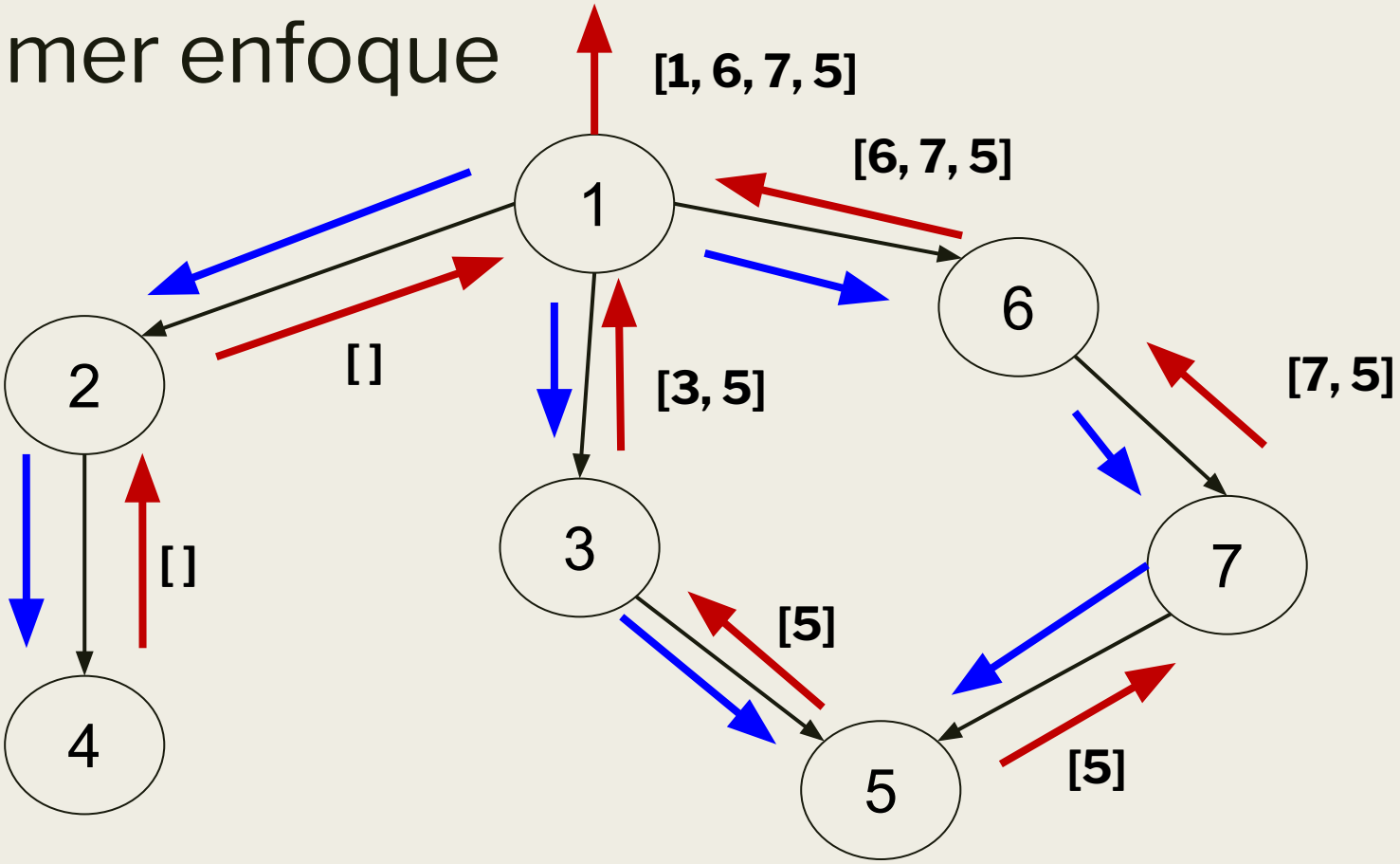
Camino más largo Grafos + Back

Escribir un algoritmo que, dado un grafo dirigido y dos vértices i, j de este grafo, devuelva el camino simple (sin ciclos) de mayor longitud del vértice i al vértice j.

Primer enfoque

- **Construyendo la solución a la vuelta de la recursión.**
- Enfoque similar a lo visto en Árbol: getLongestBranch().
    - Cada vértice “consulta” a sus adyacentes por su mejor camino.
    - Cada adyacente retorna **su** mejor camino al destino.
    - Cada vértice necesita las respuestas de todos sus adyacentes para generar su respuesta.

