Escrito por **Adrián Quiroga Linares**.

En este tema, nos adentramos en algunos de los algoritmos fundamentales en teoría de grafos. A continuación se explican los principales conceptos y métodos para manipular y analizar grafos, en particular en contextos donde el grafo es dirigido, ponderado o acíclico.

7.1 Ordenación Topológica Concepto de GDA (Grado Dirigido Acíclico):
Un GDA es un grafo dirigido que no contiene ciclos, es decir, no existen caminos que partan de un vértice y regresen a él mismo. En la matriz de caminos de un GDA, los valores en la diagonal siempre son ceros, indicando la ausencia de ciclos. Estos grafos son útiles para representar estructuras donde ciertos elementos deben preceder a otros, como las expresiones aritméticas o las ordenaciones parciales.

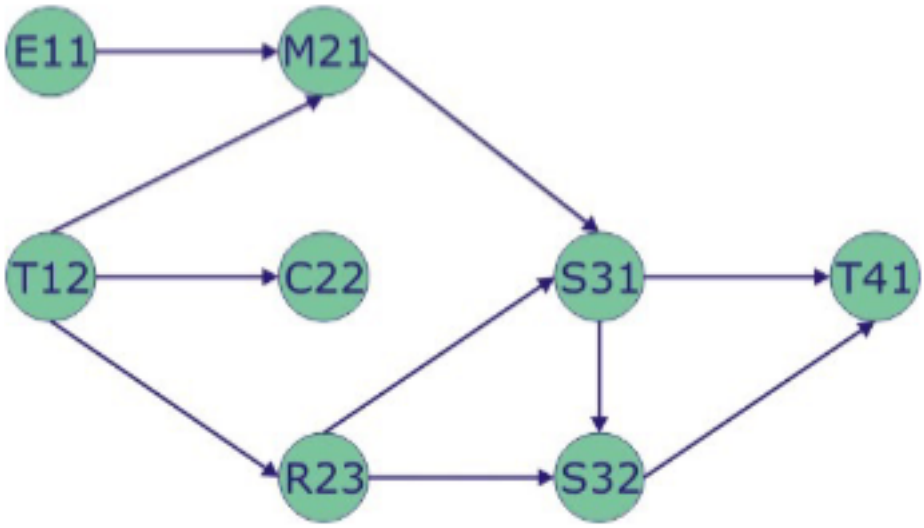


Figure 1: archivos/imagenes/Pasted image 20241110130331.png

Ordenación Parcial:
Se trata de una relación que, además de no ser reflexiva (un elemento no se relaciona consigo mismo), es transitiva: si A se relaciona con B y B con C, entonces A se relaciona con C.

Definición de Ordenación Topológica:
La ordenación topológica es una disposición lineal de los vértices de un grafo dirigido acíclico, de tal forma que si existe un camino de un vértice a otro, el vértice de origen aparece antes que el de destino en el orden.

Algoritmo para la Ordenación Topológica: 1. Se buscan los nodos con **grado entrante cero** y se colocan en una cola. 2. **Extraemos el vértice de la cola** y lo añadimos a la ordenación. 3. **Eliminamos los arcos** que parten del vértice procesado y recalculamos los grados entrantes de los vértices restantes. 4. Repetimos hasta que la cola esté vacía.

