nodos, los flujos en las ramas, el flujo total entre origen y destino.

Se utilizarán los siguientes procedimientos auxiliares: **n, p, s, c, e, f, ee, li,** los cuales imprimen respectivamente las cantidades de nodos, ramas, nodo inicial y final; las listas de predecesores, las listas de sucesores, las capacidades de flujo máximo en las ramas, las etiquetas de los nodos, los flujos en las ramas obtenidos hasta el momento y el flujo total de origen a destino obtenido hasta el momento; el contenido de la lista **ee** de nodos etiquetados y rastreados hasta el momento; y el contenido de la lista *lista* con los nodos etiquetados y no rastreados hasta el momento.

Los listados de los procedimientos auxiliares son:

```
para n
(es “nn “= :nn “nr “= :nr “ni “= :ni “nf “= :nf)
fin

para p
(inserta “Nodo) tab (es “|Lista Predecesores|)
da “i 1 repite :nn [(inserta :i) tab (es cosa (palabra “pred :i)) da “i :i + 1]
fin

para s
(inserta “Nodo) tab (es “|Lista Sucesores|)
da “i 1 repite :nn [(inserta :i) tab (es cosa (palabra “suc :i)) da “i :i + 1]
fin

para c
datos
(es “|Capacidades Maximas de Flujo|)
(inserta “I) tab (inserta “F) tab (es “|capacidad máxima|)
da “i 1 repite :nr [(inserta pr :datos) tab (inserta pr mpr :datos) tab (es cosa (palabra “u pr :datos “, pr mpr :datos))
da “datos mpr mpr mpr :datos]
fin

para e
(es “Etiquetas)
(inserta “Nodo) tab (es “Etiqueta)
da “i 1 repite :nn [(inserta :i) tab (es cosa (palabra “e :i)) da “i :i + 1]
fin

para f
datos
(es “|Flujos en Ramas|)
(inserta “I) tab (inserta “F) tab (es “flujo)
repite :nr [(inserta pr :datos) tab (inserta pr mpr :datos) tab (es cosa (palabra “f pr :datos “, pr mpr :datos)) da “datos mpr mpr mpr :datos]
fin

para ee
(es “ee “= :ee)
fin

para li
(es “li “= :lista)
fin
```

Se pueden insertar llamadas a los procedimientos en lugares estratégicos del programa para visualizar el proceso algorítmico y la manera como cambian los resultados intermedios.

A con tinuación se in dica una posibilidad:

Llamadas a **n**: En el procedimiento **calc** inmediatamente antes de **flujo (lista :ni)**
Llamadas a **p**: Inmediatamente después de la llamada a **n**.
Llamadas a **s**: Inmediatamente después de la llamada a **p**.
Llamadas a **c**: Inmediatamente después de la llamada a **s**.

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

Llamadas a **e**: En el procedimiento **etiqsuc** inmediatamente después del "**[**" en la línea que comienza ***si (ul cosa*** etc. En el procedimiento **etiqpred** inmediatamente después del "**[**" en la línea que comienza ***si (ul cosa*** etc.

Llamadas a **f**: En el procedimiento **incflujo** inmediatamente después del "**[**" en la segunda línea que comienza **si :nn =** etc.

Llamadas a **ee**: En el procedimiento **flujo** inmediatamente antes de la penúltima línea que dice **flujo mpr :lista.**

Llamadas a **li**: En el procedimiento **flujo** inmediatamente antes de la línea que comienza **etiqsuc cosa** etc.

A continuación, se muestra un segmento corto de una parte intermedia de la salida producida por el programa cuando está instrumentado con los procedimientos listados arriba y con las llamadas colocadas en los lugares indicados. Los datos corresponden al la red de la figura 2.

```
li = 1
ee = 1
li = 2 3 4 5
Etiquetas
Nodo   Etiqueta
1      *+ 9999
2      1+ 6
3      1+ 8
4      1+ 8
5      1+ 20
6      0 0
7      2+ 6
8      0 0
9      2+ 6
10     0 0
11     0 0
12     0 0
13     2+ 4
Flujos en Ramas
I      F      flujo
1      2      4
1      3      0
1      4      0
1      5      0
2      7      0
2      9      0
2      13     4
3      2      0
3      7      0
4      3      0
5      4      0
5      6      0
5      7      0
6      10     0
6      12     0
7      6      0
7      9      0
7      10     0
8      2      0
8      7      0
8      9      0
9      13     0
10     9      0
10     11     0
10     13     0
11     12     0
11     13     0
12     13     0
li = 1
ee = 1
li = 2 3 4 5
ee = 1 2
li = 3 4 5 7 9
ee = 1 2 3
li = 4 5 7 9
ee = 1 2 3 4
li = 5 7 9
ee = 1 2 3 4 5
li = 7 9 6
ee = 1 2 3 4 5 7
li = 9 6 10
```

