Análisis de Algoritmos

Introducción[1]

José Ortiz Bejar

Facultad de Ingeniería Eléctrica

14 de Febrero de 2021

Contenido

Información General

- Información en linea
  - contenido: ***https://tinyurl.com/4eafcm7p***
  - página del curso: ***http://dep.fie.umich.mx/ job/cursos/11***

Información General

- Información en linea
  - contenido: ***https://tinyurl.com/4eafcm7p***
  - página del curso: ***http://dep.fie.umich.mx/ job/cursos/11***

- Avisos
  - ***Se harán en clase y a través del correo institucional. Es responsabilidad del estudiante revisar constantemente.***

Información General

- Información en linea
  - contenido: ***https://tinyurl.com/4eafcm7p***
  - página del curso: ***http://dep.fie.umich.mx/ job/cursos/11***

- Avisos
  - ***Se harán en clase y a través del correo institucional. Es responsabilidad del estudiante revisar constantemente.***

- contacto
  - email: *jose.ortiz@umich.mx*
  - Oficina: *Departamento de estudios de posgrado, Edificio Omega II*

Información General

- Información en linea
  - contenido: ***https://tinyurl.com/4eafcm7p***
  - página del curso: ***http://dep.fie.umich.mx/ job/cursos/11***

- Avisos
  - ***Se harán en clase y a través del correo institucional. Es responsabilidad del estudiante revisar constantemente.***

- contacto
  - email: *jose.ortiz@umich.mx*
  - Oficina: *Departamento de estudios de posgrado, Edificio Omega II*
- Etiqueta en clase
  - No usar celular
  - Se puede salir/entrar sin preguntar (discretamente)
  - No fotos

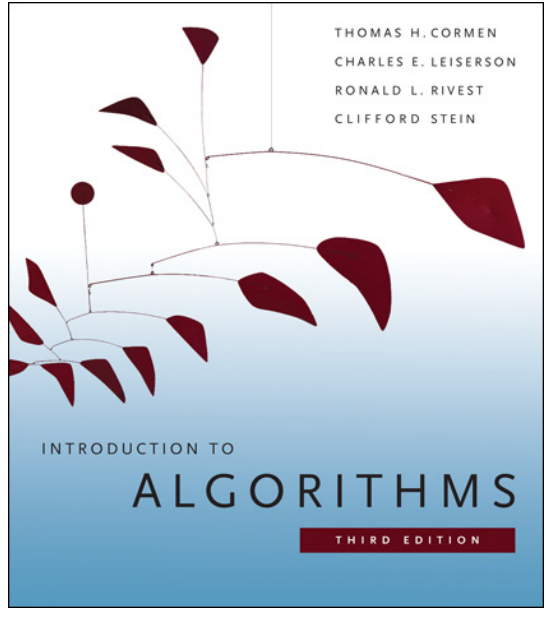
Libro de texto

Introduction to Algorithms

Third Edition

Thomas H. Cormen
Charles E. Leiserson
Ronald L. Rivest
Clifford Stein

The MIT Press



Evaluación

- 80% Examenes
  - 4 Exámenes
    - Examen 1 - Capítulos 1-2
    - Examen 2 - Capítulo 3
    - Examen 3 - Capítulo 4
    - Examen 4 - Capítulos 5-6
- 20% Proyecto de Aplicaciones de algoritmos de grafos
- Asistencia
  - Derecho a presentar examen Extra $\geq$ 80%
  - Derecho a presentar examen Adicional $\geq$ 60%

Evaluación

- 80% Examenes
  - 4 Exámenes
    - Examen 1 - Capítulos 1-2
    - Examen 2 - Capítulo 3
    - Examen 3 - Capítulo 4
    - Examen 4 - Capítulos 5-6
- 20% Proyecto de Aplicaciones de algoritmos de grafos
- Asistencia
  - Derecho a presentar examen Extra $\geq$ 80%
  - Derecho a presentar examen Adicional $\geq$ 60%
- $-$100% copias y plagio

Ideas fundamentales

- Johannes Gutenberg inventa la imprenta, aproximadamente en 1450(conocida en China, aproximadamente desde 1200)

