Algoritmos y Estructuras de Datos
Tema 2: Algoritmos Básicos

Grado Imat. Escuela ICAI

Juan C. Aguí García

January 2024

Part I

Introducción a los Algoritmos

Algoritmos como estrategia de resolución de Problemas

Algoritmo: conjunto de pasos para desarrollar una tarea concreta.

- Finalidad última de un algoritmo: permitir solucionar un problema
- Debe ser fácil de entender, codificar y depurar
- Eficiente de los recursos (tiempo y espacio)
- 5 Pasos fundamentales en formulacion del Algoritmo
  1. Entender el problema
  2. Explora ejemplos concretos relacionados
  3. Trocéalo en partes más simples o en tareas
  4. Resuelve / Simplifica
  5. Mira atrás y reformula (factoriza, optimiza)

COMILLAS ICAI

1. Entender el problema

- Reformular el problema con nuestras palabras
- ¿Qué entradas necesito para el problema?
- ¿Qué salidas voy a obtener de esas entradas?
- ¿Puedo obtener las salidas que me piden de las entradas?
- ¿Tengo información suficiente para resolver el problema?
- ¿Qué datos son importantes como parte del problema?

2. Explora ejemplos concretos relacionados

- Empieza con problemas sencillos
- Escribe 2 o 3 ejemplos con sus entradas y salidas de datos
- Extrapola a ejemplos más complejos
- Evalúa los casos excepcionales (entradas vacías,datos incompletos)
- Plantea los ejemplos con entradas inválidas

COMILLAS ICAI

Analisis de Algoritmos: Corrección y Eficiencia

Corrección

- Un algoritmo se considera `Correcto` si siempre produce el resultado esperado para el rango de entradas válido y eventualmente llega a su fin.
- Es más fácil probar la no-correción que la correción de un algoritmo, ya que la complejidad de la causística en datos de entrada puede ser muy alta. Se aplican razonamientos **formales** para esta demontración
- En algoritmos de base aleatoria el resultado no es predecible al 100 %
- Se puede demostrar que el algoritmo siempre terminará ?

Eficiencia

Eficiencia mide el uso de recursos en su ejecución: Operaciones/Tiempo, y Memoria. Se expresa (normalmente)en forma de una función del tamaño de las entradas, que describe como el coste en operaciones y/o tiempo crece con el tamaño de las entradas

- **Operaciones/Tiempo:** Número total de operaciones a realizar en la ejecución del algoritmo. Tiempo es relativo y depende de la capacidad y tipo de ordenador y de la paralelizabilidad del algoritmo
  $\Rightarrow$ **Worst Case analysis**: en qué casos se puede disparar el tiempo de ejecución ?
- **Memoria:** Total espacio de almacenamiento (RAM ?) que requerirá la ejecución del algoritmo

Algoritmos y Estructuras de datos

La eficiencia de un algoritmos está fuertemente ligado a la estructura de datos que lo soporta.
Sobre estas estructuras de datos más significativas:

Colecciones (Sets, listas, Listas enladas), las Colas, y Árboles

los algoritmos ejecutarán acciones básicas como:

Insert,append, remove, pop, search, and sort, etc...

cuyo coste básico depende fuertemente de la estructura de datos utilizada y del lenguage e implementación concreta.

$\Rightarrow$ Veamos un caso simple de gestión matrices (notebook)

Además de los algoritmos más importantes, es el objetivo de este curso el conocer las estructuras de datos principales y saber elegir la más adecuada para cada algoritmo, con objeto de aumentar su eficiencia

Part II

Modelos básicos de Algoritmos

Paradigmas clásicos (1)

Fuerza Bruta

Explora *Ciegamente* todas las soluciones posibles.
Ej. Búsqueda secuencial

Iterativo

Incrementa la precisión del resultado hasta un objetivo dado (ej. Newton-Raphson)

Backtracking

Tipos de algoritmos que construyen la solución de forma incremntal, formulando hipótesis y abandonando el camino a la solución candidata tan pronto como éste se prueba incorrecto, explorando hipótesis alternativas hasta encontrar la solución correcta. (Ej. Sudokus, laberintos, etc..)

