Fundamentos de análisis y diseño de algoritmos

Universidad del Valle

Facultad de Ingeniería

Escuela de Ingeniería de sistemas y computación

Agosto 2017

Algoritmo iterativo

Correctitud de un algoritmo iterativo

Invariantes de ciclo

Computación iterativa

Una computación iterativa se caracteriza por comenzar en un estado inicial S_0 y transformar ese estado en un conjunto de estados intermedios hasta llegar a un estado final S_j

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \ldots \rightarrow S_j$$

Todo estado se debe caracterizar por cumplir una condición, llamada invariante.

Computación iterativa

¿Qué es una especificación?

Una especificación se define como la descripción de los siguientes parámetros:

Entrada: indica las precondiciones

Salida: indica las poscondiciones

Idea iterativa: muestra cómo deberían cambiar los estados, comenzando desde el inicial hasta llegar al final

Estados: especifica la forma de cada estado en forma de tupla, además, se muestra cuál es el invariante de estado

Estado inicial: muestra los valores que forman el estado inicial

Estado final: muestra los valores que forman el estado final

Transformación de estados: de manera formal especifica cómo se realizan, en términos generales, los cambios de un estado al siguiente

Computación iterativa

¿Qué es demostrar correctitud de un algoritmo?

Un algoritmo es correcto *con respecto a* una especificación

Será correcto si para cada entrada que cumple las precondiciones, el algoritmo termina cumpliendo la poscondición

Además, para el caso específico de algoritmos iterativos, se cuenta con un método formal de probar la correctitud

Computación iterativa

Especificación para el cálculo de factorial

Entrada: N≥0

Salida: resultado=N!

Idea: Iteración

(0,1)→(1,1)→(2,2)→(3,6)→…→(N,N!)

Estados: Tupla de la forma (índice, resultado) tal que resultado=indice! (Invariante)

Estado inicial: indice=0, resutado=1

Estado final: indice=N

Transformación de estados:

(índice, resultado) → (índice +1,resultado*(índice+1))

Corrección

Un especificación es la definición de un problema en términos de su precondición Q y poscondición R

Un algoritmo A es correcto con respecto a una especificación si para cada conjunto de valores que cumplen Q, los valores de salida cumplen R

Se denota como {Q} A {R}. “A es correcto con respecto a la precondición Q y a la poscondición R”

Computación iterativa

Algoritmo para el cálculo de factorial

```
Factorial(int N){
        int indice=0;
        int resultado=1;
        while !(indice==N){
                indice=indice +1;
                resultado= resultado * indice;
        }
        System.out.println(resultado);
}
```

Estado inicial

Computación iterativa

Algoritmo para el cálculo de factorial

```
Factorial(int N){
        int indice=0;
        int resultado=1;
        while !(indice==N){
                indice=indice +1;
                resultado= resultado * indice;
        }
        System.out.println(resultado);
}
```

¿Es correcto el algoritmo con respecto a la especificación?

Computación iterativa

Especificación para el cálculo de raiz de X

Entrada: $X \geq 0 \wedge X \in R \wedge \delta > 0$

Salida: *a* tal que $|a^2 - X| \leq \delta$

Idea: Dado X, inicie la aproximación de *a* con el valor 1.0 y mejorela utilizando el cambio de *a* por $(a + X/a)/2$

$$(1, 1.0) \rightarrow (2, (1.0 + x/1.0)/2) \rightarrow \ldots \rightarrow (N, a)$$

Estados: Tupla de la forma (índice, aproximación) tal que $a > 0$ (Invariante)

Estados inicial: $a = 1.0$ $i := 1$

Estado final: *a* tal que $|a^2 - X| \leq \delta$ $i := N$

Transformación de estados: (índice, a)→(indice+1, (a+X/2)/2)

Computación iterativa

Algoritmo para el cálculo de raiz de X

```
raizIterativa(double X, double delta){
        double a=1.0;
        while (  !(Math.abs(a*a-X)<=delta) ){
                a = (a + X/a)/2.0;
        }
        System.out.println(a);
}
```

Computación iterativa

Identifique en los algoritmos Factorial(int N) y raizIterativa(double X, double delta) los estados inicial y final, asi como la transformación dada en la especificación

¿Cómo se manejan las condiciones de entrada en el algoritmo?