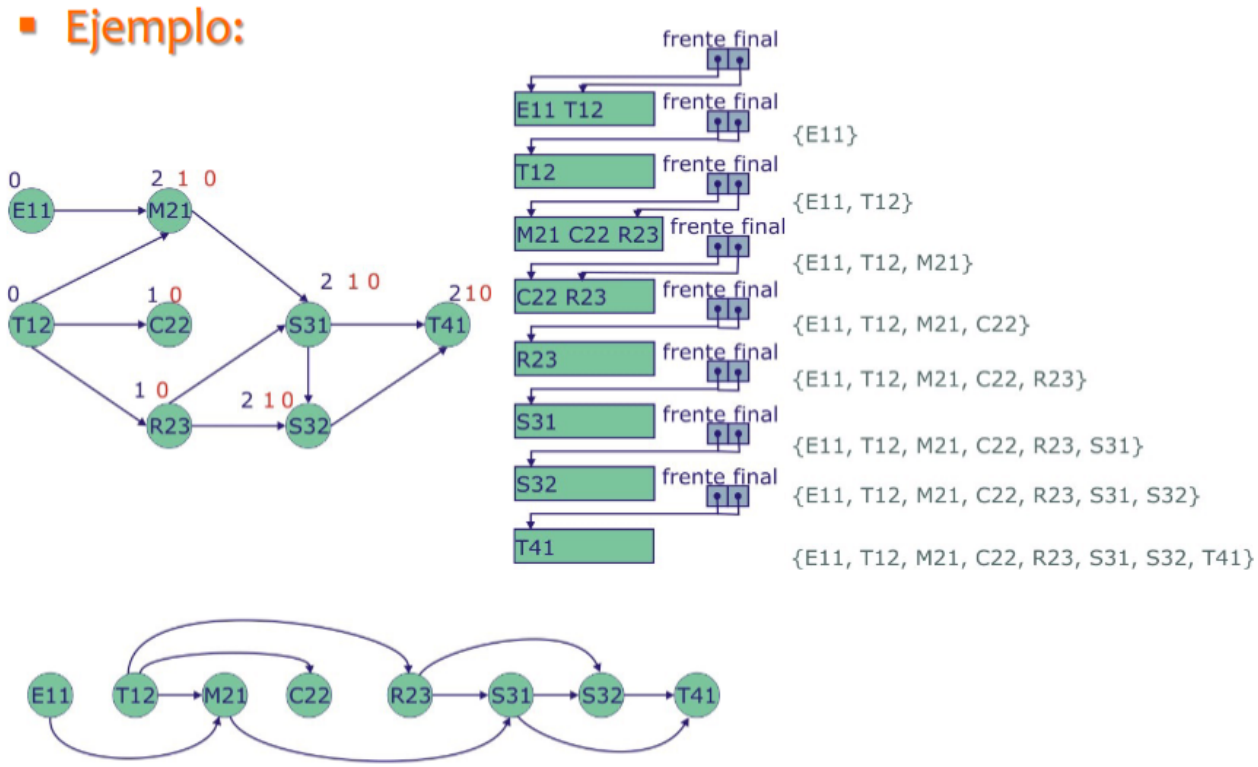


Figure 2: archivos/imagenes/Pasted image 20241110130410.png

7.2 Cálculo de la Matriz de Caminos: Algoritmo de Warshall El **Algoritmo de Warshall** permite calcular la matriz de caminos sin realizar multiplicaciones. Para cada iteración, la matriz de caminos se actualiza basándose en la información de la matriz anterior. La fórmula de actualización es:

$$P_k(i,j) = \min[1, P_{k-1}(i,j) + (P_{k-1}(i,v) \times P_{k-1}(v,j))]$$

Esto significa que el elemento (i,j) en la matriz se establece en 1 si ya había un camino previo o si existe un camino indirecto a través de un vértice intermedio.

7.3 Camino Más Corto entre dos Vértices: Algoritmo de Dijkstra El **Algoritmo de Dijkstra** se usa para encontrar el camino de menor longitud desde un vértice de origen hasta todos los demás vértices en un grafo ponderado. Este es un algoritmo voraz que selecciona, en cada paso, el vértice con el camino mínimo actual y ajusta los caminos de los vértices adyacentes si es necesario. Utiliza: - **C:** Conjunto de vértices candidatos. - **S:** Conjunto de vértices ya procesados. - **D:** Vector de distancias mínimas desde el vértice origen. - **A:** Matriz de pesos del grafo. - **P:** Vector de predecesores inmediatos.

Para realizar el algoritmo, se parte de un vértice origen, y a partir de él se establece en D el peso de los arcos con sus vértices adyacentes, poniendo en P (*en el campo que corresponda a estos vértices*) el número del vértice inicial. Se marca el vértice original como ya estudiado (*se añade a* S), y se selecciona el vértice con el que se estableció el camino más corto de todos los caminos establecidos hasta este punto.

