Fundamentos de análisis y diseño de algoritmos

Computación iterativa

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!

Factorial(n)

Idea: Iteración

$$(0,1) \rightarrow (1,1) \rightarrow (2,2) \rightarrow (3,6) \rightarrow \ldots \rightarrow (N,N!)$$

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

res = 0!

→ transformación de estados

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: $a=1.0$, índice $= 1$

Estado final: a tal que $|a^2 - X| \leq \delta$

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);
}
```

Transformación de estados:

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

$S \leftarrow S_0$

while ! isFinal(S) do

    $S \leftarrow$ Transform(S)

Computación iterativa

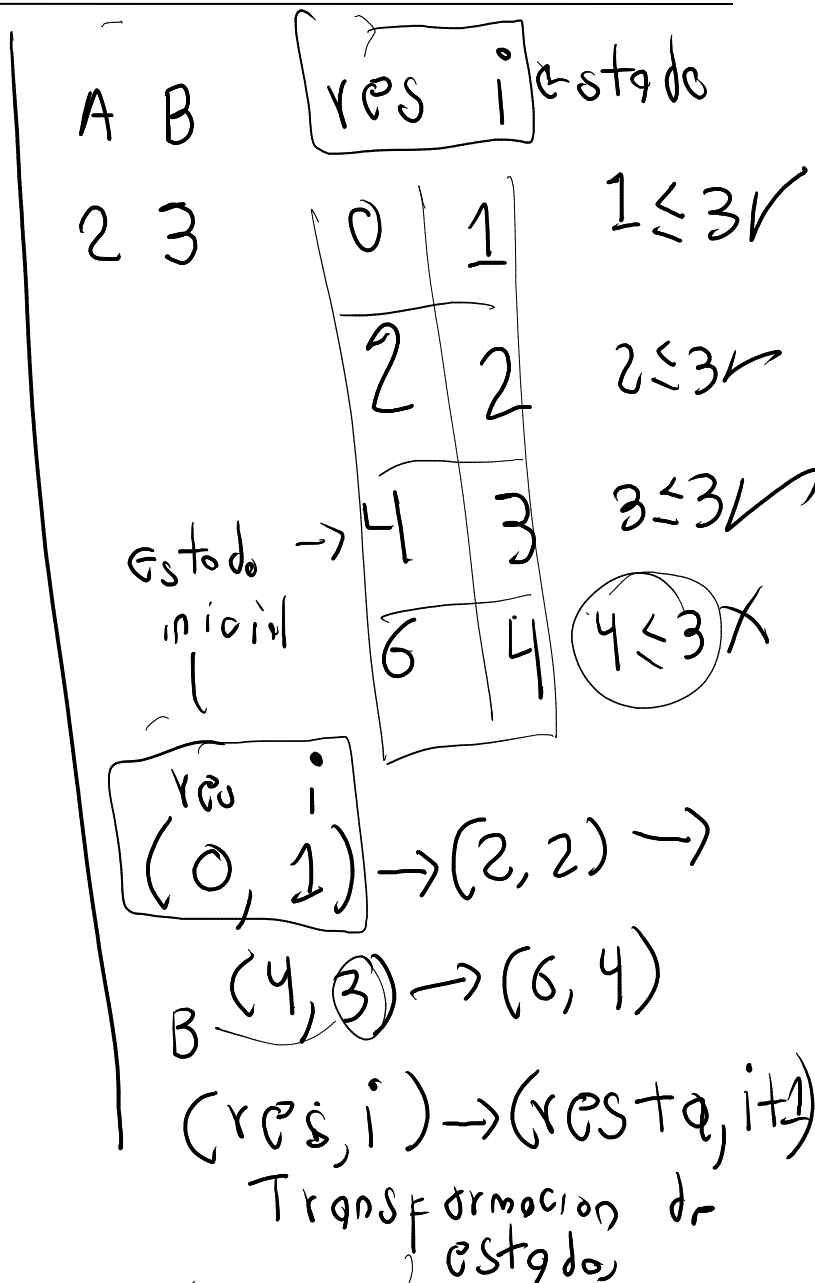
Cómo probar que un algoritmo iterativo A es correcto con respecto a un especificación (precondición Q, poscondición R)

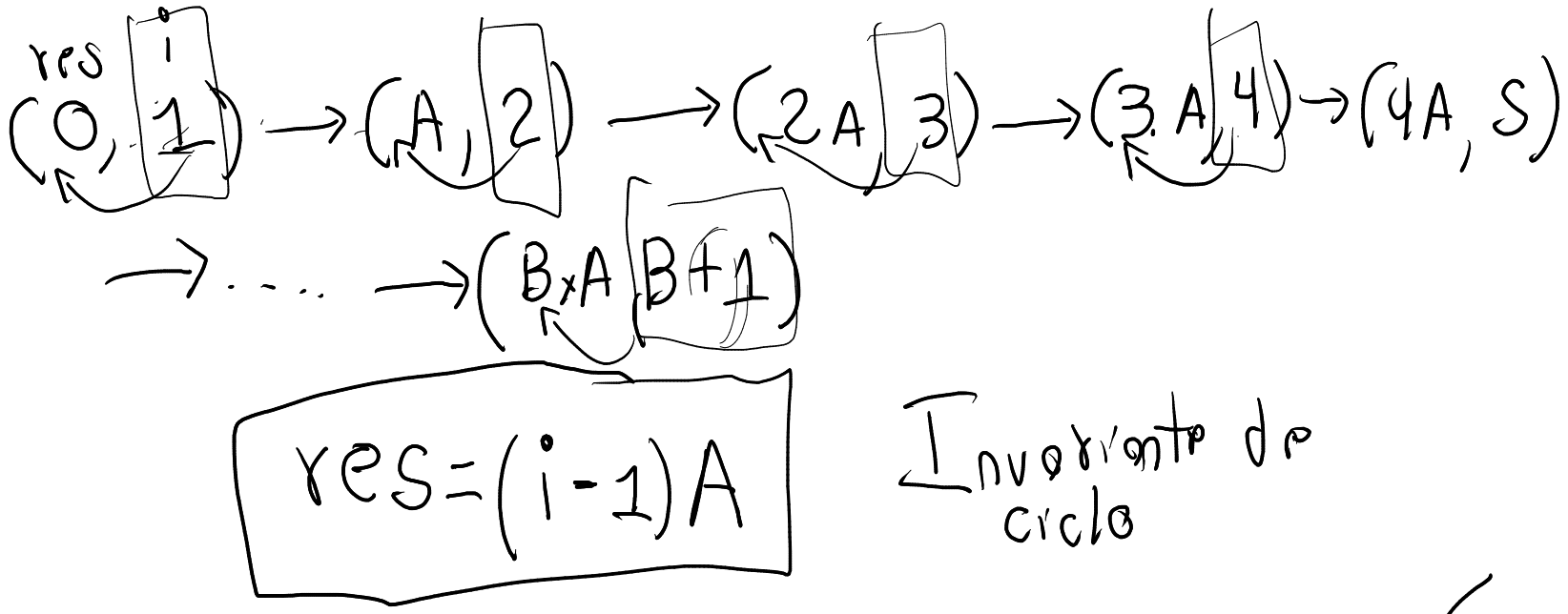
1. Inicialización: Pruebe que el estado inicial S_0 cumple el invariante
2. Invarianza: Prueba que la transformación conserva el invariante
3. Éxito: Si S es un estado final $\wedge$ se cumple el invariante $P \rightarrow R$
4. Terminación: A termina

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);
}
```

