Aula 9 - Ordenação topológica

Luís Felipe

UFF

09 de Abril de 2026

Luís Felipe

09/04/26

Ordenação topológica

Um dígrafo G é acíclico se G não contém ciclos direcionados

- Grafo direcionado acíclico (DAG)
- Toda DAG possui vértices de grau de saída 0, chamados de sumidouro, e vértices com grau de entrada 0, chamados de fonte.

Luís Felipe

09/04/26

Ordenação topológica

Um dígrafo G é acíclico se G não contém ciclos direcionados

- Grafo direcionado acíclico (DAG)
- Toda DAG possui vértices de grau de saída 0, chamados de sumidouro, e vértices com grau de entrada 0, chamados de fonte.

Uma ordenação topológica de um dígrafo acíclico G é uma ordenação $v_1, v_2, v_3, \ldots, v_{n-1}, v_n$ dos vértices de G de modo que se existe uma aresta em G de v_i a v_j, então $i < j$.

Luís Felipe

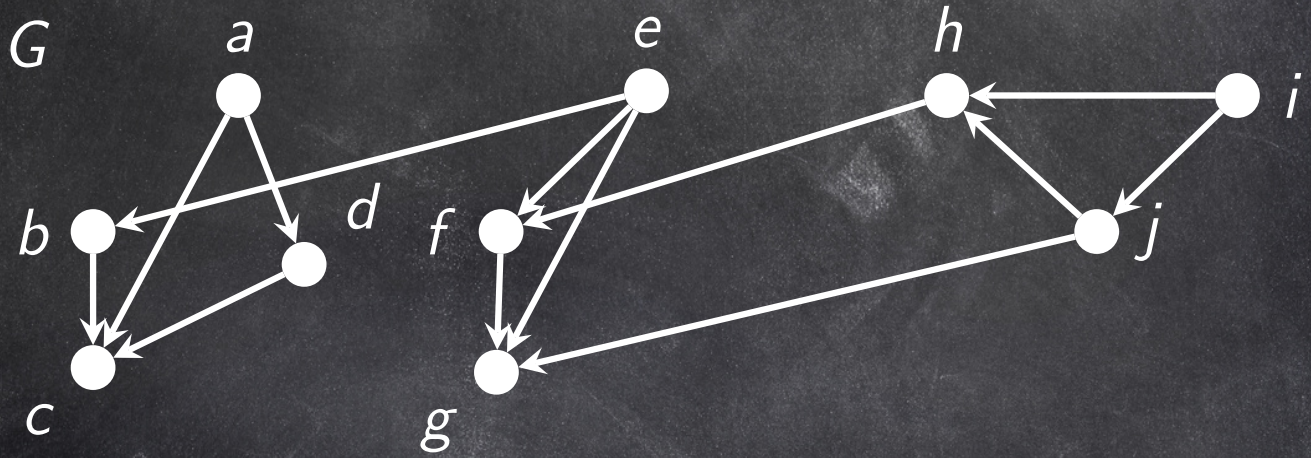
09/04/26

Ordenação topológica

Um dígrafo G é acíclico se G não contém ciclos direcionados

- Grafo direcionado acíclico (DAG)
- Toda DAG possui vértices de grau de saída 0, chamados de sumidouro, e vértices com grau de entrada 0, chamados de fonte.

Uma ordenação topológica de um dígrafo acíclico G é uma ordenação $v_1, v_2, v_3, \ldots, v_{n-1}, v_n$ dos vértices de G de modo que se existe uma aresta em G de v_i a v_j, então $i < j$.



G é acíclico.