Ideas fundamentales

- Johannes Gutenberg inventa la imprenta, aproximadamente en 1450(conocida en China, aproximadamente desde 1200)
- Sistema númerico decimal (India, aproximadamente 600)

Ideas fundamentales

Muhammad ibn Musa
al-Khwārizmī

- Johannes Gutenberg inventa la imprenta, aproximadamente en 1450(conocida en China, aproximadamente desde 1200)
- Sistema númerico decimal (India, aproximadamente 600)
- Libro de mátematica de Khwārizmī (Baghdad, aproximadamente 830)

Ideas fundamentales

Muhammad ibn Musa al-Khwārizmī

- Johannes Gutenberg inventa la imprenta, aproximadamente en 1450(conocida en China, aproximadamente desde 1200)
- Sistema númerico decimal (India, aproximadamente 600)
- Libro de mátematica de Khwārizmī (Baghdad, aproximadamente 830)
  - Métodos para sumar, multiplicar,dividir números, etc.

Ideas fundamentales

Muhammad ibn Musa
al-Khwārizmī

- Johannes Gutenberg inventa la imprenta, aproximadamente en 1450(conocida en China, aproximadamente desde 1200)
- Sistema númerico decimal (India, aproximadamente 600)
- Libro de mátematica de Khwārizmī (Baghdad, aproximadamente 830)
  - Métodos para sumar, multiplicar,dividir números, etc.
  - Procedimientos ***precisos***, ***no ambiguos***, ***mecánicos***, ***eficientes***, y ***correctos***

Ideas fundamentales

Muhammad ibn Musa al-Khwārizmī

- Johannes Gutenberg inventa la imprenta, aproximadamente en 1450(conocida en China, aproximadamente desde 1200)
- Sistema númerico decimal (India, aproximadamente 600)
- Libro de mátematica de Khwārizmī (Baghdad, aproximadamente 830)
  - Métodos para sumar, multiplicar,dividir números, etc.
  - Procedimientos ***precisos***, ***no ambiguos***, ***mecánicos***, ***eficientes***, y ***correctos***
  - *Son **algoritmos!***

Ejemplo

- Una secuencia de números

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

Ejemplo

- Una secuencia de números

$$0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$$

- La serie de Fibonacci

Leonardo da Pisa (1170 1250)
hijo de Guglielmo “Bonaccio”
a.k.a. *Leonardo Fibonacci*

La serie de Fibonacci

- Definición matemática: $F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$

La serie de Fibonacci

- Definición matemática: $F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$
- Implementación en computadora:

Racket

```
(define (F n)
  (cond
    ((= n 0) 0)
    ((= n 1) 1)
    (else (+ (F (- n 1)) (F (- n 2))))))
```

La serie de Fibonacci

- Definición matemática: $F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$
- Implementación en computadora:

Java

```
public class Fibonacci {
  public static int F(int n) {
    if (n == 0) {
      return 0;
    } else if (n == 1) {
      return 1;
    } else {
      return F(n-1) + F(n-2);
    } }
}
```

La serie de Fibonacci

- Definición matemática: $F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$
- Implementación en computadora:

C or C++

```
int F(int n) {
  if (n == 0) {
    return 0;
  } else if (n == 1) {
    return 1;
  } else {
    return F(n-1) + F(n-2);
  }
}
```

La serie de Fibonacci

- Definición matemática: $F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$
- Implementación en computadora:

Ruby

```
def F(n)
  case n
    when 0
      return 0
    when 1
      return 1
    else
      return F(n-1) + F(n-2)
  end
end
```

La serie de Fibonacci

- Definición matemática: $F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$
- Implementación en computadora:

Python

```
def F(n):
  if n == 0:
    return 0
  elif n == 1:
    return 1
  else:
    return F(n-1) + F(n-2)
```

- Definición matemática: $F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$
- Implementación en computadora:

Implementación con el operador ternario (C/C++/Java)

```
int F(int n) { return (n<2)?n:F(n-1)+F(n-2); }
```

