

# Exposición: Problemas de Análisis y Diseño de Algoritmos

## Brute Force vs Greedy

Delgado Romero, Gustavo

Torres Reategui, Joaquin

Quintanilla Quispe, Dylan

Universidad Nacional de Ingeniería  
CC371 - Análisis y Diseño de Algoritmos

25 de septiembre de 2025

# Tabla de Contenidos

- 1 Problema 1: Activity Selection Problem
- 2 Problema 2: Graph Coloring
- 3 Problema 3: Job Scheduling with Deadlines
- 4 Problema 4: Huffman Codes
- 5 Conclusiones y Recomendaciones

## Problemas a Desarrollar

- ① **Activity Selection Problem** - Selección de actividades
- ② **Graph Coloring** - Coloreo de grafos
- ③ **Job Scheduling with Deadlines** - Programación de trabajos
- ④ **Huffman Codes** - Códigos de Huffman

## Enfoques Comparados

- **Fuerza Bruta:** Solución óptima garantizada, complejidad exponencial
- **Greedy:** Solución eficiente, no siempre óptima, complejidad polinomial

# Tabla de Contenidos

## 1 Problema 1: Activity Selection Problem

- Enunciado
- Enfoque de Fuerza Bruta
- Enfoque Greedy
- Análisis comparativo de complejidad

## 2 Problema 2: Graph Coloring

## 3 Problema 3: Job Scheduling with Deadlines

## 4 Problema 4: Huffman Codes

## 5 Conclusiones y Recomendaciones

# Activity Selection Problem - Enunciado

## Descripción del Problema

Seleccionar el máximo número de actividades mutuamente compatibles de un conjunto dado. Cada actividad tiene:

- Tiempo de inicio y finalización
- Compatibilidad: No deben solaparse en el tiempo
- Objetivo: Maximizar el número de actividades seleccionadas

## Ejemplo

Actividades:  $[(1,2), (3,4), (0,6), (5,7), (8,9), (5,9)]$

Solución óptima: 4 actividades

# Activity Selection - Fuerza Bruta

## Estrategia

- Generar todos los subconjuntos posibles de actividades
- Verificar compatibilidad para cada subconjunto
- Seleccionar el subconjunto válido de mayor tamaño

## Complejidad

$O(2^n \cdot n^2)$  - Exponencial

# Activity Selection - Pseudocódigo Fuerza Bruta

```
1: [Entrada:] Un arreglo activities de  $n$  elementos donde cada elemento es una tupla (start, end)
2: [Salida:] El subconjunto de máximo tamaño de actividades mutuamente compatibles
3:  $n \leftarrow$  longitud de activities                                ▷ Cantidad de actividades
4: best  $\leftarrow$  lista vacía                                ▷ Mejor subconjunto encontrado                                ▷ Probar todos los subconjuntos posibles
5: for  $r = 1$  to  $n$  do
6:   for cada subset en combinations(activities,  $r$ ) do
7:     if is_valid(subset)  $\wedge$   $|subset| > |best|$  then                                ▷ Compatibilidad y tamaño
8:       best  $\leftarrow$  subset                                ▷ Actualizar mejor solución
9:     end if
10:  end for
11: end for return best
```

# Activity Selection - Ejemplo Fuerza Bruta I

Actividades:  $[(1,3), (2,5), (4,6)]$

**Paso 1:** Identificar actividades  $A=(1,3)$ ,  $B=(2,5)$ ,  $C=(4,6)$

**Paso 2:** Subconjuntos de tamaño 1

- $\{A\}$ ,  $\{B\}$ ,  $\{C\}$ : Todos válidos

**Mejor hasta ahora:** tamaño 1

**Paso 3:** Subconjuntos de tamaño 2

- $\{A,B\}$ :  $3 > 2 \rightarrow$  Se solapan  $\rightarrow$  Inválido
- $\{A,C\}$ :  $3 \leq 4 \rightarrow$  No se solapan  $\rightarrow$  Válido
- $\{B,C\}$ :  $5 > 4 \rightarrow$  Se solapan  $\rightarrow$  Inválido

