INSTITUTO TECNOLÓGICO DE CULIACÁN

Ingeniería en Sistemas Computacionales

TAREA 1

Alumno: Franco Flores Luis Fernando

Imparte: Mora Félix Zuriel Dathan

Culiacán Sinaloa

14/Febrero/25

Problema de programación de trabajos:

El problema de programación de trabajos (JSSP) es un problema NP-hard que apunta a encontrar formas casi óptimas de asignar trabajos u órdenes de trabajo a máquinas y líneas. Los objetivos principales incluyen minimizar costos, demoras, inventario y otros factores relevantes. Dentro del ámbito de **la gestión de órdenes de trabajo** , JSSP se destaca como un paso crítico.

Ejemplo

Programemos la Orden de Trabajo 1 (WO 1), Orden de Trabajo 2 (WO 2), Orden de Trabajo 3 (WO 3) y Orden de Trabajo 4 (WO 4), con la siguiente información:

Job	Duration (Days)	Due Date
WO 1	5	6
WO 2	3	5
WO 3	7	14
WO 4	9	16

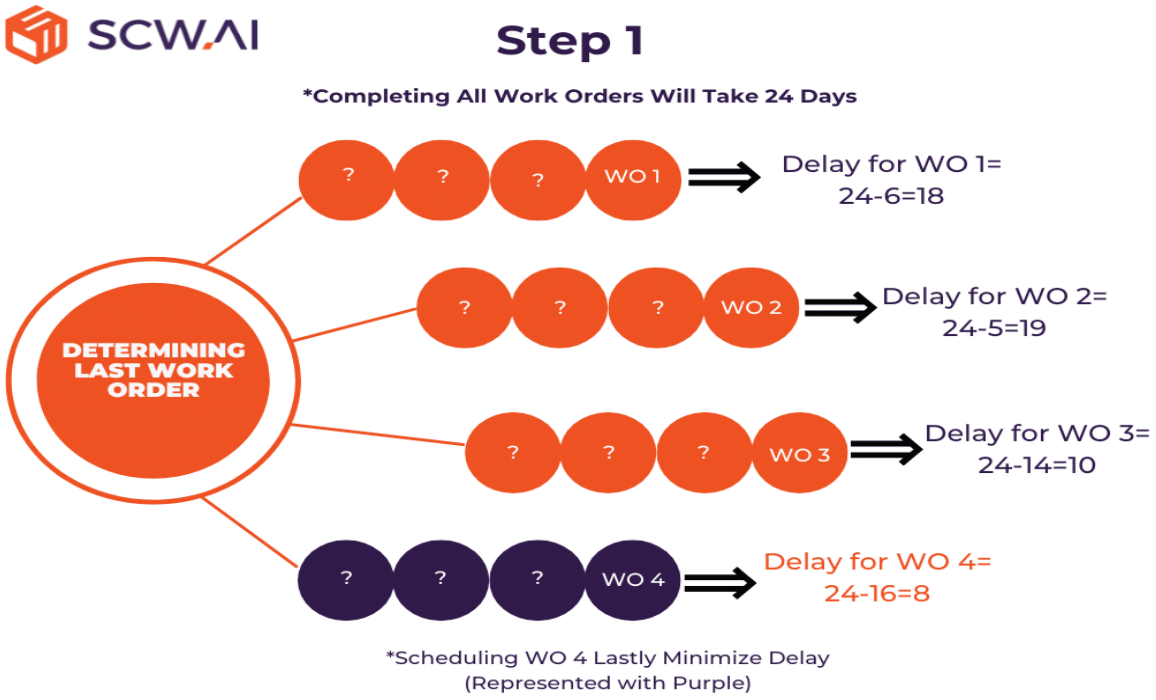
Hay 4 o 24 formas de secuenciar las órdenes de trabajo, pero con **Branch and Bound** no es necesario explorarlas todas. Usando árboles de decisión e inducción hacia atrás, primero se determina la última orden para minimizar retrasos y luego la primera.

