

# 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  $S_j$

$$S_0 \rightarrow S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_j$$

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:  $N \geq 0$

Salida: resultado =  $N!$

Idea: Iteración

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

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

Estado inicial: índice = 0, resultado = 1

Estado final: índice =  $N$ , resultado =  $N!$

Transformación de estados:

$(\text{índice}, \text{resultado}) \rightarrow (\text{índice} + 1, \text{resultado} * (\text{índice} + 1))$

# Corrección

---

Un **especificación** es la definición de un problema en términos de su precondition  $Q$  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 precondition  $Q$  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);

}

; for(int indice = 0, resultado=0; indice<=N; indice++){

Estado inicial

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

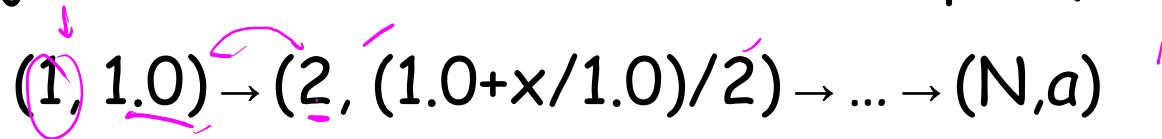
---

Especificación para el cálculo de raíz de  $X$

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

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  $(a + X/a)/2$


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

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

Estados inicial:  $a = 1.0$

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

Transformación de estados:  $(\text{índice}, a) \rightarrow (\text{índice} + 1, (a + X/a)/2)$

# Computación iterativa

---

Algoritmo para el cálculo de raíz 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, así como la transformación dada en la especificación

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

# Computación iterativa

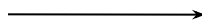
---

Algoritmo para el cálculo de factorial

```
Factorial(int N){
```

```
    int indice=0;
```

```
    int resultado=1;
```



*Condiciones iniciales*

```
    while !(indice==N){
```

```
        indice=indice +1;
```

```
        resultado= resultado * indice;
```

```
    }
```

```
    System.out.println(resultado);
```

```
}
```

# 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:

$(\text{índice}, \text{resultado}) \rightarrow (\text{índice} + 1, \text{resultado} * (\text{índice} + 1))$



# Computación iterativa

---

Algoritmo para el cálculo de raíz de X

```
raizIterativa(double X, double delta){
```

```
    double a=1.0;
```

—————→ *Condición inicial*

```
    while ( !(Math.abs(a*a-X)<=delta) ){
```

```
        a = (a + X/a)/2.0;
```

```
    }
```

```
    System.out.println(a);
```

```
}
```

# Computación iterativa

---

Algoritmo para el cálculo de raíz de X

```
raizIterativa(double X, double delta){
```

```
    double a=1.0;
```

```
    while ( !(Math.abs(a*a-X)<=delta) ){
```

```
        a = (a + X/a)/2.0;
```

Transformación de estados:

$(\text{índice}, a) \rightarrow (\text{índice}+1, (a+X/2)/2)$

```
    }
```

```
    System.out.println(a);
```

```
}
```



# Computación iterativa

---

El esquema de un algoritmo iterativo es el siguiente:

$S \leftarrow S_0$

while ! isFinal( $S$ ) do

$S \leftarrow \text{Transform}(S)$

# Computación iterativa

---

Cómo probar que un algoritmo iterativo  $A$  es correcto con respecto a una 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; ← Estado inicial
```