Computación iterativa

Algoritmo para el cálculo de factorial

```
Factorial(int N){
        int indice=0;
        int resultado=1;
        while !(indice==N){
                indice=indice +1;
                resultado= resultado * indice;
        }
        System.out.println(resultado);
}
```


Condiciones iniciales

Computación iterativa

Algoritmo para el cálculo de factorial

```
Factorial(int N){
        int indice=0;
        int resultado=1;
        while !(indice==N){
            indice=indice +1;
            resultado= resultado * indice;
        }
        System.out.println(resultado);
}
```

Estado inicial

Transformación de estados:

(índice, resultado) → (índice +1, resultado*(índice+1))

$(i, R) \longrightarrow (i+1, R(i+1))$

Computación iterativa

Algoritmo para el cálculo de raiz de X

```
raizIterativa(double X, double delta){
        double a=1.0;
        while (  !(Math.abs(a*a-X)<=delta) ){
                a = (a + X/a)/2.0;
        }
        System.out.println(a);
}
```

Condición inicial

Computación iterativa

Algoritmo para el cálculo de raiz de X

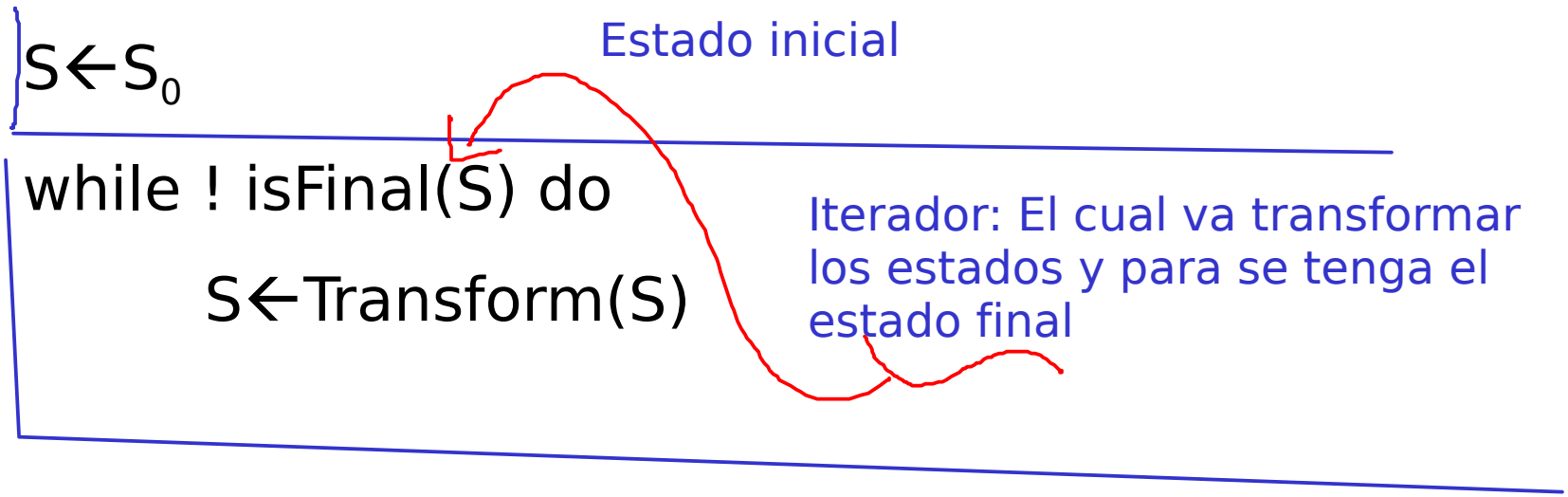
```
raizIterativa(double X, double delta){
        double a=1.0;
        while (  !(Math.abs(a*a-X)<=delta) ){
                a = (a + X/a)/2.0;
        }
        System.out.println(a);
}
```

