Caso 2: Optimización de Redes de Computadores

Maria Paula Rodriguez Ruiz

March 19, 2025

Previo a la solucion del caso es preciso recordar de forma detallada en que consisten los 3 algoritmos a utilizar:

- Algoritmo de Ford–Fulkerson para el Cálculo de Flujo Máximo.
- Algoritmo de Cancelación de Ciclos para Flujo de Costo Mínimo.
- Algoritmo de Dijkstra para el Cálculo de Caminos Mínimos.

1 Algoritmo de Ford–Fulkerson para el Cálculo de Flujo Máximo

El **algoritmo de Ford–Fulkerson** es un método utilizado para determinar el **flujo máximo** que puede enviarse a través de una red de transporte de capacidad limitada. Se aplica en un grafo dirigido $G = (V, E)$ donde cada arista (u, v) tiene una capacidad $c(u, v)$ y un flujo $f(u, v)$. El objetivo del algoritmo es encontrar la cantidad máxima de flujo que puede enviarse desde una **fuente** s hasta un **sumidero** t respetando las restricciones de capacidad y conservación del flujo.

1.1 Definición del Problema

El problema del flujo máximo se expresa de la siguiente manera:

$$\max \sum_{(s,v)\in E} f(s, v) \tag{1}$$

Sujeto a:

(a) **Restricción de capacidad:** Para cada arista $(u, v) \in E$, el flujo no puede exceder la capacidad de la arista:

$$0 \leq f(u, v) \leq c(u, v).$$

(b) **Conservación del flujo:** Para cada nodo $v \in V \setminus \{s, t\}$, la cantidad de flujo que entra en el nodo debe ser igual a la que sale:

$$\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w).$$

1.2 Pasos del Algoritmo

El algoritmo de Ford–Fulkerson se basa en la idea de encontrar **caminos aumentantes** en un **grafo residual** y aumentar el flujo en dichos caminos hasta que no se puedan encontrar más. Se ejecuta de la siguiente manera:

1. **Inicialización:** Se establece inicialmente $f(u, v) = 0$ para todas las aristas $(u, v) \in E$.

2. **Construcción del grafo residual:** Se define la capacidad residual de cada arista como:

$$c_f(u, v) = c(u, v) - f(u, v).$$

Se construye el **grafo residual** con todas las aristas (u, v) que tengan $c_f(u, v) > 0$.

3. **Búsqueda de un camino aumentante:** Se busca un camino desde la fuente s hasta el sumidero t en el grafo residual. Este camino se puede encontrar con algoritmos como **BFS (búsqueda en anchura)** o **DFS (búsqueda en profundidad)**.

4. **Cálculo del cuello de botella:** Se determina la cantidad máxima de flujo que se puede enviar a través del camino aumentante encontrado:

$$\Delta = \min_{(u,v)\in P} c_f(u, v).$$

5. **Ajuste del flujo:** Se actualiza el flujo en el camino aumentante encontrado:

$$f(u, v) \leftarrow f(u, v) + \Delta, \quad \forall (u, v) \in P.$$

Además, se ajusta el grafo residual disminuyendo la capacidad en la dirección del flujo y aumentando la capacidad en la dirección opuesta:

$$c_f(u, v) \leftarrow c_f(u, v) - \Delta, \quad c_f(v, u) \leftarrow c_f(v, u) + \Delta.$$

6. **Repetición:** Se repiten los pasos 3 a 5 hasta que no se pueda encontrar un nuevo camino aumentante en el grafo residual.

7. **Terminación:** Cuando no se encuentren más caminos aumentantes, el valor del flujo total es el flujo máximo que puede enviarse desde s hasta t.

Observaciones:

- La implementación del algoritmo varía dependiendo del método de búsqueda de caminos aumentantes. Una versión eficiente es el **algoritmo de Edmonds-Karp**, que usa **BFS** y tiene una complejidad de $O(VE^2)$.
- En nuestra implementación, utilizamos una búsqueda en profundidad (**DFS**) que ordena los vecinos disponibles en orden ascendente según la capacidad residual, priorizando los arcos con menor capacidad disponible.

2 Algoritmo de Cancelación de Ciclos para Flujo de Costo Mínimo

El **algoritmo de cancelación de ciclos** es un método utilizado para encontrar un flujo factible en una red de transporte que no solo cumpla con las restricciones de capacidad y conservación del flujo, sino que además minimice un **costo asociado** a las aristas, como el **retardo de transmisión**. Se basa en la detección y eliminación de **ciclos negativos** en el **grafo residual**.

2.1 Definición del Problema

Dado un grafo dirigido $G = (V, E)$ donde cada arista $(u, v) \in E$ tiene:

- Una capacidad $c(u, v)$ que limita el flujo máximo permitido.
- Un costo o retardo $d(u, v)$ asociado a la transmisión.

El objetivo es encontrar un flujo $f(u, v)$ que minimice el costo total:

$$\min \sum_{(u,v)\in E} d(u, v) \cdot f(u, v) \tag{2}$$

Sujeto a:

(a) **Restricción de capacidad:**

$$0 \leq f(u, v) \leq c(u, v), \quad \forall (u, v) \in E.$$

(b) **Conservación del flujo:** Para cada nodo intermedio $v \in V \setminus \{s, t\}$, la suma de flujos entrantes debe ser igual a la de flujos salientes:

$$\sum_{(u,v)\in E} f(u, v) = \sum_{(v,w)\in E} f(v, w).$$

(c) **Demanda total:** Se requiere que el flujo total enviado desde la fuente s hacia el sumidero t sea una cantidad F dada:

$$\sum_{(s,v)\in E} f(s, v) = F, \qquad \sum_{(u,t)\in E} f(u, t) = F.$$

2.2 Pasos del Algoritmo

El método de cancelación de ciclos se basa en la detección y eliminación de ciclos negativos en el grafo residual. Sus pasos son los siguientes:

1. **Construcción de un flujo inicial factible:** Se obtiene un flujo inicial que satisface la demanda F, sin considerar el costo.
2. **Construcción del grafo residual:** Se define la capacidad residual de cada arista:

   $$c_f(u, v) = c(u, v) - f(u, v).$$

   Además, se agregan aristas inversas (v, u) con capacidad $f(u, v)$ y costo negativo $-d(u, v)$, permitiendo reducir el flujo en esas direcciones si es necesario.
3. **Búsqueda de ciclos negativos:** Se utiliza el **algoritmo de Bellman-Ford** para detectar ciclos con costos negativos en el grafo residual.
4. **Cancelación del ciclo negativo:** Si se encuentra un ciclo negativo C, se determina el **flujo máximo** que puede enviarse a través de él:

   $$\Delta = \min_{(u,v)\in C} c_f(u, v).$$

   Luego, se ajusta el flujo en las aristas del ciclo restando Δ en la dirección original y sumándolo en la inversa.
5. **Repetición:** Se repiten los pasos 3 y 4 hasta que no existan más ciclos negativos en el grafo residual.
6. **Terminación:** Cuando no se puedan cancelar más ciclos negativos, el flujo encontrado es el de costo mínimo.