```
  while (i<=B){
```

```
    i=i+1;
```

```
    res=res + A;
```

```
  }
```

```
  System.out.println(res);
```

```
}
```

Qué calcula Computa(2,3)?

1) Estado  
 $(i, res)$

2) Estado inicial  
 $(1, 0)$

3) Transformación

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

4) Invariante

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

$\rightarrow (B+1, BA)$

$(i, (i-1)A)$

4) Estado final  $(B+1, BA)$

Comprobacion  $(i, (i-1)A)$

a) Estado inicial  $(1, 0)$

$(1, (1-1)A)$   
 $(1, 0)$

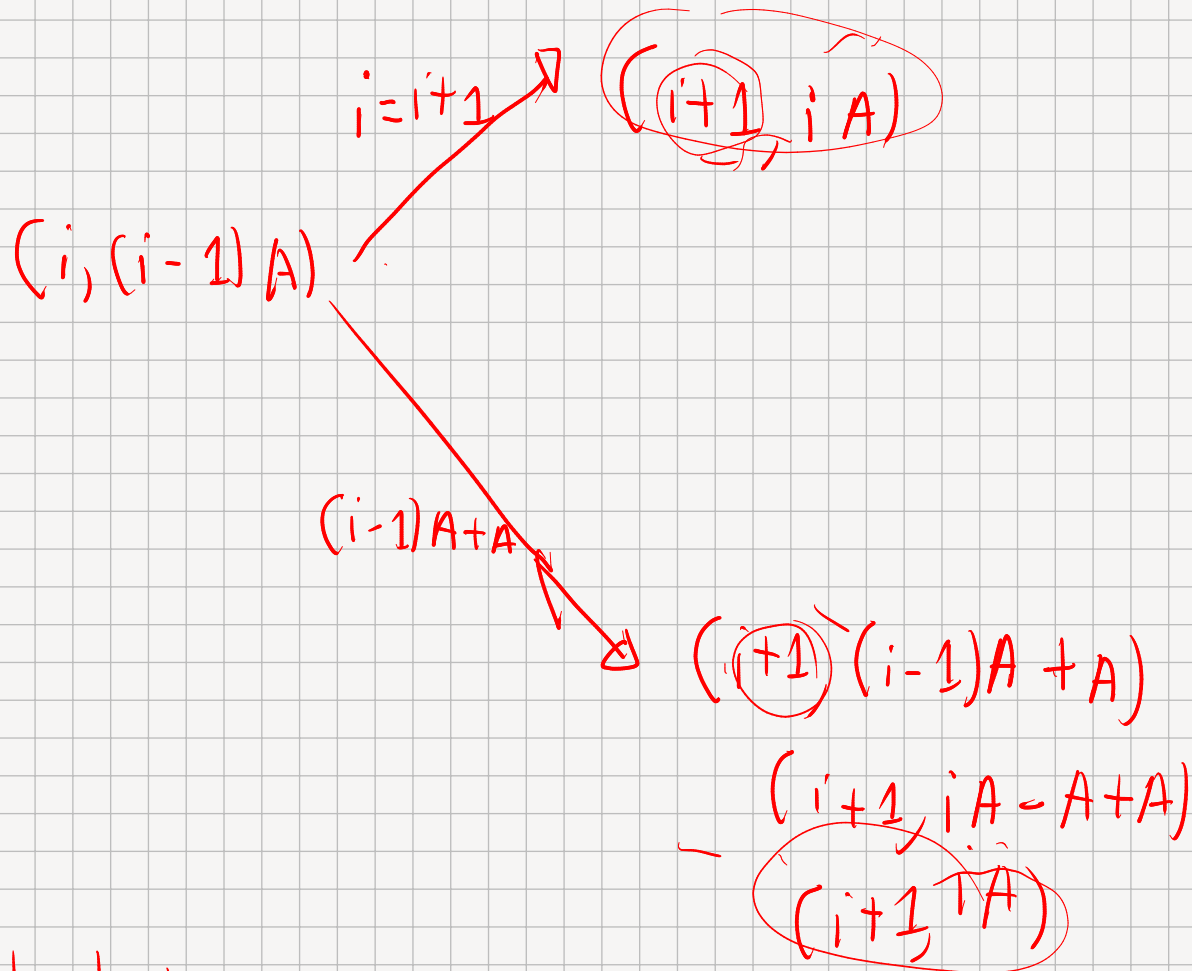
✓

b) Estado final  $(B+1, BA)$

$(B+1, (B+1-1)A)$   
 $(B+1, BA)$

$$P(n) \rightarrow P(n+1)$$

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



d) Algorithm

$$1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow B+1 \quad B \geq 0$$

# 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:  $A, B \in \mathbb{Z} \wedge B > 0$

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)

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

Invariante P:  $\text{res} = \sum_{i=1}^k A$  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)`?

Expresa la forma de los estados

Muestre la idea iterativa que presenta el algoritmo

Pruebe la correctitud

Indique la precondition 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);
```

```
}
```

