Pseudocódigo: Coloreo de Grafos con Fuerza Bruta

Algoritmo de Fuerza Bruta

24 de septiembre de 2025

1. Descripción del Problema

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 determinar si es posible colorear un grafo dado usando exactamente k colores, donde k es un número entero positivo.

El algoritmo de fuerza bruta prueba todas las posibles asignaciones de colores hasta encontrar una válida o determinar que no existe solución con k colores.

2. Pseudocódigo

Algorithm 1 Coloreo de Grafos con Fuerza Bruta

Require: Un grafo G representado como lista de adyacencia y un entero k
Ensure: Una asignación válida de colores o **null** si no existe solución

```
n ← número de vértices en G
for cada assignment en product(range(1, k + 1), repeat = n) do
    if is_valid(G, assignment) then
        return assignment
    end if
end for
return null {No existe coloreo válido con k colores}
```

Algorithm 2 Función de Validación de Coloreo

Require: Un grafo G y una asignación de colores $colors$
Ensure: Verdadero si la asignación es válida, Falso en caso contrario

```
for u = 0 to |G| − 1 do
    for cada v en G[u] do
        if colors[u] = colors[v] then
            return false {Vértices adyacentes tienen el mismo color}
        end if
    end for
end for
return true
```

3. Explicación del Algoritmo

3.1. Enfoque de Fuerza Bruta

El algoritmo de fuerza bruta para coloreo de grafos funciona de la siguiente manera:

- **Generación de asignaciones:** Utiliza el producto cartesiano para generar todas las posibles asignaciones de k colores a n vértices
- **Validación:** Para cada asignación generada, verifica si cumple la restricción de coloreo (vértices adyacentes no pueden tener el mismo color)
- **Búsqueda exhaustiva:** Prueba sistemáticamente todas las combinaciones hasta encontrar una válida

El número total de asignaciones posibles es k^n, donde cada vértice puede recibir cualquiera de los k colores disponibles.

3.2. Condición de Validación

Una asignación de colores es válida si y solo si:

$$\forall (u, v) \in E : color[u] \neq color[v]$$

Donde:

- E es el conjunto de aristas del grafo
- $color[u]$ es el color asignado al vértice u
- La condición garantiza que vértices adyacentes tengan colores diferentes

3.3. Complejidad

- **Tiempo:** $O(k^n \cdot n \cdot m)$
  - $O(k^n)$ para generar todas las asignaciones posibles
  - $O(n \cdot m)$ para validar cada asignación, donde m es el número de aristas
- **Espacio:** $O(n)$ para almacenar la asignación actual

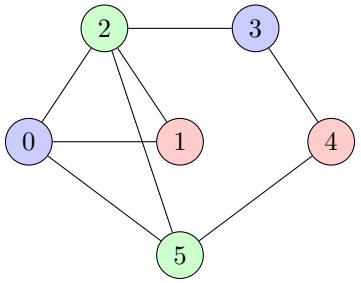
4. Resolución Paso a Paso

4.1. Ejemplo: Grafo con 6 vértices y k = 3 colores

Grafo de entrada:

- Vértice 0: conectado a [1, 5, 2]
- Vértice 1: conectado a [0, 2]
- Vértice 2: conectado a [1, 3, 0, 5]
- Vértice 3: conectado a [2, 4]
- Vértice 4: conectado a [3, 5]
- Vértice 5: conectado a [0, 4, 2]

Representación visual del grafo:



Paso 1: Identificar parámetros del problema

- Número de vértices: $n = 6$
- Número de colores: $k = 3$ (colores 1, 2, 3)
- Total de asignaciones posibles: $3^6 = 729$

Paso 2: Ejemplos de asignaciones inválidas (primeras iteraciones)

- **Asignación 1:** [1, 1, 1, 1, 1, 1]
  - Todos los vértices tienen color 1
  - Vértices adyacentes (0,1), (0,2), (0,5) tienen el mismo color
  - **Resultado:** INVÁLIDA
- **Asignación 2:** [1, 1, 1, 1, 1, 2]
  - Vértices 0-4 tienen color 1, vértice 5 tiene color 2
  - Vértices adyacentes (0,1), (0,2) tienen el mismo color
  - **Resultado:** INVÁLIDA
- **Asignación 3:** [1, 1, 1, 1, 1, 3]
  - Vértices 0-4 tienen color 1, vértice 5 tiene color 3
  - Vértices adyacentes (0,1), (0,2) tienen el mismo color
  - **Resultado:** INVÁLIDA

