Algoritmos voraces

January 20, 2011

- Utilización en problemas de optimización.
- Plantean la búsqueda de la solución a través de una serie de etapas.
- En cada etapa se toma una "decisión" (elección voraz) que nunca se reconsidera $\rightarrow$ Debe ser correcta.
- Son eficientes y fáciles de implementar.
- No pueden resolver todos los problemas de optimización.

Problema

Dada una cantidad C y un sistema monetario, encontrar mínimo número de monedas cuya suma es C.

Planteamiento:

- Se trata de un problema de optimización.
- Cada etapa consiste en la inclusion de una nueva moneda.
- La "decisión" (la nueva moneda a incluir) será la de mayor valor.
- Variables:
  - `valor_moneda[I]`: valor de la moneda I.
  - `cambio[I]`: número de monedas de tipo I.

Algoritmo

proc $devolver-cambio(cant, valor-moneda[1..N], cambio[1..N], error)$
  desde $i \leftarrow 1$ **hasta** n **hacer**
    $cambio[i] \leftarrow 0$
  fin desde
  $suma \leftarrow 0$ *//Suma de la cantidad del cambio*
  mientras $suma \neq cant \wedge \neg error$ **hacer**
    //Aún no hemos completado la cantidad
    $X \leftarrow mayor(valor-monedas, cant, suma, no-existe)$
    si $no-existe$ **entonces**
      $error \leftarrow cierto$
    si no
      $cambio[X]++$
      $suma \leftarrow suma + valor-moneda[X]$
    fin si
  fin mientras
fin proc

donde:

Algoritmo

fun $mayor(valor-moneda, cant, suma, no-existe)$
  $no-existe \leftarrow cierto$
  $maximo \leftarrow 0$
  $num-moneda \leftarrow 1$
  mientras $num-moneda \leq n$ **hacer**
    si $valor-moneda[num-moneda] + suma \leq cant$ **entonces**
      si $maximo < valor-moneda[num-moneda]$ **entonces**
        $maximo \leftarrow valor-moneda[num-moneda]$
        $no-existe \leftarrow falso$
        $quien \leftarrow num-moneda$
      fin si
    fin si
    $num-moneda++$
  fin mientras
  devolver $quien$
fin fun

Traza

$N = 3$
valor_monedas=$[10, 5, 1]$
cant $= 27$

	0	1	2	3	4	5
cambio	(0,0,0)	(1,0,0)	(2,0,0)	(2,1,0)	(2,1,1)	(2,1,2)
suma	0	10	20	25	26	27

Traza

$N = 3$
valor_monedas$=[11, 5, 1]$
cant $= 15$

	0	*1*	*2*	*3*	*4*	*5*
cambio	*(0,0,0)*	*(1,0,0)*	*(1,0,1)*	*(1,0,2)*	*(1,0,3)*	*(1,0,4)*
suma	*0*	*11*	*12*	*13*	*14*	*15*

$\Downarrow$
Solución incorrecta!!!.

Requisitos del sistema monetario para que funcione correctamente: $P, P^2, ..., P^M$ con $P > 0$ y $M > 0$

Elementos de la programación voraz y variables del programa:

Un conjunto de candidatos	Todas las monedas
Un conjunto de seleccionados	Las monedas del cambio
Una función de selección	Elegir la moneda de valor más alto
Compatibilidad de la elección con la solución	No sobrepasar la cantidad
Determinar si hemos alcanzado la solución	$Suma = cant$
El valor de la optimización	$cambio[1..N]$

Problema

Dados N objetos con un peso y un valor conocidos, y dada una mochila con capacidad en peso W_t, encuentra la composición de la mochila cuyo valor sea máximo. Los objetos son fraccionables y moldeables.

