ESQUEMAS DE TRATAMIENTO SECUENCIAL

Francisco Javier Oliver Villarroya

Francisco Javier Oliver Villarroya.– Vicedirector de la Politécnica (Informática).

1. INTRODUCCION AL TRATAMIENTO SECUENCIAL

Una solución correcta a muchos de los problemas que se plantean en programación se puede obtener de forma muy simple por aplicación de las técnicas de tratamiento secuencial.

En /LUCA85/ se denomina **secuencia** a cualquier tipo de información de naturaleza lineal a la cual se accede secuencialmente, es decir, accediendo inicialmente a un primer elemento y, a partir de él, a los demás a través de operaciones de obtención del sucesor de cada uno de ellos que, si existe, será único.

En esta definición subyace el concepto de tipo abstracto de datos cadena, formalizado en /GUTT76/, que nosotros, aunque no ecuacionalmente, estudiaremos con el mismo nivel de abstracción.

El término secuencia se usa indistintamente tanto si la información está representada en la memoria del computador, como si se trata de una abstracción del programador para razonar sobre sus algoritmos.

Introduciremos una constante de error para posibilitar un tratamiento sencillo para errores y situaciones de excepción. Somos conscientes de que la teoría matemática que subyace a este tratamiento tiene desagradables propiedades cuyo estudio sale fuera del alcance de este curso pues creemos que un tratamiento riguroso de esta cuestión requiere cierta madurez previa que aún no se ha alcanzado. Por ello elegimos seguir esta aproximación que, aunque no siendo matemáticamente sana, no desviará nuestra atención del tema que estamos abordando.

OPERACIONES

Las operaciones que definen el tipo de datos **secuencia** son:

– Constructivas:

secuencia vacía: ⟶ secuencia

comportamiento: (devuelve una secuencia vacía, sin elementos)

añadir: secuencia x elemento ⟶ secuencia

comportamiento: (devuelve una secuencia que es el resultado de añadir al final de la secuencia argumento un nuevo elemento).

– No constructivas:

primero: secuencia ⟶ elemento U (error)

comportamiento: (devuelve el primer elemento de la secuencia: un elemento distinguido de la misma; error si la secuencia es secuencia vacía)

último: secuencia ⟶ elemento U (error)

comportamiento: (devuelve el elemento que se encuentra al final de la secuencia; error si la secuencia es secuencia vacía).

sucesor : elementos x secuencia ⟶ elemento U (error)

comportamiento: (devuelve el elemento que sucede en la secuencia a un elemento dado; error si era el último. Todo elemento de la secuencia (salvo uno de ellos, el último) precede a uno de los demás (su sucesor). Esta relación entre pares de elementos es la que permite construir el acceso desde un elemento dado a su elemento sucesor).

resto : secuencia ⟶ secuencia

comportamiento: (devuelve una secuencia que es como la secuencia argumento pero sin su primer elemento).

vacía : secuencia ⟶ lógico

comportamiento: (se evalúa a cierto cuando la secuencia argumento es vacía y a falso en caso contrario)

Introducimos, por conveniencia, la siguiente operación relacionada con la constante de error:

indefinido : elemento U (error) ⟶ lógico

comportamiento: (devuelve verdad cuando el argumento es error; falso en caso contrario. Servirá para detectar que el recorrido de la secuencia ha terminado)

Las secuencias así definidas no autorizan el acceso más que a través del elemento que le precede: no se alcanza el elemento i-ésimo más que después de haber alcanzado sucesivamente los i-1 elementos que le preceden.

Se habla de **acceso secuencial** a los elementos de la secuencia y, por este motivo, utilizamos el término **tratamiento secuencial** para referirnos al tratamiento de secuencias.

2. ESQUEMAS DE TRATAMIENTO SECUENCIAL

Un **esquema algorítmico** es una descripción que abstrae los rasgos esenciales de un proceso.

Más formalmente, un esquema es una operación genérica que se implementa sobre las estructuras de datos a las que pertenecen los argumentos de entrada y salida de un problema concreto y, tal vez, sobre otros datos locales.

Estudiaremos distintas estrategias o métodos de diseño de algoritmos. Estos esquemas generales no deben entenderse como recetas para generar algoritmos sino como ideas generales aplicables sólo en ciertas ocasiones, que caracterizaremos de manera precisa.

