

UNED. Centro Asociado de VITORIA-GASTEIZ

Escuela Técnica de Informática

Programación II

Ejercicio ejemplo 4

Curso 1997/98

Los presentes apuntes han sido elaborados para su distribución gratuita a los alumnos de Programación II de Informática de Sistemas y Gestión del CENTRO ASOCIADO DE VITORIA-GASTEIZ de la Escuela Técnica de Informática de U.N.E.D.. Se ofrecen para uso personal de dichos alumnos pudiéndose reproducir y transmitir siempre que se mantenga esta nota y no se realice actividad de intercambio comercial de ningún tipo. En cualesquiera otras condiciones la copia, transmisión o almacenamiento por cualquier medio fotográfico, informático, telemático u otros requiere la autorización expresa del autor. Se agradecerá la comunicación de cualquier errata así como cualquier comentario o sugerencia.

(e-mail :jqc@jet.es)

© Jerónimo Quesada 1998

1. INTRODUCCIÓN

En estos apuntes se desarrolla un ejercicio de diseño y verificación de algoritmos con la intención de que sirva como un ejemplo más de aplicación de las técnicas expuestas en el programa de la asignatura. Se aconseja revisar anteriores ejercicios ejemplos de esta colección, en particular el publicado como EJER01 que, partiendo de un ejemplo muy simple, trata de introducir las técnicas de diseño y análisis recursivo e iterativo. Se hará referencia continua al texto base de la asignatura: "Diseño de Programas", R.Peña Marí, Ed. Prentice-Hall y se utilizará la notación definida en este texto.

2. ENUNCIADO

Se trata de diseñar y verificar un algoritmo que, dado un vector de N elementos, calcule el vector cuyos elementos son iguales a los del primero pero desplazados (circularmente) k índices a la derecha. Así para $N=5$ y $k=2$ dado el vector $a=[3,2,6,7,5]$ calculará el vector $b=[7,5,3,2,6]$. Se pide:

- a) Especificación rigurosa del problema
- b) Diseño y verificación de una versión recursiva, incluyendo análisis teórico del coste de ejecución.

- c) Diseño y verificación de una versión iterativa, incluyendo análisis teórico del coste de ejecución.
- d) Programa en MODULA 2 que implemente cada una de las versiones del algoritmo.
- e) Estudio empírico del coste de ejecución

3. ESPECIFICACIÓN

Definido el tipo *vect* como **tipo** *vect*= **vector** [1..N] **de entero**
una especificación para el algoritmo puede ser:

$$\{ Q \equiv a=A \wedge 1 \leq k \leq N \}$$

accion desplazar(a,b: **ent/sal** vect; k: natural)

$$\{ R \equiv a=A \wedge \forall \alpha \in \{ 1..N \}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \}^1$$

4. DISEÑO Y VERIFICACIÓN POR INMERSIÓN NO FINAL

Para alcanzar una solución recursiva al problema es necesario plantear una inmersión de forma que sea posible establecer una recurrencia con disminución del tamaño del problema. Se planteará en primer lugar una inmersión que dé origen a una solución recursiva no final.

En este tipo de inmersión se modifica el algoritmo debilitando la postcondición mediante la inclusión de algún parámetro adicional, la nueva especificación es:

$$\{ Q \equiv a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \}$$

accion desplazar1(a,b: **ent/sal** vect; k,i: natural)

$$\{ R \equiv a=A \wedge \forall \alpha \in \{ 1..i \}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \}$$

será $\text{desplazar}(a,b,k) = \text{desplazar1}(a,b,k,N)$

El caso trivial será aquel en el que $i=0$, en este caso no es necesario realizar ninguna operación ya que la postcondición se cumple al anularse el dominio del cuantificador \forall .

¹ Se puede intentar una postcondición basada en la operación **mod**, pero no parece resultar muy elegante dado que los índices del vector se consideran entre 1 y N y eso complica la operación.