Planteamiento:

- Es un problema de optimización.
- En cada etapa insertamos un objeto (o fracción) en la mochila.
- Posibles estrategias voraces (¿qué elemento elegir?):
  - El de menor peso.
  - El de mayor peso.
  - El de mayor proporción entre valor y peso.
- Esquema del programa:

$$maximizar \sum_{i=1}^{N} X_i V_i$$

sujeto a: $\sum_{i=1}^{N} X_i W_i \leq W_t$

Algoritmo

proc $mochila(w[1..N], v[1..N], W_t, x[1..N])$
  desde $i \leftarrow 1$ **hasta** n **hacer**
    $x[i] \leftarrow 0$
  fin desde
  $peso \leftarrow 0$
  mientras $peso < W_t$ **hacer**
    //Caben más objetos en la mochila
    $num - obj \leftarrow seleccionar(w, v, x)$
    si $peso + w[num - obj] \leq W_t$ **entonces**
      //Cabe entero
      $x[num - obj] \leftarrow 1$
      $peso \leftarrow peso + w[num - obj]$
    si no
      //Insertamos la fracción que quepa
      $x[num - obj] \leftarrow (W_t - peso)/w[num - obj]$
      $peso \leftarrow W_t$
    fin si
  fin mientras
fin proc

Traza

$N = 4$
$v = (100, 20, 50, 10)$
$w = (10, 100, 50, 2)$
$v/w = (10, 0.2, 1, 5)$
$W_t = 130$

	0	1	2	3	4
x[1..N]	(0,0,0,0)	(1,0,0,0)	(1,0,0,1)	(1,0,1,1)	(1,0.68,1,1)
Peso	0	10	12	62	130
Valor	0	100	110	160	173.6

Supongamos los objetos no fraccionables:
Ahora el código quedaría de la siguiente forma:

Algoritmo

si $peso + w[I] \leq W_t$ **entonces**
  //Cabe entero
  $x[num - obj] \leftarrow 1$
  $peso \leftarrow peso + w[num - obj]$
si no
  //Insertamos la fracción que quepa
  $x[num - obj] \leftarrow 0$
fin si

Traza

$N = 3$
$v = (11, 6, 6)$
$w = (5, 3, 3)$
$W_t = 6$

	0	*1*	*2*
x[1..N]	*(0,0,0)*	*(1,0,0)*	
Peso	*0*	*5*	
Valor	*0*	*11*	

$\Downarrow$
Erróneo!!!

Problema

Dado un tablero de ajedrez de tamaño $N \times N$ y un caballo en una casilla inicial, determina si es posible que un caballo pise cada casilla exactamente una vez.

Planteamiento:

- No es un problema de optimización.
- En cada etapa realizamos un movimiento del caballo.
- Estrategia (¿qué movimiento realizar?): mover el caballo a aquella posición en la cual domine el **menor** número de casillas no visitadas.
- Variables:
    - $T[X, Y]$: número del movimiento en el cual pasamos por X, Y.
    - Casilla no visitada: $T[X, Y] \neq 0$
    - Condición de finalización: $I = N^2$, donde I es el número de moviento realizado.

Una numeración de los movimientos del caballo

	2		3	
1				4
		C		
8				5
	7		6	

Algoritmo

proc $caballo(Tablero[1..N, 1..N], X, Y, existe - solucion)$
  desde $J \leftarrow 1$ **hasta** n **hacer**
    desde $K \leftarrow 1$ **hasta** n **hacer**
      $Tablero[J, K] \leftarrow 0$
    fin desde
  fin desde
  $fin \leftarrow falso$
  $I \leftarrow 1$
  mientras $I \leq N^2 \land \neg fin$ **hacer**
    //Movimiento I-esimo del caballo
    $Tablero[X, Y] \leftarrow I$
    $Generar - nuevo - movimiento(Tablero, X, Y, existe - mov)$
    si $I < N^2 \land \neg existe - mov$ **entonces**
      $fin \leftarrow cierto$
    si no
      si $I < N^2$ **entonces**
        $I \leftarrow I + 1$
      fin si
    fin si
  fin mientras
  si $I = N^2 \land \neg existe - mov$ **entonces**
    $existe - solucion \leftarrow cierto$
  si no
    $existe - solucion \leftarrow falso$
  fin si
fin proc