Luís Felipe

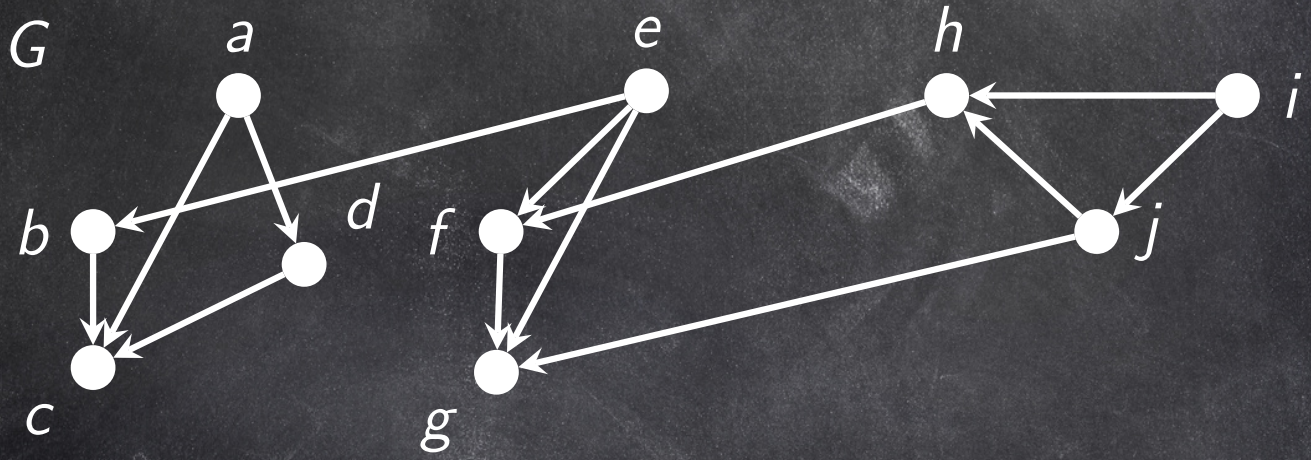
09/04/26

Ordenação topológica

Um dígrafo G é acíclico se G não contém ciclos direcionados

- Grafo direcionado acíclico (DAG)
- Toda DAG possui vértices de grau de saída 0, chamados de sumidouro, e vértices com grau de entrada 0, chamados de fonte.

Uma ordenação topológica de um dígrafo acíclico G é uma ordenação $v_1, v_2, v_3, \ldots, v_{n-1}, v_n$ dos vértices de G de modo que se existe uma aresta em G de v_i a v_j, então $i < j$.



G é acíclico. Exemplos de ordenação topológica:

Luís Felipe

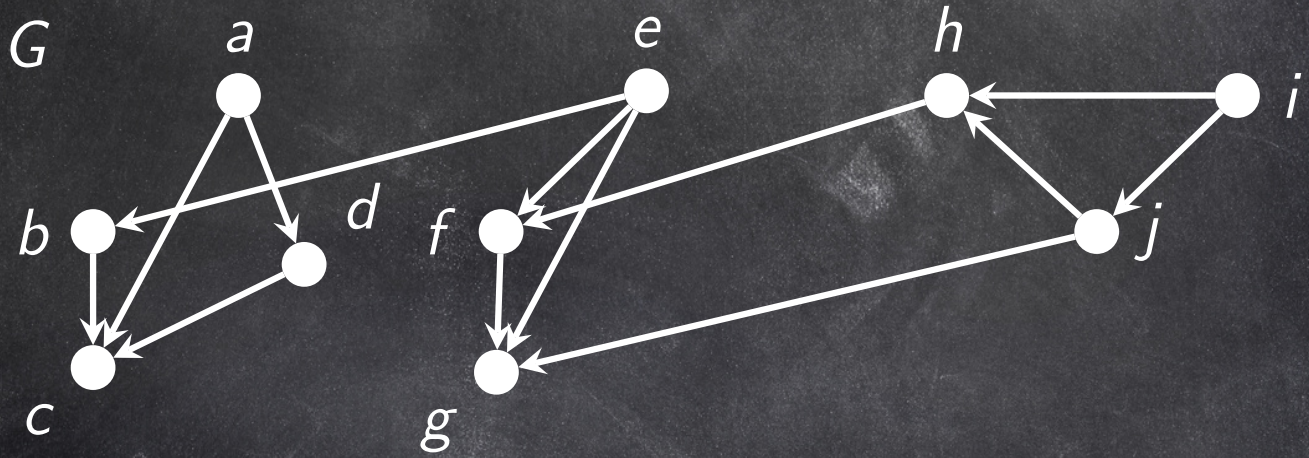
09/04/26

Ordenação topológica

Um dígrafo G é acíclico se G não contém ciclos direcionados

- Grafo direcionado acíclico (DAG)
- Toda DAG possui vértices de grau de saída 0, chamados de sumidouro, e vértices com grau de entrada 0, chamados de fonte.

Uma ordenação topológica de um dígrafo acíclico G é uma ordenação $v_1, v_2, v_3, \ldots, v_{n-1}, v_n$ dos vértices de G de modo que se existe uma aresta em G de v_i a v_j, então $i < j$.



