Diseño de Algoritmos

May 28, 2024

Definición de algoritmo

El concepto intuitivo de algoritmo, lo tenemos prácticamente todos: Un algoritmo es una serie finita de pasos para resolver un problema.
Hay que hacer énfasis en dos aspectos para que un algoritmo exista:
1. El número de pasos debe ser finito. De esta manera el algoritmo debe terminar en un tiempo finito con la solución del problema,
2. El algoritmo debe ser capaz de determinar la solución del problema.
De este modo, podemos definir algoritmo como un "conjunto de reglas operacionales inherentes a un cómputo". Se trata de un método sistemático, susceptible de ser realizado mecánicamente, para resolver un problema dado.

Características de un algoritmo

1. Entrada: definir lo que necesita el algoritmo
2. Salida: definir lo que produce.
3. No ambiguo: explícito, siempre sabe qué comando ejecutar.
4. Finito: El algoritmo termina en un número finito de pasos.
5. Correcto: Hace lo que se supone que debe hacer. La solución es correcta
6. Efectividad: Cada instrucción se completa en tiempo finito. Cada instrucción debe ser lo suficientemente básica como para que en principio pueda ser ejecutada por cualquier persona usando papel y lápiz.
7. General: Debe ser lo suficientemente general como para contemplar todos los casos de entrada.

Algoritmos vs. Problemas vs. Programas

Un **algoritmo** es un método o proceso seguido para resolver un problema.
Un **problema** es una función o asociación de entradas con salidas.
Nota: Un mismo problema puede tener varios algoritmos.
Un **programa** es una instanciación de un algoritmo en un lenguaje de programación.

Resolución de problemas

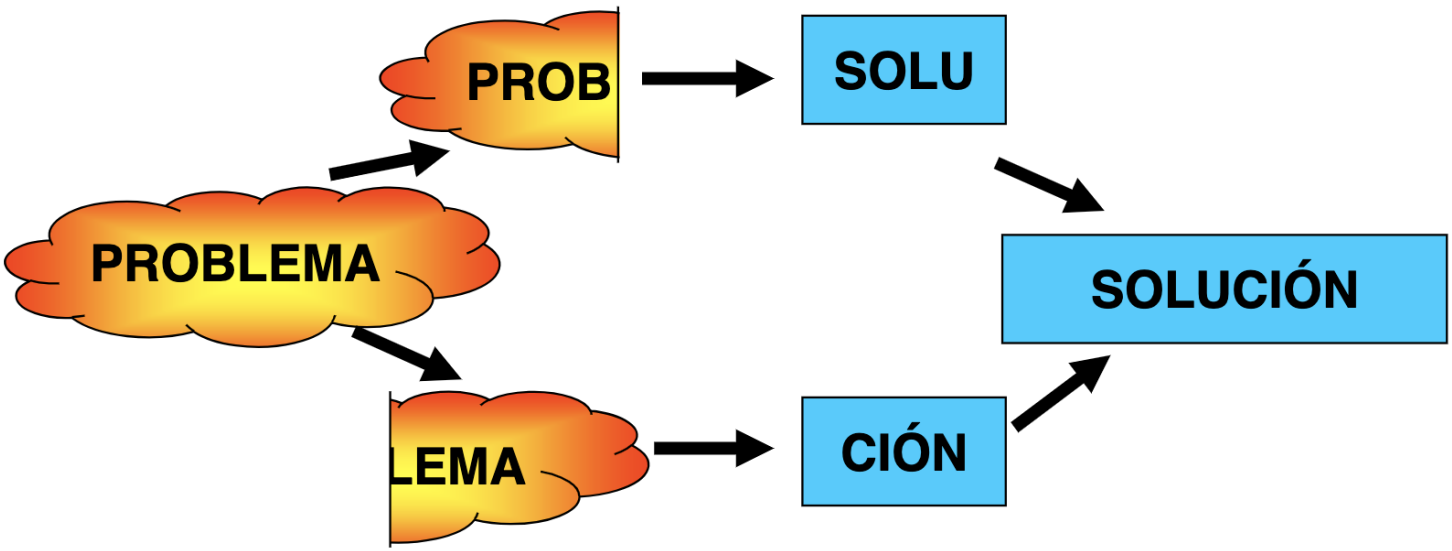
La fase de resolución del problema se puede descomponer en tres etapas:
o Análisis de alternativas y selección de la solución.
o Especificación detallada del procedimiento solución.
o Adopción o utilización de una herramienta para su implementación, si es necesaria.
En el campo de las ciencias de la computación la solución de problemas se describe mediante el diseño de algoritmos, los cuales posteriormente se implementan como programas. Los programas son procedimientos que solucionan problemas y que se expresan en un lenguaje conocido por el computador.
El resto de la unidad exploraremos distintas estrategias para elaborar algoritmos.