Transformación de estados:

(índice, a)→(indice+1, (a+X/2)/2)

Computación iterativa

El esquema de un algoritmo iterativo es el siguiente:



Computación iterativa

Cómo probar que un algoritmo iterativo A es correcto con respecto a un especificación (precondición Q, poscondición R)

1. Inicialización: Pruebe que el estado inicial S_0 cumple el invariante
2. Invarianza: Prueba que la transformación conserva el invariante
3. Éxito: Si S es un estado final ^ se cumple el invariante P → R
4. Terminación: A termina

Computación iterativa

```
Computa (int A, int B){
    int res=0, i=1;
    while (i<=B){
        i=i+1;
        res=res + A;
    }
    System.out.println(res);
}
```

1) ¿Como puedo describir los estado en este algoritmo?

(i, res)

2) ¿Como es el estado inicial?

(1, 0)

3) ¿Como cambian los estados a medida que itera (transformación de estados)?

(i, res) ⟶ (i+1, res+A)

Qué calcula Computa(2,3)?

$(1, 0) \rightarrow (2, A) \rightarrow (3, A+A) \rightarrow (4, A+A+A)$

$\rightarrow (B, \underbrace{A+A+A+..A}_{B-1}) \rightarrow (B+1, \underbrace{A+A+..A}_{B \text{ veces}})$

$\sum_{i=1}^{B} A$ ← Resultado final

$\sum_{i=1}^{p} A$

$i=1 \quad rP_3 = 0$

$i=2 \quad rP_3 = A$

$i=3 \quad rP_3 = 2A$

Invariante de ciclo

Computación iterativa

```
Computa (int A, int B){
        int res=0, i=1;
        while (i<=B){
                i=i+1;
                res=res + A;
        }
        System.out.println(res);
}
```

Q: $A,B \in Z \wedge B>0$

R: res=A*B

Algo que conocemos del problema

Computación iterativa

```
Computa (int A, int B){
        int res=0, i=1;
        while (i<=B){
                i=i+1;
                res=res + A;
        }
        System.out.println(res);
}
```

Identifique los estados y su invariante

Computación iterativa

```
Computa (int A, int B){
        int res=0, i=1;
        while (i<=B){
                i=i+1;
                res=res + A;
        }
}
```

Considere cada estado como el par (i,res)

(1,0)→(2,A)→(3,A+A) → ... →(B+1, A+ ... +A)

Invariante P: res=$\sum_{p=1}^{i-1} A$

Computación iterativa

Probar correctitud

1. Inicialización: *Pruebe que el estado inicial S_0 cumple el invariante*

   El estado inicial es (1,0), Se verifica que se cumpla el invariante, se tiene que i=1.

Computación iterativa

Probar correctitud

1. Inicialización: *Pruebe que el estado inicial S_0 cumple el invariante*

   El estado inicial es (1,0), Se verifica que se cumpla el invariante, se tiene que i=1.

$$res = \sum_{p=1}^{i-1} A = \sum_{p=1}^{0} A = 0$$

Computación iterativa

2. Invarianza: *Prueba que la transformación conserva el invariante*

Se considera que antes de entrar el ciclo, i=k y se prueba.

Si i=k, $res = \sum_{p=1}^{k-1} A$

Computación iterativa

```
Computa (int A, int B){
        int res=0, i=1;
        while (i<=B){
                i=i+1;
                res=res + A;
        }
        System.out.println(res);
}
```

Computación iterativa

2. Invarianza: *Prueba que la transformación conserva el invariante*

Se considera que antes de entrar el ciclo, i=k y se prueba.

Si i=k, $res = \sum_{p=1}^{k-1} A \rightarrow \sum_{p=1}^{k} A$

Al ejecutar la iteración, i=k+1:

Se toma/observa del algoritmo!!!