G é acíclico. Exemplos de ordenação topológica:
i j h e f g a b d c

Luís Felipe

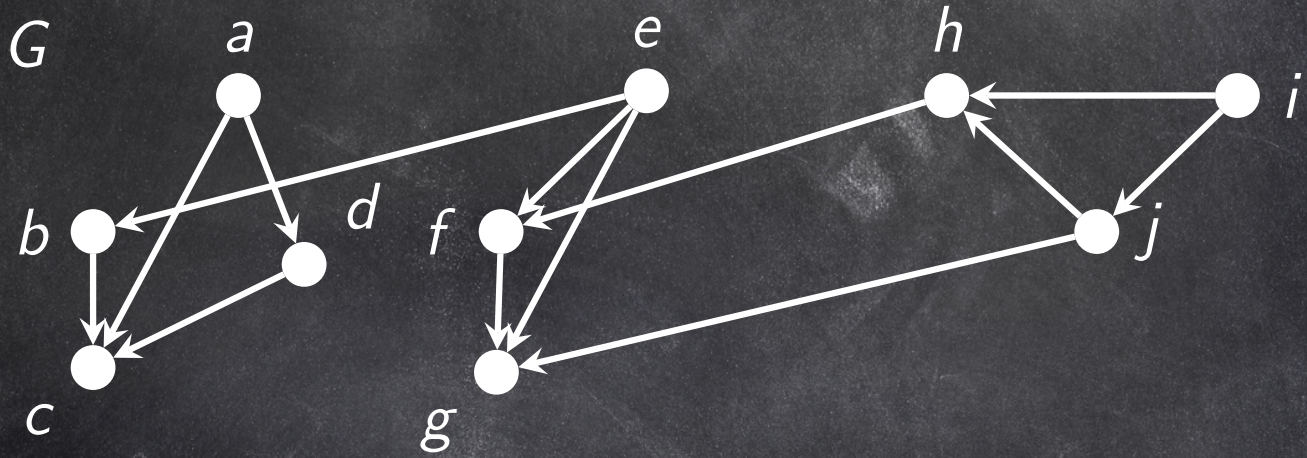
09/04/26

Ordenação topológica

Um dígrafo G é acíclico se G não contém ciclos direcionados

- Grafo direcionado acíclico (DAG)
- Toda DAG possui vértices de grau de saída 0, chamados de sumidouro, e vértices com grau de entrada 0, chamados de fonte.

Uma ordenação topológica de um dígrafo acíclico G é uma ordenação $v_1, v_2, v_3, \ldots, v_{n-1}, v_n$ dos vértices de G de modo que se existe uma aresta em G de v_i a v_j, então $i < j$.



G é acíclico. Exemplos de ordenação topológica:

$i\ j\ h\ e\ f\ g\ a\ b\ d\ c$

$e\ i\ j\ h\ a\ f\ g\ d\ b\ c$

Luís Felipe

09/04/26

Aplicações

Aplicações da ordenação topológica:

- Escalonamento de atividades: cada vértice representa uma atividade, e cada aresta direcionada de a para b indica que a atividade b só pode ser executada depois da atividade a.

Luís Felipe

09/04/26

Aplicações

Aplicações da ordenação topológica:

- Escalonamento de atividades: cada vértice representa uma atividade, e cada aresta direcionada de a para b indica que a atividade b só pode ser executada depois da atividade a.

- Escalonamento de tarefas em um único processador: cada vértice representa uma tarefa, e cada aresta direcionada de a para b indica que a tarefa a deve enviar um dado para a tarefa b (a tarefa b só pode iniciar sua execução depois que a tarefa a enviar o dado). Dado que existe um único processador para executar as tarefas, a ordenação topológica fornece uma sequência de execução das tarefas no processador.

Luís Felipe

09/04/26

Aplicações

Aplicações da ordenação topológica:

- Escalonamento de atividades: cada vértice representa uma atividade, e cada aresta direcionada de a para b indica que a atividade b só pode ser executada depois da atividade a.

- Escalonamento de tarefas em um único processador: cada vértice representa uma tarefa, e cada aresta direcionada de a para b indica que a tarefa a deve enviar um dado para a tarefa b (a tarefa b só pode iniciar sua execução depois que a tarefa a enviar o dado). Dado que existe um único processador para executar as tarefas, a ordenação topológica fornece uma sequência de execução das tarefas no processador.