Qué calcula Computa(2,3)?





1) So invariante $\overset{res}{(0}, \overset{i}{1})$ $res=(1-1)A=0A=0$ ✓

$(res, i) \rightarrow (res+A, i+1)$

2) Transformar cumpl invariante $res=(i-1).A$

$i+1 \rightarrow res=(i+1-1)A=iA$

$res+A \rightarrow (i-1)A+A=(i-1+1)A=iA$

3) S_f (res, i) = (B A, B+1)

res = (i − 1) A

res = (B + 1 − 1) A

res = B A

4) Termina se $i \leq B$, $i = 1$, $i + 1$

1, 2, 3, 4, 5, ..., B, **B + 1** ← Termina

Computacion Iterativa

que es → For/while

¿cuando aplica? → $S_0 \to S_1 \to S_2 \ldots \to S_f$

diseñamos

1) Forma del estado variables
2) Transformacion de estado
3) Terminacion

que es → $S_i = (i, j, k \ldots)$ Variables

Cambia → Transformacion

El cambio de las variables dentro del ciclo

El algoritmo termina

```
def algoritmo(n):
    [          ] ← Estado inicial
    while[     ] ← Condicion de parada
    [          ] ← Transformacion del estado
```

Correctitud → Invariante de ciclo

que es? → Lo que SIEMPRE se va a cumplir en cada iteracion relacion entre variables

Verificar

1) ¿Se cumple invariante?
2) Transformacion ¿Cumple?
3) Sf ¿Cumple?
4) ¿Termina?

Mirar si los cambios satisfacen el invariante

¿como lo obtenemos? →

1) Prueba de escritorio valores arbitrarios
2) Generalizar
3) Observar que se cumple en iteracion

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

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)

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

Invariante P: res= $\sum_{i=1}^{k} A$ para un estado i = k cualquiera

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.

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

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,

$$res = \sum_{i=1}^{k} A = kA$$

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,

$$res = \sum_{i=1}^{k} A = kA$$

Al ejecutar la iteración, i=k+1:

res = res + A

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

Se toma/observa del algoritmo!!!

