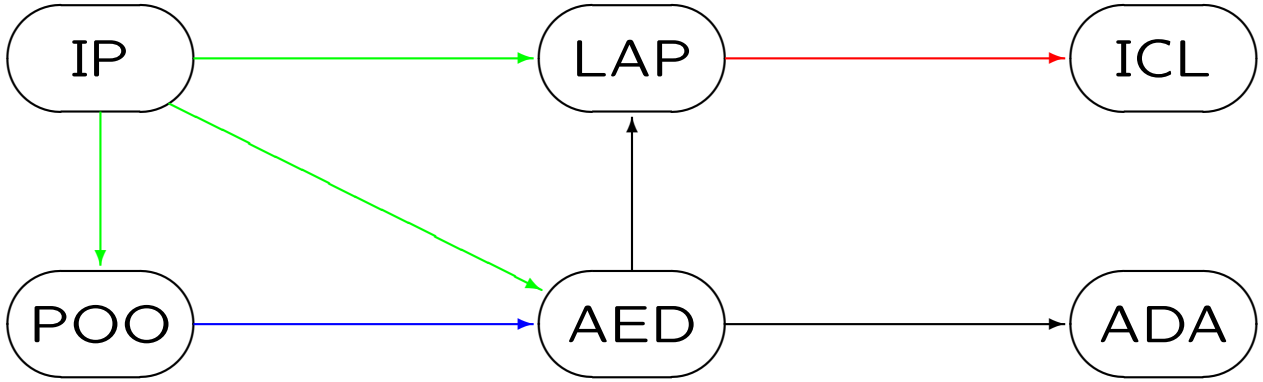
Capítulo IV

Ordenação Topológica
&
Teste à Aciclicidade
(num grafo orientado)

Ordenação Topológica

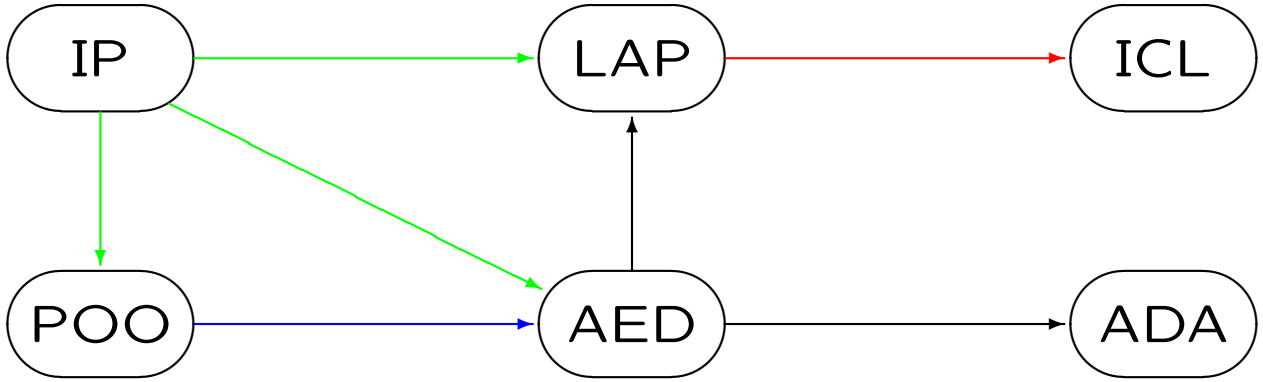
Dado um grafo $G = (V, A)$, **orientado e acíclico**, uma **ordenação topológica** de G é uma permutação de V tal que:

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

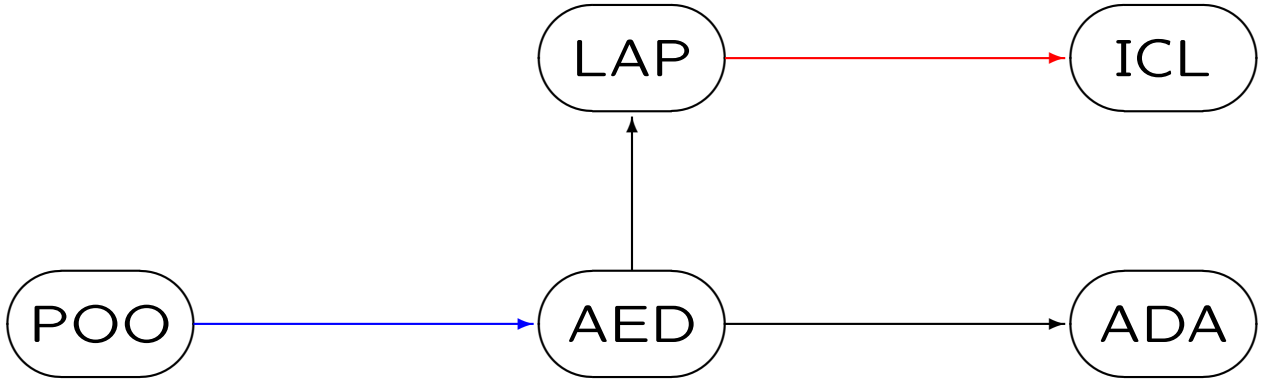


IP	POO	AED	LAP	ICL	ADA
IP	POO	AED	LAP	ADA	ICL
IP	POO	AED	ADA	LAP	ICL

Exemplo (1)

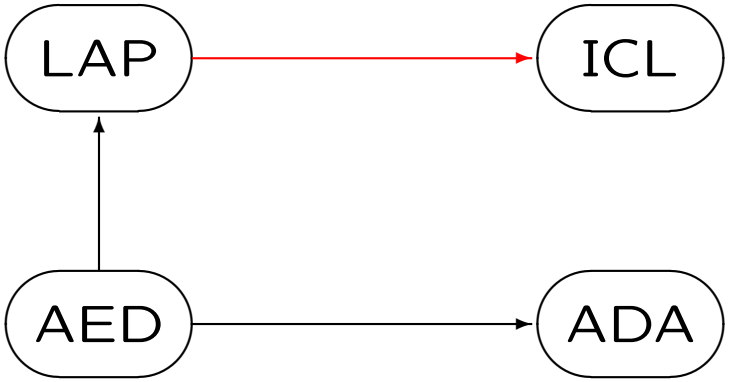


Ordenação: IP

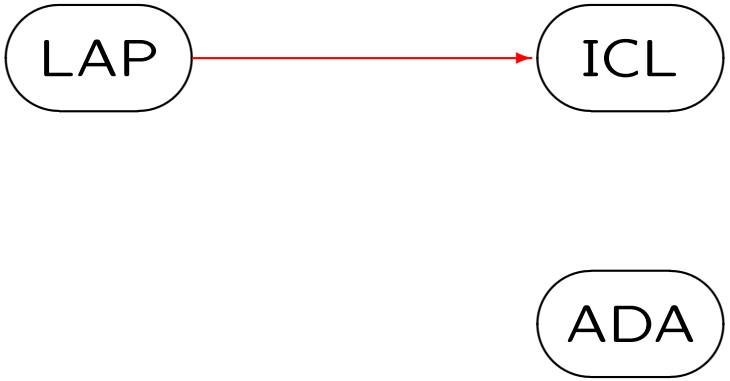


Ordenação: IP POO

Exemplo (2)