**Mejor hasta ahora:**  $\{A,C\}$

**Paso 4:** Subconjunto de tamaño 3

- $\{A,B,C\}$ : Hay solapamientos  $\rightarrow$  Inválido



Solución óptima

$\{A,C\}$  con 2 actividades

## Estrategia Óptima

- Ordenar actividades por tiempo de finalización
- Seleccionar siempre la siguiente actividad que termina primero y no se solape

## Complejidad

$O(n \log n)$  - Polinomial (óptimo)

# Activity Selection - Pseudocódigo Greedy

```
1: [Entrada:] Arreglos start y finish de  $n$  elementos con tiempos de inicio y fin
2: [Salida:] El número máximo de actividades mutuamente compatibles
3: activities  $\leftarrow$  lista de tuplas (start[i], finish[i]) para  $i = 0..n - 1$ 
4: Ordenar activities por tiempo de finalización ascendente
5: count  $\leftarrow 1$ 
6: j  $\leftarrow 0$ 
7: for  $i = 1$  to  $n - 1$  do
8:   if activities[i].start > activities[j].finish then
9:     count  $\leftarrow$  count + 1
10:    j  $\leftarrow i$ 
11:   end if
12: end for return count
```

- ▷ Crear pares inicio-fin
- ▷ Clave para optimalidad
- ▷ Al menos una actividad puede realizarse
- ▷ Índice de la última actividad seleccionada
- ▷ Verifica no solapamiento
- ▷ Selecciona actividad
- ▷ Actualiza última seleccionada

# Activity Selection - Ejemplo Greedy I

Actividades = [(1, 2), (3, 4), (0, 6), (5, 7), (8, 9), (5, 9)]

**Paso 1:** Crear tuplas y ordenar por fin: (1,2), (3,4), (0,6), (5,7), (8,9), (5,9)

## Iteraciones

- $i=1$ : (3,4). ¿ $3 > 2$ ? Sí → Seleccionar, count=2,  $j=1$
- $i=2$ : (0,6). ¿ $0 > 4$ ? No → Omitir
- $i=3$ : (5,7). ¿ $5 > 4$ ? Sí → Seleccionar, count=3,  $j=3$
- $i=4$ : (8,9). ¿ $8 > 7$ ? Sí → Seleccionar, count=4,  $j=4$
- $i=5$ : (5,9). ¿ $5 > 9$ ? No → Omitir

## Resultado Final

4 actividades: (1,2), (3,4), (5,7), (8,9)

Enfoque	Complejidad	Optimalidad
Fuerza Bruta	$O(2^n \cdot n^2)$	✓
Greedy	$O(n \log n)$	✓

## Conclusión

- Greedy es óptimo y eficiente para este problema
- Fuerza bruta solo práctico para instancias pequeñas ( $n < 20$ )
- El ordenamiento por tiempo de finalización garantiza optimalidad

# Tabla de Contenidos

## 1 Problema 1: Activity Selection Problem

## 2 Problema 2: Graph Coloring

- Enunciado
- Enfoque de Fuerza Bruta
- Enfoque Greedy
- Análisis comparativo de complejidad

## 3 Problema 3: Job Scheduling with Deadlines

## 4 Problema 4: Huffman Codes

## 5 Conclusiones y Recomendaciones

## Descripción del Problema

Asignar colores a vértices de un grafo tal que:

- Vértices adyacentes tengan colores diferentes
- Minimizar el número de colores usado (número cromático)
- Problema NP-completo en general

## Aplicaciones

- Scheduling de procesos
- Asignación de registros
- Sudoku

# Graph Coloring - Fuerza Bruta

## Estrategia

- Probar todas las asignaciones de  $k$  colores a  $n$  vértices
- Verificar validez para cada asignación
- Encontrar el mínimo  $k$  que permita coloreo válido

