PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ
FACULTAD DE CIENCIAS E INGENIERÍA

ALGORITMOS AVANZADOS
Hoja de Enunciados
Heurísticas

Compilado por Andrés Melgar

---

1. Problema del Cambio de Monedas (*Coin Change* – Versión Voraz)

Contexto del problema

El problema del cambio de monedas consiste en determinar la mínima cantidad de monedas necesarias para formar una suma determinada, utilizando un conjunto de denominaciones disponibles. Este problema puede abordarse desde distintas perspectivas algorítmicas.

La versión general del problema pertenece a la clase **NP-completo** cuando se requiere determinar si existe una combinación exacta de monedas que suma un valor objetivo, especialmente si las denominaciones no están estructuradas de forma uniforme. Sin embargo, en su formulación como problema de optimización con ciertas condiciones sobre las monedas (por ejemplo, sistemas canónicos), puede resolverse eficientemente utilizando un enfoque **voraz (*greedy*)**.

El objetivo de este ejercicio es aplicar un algoritmo voraz para resolver este problema y discutir en qué condiciones dicho enfoque proporciona soluciones óptimas.

Enunciado del problema

Se tiene un conjunto de monedas con denominaciones positivas enteras y una cantidad total de dinero que se desea entregar como cambio. No hay límite en la cantidad de monedas de cada denominación disponible.

Dado:

- Un arreglo de denominaciones de monedas $D = \{d_1, d_2, \ldots, d_k\}$, con $d_i \in \mathbb{N}$ y $d_1 > d_2 > \ldots > d_k > 0$.
- Un valor entero positivo V, que representa la cantidad de dinero a entregar como cambio.

Se desea determinar:

- La cantidad mínima de monedas necesarias para dar el cambio exacto de V, utilizando una estrategia voraz.

Objetivo del ejercicio

El estudiante deberá implementar un algoritmo en **C++** que:

a) Ordene las denominaciones de mayor a menor (si no lo estuvieran).

b) Iterativamente seleccione la mayor moneda posible que no exceda el monto restante.

c) Calcule cuántas monedas de esa denominación se pueden usar sin exceder el total.

d) Repita el proceso hasta cubrir el valor total V o hasta agotar las denominaciones disponibles.

Además, se espera que el estudiante:

- Analice el comportamiento del algoritmo.
- Discuta casos en los que el enfoque voraz **no** produce una solución óptima (por ejemplo, conjunto de monedas $\{10, 6, 1\}$ y valor $V = 12$).
- Compare, si es posible, con una solución óptima basada en programación dinámica.

Ejemplo de entrada y salida

Entrada:

- Denominaciones: $\{25, 10, 5, 1\}$
- Valor a devolver: $V = 63$

Salida esperada:

- Monedas usadas: 25×2, 10×1, 1×3
- Total de monedas: 6

2. Problema del Camino Más Cortos con Pesos Positivos (Algoritmo de Dijkstra)

Contexto

El problema de caminos más cortos pertenece a la clase **P**, ya que existen algoritmos que lo resuelven en tiempo polinomial respecto al número de vértices y aristas del grafo. En particular, el algoritmo de Dijkstra es un algoritmo **voraz (*greedy*)** que resuelve este problema de forma eficiente cuando todos los pesos de las aristas son no negativos.

Este algoritmo es ampliamente utilizado en aplicaciones reales como sistemas de navegación GPS, redes de telecomunicaciones, enrutamiento de paquetes en Internet y planificación de rutas logísticas.

Enunciado

Se tiene un grafo dirigido o no dirigido $G = (V, E)$, donde V es el conjunto de vértices y E es el conjunto de aristas. Cada arista $(u, v) \in E$ tiene asociado un peso $w(u, v) \geq 0$, que representa el costo o distancia entre los vértices u y v.

Dado un vértice fuente $s \in V$, se desea encontrar la distancia mínima desde s hacia todos los demás vértices del grafo, así como el camino correspondiente para cada uno.

Objetivo

Diseñar e implementar un algoritmo voraz que permita:

- Calcular las distancias mínimas desde un nodo fuente s hacia todos los demás nodos del grafo.
- Construir los caminos más cortos correspondientes.

El algoritmo debe cumplir con los siguientes requerimientos:

a) Utilizar una estructura de datos eficiente para seleccionar el vértice con menor distancia estimada (por ejemplo, una cola de prioridad).

b) Ser implementado en lenguaje `C++`.

c) Validar que los pesos de las aristas sean no negativos antes de ejecutar el algoritmo.

Ejemplo de entrada

Considere el siguiente grafo no dirigido con pesos positivos:

Arista	Peso
A – B	4
A – C	2
B – C	1
B – D	5
C – D	8
C – E	10
D – E	2

Origen: A.

Resultado esperado

- Distancia mínima desde A a cada vértice.
- Camino correspondiente (por ejemplo, de A a E: $A \to C \to B \to D \to E$).

3. Problema del Cajero Viajante con Aproximación Greedy (TSP Heurístico)

Contexto

El Problema del Cajero Viajante (TSP por las siglas en inglés de *Traveling Salesman Problem*) consiste en encontrar el ciclo hamiltoniano de costo mínimo en un grafo completo ponderado. Es un problema clásico de optimización combinatoria y pertenece a la clase de problemas **NP-Hard**, ya que no se conoce un algoritmo en tiempo polinomial que lo resuelva en el caso general.

Debido a su complejidad, se recurre a soluciones aproximadas mediante heurísticas. Una de las más conocidas es el algoritmo **voraz (*greedy*)**, que si bien no garantiza la solución óptima, produce resultados razonables en un tiempo eficiente.