Algoritmo

proc $Generar-nuevo-movimiento(Tablero, X, Y, existe-mov)$
  $num-cas \leftarrow \infty$ *//Número de casillas accesibles*
  desde $k \leftarrow 1$ **hasta** 8 **hacer**
    si $salto(Tablero, k, x, y, nueva-x, nueva-y)$ **entonces**
      //Generamos el k-esimo salto del caballo
      $acc \leftarrow cuenta(Tablero, nueva-x, nueva-y)$
      //Numero de casillas accesibles desde: nueva-x, nueva-y
      si $acc > 0 \wedge acc < num-cas$ **entonces**
        $num-cas \leftarrow acc$
        $solx \leftarrow nueva-x$
        $soly \leftarrow nueva-y$
      fin si
    fin si
  fin desde
  $x \leftarrow solx$
  $y \leftarrow soly$
  si $um-cas < \infty$ **entonces**
    $existe-mov \leftarrow cierto$
  si no
    $existe-mov \leftarrow falso$
  fin si
fin proc

Algoritmo

fun $cuenta(Tablero, X, Y)$
  // Número de posiciones válidas desde X,Y
  $N - pos \leftarrow 0$
  desde $J \leftarrow 1$ **hasta** 8 **hacer**
    si $salto(Tablero, J, X, Y, Nx, Ny)$ **entonces**
      $N - pos \leftarrow N - pos + 1$
    fin si
  fin desde
  devolver $N - pos$
fin fun

Algoritmo

fun $salto(Tablero, J, X, Y, Nx, Ny)$
  casos
    $J = 1:$
      $Nx \leftarrow X - 2; Ny \leftarrow Y + 1$
    $J = 2:$
      $Nx \leftarrow X - 1; Ny \leftarrow Y + 2$
    ...
    $J = 8:$
      $Nx \leftarrow X - 2, Ny \leftarrow Y - 1$
  fin casos
  si $(1 \leq Nx) \wedge (Nx \leq N) \wedge (1 \leq Ny) \wedge (Ny \leq N) \wedge (T[Nx, Ny] = 0)$ **entonces**
    $OK \leftarrow cierto$
  si no
    $OK \leftarrow falso$
  fin si
  devolver OK
fin fun

Problema

Dado un grafo dirigido y un vértice, establecer el camino mínimo desde ese vértice al resto de los vértices del grafo.

Planteamiento:

- Sea el grafo $G = (V, A)$, donde V es el conjunto de los vértices y A es el conjunto de las aristas. Sea $L[1..N, 1..N]$ la matriz de adyacencia:

$$L[1..N, 1..N] \rightarrow \begin{cases} L[I, J] \geq 0, \text{ si existe la arista} \\ \\ L[I, J] = \infty, \text{si no existe la arista} \end{cases}$$

- Consideremos los siguientes conjuntos:
  - S: aquellos nodos para los cuales hemos calculado su camino mínimo
  - $C \equiv V \setminus S$.
- En cada etapa elegiremos un nodo de C y lo insertaremos en S.
- Inicialmente: $S = 1$.
- Al final: $S = V$.

Cuestión: ¿qué nodo elegir? (elección voraz):

- Sea el camino: $\widehat{v} = \{v_1, v_2, \ldots, v_{k-1}, v_k\}$, diremos que $\widehat{v}$ es especial si $\{v_1, v_2, \ldots, v_{k-1}\} \in S$.
- Para todo nodo de C existe un camino especial que lleva a él desde 1 (quizás de longitud infinita).
- Sea $D[2..N] \rightarrow D[I]$: la longitud del camino especial más corto al nodo I donde $I \in C$.
- Inicialmente: $S = \{1\} \Rightarrow D[J] = L[1, J]\ \forall J$.
- Para aquel nodo de C que minimice el valor de D, el valor del camino especial más corto coincidirá con la longitud del camino mínimo.
- Una vez que hemos cogido un nodo de C, actualizamos D y repetimos el proceso.

