Capítulo V

Ordenação Topológica
(num grafo orientado e acíclico)

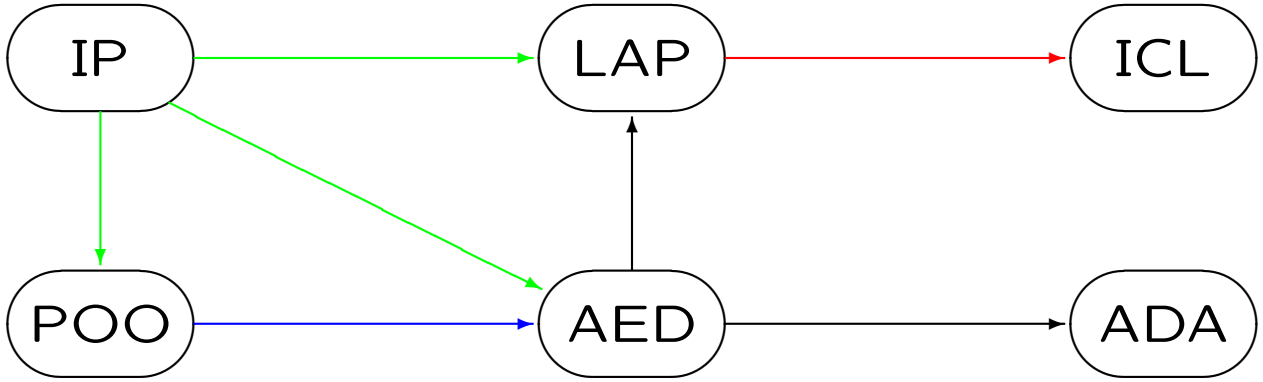
&

Teste à Aciclicidade
(num grafo orientado)

Ordenação Topológica

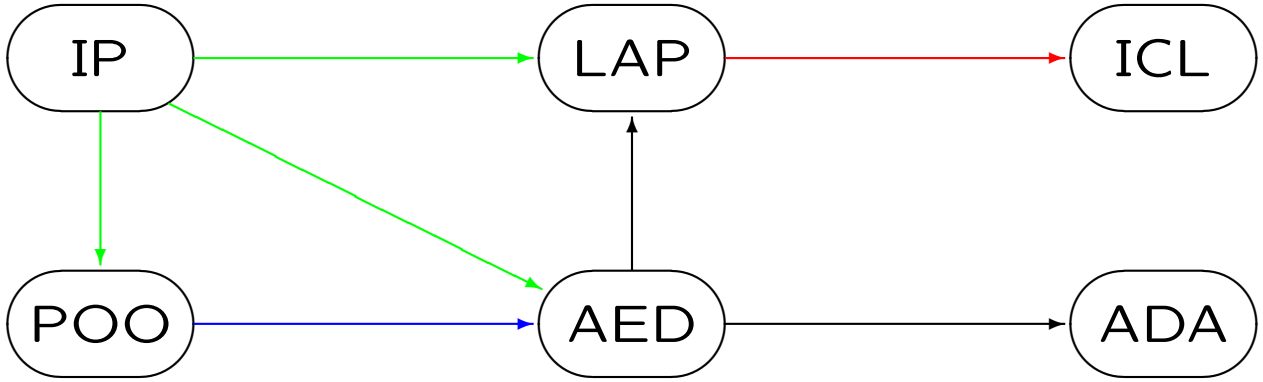
Dado um grafo $G = (V, A)$, **orientado e acíclico**, uma **ordenação topológica** de G é uma permutação de V tal que:

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

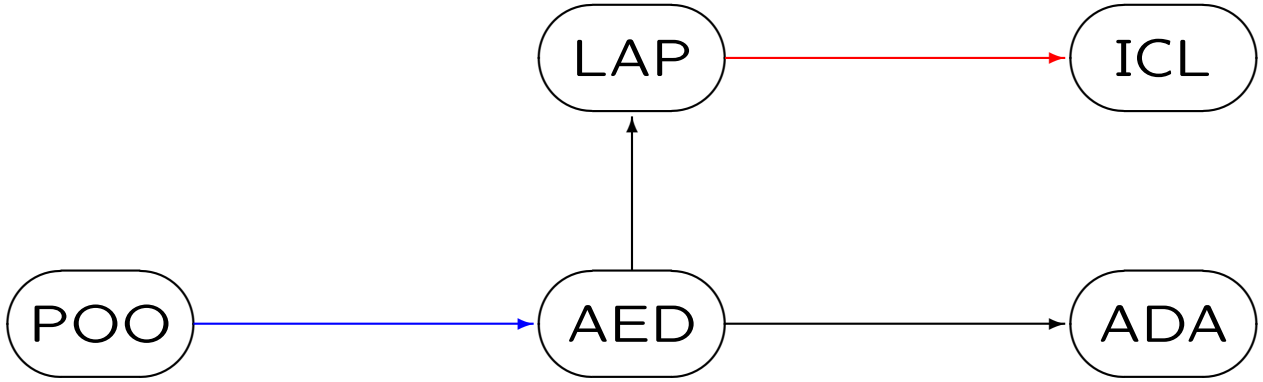


IP	POO	AED	LAP	ICL	ADA
IP	POO	AED	LAP	ADA	ICL
IP	POO	AED	ADA	LAP	ICL

Exemplo (1)

