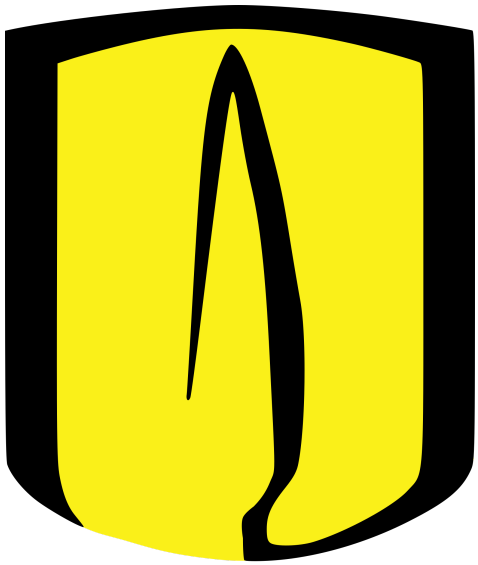

ISIS 1105 Diseño y Análisis de Algoritmos

Proyecto Parte 2 - Redes con redundancia

Daniel Santiago Muñoz
Juan David Ortiz

Universidad de los Andes
2025-II

Idea general del problema

El problema consiste en determinar si, para cada conexión entre dos nodos **A** y **B** de tipo *coaxial*, también existe una conexión equivalente de tipo *fibra óptica* que permita la comunicación entre ellos, y viceversa.

En otras palabras, queremos saber si ambos tipos de red (fibra y coaxial) generan las **mismas componentes conexas**, es decir, si en ambos casos los mismos nodos están conectados entre sí.

De esto se deduce que existen en realidad **dos grafos**:

$$G_{\text{fibra}} \quad \text{y} \quad G_{\text{coaxial}}$$

La red se considera **redundante** si ambos grafos tienen:

1. La misma cantidad de componentes conexas, y
2. Cada componente está compuesta exactamente por los mismos nodos.

Estructura de datos

Antes de describir el algoritmo, partimos del hecho de que lo realmente importante es saber si **dos nodos están conectados** o no; el camino específico entre ellos es irrelevante.

Por esta razón, la estructura de datos más apropiada es el **Union-Find (UF)**, también mencionado como **Disjoint Set Union (DSU)**.

En clase se menciona que la complejidad de las operaciones `union` y `find` depende de la altura de los árboles que representan los conjuntos. Para mantener esa altura baja, se aplican dos optimizaciones:

- **Path compression**: durante la operación `find`, se reestructura el árbol para que cada nodo apunte directamente a la raíz.
- **Union by rank o union by size**: se une el árbol más pequeño al más grande.

Gracias a estas técnicas, la complejidad amortizada de las operaciones `union` y `find` es:

$$O(\alpha(n))$$

donde α es la función inversa de Ackermann, que crece tan lentamente que puede considerarse **constante** para los tamaños del problema propuesto [1].

Algoritmo 1

El algoritmo inicia creando **dos estructuras Union-Find** independientes de tamaño `n`, representando respectivamente las conexiones de *fibra óptica* y de *cable coaxial*.
Inicialmente, cada nodo pertenece a su propio conjunto.

A medida que se procesan las conexiones una a una:

1. Se aplica `union(i, j)` en la estructura correspondiente según el tipo de conexión `k`.
2. Después de cada unión, se verifica si ambas estructuras **definen las mismas componentes conexas**, es decir, si para todo par de nodos `(u, v)` se cumple:

$$find_{\text{fibra}}(u) = find_{\text{fibra}}(v) \iff find_{\text{coaxial}}(u) = find_{\text{coaxial}}(v)$$

3. Si la condición se cumple para todos los nodos, la red es **redundante** (se imprime `1`); en caso contrario, **no es redundante** (se imprime `0`).