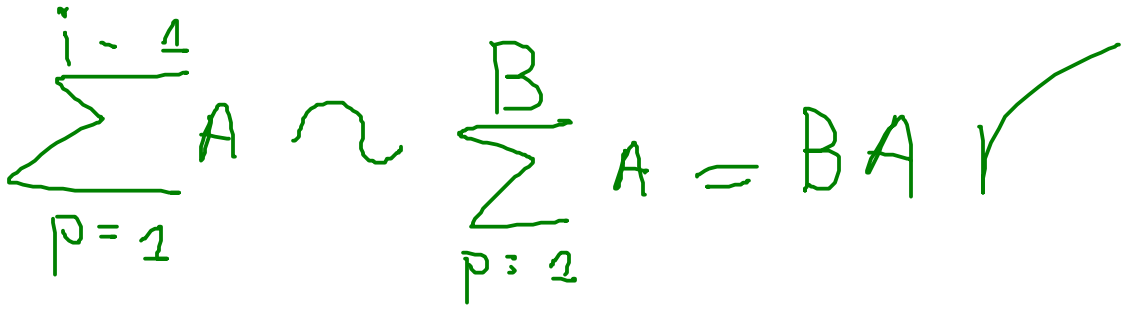
res = res + A

res = $\sum_{p=1}^{k-1} A$ + A = $\sum_{p=1}^{k+1} A$

Computación iterativa

3. Éxito: *Invariante P ^ S es un estado final → R*

El ciclo finaliza con i=B+1, este es el valor de i en el estado final. Se calcula res.



Computación iterativa

3. Éxito: *Invariante P ^ S es un estado final → R*

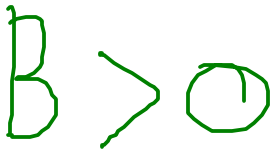
El ciclo finaliza con i=B+1, este es el valor de i en el estado final. Se calcula res:

$$res = \sum A = \sum A = \sum A = A * B$$

Computación iterativa

4. Terminación: A termina

En cada iteración i aumenta, por lo que en algún momento finito tendrá que alcanzar el valor de B y el algoritmo terminará



Computación iterativa

¿Qué calcula Computa3(4)?

Exprese la forma de los estados

Muestre la idea iterativa que presenta el algoritmo

Pruebe la correctitud

Indique la precondición y poscondición

```
Computa3 (int N){
    int A, B, i, j;
    A=0;
    i=1;
    while (i<=N){
        B=1;
        j=1;
        while (j<=3){
            B=B*i;
            j++;
        }
        A=A+B;
        i++;
    }
    System.out.println("Resultado=" + A);
}
```

N = 4

Forma estado (j, B)

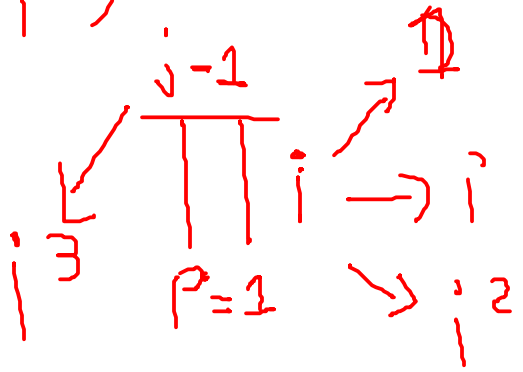
Estado inicial (1,1)

Transformación de estados
(j,B) --> (j+1, B*i)

Estado final
$(4, i^3)$

$(1, 1) \rightarrow (2, i) \rightarrow (3, i^2)$
$\rightarrow (4, i^3)$

$A = A + i^3$



$$(j, B) \to (j+1, B \times i)$$

$$\prod_{p=1}^{j-1} i \qquad \prod_{p=1}^{j} i$$

$$\prod_{p=1}^{j-1} i * i = \prod_{p=1}^{j} i$$

```
int A, B, i, j;
A=0;
i=1;
while (i<=N){
        B=1;
        j=1;
        while (j<=3){
                B=B*i;
                j++;
        }
        A=A+B;
        i++;
}
```

Forma del estado
(i,A)

Estado inicial
(1,0)

Evolucion de estados

(1,0) --> (2,i^3)--> (3, i^3 + i^3)