COMILLAS ICAI

Paradigmas clásicos (2)

Divide and Conquer

Divide el problema en segmentos menores que son más facilmente resolubles, y aplicando el proceso de división a las partes resultantes.
Secuencia $\Rightarrow$ **Divide** $\|$ **Conquer** $\|$ **Combine.**
Ej.Busqueda en lista ordenada)

Recursivo

El algoritmo se invoca a sí mismo sobre una subset de los datos. Necesita una condición de salida ! Ej. Cálculo de Factorial

MonteCarlo

Familia de algoritmos que samplean valores concretos de algoritmo sobre valores de entrada aleatoriamente escogidos, extrapolando soluciones aproximadas en base a las estadísticas de estos valores resultantes. Ej. Estimación de la media de una distribución no conocida.

Paradigmas clásicos: Secuencial vs Paralelo

Secuencial

Existe una única secuencia de cálculo en cada momento dado.

Paralelo

El algoritmo prescribe varias líneas de proceso que se ejecutan al mismo tiempo (ej. Proceso de Imágenes)

Cuidado:

- Los Threads de cálculo pueden colisionar en el acceso a los datos !
- Si el algoritmo lo permite, es posible acelerar hasta $\times n$ veces donde n es el número de procesadores[a].

[a] Pero esto no es siempre posible

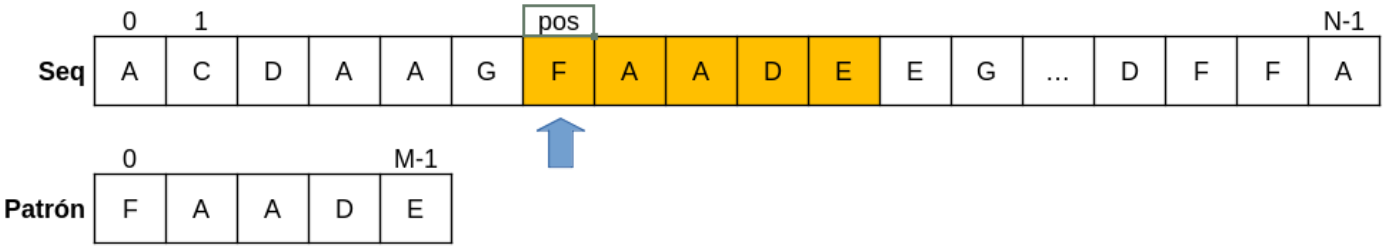
Fuerza Bruta

Consiste en resolver el problema "rudamente"

- Probar sistemáticamente todas y una de las posibles soluciones al problema y elegir
- La falta de habilidad o imaginación para mejorar el proceso es su característica fundamental
- Se confía en la capacidad de cálculo del ordenadorpara conseguir resolverlo
- Suele ser la primera versión del algoritmo, una cota superior de *"al menos se puede resolver en x tiempo"*
- Cuando usarlo:
  - No se dispone de mucho tiempo para pensar (desarrollo rápido)
  - Se ejecutará pocas veces v

Fuerza Bruta: ejemplo

Búsqueda de un patrón en una secuencia larga (ej. DNA)



- ¿Cuántos posibles encajes de P en A hay que probar?
  ⇒ $N - M + 1$
- ¿Cuál es el peor caso?
  ⇒ Cuando el match no se da, o se da en las últimas M posiciones
- En el peor caso la versión de fuerza bruta requiere del orden de $N \times M$?
- Cómo es dicha versión?
  ⇒ Ver algorimtmo a la derecha

```
function BruteForceSearch(seq,pattern)
    i, j ← 0
    M ← len(pattern)
    N ← len(seq)
    while i < N & j < M do
        if seq[i] = pattern[j] then
            i ← i + 1; j ← j + 1
        else
            i ← i − j + 1; j ← 0
        end if
    end while
    if j=M then
        pos ← i − M
    else
        pos ← None
    end if
    return pos
end function
```

Divide and Conquer

- Dividir el problema en subproblemas, similares al original, pero de tamaño menor.
- Se solucionan los problemas recursivamente. Si es suficientemente pequeño se resuelve directamente.
- Se combinan esas soluciones para construir la solución al problema original.