Complejidad computacional y espacial

Cone este enfoque, verificar si la red es redundante después de cada conexión implica comparar el estado de conectividad de **cada par de nodos posible**.
Esto se traduce en un **doble ciclo**, lo que genera una complejidad temporal de $O(n^2)$ por verificación.
Dado que esta verificación se realiza después de agregar cada una de las m conexiones, la complejidad total del algoritmo es:

$$O(mn^2)$$

En cuanto al uso de memoria, las estructuras **Union-Find** empleadas para representar las conexiones de fibra óptica y de cable coaxial almacenan, para cada nodo, un identificador de padre y, para el *path compresion*, un rango o tamaño de conjunto.
Por tanto, la complejidad espacial total del algoritmo es lineal respecto al número de nodos:

$$O(n)$$

Optimizaciones

Revisando el problema, el estado antes de hacer `union`, nos esta dando mucha información, lo cual puede ayudarnos a dar la respuesta de forma mas rapida para diferentes casos.

El nuevo `union` genera un ciclo

Al generar un ciclo, las estructuras **UF** no cambian, por lo que seguira en el mismo estado antes de hacer el `union`.

El nuevo `union` No genera un ciclo

Si no genera un ciclo, implica que unió dos conjuntos diferentes en el mismo grafo.

1. Si antes del `union` era redundante, Ahora no lo puede ser ya que en un grafo cambio la cantidad de conjuntos conectados.
2. Si antes del `union` No era redundante, esto pudo haberlo convetido o no (Algoritmo 1).

Con estas observaciones se puede dar una respuesta rapido en los diferentes caso, sin emabrgo el cuello de botella sigue siendo en el algoritmo 1.

¡Perfecto! La redacción es clara y la explicación es correcta. Solo tengo unas sugerencias menores para mejorarla:

Algoritmo 2 - Modificación al Algoritmo 1

En esta versión, en lugar de comparar todos los pares de nodos, verificamos directamente la equivalencia de los **conjuntos de componentes conexas**.

Para esto consideramos dos enfoques:

Opción A: Usar **Quick-Find**

- Implementamos **Quick-Find** que siempre usa como representante el número más pequeño
- Costo de `union`: $O(n)$ (actualiza todos los elementos del componente)
- Verificación: comparar los dos arreglos en $O(n)$
- Si los arreglos son idénticos → redundante, si no → no redundante

Opción B: Usar **Union-Find** con representantes

- Mantenemos **Union-Find** normal pero con estructura auxiliar para obtener representantes (Como en **Quick-Find**, puede ser el menor número del conjunto o directamente la raiz).
- Podemos aplicar las optimizaciones anteriores (detección de ciclos, cambios de estado)
- Verificación: $O(n)$ reconstruyendo el mapeo de componentes

Ventajas Comparativas

- **Quick-Find**: La implementación es más simple
- **Union-Find**: Permite aplicar optimizaciones para evitar verificaciones completas

Complejidad Final:
En ambos casos, como la verificación se realiza m veces:

$$O(m \cdot n)$$

Algoritmo 3 - Uso de contadores y mapas

En esta versión, el objetivo es **evitar comparar componentes completas** o pares de nodos en cada paso. Para ello, se mantiene información sobre las relaciones entre las componentes de **fibra óptica** y **coaxial** de forma incremental, usando estructuras auxiliares.

Idea principal

Adicionalmente a los **UF**, se mantienen dos mapas que relacionan los componentes de ambos grafos:

- `ff2c`: asocia cada componente de **fibra** con los componentes de **coaxial** que contiene.
- `cf2f`: asocia cada componente de **coaxial** con los componentes de **fibra** que contiene.

Decimos entonces que `ff2c[i]` son todos los conjuntos de **UF** *coaxial* que estan unidos al conjunto i del **UF** de fibra. `cf2f[j]` es lo puesto, son todos los conjuntos de **UF** *fibra* que estan unidos al conjunto j del **UF** *coaxial* .

Lo anterior implica que

$$r \in \texttt{ff2c}[i] \iff i \in \texttt{cf2f}[r]$$

Notese que r e i son raices para la red coaxial y de fibra respectivamente.