Transformación de estados

(i, A) --> (i+1, A+i^3)

Estado final

$(N+1, \underbrace{0}_{i=1} + \underbrace{1^3}_{i=2} + \underbrace{2^3}_{i=3} + \underbrace{3^3}_{i=4} \cdots \underbrace{N^3}_{i=n+1})$

$\left(N+1, \sum_{i=1}^{n+1} (i-1)^3 \right)$

Invariante de ciclo y probar

$$\sum_{i=1}^{k} (i-1)^3$$

Estado inicial

k = 1

$(1-1)^3 = 0^3 = 0$ ✓

Estado final

$k = N+1$

$$\sum_{i=1}^{N+1} (i-1)^3$$ ✓

Transformación de estados

$(i, A) \Rightarrow (i+1, A+i^3)$

$$\sum_{p=1}^{i} (p-1)^3 \quad | \quad \sum_{p=1}^{i+1} (p-1)^3$$

$$\sum_{p=1}^{i} (p-1)^3 + i^3$$

$$\sum_{p=1}^{i+1} (p-1)^3$$

Computación iterativa

¿Qué calcula Opera(2,3)?

Exprese la forma de los estados

Muestre la idea iterativa que presenta el algoritmo

Pruebe la correctitud

Indique la precondición y poscondición

```
Opera (int B, int N){
        int A, C, D;
        D=0;
        A=1;
        C=N;
        while (C>=1){
                A=A*B;
                D=D+A;
                C--;
        }
        System.out.println("Resultado=" + D);
}
```

1) Entender el algoritmo

¿Que calcula opera(2,3)?

C = 3

C = 3
A = 1*2
D = 0 +2

C = 2
A = 2*2
D = 2 + 4

C =1
A = 2*2*2
D = 2+4+8

C = 0 //No entra

Valor D
14

Invariante de ciclo: $D=\sum B^p, donde \quad A=B^{N-C}$

```
Opera (int B, int N){
    int A, C, D;
    D=0;
    A=1;
    C=N;
    while (C>=1){
        A=A*B;
        D=D+A;
        C--;
    }
    System.out.println
}
```

¿Que me permite describir un estado ?

(C, A, D)

Estado inicial

(N,1,0)

Evolucionan estados

(N,1,0) --> (N-1,1*B, 0 + 1*B)
--> (N-2, 1*B*B, 0+1*B+1*B*B)

Transformación de estados

(C,A,D) --> (C-1, A*B, D+A*B)

Estado final

$(0, 1 \times B \times B \times \dots \times B, 0 + B + B^2 + B^3 + \dots)$

$C=N \quad C=N-1 \quad C=N-2 \quad C=0$

B^N

$0 + B^1 + B^2 + B^3 + \dots + B^{N-1} + B^N$

$\sum_{p=1}^{N} B^p + 0$

$(0, B^N, \sum_{p=1}^{N} B^p)$

$\frac{B^{n+1}-1}{B-1} - 1 \leftarrow \sum_{i=0}^{n} ar^i = \frac{ar^{n+1}-a}{r-1}$

$a=1 \quad r=B$

Invariante de ciclo :)

$(K, B^K, \sum_{p=1}^{K} B^p + 0)$

Estado inicial

$C=N-K \quad (0, B^0, \sum_{p=1}^{0} B^p)$

$(0, 1, 0)$

Estado final

$(N, B^N, \sum_{p=1}^{N} B^p)$

Transformación de estados

$(C, A, D) \to (K, A, D) \to (K+1, A \times B, D + A \times B)$

$C-K$

Aplicar invariante

$(K, B^K, \sum_{p=1}^{K} B^p) \to (K+1, B^{K+1}, \sum_{p=1}^{K+1} B^p)$

Aplico la transformación

$(K, B^K, \sum_{p=1}^{K} B^p) \to (K+1, BB^K, \sum_{p=1}^{K} B^r + BB^K)$

$(K+1, B^{K+1}, \sum_{p=1}^{K} B^r + B^{K+1})$

Exito. Sumatoria de B^P

¿Termina? Si, porque itera desde N hasta 0, N es finito.