Observaciones:

- El algoritmo garantiza la obtención de un flujo factible y óptimo en términos de costo total (o retardo total en nuestro caso).
- Su implementación eficiente depende de la detección rápida de ciclos negativos. El método más común es Bellman-Ford, pero existen variaciones como **Cycle-Canceling con Algoritmo de Edmonds-Karp**.

3 Algoritmo de Dijkstra para el Cálculo de Caminos Mínimos

El **algoritmo de Dijkstra** es un método utilizado para encontrar el camino de **menor costo** (o, **menor retardo**) entre un nodo fuente s y cualquier otro nodo t en un **grafo ponderado dirigido**. Se basa en la exploración iterativa de los nodos, expandiendo siempre el nodo con menor costo acumulado.

3.1 Definición del Problema

Sea $G = (V, E)$ un grafo dirigido donde cada arista $(u, v) \in E$ tiene un peso $d(u, v)$ que representa el **retardo de transmisión** entre los nodos. El problema consiste en encontrar el camino P desde s hasta t tal que se minimice:

$$D(P) = \sum_{(i,j)\in P} d(i, j).$$

O expresado en su formulación matemática:

$$\min_{P\in\mathcal{P}(s,t)} D(P).$$

3.2 Pasos del Algoritmo

El algoritmo de Dijkstra se basa en una estrategia **voraz** (greedy), expandiendo en cada paso el nodo con la menor distancia acumulada. Los pasos son los siguientes:

1. **Inicialización:**
   - Se establece $d(s) = 0$ para la fuente.
   - Se asigna $d(v) = \infty$ para todos los demás nodos $v \in V \setminus \{s\}$.
   - Se utiliza una estructura de datos de prioridad (**cola de prioridad**) para almacenar los nodos a explorar.
2. **Selección del nodo con menor costo acumulado:** Se extrae el nodo u con la menor distancia tentativa $d(u)$ de la cola de prioridad.
3. **Relajación de vecinos:** Para cada nodo v adyacente a u, se actualiza su distancia tentativa si se encuentra un camino más corto:

   $$d(v) = \min\big(d(v),\ d(u) + d(u, v)\big).$$

   Si la distancia se actualiza, v se inserta nuevamente en la cola de prioridad.
4. **Marcado de nodo explorado:** Una vez procesado, el nodo u se considera "visitado" y no se vuelve a evaluar.

5. **Repetición:** Se repiten los pasos 2 a 4 hasta que se extraiga el nodo destino t o la cola de prioridad esté vacía.

6. **Terminación:** El valor final $d(t)$ representa el menor retardo desde s hasta t, y el camino se reconstruye a partir de los nodos previos almacenados.

Observaciones:

- La complejidad del algoritmo es $O(V^2)$ en su versión básica, pero usando colas de prioridad con montículos de Fibonacci se reduce a $O(E + V \log V)$.
- En nuestra implementación, se utiliza Dijkstra para calcular caminos de menor retardo en la red.

4 Aplicación de los algoritmos al Caso y sus problemas especificos

Dada la red con 50 nodos, disponemos de:

- Una matriz de capacidad c_{ij} de dimensión 50×50, donde $c_{ij} = 0$ indica que no existe conexión entre el nodo i y el nodo j.
- Una matriz de retardo d_{ij} de dimensión 50×50, donde $d_{ij} = 999$ indica que la conexión es inválida.
- Un vector de costos de nodos $\{p_i\}_{i=1}^{50}$.

A continuación se presenta la aplicación del modelo matemático a cada una de las preguntas planteadas. En cada caso, la formulación clásica se adapta a la solución implementada (que extrae rutas representativas y calcula el retardo total acumulado sumando los retardos de cada arista).

Pregunta 1: Flujo de Red a Mínimo Retardo

Enunciado: Se desea enviar por la red un tráfico de 90 kb/s a mínimo retardo, desde el nodo 1 (fuente) hasta los nodos 49 y 50 (destinos). Se requiere determinar el flujo que minimice el retardo y reportar la secuencia de nodos (ruta representativa) para cada destino.

Formulación: En una formulación clásica, se plantea:

$$\min \sum_{(i,j)\in E} d(i,j)\, f(i,j),$$

sujeto a:

$$0 \le f(i,j) \le c(i,j), \quad \forall (i,j) \in E,$$

$$\sum_{(1,j)\in E} f(1,j) = 90,$$

$$\sum_{(j,49)\in E} f(j,49) + \sum_{(j,50)\in E} f(j,50) = 90,$$

$$\sum_{(i,j)\in E} f(i,j) - \sum_{(j,i)\in E} f(j,i) = 0, \quad \forall i \in V \setminus \{1, 49, 50\}.$$

En la solución se extraen rutas representativas a partir del flujo obtenido y se calcula el retardo acumulado en cada ruta como:

$$D(P) = \sum_{(i,j)\in P} d(i,j).$$

Ejemplo: Suponga que, tras aplicar el **algoritmo de cancelación de ciclos**, se obtienen los siguientes resultados:

- **Ruta para el nodo 49:** Se extrae la ruta representativa $[1, 4, 8, 25, 42, 49]$. Los retardos en cada arista son:

  $$d(1,4) = 10\,\text{ms}, \quad d(4,8) = 12\,\text{ms}, \quad d(8,25) = 15\,\text{ms}, \quad d(25,42) = 20\,\text{ms}, \quad d(42,49) = 18\,\text{ms}.$$

  Por lo tanto, el retardo total acumulado para esta ruta es:

  $$D(P_{49}) = 10 + 12 + 15 + 20 + 18 = 75\,\text{ms}.$$

  Además, se envía un flujo de 52 kb/s a través de esta ruta.