Paso 1

Se calcula un tiempo total de 24 días para completar todas las órdenes. Evaluando los retrasos:

- **WO 1** → 18 días de retraso
- **WO 2** → 19 días de retraso
- **WO 3** → 10 días de retraso
- **WO 4** → 8 días de retraso (mínimo)

Dado que asignar **WO 4** al final minimiza el retraso, el algoritmo se ramifica desde este escenario.

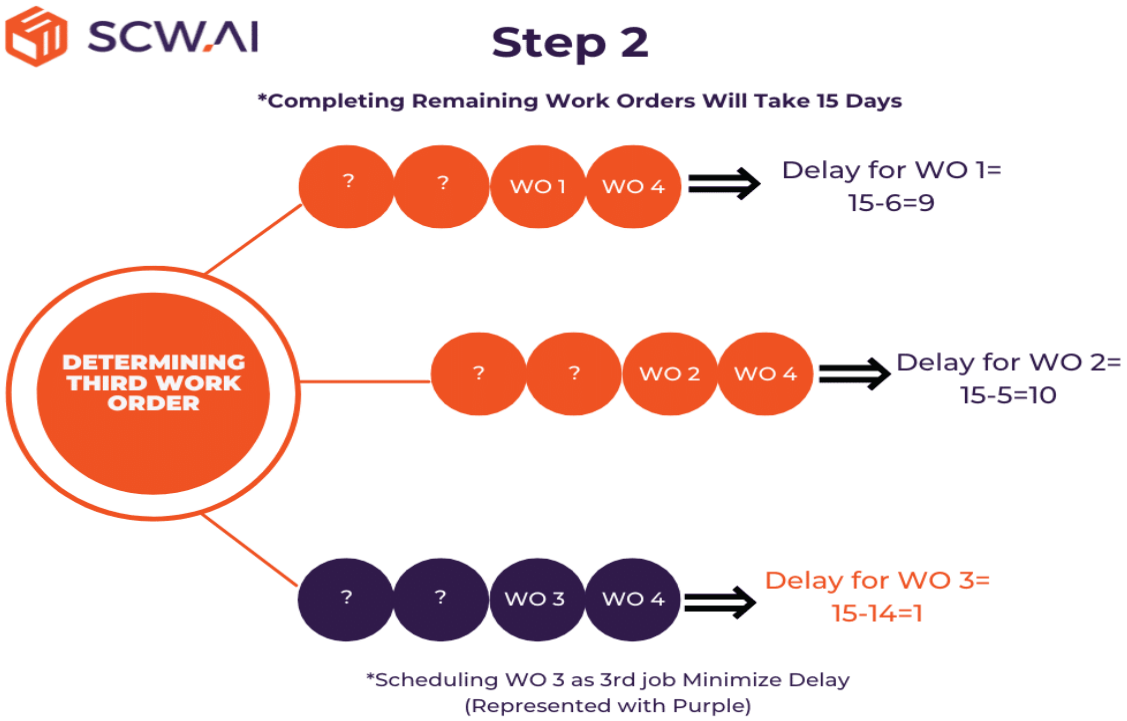


Paso 2

Con un retraso inicial de 8 días y 15 días restantes de trabajo, se evalúan los retrasos adicionales:

- **WO 1** → +9 días (total: 17 días)
- **WO 2** → +10 días (total: 18 días)
- **WO 3** → +1 día (total: 9 días)

Como **WO 3** minimiza el retraso, el algoritmo se ramifica desde este escenario.