$\sum_{p=1}^{K+1} B^p$

```
nene(int N)

C = 0
D = 1

while(C <= N)
        A = 3
        B = 5
        while( A > 0)
                B += B
                A--
        D += B
        C++
```

Estado

(A, B)

Estado inicial

(3,5)

Evolución de estados

(3,5) --> (2,10) --> (1, 20) --> (0, 40)

Transformación de estados

(A,B) -> (A-1,2B)

Estado final (0,40)

Invariante de ciclo

(K, $2^k 5$)

Estado inicial (0, 1*5) SI

Estado final (3,2^3 * 5) --> (3,40) CUmple

Transformación de estados

$(k, 2^k 5) \longrightarrow (k+1, 2^{k+1} 5)$

$(k, 2^k 5) \longrightarrow (k+1, 2(2^k 5))$

$\longrightarrow (k+1, 2^{k+1} 5)$ Cumple :)

Externo

Forma de estado (C,D)

Transformación (C,D) -> (C+1, D+40)

Estado inicial (0,1)

Estado final (N+1, (N+1)40+1)

Invariante

$(C, \sum_{i=1}^{C} 40 + 1)$

$(c, D) \rightarrow (C+1, D+40)$

$(C, \sum_{i=1}^{C} 40+1) \longrightarrow (C+1, \sum_{i=1}^{C} 40 + 1 + 40)$

$(C+1, \sum_{i=1}^{C+1} 40 + 1)$

$(C+1)40+1$

¿El algoritmo termina?

Si, porque C crece desde 0 hasta N. No hay restricción con N, porque si N < 0, nisiquiera entra.

```
BS (int A[], int N){
        int i, j, aux;
        for ( i=1; i < N ; i++)
                for ( j=N; j > i ; j--){
                        if ( A[j] < A[j - 1] ){
                                aux=A[j];
                                A[j]=A[j-1];
                                A[j-1]=aux;
                        }
}
```

A partir del procedimiento indicado a continuación, que tiene como entrada un arreglo A indexado de la forma [1..n]. Indicar:

1. Invariante de ciclo para el iterador interno (linea 3)
2. Invariante de ciclo para el iterador externo (linea 2)
3. ¿Que calcula ?

Externo

$(1, A) \to (2, A) \to \dots \to (n, A)$

Interno

$(N, A) \to (N-1, A) \rightsquigarrow \{q_1, q_2 \dots q_{n-1}, q_n\}$

$\{q_1, q_n, q_{n-1}^*, q_n^*\}$

$\to (N-2, A) \quad \{q_1, q_2, \dots q_{n-2}^*, q_{n-1}^*, q_n\}$

$(j+1, A) \to (j, A)$

$q_1, q_{i+1} \dots q_{n-2}\ q_{n-1}\ q_n\}$

Transformación de estados

$(i, A) \to (i+1, A^*)$

$\{q_1, \dots, q_i^*\ q_{i+1}, \dots, q_n\}$

inicio (N, A)

Fin (i, A^*)

$\{q_1, q_2 \dots q_i\ q_{i+1} \dots q_n\}$

$q_i \to$ menor

$N \to N-1, N-2, \dots i$

$N-(N-i)$

(K, A^*)

$0 \le K \le N-i \quad \{a_1, q_2, q_3, \dots q_k^*\} \{q_{k+1}, \dots q_{n-1}, q_n\}$

(q_k)

$\{4,3,2\}$

$\{2,4,3\} \leftarrow \{4,2,3\}$

$(K, A) \rightsquigarrow (K+1, A^*)$

Externo

$(1, A) \to (2, A) \to \dots \to (n, A)$

$(i, A) \to (i+1, A^*)$

1) q_i menor q_{i+1}, q_n

2) q_{i+1} menor q_{i+1}, q_n

$(1, A)$

(N, A^*)

$\{q_1 < q_2 < q_3 < \dots < q_n\}$

(K, A)

$\{q_1 < q_2 < q_3 < \dots q_k \dots\}$

$(K, A) \to (K+1, A^*)$

$q_1 < q_2 \dots q_k \dots q_1 < q_2 < q_{k+1}$

¿El algoritmo termina?

