DÍA 5

El problema de este día consiste en **ordenar ciertas páginas** de acuerdo con un orden específico. Se nos proporcionan dos tipos de datos:

1. **Datos con |**: Estos datos indican el orden relativo entre dos páginas. Por ejemplo, si se nos da 3|9, esto significa que, para que el orden sea correcto, la página 3 debe aparecer antes que la página 9.
2. **Datos con ,**: Estos datos representan distintas formas en las que se han ordenado las páginas. Utilizando la información proporcionada por los datos con |, debemos determinar si estas ordenaciones son válidas o no.

El problema se puede resolver utilizando un grafo, ya que es necesario modelar las relaciones entre los números (páginas) como conexiones dirigidas. Luego, verificamos si un vector dado respeta las relaciones del grafo para determinar si su orden es correcto.

1. La clase grafo

El grafo se almacena como un mapa, que tiene dos variables:

```
class Grafo {
private:
    map<int, vector<int>> conexiones;
```

- **Nodo (int)**: Nodo de origen.
- **Lista (vector<int>)**:

2. Funciones de la clase Grafo

En la clase grafo tenemos **tres** funciones públicas para crearlo:

2.1. Agregar conexión

Añade una arista (relación) dirigida desde el nodo **desde** al nodo **hasta.**

```
void agregarConexion(int desde, int hasta) {
    conexiones[desde].push_back(hasta);
```

Si **desde** no existe, se crea automáticamente. Luego, se añade **hasta** como vecino.

2.2. Obtener Vecinos

Devuelve una lista con los nodos vecinos de un nodo dado.

```
vector<int> obtenerVecinos(int nodo) {
    if (conexiones.find(nodo) != conexiones.end()) {
        return conexiones[nodo];
    } else {
        return {}; // Si el nodo no tiene vecinos, devolver una lista vacía
    }
}
```

Cómo funciona:

- Usa **find** para buscar el nodo en el mapa.
- Si existe, retorna su lista de vecinos.
- Si no, retorna un vector vacío.

2.3. Es Orden Valido

Comprueba si una lista de nodos está en un orden válido según las conexiones del grafo.

```
bool esOrdenValido(vector<int> nodos) {
    set<int> procesados; // Almacena los nodos que ya hemos procesado

    for (int nodo : nodos) {
        // Verificamos si algún vecino ya ha sido procesado
        for (int vecino : obtenerVecinos(nodo)) {
            if (procesados.count(vecino) > 0) {
                return false; // Si un vecino ya está procesado, el orden no es válido
            }
        }
        // Marcamos el nodo como procesado
        procesados.insert(nodo);
    }

    return true; // Si no hay conflictos, el orden es válido
}
```

Cómo funciona:

- Usa un conjunto (set) llamado procesados para registrar los nodos ya visitados.
- Recorre los nodos de la lista:
    - Para cada nodo, revisa sus vecinos.
    - Si algún vecino ya está en procesados, el orden no es válido y retorna false.
- Si no hay conflictos, añade el nodo a procesados.
- Al final, si todo es correcto, retorna true.

3. Función main

3.1. Lectura del archivo

```
while (getline(archivo, linea)) {
    if (linea.empty() || !isdigit(linea[0])) {
        continue; // Saltar líneas vacías o líneas que no empiezan con un número
    }

    vector<int> nodos; // Almacena los nodos leídos de la línea
    bool tieneConexion = false; // Indica si la línea representa una conexión

    // Leer los números de la línea
    for (int i = 0; i < linea.length(); i++) {
        if (isdigit(linea[i])) {
            int numero = 0;
            // Construir el número completo
            while (i < linea.length() && isdigit(linea[i])) {
                numero = numero * 10 + (linea[i] - '0');
                i++;
            }
            nodos.push_back(numero);

            // Si después del número hay un '|', es una conexión
            if (i < linea.length() && linea[i] == '|') {
                tieneConexion = true;
```

Como funciona:

- Si la línea está vacía o no comienza con un número, se salta.
- **Extraer números y detectar conexiones**:
  - Recorre cada carácter de la línea.
  - Si encuentra un número, lo convierte a entero y lo almacena en **nodos**.
- Si después del número hay un carácter |, marca **tieneConexion** como **true**.

3.2. Determinar si tiene orden valido

```
if (tieneConexion && nodos.size() >= 2) {
    // Si la línea es una conexión, agregarla al grafo
    grafo.agregarConexion(nodos[0], nodos[1]);
} else {
    // Si no es una conexión, verificar el orden de los nodos
    if (grafo.esOrdenValido(nodos)) {
        solucion += nodos[nodos.size() / 2]; // Sumar el nodo del medio
```

- **Líneas con conexión (|)**:
  Si hay dos nodos y una conexión (**tieneConexion**), añade la conexión al grafo.

- **Líneas sin conexión**:
  Comprueba si los nodos están en un orden válido usando **esOrdenValido**.
  Si el orden es válido, suma el valor del nodo central a solución.