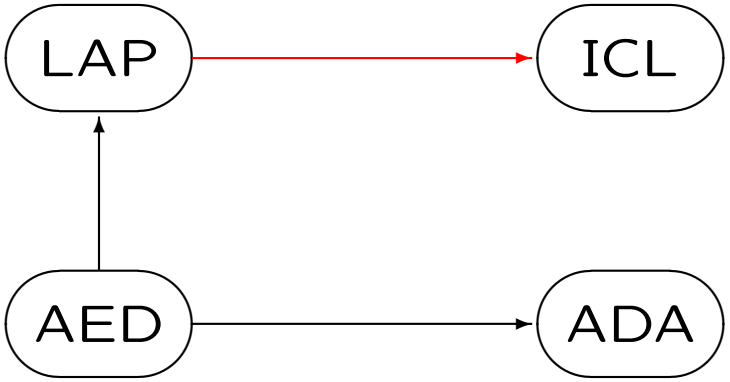


Ordenação: IP

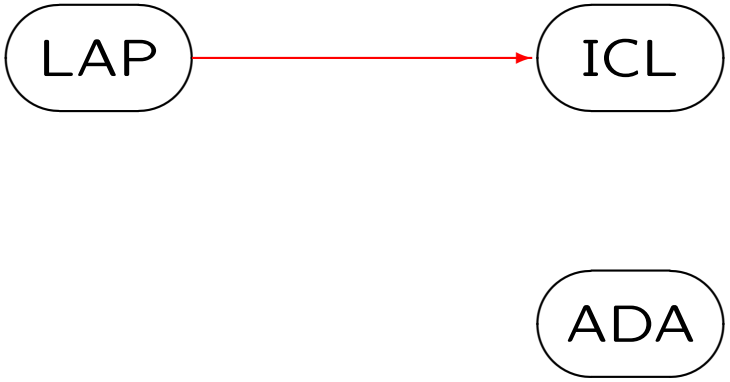


Ordenação: IP POO

Exemplo (2)



Ordenação: IP POO AED



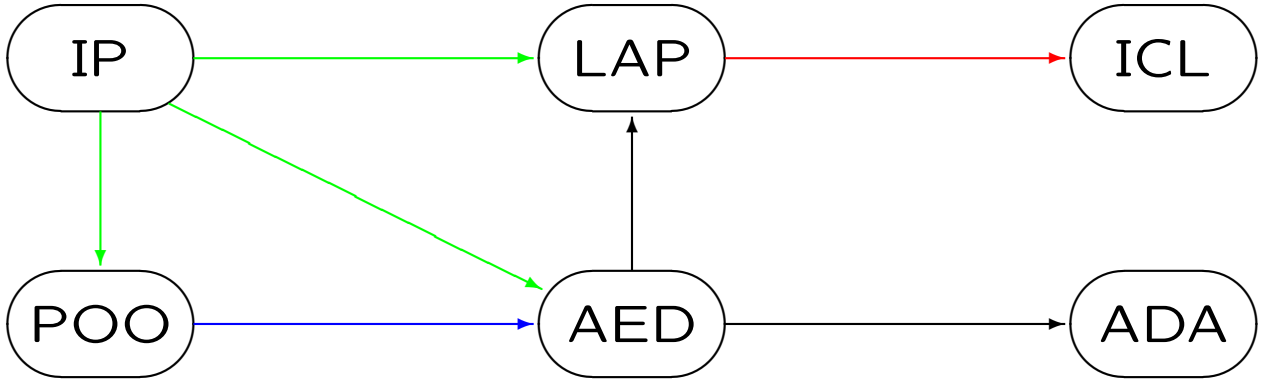
Ordenação: IP POO AED LAP (uma das alternativas)

Exemplo (3)

