Optimización de Redes de Distribución Energética

Daniel Machado Perez
Ariel Gonzalez Gómez
Alex Bas Beovides
Leonardo Artiles Montero

7 de febrero de 2025

Contexto práctico

En una red de distribución energética de la UNE, las subestaciones están interconectadas mediante una red de líneas de transmisión. Cada línea tiene un costo de mantenimiento. Cuando ocurre un fallo en una línea de transmisión [1], la red debe reorganizarse dinámicamente para minimizar los costos de reparación y garantizar que el suministro energético a las regiones más críticas sea continuo [2].

El jefe de la UNE desea dividir la red en k grupos independientes (subredes autónomas) durante situaciones de mantenimiento o emergencias, asegurándose de que la desconexión sea lo menos costosa posible (en términos de la suma de los costos de las líneas cortadas).

Adicionalmente, al jefe de la UNE le llegó de las altas esferas la orientación de resolver otra tarea. Se quiere garantizar que existan zonas priorizadas que no sufran cortes de electricidad por su relevancia. Estas zonas pueden incluir hospitales, instituciones importantes como el ICRT, residencias de mandatarios... Por ello, se plantea la necesidad de determinar la cantidad mínima de aristas que deben removerse para obtener al menos una componente conexa con exactamente t nodos, representando estas zonas priorizadas.

Formalización del problema

Dado un grafo ponderado $G = (V, E)$, donde:

- V: Conjunto de nodos que representan subestaciones.
- E: Conjunto de aristas que representan líneas de transmisión, cada una con un costo $w(e)$ asociado.

[1] Nada frecuente, por cierto.
[2] Qué buen chiste.

- k: Número de subredes requeridas.
- t: Tamaño de la componente priorizada.

Problemas:

1. Encontrar un conjunto mínimo de aristas $E' \subseteq E$ tal que al eliminar E':
   a) G se divide en k componentes conexas.
   b) La suma de los costos de las aristas eliminadas $\sum_{e \in E'} w(e)$ es mínima.
2. Determinar la cantidad mínima de aristas a remover del grafo G tal que quede al menos una componente conexa con exactamente t nodos.

Justificación del Problema

El problema que planteamos tiene una relevancia práctica evidente y un interés computacional significativo, como se detalla a continuación:

Importancia Práctica

- **Gestión de Redes Energéticas:** Las redes de transmisión eléctrica son infraestructuras críticas en cualquier sociedad moderna. Garantizar que dichas redes sean resilientes a fallos y puedan reorganizarse eficientemente en situaciones de emergencia o mantenimiento es fundamental para asegurar la continuidad del suministro eléctrico. Este problema no solo aborda la minimización de costos económicos asociados con los cortes, sino también prioriza el suministro de energía a zonas críticas como hospitales, instituciones públicas relevantes o residencias estratégicas.
- **Seguridad y Resiliencia:** Dividir la red en k subredes autónomas durante emergencias no solo reduce los costos, sino que fortalece la seguridad al evitar apagones masivos. Además, identificar las aristas mínimas necesarias para garantizar una subred priorizada permite planificar contingencias con antelación.

Interés Computacional

- **Complejidad Algorítmica:** El problema presenta retos importantes desde el punto de vista computacional. Dividir un grafo ponderado en k subcomponentes conexas con costo mínimo, así como garantizar subredes priorizadas, son problemas que combinan técnicas avanzadas de análisis de grafos, optimización y conectividad.
- **Modelación Compleja:** Formalizar este problema implica trabajar con múltiples restricciones, como los costos de las aristas, cardinalidad de componentes conexas y prioridades en nodos específicos. Estos desafíos hacen que sea ideal para aplicar y demostrar habilidades avanzadas de diseño y análisis algorítmico.