Algorithm 1 sort using D& C

```
1:  function SORT(seq)
2:      if len(seq) = 2 then
3:          BASICSORT(seg)                ▷ Makes Simple linear sort of short sequence
4:          return
5:      else
6:          left ← LEFTHALF(seq)          ▷ Returns first half of seq
7:          right ← RIGHTHALF(seq)        ▷ Returns second half of seq
8:          SORT(left)                    ▷ recursive call
9:          SORT(right)                   ▷ id.
10:         MERGE(left,half)              ▷ Merge sorted sequences into one keeping sort order
11:     end if
12: end function
```

Recursividad

Cuando el problema se puede descomponer en la aplicación del mismo algoritmo a un subset de los datos

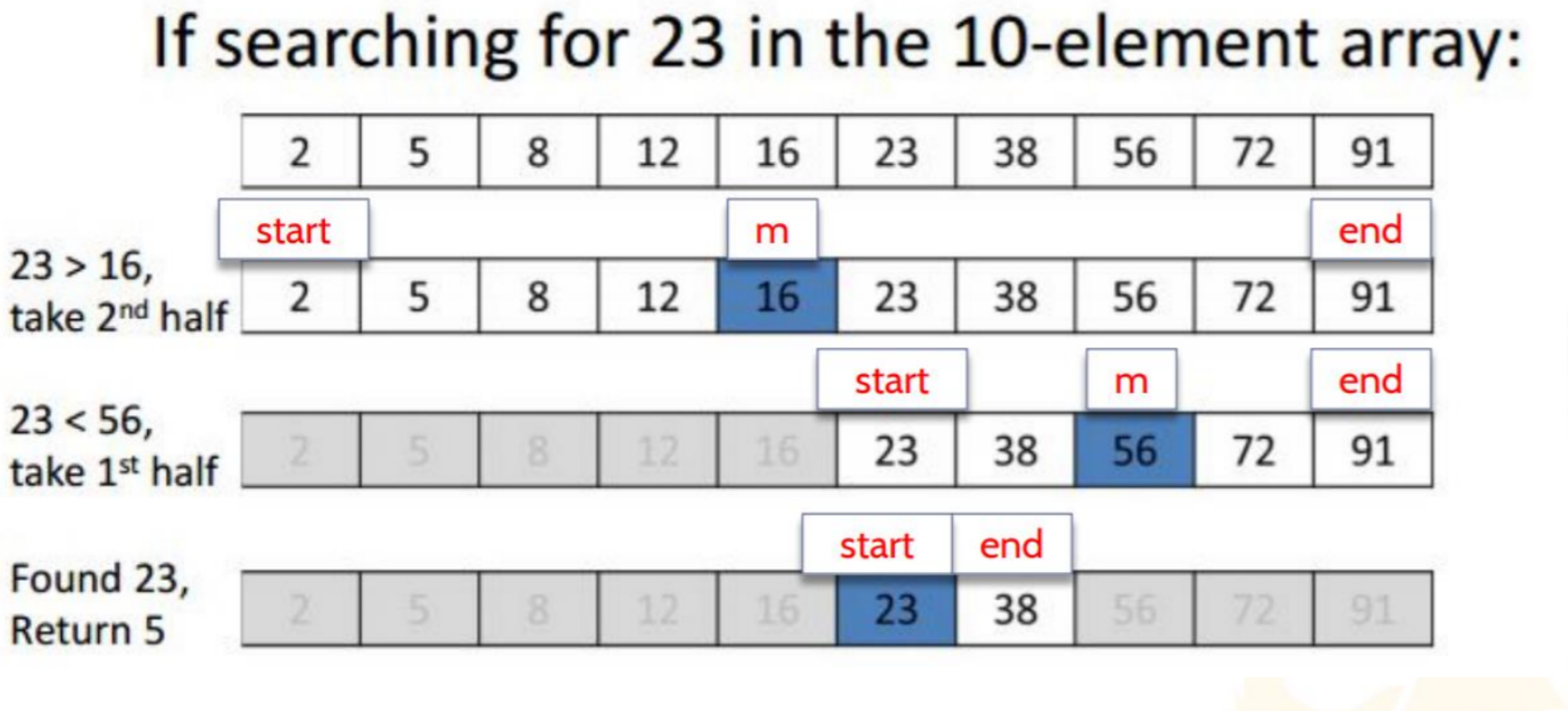
- Son llamadas a la una función dentro de la propia función
- Cuidado con no generar bucles infinitos ⇒ Debe tener una condición de parada o salida
- Ej. Factorial, serie Fibonnaci, etc. . .

