Problemas de Diseño y Análisis de Algoritmos

Marco Antonio Ochil Trujillo Kevin Majim Ortega Alvarez
Yoan René Ramos Corrales

September 22, 2024

Contents

1 Transformación de Redes de Esqueletos, el problema

Marco el nigromante se dedicaba a unir huesos de criaturas muertas para armar esqueletos bizarros. Cada hueso se une a al menos otro por conexiones mágicas y está nombrado con un número entero.

De pronto, moviendo huesos por aquí y por allá, creó una criatura tan horrorosa que parecía salida de la mismísima Australia, pero muerta. Marco sintió amor a primera vista, completa fascinación.

Con una criatura tan horrenda, el resto de sus creaciones palidecían en comparación, pero desmantelarlas representaba un desperdicio de componentes mágicos. De pronto se le ocurrió una idea. Quizás podía convertir alguna de sus otras criaturas en una copia (o al menos algo medio parecido) de esta bizarra genialidad sólo renombrando los huesos que las conformaban. Para él, una criatura es parecida a otra cuando todos sus huesos dos a dos están conectados a exactamente la misma cantidad de huesos y estos huesos conectados tienen los mismos nombres dos a dos.

Ayude a Marco a encontrar un método con el que saber si un esqueleto arbitrario puede convertirse en una copia parecida a su criatura favorita sólo cambiándole los nombres a sus huesos.

2 Transformación del problema

Luego de varias lecturas al problema, decidimos que estábamos ante un problema de grafos donde estos, eran las criaturas. Antes de afirmar la suposición anterior debíamos encontrar una forma de construir los grafos a partir de las criaturas:
Construcción: Cada hueso de una criatura puede ser representado como los vértices de un grafo y las conexiones mágicas entre ellos como las aristas. Por lo tanto, cada criatura puede ser representada como un grafo.

Una vez construidos los grafos a partir de cada criatura, solo resta renombrar cada vértice de estos para saber si dos criaturas son parecidas. Luego este problema de encontrar criaturas parecidas, es equivalente a encontrar una función biyectiva que mapee los nodos de un grafo a otro, o lo que es lo mismo, verificar que dos grafos que sean isomorfos.

2.1 Definición de grafos isomorfos

Dos grafos son isomorfos si podemos encontrar una correspondencia uno a uno entre sus vértices de tal manera que las aristas también coincidan.

Sean $G = (V_G, E_G)$ y $H = (V_H, E_H)$ dos grafos donde:

- V_G y V_H son los nodos de cada grafo respectivamente
- E_G y E_H son las aristas de cada grafo respectivamente

Un isomorfismo de grafos es una función $f : V_G \to V_H$ que establece una biyección entre los vértices de G y H tal que:

Para cada par de vértices $u, v \in V_G$, $(u, v) \in E_G$ si y solo si $(f(u), f(v)) \in E_H$.

3 Primeros acercamientos a la solución

Tipo de Problema

El problema de isomorfismo de grafos es un problema NP.

Demostración: Para que un problema sea NP, necesitamos poder verificar si una posible solución es válida en tiempo polinomial.
Supongamos que tenemos f una función que establece una biyección entre los vértices de G y H. f debe asignar a cada vértice de G un vértice en H (para que sea inyectiva) y todos los vértices de H deben ser cubiertos por f (para que sea sobreyectiva). Verificar entonces que f es biyectiva se puede lograr en $O(n)$ (siendo n la cantidad de nodos en G y en H).
Luego de que tenemos f biyectiva, recorremos todas las aristas $(u, v) \in G$ y verificamos que existe $(f(u), f(v)) \in H$. Por lo tanto si G tiene n nodos y m aristas, y la revisión de cada arista toma tiempo constante dado que es aplicar una función de mapeo a los nodos que forman esa arista, la complejidad total de la verificación sería $O(m)$, lo cual es polinomial en función de las aristas, en el caso peor que el grafo sea completo, sería $O(n^2)$

3.1 Resolviendo el problema con Fuerza Bruta

Nuestro primer acercamiento a la solución del problema fue hacer un algoritmo de fuerza bruta, con las siguientes especificaciones:

- **Entradas:** Dos grafos G_1 y G_2 representados por matrices de adyacencia de $n \times n$, donde n es la cantidad de nodos de cada grafo
- **Salida:** Un valor booleano, donde true significa que ambos grafos son isomorfos, y false que no

Luego sigue los siguientes pasos:

1. **Verificación del tamaño:** Primero, se compara el número de nodos en los dos grafos. Si tienen un número diferente de nodos, no pueden ser isomorfos
2. **Generación de permutaciones:** Si ambos grafos tienen el mismo número de nodos, el algoritmo genera todas las posibles permutaciones de los nodos del primer grafo. Cada permutación es una posible reorganización de los nodos.
3. **Comprobación de isomorfismo:** Para cada permutación de los nodos del primer grafo, el algoritmo revisa si reorganizando los vértices de acuerdo con esa permutación se obtiene el segundo grafo.