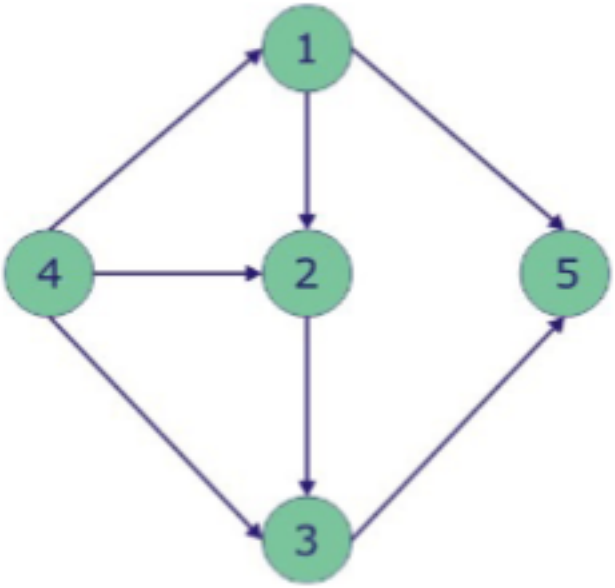


Figure 3: archivos/imagenes/Pasted image 20241110130607.png

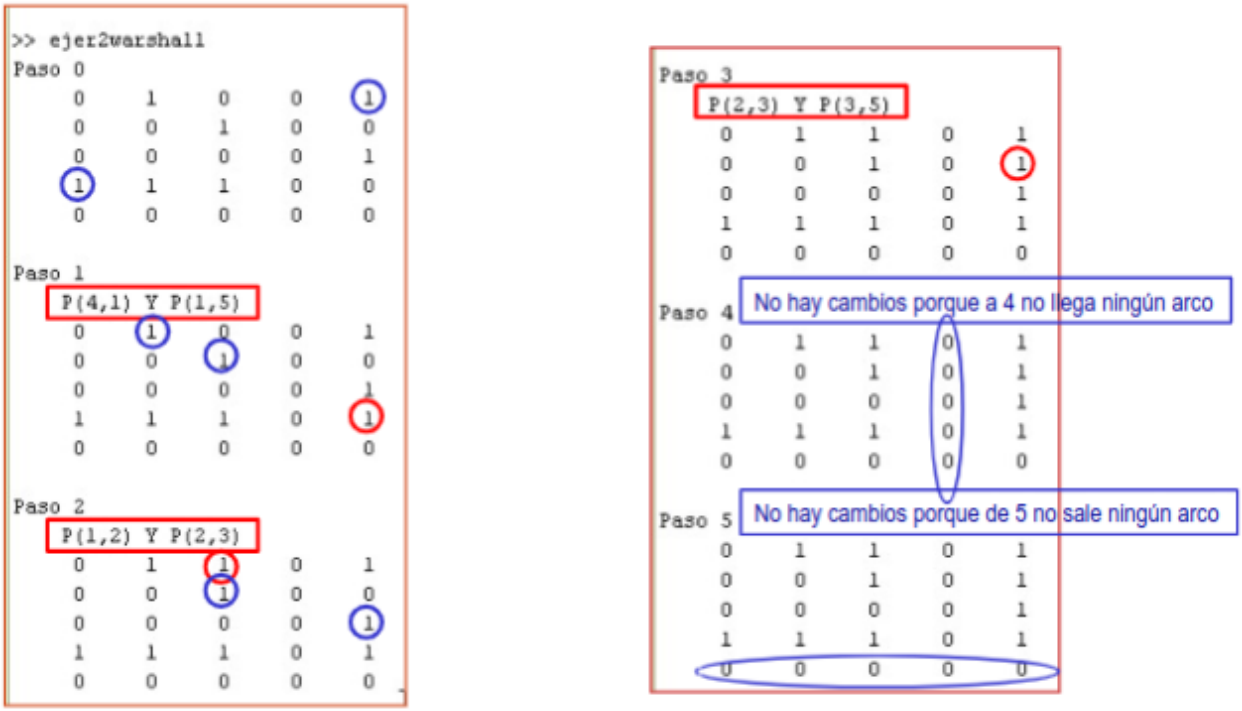


Figure 4: archivos/imagenes/Pasted image 20241110130551.png

Se repiten estos pasos sobreescribiendo solo el valor de aquellos caminos que tuvieran una longitud mayor a la obtenida por medio del vértice de estudio actual. Como se ve, tiene la desventaja de que si queremos saber el camino mínimo entre cualquier par de vértices, tenemos que realizar el algoritmo partiendo de todos los vértices del grafo y luego comparar.