Aplicaciones Generales

El problema general de partición de grafos tiene aplicaciones relevantes en múltiples dominios. Entre las más destacadas se encuentran las siguientes:

- **Particionamiento y Clustering:** La partición de grafos es una técnica ampliamente utilizada en el análisis de datos. En este contexto, las similitudes entre objetos se modelan como una matriz de adyacencia ponderada, que contiene la información necesaria para realizar agrupamientos óptimos.
- **Confiabilidad de Redes:** Determinar la conectividad mínima de una red es esencial en el diseño y análisis de redes confiables. Las aplicaciones incluyen redes de telecomunicaciones, diseño de infraestructura crítica y sistemas distribuidos.
- **Algoritmos de Optimización:** Los cálculos de cortes mínimos son fundamentales en algoritmos de eliminación de subtours para resolver problemas combinatorios como el *Traveling Salesman Problem* (TSP). Este enfoque ha demostrado ser clave en algoritmos basados en planos de corte para problemas de grafos conexos.
- **Diseño de Circuitos y CAD:** En diseño de circuitos integrados (VLSI), la partición eficiente de circuitos es un paso crucial en herramientas de diseño asistido por computadora (CAD). Esta técnica también se utiliza en modelos de optimización a gran escala y problemas de elementos finitos.
- **Compiladores para Lenguajes Paralelos:** En compiladores de lenguajes paralelos, la partición de grafos optimiza la distribución de operaciones del programa en arquitecturas de memoria distribuida, mejorando la eficiencia computacional.
- **Modelos Físicos y Visión Computacional:** Los problemas de partición y corte en grafos también aparecen en la resolución de modelos físicos como los *spin glass models*, en programación cuadrática 0-1 sin restricciones, y en problemas de segmentación de imágenes, donde las aristas modelan relaciones entre píxeles adyacentes.

Segunda Parte del Problema: Subred Priorizada

Un aspecto crucial del problema es la identificación de las aristas mínimas necesarias para garantizar que una subred específica, considerada prioritaria, permanezca operativa en situaciones de emergencia. Esto tiene aplicaciones concretas como:

- **Planificación de Contingencias:** En sistemas críticos, como redes eléctricas o de telecomunicaciones, es esencial garantizar que ciertas áreas estratégicas sigan operativas. Por ejemplo, hospitales, estaciones de emergencia y centros de comando requieren una conexión continua incluso en escenarios de fallo masivo.
- **Optimización de Recursos:** La identificación de aristas mínimas permite optimizar los recursos disponibles, como redundancias de red, mantenimiento preventivo y planificación de reparaciones, reduciendo los costos operativos.

- **Resiliencia en Infraestructuras Críticas:** El diseño de infraestructuras resilientes exige priorizar las conexiones más críticas dentro de la red. Este enfoque no solo mejora la robustez general, sino que también asegura una recuperación más rápida y eficiente ante eventos adversos.

La solución de esta parte del problema se basa en un análisis detallado de la conectividad de la red, utilizando técnicas avanzadas de teoría de grafos y algoritmos de optimización para garantizar que las restricciones se cumplan con el menor costo posible.

Trabajo propuesto [3]

1. **Demostración de NP-Completitud**: Demostración de que la primera parte del problema es **NP-Completo**, utilizando una reducción al problema *Max-Clique*.
2. **Reducción a árboles y solución greedy**: Reducción de la primera parte del problema a árboles y diseñar una solución greedy con complejidad $O(n \log n)$.
3. **Algoritmo de aproximación con flujo**: Proponer un algoritmo de aproximación basado en técnicas de flujo para resolver la primera parte del problema en grafos generales.
4. **Reducción de la segunda parte del problema a árboles**: Reducir la segunda parte del problema a árboles y resolverlo utilizando un increible algoritmo de programación dinámica, con una demostración de complejidad $O(n^2)$ inesperada.

---

1. Demostración de NP-Completitud

2. Demostración de que el problema de decisión es NP-Completo

El problema de decisión se puede definir de la siguiente manera:

Entrada: Un grafo $G = (V, E)$, un número entero positivo k, una función no negativa de pesos sobre las aristas $w(e)$ y un umbral M.