Enunciado del problema

Dado un conjunto de ciudades, y las distancias entre cada par de ellas (simétricas, positivas y que satisfacen la desigualdad triangular), se desea encontrar una ruta que:

- Comience en una ciudad origen,
- Visite exactamente una vez cada una de las demás ciudades,
- Y retorne a la ciudad de origen,

de modo tal que el recorrido total sea mínimo.

Objetivo

Implementar una heurística voraz para aproximar la solución del TSP. La heurística consiste en:

a) Empezar desde una ciudad arbitraria (por ejemplo, la ciudad 0),

b) En cada paso, elegir la ciudad más cercana que aún no ha sido visitada,

c) Repetir hasta que todas las ciudades hayan sido visitadas,

d) Finalmente, regresar a la ciudad de origen.

Observaciones adicionales

- Esta heurística se conoce como ***Nearest Neighbor Algorithm***.
- La solución obtenida puede ser subóptima, pero su simplicidad y eficiencia la hacen útil en escenarios donde se requiere una solución rápida.
- El algoritmo tiene una complejidad temporal de $O(n^2)$, donde n es el número de ciudades.

Ejemplo

Supongamos que se tiene el siguiente conjunto de 5 ciudades, y sus distancias se representan en la siguiente matriz simétrica:

$$D = \begin{bmatrix} 0 & 2 & 9 & 10 & 7 \\ 2 & 0 & 6 & 4 & 3 \\ 9 & 6 & 0 & 8 & 5 \\ 10 & 4 & 8 & 0 & 6 \\ 7 & 3 & 5 & 6 & 0 \end{bmatrix}$$

El objetivo es recorrer todas las ciudades partiendo desde la ciudad 0, utilizando el algoritmo *Nearest Neighbor*.

Ejecución paso a paso:

- Ciudad actual: 0. Ciudades no visitadas: {1, 2, 3, 4}. Ciudad más cercana: 1 (distancia 2).
- Ciudad actual: 1. Ciudades no visitadas: {2, 3, 4}. Ciudad más cercana: 4 (distancia 3).
- Ciudad actual: 4. Ciudades no visitadas: {2, 3}. Ciudad más cercana: 2 (distancia 5).
- Ciudad actual: 2. Ciudades no visitadas: {3}. Ciudad más cercana: 3 (distancia 8).
- Ciudad actual: 3. Regresar a ciudad inicial 0 (distancia 10).

Ruta: $0 \to 1 \to 4 \to 2 \to 3 \to 0$
Distancia total: $2 + 3 + 5 + 8 + 10 = 28$

4. Problema de Selección de Actividades (*Activity Selection Problem*)

Contexto

El Problema de Selección de Actividades es un problema clásico de optimización que pertenece a la clase **P**, dado que existe una solución óptima que puede encontrarse en tiempo polinomial mediante un algoritmo **voraz**. Su resolución eficiente es posible gracias a la estructura del problema, que permite la toma de decisiones locales óptimas que conducen a una solución global óptima.

Enunciado

Se tiene un conjunto de n actividades $\{A_1, A_2, \ldots, A_n\}$, donde cada actividad A_i tiene un tiempo de inicio s_i y un tiempo de finalización f_i tal que $0 \leq s_i < f_i$. Dos actividades son compatibles si sus intervalos de tiempo no se sobrelapan.

El objetivo es seleccionar el subconjunto más grande posible de actividades mutuamente compatibles, de modo que se puedan realizar sin conflictos de horario.

Objetivo

Diseñar un algoritmo voraz que seleccione el máximo número de actividades compatibles, basándose en la estrategia de seleccionar siempre la actividad que termine más temprano (es decir, con el menor f_i posible), eliminando del conjunto aquellas que se solapan con ella.

El algoritmo debe cumplir con las siguientes condiciones:

- Procesar el conjunto de actividades en tiempo $O(n \log n)$ debido a la necesidad de ordenarlas por tiempo de finalización.
- Seleccionar iterativamente las actividades que no generen conflicto con las previamente seleccionadas.
- Retornar el conjunto de actividades seleccionadas y la cantidad total de actividades compatibles.

Ejemplo

Considérese el siguiente conjunto de $n = 6$ actividades, cada una con un tiempo de inicio s_i y un tiempo de finalización f_i:

Actividad	Inicio (s_i)	Final (f_i)
A_1	1	4
A_2	3	5
A_3	0	6
A_4	5	7
A_5	8	9
A_6	5	9

3
2
6
2
1
4

Paso 1: Ordenar las actividades por tiempo de finalización (f_i) ascendente:

$$A_1(1,4),\ A_2(3,5),\ A_3(0,6),\ A_4(5,7),\ A_5(8,9),\ A_6(5,9)$$

Paso 2: Aplicar el algoritmo voraz:

- Seleccionar A_1 (termina en 4).
- A_2 empieza en 3 $\Rightarrow$ se solapa con A_1 $\Rightarrow$ descartar.
- A_3 empieza en 0 $\Rightarrow$ se solapa con A_1 $\Rightarrow$ descartar.
- A_4 empieza en 5 $\Rightarrow$ es compatible con A_1 $\Rightarrow$ seleccionar.
- A_5 empieza en 8 $\Rightarrow$ es compatible con A_4 $\Rightarrow$ seleccionar.
- A_6 empieza en 5 $\Rightarrow$ se solapa con A_4 $\Rightarrow$ descartar.

Actividades seleccionadas: A_1, A_4, A_5
Resultado: Máximo subconjunto de actividades compatibles = 3 actividades.