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

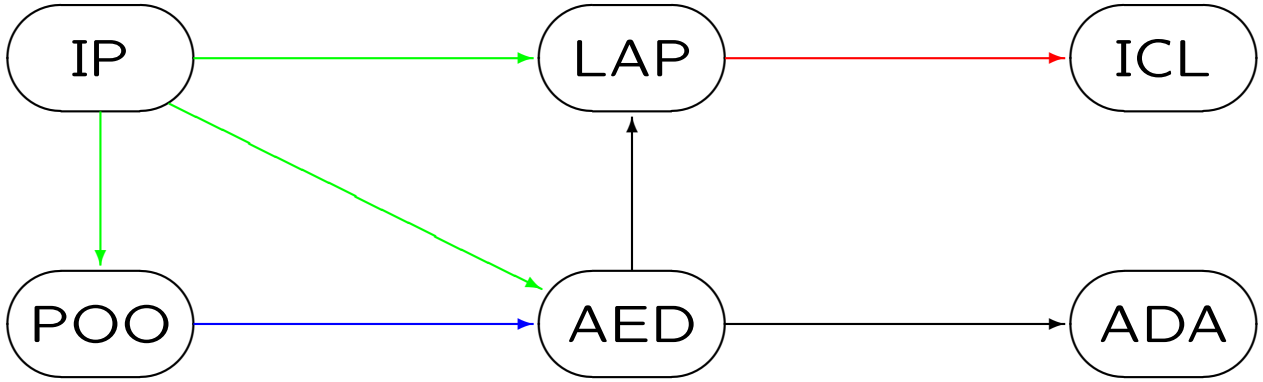
Ordenação: IP POO AED LAP ICL ADA

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	

Número de Antecessores e Saco



LAP	2	1	1	**0**	0	0	0
ICL	1	1	1	1	**0**	0	0
AED	2	1	**0**	0	0	0	0
ADA	1	1	1	**0**	0	0	0
IP	**0**	0	0	0	0	0	0
POO	1	**0**	0	0	0	0	0
Saco:	IP	POO	AED	ADA, LAP	ADA, ICL	ADA	$\emptyset$
Perm:		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...<>(?);
    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;
}
```

Complexidade Temporal de **topologicalSort**

Grafo em matriz de adjacências (suc.)

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

Criação de permutation, ready e inDegree	$\Theta(1)$
1º ciclo (executado $\forall w \in V$)	$\Theta(\|V\|^2)$
• inicialização de inDegree[w]: $\Theta(\|V\|)$	
• teste e possível inserção de w em ready: $\Theta(1)$	
2º ciclo (executado $\forall w \in V$)	$\Theta(\|V\|^2)$
• remoção de w de ready: $\Theta(1)$	
• inserção de w em permutation: $\Theta(1)$	
• iteração dos sucessores de w: $\Theta(\|V\|)$	
• tratamento de cada sucessor de w: $\Theta(1)$	
TOTAL	$\Theta(\|V\|^2)$

Complexidade Temporal de **topologicalSort**

Grafo em vetor de listas de adjacências (suc.) e vetor de inteiros (com os graus de entrada)

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

Criação de permutation, ready e inDegree	$\Theta(1)$
1º ciclo (executado $\forall w \in V$)	$\Theta(\|V\|)$
• inicialização de inDegree[w]: $\Theta(1)$	
• teste e possível inserção de w em ready: $\Theta(1)$	
2º ciclo (executado $\forall w \in V$)	$\Theta(\|V\| + \|A\|)$
• remoção de w de ready: $\Theta(1)$	
• inserção de w em permutation: $\Theta(1)$	
• iteração dos sucessores de w: $\Theta(\|\text{Suc}(w)\|)$	
• tratamento de cada sucessor de w: $\Theta(1)$	
TOTAL	$\Theta(\|V\| + \|A\|)$

Complexidade Espacial de **topologicalSort**

Vetor permutation	$\Theta(\|V\|)$
Vetor inDegree	$\Theta(\|V\|)$
Saco ready	$O(\|V\|)$
TOTAL	$\Theta(\|V\|)$

Teste à Aciclicidade (1)

(Acyclicity Checking)

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

Complexidade Temporal de **isAcyclic**

Grafo em matriz de adjacências (suc.)

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

Criação de ready e inDegree	$\Theta(1)$
1º ciclo (executado $\forall w \in V$)	$\Theta(\|V\|^2)$
• inicialização de inDegree[w]: $\Theta(\|V\|)$	
• teste e possível inserção de w em ready: $\Theta(1)$	
2º ciclo (executado **no máximo** $\forall w \in V$)	$O(\|V\|^2)$
• remoção de w de ready: $\Theta(1)$	
• iteração dos sucessores de w: $\Theta(\|V\|)$	
• tratamento de cada sucessor de w: $\Theta(1)$	
TOTAL	$\Theta(\|V\|^2)$

Complexidade Temporal de **isAcyclic**

Grafo em vetor de listas de adjacências (suc.) e vetor de inteiros (com os graus de entrada)

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

Criação de ready e inDegree $\Theta(1)$

1º ciclo (executado $\forall w \in V$) $\Theta(|V|)$

- inicialização de inDegree[w]: $\Theta(1)$
- teste e possível inserção de w em ready: $\Theta(1)$

2º ciclo (executado **no máximo** $\forall w \in V$) $O(|V| + |A|)$

- remoção de w de ready: $\Theta(1)$
- iteração dos sucessores de w: $\Theta(|\text{Suc}(w)|)$
- tratamento de cada sucessor de w: $\Theta(1)$

TOTAL $O(|V| + |A|)$

Complexidade Espacial de **isAcyclic**

Vetor inDegree	$\Theta(\|V\|)$
Saco ready	$O(\|V\|)$
TOTAL	$\Theta(\|V\|)$