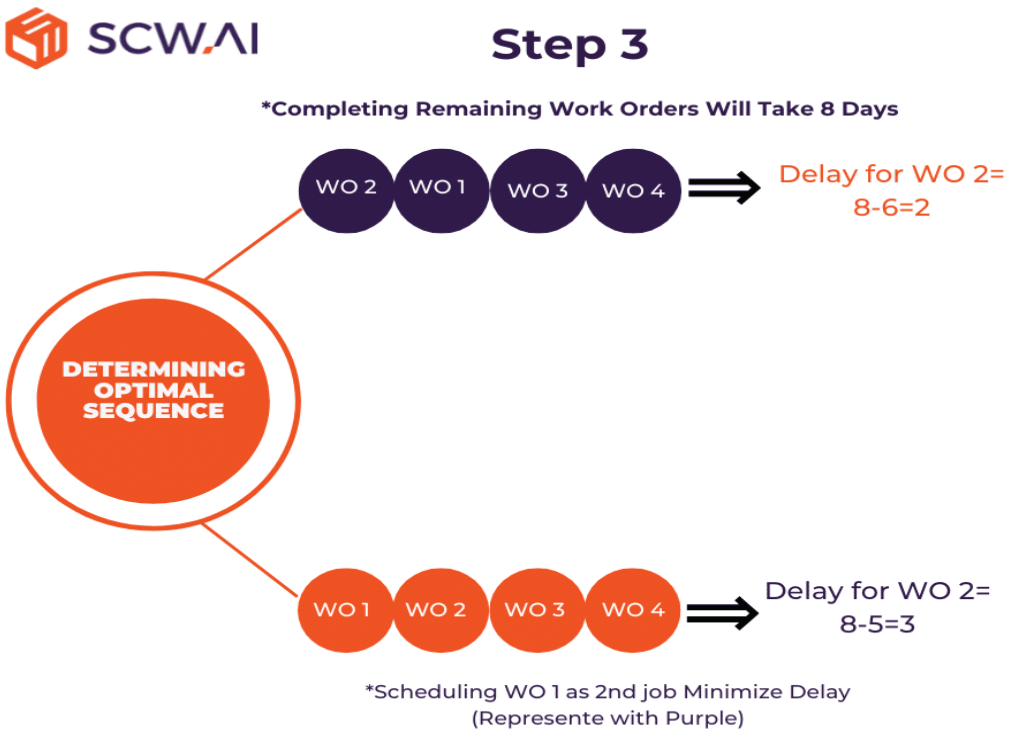
Paso 3

Con 9 días de retraso y 8 días restantes de trabajo, se evalúan los retrasos adicionales:

- **WO 1 como segundo trabajo** → +2 días (total: 11 días)
- **WO 2 como segundo trabajo** → +3 días (total: 12 días)

Dado que **WO 1** minimiza el retraso, el algoritmo sigue esta opción. La **WO 2** se programa como la primera tarea sin demoras adicionales.

Secuencia óptima: **WO 2 → WO 1 → WO 3 → WO 4** (total: 12 días de retraso).



Problema de las N reinas

El problema de las N reinas es un problema clásico de la computación y la teoría de la complejidad que consiste en colocar N reinas en un tablero de ajedrez N × N de manera que ninguna de ellas se ataque entre sí. Esto significa que:

1. No puede haber dos reinas en la misma fila.
2. No puede haber dos reinas en la misma columna.
3. No puede haber dos reinas en la misma diagonal (ascendente o descendente).

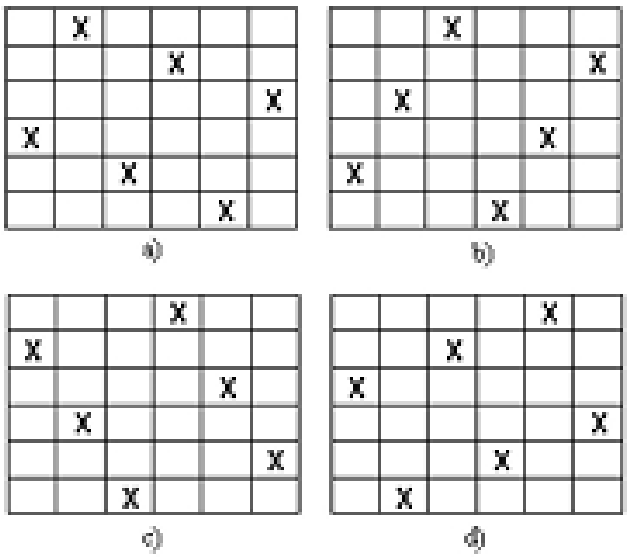
Ejemplo

Para comprender mejor el problema de las n reinas, analizaremos el caso específico de n=6n = 6n=6. Este problema consiste en colocar 6 reinas en un tablero de 6x6 sin que se ataquen entre sí.