- **Ruta para el nodo 50:** Se extrae la ruta representativa $[1, 4, 8, 25, 42, 48, 50]$. Los retardos en cada arista son:

  $$d(1,4) = 10\,\text{ms}, \quad d(4,8) = 12\,\text{ms}, \quad d(8,25) = 15\,\text{ms}, \quad d(25,42) = 20\,\text{ms}$$

  $$d(42,48) = 30\,\text{ms}, \quad d(48,50) = 28\,\text{ms}.$$

  Por lo tanto, el retardo total acumulado en esta ruta es:

  $$D(P_{50}) = 10 + 12 + 15 + 20 + 30 + 28 = 115\,\text{ms}.$$

  En esta ruta se envía un flujo de 38 kb/s.

En total, el flujo enviado es:

$$52\,\text{kb/s} + 38\,\text{kb/s} = 90\,\text{kb/s},$$

y el retardo total combinado es:

$$75\,\text{ms} + 115\,\text{ms} = 190\,\text{ms}.$$

Pregunta 2: Flujo Máximo y Retardo Total

Enunciado: Determinar el flujo máximo que soporta la red entre el nodo 1 (fuente) y los nodos 49 y 50 (destinos), y calcular el retardo total en la red cuando se envía ese flujo máximo.

Formulación: La formulación clásica para el flujo máximo es:

$$\max \sum_{(1,j)\in E} f(1,j),$$

sujeto a:

$$0 \le f(i,j) \le c(i,j), \quad \forall (i,j) \in E,$$

$$\sum_{(i,j)\in E} f(i,j) - \sum_{(j,i)\in E} f(j,i) = 0, \quad \forall i \in V \setminus \{1, 49, 50\}.$$

En nuestra implementación se utiliza el método de Ford–Fulkerson para obtener el flujo máximo. A partir del flujo residual se extraen rutas representativas para cada destino, y se calcula el retardo total acumulado en cada ruta mediante:

$$D(P) = \sum_{(i,j)\in P} d(i,j).$$

Ejemplo: Suponga que, aplicando el **algoritmo Ford–Fulkerson**, se obtiene un flujo máximo de 91 kb/s, que se distribuye de la siguiente manera:

- **Ruta para el nodo 49:** Se extrae la ruta representativa $[1, 2, 6, 8, 9, 25, 38, 40, 49]$. Los retardos en cada arista son:

$$d(1,2) = 12\,\text{ms}, \quad d(2,6) = 15\,\text{ms}, \quad d(6,8) = 10\,\text{ms}, \quad d(8,9) = 20\,\text{ms}, \quad d(9,25) = 18\,\text{ms},$$

$$d(25,38) = 25\,\text{ms}, \quad d(38,40) = 30\,\text{ms}, \quad d(40,49) = 33\,\text{ms},$$

  de donde se tiene:

$$D(P_{49}) = 12 + 15 + 10 + 20 + 18 + 25 + 30 + 33 = 163\,\text{ms}.$$

  Además, se envía un flujo de 52 kb/s a través de esta ruta.

- **Ruta para el nodo 50:** Se extrae la ruta representativa $[1, 2, 6, 8, 9, 25, 38, 41, 43, 48, 50]$ con los siguientes retardos:

$$d(1,2) = 12\,\text{ms}, \quad d(2,6) = 15\,\text{ms}, \quad d(6,8) = 10\,\text{ms}, \quad d(8,9) = 20\,\text{ms}, \quad d(9,25) = 18\,\text{ms},$$

$$d(25,38) = 25\,\text{ms}, \quad d(38,41) = 32\,\text{ms}, \quad d(41,43) = 28\,\text{ms}, \quad d(43,48) = 30\,\text{ms}, \quad d(48,50) = 35\,\text{ms},$$

  lo que resulta en:

$$D(P_{50}) = 12 + 15 + 10 + 20 + 18 + 25 + 32 + 28 + 30 + 35 = 225\,\text{ms}.$$

  En esta ruta se envía un flujo de 39 kb/s.

Así, el flujo máximo total enviado es:

$$52\,\text{kb/s} + 39\,\text{kb/s} = 91\,\text{kb/s},$$

y el retardo total combinado (sumando los retardos de ambas rutas) es:

$$163\,\text{ms} + 225\,\text{ms} = 388\,\text{ms}.$$

Pregunta 3: Camino de Retardo Mínimo (Nodo 49)

Enunciado: Determinar una única ruta de mínimo retardo entre el nodo 1 (fuente) y el nodo 49 (destino).

Formulación: Se resuelve el problema:

$$\min_{P \in \mathcal{P}(1,49)} \sum_{(i,j) \in P} d(i,j),$$

empleando el algoritmo de Dijkstra, en el que cada arista (i, j) tiene peso $d(i, j)$.

Ejemplo: Suponga que el **algoritmo (Dijkstra utilizando** $w(i,j) = d(i,j)$ encuentra la ruta:

$$[1, 14, 25, 42, 49],$$

y que los retardos son:

$$d(1,14) = 20\,\text{ms}, \quad d(14,25) = 18\,\text{ms}, \quad d(25,42) = 22\,\text{ms}, \quad d(42,49) = 16\,\text{ms}.$$

Entonces el retardo total es:

$$D(P) = 20 + 18 + 22 + 16 = 76\,\text{ms}.$$

Pregunta 4: Camino de Retardo Mínimo (Nodo 50)

Enunciado: Determinar una única ruta de mínimo retardo entre el nodo 1 (fuente) y el nodo 50 (destino).

Formulación: Se plantea:

$$\min_{P \in \mathcal{P}(1,50)} \sum_{(i,j) \in P} d(i,j),$$

empleando el algoritmo de Dijkstra, en el que cada arista (i,j) tiene peso $d(i,j)$.