Algoritmo

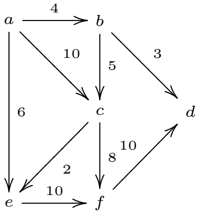
proc $camino(L[1..N, 1..N], D[2..N])$
  //El nodo 1 es el origen
  desde $J \leftarrow 2$ **hasta** N **hacer**
    $D[J] \leftarrow L[1, J]$
  fin desde
  $C \leftarrow 2, \ldots, N$
  desde $J \leftarrow 1$ **hasta** $N - 2$ **hacer**
    //Determinamos aquel v de C que minimiza D
    $v \leftarrow minimo(D, C)$
    //Lo insertamos en S
    $C \leftarrow C \setminus \{v\}$
    //Actualizamos C
    desde $w \leftarrow 1$ **hasta** N **hacer**
      si $w \in C$ **entonces**
        $D[w] \leftarrow min\{D[w], D[v] + L[v, w]\}$
      fin si
    fin desde
  fin desde
fin proc

Versión generalizada:

Algoritmo

proc $camino(L[1..N, 1..N], ini, D[2..N])$
  //El nodo ini es el origen
  desde $J \leftarrow 1$ **hasta** N **hacer**
    $D[J] \leftarrow L[ini, J]$
  fin desde
  $C \leftarrow V \setminus ini$
  desde $J \leftarrow 1$ **hasta** $N - 2$ **hacer**
    //Determinamos aquel v de C que minimiza D
    $v \leftarrow minimo(D, C)$
    //Lo insertamos en S
    $C \leftarrow C \setminus \{v\}$
    //Actualizamos C
    desde $w \leftarrow 1$ **hasta** N **hacer**
      si $w \in C$ **entonces**
        $D[w] \leftarrow min\{D[w], D[v] + L[v, w]\}$
      fin si
    fin desde
  fin desde
fin proc

Traza:



	0	1	2	3	4
D[1..N]	(0,4,∞,10,6,∞)	(0,4,9,7,6,∞)	(0,4,9,7,6,16)	(0,4,9,7,6,16)	(0,4,9,7,6,16)
C	(b,c,d,e,f)	(c,d,e,f)	(c,d,f)	(d,f)	(f)

Solución: D=(0,4,7,9,6,16).

Definición

Sea un grafo conexo no dirigido, $G = (V, A)$, entonces $T = (V, \hat{A})$ será un árbol de recubrimiento si T es conexo y $\hat{A} \subseteq A$.
$T = (V, \hat{A})$ será un árbol de recubrimiento mínimo si, además, la suma de las aristas $\hat{A}$ es mínima.

Problema

Sea un grafo conexo no dirigido, $G = (V, A)$, determina su árbol de recubrimiento mínimo.

Características voraces:

- Conjunto de candidatos: todas las aristas del grafo.
- Conjunto de seleccionados: aquellas que recubren el grafo con longitud mínima.
- Función a optimizar: la longitud de las aristas seleccionadas.

Planteamiento:

- Consideremos el árbol de recubrimiento parcial: $F = (R, T)$, donde:
  - R: es el conjunto de los vértices para los cuales tenemos calculado un árbol de recubrimiento mínimo.
  - $C \equiv V \setminus R$.
  - T: es el conjunto de aristas de ese árbol de recubrimiento mínimo.
- En cada etapa introduciremos un nodo de C en R y una arista $\leftrightarrow$ Nuestro árbol de recubrimiento crecerá.
- Inicialmente: $R = \{1\}$ y $T = \emptyset$.
- Finalizará cuando $R = V \Rightarrow T$ es el árbol de recubrimiento de $G = (V, A)$.

