Grupo 1

Primer parcial

Buen algoritmo

Un algoritmo es una secuencia de pasos para resolver un problema específico. Un buen algoritmo se caracteriza por ser simple, claro y eficiente.

Características

- Simplicidad: Fácil de entender, mantener y modificar.
- Claridad: Debe ser explícito en cuanto a sus operaciones.
- Eficiencia: Optimizado en tiempo y recursos, evaluado mediante notación Big O.

Tiempo de ejecución y recurrencias

El tiempo de ejecución de un programa, denotado como T(n), describe cómo varía su rendimiento en función del tamaño de la entrada n. Se utiliza la complejidad asintótica para prever el rendimiento a gran escala.

Las recurrencias describen el comportamiento de algoritmos recursivos, expresando su costo en función de subproblemas más pequeños.

Generalmente tiene la forma de:

$T(n) = a \cdot T(n/b) + f(n)$:

Complejidad asintótica y notación Big(O)

La complejidad asintótica es el estudio de cómo el tiempo de ejecución de un algoritmo varía conforme el tamaño de la entrada crece, especialmente cuando esta es muy grande.

La notación Big O (O) se utiliza para expresar la complejidad asintótica, proporcionando una medida estandarizada de la eficiencia de un algoritmo en términos de su peor caso.

Recursividad

La recursividad permite que una función se llame a sí misma para resolver subproblemas más pequeños. Se basa en:

Caso base: Detiene la recursión.

Llamada recursiva: Resuelve subproblemas más pequeños.

Divide y vencerás

Se aplica cuando un problema complejo puede descomponerse en subproblemas más simples del mismo tipo.

Pasos:

- Caso base
- Dividir el problema
- resolver recursivamente los subproblemas
- Combinar las soluciones para resolver el problema original

Quicksort

Algoritmo de ordenamiento.

- Se elige pivote
- menores → pivote → mayores.
- Repetir el proceso con las sublistas.
- Mejor caso = n . log n
- Peor caso = n^2

MergeSort

Algoritmo de ordenamiento por mezcla

- Dividir la lista en dos sublistas.
- Ordenar cada sublista recursivamente.
- Mezclar las dos sublistas.
- Complejidad = n . log n

Greedy

Decisión óptima local → Solución óptima global

- siempre óptimo: conjunto canónico

  monedas = {1, 5, 10, 25}

- no siempre óptimo

  monedas = {1, 3, 4}

```java
public static void Monto(int monto) {
    int[] billetes = {1,5,10,25};
    int[] cantidad = new int[billetes.length];
    for(int i = billetes.length - 1; i >= 0 ; i--) {
        cantidad[i]= monto/billetes[i];
        monto=monto-billetes[i]*cantidad[i];
    }
    for(int i = billetes.length - 1; i >= 0 ; i--) {
        if (cantidad[i]!=0) {
            System.out.println(cantidad[i] + " billetes de " + billetes[i]);
        }
    }
}
```

monto = 61

61/25 = 2 → 61 - 25*2 = 11

11/10 = 1 → 11 - 10*1 = 1

1 / 5 = 0 → 1 - 5*0 = 1

1 / 1 = 1 → 1 - 1*1 = 0

2 billetes de 25

1 billete de 10

1 billete de 1

MOCHILA FRACCIONAL

- mochila con peso W.
- cantidad de objetos n.
- cada objeto tiene un valor y un peso.

MOCHILA FRACCIONAL - MÉTODO GREEDY

- ordenar listas → valor / peso

	valor	peso	v / p
obj 1	12	4	3
obj 2	10	5	2
obj 3	11	3	3,6

- recorrer la lista
- entra → agrego
- no entra → fracción

obj3 - obj1 - obj2

Grafos

Existen los dirigidos y los no dirigidos:

- Dirigido: Solo de V1 a V2
- No dirigido: V1 a V2 y viceversa

Por otro lado están los ponderados y los no ponderados:

- Ponderados: las aristas tienen peso

(Están los conexos, los ciclos, los árboles)

Representación: lista y matriz de adyacencia

Árbol de Recubrimiento Mínimo (MST)

- Se trabaja con grafos ponderados
- Es un subgrafo conexo (incluye todos los V del grafo original)
- Árbol = no ciclos
- El costo total del árbol debe ser el mínimo posible

Se elige un nodo y se va buscando el camino de menor costo hasta el siguiente vértice. Ahora desde este vértice busco el siguiente camino de menor costo y así hasta recorrer todo el grafo (A menos que en el primer vértice hubiera otra camino menos costoso que todos los del siguiente)

Algoritmo de Prim y algoritmo de Kruskal

Prim: encuentra el MST de un grafo no dirigido

(Se agrega un nodo al MST, se busca la arista de menor peso que lo una con un nodo fuera del MST y se agrega el peso de la arista y el nodo al MST)

Kruskal: encuentra el MST de un grafo no dirigido

(Ordena las aristas por peso y las añade una a una al MST, sin ciclos (V-1 aristas). Al final queda todo el grafo cubierto igual que con Prim)

Complejidades

Prim es O(V^2) con matriz de adyacencia, donde V es la cantidad de vértices

(Más eficiente cuando E ≅ V^2) → grafos más densos

Kruskal es O(E log E), donde E es la cantidad de aristas. E log E es por el ordenar las aristas

(Más eficiente cuando E < V^2) → grafos más dispersos

Algoritmo de Dijkstra

Distancia mínima de un vértice origen (específico) a todos los vértices (grafo ponderado y no pesos negativos). Estrategia Greedy

Seleccionamos el origen, buscamos el camino más corto con sus vecinos. Primero se pone que el d[V origen] = ∞ (como que estuviera aislado) y de ahí se compara la distancia a sus vecinos (d[V] = 0 al principio, porque apunta a sí mismo, el resto de V van en infinito al principio)

Complejidad variable según cola de prioridad:

$O((V + E) \log V) \rightarrow$ Heap binario

$O(E + V \log V) \rightarrow$ Heap de Fibonacci

Programación dinámica

Es una técnica algorítmica utilizada para resolver problemas complejos dividiéndolos en subproblemas más pequeños y reutilizando las soluciones de estos.

Principios:

Subproblemas: Los mismos subproblemas aparecen repetidamente, por lo que se almacenan sus soluciones para evitar recalcular.

Estructura óptima: La solución óptima de un problema se construye a partir de las soluciones óptimas de sus subproblemas.

La complejidad suele ser polinómica, típicamente O(n^2) o $O(n \cdot W)$

Algoritmo de Floyd-Warshall

- camino más corto entre pares de vértices.
- matriz

  existe arista → peso

  no existe → infinito

  [i][i] → 0

- 3 ciclos → recorrer vértices

  → recorrer matriz