## Complejidad

$O(k^n \cdot n \cdot m)$  - Exponencial



# Graph Coloring - Pseudocódigo Fuerza Bruta

```
1: [Entrada:] Grafo  $G$  (lista de adyacencia), número máximo de colores  $k$ 
2: [Salida:] Asignación válida de colores o null
3: for cada assignment en product(range( $k$ ), repeat =  $n$ ) do
4:   if is_valid( $G$ , assignment) then
5:     end if
6: end for return null
```

- ▷ Todas las asignaciones de  $k$  colores
- ▷ Verifica restricción de coloreo **return** *assignment*

```
1: function IS_VALID( $G$ , colors)
2:   for  $u = 0$  to  $|G| - 1$  do
3:     for cada  $v$  en  $G[u]$  do
4:       if colors[ $u$ ] = colors[ $v$ ] then return false
5:       end if
6:     end for
7:   end for return true
8: end function
```

- ▷ Recorrer todos los vértices
- ▷ Revisar vecinos
- ▷ Vértices adyacentes con mismo color

## Grafo con 3 vértices: Triángulo

- **k=2**: Probar  $2^3 = 8$  asignaciones, ninguna válida
- **k=3**: Probar  $3^3 = 27$  asignaciones
- Encontrar solución:  $[0,1,2]$  - 3 colores

## Limitaciones

- 10 vértices con 3 colores:  $3^{10} = 59,049$  pruebas
- 20 vértices:  $3^{20} > 3.4$  billones de pruebas

## Estrategia

- Procesar vértices en orden fijo
- Asignar el menor color disponible no usado por vecinos
- No garantiza número cromático mínimo

## Complejidad

$O(V^2 + E)$  - Polinomial

# Graph Coloring - Pseudocódigo Greedy

```
1: [Entrada:] Grafo  $G$  (lista de adyacencia) con  $V$  vértices
2: [Salida:] Asignación válida de colores
3:  $result \leftarrow$  arreglo de tamaño  $V$  inicializado en  $-1$ 
4:  $result[0] \leftarrow 0$ 
5:  $available \leftarrow$  arreglo booleano de tamaño  $V$  en false
6: for  $u = 1$  to  $V - 1$  do
7:   for  $i = 0$  to  $|adj[u]| - 1$  do
8:     if  $result[adj[u][i]] \neq -1$  then
9:        $available[result[adj[u][i]]] \leftarrow \text{true}$ 
10:    end if
11:  end for
12:   $cr \leftarrow 0$ 
13:  while  $cr < V$  do
14:    if  $available[cr] = \text{false}$  then
15:      break
16:    end if
17:     $cr \leftarrow cr + 1$ 
18:  end while
19:   $result[u] \leftarrow cr$ 
20:  for  $i = 0$  to  $|adj[u]| - 1$  do
21:    if  $result[adj[u][i]] \neq -1$  then
22:       $available[result[adj[u][i]]] \leftarrow \text{false}$ 
23:    end if
```

- ▷ Colores por vértice
- ▷ Color al primer vértice
- ▷ Colores ocupados por vecinos
- ▷ Marcar colores de vecinos como no disponibles

▷ Encontrar el primer color disponible

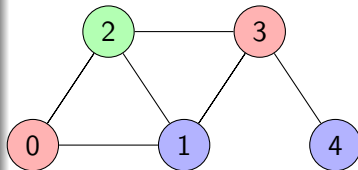
▷ Asignar color encontrado

▷ Resetear disponibilidad para el siguiente vértice

# Graph Coloring - Ejemplo Greedy

## Grafo de 5 vértices

- V0: Color 0
- V1: Color 1 (vecino de V0)
- V2: Color 2 (vecino de V0,1)
- V3: Color 0 (vecino de V1,2)
- V4: Color 1 (vecino de V3)



## Resultado

3 colores usados (óptimo para este grafo)

Enfoque	Complejidad	Optimalidad
Fuerza Bruta	$O(k^n \cdot n \cdot m)$	✓
Greedy	$O(V^2 + E)$	×

## Conclusión

- Fuerza bruta garantiza optimalidad pero es impráctica
- Greedy es eficiente pero puede usar más colores de los necesarios
- En la práctica: Greedy para grafos grandes, heurísticas avanzadas para mejor aproximación