Para encontrar estas soluciones, podemos partir de una solución inicial, denominada **a)**, y derivar las demás mediante transformaciones:

- **La solución b)** se obtiene al girar la solución **a)** 90 grados en el sentido de las agujas del reloj.

- **La solución c)** es el reflejo de la solución **b)** con respecto al borde derecho, como si colocáramos un espejo en ese lado.
- **La solución d)** se obtiene reflejando la solución **a)** con respecto al borde derecho del tablero.



El método de backtracking (o retroceso) es una técnica de búsqueda en la que exploramos todas las posibles configuraciones válidas, retrocediendo cuando encontramos una opción inválida.

Concepto de Backtracking

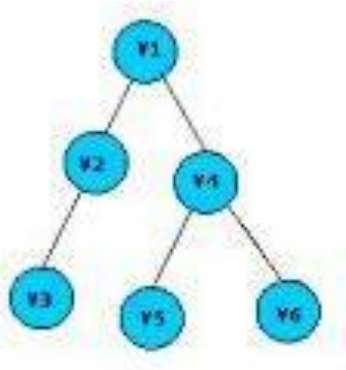
1. **Colocar una reina en la primera fila.**
2. **Intentar colocar la siguiente reina en la siguiente fila, evitando conflictos con las anteriores.**
3. **Si se encuentra una posición válida, continuar con la siguiente fila.**
4. **Si no hay posiciones válidas en una fila, retroceder a la fila anterior y mover la última reina colocada a la siguiente posición posible.**
5. **Repetir hasta completar todas las reinas o probar todas las combinaciones posibles.**

Árbol de expansión mínima (MST)

Un árbol es un tipo especial de grafo en el que existe un único nodo desde el cual se puede acceder a todos los demás. Además, cada nodo tiene un único predecesor, excepto el nodo inicial, que no tiene ninguno.

Otra manera de definir un árbol es:

- **Un grafo conexo y sin ciclos.**
- **Un grafo con n vértices y n−1 aristas, donde n representa el número total de nodos.**



- **Grado de un nodo**: Se refiere a la cantidad de subárboles que dependen de ese nodo. Por ejemplo, si un nodo tiene dos subárboles, su grado es 2.
- **Hojas**: Son los nodos terminales de un árbol, es decir, aquellos que no tienen nodos descendientes.
- **Árbol de máximo alcance**: Es el árbol que se obtiene a partir de un grafo conexo y sin ciclos.
- **Árbol de mínima expansión**: Es un árbol de máximo alcance con la menor suma posible de pesos en sus aristas. En otras palabras, es el árbol que conecta todos los nodos con el menor costo total.

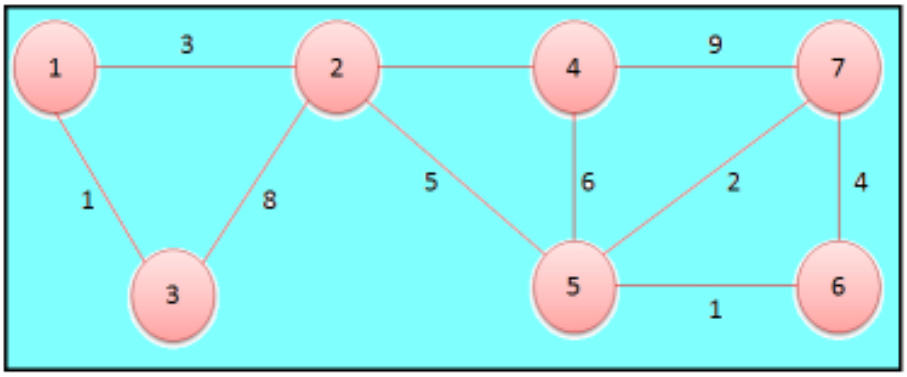
Ejemplo

ALGORITMO DE KRUSKAL

El algoritmo de Kruskal permite hallar el árbol minimal de cualquier grafo valorado (con capacidades).

