Algoritmos

Computadora vs Computacion

La computacion es un concepto mas amplio que las computadoras. Se refiere a la resolucion de problemas mediante algoritmos, ya sea realizada por humanos o maquinas.

La historia del algoritmo

- El algoritmo es una serie finita de pasos para resolver un problema.
- Tiene dos aspectos claves:
    i. Finitud en el numero de pasos
    ii. Capacidad para determinar la solucion

Definicion

Conjunto de reglas operacionales inherentes a un computo, un metodo sistematico para resolver problemas.

Caracteristicas

1. Entrada: Define lo que necesita el algoritmo
2. Salida: Define lo que produce
3. No ambiguo: Explicito, siempre sabe que comando ejecutar.
4. Finito: Termina en un numero finito de pasos
5. Correcto: Hace lo que se supone que debe hacer, la solucion es correcta
6. Efectividad: Cada instruccion se completa en tiempo finito y es lo suficientemente basica para ejecutarse manualmente
7. General: Suficientemente general para contemplar todos los casos de entrada.

Algoritmos Vs Problemas Vs Programas

Un algoritmo es un metodo o proceso para resolver un problema Un problema es una funcion que asocia entradas con salidas, puede tener varios algoritmos Un programa es una instancia de un algoritmo en un lenguaje de programacion

Resolucion de problemas

Fases: Analisis de alternativas y selccion de la solucion. especificaion detallada del procedimiento y adopcion o utilizacion de herramientas para la implementacion. En ciencias de la ocmputacion la

solucion se describe mediante el diseño de algoritmos implementados como programas.

Los algoritmos son fundamentales en la resolucion de problemas computacionales y su diseño eficiente es esencial para la programacion y la computacion en general.

Divide y venceras

Es una estrategia para diseñar algoritmos recursivos 3 PASOS:

1. Descomponer el problema en subproblemas mas pequeños
2. Resolver los subproblemas
3. Combinar las soluciones para obtener la solucion general

Estructura general de algoritmos

```
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))
```

Requisitos para aplicar DyV

1. Elegir cuidadosamente el caso base
2. Posibilidad de descomponer y recomponer soluciones eficientemente
3. Subproblemas deben ser aproximadamente del mismo tamaño

Ventajas

- Algoritmos elegantes y eficientes.
- Paralelizacion posible si los subproblemas son independientes

Desventajas

- Subproblemas deben ser independientes
- Requiere intuicion para descomponer problemas
- En algunos casos puede generar trabajo extra al recomputar valores(ej: fibonacci)
- Demostrar eficiencia requiere matematicas avanzadas

Algoritmos Voraces

- Toman decisiones basadas en informacion inmediata sin considerar consecuencias futuras.
- Son usados en problemas de optimizacion
- Sus elementos claves son:
  - Conjunto de candidatos
  - Decisiones ya tomadas
  - funciones que determinan si un conunto es solucion y si es optima

Procedimiento

1. Inicializamos, la eleccion de solucion es vacia
2. En cada paso, se añade el mejor candidato no escogido segun funcion de seleccion
3. Se verifica si el conjunto resultante es una solucion al problema

Ejemplo:

- Problema: Dar cambio minimizando billetes
- Estrategia: entregar billete de denominacion mas alta posible sin pasasrse del valor.
- Ejemplo practico: Cliente gasta 960 pesos, paga con 1000, la mejor forma es dar de vuelto 2 billetes de 20 pesos

```
total = 20
candidatos = [1000, 500, 200, 100, 50, 20, 10] * 2

def es_solucion(eleccion_actual):
    return(sum(eleccion_actual) == total)

def elegir_candidato():
    return(max(candidatos))

def es_factible(eleccion):"
    return(sum(eleccion)<=total)

eleccion_actual = []
while not es_solucion(eleccion_actual):
    x = elegir_candidato()
    candidatos.remove(x)
    if es_factible(eleccion_actual + [x]):
        eleccion_actual.append(x)
print(eleccion_actual)
```

Ventajas

- Utiles en problemas de optimizacion
- Simpleza y eficiencia

Desventajas

- No siempre garantizan la solucion optima
- Necesita cuidado y a veces una demostracion matematica de su correcion

Busqueda Exhaustiva

Es una tecnica que implica enumerar sistematicamente todos los posibles candidatos para la solucion de un problema y verificar si cada candidato satisface la solucion.

Ejemplo: Un algoritmo de BE para encontrar un divisor de un numero natural, consiste en enumerar todos los enteros desde 2 hasta n-1 verificando si cada uno divide a n sin dejar resto.

```
N = 2047

def es_solucion(intento):
    if N % intento == 0:
        return true
    return False
def candidatos():
    for i in range(2,N):
        yield i

def resolver():
    for candidato in candidatos():
        if es_solucion(candidato):
            return candidato
    return -1 # error

solucion = resolver()
print(f'{solucion} divide a {N}')
```

Pasos para implementar:

1. Identificar el conjunto de candidatos a solucion
2. Representar formalmente cada candidato
3. Dar un orden logico a los elementos del conjunto de candidatos
4. Establecer el objetivo y cuando detener la busqueda
5. Escribir la funcion es_solucion en python
6. Escribir un generador que provea uno a uno los candidatos en el orden establecido
7. Iterar sobre los candidatos hasta encontrar una solucion

Ventajas

- Facil implementacion y lectura
- Adaptable para encontrar todas las soluciones a una cantidad especifica

Desventajas

- La cantidad de soluciones a explorar crece exponencialmente con el tamaño del problema
- Depende del poder de computo de la maquina

Para concluir es una estrategia directa y facil de entender, pero puede volverse impracticable para problemas grandes debido a su complejidad exponencial.

ß