3.2 Complejidad temporal

Dado que el algoritmo realiza una búsqueda en el espacio de las permutaciones de nodos de un grafo, la búsqueda se hace en $O(n!)$, y por cada una se hace una comparación entre las matrices de adyacencia en $O(n^2)$, por lo que, la complejidad del algoritmo para el caso peor es de $O(n! \cdot n^2)$

La complejidad espacial es de $O(n^2)$, debido al trabajo con las matrices de adyacencia

4 Soluciones finales

Luego de una pequeña revisión en la literatura, encontramos algunas invariantes que podíamos usar para reducir el espacio de búsqueda, además encontramos otros algoritmos, los cuales no son exactos (poseen un porciento de fallo) pero que mejoran considerablemente la complejidad temporal.

4.1 Mejorando el algoritmo de Fuerza Bruta

Manteniendo la idea esencial del algoritmo, así como la entrada y la salida, añadimos las siguientes mejoras, que consisten en hacer verificaciones a condiciones necesarias que deben cumplir dos grafos isomorfos:

1. Revisamos los grados de cada vértice de los grafos y los ordenamos, si alguno difiere, los grafos no son isomorfos

2. Por cada nodo, revisamos los grados de los vértices vecinos y los ordenamos, si encontramos que, para ningún vértice del mismo grado, los grados de los vecinos no corresponden, entonces no son isomorfos

3. Utilizamos una estrategia algebraica, construimos matrices de permutación, a partir de la identidad y cada una se aplica a la matriz de adyacencia

Complejidad temporal con las mejoras

En el caso peor, la complejidad del algoritmo es ligeramente más mala que la del fuerza bruta anterior, $O(n^2 + n! \cdot n^2)$, dado que aún debe buscar en el espacio de las permutaciones y comparar las matrices resultantes de aplicar estas permutaciones, sin embargo, al realizar antes un filtrado, verificando que se cumplan las condiciones necesarias (ambas se verifican en $O(n^2)$), hace que posea esta complejidad ligeramente peor, aunque en la práctica este enfoque es mejor, dado que puede descartar muchos más grafos que no sean isomorfos, en $O(n^2)$.

La complejidad espacial es de $(O(n^2))$ debido al trabajo con las matrices de adyacencia.

4.2 Utilizando algoritmos aproximados

Teniendo ahora una aproximación un poco más inteligente (igualmente bruta), decidimos implementar varias soluciones aproximadas, pero antes demostremos algunas condiciones necesarias.

4.3 Demostración de condiciones necesarias en grafos isomorfos

Si dos grafos G_1 y G_2 son isomorfos entonces cumplen que:

1. Ambos grafos deben tener el mismo número de vértices
2. Para cada vértice, el número de conexiones (grado) debe ser el mismo en ambos grafos. Esto significa que la secuencia de grados de los vértices debe coincidir cuando se ordena.
3. Si los vértices están etiquetados, las correspondencias de etiquetas también deben coincidir.

1: Como G_1 y G_2 son isomorfos existe una función biyectiva f que mapea cada vértice de G_1 a G_2, como f es inyectiva entonces a cada nodo de G_1 le corresponde un único nodo en G_2 y como es sobreyectiva todos los nodos de G_2 son cubiertos, por tanto, G_1 y G_2 poseen la misma cantidad de nodos
2: Supongamos que el grado de un par de nodos $u \in G_1$ y $v \in G_2$ es distinto $(gr(u) > gr(v))$. Dado que G_1 y G_2 son isomorfos entonces existe f una función sobreyectiva que mapea cada vértice de G_1 a G_2, y sea $v = f(u)$, luego como $gr(u) > gr(f(u))$, va a existir una arista $(u, w) \in G_1$, $(f(u), f(w)) \notin G_2$, lo cual es una contradicción con la definición de isomorfismo
3: Supongamos que las etiquetas no coinciden(Llamemos a las etiquetas C_u etiqueta del nodo u). Sean $u \in G_1$ y $v \in G_1$, tal que $C_u \neq C_v$ pero $C_{f(u)} = C_{f(v)}$.
Dado que G_1 y G_2 son isomorfos entonces existe f una función sobreyectiva que mapea cada vértice de G_1 a G_2. , como f mapea los vértices de G_1 a G_2 debe mantener la correspondencia, por lo tanto las etiquetas $C_{f(u)} \neq C_{f(v)}$, lo cual contradice nuestra suposición inicial. por lo tanto, la suposición es incorrecta.