- Propiedad: Sea $F = (R, T)$ un árbol de recubrimiento mínimo parcial y sea la arista (v, w) de menor longitud donde $v \in R$ y $w \notin R$. Entonces $(R \cup \{w\}, T \cup \{(v, w)\})$ es un árbol de recubrimiento mínimo.
- Esquema del algoritmo:

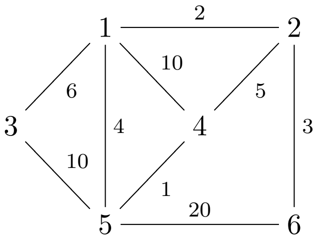
  $T \leftarrow \emptyset$
  $R \leftarrow azar(G(V, A))$
  mientras $R \neq V$ **hacer**
    buscar arista (v,w) de longitud minima
    $T \leftarrow T \cup \{(v, w)\}$
    $R \leftarrow R \cup \{w\}$
  fin mientras

- Variables utilizadas:
  - $max - proximo[1..N] \rightarrow max - proximo[i]$: vértice de R más próximo a i.
  - $dis - min[1..N] \rightarrow dis - min[i]$: distancia al vértice de R más próximo a i.

Algoritmo

proc $Prim(L[1..N, 1..N], T)$
  $T \leftarrow \emptyset$
  desde $J \leftarrow 2$ **hasta** N **hacer**
    $mas - proximo[J] \leftarrow 1$
    $dis - min[J] \leftarrow L[J, 1]$
  fin desde
  desde $I \leftarrow 1$ **hasta** $N - 1$ **hacer**
    $minimo \leftarrow \infty$
    desde $J \leftarrow 2$ **hasta** N **hacer**
      si $0 \leq dis - min[J] < minimo$ **entonces**
        $minimo \leftarrow dis - min[J]$
        $K \leftarrow J$
      fin si
    fin desde
    $T \leftarrow T \cup \{mas - proximo[K], K\}$
    $dis - min[K] \leftarrow -1$
    //Actualizamos
    desde $J \leftarrow 2$ **hasta** N **hacer**
      si $L[J, K] < dis - min[J]$ **entonces**
        $dis - min[J] \leftarrow L[J, K]$
        $mas - proximo[J] \leftarrow K$
      fin si
    fin desde
  fin desde
fin proc

Traza:



	0	1	2
R	{1}	{1,2}	{1,2,6}
T	$\emptyset$	{(1,2)}	{(1,2),(2,6)}
mas-proximo	(-,1,1,1,1,1)	(-,1,1,2,1,2)	(-,1,1,2,1,2)
dis-min	(-,2,6,10,4,∞)	(-,-1,6,5,4,3)	(-,-1,6,5,4,-1)

	3	4	5
R	{1,2,5,6}	{1,2,4,5,6}	{1,2,3,4,5,6}
T	{(1,2),(2,6),(1,5)}	{(1,2),(2,6),(1,5),(4,5)}	{(1,2),(2,6),(1,5),(4,5),(3,1)}
mas-proximo	(-,1,1,5,1,2)	(-,1,1,5,1,2)	(-,1,1,5,1,2)
dis-min	(-,-1,6,1,-1,-1)	(-,-1,6,-1,-1,-1)	(-,-1,-1,-1,-1,-1)

Planteamiento:

- En cada momento suponemos que tenemos varios subárboles de recubrimiento mínimo parciales: $S_i = (N_i, A_i)$ donde $V = \cup N_i\ \forall i$.
- **Propiedad:** Si $\{(v, w)\}$ es la arista de menor longitud que cumple $v \in S_i$ y $w \in S_j (i \neq j)$ donde S_i, S_j son dos árboles de recubrimiento parciales, entonces $(N_i \cup N_j, A_i \cup A_j \cup \{(v, w)\})$ es un árbol de recubrimiento parcial.
- Inicialmente tendremos N subárboles de recubrimientos parciales ($\equiv$ componentes conexas): $S_k = (v_k, \emptyset)$.
- En cada etapa añadiremos la arista de menor longitud que una vértices de diferentes componentes conexas y las fusionaremos.
- Fusionar dos componentes conexas consiste en reetiquetar una de ellas con la etiqueta de la otra.
- Al final tendremos una única componente conexa cuyas aristas son el árbol de recubrimiento mínimo.