Computación iterativa

3. Éxito: *Invariante $P \wedge S$ es un estado final $\rightarrow R$*

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

Computación iterativa

3. Éxito: *Invariante $P \wedge S$ es un estado final $\rightarrow R$*

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

$$res = \sum_{i=1}^{B} A = BA$$

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

Debe calcular la invariante para el ciclo interno y el ciclo externo

```
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);
}
```

$B = i^3$

$A = A + i^3$

B	j	
1	1	$1 \leq 3$ ✓
i	2	$2 \leq 3$ ✓
i^2	3	$3 \leq 3$
i^3	4	$4 \leq 3$ ✗

Estado (i, A)

Estado inicial $(1, 0)$

Transformación $(i, A) \longrightarrow (i+1, A+i^3)$

Prueba $N = 5$

$(1, 0) \longrightarrow (2, 0+1^3) \longrightarrow (3, 0+1^3+2^3) \longrightarrow$

$(4, 0+1^3+2^3+3^3) \longrightarrow (5, 0+1^3+2^3+3^3+4^3)$

$\longrightarrow (6, 0^3+1^3+2^3+3^3+4^3+5^3)$

N Generalización

$$\left(N+1, \sum_{K=0}^{N} K^3\right)$$ Estado Final

Invariante

$$\left(i, \sum_{k=0}^{i-1} k^3\right)$$

$$A = \sum_{k=0}^{i-1} k^3$$

1) Estado inicial $(1, 0)$ $\quad i \quad A$ $\quad A = \sum_{k=0}^{0} i^3 = 0^3$ ✓

2) Transformación

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

(i, A)

$i := i+1$

$$\sum_{k=0}^{(i+1)-1} k^3 = \sum_{k=0}^{i} k^3$$

$A := A + i^3$

$$\underbrace{\sum_{k=0}^{i-1} k^3}_{A} + i^3 = 0^3 + 1^3 + 2^3 + \dots (i-1)^3 + i^3$$

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

3) Estado final $\left(N+1, \sum_{k=0}^{N} k^3\right)$

$$\left(i, \sum_{k=0}^{i-1} k^3\right)$$

$$A = \sum_{k=0}^{N+1-1} k^3 = \sum_{k=0}^{N} k^3$$

4) ¿Termina?

$i \leq n$

$i = 1$ Cambio $i = i + 1$

$1, 2, 3, 4, \ldots n, n+1$ ✓

```
Algoritmo(int n){
        i = 0
        s = 3
        while(i<=n){
             j = 0
             p = 4
            while(j <= 2n){
                  p += 2
                  j+=1
             }
            s+=2p
            i+=2
       }
}
```

Invariante de ciclo externo:

$$p = 3 + \sum_{s=1}^{k} 8n \text{ Donde } s = \frac{i}{2}$$

Invariante de ciclo interno:

$$p = 4 + \sum_{j=1}^{k} 2$$

Computación iterativa

¿Qué calcula Algoritmo(6)?

Exprese la forma de los estados

Muestre la idea iterativa que presenta el algoritmo

Pruebe la correctitud

Indique la precondición y poscondición

```
def algoritmo1(n)
    i = 1
    res = 1
    while i<2*n:
        res *= i
        i+=1

    return res
```

$S_i = (i, res)$ $S_0 : (1, 1)$

Transformación

$(i, res) \rightarrow (i+1, i \times res)$

Prueba

$(1, 1) \rightarrow (2, 1) \rightarrow (3, 2) \rightarrow (4, 6)$

$\rightarrow (5, 24) \rightarrow (6, 120) \rightarrow \cdots (2n, ?)$

$1! = 1$

$2 = 2$

$3 = 6$

$(2n, (2n-1)!)$

Invariante $(i, (i-1)!)$

Inv $(i, (i-1)!)$

Inicial $(1, 1) \quad (1, 0!)$

Transformación

$$(i, res) \rightarrow (i+1, res \times i)$$

$(i, (i-1)!)$

$i := i+1$ $\quad res = i!$

$i\,(i-1)! = i \times (i-1) \times (i-2) \times (i-3) \times \dots \times 2 \times 1$

$= i!$

Estado final $(2n; (2n-1)!)$

$I_n\ (i; (i-1)!)$

$i = 2n$

Terminal $i = 1, 2, 3, \ldots, n, \ldots 2n$

$i < 2n$

```
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[i] ){
                aux=A[j];
                A[j]=A[i];
                A[j-1]=aux;
            }
}
```

Invariante Externo

El subarreglo A[i...n] tiene su menor elemento en la posición A[i]

Invariante Interno:

El subarreglo A[i..j] cumple que A[i] < A[j]

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 ?

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:

Notación de crecimiento de funciones