# Tabla de Contenidos

- 1 Problema 1: Activity Selection Problem
- 2 Problema 2: Graph Coloring
- 3 Problema 3: Job Scheduling with Deadlines
  - Enunciado
  - Enfoque de Fuerza Bruta
  - Enfoque Greedy
  - Análisis comparativo de complejidad
- 4 Problema 4: Huffman Codes
- 5 Conclusiones y Recomendaciones

## Descripción del Problema

Programar trabajos con:

- Deadline: Tiempo límite para completar
- Profit: Ganancia por completar a tiempo
- Cada trabajo toma 1 unidad de tiempo
- Objetivo: Maximizar ganancia total

## Aplicaciones

- Scheduling de procesos en CPU
- Planificación de producción
- Asignación de recursos limitados



## Estrategia

- Generar todas las permutaciones de trabajos
- Para cada permutación, programar trabajos en orden
- Solo incluir trabajos que cumplan deadline
- Seleccionar permutación con máxima ganancia

## Complejidad

$O(n! \cdot n)$  - Factorial

# Job Scheduling - Pseudocódigo Fuerza Bruta

```
1: [Entrada:] Una lista jobs de n elementos donde cada elemento es (id, deadline, profit)
2: [Salida:] La mejor programación y la ganancia máxima
3: max_profit  $\leftarrow$  0
4: best_schedule  $\leftarrow$  lista vacía
5: for cada perm en permutations(jobs) do
6:   time  $\leftarrow$  0
7:   profit  $\leftarrow$  0
8:   schedule  $\leftarrow$  lista vacía
9:   for cada job en perm do
10:    if time < job.deadline then
11:      schedule.append(job)
12:      profit  $\leftarrow$  profit + job.profit
13:      time  $\leftarrow$  time + 1
14:    end if
15:  end for
16:  if profit > max_profit then
17:    max_profit  $\leftarrow$  profit
18:    best_schedule  $\leftarrow$  schedule
19:  end if
20: end for return best_schedule, max_profit
```

▷ Probar todas las permutaciones posibles

▷ Tiempo actual

▷ Ganancia acumulada

▷ Programación actual

▷ Verifica deadline

# Job Scheduling - Ejemplo Fuerza Bruta

Trabajos: [(a,2,100), (b,1,19), (c,2,27), (d,1,25), (e,3,15)]

- **Permutación 1:** [a,b,c,d,e] → Ganancia: 142
- **Permutación 2:** [b,a,c,d,e] → Ganancia: 134
- **Permutación 3:** [d,a,c,b,e] → Ganancia: 140
- **Mejor:** [a,c,e] con ganancia 142

## Limitaciones

5 trabajos = 120 permutaciones, 10 trabajos = 3.6 millones

# Job Scheduling - Enfoque Greedy

## Estrategia

- Ordenar trabajos por profit descendente
- Para cada trabajo, asignar al slot más tardío disponible dentro de su deadline

## Complejidad

$O(n \log n + n \cdot d)$  - Polinomial

# Job Scheduling - Pseudocódigo Greedy

```
1: [Entrada:] Lista jobs donde cada trabajo tiene (id, deadline, profit)
2: [Salida:] Lista de trabajos seleccionados y ganancia total
3: Ordenar jobs por ganancia en orden descendente
4:  $max\_deadline \leftarrow \max\{job.deadline \mid \forall job \in jobs\}$ 
5: slots  $\leftarrow$  arreglo de tamaño  $max\_deadline + 1$  inicializado en  $-1$ 
6: total_profit  $\leftarrow 0$ , scheduled_jobs  $\leftarrow$  lista vacía
7: for cada job en jobs do
8:   for  $t \leftarrow job.deadline$ ;  $t \geq 1$ ;  $t \leftarrow t - 1$  do
9:     if slots[t] =  $-1$  then
10:       slots[t]  $\leftarrow job.id$ 
11:       total_profit  $\leftarrow total\_profit + job.profit$ 
12:       scheduled_jobs.append(job.id)
13:       break
14:     end if
15:   end for
16: end for return scheduled_jobs, total_profit
```