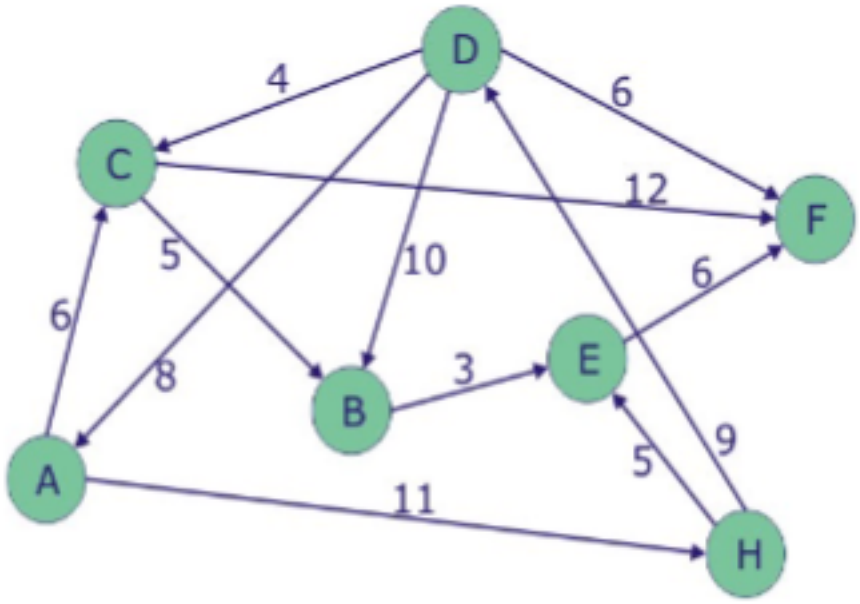


Figure 5: archivos/imagenes/Pasted image 20241110130652.png

7.4 Camino más corto entre cualquier par de vértices: Algoritmo de Floyd Para evitar el problema anterior, se implementó el **algoritmo de Floyd**. Este algoritmo, en vez de un vector como en el caso anterior, devuelve una matriz D donde cada campo (i, j) contiene el coste mínimo de los caminos que van desde i hasta j. Para realizarel algoritmo, se siguen los mismos pasos que con el **Algoritmo de Wharsall** para encontrar las distintas matrices de caminos de las diferentes longitudes.

En este caso: - La matriz D_1 muestra los caminos más cortos entre cualquier par de vértices con longitud 1. - La matriz D_2 muestra los caminos más cortos entre cualquier par de vértices con **longitud 2 o inferior**, y así sucesivamente.