Ejemplo: Suponga que el **algoritmo Dijkstra utilizando** $w(i,j) = d(i,j)$ encuentra la ruta:

$$[1, 14, 29, 43, 50],$$

con retardos:

$$d(1,14) = 20\,\text{ms}, \quad d(14,29) = 18\,\text{ms}, \quad d(29,43) = 22\,\text{ms}, \quad d(43,50) = 21\,\text{ms},$$

el retardo total acumulado es:

$$D(P) = 20 + 18 + 22 + 21 = 81\,\text{ms}.$$

Pregunta 5: Camino de Costo Mínimo (Nodo 49)

Enunciado: Se requiere decidir una única ruta para el envío de información entre el nodo 1 (fuente) y el nodo 49 (destino) que minimice el costo de procesamiento, ignorando el retardo en la función objetivo.

Formulación: Se define el costo de cada arista (i,j) como:

$$w(i,j) = p_j,$$

donde p_j es el costo de procesamiento del nodo j. Así, la formulación es:

$$\min_{P \in \mathcal{P}(1,49)} \sum_{(i,j) \in P} p_j,$$

sujeto a:

$$0 \leq f(i,j) \leq c(i,j), \quad \forall (i,j) \in E,$$

$$\sum_{(1,j) \in E} f(1,j) = F, \quad \text{(con } F \text{ igual a la cantidad de flujo a enviar)},$$

$$\sum_{(i,j) \in E} f(i,j) - \sum_{(j,i) \in E} f(j,i) = 0, \quad \forall i \in V \setminus \{1, 49\}.$$

Además, se calcula el retardo total acumulado a lo largo de la ruta:

$$D(P) = \sum_{(i,j) \in P} d(i,j).$$

Ejemplo: Suponga que se tienen los siguientes costos de procesamiento:

$$p_6 = 34, \quad p_{24} = 45, \quad p_{42} = 50, \quad p_{49} = 40,$$

y que los retardos en las aristas son:

$$d(1,6) = 10\,\text{ms}, \quad d(6,24) = 20\,\text{ms}, \quad d(24,42) = 15\,\text{ms}, \quad d(42,49) = 25\,\text{ms}.$$

Si el **algoritmo Dijkstra utilizando** $w(i,j) = p_j$**)** encuentra la ruta:

$$[1, 6, 24, 42, 49],$$

entonces el costo total de la ruta es:

$$34 + 45 + 50 + 40 = 169,$$

y el retardo total acumulado es:

$$10 + 20 + 15 + 25 = 70\,\text{ms}.$$

Pregunta 6: Camino de Costo Mínimo (Nodo 50)

Enunciado: De forma análoga a la Pregunta 5, se requiere determinar una única ruta para el envío de información entre el nodo 1 (fuente) y el nodo 50 (destino) que minimice el costo de procesamiento.

Formulación: Se define el costo de cada arista (i,j) como:

$$w(i,j) = p_j,$$

donde p_j es el costo de procesamiento del nodo j. Así, la formulación es:

$$\min_{P \in \mathcal{P}(1,50)} \sum_{(i,j)\in P} p_j,$$

sujeto a:

$$0 \le f(i,j) \le c(i,j), \quad \forall (i,j) \in E,$$

$$\sum_{(1,j)\in E} f(1,j) = F, \quad (\text{con } F \text{ igual a la cantidad de flujo a enviar}),$$

$$\sum_{(i,j)\in E} f(i,j) - \sum_{(j,i)\in E} f(j,i) = 0, \quad \forall i \in V \setminus \{1, 50\}.$$

Además, se calcula el retardo total acumulado a lo largo de la ruta:

$$D(P) = \sum_{(i,j)\in P} d(i,j).$$

Ejemplo: Con los siguientes costos:

$$p_6 = 34, \quad p_{23} = 39, \quad p_{41} = 66, \quad p_{50} = 64,$$

y con retardos:

$$d(1,6) = 10\,\text{ms}, \quad d(6,23) = 12\,\text{ms}, \quad d(23,41) = 18\,\text{ms}, \quad d(41,50) = 20\,\text{ms},$$

Si el **algoritmo Dijkstra utilizando** $w(i,j) = p_j$**)** encuentra la ruta:

$$[1, 6, 23, 41, 50],$$

entonces el costo total es:

$$34 + 39 + 66 + 64 = 203,$$

y el retardo total acumulado es:

$$10 + 12 + 18 + 20 = 60\,\text{ms}.$$

Pregunta 7: Caminos de Retardo Mínimo con Restricción $d > 40$

Enunciado: Para garantizar una transmisión estable, se requiere que el retardo en cada arista sea mayor a 40 ms. Se debe hallar el camino de mínimo retardo entre el nodo 1 y el destino (49 o 50) considerando únicamente aristas con $d(i,j) > 40$.

Formulación: Se filtra el grafo para conservar solo las aristas que cumplen $d(i,j) > 40$ y se plantea:

$$\min_{\substack{P \in \mathcal{P}(1,t) \\ d(i,j)>40\ \forall (i,j)\in P}} \sum_{(i,j)\in P} d(i,j),$$

donde t es el destino a evaluar.

Ejemplo: Para el destino 49, suponga que el **algoritmo (Dijkstra utilizando** $w(i,j) = d(i,j) > 40$ obtiene la ruta:

$$[1, 12, 25, 31, 40, 49],$$

con los siguientes retardos:

$$d(1,12) = 42\,\text{ms}, \quad d(12,25) = 44\,\text{ms}, \quad d(25,31) = 42\,\text{ms}, \quad d(31,40) = 44\,\text{ms}, \quad d(40,49) = 44\,\text{ms}.$$

Así, el retardo total es:

$$42 + 44 + 42 + 44 + 44 = 216\,\text{ms}.$$

Para el destino 50, suponga que el **algoritmo Dijkstra utilizando** $w(i,j) = d(i,j) > 40$ obtiene la ruta:

$$[1, 12, 25, 36, 39, 43, 48, 50],$$

con retardos:

$$d(1,12) = 42\,\text{ms}, \quad d(12,25) = 44\,\text{ms}, \quad d(25,36) = 43\,\text{ms}, \quad d(36,39) = 45\,\text{ms}$$

$$d(39,43) = 44\,\text{ms}, \quad d(43,48) = 43\,\text{ms}, \quad d(48,50) = 45\,\text{ms}.$$

El retardo total acumulado es:

$$42 + 44 + 43 + 45 + 44 + 43 + 45 = 306\,\text{ms}.$$

Pregunta 8: Ruta Óptima Combinada (Suma Ponderada)

Enunciado: Se desea encontrar una única ruta que minimice simultáneamente el costo de procesamiento y el retardo total. Para ello, se utiliza el método de suma ponderada, en el cual se asignan parámetros α y β que determinan la importancia relativa de cada criterio. Al fijar $\alpha = \beta = 1$, se otorga un peso equilibrado (50/50) a ambos criterios. Si se aumenta el valor de α se prioriza el costo de procesamiento, mientras que si se aumenta β se prioriza el retardo.