Aprenderemos a instanciar los esquemas genéricos para un problema concreto, identificando las estructuras de datos y operaciones propias del esquema con las del problema, previa comprobación de que éstas satisfacen los requisitos necesarios.

Todos los esquemas se estudian de la misma forma:

1) Caracterización del dominio de aplicación.
2) Presentación del método general y sus posibles variantes.
3) Estudio general de su eficiencia.
4) Aplicaciones. El énfasis se hará, sobre todo, en los aspectos comunes de los algoritmos y no en la casuística propia de cada uno de ellos. Los aspectos específicos serán, a menudo, obviados y los ejemplos serán desarrollados con un alto nivel de abstracción. Será muy adecuado el uso de abstracciones de datos ya que manejaremos tipos de datos que no serán de momento implementados sino sólo manipulados a través de sus operaciones.

La **aplicación** del esquema pasa por varias etapas:

i) Estudiar su aplicabilidad.
ii) Identificar el problema con el esquema.

i) Estudiar su aplicabilidad.
ii) Identificar el problema con el esquema.
iii) Instanciación del esquema.
iv) Posibles mejoras para optimizar su legabilidad y/o eficiencia.

En esencia, existen dos tipos de **tratamiento secuencial**:

a) de **recorrido**. Aplicación de una misma operación a todos los elementos de la secuencia.

b) de **búsqueda**. Recorrido hasta encontrar un elemento que cumpla una cierta propiedad, conocida a priori.

2.1. Ambito de aplicación

Los esquemas de recorrido y búsqueda se aplican a gran variedad de problemas informáticos. Son métodos generales de resolución de problemas, aplicables siempre que dispongamos de una información secuencial y pretendamos enumerarla o buscar alguno de sus elementos.

Disponiendo de esquemas abstractos correctos para estos dos tratamientos (y sus casos particulares), todo problema caracterizado como tratamiento secuencial se puede resolver aplicando el esquema oportuno, previa identificación de las operaciones del esquema con sus correspondientes en nuestro problema.

2.2. PRESENTACION DE LOS ESQUEMAS:

Nuestro objetivo consiste en diseñar dos algoritmos genéricos que permitan avanzar sobre una secuencia hasta recorrerla por completo, en el primer caso y hasta que detectemos el elemento buscado, en el segundo.

El **esquema genérico para el recorrido de secuencias** es el siguiente:

ESQUEMA DE RECORRIDO

acción recorrido (s : secuencia);
var
– declaración de variables;
finvar
– acciones iniciales;

```
    mientras no indefinido (e) hacer
        tratar (e);
        e := sucesor (e, s);
    finmientras;
    acciones    finales;
finacción;
```
en donde:

Tratar (e: elemento) es la acción a realizar sobre cada elemento de la secuencia.

Como vemos, en el esquema genérico no se plantea cuál es la acción específica a realizar sobre cada elemento pues eso dependerá del uso que hagamos de él al resolver un problema en concreto.

El planteamiento para el esquema de búsqueda es análogo, excepto para la condición de continuación del recorrido por la secuencia y las acciones a realizar según el éxito o fracaso:

ESQUEMA DE BUSQUEDA:

```
acción búsqueda ( s : secuencia);
var
   –declaración de variables;
finvar;
   acciones    iniciales;
   e := primero (s);
   mientras    [No indefinido (e)] y cond
  [No propiedad (e)] hacer

   finmientras;
   opción
      indefinido (e) : fracaso (s);
      indefinido (e) : encontrado (e, s);
   finopción;
   acciones    finales;
finacción;
```
en donde:

propiedad (e : elemento) devuelve lógico
se evalúa a cierto si e cumple la propiedad del elemento buscado
falso si e no la cumple
indefinido si e es error

encontrado (e : elemento ; s : secuencia)
acciones caso de éxito en la búsqueda
fracaso (s: secuencia)
acciones caso de no éxito

La semántica de la función booleana propiedad impone la necesidad de utilizar la conectiva lógica y condicional, cuya definición es:

DEFINICION DE y condicional:

A y_ cond B = si A entonces B si no falso finsi

TABLA DE VERDAD		
A	B	A y_c B
F	F	F
F	T	F
F	U	F
T	F	F
T	T	T
T	U	U
U	F	F
U	T	F
U	U	F

A veces, se puede mantener el esquema de búsqueda con ŷ en vez de con y cond, siempre que la evaluación de propiedad (e) no produzca un **error** para e = indefinido.