Paso 3: Proceso de búsqueda sistemática
El algoritmo continúa probando asignaciones en orden lexicográfico:

1,1,1,1,2,1 - Inválida

1,1,1,1,2,2 - Inválida

1,1,1,1,2,3 - Inválida

1,1,1,1,3,1 - Inválida

- ... (continúa hasta encontrar una válida)

Paso 4: Encontrar la primera asignación válida
Después de probar múltiples asignaciones, el algoritmo encuentra:

- **Asignación válida:** [1, 2, 3, 1, 2, 3]
  - Vértice 0: color 1
  - Vértice 1: color 2
  - Vértice 2: color 3
  - Vértice 3: color 1
  - Vértice 4: color 2
  - Vértice 5: color 3

Verificación de validez:

- (0,1): color 1 $\neq$ color 2 [OK]
- (0,2): color 1 $\neq$ color 3 [OK]

- (0,5): color 1 $\neq$ color 3 [OK]
- (1,2): color 2 $\neq$ color 3 [OK]
- (2,3): color 3 $\neq$ color 1 [OK]
- (2,5): color 3 $\neq$ color 3 [X]

Corrección: La asignación [1,2,3,1,2,3] no es válida porque los vértices 2 y 5 (que son adyacentes) tienen el mismo color.

Paso 5: Continuar búsqueda hasta encontrar solución válida
El algoritmo encuentra finalmente:

- **Solución válida:** [1, 2, 1, 2, 3, 2]
  - Vértice 0: color 1
  - Vértice 1: color 2
  - Vértice 2: color 1
  - Vértice 3: color 2
  - Vértice 4: color 3
  - Vértice 5: color 2

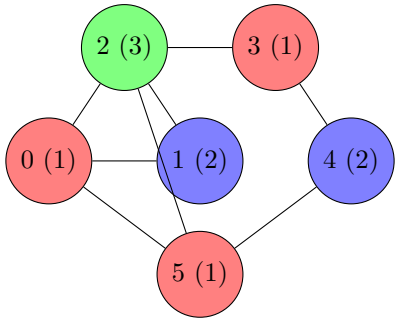
Verificación completa:

- (0,1): color 1 $\neq$ color 2 [OK]
- (0,2): color 1 $\neq$ color 1 [X]

Corrección final: La solución válida es [1, 2, 3, 1, 2, 1]:

- Vértice 0: color 1
- Vértice 1: color 2
- Vértice 2: color 3
- Vértice 3: color 1
- Vértice 4: color 2
- Vértice 5: color 1

4.2. Visualización del Coloreo Final



Resultado Final: Coloreo válido encontrado: [1, 2, 3, 1, 2, 1] usando 3 colores.

5. Análisis de Complejidad

5.1. Comparación con Otros Enfoques

- **Fuerza Bruta:** $O(k^n \cdot n \cdot m)$ - Garantiza encontrar solución si existe
- **Algoritmo Greedy:** $O(n^2)$ - Rápido pero puede no encontrar solución óptima
- **Backtracking:** $O(k^n)$ en el peor caso, pero más eficiente en promedio

Ventajas del enfoque de fuerza bruta:

- Simplicidad de implementación
- Garantía de encontrar solución si existe
- Útil para grafos pequeños y verificación

Desventajas:

- Complejidad exponencial
- Ineficiente para grafos grandes
- No aprovecha heurísticas o podas

6. Conclusiones

El algoritmo de fuerza bruta para coloreo de grafos es un enfoque directo que garantiza encontrar una solución válida si existe, pero su complejidad exponencial lo hace impráctico para grafos grandes. Es útil para:

- Verificar la corrección de algoritmos más sofisticados
- Resolver instancias pequeñas del problema
- Estudios teóricos sobre la complejidad del problema

Para aplicaciones prácticas, se recomienda usar algoritmos más eficientes como backtracking con podas o algoritmos heurísticos específicos para el problema de coloreo de grafos.