Pregunta: Determinar si es posible particionar G en k componentes conexas de manera que la suma de los pesos de las aristas que conectan nodos de componentes conexas distintas sea a lo sumo M.

Por simplicidad, en lo siguiente llamaremos $P - decision$ a este problema.

[3] ATENCIÓN SPOILERS

2.1. Pertenencia a NP

Para verificar si $P-decision$ pertenece a NP, se necesita demostrar que cualquier solución candidata puede ser verificada en tiempo polinomial.

Dado un grafo $G = (V, E)$, una partición de los vértices en k subconjuntos $S_1, S_2, \ldots, S_k$ y un umbral M, se puede verificar en tiempo polinomial si la solución es válida de la siguiente manera:

- Comprobar que la partición es válida, es decir, que cada S_i es no vacío, la unión de todos los S_i es V y las intersecciones son vacías ($S_i \cap S_j = \emptyset$ para $i \neq j$). Esto se puede realizar en tiempo $O(|V|)$ usando estructuras para marcar vértices visitados.
- Comprobar que cada subconjunto S_i es una componente conexa. Esto se puede hacer con BFS/DFS en complejidad $O(|V| + |E|)$.
- Comprobar que la suma de los pesos de las aristas que cruzan componentes conexas es menor o igual que M. Esto se resuelve iterando por todas las aristas, y si una arista involucra vértices de subconjuntos distintos, se suma $w(e)$ al total. Al final se verifica si $w(e) \leq M$. La complejidad es $O(|E|)$.

Dado que cada paso se puede realizar en tiempo polinomial, $P - decision$ pertenece a NP.

2.2. NP-Hardness

Procederemos a demostrar que $P - decision$ es NP-Hard mediante una reducción polinomial desde el problema de decisión de **Clique Máximo**, que es NP-Completo.

Recordemos la definición de **Clique Máximo**:
Entrada: Un grafo no dirigido $G = (V, E)$ y $M \in \mathbb{Z}^+$
Pregunta: Determinar si existe un clique en G de tamaño mayor o igual que M.

2.2.1. Reducción desde Clique Máximo

Consideremos un grafo no dirigido con pesos de $\{0, 1\}$ en las aristas. Encontrar la mínima configuración de aristas que al ser eliminadas particionan el grafo en k componentes conexas, es equivalente a maximizar la cantidad de aristas dentro de cada componente conexa. Supongamos que G tiene un clique H de tamaño $M = |V| - (k - 1)$. Entonces el número de aristas entre cualquier vértice $v \notin H$ y $u \in H$ es menor que la cantidad de aristas entre vértices dentro de H, pues en un clique hay una arista entre cualquier par de vértices:

$$\sum_{u \in H, v \notin H} w(e_{uv}) < \sum_{u \in H, v \in H} w(e_{uv})$$

Entonces cuando se divide el grafo en k particiones con una cantidad mínima de aristas de cruce entre componentes conexas, el clique H debe ser una de esas componentes.

$\Rightarrow$ El problema de decisión de **Clique Máximo** puede reducirse en tiempo polinomial a $P - decision$.
Como se sabe que **Clique Máximo** es NP-Completo $\Rightarrow$ $P - decision$ es NP-Hard.

2.3. NP-Completitud

Dado que $P - decision$ pertenece a NP y es NP-Hard, concluimos que es NP-Completo.
■

3. Demostración de que el problema de optimización es NP-Hard

El problema de optimización se puede definir de la siguiente manera:

Entrada: Un grafo $G = (V, E)$, un número entero positivo k y una función no negativa de pesos sobre las aristas $w(e)$.

Salida: Encontrar una partición de G en k componentes conexas de manera que la suma de los pesos de las aristas que conectan nodos de componentes conexas distintas es mínima.

Por simplicidad, en lo siguiente llamaremos $P - optimization$ a este problema.

