

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

Entrada: Lista de actividades (*start*, *end*)

Salida: Subconjunto de máximo tamaño de actividades compatibles

```
1:  $n \leftarrow$  número de actividades
2:  $best \leftarrow$  lista vacía
3: for  $r = 1$  to  $n$  do
4:   for cada subset en combinations(activities,  $r$ ) do
5:     if  $is\_valid(subset) \wedge |subset| > |best|$  then
6:        $best \leftarrow subset$ 
7:     end if
8:   end for
9: end for return  $best$ 
```

Activity Selection - Ejemplo Fuerza Bruta

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

- **Subconjuntos de tamaño 1:** Todos válidos
- **Subconjuntos de tamaño 2:**
 - $\{A,B\}$: Inválido (se solapan)
 - $\{A,C\}$: Válido
 - $\{B,C\}$: Inválido (se solapan)
- **Subconjunto de tamaño 3:** 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

Entrada: Arreglos *start* y *finish* de n actividades

Salida: Número máximo de actividades compatibles

```
1: Ordenar actividades por finish
2:  $count \leftarrow 1, j \leftarrow 0$ 
3: for  $i = 1$  to  $n - 1$  do
4:   if  $start[i] > finish[j]$  then
5:      $count \leftarrow count + 1$ 
6:      $j \leftarrow i$ 
7:   end if
8: end for return  $count$ 
```

Activity Selection - Ejemplo Greedy

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

- Seleccionar $(1,2)$
- Seleccionar $(3,4)$ - No se solapa
- Omitir $(0,6)$ - Se solapa
- Seleccionar $(5,7)$ - No se solapa
- Seleccionar $(8,9)$ - No se solapa
- Omitir $(5,9)$ - Se solapa

Resultado

4 actividades seleccionadas: $[(1,2), (3,4), (5,7), (8,9)]$

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

Entrada: Grafo G , número máximo de colores k

Salida: Asignación válida de colores o null

```
1: for cada assignment en product(range(k), repeat = n) do
2:   if is_valid( $G$ , assignment) then return assignment
3:   end if
4: end for return null

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
```


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

Entrada: Grafo G con V vértices

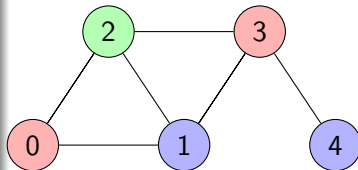
Salida: Asignación válida de colores

- 1: $result \leftarrow$ arreglo de tamaño V inicializado en -1
- 2: $result[0] \leftarrow 0$
- 3: **for** $u = 1$ to $V - 1$ **do**
- 4: Marcar colores de vecinos como no disponibles
- 5: Encontrar primer color disponible cr
- 6: $result[u] \leftarrow cr$
- 7: Resetear marcas de disponibilidad
- 8: **end for** **return** $result$

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

Entrada: Lista de trabajos (*id*, *deadline*, *profit*)

Salida: Programación con máxima ganancia

```
1:  $max\_profit \leftarrow 0$ ,  $best\_schedule \leftarrow []$ 
2: for cada perm en permutations(jobs) do
3:    $time \leftarrow 0$ ,  $profit \leftarrow 0$ ,  $schedule \leftarrow []$ 
4:   for cada job en perm do
5:     if  $time < job.deadline$  then
6:        $schedule.append(job)$ ,  $profit \leftarrow profit + job.profit$ 
7:        $time \leftarrow time + 1$ 
8:     end if
9:   end for
10:  if  $profit > max\_profit$  then
11:     $max\_profit \leftarrow profit$ ,  $best\_schedule \leftarrow schedule$ 
12:  end if
13: end for return  $best\_schedule$ ,  $max\_profit$ 
```

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

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

Entrada: Lista de trabajos (*id*, *deadline*, *profit*)

Salida: Programación y ganancia total

```
1: Ordenar trabajos por profit descendente
2:  $max\_d \leftarrow \text{máx}(\text{deadlines})$ 
3:  $slots \leftarrow$  arreglo de tamaño  $max\_d + 1$  inicializado en -1
4:  $total\_profit \leftarrow 0$ 
5: for cada job en trabajos ordenados do
6:     for  $t = job.deadline$  to 1 do
7:         if  $slots[t] = -1$  then
8:              $slots[t] \leftarrow job.id$ ,  $total\_profit \leftarrow total\_profit + job.profit$ 
9:             break
10:        end if
11:    end for
12: end for return trabajos en slots,  $total\_profit$ 
```

Job Scheduling - Ejemplo Greedy

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

- **a**(deadline=2): Slot 2 \rightarrow Profit=100
- **c**(deadline=2): Slot 1 \rightarrow Profit=127
- **d**(deadline=1): No hay slot disponible
- **b**(deadline=1): No hay slot disponible
- **e**(deadline=3): Slot 3 \rightarrow Profit=142

Resultado

Programación: [c, a, e] con ganancia 142 (óptima)

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

Entrada: Símbolos y frecuencias

Salida: Asignación óptima de códigos

```
1:  $n \leftarrow$  número de símbolos
2:  $best\_cost \leftarrow \infty$ ,  $best\_assignment \leftarrow$  null
3: for cada  $tree$  en  $generate\_trees(n)$  do
4:    $lengths \leftarrow code\_lengths(tree)$ 
5:   for cada  $perm$  en  $permutations(range(n))$  do
6:      $cost \leftarrow \sum freqs[perm[i]] \cdot lengths[i]$ 
7:     if  $cost < best\_cost$  then
8:        $best\_cost \leftarrow cost$ ,  $best\_assignment \leftarrow$  asignación
9:     end if
10:  end for
11: 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

Entrada: Símbolos y frecuencias

Salida: Códigos Huffman óptimos

```
1:  $pq \leftarrow$  min-heap vacío
2: for cada símbolo do
3:    $pq.push(Node(freq))$ 
4: end for
5: while  $|pq| > 1$  do
6:    $l \leftarrow pq.pop(), r \leftarrow pq.pop()$ 
7:    $new \leftarrow Node(l.freq + r.freq)$ 
8:    $new.left \leftarrow l, new.right \leftarrow r$ 
9:    $pq.push(new)$ 
10: end while
11:  $root \leftarrow pq.pop()$ 
12: Generar códigos con recorrido preorden return códigos
```

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!