Exposición: Algoritmos de Fuerza Bruta vs Greedy

Análisis Comparativo de Estrategias Algorítmicas

Delgado Romero, Gustavo Torres Reategui, Joaquin

Universidad Nacional de Ingeniería
CC371 - Análisis y Diseño de Algoritmos

24 de septiembre de 2025

Tabla de Contenidos

Resumen de Problemas

Problemas a Comparar

1. **Activity Selection Problem** - Selección de actividades compatibles
2. **Graph Coloring** - Coloreo de grafos con restricciones
3. **Job Scheduling** - Programación de trabajos con deadlines
4. **Huffman Codes** - Codificación óptima de símbolos

Compararemos las implementaciones de fuerza bruta y greedy para cada problema.

Tabla de Contenidos

Enunciado — Activity Selection

Enunciado

El problema de selección de actividades consiste en seleccionar el máximo número de actividades mutuamente compatibles de un conjunto dado. Cada actividad tiene un tiempo de inicio y un tiempo de finalización. Dos actividades son compatibles si no se solapan en el tiempo.

Objetivo

Encontrar el subconjunto de máximo tamaño que no contenga actividades que se solapen.

Pseudocódigo — Fuerza Bruta

Algoritmo Selección de Actividades con Fuerza Bruta

Entrada: Lista de actividades con tiempos de inicio y finalización

Salida: Subconjunto de máximo tamaño de actividades compatibles

```
1:  n ← longitud de actividades
2:  best ← lista vacía
3:  for r = 1 to n do
4:      for cada subconjunto de tamaño r en combinaciones de actividades do
5:          if subconjunto es válido y |subconjunto| > |best| then
6:              best ← subconjunto
7:          end if
8:      end for
9:  end for
10: return best
```

Pseudocódigo — Validación

Algoritmo Función de Validación

Entrada: Subconjunto de actividades

Salida: Verdadero si no hay solapamientos

```
1: for cada par de actividades (i, j) en el subconjunto do
2:     if actividades i y j se solapan then
3:         return false
4:     end if
5: end for
6: return true
```

Pseudocódigo — Greedy

Algoritmo Selección de Actividades con Greedy

Entrada: Lista de actividades con tiempos de inicio y finalización

Salida: Número máximo de actividades compatibles

```
1: Ordenar actividades por tiempo de finalización
2: count ← 1
3: j ← 0                                   ▷ Última actividad seleccionada
4: for i = 1 to n − 1 do
5:     if actividad i no se solapa con actividad j then
6:         count ← count + 1
7:         j ← i
8:     end if
9: end for
10: return count
```

Código — Fuerza Bruta

Código — Greedy

Comparación — Activity Selection

Aspecto	Fuerza Bruta	Greedy
Complejidad Temporal	$O(2^n \cdot n^2)$	$O(n \log n)$
Complejidad Espacial	$O(n)$	$O(n)$
Optimalidad	Garantizada	Garantizada
Implementación	Compleja	Simple
Escalabilidad	Limitada	Excelente
Ventajas	Solución óptima garantizada	Eficiente y óptimo
Desventajas	Exponencial, lento	Depende del orden

Tabla de Contenidos

Enunciado — Graph Coloring

Enunciado

El problema de coloreo de grafos consiste en asignar colores a los vértices de un grafo de manera que ningún par de vértices adyacentes compartan el mismo color. El objetivo es encontrar una coloración válida usando el menor número posible de colores.

Pseudocódigo — Fuerza Bruta

Algoritmo Coloreo de Grafos con Fuerza Bruta

Entrada: Grafo G y número de colores k

Salida: Asignación válida de colores o null

```
1: n ← número de vértices
2: for cada asignación en producto cartesiano de k colores do
3:     if asignación es válida then
4:         return asignación
5:     end if
6: end for
7: return null
```

Pseudocódigo — Greedy

Algoritmo Coloreo de Grafos con Greedy

Entrada: Grafo G con V vértices

Salida: Asignación de colores para todos los vértices

```
1:  result[0] ← 0                                   ▷ Primer color al primer vértice
2:  available ← arreglo de V elementos en false
3:  for u = 1  to V − 1 do
4:      for cada vértice adyacente i a u do
5:          if result[i] ≠ −1 then
6:              available[result[i]] ← true
7:          end if
8:      end for
9:      cr ← primer color disponible
10:     result[u] ← cr
11:     Resetear available para siguiente iteración
12: end for
```

Código — Fuerza Bruta

Código — Greedy

Comparación — Graph Coloring

Aspecto	Fuerza Bruta	Greedy
Complejidad Temporal	$O(k^n \cdot n \cdot m)$	$O(V^2 + E)$
Complejidad Espacial	$O(n)$	$O(n)$
Optimalidad	Garantizada	No garantizada
Implementación	Compleja	Simple
Escalabilidad	Muy limitada	Buena
Ventajas	Solución óptima	Eficiente, práctico
Desventajas	Exponencial, lento	Puede usar más colores

Tabla de Contenidos

Enunciado — Job Scheduling

Enunciado

El problema de programación de trabajos con deadline consiste en seleccionar y programar un subconjunto de trabajos de manera que se maximice la ganancia total, respetando las restricciones de deadline. Cada trabajo tiene un deadline y una ganancia asociada.