Formulación: Se define la función de costo para cada arista (i,j) como:

$$w(i,j) = \alpha\, p_j + \beta\, d(i,j),$$

y se plantea el siguiente problema:

$$\min_{P\in\mathcal{P}(1,t)} \sum_{(i,j)\in P} \left[\alpha\, p_j + \beta\, d(i,j)\right],$$

donde el destino t se evalúa para $t = 49$ y para $t = 50$. Con $\alpha = \beta = 1$:

- Para el destino $t = 49$, se fija $\alpha = 1$ y $\beta = 1$. La función objetivo es:

$$\min_{P \in \mathcal{P}(1,49)} \sum_{(i,j) \in P} [p_j + d(i,j)] .$$

- Para el destino $t = 50$, se fija $\alpha = 1$ y $\beta = 14$. La función objetivo es:

$$\min_{P \in \mathcal{P}(1,50)} \sum_{(i,j) \in P} [p_j + \ d(i,j)] .$$

Ejemplo: Para el destino 49: Suponga que se tienen los siguientes valores para los nodos involucrados en la ruta:

$$p_6 = 34, \quad p_{24} = 45, \quad p_{42} = 50, \quad p_{49} = 40,$$

y los retardos en las aristas son:

$$d(1,6) = 10\,\text{ms}, \quad d(6,24) = 20\,\text{ms}, \quad d(24,42) = 15\,\text{ms}, \quad d(42,49) = 25\,\text{ms}.$$

Aplicando **algoritmo Dijkstra utilizando** $w(i,j) = \alpha\, p_j + \beta\, d(i,j)$, se obtiene la ruta:

$$[1, 6, 24, 42, 49].$$

El costo total de procesamiento es:

$$34 + 45 + 50 + 40 = 169,$$

y el retardo total acumulado es:

$$10 + 20 + 15 + 25 = 70\,\text{ms}.$$

Por lo tanto, el valor objetivo para la ruta hacia el nodo 49 es:

$$169 + 70 = 239.$$

Para el destino 50: Suponga que se tienen los siguientes valores:

$$p_6 = 34, \quad p_{23} = 39, \quad p_{41} = 66, \quad p_{50} = 64,$$

y los retardos correspondientes:

$$d(1,6) = 10\,\text{ms}, \quad d(6,23) = 12\,\text{ms}, \quad d(23,41) = 18\,\text{ms}, \quad d(41,50) = 20\,\text{ms}.$$

Aplicando nuevamente **algoritmo Dijkstra utilizando** $w(i,j) = \alpha\, p_j + \beta\, d(i,j)$, se obtiene la ruta:

$$[1, 6, 23, 41, 50].$$

El costo total de la ruta es:

$$34 + 39 + 66 + 64 = 203,$$

y el retardo total acumulado es:

$$10 + 12 + 18 + 20 = 60\,\text{ms}.$$

El valor objetivo para la ruta hacia el nodo 50 es:

$$203 + 60 = 263.$$

Comparando ambos valores objetivo, se elige la ruta con menor valor. En este ejemplo, la ruta óptima es la que conduce al nodo 49, pues su valor objetivo es 239, mientras que la ruta hacia el nodo 50 tiene un valor de 263.

5 Implementación

En esta sección, presentamos la implementación práctica de la solución a cada una de las preguntas formuladas anteriormente. Se ha desarrollado un código en **Python**, el cual hace uso de estructuras de datos eficientes y algoritmos de optimización para calcular las rutas y flujos óptimos en la red.

Link al Entorno de Trabajo

El código se encuentra disponible en Google Colab en el siguiente enlace:

https://colab.research.google.com/drive/1N1D3s0gN5ue8kDiR9geHIa6ksbgqTC9J?usp=sharing

Nota: Para ejecutar correctamente el código, es necesario cargar los tres archivos de Excel en la sección de archivos de Colab (`T22430C2.xlsx`, `T22430R2.xlsx`, `T22430Cost.xlsx`), ya que contienen la información de la red.

Explicación Detallada del Código

A continuación se describe, paso a paso, el funcionamiento del código implementado para resolver el caso, el cual se divide en varias etapas fundamentales:

1. **Lectura de los Datos:**
   - Se utilizan las librerías `pandas` y `numpy` para cargar los datos de tres archivos Excel:
     - `T22430C2.xlsx`: Contiene la matriz de capacidad de la red de dimensiones 50×50. Cada celda c_{ij} indica la capacidad (en kb/s) de la conexión entre el nodo i y j. Si $c_{ij} = 0$, no existe conexión.
     - `T22430R2.xlsx`: Contiene la matriz de retardos (en ms) de dimensiones 50×50. Se utiliza el valor 999 para indicar conexiones inválidas.
     - `T22430Cost.xlsx`: Contiene un vector de costos de procesamiento para los 50 nodos.
   - Estos datos se convierten en matrices o vectores para su uso posterior en la construcción del grafo.
2. **Construcción del Grafo:**
   - Se define un diccionario `graph` cuyos índices son los números de nodo (de 1 a 50).
   - Para cada par de nodos i y j, se extraen la capacidad c y el retardo d correspondientes.
   - Si $c > 0$ y $d < 999$ (es decir, existe conexión válida), se agrega a la lista de adyacencia del nodo $i + 1$ la tupla $(j + 1, d, c)$.
   - De esta forma, cada nodo cuenta con un listado de nodos a los que se conecta, junto con el retardo y la capacidad de cada conexión.
3. **Funciones Auxiliares:**
   - **extract_path_from_flow:** Esta función recibe un diccionario de flujos (donde cada clave es una arista (u, v) y su valor es el flujo asignado) y utiliza una búsqueda en profundidad (DFS) para reconstruir una ruta representativa desde un nodo fuente a un nodo destino. Se construye una lista de adyacencia a partir de las aristas con flujo positivo y se recorre hasta llegar al destino.
4. **Implementación de los Algoritmos de Optimización:** El código implementa tres algoritmos principales:
   a. **Ford–Fulkerson:**