Algorithm 2 Factorial (Recursive)

```
1: function FACTORIAL(n)
2:     if n = 1 then
3:         Return 1
4:     else
5:         Return n*FACTORIAL(n − 1)
6:     end if
7: end function
```

```python
def factorial(n):
    """Implements Factorial in
       a recursive manner
    """
    if n == 1:
        return 1
    else:
        return n*factorial(n-1)
```

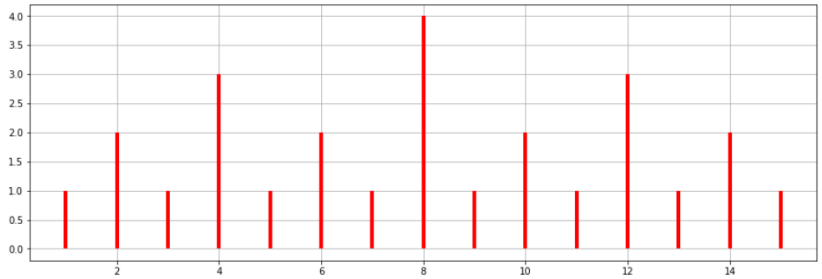
Recursividad (2)



Busqueda recursiva en listas ordenadas es una forma eficiente de búsqueda que explota la ordenación previa de la lista para utilizar un método tipo *bisección* para decidir en qué mitad, y así sucesivamente, está el objeto buscado

Recursividad + Divide & Conquer

En algunos casos el modelo de **Divide & Conquer** se complementa con la propia recursión, cuando la solución es autosimilar... Por ejemplo para dibujar la siguiente regla



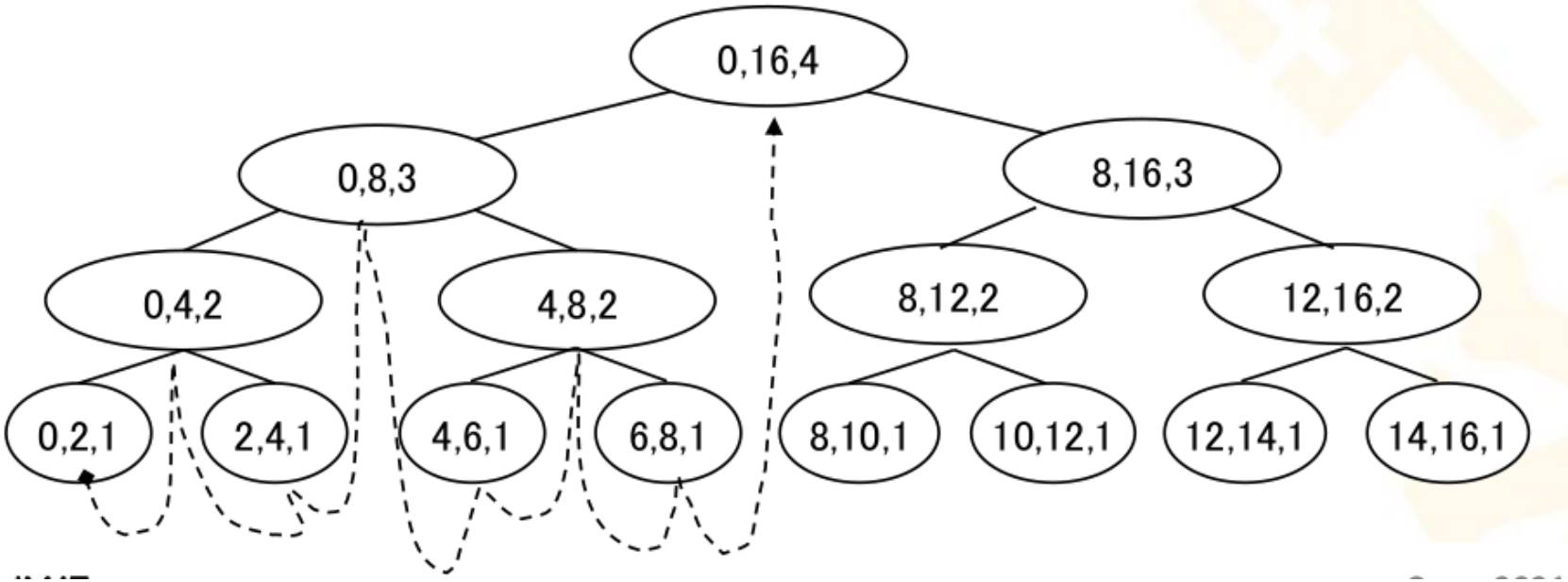
Basta con ejecutar el siguiente código:

```
1  def drawMark(center, levels,ax):
2      ax.vlines(center,0.0,levels, lw=1, color='r')
3
4  def drawRule(first, last, levels,ax=None):
5      # first case..
6      if not ax:
7          fig, ax = plt.subplots(figsize=(15,5))
8          ax.grid()
9
10     if levels > 0:
11         center = (last + first)/2
12         drawRule(first, center, levels-1,ax)
13         drawMark(center, levels,ax)
14         drawRule(center, last, levels-1,ax)
```

