Grafos

Ordenamiento Topológico

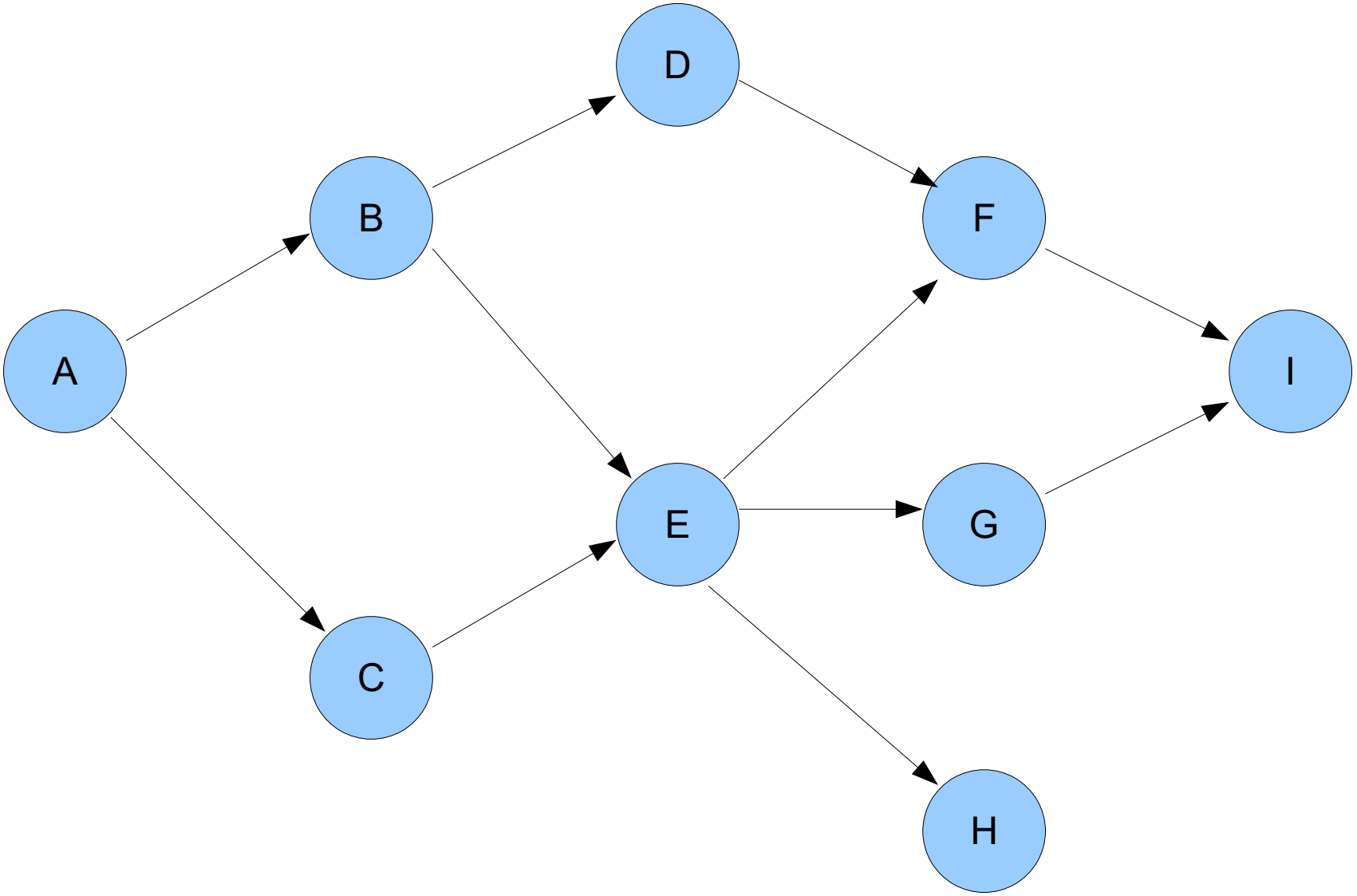
Ordenamiento topológico

- Consideremos una situación donde se tienen una serie de tareas que tienen dependencias unas de otras, por ejemplo:
  - Compilar un programa y sus módulos
  - Ejecutar actividades de un proyecto
  - Cursos en la malla curricular
- El problema es encontrar un orden factible para ejecutar las distintas tareas.

Consideraciones para el ordenamiento topológico

- El grafo debe ser un DAG
- Un recorrido en profundidad permite ordenar los nodos del grafo tanto en pre-orden como en post-orden.
- Si enumeramos los nodos en post-orden, un nodo solo se enumera después de haber visitado todas sus dependencias.
- Si enumeramos los nodos en orden inverso del post-orden, se tienen un ordenamiento topológico.

Ejemplo



Implementación de la solución (1)

Se implementa un recorrido DFS que guarde un registro de los nodos visitados en post-orden.

```
public class DepthFirstOrder {
    private boolean[] marked;
    private int[] pre;
    private int[] post;
    private Queue<Integer> preorder;
    private Queue<Integer> postorder;
    private int preCounter;
    private int postCounter;

    public DepthFirstOrder(Digraph G) {
        pre    = new int[G.V()];
        post   = new int[G.V()];
        postorder = new Queue<Integer>();
        preorder  = new Queue<Integer>();
        marked    = new boolean[G.V()];
        for (int v = 0; v < G.V(); v++)
            if (!marked[v]) dfs(G, v);

    }
    private void dfs(Digraph G, int v) {
        marked[v] = true;
        pre[v] = preCounter++;
        preorder.enqueue(v);
        for (int w : G.adj(v)) {
            if (!marked[w]) {
                dfs(G, w);
            }
        }
        postorder.enqueue(v);
        post[v] = postCounter++;
    }
```

Ver DepthFirstOrder.java

Implementación de la solución (2)

Luego de hacer el recorrido se invierte el orden de los nodos de la lista en post-orden.

```
public class Topological {
    private Iterable<Integer> order;
    private int[] rank;

    public Topological(Digraph G) {
        DirectedCycle finder = new DirectedCycle(G);
        if (!finder.hasCycle()) {
            DepthFirstOrder dfs = new DepthFirstOrder(G);
            order = dfs.reversePost();
            rank = new int[G.V()];
            int i = 0;
            for (int v : order)
                rank[v] = i++;
        }
    }
…

public class DepthFirstOrder {

    public Iterable<Integer> reversePost() {
        Stack<Integer> reverse = new Stack<Integer>();
        for (int v : postorder)
            reverse.push(v);
        return reverse;
    }
```

Ver Topological.java

Ejemplo

```
public static void main(String[] args) throws Exception {
    String[] nombres = {"A","B","C","D","E","F","G","H","I"};
    DigrafoConNombres g = new DigrafoConNombres(nombres);
    g.addEdge("A", "B");
    g.addEdge("A", "C");
    g.addEdge("B", "D");
    g.addEdge("B", "E");
    g.addEdge("C", "E");
    g.addEdge("D", "F");
    g.addEdge("E", "F");
    g.addEdge("E", "G");
    g.addEdge("E", "H");
    g.addEdge("F", "I");
    g.addEdge("G", "I");

    StdOut.println(g);

    Topological topological = new Topological(g.getDigraph());
    for(int i: topological.order()) {
        StdOut.println(topological.rank(i)+" : "+g.getNombre(i));
    }

}
```

Aplicaciones del ordenamiento Topológico

- Diagramas de Gantt: Una forma de representar dependencias y duraciones de las actividades en un proyecto.
- Cálculo de la ruta crítica (Critical Path Method): La lista de tareas que determina el tiempo mínimo para ejecutar un proyecto.