Variables:

- El grafo inicial $G = (V, A) \rightarrow L[1..N, 1..N]$.
- $c[1..N] \rightarrow c[i]$: componente conexa a la que pertenece el nodo i.
- Inicialmente: $c[i] = i$
- Al final $c[i] = 1$ (dependiendo de la política de etiquetado).

Algoritmo

proc $kruskal(L[1..N, 1..N], T)$
  $inicializar(comp - conexa[1..N])$
  $T \leftarrow \emptyset$
  $ordenar - aristas(L, aristas, num - aristas)$
  $I \leftarrow 0$
  mientras $\neg fin(comp - conexa) \wedge I < num - aristas$ **hacer**
    $I \leftarrow I + 1$
    $C_1 \leftarrow comp - conexa[arista[I].origen]$
    $C_2 \leftarrow comp - conexa[arista[I].destino]$
    si $C_1 \neq C_2$ **entonces**
      $fusionar - comp - conexas(comp - conexa, C_1, C_2)$
      $T \leftarrow T \cup (arista[I].origen, arista[I].destino)$
    fin si
  fin mientras
fin proc

donde:

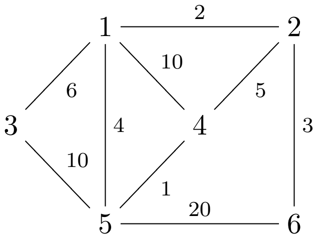
Algoritmo

proc $inicializar(comp-conexa[1..N])$
  $inicializar(comp-conexa[1..N])$
  desde $J \leftarrow 1$ **hasta** N **hacer**
    $comp-conexa[J] \leftarrow J$
  fin desde
fin proc
proc $fusionar-comp-conexas(comp-conexa[1..N], A, B)$
  si $A > B$ **entonces**
    $aux \leftarrow A$
    $A \leftarrow B$
    $B \leftarrow aux$
  fin si
  desde $J \leftarrow 1$ **hasta** N **hacer**
    si $comp-conexa[J] = B$ **entonces**
      $comp-conexa[J] \leftarrow A$
    fin si
  fin desde
fin proc

Algoritmo

proc $fin(comp - conexa)$
  $seguir \leftarrow cierto$
  mientras $I \leq N \wedge seguir$ **hacer**
    si $comp - conexa[I] \neq 1$ **entonces**
      $seguir \leftarrow falso$
    fin si
    $I++$
  fin mientras
  devolver $seguir$
fin proc

Traza:



	0	1	2	3	4	5
c[1..6]	(1,2,3,4,5,6)	(1,2,3,4,4,6)	(1,1,3,4,4,6)	(1,1,3,4,4,1)	(1,1,3,1,1,1)	(1,1,1,1,1,1)
T	$\emptyset$	{(4,5)}	{(4,5),(1,5)}	{(4,5),(1,5),(2,6)}	{(4,5),(1,2),(2,6),(1,5)}	{(4,5),(1,2),(2,6),(1,5),(1,3)}

Problema

En una región con una determinada capital deseamos comunicar el mayor número de ciudades mediante autopistas. La red debe ser tal que se pueda ir desde la capital a cualquier ciudad de la red exclusivamente por autopistas. Para ello disponemos de un presupuesto. Son conocidas todas las distancias entre ciudades. Además, existen carreteras ya construidas entre algunas de ellas. El coste por kilómetros de autopista, p, se reduce a una cuarta parte si hay una carretera ya construida. Mediante un algoritmo voraz que siempre llegue a la solución resuelve el problema.

Esquema de la solución:

- D[1..N,1..N]: distancias entre las ciudades.
- Carreteras[1..N,1..N] $\rightarrow$ carretera[I,J] : cierto si existe la carretera entre I y J.
- p: precio por kilómetro de la autopista.
- red: tramos de autopistas realizados.