Conclusiones

Se ha presentado el diseño y realización de un programa en el lenguaje Logo para el cálculo de flujo máximo en una red orientada con capacidades de flujo máximo en las ramas. El programa se basa en el algoritmo de Ford y Fulkerson. La estrategia encuentra veredas aumentantes de flujo en orden de número de ramas, por lo que se puede garantizar la eficiencia cúbica del algoritmo, a diferencia del algoritmo original de Ford y Fulkerson que podría tardarse un tiempo exponencial (Ahuja *et al*., 1993). Se desarrolló el programa en Logo con el propósito de ilustrar el poderío de dicho lenguaje en aplicaciones en las que nor malmente se utilizan lenguajes como FORTRAN, Pas-

DOI: http://dx.doi.org/10.22201/fi.25940732e.2006.07n1.004

cal, C o BASIC, los cuales soportan arreglos uni y multidimensionales, así como algunos de ellos (Pascal y C), listas ligadas y apuntadores. Logo no soporta arreglos; sin em bargo, se puede implantar el equivalente a los mismos por medio de distintas vari ables para cada componente, incorporando los índices en el nombre de la vari able. En el artículo se muestra la manera de hacerlo. De esta forma, se puede ahorrar memoria e incluir solamente las variables que aparecen en la red, las cuales en aplicaciones re ales son un porcentaje muy bajo de las posibles. Por otra parte, los componentes de los "arreglos" que se manejan en el artículo, frecuentemente son listas de longitud vari able durante el curso del programa, y las listas si son estructuras de datos que maneja bien el lenguaje Logo. En este artículo se proporciona un listado completo del programa en LogoWriter con una discusión detallada de su implantación y dos redes resueltas en todo detalle. Aquí no se pretende enseñar LogoWriter. El lector que requiera ayuda con las instrucciones puede consultar Logo Computer Sys tems Inc., 1990. Para hacer más útil el programa en la educación, también se proporciona un juego de instrumentos educativos (procedimientos) para que el mae stro y estudiante puedan observar el curso del algoritmo, como los cambios en las estructuras de datos para poder comparar con lo que se obtendría al aplicar manualmente el algoritmo de Ford y Fulkerson.

El autor espera que después de leer este artículo, el lector quede convencido que el lenguaje Logo no es sólo un lenguaje para niños, sino un lenguaje completo que tiene algunas entradas simples, desde las cuales lo pueden abordar los niños pero que les puede servir para crecer intelectualmente y utilizarlo el resto de su vida en aplicaciones tan complejas como aquellas en las que utiliza otros lenguajes de computadora.

Referencias

Ahuja R.K., Magnanti T.L. y Orlin J.B. (1993). *Network Flows: Theory, Algo rithms, and Appli cations*. Prentice-Hall, Upper Saddle River, NJ.

Ford Jr. L.R. y Fulkerson D.R. (1962). *Flows in Networks*. Princeton University Press, Princeton, NJ.

Jensen P.A. y Barnes J.W. (1980). *Network Flow Programming.* John Wiley & Sons, Inc., New York, pp. 165.

Logo Computer Systems, Inc. (1990). *LogoWriter: Guía de Referencia.* Macrobit Editores, México.

Bibliografía sugerida

Cristofides N. (1975). *Graph Theory an Algo - rithmic Approach.* Academic Press, Inc., Londres.

Semblanza del autor

Marco Antonio Murray-Lasso. Realizó la licenciatura en ingeniería mecánica-eléctrica en la Facultad de Ingeniería de la UNAM. El Instituto de Tecnología de Massachussetts (MIT) le otorgó los grados de maestro en ciencias en ingeniería eléctrica y doctor en ciencias cibernéticas. En México, ha laborado como investigador en el Instituto de Ingeniería y como profesor en la Facultad de Ingeniería (UNAM) durante 44 años; en el extranjero, ha sido asesor de la NASA en diseño de circuitos por computadora para aplicaciones espaciales, investigador en los Laboratorios Bell, así como profesor de la Universidad Case Western Reserve y Newark College of Engineering, en los Estados Unidos. Fue el presidente fundador de la Academia Nacional de Ingeniería de México; vicepresidente y presidente del Consejo de Academias de Ingeniería y Ciencias Tecnológicas (organización mundial con sede en Washington que agrupa las Academias Nacionales de Ingeniería) y secretario de la Academia Mexicana de Ciencias. Actualmente es jefe de la Unidad de Enseñanza Auxiliada por Computadora de la Secretaría de Estudios de Posgrado de la Facultad de Ingeniería de la UNAM, investigador nacional, consejero educativo del MIT y consultor de la UNESCO, así como Presidente Fundador del Consejo de Honor de la Academia Mexicana de Ciencias, Artes, Tecnología y Humanidades.