3.1. NP-Hardness

Como se conoce que dado un problema, si su variante de decisión es NP-Completo entonces su variante de optimización es NP-Hard, y se demostró que $P - decision$ es NP-Completo $\Rightarrow$ $P - optimization$ es NP-Hard.

4. Reducción a árboles y solución greedy

En esta sección se expone la idea de los algoritmos Greedy para resolver el problema en dos casos particulares: cuando el grafo es un árbol y cuando es un bosque.

5. Caso en que el grafo es un árbol

Dado un árbol $T = (V, E)$ con pesos no negativos en sus aristas, la idea es aprovechar la propiedad de que, al remover $k - 1$ aristas, el árbol se divide en k componentes conexas. El algoritmo Greedy consiste en:

1. Ordenar las aristas del árbol en forma no decreciente según su peso.
2. Seleccionar las $k - 1$ aristas de menor peso.

5.1. Análisis de Complejidad

- Ordenar las aristas requiere $O(n \log n)$, donde n es el número de vértices (recordando que un árbol tiene $n-1$ aristas).
- La selección de las $k-1$ aristas es $O(k)$, lo cual es polinomial.

Por lo tanto, el algoritmo Greedy para árboles se resuelve en tiempo polinomial.

5.2. Demostración de Correctitud

Recordemos que, en un árbol, al eliminar ciertas aristas se incrementa el número de componentes conexas.

[Número de componentes en un árbol] Si eliminamos todas las aristas de un árbol con n vértices, es decir, eliminamos $n-1$ aristas, obtenemos n componentes conexas.

Procedemos por inducción en el número de vértices n.

Caso base: Si $n = 2$, un árbol tiene exactamente una arista. Al remover esta única arista, el grafo se divide en 2 componentes conexas, lo cual cumple la afirmación.

Hipótesis de inducción: Supongamos que para un árbol con $n = k-1$ vértices, removiendo $(k-2)$ aristas obtenemos $k-1$ componentes conexas.

Paso inductivo: Consideremos un árbol T con k vértices. Removamos una arista e del árbol. Al hacerlo, el árbol se divide en dos subárboles, digamos T_1 y T_2, que tienen n_1 y n_2 vértices respectivamente, donde $n_1+n_2 = k$ y ambos $n_1, n_2 \geq 1$. Por la hipótesis de inducción, removiendo $n_1 - 1$ aristas de T_1 se obtienen n_1 componentes conexas y removiendo $n_2 - 1$ aristas de T_2 se obtienen n_2 componentes conexas. En total, removiendo $1+(n_1-1)+(n_2-1) = n_1 + n_2 - 1 = k-1$ aristas, obtenemos $n_1 + n_2 = k$ componentes conexas.

5.3. Pseudocódigo

```
1 Entrada:  rbol  T = (V, E) con pesos no negativos, entero k
2 Salida: Conjunto de aristas C que forma un corte m nimo k
3
4 1. Ordenar las aristas E de T en forma no decreciente seg n su peso:
5       w(e1) <= w(e2) <= ... <= w(e_{|V|-1})
6 2. Seleccionar las primeras k-1 aristas de la lista ordenada y
     asignarlas a C.
7 3. Retornar C.
```

6. Caso en que el grafo es un bosque

Para un bosque $F = (V, E)$ con t componentes conexas, la idea es similar. Se debe notar que, para obtener una partición en k componentes, es necesario remover $k-t$ aristas, ya que el bosque ya cuenta con t componentes. El algoritmo Greedy consiste en:

1. Ordenar todas las aristas del bosque en forma no decreciente según su peso.
2. Seleccionar las $k - t$ aristas de menor peso.

6.1. Análisis de Complejidad

- Ordenar las aristas requiere $O(|E| \log |E|)$; dado que el bosque tiene a lo sumo $|V| - 1$ aristas, se cumple que es polinomial.
- La selección de las $k - t$ aristas es $O(k)$.

Por lo tanto, este algoritmo también se ejecuta en tiempo polinomial.

