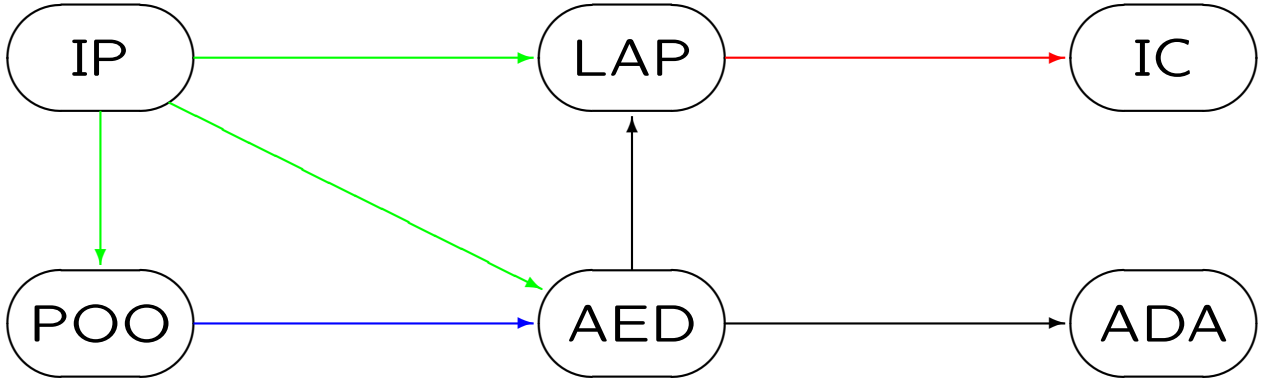
# Capítulo IV

# Ordenação Topológica
# &
# Teste à Aciclicidade
# (num grafo orientado)

# Ordenação Topológica

Dado um grafo $G = (V, A)$, **orientado e acíclico**, uma **ordenação topológica** de $G$ é uma permutação de $V$ tal que:

$$\forall (x, y) \in A \quad x \text{ precede } y.$$

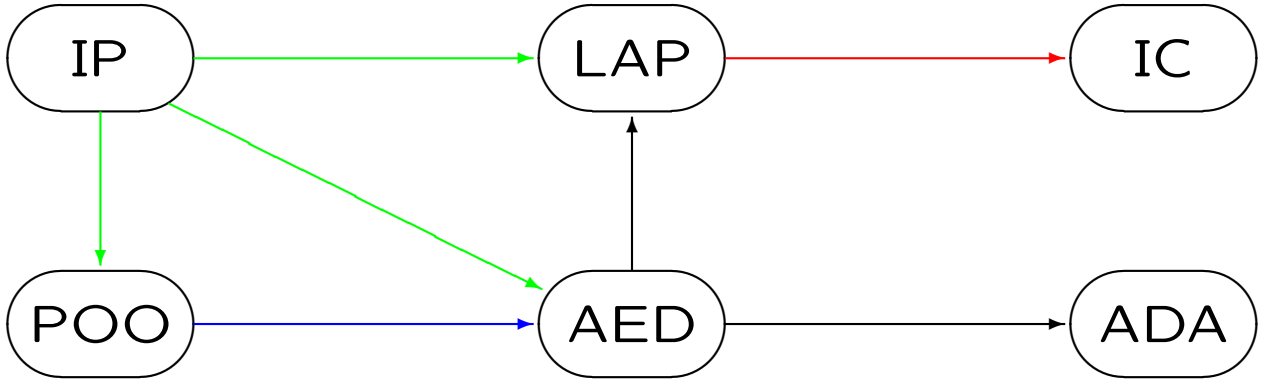



IP POO AED LAP IC ADA

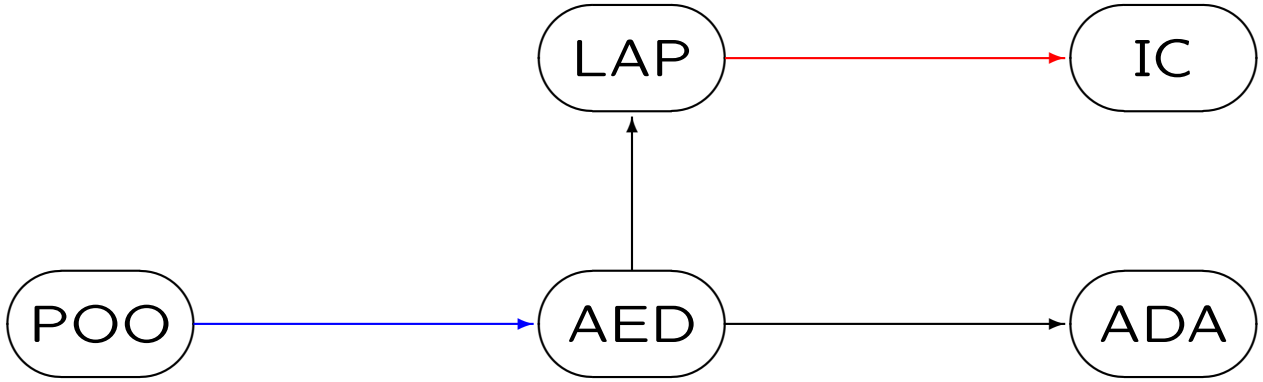
IP POO AED LAP ADA IC

IP POO AED ADA LAP IC

# Exemplo (1)

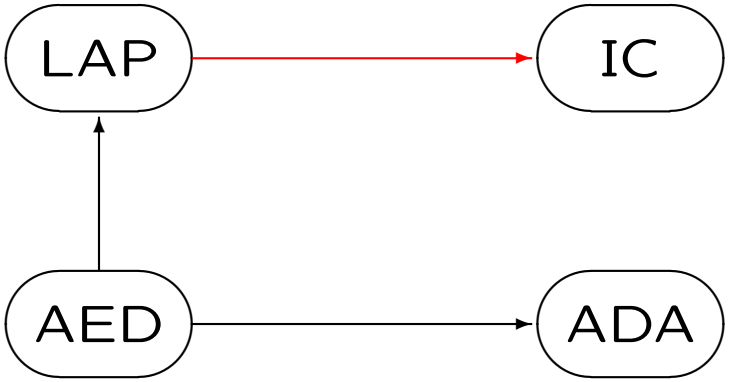



**Ordenação:** IP

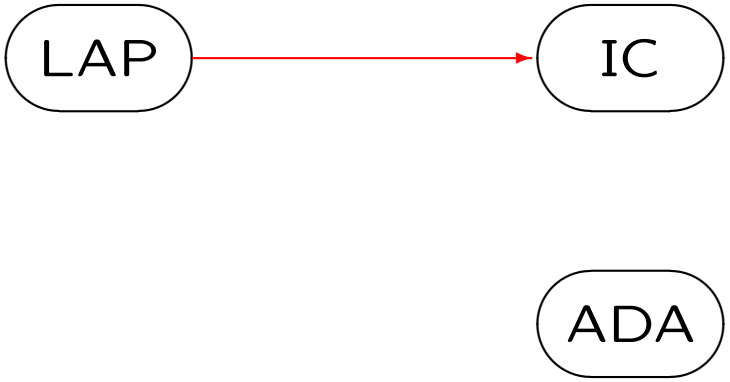



**Ordenação:** IP POO

# Exemplo (2)

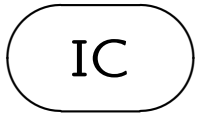



**Ordenação:** IP POO AED




**Ordenação:** IP POO AED LAP (uma das alternativas)

# Exemplo (3)

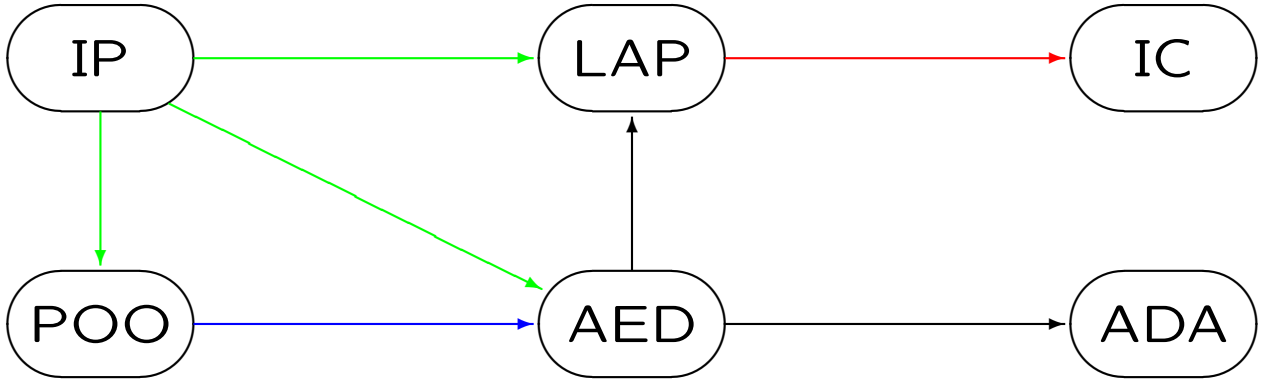




**Ordenação:** IP POO AED LAP IC (uma das alternativas)


**Ordenação:** IP POO AED LAP IC ADA

# Número de Antecessores




| | | | | | | | |
|---|---|---|---|---|---|---|---|
| LAP | 2 | 1 | 1 | **0** | — | — | — |
| IC | 1 | 1 | 1 | 1 | **0** | — | — |
| AED | 2 | 1 | **0** | — | — | — | — |
| ADA | 1 | 1 | 1 | **0** | **0** | **0** | — |
| IP | **0** | — | — | — | — | — | — |
| POO | 1 | **0** | — | — | — | — | — |
| **Ordenação:** | IP | POO | AED | LAP | IC | ADA | |

# Ordenação Topológica (1)

```
void topologicalSort( Digraph graph )
{
    Queue<Vertex> ready = new QueueIn...<Vertex>(?);
    int[] inCounter = new int[ graph.numVertices() ];

    for every Vertex v in graph.vertices()
    {
        inCounter[v] = graph.inDegree(v);
        if ( inCounter[v] == 0 )
            ready.enqueue(v);
    }
```

# Ordenação Topológica (2)

```
    while ( !ready.isEmpty() )
    {
        Vertex vertex = ready.dequeue();     TREAT(vertex);