La serie de Fibonacci

- Definición matemática: $F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$
- Implementación en computadora:

"pseudo-code"

Fibonacci(n)

```
1  if n == 0
2       return 0
3  elseif n == 1
4       return 1
5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

Preguntas que debemos hacernos

```
Fibonacci(n) 1  if n == 0
             2      return 0
             3  elseif n == 1
             4      return 1
             5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

Preguntas que debemos hacernos

```
Fibonacci(n) 1  if n == 0
             2      return 0
             3  elseif n == 1
             4      return 1
             5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

1. ¿El algoritmo es *correcto?*
   - ¿Siempre termina ?
   - Si termina, ¿la salida es correcta?

Preguntas que debemos hacernos

```
Fibonacci(n) 1   if n == 0
             2       return 0
             3   elseif n == 1
             4       return 1
             5   else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

1. ¿El algoritmo es *correcto?*
   - ¿Siempre termina ?
   - Si termina, ¿la salida es correcta?

2. ¿Cuánto *tiempo* le toma completar la el cálculo?

Preguntas que debemos hacernos

```
Fibonacci(n) 1   if n == 0
             2       return 0
             3   elseif n == 1
             4       return 1
             5   else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

1. ¿El algoritmo es *correcto?*
   - ¿Siempre termina ?
   - Si termina, ¿la salida es correcta?

2. ¿Cuánto *tiempo* le toma completar la el cálculo?

3. ¿Existe un algoritmo mejor?

```
Fibonacci(n) 1  if n == 0
             2      return 0
             3  elseif n == 1
             4      return 1
             5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$$

```
Fibonacci(n) 1  if n == 0
             2      return 0
             3  elseif n == 1
             4      return 1
             5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

$$F_n = \begin{cases} 0 & \text{if } n = 0 \\ 1 & \text{if } n = 1 \\ F_{n-1} + F_{n-2} & \text{if } n > 1 \end{cases}$$

- Se puede probar que el algoritmo es correcto
  - asumimos $n \geq 0$

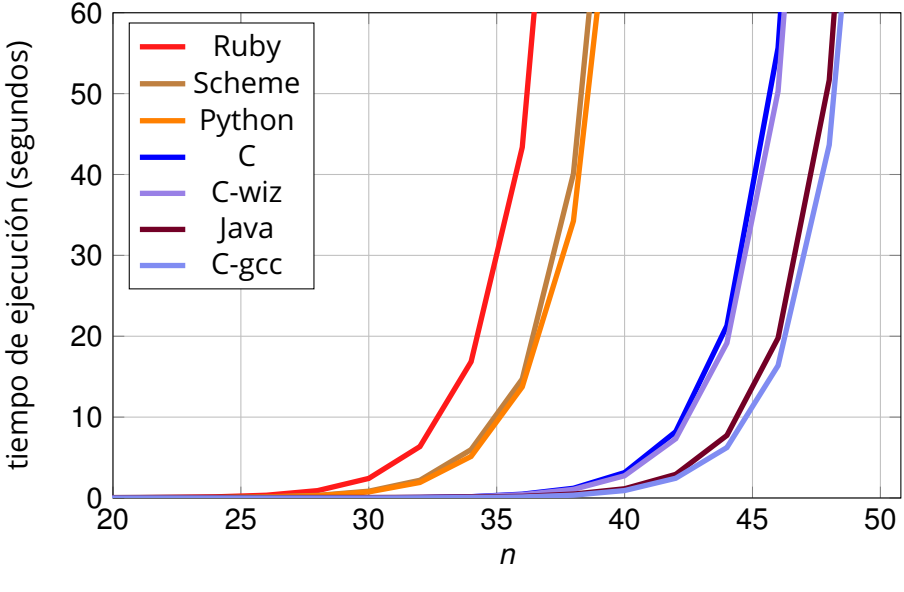
Desempeño

- ¿Cuánto tiempo dura la ejecución?

- ¿Cuánto tiempo dura la ejecución?

  Resultados tomados de la presentación original…

Resultados


Ruby
Scheme
Python
C
C-wiz
Java
C-gcc
tiempo de ejecución (segundos)
n
0
10
20
30
40
50
60
20
25
30
35
40
45
50

Comentarios

Comentarios

- Diferentes implementaciónes se comportan *diferente*
  - Dejar que el compilador optimice
  - optimizaciones simples dependientes del lenguaje no hacen gran diferencia
  - *todas* las implementaciones alcanzan un limite ...

Comentarios

- Diferentes implementaciónes se comportan *diferente*
  - Dejar que el compilador optimice
  - optimizaciones simples dependientes del lenguaje no hacen gran diferencia
  - *todas* las implementaciones alcanzan un limite …
- Conclusién: ***el problema es el algoritmo***

Complejidad

- Es necesario caracterizar matemáticamente el desempeño de los algoritmos.

  Se conoce como ***complejidad computacional*** del algoritmo

Complejidad

- Es necesario caracterizar matemáticamente el desempeño de los algoritmos.

  Se conoce como ***complejidad computacional*** del algoritmo

- Sea $T(n)$ el número de ***operaciones básicas*** necesarias para cálcular **Fibonacci**(n)

- Es necesario caracterizar matemáticamente el desempeño de los algoritmos.

  Se conoce como ***complejidad computacional*** del algoritmo

- Sea $T(n)$ el número de ***operaciones básicas*** necesarias para cálcular **Fibonacci**(n)