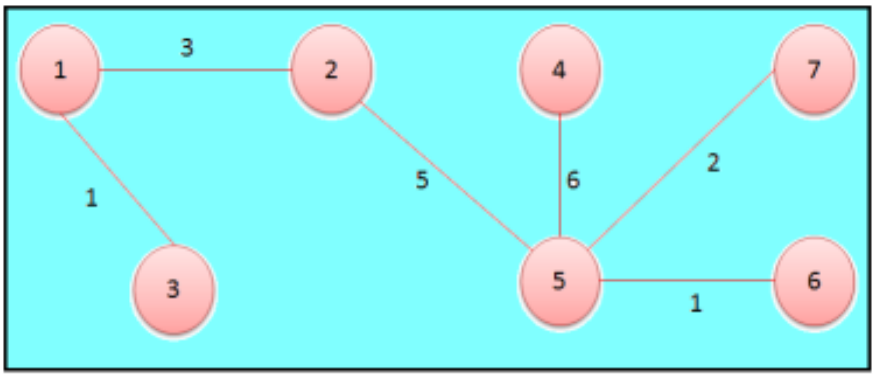
1. **Se marca la arista con menor valor. Si hay más de una, se elige cualquiera de ellas.**

2. **De las aristas restantes, se marca la que tenga menor valor, si hay más de una, se elige cualquiera de ellas.**
3. **Repetir el paso 2 siempre que la arista elegida no forme un ciclo con las ya marcadas.**
4. **El proceso termina cuando tenemos todos los nodos del grafo en alguna de las aristas marcadas, es decir, cuando tenemos marcados *n-1* arcos, siendo *n* el número de nodos del grafo.**



Siguiendo el algoritmo de Kruskal, tenemos:

- **Elegimos, por ejemplo, la arista *(5, 6)* = *1* (menor valor) y la marcamos.**
- **Elegimos la siguiente arista con menor valor *(1, 3)* = *1* y la marcamos.**
- **Elegimos la siguiente arista con menor valor *(5, 7)* = *2* y la marcamos, ya que no forma ciclos con ninguna arista de las marcadas anteriormente.**
- **Elegimos la siguiente arista con menor valor *(1, 2)* = *3* y la marcamos, ya que no forma ciclos con ninguna arista de las marcadas anteriormente.**
- **Elegimos la siguiente arista con menor valor *(6, 7)* = *4* y la desechamos, ya que forma ciclos con las aristas *(5, 7)* y *(5, 6)* marcadas anteriormente.**
- **Elegimos la siguiente arista con menor valor *(2, 5)* = *5* y la marcamos, ya que no forma ciclos con ninguna arista de las marcadas anteriormente.**
- **Elegimos la siguiente arista con menor valor *(4, 5)* = *6* y la marcamos, ya que no forma ciclos con ninguna arista de las marcadas anteriormente.**
- **FIN. Finalizamos dado que los *7* nodos del grafo están en alguna de las aristas, o también ya que tenemos marcadas *6* aristas (*n-1*).**
- **Por tanto el árbol de mínima expansión resultante sería:**



Problema del agente viajero (TSP)