        for every Vertex w in graph.outAdjacentVertices(vertex)
        {
            inCounter[w]--;
            if ( inCounter[w] == 0 )
                ready.enqueue(w);
        }
    }

}
```

# Complexidade

| **Implementação do Grafo** $(V, A)$ | **Ordenação Topológica** (grafo orientado e acíclico) |
|---|---|
| Matriz | $\Theta(\|V\|^2 + \|V\| \times \text{custo(TREAT)})$ |
| Vetor de Listas | $\Theta(\|V\| + \|A\| + \|V\| \times \text{custo(TREAT)})$ |

**ready**.size() $\leq |V|$

# Teste à Aciclicidade (1)

```
boolean isAcyclic( Digraph graph )
{
    Queue<Vertex> ready = new QueueIn...<Vertex>(?);
    int[] inCounter = new int[ graph.numVertices() ];

    for every Vertex v in graph.vertices()
    {
        inCounter[v] = graph.inDegree(v);
        if ( inCounter[v] == 0 )
            ready.enqueue(v);
    }
    int numSortedVertices = 0;
```

# Teste à Aciclicidade (2)

```
    while ( !ready.isEmpty() )
    {
        Vertex vertex = ready.dequeue();     TREAT(vertex);
        numSortedVertices++;
        for every Vertex w in graph.outAdjacentVertices(vertex)
        {
            inCounter[w]--;
            if ( inCounter[w] == 0 )
                ready.enqueue(w);
        }
    }

    return numSortedVertices == graph.numVertices();
}
```

# Complexidade

| **Implementação do Grafo** $(V, A)$ | **Teste à Aciclicidade** (grafo orientado) |
|---|---|
| Matriz | $O(\|V\|^2 + \|V\| \times \text{custo(TREAT)})$ |
| Vetor de Listas | $O(\|V\| + \|A\| + \|V\| \times \text{custo(TREAT)})$ |

**ready**.size() $\leq |V|$