Esta última demostración nos permite formar clases de equivalencia entre los nodos que mantengan las mismas etiquetas en ambos grafos.

4.4 Algoritmo de verificación por aristas

- **Propósito:** Verificar si dos grafos, representados por matrices de adyacencia G_1 y G_2, son isomorfos utilizando una estrategia basada en los grados de los nodos y las aristas.
- **Entradas:** Dos grafos G_1 y G_2 representados por matrices de adyacencia de $n \times n$, donde n es la cantidad de nodos de cada grafo

- **Salida:** Un valor booleano, donde true significa que ambos grafos son isomorfos, y false que no

Pasos del algoritmo:

1. Calcula el grado de cada nodo de los grafos G_1 y G_2
2. Recorre los grafos buscando las aristas entre los pares de nodos i, j si existe una arista entre los nodos, guarda la arista de la siguiente forma (i, j) donde i es el nodo con menor grado
3. Luego ordena las aristas en función de los grados de los nodos que relacionan, y si no coinciden luego de la ordenación, los grafos no son isomorfos

Complejidad: Como para cada nodo hay que calcular el grado que posee, este cálculo se realiza en $O(n^2)$, luego revisa que aristas existe entre cada par de nodos tambien en $O(n^2)$, por ultimo hace un ordenamiento de las aristas lo cual se hace en $O(m \log m)$, en el peor caso sería $O(n^2 \log n)$. Por tanto la complejidad del algoritmo en el caso pero es de $O(n^2 \log n)$.

La complejidad espacial es de $O(n^2)$ debido al trabajo con las matrices de adyacencia

Este algoritmo aproximado, tiene un margen de error del 2.1%, para mil casos de prueba falló en 21

4.5 Algoritmo de verificación por etiquetas

Basándonos en la idea del algoritmo de Weisfeiler-Leman, creamos el siguiente algoritmo. La idea detrás de este enfoque es refinar las etiquetas de los nodos del grafo iterativamente, de manera que vértices estructuralmente equivalentes reciban las mismas etiquetas, formando en cada iteración nuevas clases de equivalencia.

- **Entradas:** Dos grafos G_1 y G_2 representados por matrices de adyacencia de $n \times n$, donde n es la cantidad de nodos de cada grafo
- **Salida:** Un valor booleano, donde true significa que ambos grafos son isomorfos, y false que no

Las etiquetas son otorgadas en cada iteración, y tienen la siguiente forma:

1. En esta iteración se les otorga a los nodos una etiqueta basada en su grado. La etiqueta en este caso consiste precisamente en el grado del nodo
2. Aquí se refinan basándose en los grados de sus vecinos. Con la etiqueta anterior, se verifican las etiquetas de los nodos vecinos para cada uno de los nodos del grafo, y se ordenan de mayor a menor, dando esta etiqueta a cada vértice

3. En esta última se refinan las etiquetas basándose en las etiquetas de sus vecinos, toma en cuenta no solo los grados de los vecinos, sino también la estructura de las conexiones entre los vecinos de un nodo y cómo estas conexiones se relacionan con sus respectivas etiquetas de refinamiento anteriores

4. Si después de estas iteraciones, las clases de equivalencia de cada grafo coinciden, se concluye que los grafos podrían ser isomorfos

A este algoritmo se le pueden incluir muchos más refinamientos y etiquetas, pero aun así no será exacto, debido a que en cada iteración lo que se añaden para refinar las etiquetas, son condiciones necesarias que deben cumplir dos grafos para que sean isomorfos, el hecho de que dos nodos reciban la misma etiqueta no garantiza que sean equivalentes. Luego si se añaden suficientes condiciones necesarias, el algoritmo podrá identificar aún más grafos como isomorfos. Esta aproximación, si es exacta para descartar grafos que no son isomorfos.

La complejidad del algoritmo es $O(n^2 \log n)$, debido a que en cada iteración para el refinamiento de las etiquetas, necesito revisar las etiquetas de mis vecinos, lo cual da una complejidad de $O(n)$, luego se debe ordenar esas etiquetas ($O(n \log n)$), luego el coste por iteración seria $O(n^2 \log n)$, la comparación se realiza en $O(n^2)$ siendo n la cantidad de nodos en los grafos.

La complejidad espacial es $O(n^2)$ debido al trabajo con las matrices de adyacencia.