Algoritmo

proc
$construir - autopistas(D[1..N, 1..N], Carreteras[1..N, 1..N], p, presupuesto, T, capital)$
  $R \leftarrow \{capital\}$
  $T \leftarrow vacio$
  // Creación del grafo de costes:
  desde $I \leftarrow 1$ **hasta** N **hacer**
    desde $desdeJ \leftarrow 1$ **hasta** N **hacer**
      si $carreteras[I, J]$ **entonces**
        $coste[I, J] \leftarrow D[I, J] * P/4$
      si no
        $coste[I, J] \leftarrow D[I, J] * P$
      fin si
    fin desde
  fin desde
  $dis - min[capital] \leftarrow -1$
  desde $J \leftarrow 1$ **hasta** N **hacer**
    si $J \neq capital$ **entonces**
      $mas - proximo[J] \leftarrow capital$
      $dis - min[J] \leftarrow coste[capital, J]$
    fin si
  fin desde
  ...
fin proc

Algoritmo

proc
$construir - autopistas(D[1..N, 1..N], Carreteras[1..N, 1..N], p, presupuesto, T, capital)$
 ...
 $Num \leftarrow 1$
 mientras $Num \leq N - 1 \wedge \neg fin$ **hacer**
  $minimo \leftarrow \infty$
  desde $J \leftarrow 1$ **hasta** N **hacer**
   si $0 \leq dis - min[J] < minimo$ **entonces**
    $minimo \leftarrow dis - min[J]$
    $K \leftarrow J$
   fin si
  fin desde
  si $coste[K, mas - proximo[K]] \leq presupuesto$ **entonces**
   $T \leftarrow T \cup \{mas - proximo[K], K\}$
   $dis - min[K] \leftarrow -1$
   //Actualizamos
   desde $J \leftarrow 1$ **hasta** N **hacer**
    si $coste[K, mas - proximo[J]] \leq presupuesto$ **entonces**
     $dis - min[J] \leftarrow coste[J, K]$
     $mas - proximo[J] \leftarrow K$
    fin si
   fin desde
   $presupuesto \leftarrow presupuesto - coste[K, mas - proximo[K]]$
  si no
   $fin \leftarrow cierto$
  fin si
 fin mientras
fin proc

Problema

En el año 82 d.C. Plinio El Joven se presentó por octava vez a las oposiciones de cuestor. El ejercicio que tenía que resolver era el siguiente. En una provincia romana amenazada por el enemigo hay N ciudades. Cada ciudad de la región no puede hacer por sí sola frente al enemigo pero puede resistir cierto tiempo hasta que llegue la única legión que existe en la provincia. La estrategia para defender la región es la siguiente: colocar la legión en la ciudad adecuada de forma que sea capaz de socorrer el mayor número de ciudades, es decir, de llegar antes de que la ciudad caiga. Un ejemplo, una ciudad con un tiempo de resistencia grande podría estar situada lejos de la ciudad donde se encuentra la legión. Diseñar un algoritmo, total o parcialmente voraz, que determine en qué ciudad colocar la legión. Datos:

- *El tiempo de viaje de la legión entre cada par de ciudades.*
- *El tiempo de resistencia de cada ciudad.*
- *El tiempo de alerta se supone cero.*

Algoritmo

proc $legiones(L[1..N, 1..N], Tr[1..N], Lugar)$
  $max \leftarrow 0$
  desde $origen \leftarrow 1$ **hasta** N **hacer**
    $num \leftarrow 0$
    $dijkstra(L, origen, D[1..N])$
    desde $q \leftarrow 1$ **hasta** N **hacer**
      si $Tr[q] \geq D[q]$ **entonces**
        $num++$
      fin si
    fin desde
    si $max < num$ **entonces**
      $lugar \leftarrow origen$
      $max \leftarrow num$
    fin si
  fin desde
fin proc