Por tanto, la matriz D_n, siendo n el número de nodos, nos mostrará los caminos más cortos entre cualquier par de vértices, independientemente de la longitud del camino.

7.5 Control de flujo El control de flujo supone la forma de controlar la cantidad de objetos o elementos que se transportan de un lugar a otro. Algunos ejemplos son: - Flujo de mercancías entre el lugar donde se producen y donde se reciben. - Flujo de personas que realizan un desplazamiento entre dos puntos señalados. - Flujo de agua por medio de un sistema de tuberías.

Muchas veces, lo que se pretende es maximizar el flujo, es decir, transportar la mayor cantidad posible de elementos desde el punto de partida (*fuente* s) hasta un punto

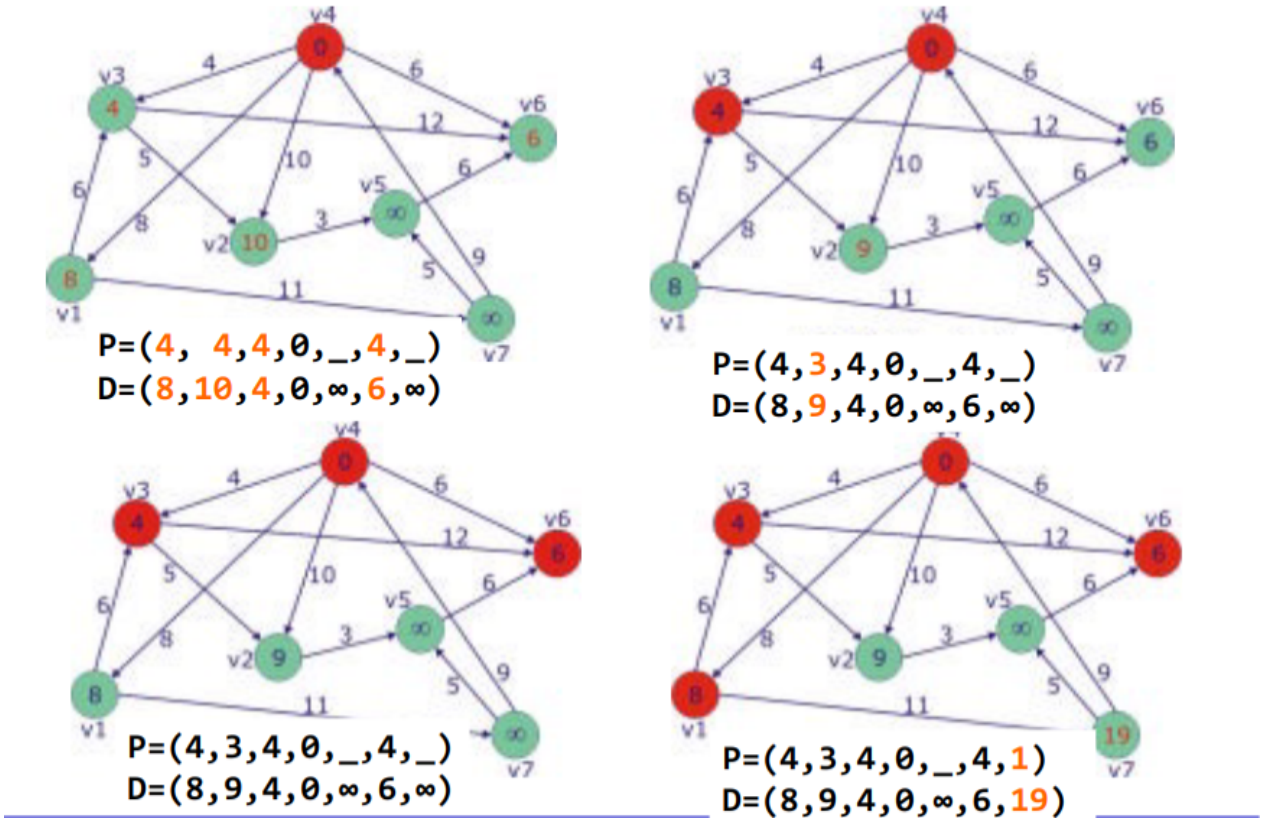
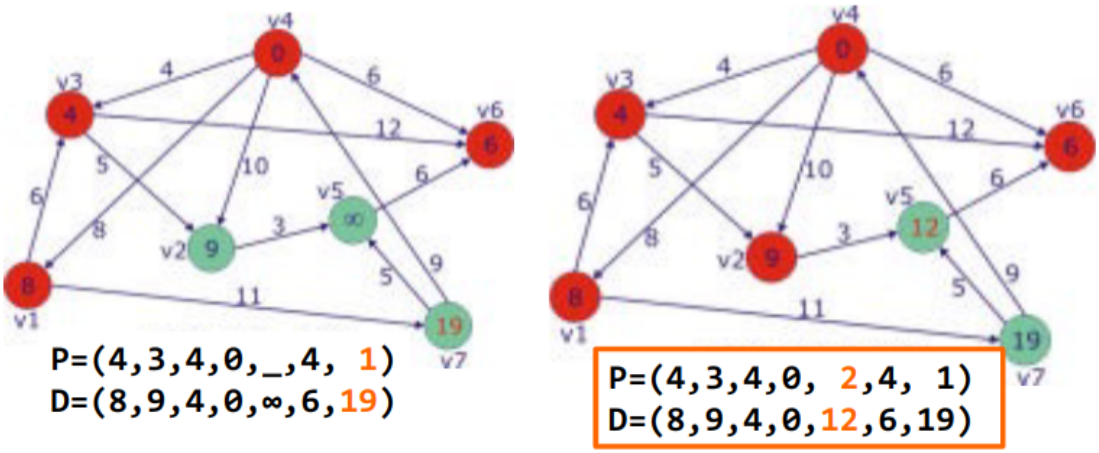