6.2. Pseudocódigo

```
1 Entrada: Bosque F = (V, E) con pesos no negativos, entero k,
2          donde t es el n mero de componentes conexas en F.
3 Salida: Conjunto de aristas C que forma un corte m nimo k en F
4
5 1. Calcular t, el n mero de componentes conexas de F.
6 2. Ordenar las aristas E de F en forma no decreciente seg n su peso:
7       w(e1) <= w(e2) <= ... <= w(e_{|V|-?})
8 3. Seleccionar las primeras k-t aristas de la lista ordenada y
     asignarlas a C.
9 4. Retornar C.
```

---

7. Reducción de la segunda parte del problema a árboles

8. Enunciado del problema

Dado un árbol con N nodos, calcular para todo $K \leq N$ el mínimo número de aristas a ser cortadas tal que quede una componente conexa con exactamente K nodos.

9. Solución

Resolvamos el problema utilizando programación dinámica. Rootiemos el árbol T, llamemos T_u al subárbol del nodo u. Formulemos nuestra DP de la siguiente forma:

Sea $dp[u][i]$ la mínima cantidad de aristas que hay que eliminar del árbol para obtener una componente conexa en T_u que contenga al nodo u y sea de tamaño i. Es trivial que teniendo este arreglo rellenado podemos calcular la solución para cada K en $O(N)$.

Veamos cómo calcular esta DP. Denotemos T_u^j como el árbol T_u pero conteniendo solo sus j primeros hijos y sus subárboles.

Llamemos $dp_j[u][i]$ el valor de $dp[u][i]$ en T_u^j, lo cual es básicamente habiendo procesado los j primeros hijos de u. La idea es calcular $dp_h[u][i]$ donde h es la cantidad de nodos hijos de u, es decir, $h = |\{v_1, v_2, v_3, \ldots, v_h\}|$ donde v_i es el hijo i-ésimo de u

.

Empezamos con $j = 0$, en ese caso u es tomado como una hoja en T_u^j, así que $dp_j[u][1] = 0$. Luego computamos los valores de $dp_j[u][i]$ siguiendo la siguiente fórmula:

$$dp_j[u][i] = \text{mín}\left(dp_{j-1}[u][i] + 1, \min_{\substack{a+b=i \\ a,b\geq 1}}(dp_{j-1}[u][a] + dp[v_j][b])\right)$$

Lo cual es básicamente tomar el mínimo entre:

- Cortar la arista hacia el j-ésimo hijo de u.
- El mínimo para todo a, b de tener a nodos en T_u^{j-1} y tener b nodos en el subárbol del j-ésimo hijo de u.

10. Análisis de complejidad

10.1. Cota máxima del algoritmo

Podríamos establecer una cota máxima de $O(N^3)$ para nuestro algoritmo, ya que para cada nodo u, debemos llenar todos los valores de i en $dp[u][i]$, y para hacer esto tenemos que recorrer todas las posibles particiones a, b tal que $a + b = i$. Sin embargo, podemos encontrar una mejor cota.

10.2. Enunciado

Para un nodo fijo u, el número de operaciones necesarias para computar todos los valores de $dp[v][:]$ donde v es un nodo que pertenece a T_u es $O(|T_u|^2)$.

10.3. Demostración

Demostremos este resultado por inducción sobre la profundidad del árbol T_u.

- Caso base: Si T_u es una hoja, el lema se cumple trivialmente, ya que solo necesitamos $O(1)$ operaciones.
- Paso inductivo: Supongamos que el nodo u tiene h hijos, denotados como $v_1, v_2, \ldots, v_h$, y definamos $a_j = |T_{v_j}|$.