Problema: Como decidir se um dado dígrafo é acíclico?

Luís Felipe

09/04/26

Aplicações

Aplicações da ordenação topológica:

- Escalonamento de atividades: cada vértice representa uma atividade, e cada aresta direcionada de a para b indica que a atividade b só pode ser executada depois da atividade a.

- Escalonamento de tarefas em um único processador: cada vértice representa uma tarefa, e cada aresta direcionada de a para b indica que a tarefa a deve enviar um dado para a tarefa b (a tarefa b só pode iniciar sua execução depois que a tarefa a enviar o dado). Dado que existe um único processador para executar as tarefas, a ordenação topológica fornece uma sequência de execução das tarefas no processador.

Problema: Como decidir se um dado dígrafo é acíclico?

Solução: Basta rodar uma busca em profundidade sobre G. Se esta busca não produzir nenhuma aresta de retorno, então G é acíclico.

Luís Felipe

09/04/26

Aplicações

Aplicações da ordenação topológica:

- Escalonamento de atividades: cada vértice representa uma atividade, e cada aresta direcionada de a para b indica que a atividade b só pode ser executada depois da atividade a.

- Escalonamento de tarefas em um único processador: cada vértice representa uma tarefa, e cada aresta direcionada de a para b indica que a tarefa a deve enviar um dado para a tarefa b (a tarefa b só pode iniciar sua execução depois que a tarefa a enviar o dado). Dado que existe um único processador para executar as tarefas, a ordenação topológica fornece uma sequência de execução das tarefas no processador.

Problema: Como decidir se um dado dígrafo é acíclico?

Solução: Basta rodar uma busca em profundidade sobre G. Se esta busca não produzir nenhuma aresta de retorno, então G é acíclico.

Obs.: Note que se uma busca sobre G não produz aresta de retorno, então nenhuma outra busca produzirá uma aresta de retorno.

Luís Felipe

09/04/26

Ordenação topológica via DFS

Algoritmo para ordenação topológica de um dígrafo G:

Luís Felipe

09/04/26

Ordenação topológica via DFS

Algoritmo para ordenação topológica de um dígrafo G:

1. Executar uma busca em profundidade (DFS) em G.

Luís Felipe

09/04/26

Ordenação topológica via DFS

Algoritmo para ordenação topológica de um dígrafo G:

1. Executar uma busca em profundidade (DFS) em G.
2. Se a busca encontrar alguma aresta de retorno, então:
   - parar o algoritmo;
   - o grafo G possui ciclo e não admite ordenação topológica.

Luís Felipe

09/04/26

Ordenação topológica via DFS

Algoritmo para ordenação topológica de um dígrafo G:

1. Executar uma busca em profundidade (DFS) em G.
2. Se a busca encontrar alguma aresta de retorno, então:
   - parar o algoritmo;
   - o grafo G possui ciclo e não admite ordenação topológica.
3. Caso contrário, considerar os tempos de saída $PS(v)$ gerados pela DFS.

Luís Felipe

09/04/26

Ordenação topológica via DFS

Algoritmo para ordenação topológica de um dígrafo G:

1. Executar uma busca em profundidade (DFS) em G.
2. Se a busca encontrar alguma aresta de retorno, então:
   - parar o algoritmo;
   - o grafo G possui ciclo e não admite ordenação topológica.
3. Caso contrário, considerar os tempos de saída $PS(v)$ gerados pela DFS.
4. Ordenar os vértices em ordem decrescente de $PS(v)$.

Luís Felipe

09/04/26

Ordenação topológica via DFS

Algoritmo para ordenação topológica de um dígrafo G:

1. Executar uma busca em profundidade (DFS) em G.
2. Se a busca encontrar alguma aresta de retorno, então:
   - parar o algoritmo;
   - o grafo G possui ciclo e não admite ordenação topológica.
3. Caso contrário, considerar os tempos de saída $PS(v)$ gerados pela DFS.
4. Ordenar os vértices em ordem decrescente de $PS(v)$.
5. Se $PS(v_1) > PS(v_2) > \cdots > PS(v_n)$, então $v_1, v_2, \ldots, v_n$ é uma ordenação topológica de G.