Figure 6: archivos/imagenes/Pasted image 20241110130714.png



Camino v4->v1: longitud=8, P(1)=4: v4→v1

Camino v4->v2: longitud=9, P(2)=3: v4→…→v3→v2, P(3)=4: v4→v3→v2

Camino v4->v3: longitud=4, P(3)=4: v4→v3

Camino v4->v5: longitud=12, P(5)=2: v4→…→v2→v5
P(2)=3: v4→…→v3→v2→v5, P(3)=4: v4→v3→v2→v5

Camino v4->v6: longitud=6, P(6)=4: v4→v6

Camino v4->v7: longitud=19, P(7)=1: v4→…→v1→v7, P(1)=4: v4→v1→v7

Figure 7: archivos/imagenes/Pasted image 20241110130732.png

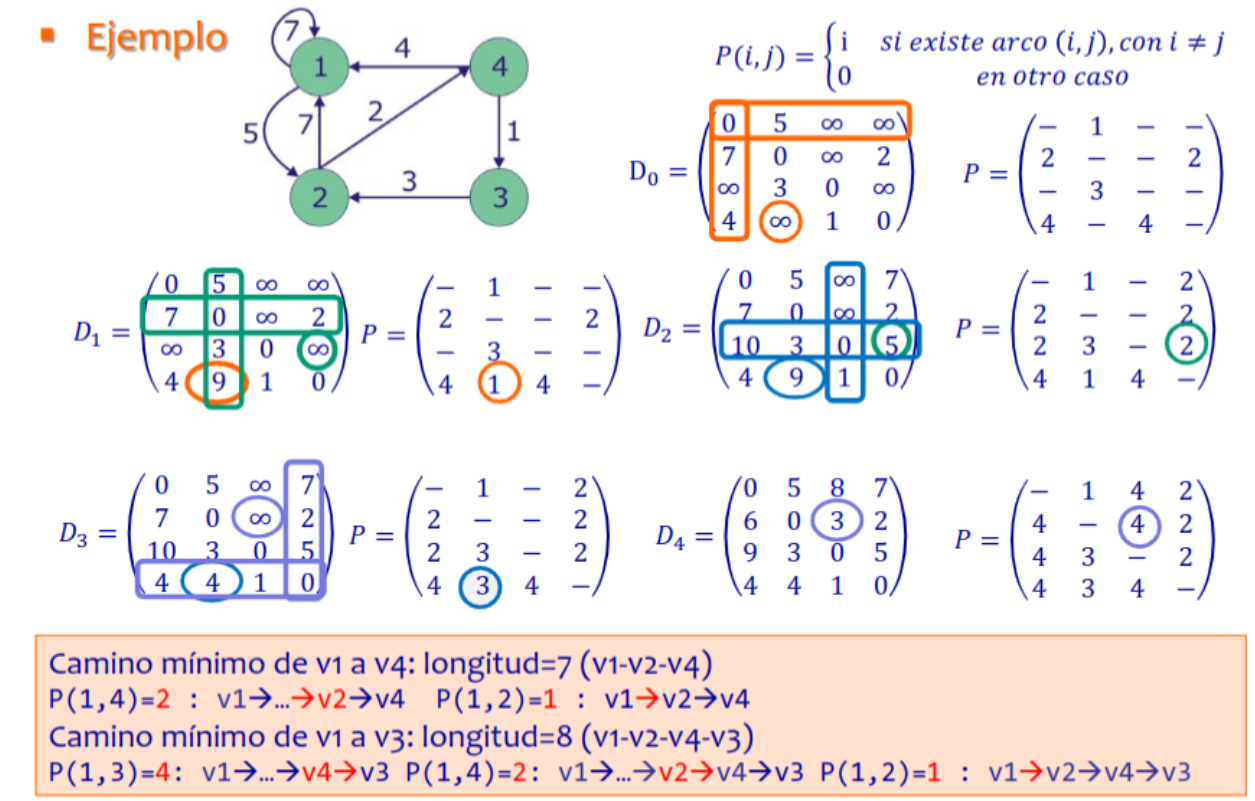


Figure 8: archivos/imagenes/Pasted image 20241110132055.png

final (*sumidero* t). Para este tipo de problemas, un **grafo valorado** es la estructura perfecta. En muchos casos, incluso un grafo dirigido valorado, dependiendo de las especificaciones concretas.

Para resolver estos problemas de maximización de flujo, se emplea el **Algoritmo de Ford-Fulkerson**, que busca caminos entre el nodo fuente y el sumidero en los que se pueda incrementar el flujo todavía más, para obtener en el sumidero la mayor cantidad de elementos posibles.