Al fusionar el j-ésimo hijo, es decir, al calcular $dp_j[u][:]$, recorremos todos los valores de $dp_{j-1}[u][a]$ y $dp[v_j][b]$ para todo a y b. Sin embargo, el valor de a puede ser escogido en $|T_u^{j-1}|$ formas y el de b en $|T_{v_j}|$ formas, lo que nos da un número total de operaciones:

$$O(|T_u^{j-1}| \cdot |T_{v_j}|) = O\left((1 + a_1 + a_2 + \cdots + a_{j-1}) \cdot a_j\right)$$

Por la hipótesis de inducción, calcular todos los valores de dp para los nodos en el subárbol de u tomará:

$$O\left(\sum_{j=1}^{h} a_j^2\right)$$

Sumando el número de operaciones necesarias para calcular los valores de $dp[v][:]$ para todos los vértices v pertenecientes a T_u, obtenemos:

$$O\left(\sum_{i=1}^{h}\sum_{j=1}^{h} a_i \cdot a_j + \sum_{j=1}^{h} a_j + \sum_{j=1}^{h} a_j^2\right)$$

Como el primer término es una doble sumatoria sobre los tamaños de los subárboles, podemos agrupar los términos para obtener:

$$O\left(\left(1 + \sum_{j=1}^{h} a_j\right)^2\right) = O(|T_u|^2)$$

10.4. Demostración combinatoria alternativa

También podemos obtener una demostración combinatoria del enunciado. Al fusionar un subárbol se realizan:

$$O(|T_u^{j-1}| \cdot |T_{v_j}|)$$

operaciones. Esto se puede interpretar como el número de pares de nodos que tienen a u como su ancestro común más cercano (LCA). Como cualquier par de nodos tiene un único LCA, cada par se cuenta una sola vez, lo que nos lleva a una cota total de:

$$O(N^2)$$

11. Demostración de Correctitud

Sea un árbol T con N nodos y sea T_u el subárbol enraizado en el nodo u. Para cada nodo u y para cada entero i en $1, \ldots, |T_u|$, definimos $dp[u][i]$ como el mínimo número de aristas a cortar en T_u para obtener una componente conexa de tamaño i que contiene a u. Además, al procesar el nodo u consideramos sus hijos en un orden fijo y definimos T_u^j como el árbol formado por u y los subárboles de sus primeros j hijos. Sea

$dp_j[u][i] =$ costo mínimo en T_u^j para obtener una componente conexa de tamaño i que contiene a u.

El caso base es $j = 0$, donde el único nodo es u, de modo que

$$dp_0[u][1] = 0,$$

y para $i \neq 1$ se establece $dp_0[u][i] = +\infty$ (o un valor que indique imposibilidad).

Procedemos a demostrar la correctitud mediante doble inducción: primero sobre el número de hijos procesados (índice j) y luego sobre la estructura del árbol.

1. Inducción sobre la fusión de hijos en u

Caso base: Con $j = 0$, el subárbol parcial T_u^0 consiste únicamente en u; por lo tanto,

$$dp_0[u][1] = 0,$$

lo cual es correcto.

Paso inductivo: Supongamos que para cierto $j - 1$ la tabla $dp_{j-1}[u][\cdot]$ calcula correctamente el costo mínimo para cada tamaño i en el subárbol T_u^{j-1}. Sea v_j el j-ésimo hijo de u. Al incorporar el subárbol T_{v_j}, se tienen dos opciones:

1. **No incluir T_{v_j}:** Se corta la arista (u, v_j) y se conserva el estado anterior, con un costo adicional de 1:
$$dp_j[u][i] \leq dp_{j-1}[u][i] + 1.$$
2. **Fusionar T_{v_j}:** Se toma una componente en T_u^{j-1} de tamaño a (con costo $dp_{j-1}[u][a]$) y una componente en T_{v_j} de tamaño b (con costo $dp[v_j][b]$); al fusionarlas se obtiene una componente de tamaño $a + b$ con costo:
$$dp_j[u][a + b] \leq dp_{j-1}[u][a] + dp[v_j][b].$$

Por lo tanto, para cada i se tiene:

$$dp_j[u][i] = \text{mín}\left\{ dp_{j-1}[u][i] + 1,\ \min_{\substack{a+b=i \\ a,b\geq 1}} \Big(dp_{j-1}[u][a] + dp[v_j][b] \Big) \right\}.$$

Dado que, por hipótesis inductiva, $dp_{j-1}[u][\cdot]$ y $dp[v_j][\cdot]$ son correctos, la fusión produce correctamente $dp_j[u][\cdot]$ en T_u^j.