▷ Prioriza ganancias altas

▷ Slots de tiempo

▷ Buscar slot más tardío disponible

# Job Scheduling - Ejemplo Greedy I

Trabajos = [A(2,100), B(1,19), C(2,27), D(1,25), E(3,15)]

Orden por ganancia: A(100), C(27), D(25), B(19), E(15)

## Asignación

- A (d=2): buscar  $t=2..1 \rightarrow t=2$  libre  $\rightarrow$  slots[2]=A, total=100
- C (d=2):  $t=2$  ocupado,  $t=1$  libre  $\rightarrow$  slots[1]=C, total=127
- D (d=1):  $t=1$  ocupado  $\rightarrow$  no asignar
- B (d=1):  $t=1$  ocupado  $\rightarrow$  no asignar
- E (d=3):  $t=3$  libre  $\rightarrow$  slots[3]=E, total=142

## Resultado Final

Programación: [C, A, E] con ganancia total 142

Enfoque	Complejidad	Optimalidad
Fuerza Bruta	$O(n! \cdot n)$	✓
Greedy	$O(n \log n + n \cdot d)$	×

## Conclusión

- Greedy no siempre es óptimo pero es muy eficiente
- En la práctica funciona bien para la mayoría de casos
- Fuerza bruta solo para instancias muy pequeñas ( $n < 10$ )

# Tabla de Contenidos

- 1 Problema 1: Activity Selection Problem
- 2 Problema 2: Graph Coloring
- 3 Problema 3: Job Scheduling with Deadlines
- 4 Problema 4: Huffman Codes**
  - Enunciado
  - Enfoque de Fuerza Bruta
  - Enfoque Greedy
  - Análisis comparativo de complejidad
- 5 Conclusiones y Recomendaciones



## Descripción del Problema

Encontrar código de prefijo óptimo para compresión de datos:

- Símbolos con frecuencias dadas
- Código de longitud variable
- Objetivo: Minimizar longitud promedio
- Costo =  $\sum f_i \cdot l_i$

## Aplicaciones

- Compresión de archivos (ZIP, JPEG, MP3)
- Transmisión eficiente de datos

## Estrategia

- Generar todos los árboles binarios completos con  $n$  hojas
- Para cada árbol, calcular longitudes de código
- Probar todas las asignaciones de símbolos a hojas
- Seleccionar asignación con mínimo costo

## Complejidad

$O(C_{n-1} \cdot n! \cdot n)$  - Super-exponencial

# Huffman Codes - Pseudocódigo Fuerza Bruta

```
1: [Entrada:] Símbolos y frecuencias
2: [Salida:] Asignación óptima de códigos
3:  $n \leftarrow$  número de símbolos
4:  $best\_cost \leftarrow \infty$ ,  $best\_assignment \leftarrow \text{null}$ 
5: for cada  $tree$  en  $generate\_trees(n)$  do
6:    $lengths \leftarrow code\_lengths(tree)$ 
7:   for cada  $perm$  en  $permutations(range(n))$  do
8:      $cost \leftarrow \sum freqs[perm[i]] \cdot lengths[i]$ 
9:     if  $cost < best\_cost$  then
10:        $best\_cost \leftarrow cost$ ,  $best\_assignment \leftarrow$  asignación
11:     end if
12:   end for
13: end for return  $best\_cost, best\_assignment$ 
```

# Huffman Codes - Ejemplo Fuerza Bruta

4 símbolos: [A:5, B:2, C:1, D:1]

- Número de árboles:  $C_3 = 5$
- Permutaciones por árbol:  $4! = 24$
- Total:  $5 \times 24 = 120$  asignaciones
- **Mejor:** A→1, B→2, C→2, D→3 (Costo=14)

## Limitaciones

6 símbolos:  $C_5 = 42$  árboles  $\times 6! = 720 \times 6 = 181,440$  operaciones

## Estrategia Óptima

- Usar min-heap por frecuencias
- Combinar siempre los dos nodos de menor frecuencia
- Construir árbol de abajo hacia arriba