Ademas se tienen `froots` y `croots` que son la cantidad de raices en cada **UF**, en otras palabras, la cantidad de conjuntos en cada grafo.

Notese que `froots` y `croots` nunca pueden ser mayor a n. Hacer un `union` entre dos conjuntos que no estaban previamente conectados, disminuye la cantidad de conjuntos en la red correspondiente.

Finalmente se encuentra `pairs`, el cual representa la cantidad de **pares únicos** `(fibra_root, coaxial_root)` donde existe al menos un nodo que pertenece a ambos componentes.

Decimos que existe un `pair` entre una raíz de fibra `i` y una raíz coaxial `c` si `c ∈ ff2c[i]` (o equivalentemente `i ∈ cf2f[c]`).

La clave es que al realizar un `union` entre dos componentes de fibra, si una misma raíz coaxial `c` estaba presente en ambos componentes antes de la unión, entonces dos `pairs` distintos `(A,c)` y `(B,c)` se consolidan en un solo `pair` `(A∪B,c)`, reduciendo el conteo total.

Funcionamiento general

1. **Inicialización**: cada nodo inicia en su propio conjunto, tanto en `dsu_fibra` como en `dsu_coaxial`. Por lo tanto, inicialmente hay `n` componentes en cada grafo, y cada componente de un tipo se asocia uno a uno con una del otro tipo.

Como cada nodo esta aislado, `froots=croots=pairs=n`.

2. **Procesamiento de conexiones**: por cada nueva conexión `(u, v, k)`:
   - Se determinan las raíces actuales de los nodos involucrados en el DSU correspondiente.
   - Si pertenecen a diferentes componentes, se realiza la unión y se actualizan las estructuras auxiliares `ff2c` y `cf2f`:
     - Se calcula la **intersección** entre los conjuntos relacionados para reducir los `pairs` redundantes.
     - Se fusionan los conjuntos correspondientes, manteniendo la consistencia entre las dos estructuras.
3. **Verificación de redundancia**: después de cada conexión, la red se considera redundante si se cumple que:

$$\text{pares} = \text{componentes}_{\text{fibra}} = \text{componentes}_{\text{coaxial}}$$

   En ese caso se imprime `1`, y `0` en caso contrario.

Complejidad

- Cada operación `find` o `union` en los DSU tiene costo amortizado $O(\alpha(n))$
- Las operaciones con los mapas `ff2c` y `cf2f` realizan:
  - Intersección de conjuntos: $O(\min(|A|, |B|))$ (Encontrar **un** elemento de A en B toma $O(1)$ por usar hash)
  - Transferencia de elementos: $O(\min(|A|, |B|))$

La clave es que siempre trabajamos con el conjunto más pequeño. En el peor caso, una operación individual podría costar $O(k), \quad k \leq n/2$, pero gracias a la **unión por tamaño**, cada vez que movemos un conjunto, su tamaño se duplica como mínimo. Esto garantiza que cada elemento se mueve como máximo $O(\log n)$ veces durante todas las operaciones.

Por lo tanto, el **costo amortizado por operación** es $O(\log n)$.

Como realizamos m operaciones, la complejidad temporal amortizada total es:

$$O(m \log n)$$

Respecto a la complejidad espacial:

- Los dos estructuras **DSU** requieren $O(n)$
- Los mapas `ff2c` y `cf2f` pueden almacenar hasta $O(n^2)$ elementos en el peor caso, ya que cada uno de los n conjuntos podría contener $O(n)$ elementos.

La complejidad espacial total es

$$O(n^2)$$

Referencias

[1] GeeksforGeeks, “Introduction to Disjoint Set Data Structure (Union-Find Algorithm),” *GeeksforGeeks*, Jul. 24, 2025. [Online]. Available: https://www.geeksforgeeks.org/dsa/introduction-to-disjoint-set-data-structure-or-union-find-algorithm/