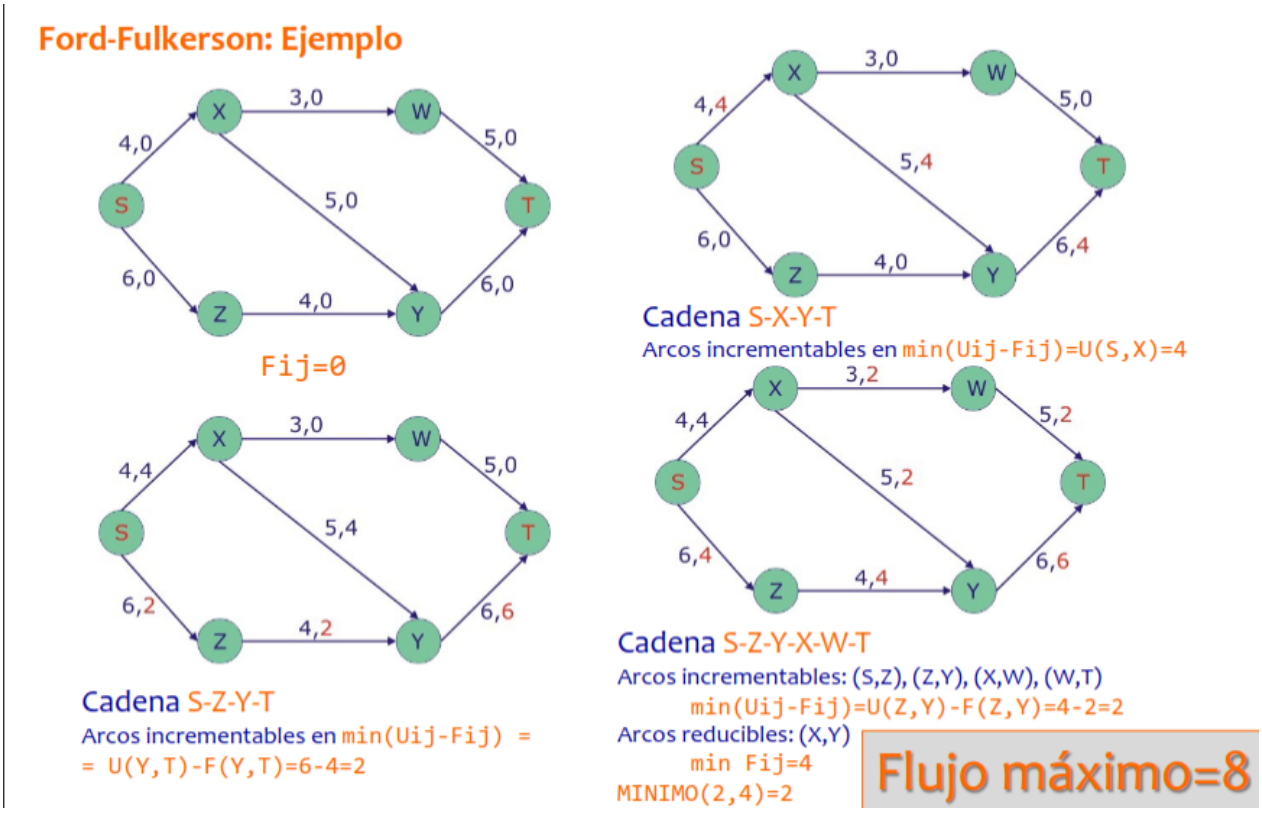


Figure 9: archivos/imagenes/Pasted image 20241110132251.png

7.6 Árbol de expansión de coste mínimo A veces, a partir de un grafo no dirigido, se pretenden modelar relaciones simétricas entre elementos. En esta situación, tenemos varias definiciones importantes: - **Red conectada**: Red (grafo que representa las relaciones entre elementos) en la que desde un vértice se puede llegar a cualquier otro por medio de un camino. - **Árbol**: Red que es subconjunto de la red original, siendo también una red conectada, con n nodos y $n-1$ vértices, y a la que, si se le añade un arco más, se forma un ciclo. - **Árbol de expansión**: Árbol que contiene todos los vértices de la red original. - **Árbol de expansión de coste mínimo**: Árbol de expansión en el cual el peso de sus arcos suma el menor valor posible.

Buscar un árbol de expansión es una forma de saber si una red es conectada. Los árboles de expansión de coste mínimo tienen muchas aplicaciones: - Diseño de redes de telecomunicaciones, para encontrar la forma más barata o corta de conectar todos los nodos. - Diseño del menor número de caminos que unan todos los pueblos de una

localidad de la forma más corta posible.

Para encontrar estos árboles de expansión existen dos algoritmos: el **Algoritmo de Prim** y el **Algoritmo de Kruskal**.

7.6.1 Algoritmo de Prim Es un **algoritmo voraz**, ya que en cada paso se añade el arco más corto disponible al árbol (mejor solución en cada paso). Se parte de un vértice inicial y se añade el vértice adyacente cuyo arco entre ambos tenga el menor peso. A continuación, entre los vértices ya conectados, se busca el arco de menor peso que conecte sus vértices adyacentes que no hayan sido conectados todavía. Estos pasos se repiten hasta haber conectado todos los vértices.

