25. Algoritmos heurísticos

25.1 Concepto de heurística.

Se denomina heurística al arte de inventar. En programación se dice que un algoritmo es heurístico cuando la solución no se determina en forma directa, sino mediante ensayos, pruebas y reensayos.

El método consiste en generar candidatos de soluciones posibles de acuerdo a un patrón dado; luego los candidatos son sometidos a pruebas de acuerdo a un criterio que caracteriza a la solución. Si un candidato no es aceptado, se genera otro; y los pasos dados con el candidato anterior no se consideran. Es decir, existe inherentemente una vuelta atrás, para comenzar a generar un nuevo candidato; por esta razón, este tipo de algoritmo también se denomina "con vuelta atrás" (backtracking en inglés).

Estudiemos el siguiente problema, propuesto por Wirth y desarrollado por Dijkstra:

25.2 Aplicación. Generación de secuencias.

Se desea generar secuencias en orden 'alfabético' de tres caracteres '0','1' y '2', hasta obtener una secuencia de largo 100. Las secuencias generadas no deben contener subsecuencias adyacentes iguales.

Una lista de las primeras secuencias que cumplen es:

0
01
010
0102
01020
010201
0102010
0102012
.....

La lista indica lo que se entiende por ordenamiento alfabético, en este caso.

Observando las soluciones, cada una de ellas es una extensión (a lo menos en un dígito) de la anterior.

Denominaremos buena una secuencia que cumple la condición de no contener subsecuencias adyacentes iguales.

Si una secuencia de ensayo es buena, se la debe imprimir y 'extender' con un '0' para generar la candidata siguiente. Si ésta no se buena, se 'incrementa' para obtener la próxima. Se entiende por incrementar la operación de remover los dígitos finales iguales a '2' e incrementar en uno el último dígito resultante. Las operaciones de extender e incrementar aseguran la generación alfabética.

Además observamos que una nueva secuencia se obtiene de la anterior, según se ha explicado, lo cual hace necesario sólo revisar subsecuencias adyacentes; de las cuales, una debe contener al último dígito de la secuencia. Esta observación reduce, proporcionalmente, de m^2 a m el número de comparaciones para efectuar el test de secuencia buena.

Un aspecto importante es considerar el inicio y finalización del algoritmo, de tal manera que esas condiciones queden claramente establecidas.

Una posibilidad es :

```
secuencia ensayo:=secuencia vacía;
repeat
    extender secuencia de ensayo;
    while hayiguales do
      incremente secuencia de ensayo;
    imprima secuencia aceptada
until largo = 100
```

Las repeticiones anidadas evitan, en general, largas pruebas supérfluas.

Una alternativa que explica el algoritmo, sin tanto detalle, es la siguiente:

```
secuencia ensayo:=secuencia vacía;
repeat
  transforme secuencia hasta lograr la próxima solución;
  imprima secuencia aceptada
until largo = 100
```

Si se establece una estructura de datos, puede seguir descendiéndose en el nivel de detalle.

Con var s : array [1..n] of char;
m : 0..n;

La acción primitiva de extender la secuencia puede plantearse:

```
procedure extension;
begin
   m:=m+1 ; s[m]='0'
end;
```

La secuencia vacía es simplemente:

```
m:=0
```

La operación primitiva incrementar debe considerar que:

- Cuando s[m]='2' no hay sucesor.
- No se debe incrementar la secuencia vacía.
- A lo más debe acortarse en 2 componentes, ya una secuencia candidata se genera agregando sólo un caracter a una secuencia buena, la cual no debe contener 2 caracteres adyacentes iguales.

Un desarrollo posible es:

```
procedure incremente;
begin
  if s[m] < '2'
  then s[m]:=succ(s[m])
  else
  begin              {acortamiento o vuelta atrás}
    m:=m-1;
    if m > 0         {test no vacía}
    then
      if s[m] < '2'
      then s[m]:=succ(s[m])
      else
        begin        {último acortamiento}
          m:=m-1;
          if m > 0 then s[m]:=succ(s[m])
        end
```

```
    end
  end;
```

La dificultad del algoritmo es la prevención de llegar a la secuencia vacía. Si se conoce que ésto no sucederá; y por el contrario, si se conoce que es posible generar una secuencia de largo 100, puede escribirse más simplemente:

```
procedure incremente;
begin
  while s[m]='2' do m:=m-1;
  s[m]:=succ(s[m])
end;
```

El procedimiento imprima es trivial :

```
procedure imprima;
var  i : 1..n;
begin
  for i:=1 to m do write(s[i]);
  writeln
end;
```

Entonces el algoritmo queda:

```
m:=0;
repeat
   extension;
   while hayiguales {and (m>0)} do incremente;
   imprima
until (m=n) {or (m=0)};
```

Las condiciones para filtrar la secuencia vacía son supérfluas, en este caso. Ya que del ejemplo mostrado al inicio, se ve que la generación alfabética no lleva a la secuencia vacía.

25.3 Problemas propuestos.

a) Como aplicación puede desarrollarse una heurística para la generación en orden ascendente de 100 números de un conjunto m.

El conjunto m está definido según:

- El número 1 está en m
- Si x está en m, entonces: y = 2*x+1 ; z = 3*x+1 también pertenecen a m.

Los primeros elementos son:

m = {1 ; 3 ; 4 ; 7 ; 9 ; 10 ; ... }

b) Otros ejemplos clásicos son:

- Generación de los primeros n números primos.

- Ubicar todas las posiciones que pueden ocupar 8 reinas en un tablero de ajedrez, y tal que no se ataquen entre sí.

- Dado un entero n, determinar el menor número s que puede ser descompuesto en la suma de dos potencias de n, en al menos dos formas no triviales.

Ej : n=1 s = 2 = 0^1+ 2^1 = 1^1+ 1^1
n=2 s = 25 = 0^2+ 5^2 = 3^2+ 4^2
n=3 s =1729 = 1^3+12^3 = 9^3+10^3