Luís Felipe

09/04/26

Ordenação topológica via DFS

Algoritmo para ordenação topológica de um dígrafo G:

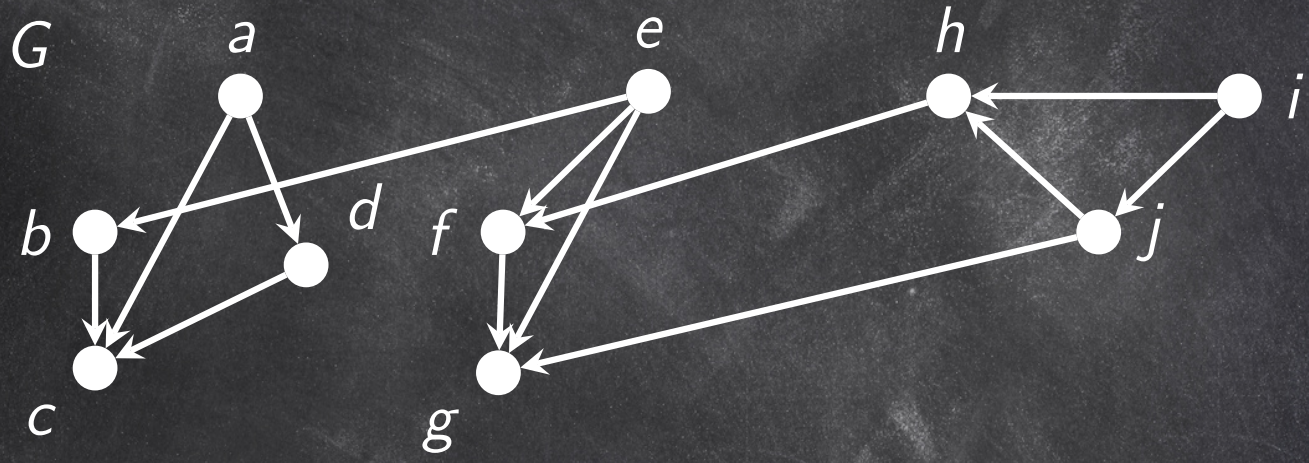
1. Executar uma busca em profundidade (DFS) em G.
2. Se a busca encontrar alguma aresta de retorno, então:
   - parar o algoritmo;
   - o grafo G possui ciclo e não admite ordenação topológica.
3. Caso contrário, considerar os tempos de saída $PS(v)$ gerados pela DFS.
4. Ordenar os vértices em ordem decrescente de $PS(v)$.
5. Se $PS(v_1) > PS(v_2) > \cdots > PS(v_n)$, então $v_1, v_2, \ldots, v_n$ é uma ordenação topológica de G.

```
1   funcao ordenacao_topologica(G):
2
3       rodar DFS em G
4
5       se existe aresta de retorno entao
6           imprimir "G possui ciclo"
7           parar
8
9       ordenar os vertices v por PS(v) em ordem decrescente
10
11      retornar lista ordenada
```

Luís Felipe

09/04/26

Exemplo

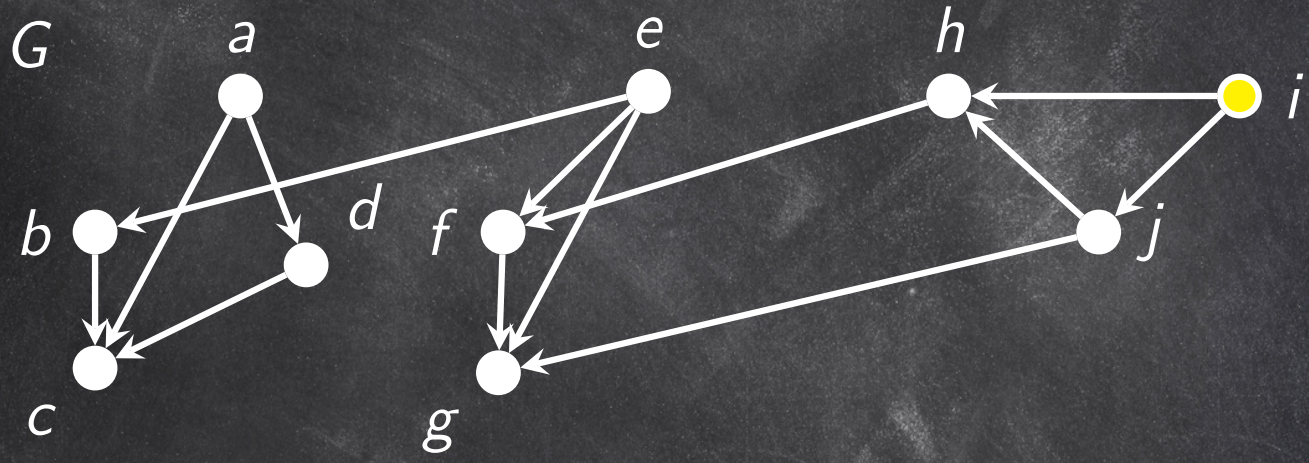


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	0	0	0	0	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