## Complejidad

$O(n \log n)$  - Polinomial (óptimo)

# Huffman Codes - Pseudocódigo Greedy

```
1: [Entrada:] Lista de símbolos  $s$  y frecuencias  $freq$ 
2: [Salida:] Códigos Huffman óptimos
3:  $n \leftarrow$  longitud de  $s$ 
4:  $pq \leftarrow$  cola de prioridad (min-heap) vacía      ▷ Menor frecuencia primero      ▷ Crear nodos hoja para cada símbolo
5: for  $i = 0$  to  $n - 1$  do
6:    $tmp \leftarrow Node(freq[i])$ 
7:    $pq.push(tmp)$ 
8: end for                                          ▷ Construir árbol de Huffman
9: while  $|pq| \geq 2$  do
10:   $l \leftarrow pq.pop()$ 
11:   $r \leftarrow pq.pop()$ 
12:   $new \leftarrow Node(l.freq + r.freq)$ 
13:   $new.left \leftarrow l, new.right \leftarrow r$ 
14:   $pq.push(new)$ 
15: end while
16:  $root \leftarrow pq.pop()$ 
17: Generar códigos con recorrido preorden desde  $root$  return lista de códigos por símbolo
```

# Huffman Codes - Ejemplo Greedy

Símbolos: [a:5, b:9, c:12, d:13, e:16, f:45]

- Combinar  $a(5) + b(9) = 14$
- Combinar  $c(12) + d(13) = 25$
- Combinar  $14 + e(16) = 30$
- Combinar  $25 + 30 = 55$
- Combinar  $f(45) + 55 = 100$

## Resultado

Códigos óptimos:  $f \rightarrow 0$ ,  $c \rightarrow 100$ ,  $d \rightarrow 101$ ,  $a \rightarrow 1100$ ,  $b \rightarrow 1101$ ,  $e \rightarrow 111$

Enfoque	Complejidad	Optimalidad
Fuerza Bruta	$O(C_{n-1} \cdot n! \cdot n)$	✓
Greedy	$O(n \log n)$	✓

## Conclusión

- Huffman greedy es óptimo y eficiente
- Raro caso donde greedy garantiza optimalidad
- Fuerza bruta completamente impráctica
- Algoritmo ampliamente usado en compresión



# Tabla de Contenidos

- 1 Problema 1: Activity Selection Problem
- 2 Problema 2: Graph Coloring
- 3 Problema 3: Job Scheduling with Deadlines
- 4 Problema 4: Huffman Codes
- 5 Conclusiones y Recomendaciones

# Conclusiones Generales

## Comparativa de Enfoques

Problema	BF Optimal	Greedy Optimal	BF Complejidad	Greedy Complejidad
Activity Selection	✓	✓	$O(2^n n^2)$	$O(n \log n)$
Graph Coloring	✓	×	$O(k^n nm)$	$O(V^2 + E)$
Job Scheduling	✓	×	$O(n!n)$	$O(n \log n + nd)$
Huffman Codes	✓	✓	$O(C_{n-1} n!n)$	$O(n \log n)$

## Insights Clave

- Greedy es óptimo cuando el problema exhibe **subestructura óptima**
- La **elección greedy** debe preservar la posibilidad de solución óptima
- Fuerza bruta es invaluable para verificar correctitud de algoritmos

## Cuándo usar cada enfoque

- **Fuerza Bruta:**

- Problemas pequeños ( $n \leq 20$ )
- Verificación de algoritmos
- Casos donde se necesita optimalidad garantizada

- **Greedy:**

- Problemas grandes donde optimalidad no es crítica
- Cuando el problema exhibe subestructura óptima
- Aplicaciones en tiempo real que requieren rapidez

## Estrategia Híbrida

En la práctica: Empezar con greedy, usar fuerza bruta para casos pequeños o como verificación

# ¿Preguntas?

### Contacto

- Delgado Romero
- Torres Reategui
- Quintanilla Quispe

¡Gracias por su atención!