Al procesar todos los hijos de u (es decir, $j = h$, donde h es el número total de hijos de u), se obtiene:

$$dp_h[u][\cdot] = dp[u][\cdot],$$

lo que demuestra la corrección del estado $dp[u][\cdot]$ en T_u.

2. Inducción en la estructura del árbol

Asumiendo que para cada hijo v de u la tabla $dp[v][\cdot]$ es correcta, el proceso de fusión en u descrito anteriormente garantiza que $dp[u][\cdot]$ se calcula correctamente. Por inducción sobre la estructura del árbol, la DP es correcta para todo el árbol T.

12. Pseudocódigo en $O(N^2)$

A continuación se presenta el pseudocódigo que implementa la solución mediante DFS y fusión de estados. Se aprovecha que, al fusionar el estado del nodo u (de tamaño size$[u]$) con el del hijo v (de tamaño size$[v]$), se realizan size$[u] \times$ size$[v]$ operaciones, y gracias a la propiedad combinatoria (cada par de nodos se fusiona una única vez, correspondiente a su LCA) la complejidad total es $O(N^2)$.

```
1  // Constante que representa un valor "infinito"
2  const INF = infinito;
3
4  // dp[u][i]: Costo minimo para obtener una componente conexa de
       tama o i en T_u que contiene a u.
5  // size[u]: Tamano actual del arreglo dp[u] (inicialmente 1, pues
       solo se cuenta el nodo u).
6
7  procedure DFS(u, parent)
8      // Inicializar dp[u] con el caso base: solo el nodo u.
9      dp[u] := array[1 ... (tamano maximo posible)] of INF;
10     dp[u][1] := 0;
11     size[u] := 1;
12
13     // Procesar cada hijo v de u (evitando regresar al padre)
14     for each v in vecinos(u) do
15         if v == parent then continue;
16         DFS(v, u);
17
18         // Preparar un nuevo arreglo para fusionar dp[u] y dp[v]
19         newSize := size[u] + size[v];
20         newDP := array[1 ... newSize] of INF;
21
22         // Opcion 1: No fusionar v (cortar la arista (u,v))
```

```
23          for i from 1 to size[u] do
24              newDP[i] := min(newDP[i], dp[u][i] + 1);
25          end for;
26
27          // Opcion 2: Fusionar el subarbol de v sin cortar (usar dp[v
                ])
28          for i from 1 to size[u] do
29              for j from 1 to size[v] do
30                  newDP[i + j] := min(newDP[i + j], dp[u][i] + dp[v][j
                        ]);
31              end for;
32          end for;
33
34          // Actualizar dp[u] y su tamano
35          size[u] := newSize;
36          dp[u] := newDP;
37      end for;
38  end procedure;
39
40  // Funcion principal: se asume que el arbol tiene N nodos.
41  procedure Main()
42      // Se asume que el arbol esta almacenado en una lista de
            adyacencia: G.
43      // Se elige un nodo arbitrario como raiz, por ejemplo, 1.
44      DFS(1, NIL);
45
46      // Actualizar la respuesta global para cada tamano K.
47      // La solucion final es el minimo dp[u][K] entre todos los nodos
            u.
48      answer := array[1 ... N] of INF;
49      for each nodo u de 1 a N do
50          for k from 1 to size[u] do
51              answer[k] := min(answer[k], dp[u][k]);
52          end for;
53      end for;
54
55      // Imprimir o retornar la respuesta para cada K = 1, 2,    , N.
56  end procedure;
```

Listing 1: Pseudocódigo en $O(N^2)$