```
Fibonacci(n) 1  if n == 0
             2      return 0
             3  elseif n == 1
             4      return 1
             5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

- Es necesario caracterizar matemáticamente el desempeño de los algoritmos.

  Se conoce como ***complejidad computacional*** del algoritmo

- Sea $T(n)$ el número de ***operaciones básicas*** necesarias para cálcular **Fibonacci**(n)

```
Fibonacci(n) 1  if n == 0
             2      return 0
             3  elseif n == 1
             4      return 1
             5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

$T(0) = 2; T(1) = 3$

- Es necesario caracterizar matemáticamente el desempeño de los algoritmos.

  Se conoce como ***complejidad computacional*** del algoritmo

- Sea $T(n)$ el número de ***operaciones básicas*** necesarias para cálcular **Fibonacci**(n)

```
Fibonacci(n) 1  if n == 0
             2      return 0
             3  elseif n == 1
             4      return 1
             5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

$T(0) = 2; T(1) = 3$

$T(n) = T(n-1) + T(n-2) + 3$

- Es necesario caracterizar matemáticamente el desempeño de los algoritmos.

  Se conoce como ***complejidad computacional*** del algoritmo

- Sea $T(n)$ el número de ***operaciones básicas*** necesarias para cálcular **Fibonacci**(n)

```
Fibonacci(n) 1  if n == 0
             2      return 0
             3  elseif n == 1
             4      return 1
             5  else return Fibonacci(n − 1) + Fibonacci(n − 2)
```

$T(0) = 2; T(1) = 3$

$T(n) = T(n-1) + T(n-2) + 3 \quad \Rightarrow T(n) \geq F_n$

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

dado que $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

$$F_n \geq 2F_{n-2}$$

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

dado que $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4})$$

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

dado que $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6}))$$

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

dado que $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \ldots$$

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

dado que $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \ldots \geq 2^{\frac{n}{2}}$$

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

dado que $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \ldots \geq 2^{\frac{n}{2}}$$

Esto implica que

$$T(n) \geq (\sqrt{2})^n \approx (1.4)^n$$

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

dado que $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \ldots \geq 2^{\frac{n}{2}}$$

Esto implica que

$$T(n) \geq (\sqrt{2})^n \approx (1.4)^n$$

- $T(n)$ ***crece exponencialmente*** con n

Complejidad (2)

- Veamos como se comporta F_n conforme n crece

$$T(n) \geq F_n = F_{n-1} + F_{n-2}$$

dado que $F_n \geq F_{n-1} \geq F_{n-2} \geq F_{n-3} \geq \ldots$

$$F_n \geq 2F_{n-2} \geq 2(2F_{n-4}) \geq 2(2(2F_{n-6})) \geq \ldots \geq 2^{\frac{n}{2}}$$

Esto implica que

$$T(n) \geq (\sqrt{2})^n \approx (1.4)^n$$

- $T(n)$ ***crece exponencialmente*** con n
- ¿Es posible hacerlo mejor?