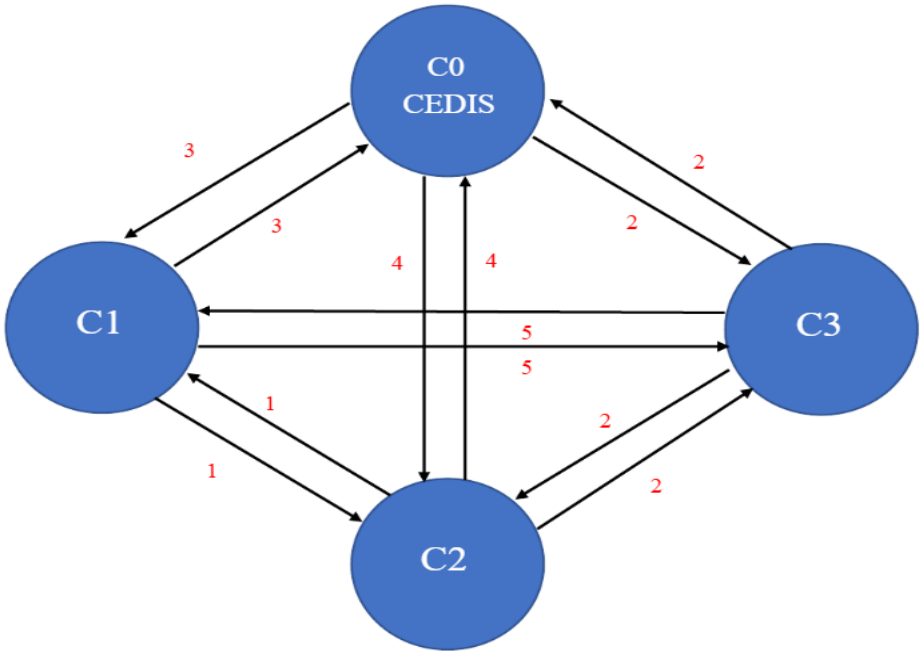
El **problema del agente viajero (TSP - Traveling Salesman Problem)** es un problema de optimización combinatoria que busca encontrar la ruta más corta para un viajero que debe visitar un conjunto de ciudades exactamente **una vez** y regresar a la ciudad de origen. Se trata de un problema **NP-difícil**, lo que significa que no existe un algoritmo eficiente conocido para resolverlo en todos los casos. Se utiliza en áreas como logística, planificación de rutas y optimización de redes.

Ejemplo

La empresa LALA busca las rutas más cortas en tiempo y costos para repartir sus productos desde su CEDIS a sus diferentes clientes dentro de la ciudad. En base al método del agente viajero encuentra la solución más optima utilizando la fuerza bruta.

PASO 1: Teniendo el grafico o mapa del punto de partida y los lugares a visitar, el primer paso es escribir los costos (ya sea en términos de dinero o distancias) dentro de este.

	C0	C1	C2	C3
C0	0	3	4	2
C1	3	0	1	5
C2	4	1	0	2
C3	2	5	2	0



PASO 2: De acuerdo al mapa anterior se procede a establecer las posibles rutas alternas para repartir los productos.

C0-C1-C2-C3-C0

C0-C3-C2-C1-C0

C0-C1-C3-C2-C0

C0-C3-C1-C2-C0

C0-C2-C1-C3-C0

C0-C2-C3-C1-C0

Una vez descifradas las rutas a través de la siguiente fórmula matemática, se procese a calcular el numero de rutas reales posibles para realizar la repartición. (Esto se hace ya que como podemos observar en las rutas anteriores hay algunas en las cuales el recorrido es el mismo, pero al revés y por ende son rutas repetitivas)

$$(n-1!) / 2$$

n = número de nodos totales.

! (factorial)

Sustituyendo la formula:

$$\frac{(4-1!)}{2} = \frac{3!}{2} = \frac{3*2*1}{2} = \frac{6}{2}$$

PASO 3: De acuerdo a la formula anterior, tenemos un total de 3 rutas posibles, asi que de las rutas anteriores procedemos a eliminar aquellas que se repitan

~~C0-C1-C2-C3-C0 =8~~

C0-C3-C2-C1-C0 = 8

~~C0-C1-C3-C2-C0 = 14~~

C0-C3-C1-C2-C0 = 14

~~C0-C2-C1-C3-C0 = 12~~

C0-C2-C3-C1-C0 = 12

PASO 4: Con las rutas repetidas eliminadas, procedemos a comparar los costos / distancias por cada ruta para poder seleccionar la más optima (la de menor costo o distancia).

C0-C3-C2-C1-C0 = 8

C0-C3-C1-C2-C0 = 12

C0-C2-C3-C1-C0 = 14

Como podemos observar, una vez calculando los costos obtenemos que la ruta mas optima para realizar el recorrido es: C0-C3-C2-C1-C0 con un total de 8 km recorridos.