COMILLAS ICAI

Recursividad + Divide & Conquer (2)

Lo que resulta en el siguiente arbol de llamadas recursivo



MonteCarlo

Los algoritmos de MonteCarlo se alimentan de una secuencia aleatoria de los parámetros de entrada y extraen consecuencias con `significado Estadístico` en base a los resultados de un algoritmo relaccionado con el problema. Ejemplo: Podemos calcular el número π en base al porcentaje de puntos (x, y)

donde $x, y \in U(0, 1)$ que cumplan la condición de que $\sqrt{x^2 + y^2} < 1$

It is just one line of Python for a very inefficient calculation of π

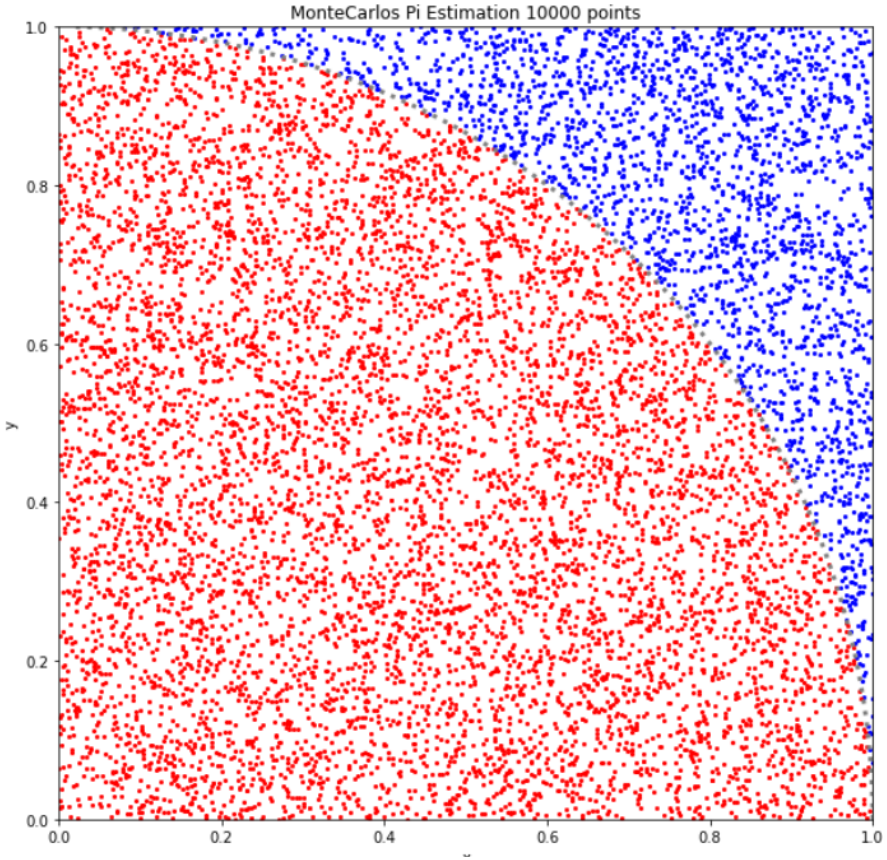
```
1  def norm(x,y): return math.sqrt(x*x + y*y)
2  def MC_PI(n) -> float:
3      return 4*sum( True for _ in range(n) if norm(random.random(),random.random()) <= 1.0)/ n
4  MC_PI(10000000)
```

✓ 4.8s

```
3.1427256
```

MonteCarlo (2)

Con un montón de datos detrás...



Estos algoritmos son frecuentemente utilizados en el cálculo de integrales no resolubles analíticamente !

COMILLAS ICAI

Eof Tema 2: Gracias !

"An algorithm matched us as soul mates, and yet it can't suggest a movie we both want to watch."

COMILLAS ICAI