Pseudocódigo — Fuerza Bruta

Algoritmo Programación de Trabajos con Fuerza Bruta

Entrada: Lista de trabajos con deadline y ganancia

Salida: Mejor programación y ganancia máxima

```
1:  best_profit ← 0
2:  best_schedule ← lista vacía
3:  for cada permutación de trabajos do
4:      time ← 0, profit ← 0
5:      for cada trabajo en la permutación do
6:          if time < trabajo.deadline then
7:              Incluir trabajo, actualizar ganancia y tiempo
8:          end if
9:      end for
10:     if profit > best_profit then
11:         Actualizar mejor solución
12:     end if
```

Pseudocódigo — Greedy

Algoritmo Programación de Trabajos con Greedy

Entrada: Lista de trabajos con deadline y ganancia

Salida: Trabajos seleccionados y ganancia total

```
1:  Ordenar trabajos por ganancia descendente
2:  max_deadline ← deadline máximo
3:  slots ← arreglo de slots inicializado en -1
4:  for cada trabajo en orden de ganancia do
5:      for t = trabajo.deadline 1 do
6:          if slots[t] = −1 then
7:              Asignar trabajo al slot t
8:              Actualizar ganancia total
9:              break
10:         end if
11:     end for
12: end for
```

Código — Fuerza Bruta

Código — Greedy

Comparación — Job Scheduling

Aspecto	Fuerza Bruta	Greedy
Complejidad Temporal	$O(n! \cdot n)$	$O(n \log n + n \cdot d)$
Complejidad Espacial	$O(n)$	$O(d)$
Optimalidad	Garantizada	No garantizada
Implementación	Compleja	Simple
Escalabilidad	Limitada	Buena
Ventajas	Solución óptima	Eficiente, práctico
Desventajas	Factorial, lento	Puede ser subóptimo

Tabla de Contenidos

Enunciado — Huffman Codes

Enunciado

El problema de codificación de Huffman consiste en encontrar el código de longitud variable óptimo para un conjunto de símbolos dados sus frecuencias de aparición. El objetivo es minimizar la longitud promedio de codificación.

Pseudocódigo — Fuerza Bruta

Algoritmo Códigos de Huffman con Fuerza Bruta

Entrada: Símbolos y frecuencias

Salida: Costo mínimo y asignación óptima

```
1:  best_cost ← ∞
2:  for cada árbol binario completo con n hojas do
3:      lengths ← longitudes de código del árbol
4:      for cada permutación de asignación de símbolos do
5:          cost ← calcular costo total
6:          if cost < best_cost then
7:              Actualizar mejor solución
8:          end if
9:      end for
10: end for
11: return best_cost, best_assignment
```

Pseudocódigo — Greedy

Algoritmo Códigos de Huffman con Greedy

Entrada: Símbolos y frecuencias

Salida: Códigos Huffman para cada símbolo

1: Crear min-heap con nodos hoja para cada símbolo
2: **while** heap tiene más de un elemento **do**
3: $l \leftarrow$ extraer mínimo
4: $r \leftarrow$ extraer mínimo
5: Crear nodo interno con frecuencia $l.freq + r.freq$
6: Insertar nodo interno en heap
7: **end while**
8: Generar códigos mediante recorrido preorden
9: **return** códigos generados

Código — Fuerza Bruta

Código — Greedy

Comparación — Huffman Codes

Aspecto	Fuerza Bruta	Greedy
Complejidad Temporal	$O(C_{n-1} \cdot n! \cdot n)$	$O(n \log n)$
Complejidad Espacial	$O(n)$	$O(n)$
Optimalidad	Garantizada	Garantizada
Implementación	Muy compleja	Simple
Escalabilidad	Muy limitada	Excelente
Ventajas	Solución óptima	Eficiente y óptimo
Desventajas	Exponencial, impracticable	Ninguna significativa

Conclusiones Generales

Análisis Comparativo

- **Fuerza Bruta**: Garantiza solución óptima pero con complejidad exponencial
- **Greedy**: Eficiente y práctico, pero no siempre óptimo
- **Trade-offs**: Optimalidad vs. eficiencia computacional
- **Aplicabilidad**: Depende del tamaño del problema y requisitos

Recomendaciones

Cuándo usar cada enfoque

- **Fuerza Bruta**: Problemas pequeños, verificación de algoritmos, casos donde la optimalidad es crítica
- **Greedy**: Problemas grandes, aplicaciones en tiempo real, cuando la eficiencia es prioritaria
- **Híbrido**: Combinar ambos enfoques según el contexto del problema

Consideraciones

Siempre evaluar el tamaño del problema, requisitos de optimalidad y restricciones de tiempo antes de elegir el enfoque.