En el caso no trivial se realizará una llamada recursiva para $i-1$ y para cumplir la postcondición se ha de realizar una operación adicional consistente en asignar a $b[i+k]$ o $b[i+k-N]$ el elemento de índice desplazado k lugares a la izquierda en el vector a (en sentido circular). La solución queda:

$$\{ Q \equiv a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \}$$

accion desplazar1(a,b: ent/sal vect; k,i: natural)

caso $i=0 \rightarrow$ **nada**

□ $i>0 \rightarrow$ desplazar1(a,b,k,i-1)
 si $i+k \leq N$ **entonces**
 $b[i+k]=a[i]$
 sino
 $b[i+k-N]=a[i]$
 fsi

$$\{ R \equiv a=A \wedge \forall \alpha \in \{1..i\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \}$$

Se verifica a continuación:

$$1) Q(x) \Rightarrow Bt(x) \vee Bnt(x)$$

$$a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \Rightarrow i=0 \vee i>0, \text{ lo que es inmediato}$$

$$2) Q(x) \wedge Bnt(x) \Rightarrow Q(s(x))$$

$$a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \wedge i>0 \equiv a=A \wedge 1 \leq k \leq N \wedge 0 < i \leq N \text{ y por tanto se tiene que } a=A \wedge 1 \leq k \leq N \wedge 0 \leq i-1 \leq N$$

$$3) Q(x) \wedge Bt(x) \Rightarrow R(x, \text{triv}(x))$$

$$a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \wedge i=0 \equiv a=A \wedge 1 \leq k \leq N \wedge i=0 \Rightarrow$$

$$a=A \wedge \text{cierto} \equiv \forall \alpha \in \{1..0\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \text{ (ya que se anula el dominio del cuantificador)}$$

$$4) Q(x) \wedge Bnt(x) \wedge R(s(x), y') \Rightarrow R(x, c(y', x))$$

El miembro izquierdo de la implicación es:

$$a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \wedge i>0 \wedge \forall \alpha \in \{1..i-1\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$$

La operación adicional $c(y',x)$ es

si $i+k \leq N$ **entonces**
 $b[i+k]=a[i]$
sino
 $b[i+k-N]=a[i]$
fsi

luego se tendrá que $R(x,c(y',x))$ es:

$a=A \wedge$
 $\forall \alpha \in \{1..i-1\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \wedge (i+k \leq N \wedge b[i+k]=a[i]) \vee (i+k > N \wedge b[i+k-N]=a[i])$

lo que es equivalente a la postcondición.

5) La función $t(a,b,k,i)=i$ cumple esta condición

6) $t(a,b,k,i-1)=i-1 < i=t(a,b,k,i)$

Para el coste se puede establecer la recurrencia:

$T(N)=T(N-1)+c$ siendo ya que la operación adicional es de coste constante y se realiza una sola llamada recursiva con tamaño de problema disminuido en una unidad. La solución de esta recurrencia da: $T(N) \in \theta(N)$

5. DISEÑO POR INMERSIÓN FINAL

Se planteará ahora una inmersión que dé lugar a una solución recursiva final. En este caso se mantiene la postcondición de la función original y se fortalece la precondición añadiendo una conjunción que contiene la postcondición debilitada. La nueva especificación queda:

$\{ Q \equiv a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \wedge$
 $\forall \alpha \in \{1..i\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \}$

accion $\text{desplazar2}(a,b: \text{ent/sal vect}; k,i: \text{natural})$

$\{ R \equiv a=A \wedge$
 $\forall \alpha \in \{1..N\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$

y será $\text{desplazar}(a,b,k)=\text{desplazar2}(a,b,k,0)$

Ahora se tendrá un caso trivial cuando $i=N$ ya que en este caso se cumple la postcondición sin necesidad de realizar ninguna operación. Cuando $i < N$ se llamará recursivamente a la función

incrementando i en una unidad, pero antes se han de realizar la operación necesaria para que en esta nueva llamada se cumpla la precondition. La operación adicional consistirá en asignar $a[i+1]$ a $b[i+1+k]$ o $b[i+1+k-N]$ (según sea $i+1+k$ menor o mayor de N). La solución queda:

$$\{ Q \equiv a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \wedge \\ \forall \alpha \in \{1..i\}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \}$$

accion desplazar2(a,b: ent/sal vect; k,i: natural)

caso $i=N \rightarrow$ **nada**

\square $i < N \rightarrow$ **si** $i+1+k \leq N$ **entonces**

$$b[i+1+k]=a[i+1]$$

sino

$$b[i+1+k-N]=a[i+1]$$

fsi

desplazar2(a,b,k,i+1)

$$\{ R \equiv a=A \wedge \\ \forall \alpha \in \{1..N\}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \}$$

5.1. Verificación y análisis de coste asintótico

$$1) Q(x) \Rightarrow Bt(x) \vee Bnt(x)$$

$a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \Rightarrow i=N \vee i < N$, lo que es inmediato

$$2) Q(x) \wedge Bnt(x) \Rightarrow Q(s(x))$$

$a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \wedge i < N \equiv a=A \wedge 1 \leq k \leq N \wedge 0 < i < N$ y por tanto se tiene que $a=A \wedge 1 \leq k \leq N \wedge 0 \leq i+1 \leq N$ además en la operación previa a la llamada recursiva se asigna $a[i+k]$ al elemento de b adecuado para que verificándose antes de esa asignación que:

$$\forall \alpha \in \{1..i\}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$$

tras ella se tenga:

$$\forall \alpha \in \{1..i+1\}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$$

$$3) Q(x) \wedge Bt(x) \Rightarrow R(x, \text{triv}(x))$$

$$a=A \wedge 1 \leq k \leq N \wedge 0 \leq i \leq N \wedge$$

$\forall \alpha \in \{1..i\}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \wedge i=N$ es evidente que implica la postcondición

$$4) Q(x) \wedge Bnt(x) \wedge R(s(x), y') \Rightarrow R(x, c(y', x))$$

Es inmediato ya que al ser función recursiva final $c(y', x)$ es la operación nula y $R(s(x), y') \Rightarrow R(x, c(y', x))$

5) La función $t(a, b, k, i) = N - i$ cumple esta condición

$$6) t(a, b, k, i-1) = N - (i+1) < N - i = t(a, b, k, i)$$

Para el coste se puede establecer la recurrencia:

$T(N) = T(N-1) + c$ siendo ya que la operación adicional es de coste constante y se realiza una sola llamada recursiva con tamaño de problema disminuido en una unidad. La solución de esta recurrencia da: $T(N) \in \theta(N)$

6. DISEÑO Y VERIFICACIÓN ITERATIVO

La solución iterativa ha de ser de la forma:

$$\{ Q \equiv a = A \wedge 1 \leq k \leq N \}$$

accion desplazar(a, b: **ent/sal** vect; k: natural)

inicializar;

mientras B **hacer** {P}

restablecer;

avanzar;

fmientras

$$\{ R \equiv a = A \wedge$$

$$\forall \alpha \in \{ 1..N \}. (\alpha + k \leq N \wedge b[\alpha + k] = a[\alpha]) \vee (\alpha + k > N \wedge b[\alpha + k - N] = a[\alpha])$$

En la que **P** es el invariante del bucle y **B** la condición de protección del bucle. Se tendrá $P \wedge \neg B \Rightarrow R$

El primer paso en el diseño es la obtención de P y B. Para ello es conveniente expresar la postcondición R como conjunción de dos predicados de forma que uno de ellos se tome como invariante y el otro como inverso de la condición de protección del bucle. Por ejemplo:

$$R \equiv a = A \wedge$$

$$\forall \alpha \in \{ 1..i \}. (\alpha + k \leq N \wedge b[\alpha + k] = a[\alpha]) \vee (\alpha + k > N \wedge b[\alpha + k - N] = a[\alpha]) \wedge i = N$$
 Siendo:

$$P \equiv \forall \alpha \in \{ 1..i \}. (\alpha + k \leq N \wedge b[\alpha + k] = a[\alpha]) \vee (\alpha + k > N \wedge b[\alpha + k - N] = a[\alpha]) \text{ y } \neg B \equiv i = N$$

la operación **inicializar** ha de asegurar que el invariante se cumpla antes de iniciar la iteración, con la operación:

$i:=0$

se consigue este objetivo fácilmente al anular el dominio del cuantificador \forall

la protección del bucle será $B \equiv i \neq N$, y puesto que i parte de valor 0 y el bucle finaliza para $i=N$, la operación **avanzar**, que como su nombre indica ha de hacer que el algoritmo avance hacia la condición de finalización, será:

$i:=i+1$

con lo que el algoritmo queda:

$\{ Q \equiv a=A \wedge 1 \leq k \leq N \}$

accion desplazar(a,b: ent/sal vect; k: natural)

$i:=0$;

mientras $i \neq N$ **hacer** {P}

restablecer;

$i:=i+1$;

fmientras

[7]

$\{ R \equiv a=A \wedge$

$\forall \alpha \in \{ 1..N \}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$

La operación **restablecer** ha de conseguir que el invariante, que se cumplirá antes de ejecutarla, siga cumpliéndose tras la operación avanzar. Antes de la operación restablecer se cumple el invariante para i :

$P_i \equiv \forall \alpha \in \{ 1..i \}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$

y tras la operación avanzar se ha de cumplir el invariante para $i+1$:

$P_{i+1} \equiv \forall \alpha \in \{ 1..i+1 \}. (\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$

Se tiene $P_{i+1} = P_i \wedge (i+1+k \leq N \wedge b[i+1+k]=a[i+1]) \vee (i+1+k > N \wedge b[i+1+k-N]=a[i+1])$

y el último predicado de la conjunción del segundo miembro será cierto si la operación restablecer es:

si $i+1+k \leq N$ **entonces**

$b[i+1+k]=a[i+1]$

sino

$b[i+1+k-N]=a[i+1]$

fsi

El algoritmo completo queda:

```

{ Q≡a=A ∧ 1≤k≤N }
accion desplazar(a,b: ent/sal vect; k: natural)
  i:=0;
  mientras i≠N hacer {P}
    si i+1+k≤N entonces
      b[i+1+k]=a[i+1]
    sino
      b[i+1+k-N]=a[i+1]
    fsi
    i:=i+1;
  fmientras
{R≡a=A ∧
  ∀ α ∈ {1..N}.(α+k≤N ∧ b[α+k]=a[α])∨(α+k>N ∧ b[α+k-N]=a[α])

```

[7]

La verificación es:

1) $P \wedge B \Rightarrow R$

Por diseño se tiene:

$R \equiv a=A \wedge$
 $\forall \alpha \in \{1..i\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \wedge i=N$ Siendo:

$P \equiv \forall \alpha \in \{1..i\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$

y

$B \equiv i=N$

2) $Q \text{ Inic } P$

Al hacer $i=0$ se anula el dominio del cuantificador \forall en el predicado del invariante y este se cumple.

3) $\{P \wedge B\} S \{P\}$

Se ha de comprobar que, suponiendo que a la entrada del bucle se cumple el invariante y teniendo en cuenta que también se ha de cumplir la condición de protección del bucle, tras realizar las operaciones internas al bucle se sigue cumpliendo el invariante. A la entrada del bucle se tiene:

$\forall \alpha \in \{1..i\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \wedge i \neq N$

en las operaciones internas se asigna $a[i+1]$ a $b[i+1+k]$ si $i+1+k \leq N$ o a $b[i+1+k-N]$ si $i+k > N$, luego se incrementa i en una unidad. Por lo tanto a la salida del bucle se tendrá:

$$\forall \alpha \in \{1..i\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha]) \\ \wedge (i+1+k \leq N \wedge b[i+1+k]=a[i+1]) \vee (i+1+k > N \wedge b[i+1+k-N]=a[i+1])$$

lo que equivale a:

$$\forall \alpha \in \{1..i+1\}.(\alpha+k \leq N \wedge b[\alpha+k]=a[\alpha]) \vee (\alpha+k > N \wedge b[\alpha+k-N]=a[\alpha])$$

que es el invariante para $i+1$.

$$4) P \wedge B \Rightarrow t \geq 0$$

tomando como función limitadora $t(i)=N-i$ es inmediato que cuando $P \wedge B \Rightarrow i \leq N$ y por tanto $t(i) \geq 0$

$$5) \{P \wedge B \wedge t=T\} S \{t < T\}$$

Se trata de comprobar que las instrucciones interiores al bucle hacen que decrezca la función limitadora.