Primer enfoque



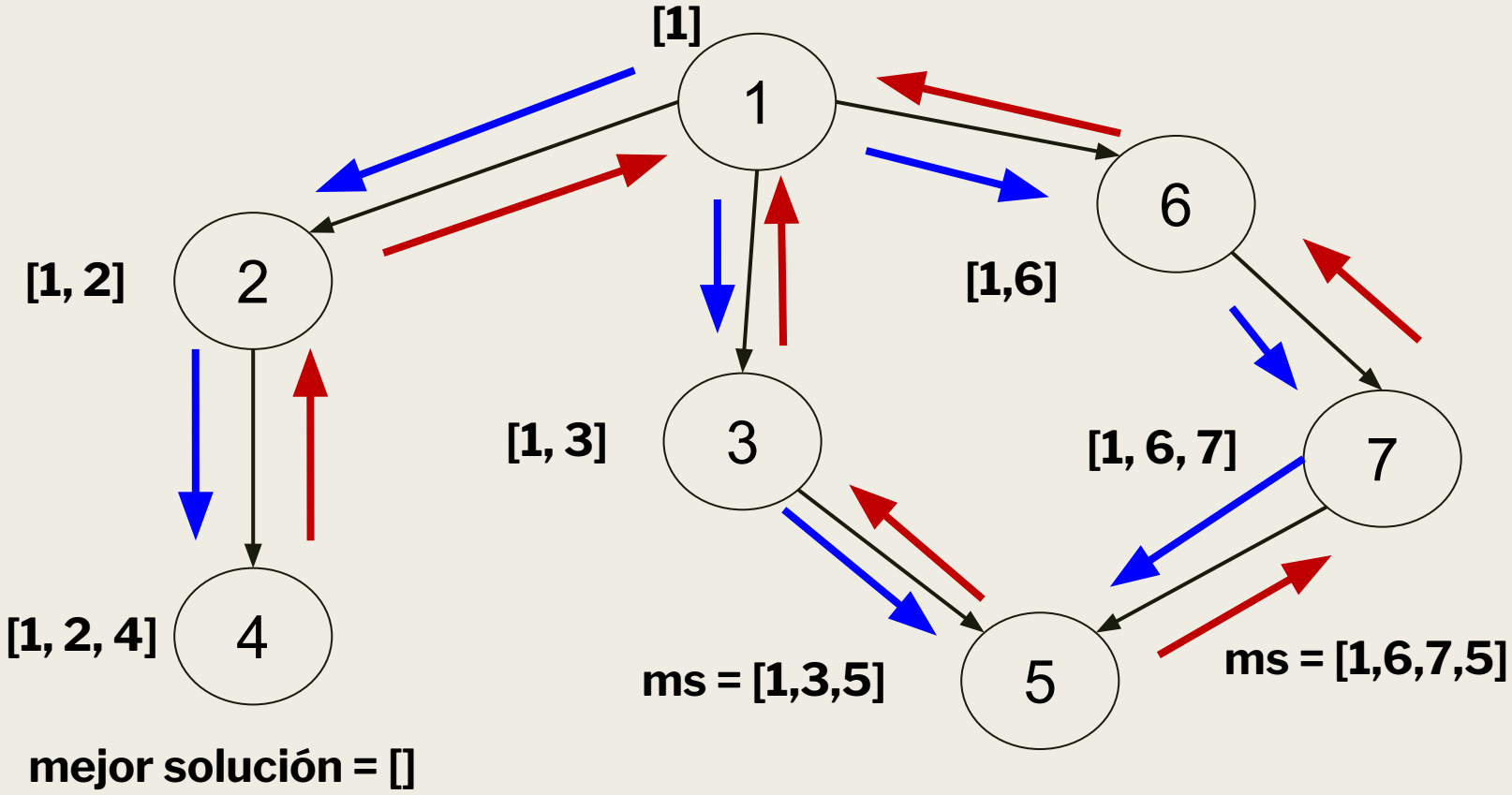
Primer enfoque

```
public ArrayList<Integer> caminoMayor(int origen, int destino) {
        ArrayList<Integer> caminoMayor = new ArrayList<Integer>();
        this.visitados.add(origen);
        if (origen == destino) {
                caminoMayor .add(origen);
        } else {
                Iterator<Integer> it_ady = this.grafo.obtenerAdyacentes(origen);
                while (it_ady.hasNext()) {
                        Integer ady = it_ady.next();
                        if (!this.visitados.contains(ady)) {
                                ArrayList<Integer> camino = caminoMayor(ady, destino);
                                if (!camino.isEmpty() && (camino .size() >= caminoMayor .size())) {
                                        caminoMayor.clear();
                                        caminoMayor.add(origen);
                                        caminoMayor.addAll(camino);
                                }
                        }
                }
        }
        this.visitados.remove(origen);
        return caminoMayor ;
}
```

Segundo enfoque

- **Llevamos la solución parcial en cada momento.**
- Enfoque más similar a la estructura clásica de Backtracking.
    - Necesitamos un “estado” (en este caso el camino parcial)
    - Tenemos que realizar acciones para avanzar de estado.
    - Tenemos que deshacer las acciones para volver al estado anterior.
- **La mejor solución** es más simple llevarla como un parámetro de clase.
- Generalmente necesitamos un método auxiliar (configurar estado inicial).

Segundo enfoque



mejor solución = []

Segundo enfoque

```
public ArrayList<Integer> otroCaminoMayor(int origen, int destino) {
        this.visitados.clear();
        this.caminoMayor.clear();

        //Configurar estado inicial
        ArrayList<Integer> caminoParcial = new ArrayList<Integer>();
        caminoParcial.add(origen);
        this.visitados.add(origen);

        this.otroCaminoMayor(origen, destino, caminoParcial);

        return this.caminoMayor;
}
```

Segundo enfoque

```
private void otroCaminoMayor(int origen, int destino, ArrayList<Integer> caminoParcial) {
        if (origen == destino) {
                if (caminoParcial.size() >= caminoMayor.size()) {
                        caminoMayor.clear();
                        caminoMayor.addAll(caminoParcial);
                }
        } else {
                Iterator<Integer> it_ady = this.grafo.obtenerAdyacentes(origen);
                while (it_ady.hasNext()) {
                        Integer ady = it_ady.next();
                        if (!visitados.contains(ady)) {
                                caminoParcial.add(ady);
                                this.visitados.add(ady);
                                otroCaminoMayor(ady, destino, caminoParcial);
                                caminoParcial.remove(ady);
                                this.visitados.remove(ady);
                        }
                }
        }
}
```