09/04/26

Exemplo

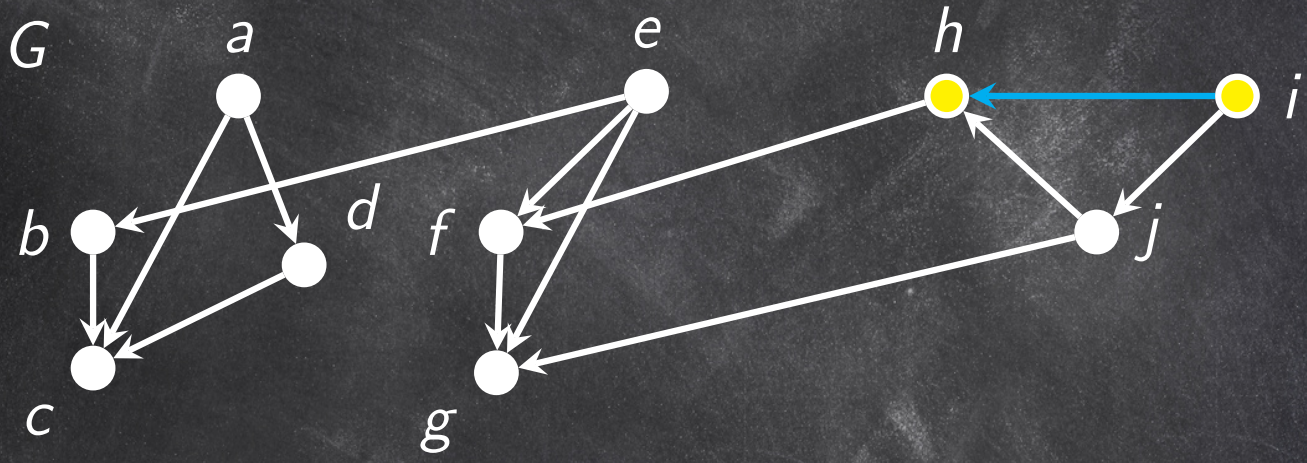


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	0	0	0	1	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

09/04/26

Exemplo

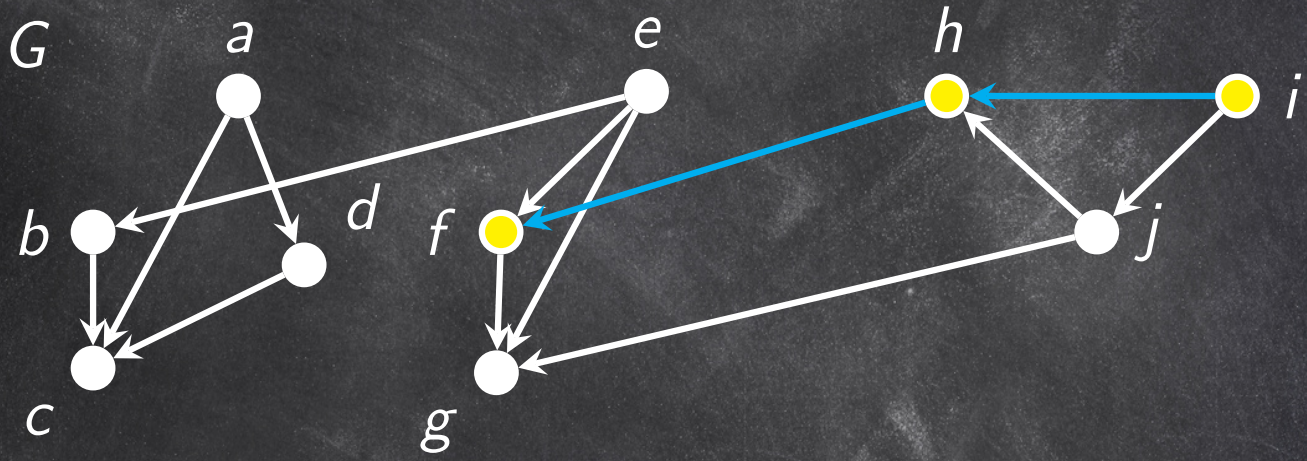


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	0	0	2	1	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