Aún cuando no se da esta circunstancia, siempre es posible la transformación:

```
acción búsqueda (s : secuencia);
var
   –declaración de variables;
finvar;
   acciones   iniciales;
   e : = primero (s) ; seguir : = cierto;
   mientras No indefinido (e)  ŷ segura hacer

      si propiedad (e)
         entonces
            seguir : = falso
         si no
            e := sucesor (e, s)
      finsi;
   finmientras;
   opción
      indefinido (e) : fracaso (s);
      indefinido (e) : encontrado (e, s);

   finopción;
   acciones _ finales;
finacción;
```

VARIANTES DE LOS ESQUEMAS DE RECORRIDO Y BUSQUEDA

ESQUEMAS CON CENTINELA:

En algunos casos es posible disponer de un último elemento de la secuencia, ficticio, que actúa de centinela, avisando del final de la misma. Si ello ocurre, nunca se alcanza la situación excepcional de elemento indefinido. Con esta técnica, los esquemas se modifican y pasan a ser:

ESQUEMA DE RECORRIDO CON CENTINELA

```
acción recorrido_con_centinela (s: secuencia);
var
   – declaración de variables;
finvar;
     acciones iniciales;
    e := primero (s);
    mientras e <> último (s) hacer
        tratar (e);
        e := sucesor (e, s);
     finmientras;
     acciones_finales;
finacción;
```

ESQUEMA DE BUSQUEDA:

```
acción búsqueda_con_centinela (s : secuencia);
var
  – declaración de variables;
finvar;
     acciones iniciales:
     e := primero (s);
     mientras [e <> último (s)] y [No propiedad (e)] hacer
        e: =sucesor (e, s);
     finmientras;
     opción
        e = último (s) : fracaso (s);
        e <> último (s) : encontrado (e, s);
     finopción;
     acciones finales;
finacción;
```

ESTUDIO GENERAL DE LA EFICIENCIA DE LOS ESQUEMAS:

La eficiencia de ambos esquemas genéricos es similar ya que en los dos casos se realiza una iteración que, en el caso peor, se produce tantas veces como elementos tenga la secuencia bajo estudio. Por tanto, será de orden **n**, considerando **n** el número de elementos.

Evidentemente, además aparece el coste que produzcan las operaciones **tratar (e)** para el esquema de recorrido y **fracaso (s)** y **encontrado (e, s)** para el de búsqueda, que harán que el orden total sea **n** multiplicado por el orden de las operaciones en cada caso.

Así, el coste en tiempo de ejecución de estos dos algoritmos es, en principio, **O (n).**

2.3. Ejemplos de aplicación de los esquemas:

EJEMPLO N.º 1

ENUNCIADO 1:

Calcular los números de Fibonacci menores que F (con F > = 0), siendo la definición de número de Fibonacci:

f0 = 0
f1 = 1
fi = $f_{i-1} + f_{i-2}$ para i > = 2

es decir, los números de Fibonacci son los términos de la secuencia

$$0, 1, 1, 2, 3, 5, 8 \ldots \underbrace{(f_{i-1} + f_{i-2})}_{f_i} \ldots$$

i) ESTUDIO DE APLICABILIDAD:

Una solución correcta, como podemos comprobar, se obtendrá por aplicación del esquema de recorrido, pero en el que la secuencia subyacente no puede ser

f0, f1, f2,...

ya que, a partir de f_i, no es posible calcular su sucesor f_{i-1}

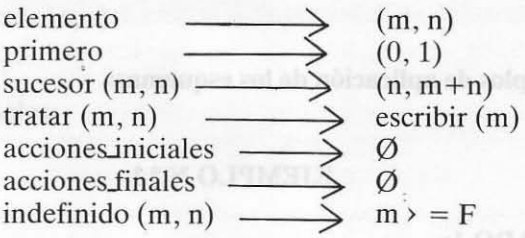
Dado que un término de la secuencia se obtiene en el caso general de la suma de los 2 anteriores, la secuencia correcta a la que debe

aplicarse el esquema está formada por los pares de términos:

$$(0, 1), (1, 1), (1, 2), \ldots (f_{i-1}, f_i), (f_i, f_{i+1})\ldots$$

ii) IDENTIFICACION DEL PROBLEMA CON EL ESQUEMA:

Para poder aplicar el esquema genérico de recorrido, lo primero que debemos hacer es la identificación del problema con el esquema, es decir, analizar en qué se traduce cada una de las operaciones que parametrizan el esquema para el caso que nos ocupa:



iii) INSTANCIACION:

Con todo ello, una solución al cálculo de los números de Fibonacci, obtenida por aplicación del esquema genérico de recorrido de una secuencia sería:

```
acción Fibonacci (F : entero);
var
   m, n, temp : entero;
finvar;
   m := 0 ; n := 1;
   mientras m < F hacer
      escribir (m);
      temp := m + n;
      m := n;
      n := temp
   finmientras
finacción;
```

EJEMPLO N.º 2

ENUNCIADO 2:

Descubrir si una palabra dada es palíndrome (es decir, igual a su imagen especular).

Una definición inductiva de palabra palíndrome es:

a) La palabra vacía es palíndrome
b) Un símbolo es una palabra palíndrome
c) Si p es palíndrome, y, s es un símbolo entonces se cumple que:

s.p.s es palíndrome

d) Sólo con palíndromes las palabras obtenidas por a), b) y c)

i) ESTUDIO DE APLICABILIDAD:

La secuencia implícita en el problema es la de los pares de índices

(izquierdo, derecho)

que caracterizan los elementos extremos de la palabra en curso:

(1, n), (2, n−1), (3, n−2)...

Se trata, en este caso, de aplicar el esquema de búsqueda para encontrar el primer par de índices correspondientes a extremos distintos.

El éxito de la búsqueda significa palabra no palíndrome.
El fracaso significa palabra palíndrome.

ii) IDENTIFICACION DEL PROBLEMA CON EL ESQUEMA:

La correspondencia entre tipos y operaciones del problema y las del esquema es, en este caso:

elemento	→	(i, d)
primero	→	(1, n)
sucesor (i, d)	→	(i+1, d−1)
indefinido (i, d)	→	i d
propiedad (i, d)	→	(p (i) ‹› p (d))
fracaso	→	devuelve cierto (palíndrome)
encontrado	→	devuelve falso (no palíndrome)

ii) INSTANCIACION:

En este caso, la solución obtenida será una función que devolverá cierto si la palabra cumple las características para ser palíndrome y falso en caso contrario.

Partimos de la declaración habitual:

```
tipo
    palabra = vector ( 1 : n) de carácter
fintipo;
var
    p: palabra
finvar;
```

que ya conocemos

Así, nos queda:

```
función palíndrome ( p : palabra) devuelve lógico;
var
    n, i, d : entero;
finvar;
    n := longitud (p);
i := 1; d := n;
    mientras (i < = d) y   condicional [p (i) = p (d)] hacer
    i := i + 1; d := d−1
    finmientras;
    opción
        i > de: devuelve cierto;
        i < = de: devuelve falso;
finopción;
finfunción;
```

Sin n > = 2, se puede sustituir y_condicional por ^ pues i y d se mantendrían en el rango 1... n

3. REFERENCIAS

/GOGU84/ Coguen, J.
Parameterized Programming
IEEE Trnas. on Soft. Eng., Vol SE-10, n.º 5, pp. 528-543.
Septiembre, 1984.

/GUTT76/ Guttag, J. Horowitz, E. Musser, D.
The Design of Data Types Specifications
Proc. Sec. Int. Conf. on Software Engineering
Octubre, 1976

/LUCA85/ Lucas, M. Peyrin, J.P. Scholl, P.C.
Algorítmica y Representación de Datos
Versión castellana de N. Castell, P. Botella y A. Llamosí
Ed. MASSON, 1985

/PEÑA86/ Peña, R.
Ingeniería de la Programación
Facultad de Ingeniería - Universidad de Antioquía
Programa de extensión académica CESET
1986

/SCHO78/ Scholl, P.C.
Le Traitement Sequentiel: une classe de problemes et une methode de construction de programmes
AFCET INFORMATIQUE
Noviembre, 1978