Divide y Vencerás

Muchos algoritmos útiles son recursivos, es decir, se llaman a sí mismos para resolver subproblemas más pequeños.
La técnica Divide y Vencerás permite generar algoritmos recursivos para resolver una amplia variedad de problemas. Consiste en:

1. Descomponer un problema en un conjunto de subproblemas más pequeños.
2. Resolver los subproblemas más pequeños.
3. Combinar las soluciones de los subproblemas más pequeños para obtener la solución general.

Divide y Vencerás



Divide y Vencerás

Muchos algoritmos útiles tienen una estructura recursiva, de modo que para resolver un problema se llaman recursivamente a sí mismos una o más veces para solucionar subproblemas muy similares. Esta estructura obedece a una estrategia divide-y-vecerás (*divide & conquer*, o simplemente D&C), en que se ejecuta tres pasos en cada nivel de la recursión:

1. **Dividir** el problema en varios subproblemas similares al problema original pero de menor tamaño;
2. **Conquistar**: Resolver recursivamente los subproblemas. Si los tamaños de los subproblemas son suficientemente pequeños, entonces resuelven los subproblemas de manera directa; y luego,
3. **Combinar** estas soluciones para crear una solución al problema original.

Divide y Vencerás

La estructura general de un algoritmo de este tipo es

```
def resolver(problema):
    if es_caso_base(problema):
        return resolver_caso_base(problema)

    subproblema1, subproblema2 = dividir(problema)
    solucion1 = resolver(subproblema1)
    solucion2 = resolver(subproblema2)

    return combinar(solucion1, solucion2)

print(resolver(problema))
```

Solo necesitamos identificar

- Una función `es_caso_base` que verifique si estamos en un caso base.
- Una función `resolver_caso_base` que le da solución a los casos base.
- Una función `dividir` que nos indique como dividir el problema en varios subproblemas relacionados, pero de menor tamaño.
- Una función `combinar` que nos permita juntar las soluciones parciales del caso anterior para obtener la solución.

Notas

- La cantidad de subproblemas en el que dividiremos el problema original puede variar.

Divide y Vencerás

Para aplicar la estrategia Divide y Vencerás es necesario que se cumplan tres condiciones:
· Debemos elegir cuidadosamente cuales serán nuestros casos base, es decir, cuando resolveremos el problema sin necesidad de dividir. · Tiene que ser posible descomponer el caso en subcasos y recomponer las soluciones parciales de forma eficiente. · Los subcasos deben ser, en lo posible, aproximadamente del mismo tamaño.

Divide y Vencerás

Problema Calcular el enésimo número de Fibonacci.

```
def es_caso_base(p):
    return True if p <=1 else False

def resolver_caso_base(p):
    return p
```

Divide y Vencerás

Problema Calcular el enésimo número de Fibonacci.

```
def dividir(p):
    return p - 1, p - 2

def combinar(s1, s2):
    return s1 + s2
```

Divide y Vencerás

Problema Calcular el enésimo número de Fibonacci.

```
def resolver(problema):
    if es_caso_base(problema):
        return resolver_caso_base(problema)
    subproblema1, subproblema2 = dividir(problema)
    solucion1 = resolver(subproblema1)
    solucion2 = resolver(subproblema2)

    return combinar(solucion1, solucion2)

print(resolver(problema))
```

Divide y Vencerás

Ventajas

- Por lo general, da como resultado algoritmos elegantes y eficientes.
- Como los subproblemas son independientes, si poseemos una máquina con más de un procesador podemos *paralelizar* la resolución de los subproblemas, acortando así el tiempo de ejecución de la solución.

Divide y Vencerás

Desventajas

- Los subproblemas en los que dividimos el problema original deben ser *independientes*, de lo contrario, intentar aplicar Divide y Vencerás estará condenado al fracaso.
- Se necesita cierta intuición para ver cuál es la mejor manera de descomponer un problema en partes. Esto solo se consigue con la práctica.
- A veces, como en el caso de Fibonacci, los subproblemas son *independientes* pero no *disjuntos*, lo que puede ocasionar trabajo extra al recomputar muchas veces el mismo valor. La técnica de **Programación Dinámica** es más apropiada en estos casos, pero no la veremos en este curso.
- Si bien los algoritmos que genera utilizar esta técnica son eficientes, demostrar tal eficiencia requiere matemáticas avanzadas.

Divide y vencerás

Ejercicio 1

Escriba un algoritmo divide y vencerás para ordenar una lista de números enteros

Ejercicio 2

Dada una lista ordenada de números enteros, determinar eficientemente si un número se encuentra en ella o no.