09/04/26

Exemplo

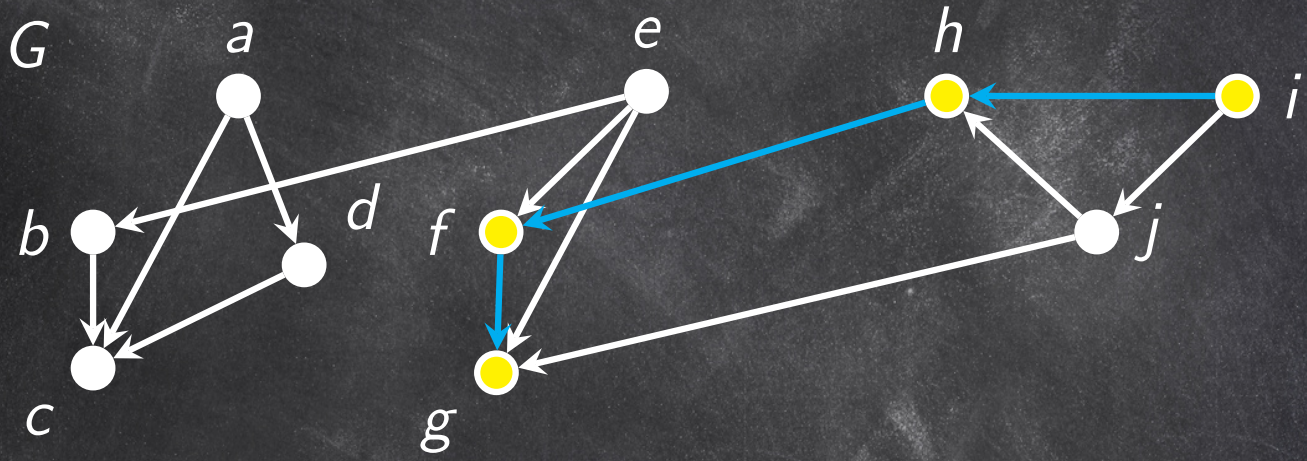


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	0	2	1	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