- Se inicializa un diccionario `residual` para almacenar las capacidades residuales de cada arista.
- Se utiliza una función recursiva `find_path_dfs` que, mediante DFS, busca un camino aumentante desde la fuente s hasta el sumidero t en el grafo residual.
- Se determina el "cuello de botella" Δ (el mínimo de la capacidad residual a lo largo del camino) y se actualizan las capacidades residuales en cada arista del camino.
- Este proceso se repite hasta que no se encuentre un camino aumentante, lo que indica que se ha alcanzado el flujo máximo.

b. **Cancelación de Ciclos:**

- Se construye un grafo modificado que incluye un "super-sumidero" al que se conectan los nodos destino (en este caso, 49 y 50) con capacidad infinita (o un valor muy grande).
- Se inicializa un flujo nulo para todas las aristas del grafo modificado.
- Se define una función `build_residual` que genera el grafo residual, calculando para cada arista la capacidad residual y agregando las aristas inversas con costo negativo.
- Se utiliza una búsqueda en anchura (BFS) para encontrar caminos aumentantes y se ajusta el flujo a lo largo del camino, considerando el cuello de botella.
- Se detectan y cancelan ciclos negativos en el grafo residual (usando una variante del algoritmo de Bellman–Ford) para minimizar el costo total del flujo.

c. **Dijkstra:**

- Se implementa el algoritmo de Dijkstra de forma genérica, donde se recibe una función de peso `weight_func` que determina el costo de cada arista.
- Se inicializan las distancias, se utiliza una cola de prioridad para seleccionar el nodo con la menor distancia acumulada y se relajan las aristas de cada nodo.
- Se reconstruye el camino óptimo utilizando la información de predecesores.

5. **Resolución de las Preguntas:** Se definen funciones específicas (`solve_q1()`, `solve_q2()`, `solve_q3()`, `solve_q4()`, `solve_q5()`, `solve_q6()`, `solve_q7()`, `solve_q8()`) para resolver cada pregunta. Cada función:

   1. **Invocación del Algoritmo Correspondiente:** Cada función de resolución comienza llamando al algoritmo diseñado para responder la pregunta:

      - **Invocación del Algoritmo Correspondiente:**

        Para la resolución de las preguntas Q1 y Q2 se invoca la función `cycle_canceling_min_cost_flow()` (o bien `ford_fulkerson_max_flow_special()` en el caso de Q2) para calcular un flujo representativo en la red.
      - Para Q3 y Q4 se llama al algoritmo de Dijkstra con la función de peso $w(i,j) = d(i,j)$.
      - Para Q5 y Q6 se utiliza Dijkstra con la función de peso $w(i,j) = p_j$ para minimizar el costo de procesamiento.
      - Para Q7 se filtra el grafo, conservando únicamente las aristas con $d(i,j) > 40$ ms, y se ejecuta Dijkstra sobre dicho grafo.
      - Para Q8 se define una función de costo combinada $w(i,j) = \alpha\, p_j + \beta\, d(i,j)$ y se aplica Dijkstra para obtener la ruta que minimice la suma ponderada.

   2. **Extracción y Reconstrucción de la Ruta Representativa:** Dependiendo del algoritmo:

      - En Q1 y Q2, luego de calcular el flujo (ya sea mediante cancelación de ciclos o Ford–Fulkerson), se utiliza la función auxiliar `extract_path_from_flow()` para reconstruir una ruta representativa. Esta función utiliza una búsqueda en profundidad (DFS) sobre el diccionario de flujos positivos, construyendo una lista de nodos que conforman la ruta desde la fuente hasta cada destino.

- En Q3, Q4, Q5, Q6 y Q8, el algoritmo de Dijkstra devuelve directamente el camino óptimo (almacenado en la variable `path`) y la distancia acumulada (que representa el retardo o costo total, según corresponda).

3. **Cálculo del Retardo Total Acumulado:** Para cada ruta encontrada se calcula el retardo total acumulado:
   - Se recorre la lista de nodos obtenida (por ejemplo, $[1, 6, 24, 42, 49]$) y para cada par de nodos consecutivos (u, v) se busca en la estructura del grafo la arista correspondiente y se suma el retardo $d(u, v)$.
   - Esta suma se almacena y se utiliza para evaluar el desempeño de la ruta en términos de latencia.
4. **Impresión de Resultados:** Cada función de resolución imprime los resultados de forma clara, incluyendo:
   - La secuencia de nodos de la ruta encontrada.
   - El retardo total acumulado en la ruta.
   - En el caso de Q1 y Q2, también se imprime el flujo entregado a cada destino y la suma total del flujo enviado.
   - En Q5, Q6 y Q8, se muestran tanto el costo total (sumatoria de los costos de los nodos utilizados) como el retardo acumulado, y se calcula el valor objetivo combinado (en Q8).
5. **Ejecución Principal:** La función `main()` llama a todas las funciones de solución en secuencia, mostrando los resultados en la salida estándar.

Esta explicación detalla la estructura y el funcionamiento del código, desde la lectura de datos hasta la ejecución de los algoritmos de optimización, y demuestra cómo se utilizan las rutas y los flujos calculados para responder a cada una de las preguntas del caso.