Un mejor algoritmo

- Nuevamente la secuencia es 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .

Un mejor algoritmo

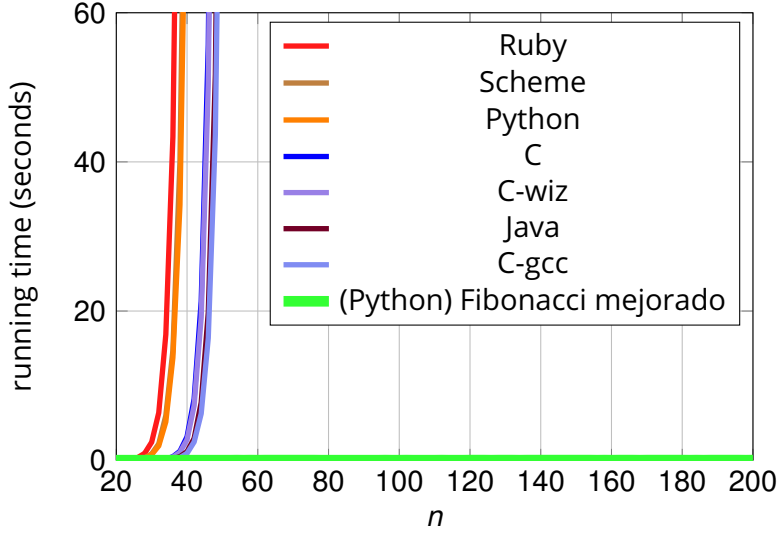
- Nuevamente la secuencia es $0, 1, 1, 2, 3, 5, 8, 13, 21, 34, \ldots$
- **Idea:** Podemos memorizar F_n y reducir el número de operaciones.

Un mejor algoritmo

- Nuevamente la secuencia es 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, . . .
- **Idea:** Podemos memorizar F_n y reducir el número de operaciones.

```
SmartFibonacci(n)  1  if n == 0
                   2        return 0
                   3  elseif n == 1
                   4        return 1
                   5  else pprev = 0
                   6       prev = 1
                   7       for i = 2 to n
                   8             f = prev + pprev
                   9             pprev = prev
                  10             prev = f
                  11  return f
```

Resultados


running time (seconds)
60
40
20
0
20
40
60
80
100
120
140
160
180
200
n
Ruby
Scheme
Python
C
C-wiz
Java
C-gcc
(Python) Fibonacci mejorado

Complejidad de SmartFibonacci

```
SmartFibonacci(n)  1   if n == 0
                   2       return 0
                   3   elseif n == 1
                   4       return 1
                   5   else prev = 0
                   6       pprev = 1
                   7       for i = 2 to n
                   8           f = prev + pprev
                   9           pprev = prev
                  10           prev = f
                  11   return f
```

Complejidad de SmartFibonacci

```
SmartFibonacci(n)  1  if n == 0
                   2      return 0
                   3  elseif n == 1
                   4      return 1
                   5  else prev = 0
                   6      pprev = 1
                   7      for i = 2 to n
                   8          f = prev + pprev
                   9          pprev = prev
                  10          prev = f
                  11  return f
```

$T(n) =$

Complejidad de SmartFibonacci

```
SmartFibonacci(n)  1   if n == 0
                   2       return 0
                   3   elseif n == 1
                   4       return 1
                   5   else prev = 0
                   6       pprev = 1
                   7       for i = 2 to n
                   8           f = prev + pprev
                   9           pprev = prev
                  10           prev = f
                  11   return f
```

$T(n) = 6 + 6(n-1)$

Complejidad de SmartFibonacci

```
SmartFibonacci(n)  1   if n == 0
                   2       return 0
                   3   elseif n == 1
                   4       return 1
                   5   else prev = 0
                   6       pprev = 1
                   7       for i = 2 to n
                   8           f = prev + pprev
                   9           pprev = prev
                  10           prev = f
                  11   return f
```