Si, porque si N>0, va variar entre 1 y N

Proceso iterativo

1) Identificar el estado (iterador y salida)

2) Identificar el estado inicial

3) Identificar transformación de estados. MIrar como cambia el estado cada vez que itera. Identificar estado final

4)Identificar la invariante de ciclo. Dado una iteración k ¿QUe se cumple siempre?

5) Mirar si la invariante:

a) EL estado inicial
b) El estado final
c) La transformación de estados: Inducción. SI se cumple para k debe cumplirse para k+1

6) ¿El algoritmo termina?

Esto se hace para cada ciclo, sea interno o externo. Id

```
algoritmo2(int a, int n){
  int b = n;
  int c = 3;
  int r = 0;

  while(b >= 0){
    while(c>=0){
      r+=a;
      c--;
    }
    r+=n;
    c = 3;
    b--;
  }
  print(r);

}
```

1
1
1
$n-0+1+1=(n+2)$
$(n+1)(c+2)=5(n+1)$
$(n+1)(c+1)=4(n+1)$
$(n+1)(c+1)=4(n+1)$
$n+1$
$n+1$
$n+1$
1

$4+16(n+1)+(n+2)$

$22+17n$

Intento

(c, r)

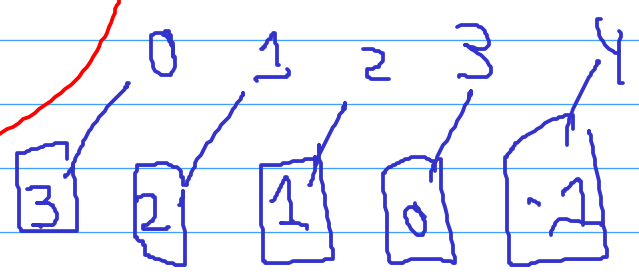
Inicio

(3, 0)

Transf estados $(c,r) \to (c-1, r+q)$

Est final $\{-1, 4q\}$

Inv $c = k$



$$\left(c-k,\ \sum_{j=1}^{k} q + 0 = qk\right)$$

$$\sum_{k=0}^{3}$$

Inicio

$(3, 0)$

Fino

$\left(\frac{-1}{2}, 4q\right)$

$k = 4$

Transf estado

$(c, r) \to (c-1, r+q)$

$\left(-1, 4q\right)\checkmark$

$(c-k, r) \to (c-(k+1), r+q)$

$(c-k, kq) \to (c-(k+1), kq+q)$

$[c-(k+1), q(k+1)]$

Externo $k=0$

$(b, r) \to (b-1, r + (n+1)(4q+n))$ Transf

Interno $(n, 0)$ ←

Final $(-1, (n+1)(4q+n))$

Invariante

$\left(n-k, \sum_{j=1}^{k} (4q+n) + 0\right)$

$(n-k, k(4q+n))$

Transformación

Inicial

$(n-0, 0)$

$(n, 0)$ ✓

Final

$(-1, (n+1)(4q+n))$

$k = n+1$

$(-1, (n+1)(4q+n))$

$(b-k, r) \to (b-(k+1), r+4q+n)$

$(b-k, k(4q+n)) \to (b-(k+1), k(4q+n)+(4q+n))$

$(k+1)(4q+n)$

El algoritmo termina siempre y cuando n >= 0, ya que se itera desde n hasta 0 en los dos ciclos.

```
//Para n > 2
for(int i=2; i<n; i=i*i){
        for(int j=0; j<n*n; j++){
                //...
        }
}
```

$(2^4)^2 = 2^6$

2 4 16 16^2

1 2 4 8 $\log_2(x)$

→ 0 1 2 3

$(n^2+1) \times (\lfloor \log_2(\log_2(n)) \rfloor + 2)$

Referencias

Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. Introduction to Algorithms, Third Edition (3rd ed.). The MIT Press. Pages 18-20

Gracias

Próximo tema:

Complejidad computacional