$B = 1$

$A = A + B$

$\rightarrow i++$

$i++$   
 $B = i^3$   
 $A = A + B$

$j = 1$

$B_1 = B \times i$

$j = 2$

$B_2 = B_1 \times i$

$j = B_3 = B_2 \times i$

$B_3 = (B_1 \times i) \times i$

$B_3 = (B \times i) \times i \times i$

$B = B \times i^3$

$B = i^3$

Estado  $(i, A_i)$

Inicial  $(1, 0)$

Transformación

$$(1, 0) \rightarrow (2, 0+1^3) \rightarrow (3, 0+1^3+2^3) \rightarrow (4, 0+1^3+2^3+3^3) \\ \rightarrow (N+1, 0+1^3+2^3+\dots+N^3)$$

$$(N+1, \sum_{i=0}^N i^3)$$

Estado Final :

Invariante:  $(i, \sum_{p=0}^{i-1} p^3)$

$$\sum_{p=0}^2 p^3 = 0^3 + 1^3 + 2^3$$

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:

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

```
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
```

```
    }
```

Internos

Estado  $(j, p)$

Estado inicial  $(0, 4)$

Transformación de  
estados

$(0, 4) \rightarrow (1, 6) \rightarrow (2, 8) \rightarrow \dots$   
 $(0, 4) \rightarrow (1, 4+2) \rightarrow (2, 4+2+2) \rightarrow \dots$   
 $\dots \rightarrow (2n+1, 4+2(2n+1))$   $2(2n+1)$

$(j, p) \rightarrow (j+1, p+2)$

Invariante

$\rightarrow p = 4 + 2j$   
 $(j, 4 + 2j)$

1) Estado inicial  $(0, 4)$   $(j, 4 + 2j) \leftarrow$   
 $(0, 4 + 2(0))$   $(0, 4)$

2) Estado Final  $(2n+1, 4 + 2(2n+1))$

3) Transformación de estado  $P(j) \rightarrow P(j+1)$

$j = j+1 \rightarrow (j+1, 4 + 2(j+1))$   
 $(j, P) \rightarrow (j+1, 4 + 2j + 2)$

$P = P+2 \rightarrow (j+1, P+2)$

4) Termina?  $n \geq 0$

$j \quad 0 \rightarrow 1 \rightarrow 2 \rightarrow 3 \rightarrow \dots \rightarrow 2n+1$

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

}

$$p = 2(2n+1) + 4$$

$$s += 4(2n+1) + 8$$

↙ n par

Estado  
(i, s)

Estado inicial  
(0, 3)

Transformación

$$(i, s) \rightarrow (i+2, s + 8n + 12)$$

$$(0, 3) \rightarrow (2, 3 + 8n + 12) \rightarrow$$

$$(4, 3 + 2(8n + 12)) \rightarrow$$

$$(6, 3 + 3(8n + 12)) \rightarrow$$

$$(8, 3 + 4(8n + 12)) \rightarrow$$

$$(i, \frac{i}{2}(8n + 12) + 3)$$

Inv ciclo

$$(n+2, \frac{n+2}{2}(8n+12)+3)$$

1) Estado inicial  $i = 0$

$$S = \frac{i}{2}(8n+12) + 3 = 3 \quad \checkmark$$

2) Estado final  $i = n+2$

$$S = \frac{n+2}{2}(8n+12) + 3 \quad \checkmark$$

3) Transformacion  $P(i) \rightarrow P(i+2)$

$$(i, S) \xrightarrow{i \rightarrow i+2} (i+2, \frac{i+2}{2}(8n+12) + 3)$$
$$(i, S) \xrightarrow{S \rightarrow S+8n+12} (i+2, \frac{i}{2}(8n+12) + 3 + 8n+12) \quad \checkmark$$
$$(i, S) \xrightarrow{S \rightarrow S+8n+12} (i+2, \frac{i}{2}(8n+12) + 3 + 8n+12) \quad \checkmark$$

4) Termina?  $0 \rightarrow 2 \rightarrow 4 \rightarrow \dots \rightarrow n+2$   $\underbrace{\hspace{10em}}_S$



# Computación iterativa

---

¿Qué calcula `Algoritmo(6)`?

Expresa la forma de los estados

Muestre la idea iterativa que presenta el algoritmo

Pruebe la correctitud

Indique la precondition y poscondición

```

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 (línea 3)
2. Invariante de ciclo para el iterador externo (línea 2)
3. ¿Qué 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