Ejemplo de Salida Después de la Ejecución

Al ejecutar el código en Colab, se obtiene la siguiente salida:

```
=== Q1: Minimum Delay Flow for 90 kb/s (Multi-Destination) ===
Path from 1 to 49: [1, 4, 8, 25, 42, 49] with delay = 95 ms
Flow delivered to node 49: 52.0 kb/s
Path from 1 to 50: [1, 4, 8, 25, 42, 48, 50] with delay = 138 ms
Flow delivered to node 50: 38.0 kb/s
Total flow delivered = 90.0 kb\s
Combined delay of both paths = 233 ms

=== Q2: Maximum Flow with Representative Path Delays ===
Maximum flow delivered to the super-sink = 91.0 kb/s
Path from 1 to 49: [1, 2, 6, 8, 9, 25, 38, 40, 49] with delay = 243 ms
Flow delivered to node 49: 52.0 kb/s
Path from 1 to 50: [1, 2, 6, 8, 9, 25, 38, 41, 43, 48, 50] with delay = 318 ms
Flow delivered to node 50: 39.0 kb/s
Total flow delivered = 91.0 kb/s
Combined delay of both paths = 561 ms

=== Q3: Min-Delay Path from 1 to 49 ===
Min-delay path from 1 to 49: [1, 14, 25, 42, 49]
Total delay = 78.0
```

```
=== Q4: Min-Delay Path from 1 to 50 ===
Min-delay path from 1 to 50: [1, 14, 29, 43, 50]
Total delay = 81.0

=== Q5: Min-Cost (Node Cost) Path from 1 to 49 ===
Min-cost path from 1 to 49: [1, 6, 24, 42, 49]
Total node cost = 151.0
Total delay = 87

=== Q6: Min-Cost (Node Cost) Path from 1 to 50 ===
Min-cost path from 1 to 50: [1, 6, 23, 41, 50]
Total node cost = 203.0
Total delay = 116

=== Q7: Min-Delay Paths with Edge Delay > 40 ===
Min-delay path (delay > 40) from 1 to 49: [1, 12, 25, 31, 40, 49],
total delay = 216 ms
  Edge 1 -> 12: delay = 42 ms
  Edge 12 -> 25: delay = 44 ms
  Edge 25 -> 31: delay = 42 ms
  Edge 31 -> 40: delay = 44 ms
  Edge 40 -> 49: delay = 44 ms

Min-delay path (delay > 40) from 1 to 50: [1, 12, 25, 36, 39, 43, 48, 50],
total delay = 306 ms
  Edge 1 -> 12: delay = 42 ms
  Edge 12 -> 25: delay = 44 ms
  Edge 25 -> 36: delay = 43 ms
  Edge 36 -> 39: delay = 45 ms
  Edge 39 -> 43: delay = 44 ms
  Edge 43 -> 48: delay = 43 ms
  Edge 48 -> 50: delay = 45 ms

=== Q8: Weighted-Sum (Min Cost and Min Delay) Route ===
Weighted-sum route from 1 to 49: [1, 6, 24, 42, 49]
  Sum of node costs = 151
  Sum of delays = 87
  Objective value = 238.0

Weighted-sum route from 1 to 50: [1, 13, 29, 43, 50]
  Sum of node costs = 210
  Sum of delays = 84
  Objective value = 294.0

Best overall route based on weighted sum:
Destination: 49
Route: [1, 6, 24, 42, 49]
Sum of node costs: 151
Sum of delays: 87
Objective value: 238.0
```

Análisis y Conclusiones

A partir de la salida obtenida tras ejecutar el código, se pueden extraer diversas conclusiones que evidencian el comportamiento y la efectividad de los algoritmos implementados:

- **Flujo de Red a Mínimo Retardo (Q1):** Se logró enviar 90 kb/s de flujo desde el nodo 1 a los nodos 49 y 50, distribuyéndose 52 kb/s a la ruta $[1, 4, 8, 25, 42, 49]$ y 38 kb/s a la ruta $[1, 4, 8, 25, 42, 48, 50]$. El retardo acumulado en la ruta hacia el nodo 49 fue de 95 ms y en la ruta hacia el nodo 50 fue de 138 ms, resultando en un retardo combinado de 233 ms y con un flujo combinado de 90kb/s como se requeria. Esto indica que, a pesar de repartir el flujo, el costo en términos de retardo se mantiene relativamente bajo, lo que es crucial para aplicaciones que demandan alta calidad en la transmisión.
- **Flujo Máximo y Retardo Total (Q2):** El algoritmo de Ford–Fulkerson, aplicado para calcular el flujo máximo, determinó un flujo total de 91 kb/s, ligeramente superior al requerido, lo que evidencia que la red puede soportar un poco más de carga. Se extrajeron rutas representativas:
  - Para el nodo 49: $[1, 2, 6, 8, 9, 25, 38, 40, 49]$ con un retardo de 243 ms y 52 kb/s enviados.
  - Para el nodo 50: $[1, 2, 6, 8, 9, 25, 38, 41, 43, 48, 50]$ con un retardo de 318 ms (combinado, el total para ambas rutas fue de 561 ms) y 39 kb/s enviados.

  Estos resultados muestran la capacidad de la red para manejar flujos elevados, aunque con un retardo mayor en comparación con la solución de mínimo retardo, lo que implica la existencia de caminos más largos o con aristas de mayor retardo en el flujo máximo.
- **Caminos de Mínimo Retardo (Q3 y Q4):** El uso del algoritmo de Dijkstra permitió encontrar caminos directos y rápidos.
  - La ruta de mínimo retardo para el nodo 49 fue $[1, 14, 25, 42, 49]$ con un retardo total de 78 ms.
  - Para el nodo 50, la ruta fue $[1, 14, 29, 43, 50]$ con 81 ms de retardo.

  Estos resultados confirman que, cuando se ignoran otros criterios, es posible encontrar rutas que minimizan el retardo de manera efectiva.
- **Caminos de Mínimo Costo (Q5 y Q6):** Al utilizar Dijkstra con la función de costo definida por el costo fijo de procesamiento de cada nodo ($w(i, j) = p_j$), se obtuvieron:
  - Para el nodo 49, la ruta $[1, 6, 24, 42, 49]$ con un costo total de 151 y un retardo acumulado de 87 ms.
  - Para el nodo 50, la ruta $[1, 6, 23, 41, 50]$ con un costo total de 203 y un retardo de 116 ms.