Ordenação: IP POO AED



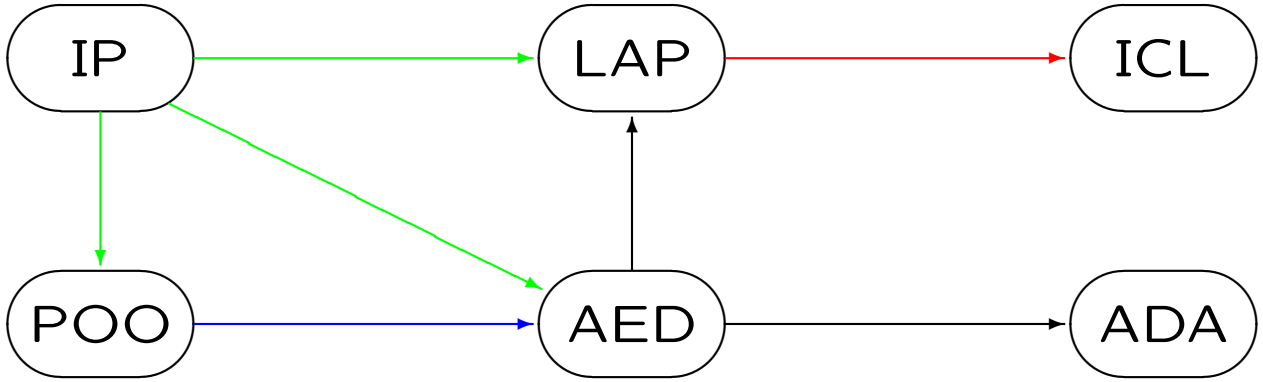
Ordenação: IP POO AED LAP (uma das alternativas)

Exemplo (3)

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

Ordenação: IP POO AED LAP ICL ADA

Número de Antecessores



LAP	2	1	1	**0**	—	—	—
ICL	1	1	1	1	**0**	—	—
AED	2	1	**0**	—	—	—	—
ADA	1	1	1	**0**	**0**	**0**	—
IP	**0**	—	—	—	—	—	—
POO	1	**0**	—	—	—	—	—
Permutação:	IP	POO	AED	LAP	ICL	ADA	

Ordenação Topológica (1)

(Topological Sorting)

```
Node[] topologicalSort( Digraph graph )
{
    Node[] permutation = new Node[ graph.numNodes() ];
    int permSize = 0;
    Bag<Node> ready = new BagIn...<Node>(?);
    int[] inDegree = new int[ graph.numNodes() ];

    for every Node v in graph.nodes()
    {
        inDegree[v] = graph.inDegree(v);
        if ( inDegree[v] == 0 )
            ready.add(v);
    }
```

Ordenação Topológica (2)

```
    do {
            Node node = ready.remove();
            permutation[ permSize++ ] = node;
            for every Node v in graph.outAdjacentNodes(node)
            {
                inDegree[v]--;
                if ( inDegree[v] == 0 )
                    ready.add(v);
            }
        }
    while ( !ready.isEmpty() );

    return permutation;
}
```

Complexidades

(se add, remove e isEmpty de Bag forem $\Theta(1)$)

Implementação do Grafo (V, A)	**Ordenação Topológica** (grafo orientado e acíclico)
Matriz	$\Theta(\|V\|^2)$
Vetor de Listas	$\Theta(\|V\| + \|A\|)$

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

Teste à Aciclicidade (1)

(Acyclicity Checking)

```
boolean isAcyclic( Digraph graph )
{
    int numProcNodes = 0;
    Bag<Node> ready = new BagIn...<Node>(?);
    int[] inDegree = new int[ graph.numNodes() ];

    for every Node v in graph.nodes()
    {
        inDegree[v] = graph.inDegree(v);
        if ( inDegree[v] == 0 )
            ready.add(v);
    }
```

Teste à Aciclicidade (2)

```
    while ( !ready.isEmpty() )
    {
        Node node = ready.remove();
        numProcNodes++;
        for every Node v in graph.outAdjacentNodes(node)
        {
            inDegree[v]--;
            if ( inDegree[v] == 0 )
                ready.add(v);
        }
    }
    return numProcNodes == graph.numNodes();
}
```

Complexidades

(se add, remove e isEmpty de Bag forem $\Theta(1)$)

Implementação do Grafo (V, A)	**Teste à Aciclicidade** (grafo orientado)
Matriz	$O(\|V\|^2)$
Vetor de Listas	$O(\|V\| + \|A\|)$

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