$T(n) = 6 + 6(n-1) = 6n$

Complejidad de **SmartFibonacci**

```
SmartFibonacci(n)  1   if n == 0
                   2       return 0
                   3   elseif n == 1
                   4       return 1
                   5   else prev = 0
                   6       pprev = 1
                   7       for i = 2 to n
                   8           f = prev + pprev
                   9           pprev = prev
                  10           prev = f
                  11   return f
```

$T(n) = 6 + 6(n-1) = 6n$

La *complejidad* de **SmartFibonacci**(n) es ***lineal*** en función de n

Programación competitiva

Evaluaciones para ingresar a empresas BigTech utilizan problemas estilo competencia.

Programación competitiva

Evaluaciones para ingresar a empresas BigTech utilizan problemas estilo competencia.

- Google *Code Jam 2022*
  - Qualification Round. Abril 1 23:00 (27 hrs)

Programación competitiva

Evaluaciones para ingresar a empresas BigTech utilizan problemas estilo competencia.

- Google *Code Jam 2022*
  - Qualification Round. Abril 1 23:00 (27 hrs)
- International Collegiate Programming Contest (ICPC)
  - Regional en agosto de 2022 (fecha exacta por definir).
- Facebook (META)
  - META Hack (12 de Febrero - concluido).

Programación competitiva

Evaluaciones para ingresar a empresas BigTech utilizan problemas estilo competencia.

- Google *Code Jam 2022*
  - Qualification Round. Abril 1 23:00 (27 hrs)
- International Collegiate Programming Contest (ICPC)
  - Regional en agosto de 2022 (fecha exacta por definir).
- Facebook (META)
  - META Hack (12 de Febrero - concluido).

Donde practicar

Programación competitiva

Evaluaciones para ingresar a empresas BigTech utilizan problemas estilo competencia.

- Google *Code Jam 2022*
  - Qualification Round. Abril 1 23:00 (27 hrs)
- International Collegiate Programming Contest (ICPC)
  - Regional en agosto de 2022 (fecha exacta por definir).
- Facebook (META)
  - META Hack (12 de Febrero - concluido).

Donde practicar

- Olimpiada Mexicana de informática - `http://omegaup.com`

Programación competitiva

Evaluaciones para ingresar a empresas BigTech utilizan problemas estilo competencia.

- Google *Code Jam 2022*
  - Qualification Round. Abril 1 23:00 (27 hrs)
- International Collegiate Programming Contest (ICPC)
  - Regional en agosto de 2022 (fecha exacta por definir).
- Facebook (META)
  - META Hack (12 de Febrero - concluido).

Donde practicar

- Olimpiada Mexicana de informática - `http://omegaup.com`
- Carribean Online Judge - `https://www.ecured.cu/Caribbean_Online_Judge`

Evaluaciones para ingresar a empresas BigTech utilizan problemas estilo competencia.

- Google *Code Jam 2022*
  - Qualification Round. Abril 1 23:00 (27 hrs)
- International Collegiate Programming Contest (ICPC)
  - Regional en agosto de 2022 (fecha exacta por definir).
- Facebook (META)
  - META Hack (12 de Febrero - concluido).

Donde practicar

- Olimpiada Mexicana de informática - `http://omegaup.com`
- Carribean Online Judge - `https://www.ecured.cu/Caribbean_Online_Judge`
- UVa - `https://onlinejudge.org/`

Evaluaciones para ingresar a empresas BigTech utilizan problemas estilo competencia.

- Google *Code Jam 2022*
  - Qualification Round. Abril 1 23:00 (27 hrs)
- International Collegiate Programming Contest (ICPC)
  - Regional en agosto de 2022 (fecha exacta por definir).
- Facebook (META)
  - META Hack (12 de Febrero - concluido).

Donde practicar

- Olimpiada Mexicana de informática - `http://omegaup.com`
- Carribean Online Judge - `https://www.ecured.cu/Caribbean_Online_Judge`
- UVa - `https://onlinejudge.org/`
- HackerRank - `https://www.hackerrank.com/`