  Esto demuestra que, para minimizar el costo de procesamiento, la red prefiere rutas que pueden implicar un mayor retardo en el caso del nodo 50, lo que resalta la importancia de elegir el criterio adecuado según la aplicación.
- **Restricción de Retardo Mayor a 40 ms (Q7):** Al imponer la restricción $d(i, j) > 40$ ms, se forzó al algoritmo a elegir rutas alternativas.
  - Para el nodo 49 se obtuvo la ruta $[1, 12, 25, 31, 40, 49]$ con un retardo total de 216 ms.
  - Para el nodo 50, la ruta resultante fue $[1, 12, 25, 36, 39, 43, 48, 50]$ con 306 ms de retardo.

  Esto indica que, al exigir una calidad mínima de conexión (retardo mayor a 40 ms), se sacrifica el rendimiento en términos de retardo total, lo cual es relevante en entornos donde dicha restricción es necesaria para la estabilidad de la transmisión.
- **Ruta Óptima Combinada (Q8):** Se utilizó el método de suma ponderada, en el que la función de costo es:

  $$w(i, j) = \alpha\, p_j + \beta\, d(i, j).$$

  Donde para el destino 49 y 50 se fijaron $\alpha = 1$ y $\beta = 1$.

- Para el nodo 49, la ruta obtenida fue $[1, 6, 24, 42, 49]$ con una suma de costos de 151 y un retardo total de 87, resultando en un valor objetivo de $151 + 87 = 238$.
- Para el nodo 50, se obtuvo la ruta $[1, 13, 29, 43, 50]$ con una suma de costos de 210 y un retardo de 84, dando un valor objetivo de $210 + 84 = 294$.

Comparando ambos valores, se elige la ruta para el nodo 49, ya que su valor objetivo (238) es menor que el de la ruta hacia el nodo 50 (294). Esto demuestra la flexibilidad del modelo para equilibrar costo y retardo, pudiéndose ajustar los parámetros α y β según la prioridad requerida.

Conclusiones preliminares: La implementación práctica confirma que:

- La red puede manejar el flujo requerido (90 kb/s) con diferentes distribuciones de flujo y con distintos niveles de retardo, dependiendo del criterio de optimización.
- El algoritmo de Ford–Fulkerson y su variante para flujo de costo mínimo permiten identificar la capacidad máxima de la red y ajustar el flujo para minimizar el retardo total.
- Dijkstra es eficaz para hallar rutas óptimas en términos de retardo y costo, lo que permite obtener soluciones precisas cuando se priorizan criterios específicos.
- Las restricciones adicionales, como la exigencia de que el retardo en cada arista sea mayor a 40 ms, impactan significativamente en las rutas seleccionadas, lo que es relevante en escenarios de alta calidad de servicio.
- El método de suma ponderada permite ajustar de manera flexible la importancia relativa del costo y el retardo, siendo crucial para aplicaciones que requieran un equilibrio entre ambos criterios.

Sin embargo, mediante la comparación entre las distintas preguntas podemos extraer aún más conclusiones:

- **Comparación entre Q1 y Q2:** En la **Pregunta 1**, se implementa una solución que minimiza el retardo total del flujo para enviar 90 kb/s. Se obtuvo que la ruta representativa hacia el nodo 49 presentaba un retardo de 95 ms y la ruta hacia el nodo 50 un retardo de 138 ms, logrando un retardo combinado de 233 ms. Esta solución se centra en la minimización del retardo, lo que da prioridad a la rapidez en la transmisión.

  En cambio, en la **Pregunta 2**, al calcular el flujo máximo mediante Ford–Fulkerson, se consiguió un flujo total de 91 kb/s. Sin embargo, las rutas representativas extraídas para maximizar el flujo muestran retardos considerablemente mayores (243 ms para la ruta hacia el nodo 49 y 318 ms para la ruta hacia el nodo 50, con un retardo combinado de 561 ms). Esto evidencia que, al maximizar el flujo, la red utiliza caminos que, si bien permiten transportar mayor cantidad de información, lo hacen a costa de incrementar el retardo.
- **Comparación entre Q3, Q4, Q5, Q6 y Q8:** Las **Preguntas 3 y 4** se enfocan en obtener las rutas de mínimo retardo utilizando Dijkstra, obteniéndose caminos de 78 ms y 81 ms para los destinos 49 y 50, respectivamente. Por otro lado, en las **Preguntas 5 y 6** se busca minimizar el costo de procesamiento (definido como la suma de los costos de los nodos intermedios), obteniéndose una ruta de menor costo para el nodo 49 ([1, 6, 24, 42, 49]) con un costo total de 151 y retardo de 87 ms, y otra ruta para el nodo 50 ([1, 6, 23, 41, 50]) con un costo total de 203 y retardo de 116 ms.

  Ahora bien, la **Pregunta 8** emplea el método de suma ponderada para balancear simultáneamente costo y retardo. Con los parámetros fijados, se observa que la ruta óptima para el destino 49 coincide con la obtenida en la Pregunta 5 (valor objetivo de 238), lo que indica que, al equilibrar ambos criterios, se selecciona la ruta de menor costo, aunque ésta no sea la de menor retardo (como la obtenida en Q3), por lo que efectivamente se esta equilibrando. Para el destino 50, la solución ponderada no corresponde ni a la ruta de mínimo retardo (Q4) ni a la de mínimo costo (Q6), sino a una ruta que logra un compromiso entre ambos criterios. Ambos, demostrando que el balance logrado es efectivo para identificar la opción globalmente óptima.

- **Conclusión Final:**
  - La red puede manejar el flujo requerido, pero la estrategia de optimización seleccionada afecta notablemente los resultados: minimizar el retardo (Q1) resulta en caminos más rápidos pero con menor flujo, mientras que maximizar el flujo (Q2) incrementa el retardo.
  - El uso de Dijkstra para determinar caminos óptimos en función del retardo (Q3 y Q4) y del costo (Q5 y Q6) permite identificar distintas rutas que satisfacen criterios específicos, evidenciando la diversidad de soluciones posibles en la red.
  - El método de suma ponderada (Q8) resulta una herramienta flexible para equilibrar ambos criterios. En el ejemplo, la ruta óptima combinada seleccionada (para el destino 49) demuestra que, al ajustar los parámetros de ponderación, se puede alcanzar un compromiso que priorice de forma equilibrada tanto el costo de procesamiento como el retardo total, en lugar de optar exclusivamente por uno de los dos.