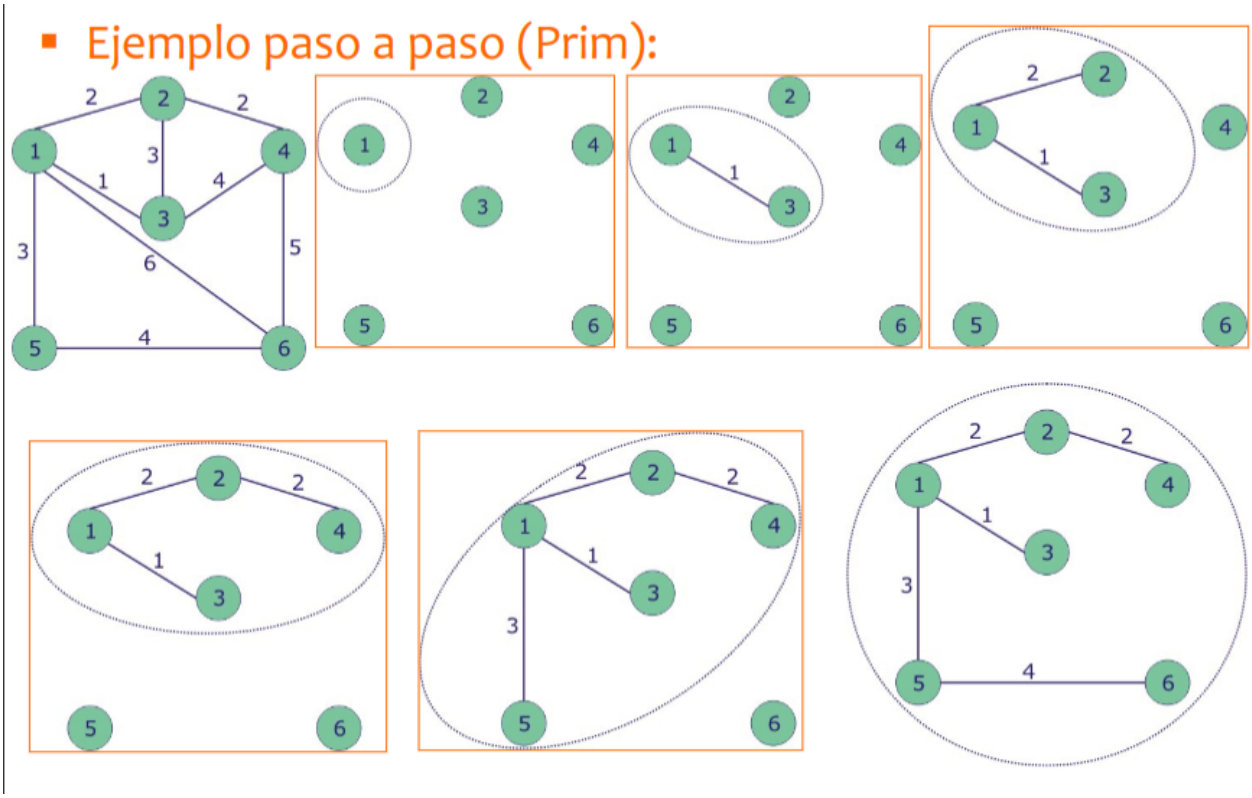


Figure 10: archivos/imagenes/Pasted image 20241110132337.png

7.6.2 Algoritmo de Kruskal Se parte del conjunto de vértices del grafo y se selecciona el arco de menor peso, uniendo dos vértices y reduciendo en uno el número de componentes conexas. A continuación, se selecciona el siguiente arco de menor peso que también conecte vértices de dos componentes conexas distintas, reduciendo otra vez el número total en uno. Si no se respetara esta condición, se producirían ciclos. El algoritmo finaliza cuando solo queda una componente conexa.

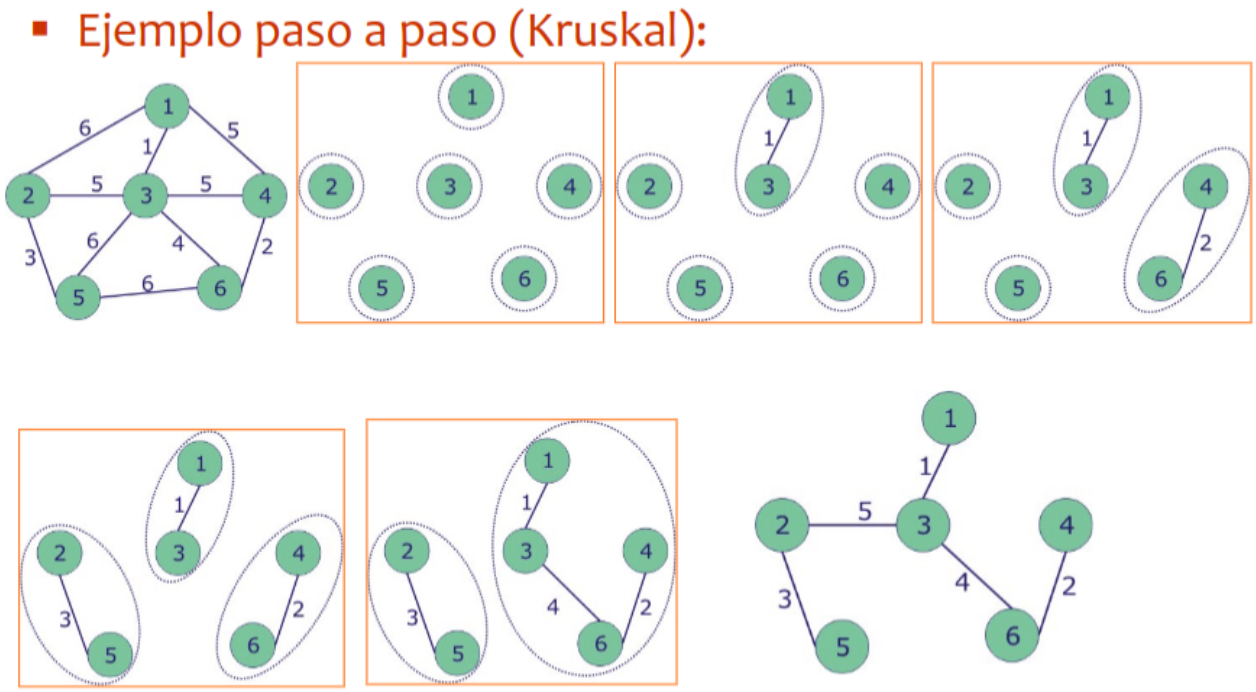


Figure 11: archivos/imagenes/Pasted image 20241110132356.png