Esto es inmediato ya que $t(i+1)=N-(i+1) < N-i=t(i)$

Eficiencia:

En el bucle se repiten $\theta(N)$ las operaciones interiores al bucle, todas ellas son operaciones de asignación y por tanto de coste constante por lo que el algoritmo será de orden:

$$T(n)=\theta(N)\theta(1)=\theta(N)$$

Se deja como ejercicio el análisis de los paralelismos entre soluciones recursiva final e iterativa

7. CODIFICACION EN MODULA 2

No se realiza análisis de tiempo de ejecución en este ejemplo, el procedimiento a seguir es análogo al explicado para el Ejercicio 1. También se deja como ejercicio la programación de Desplazar1.

```
MODULE Ejercicio4;

FROM InOut IMPORT WriteString, WriteCard, WriteLn, Read;
FROM TimeDate IMPORT Time, GetTime;

CONST
  N = 10;

TYPE
  vect = ARRAY [1..N] OF INTEGER;

VAR
  VectorA, VectorB : vect;

PROCEDURE Desplazar2(VAR a,b:vect;k,i: CARDINAL);
BEGIN
  IF i<N THEN
    IF i+1+k<=N THEN
      b[i+1+k]:=a[i+1];
    ELSE
      b[i+1+k-N]:=a[i+1];
    END;
    Desplazar2(a,b,k,i+1);
  END
END Desplazar2;

PROCEDURE DesplazarIter(VAR a,b: vect;k: CARDINAL);
VAR i: CARDINAL;
BEGIN
  i:=0;
  WHILE i<N DO
    IF i+1+k<=N THEN
      b[i+1+k]:=a[i+1];
    ELSE
      b[i+1+k-N]:=a[i+1];
    END;
    i:=i+1;
  END
END DesplazarIter;
```

(* PROGRAMA PRINCIPAL *)

VAR i,k,t: CARDINAL; t1,t2: Time;s:INTEGER;

BEGIN

(* Se rellena el primer vector con valores apropiados *)

FOR i:=1 TO N DO

VectorA[i]:=i;

END;

(* Se verifica el funcionamiento correcto de cada una de las funciones *)

WriteLn;WriteLn;

WriteString("Resultado de Desplazar2:");

WriteLn;

WriteString("Vector A: ");

FOR i:=1 TO N DO

WriteCard(VectorA[i],3);

END;

Desplazar2(VectorA,VectorB,3,0);

WriteLn;

WriteString("Vector B: ");

FOR i:=1 TO N DO

WriteCard(VectorB[i],3);

END;

WriteLn;WriteLn;

WriteString("Resultado de DesplazarIter:");

WriteLn;

WriteString("VectorA: ");

FOR i:=1 TO N DO

WriteCard(VectorA[i],3);

END;

DesplazarIter(VectorA,VectorB,3);

WriteLn;

WriteString("VectorB: ");

FOR i:=1 TO N DO

WriteCard(VectorB[i],3);

END

END Ejercicio4.

8. BIBLIOGRAFÍA

1. R.Peña. "Diseño de Programas: formalismo y abstracción". Prentice Hall. 1993
2. R.Peña. "Diseño de Programas: formalismo y abstracción, 2ª Edición". Prentice Hall. 1997
3. J.L.Balcázar. "Programación Metódica". McGraw-Hill. 1993
4. J.I.Mayorga Toledano, M.Rodríguez Artacho y F.López Ostenero. Colección de Problemas de Programación II. 1998