09/04/26

Exemplo

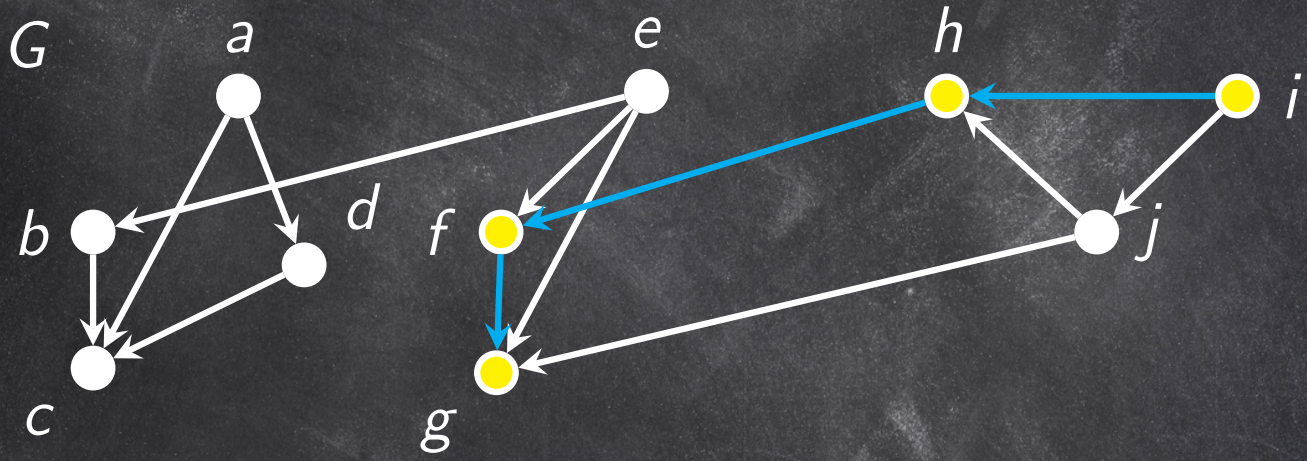


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

09/04/26

Exemplo

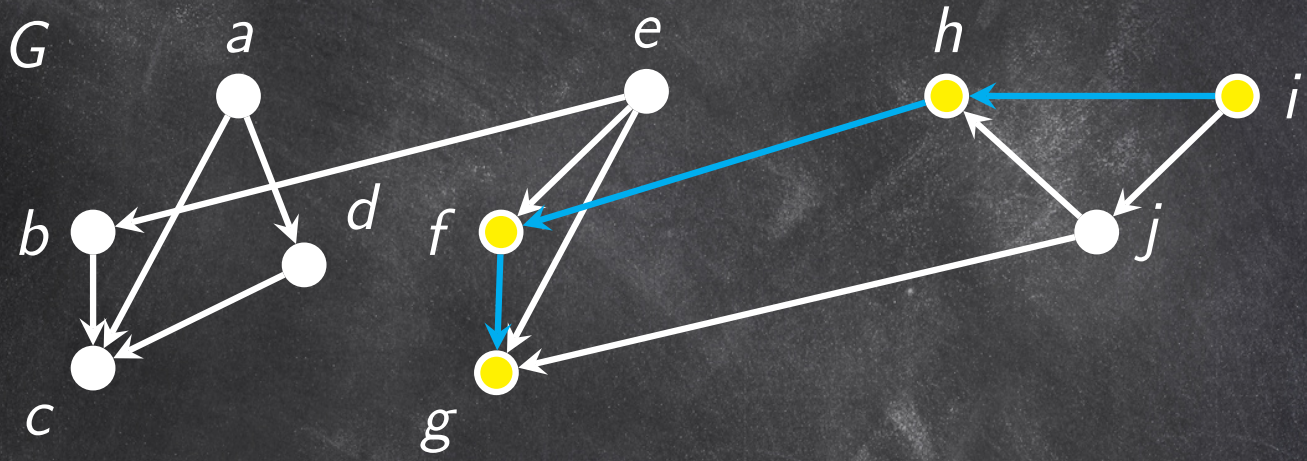


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	0
PS(v)	0	0	0	0	0	0	5	0	0	0

Luís Felipe

09/04/26

Exemplo

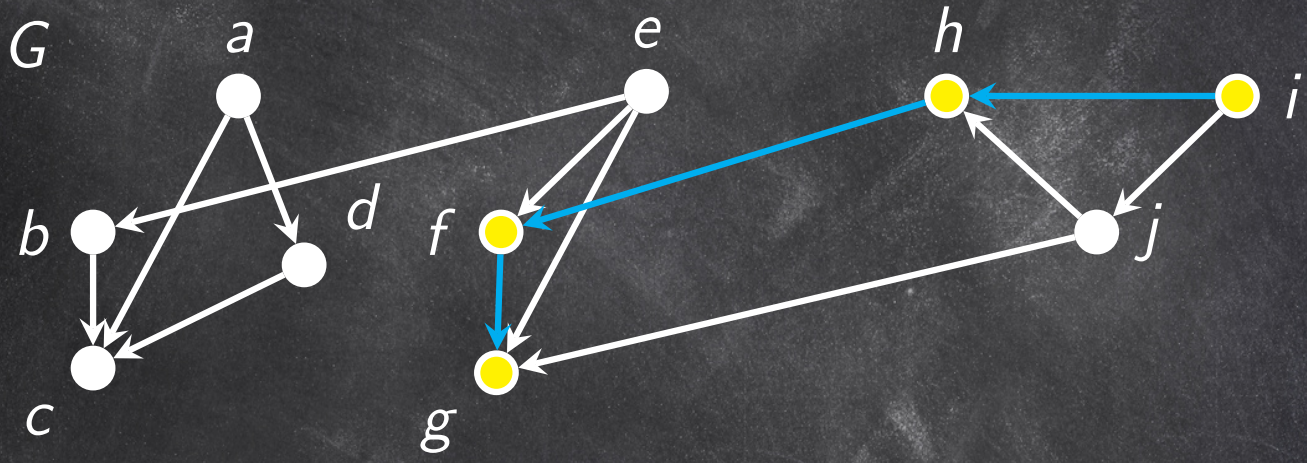


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	0
PS(v)	0	0	0	0	0	6	5	0	0	0

Luís Felipe

09/04/26

Exemplo

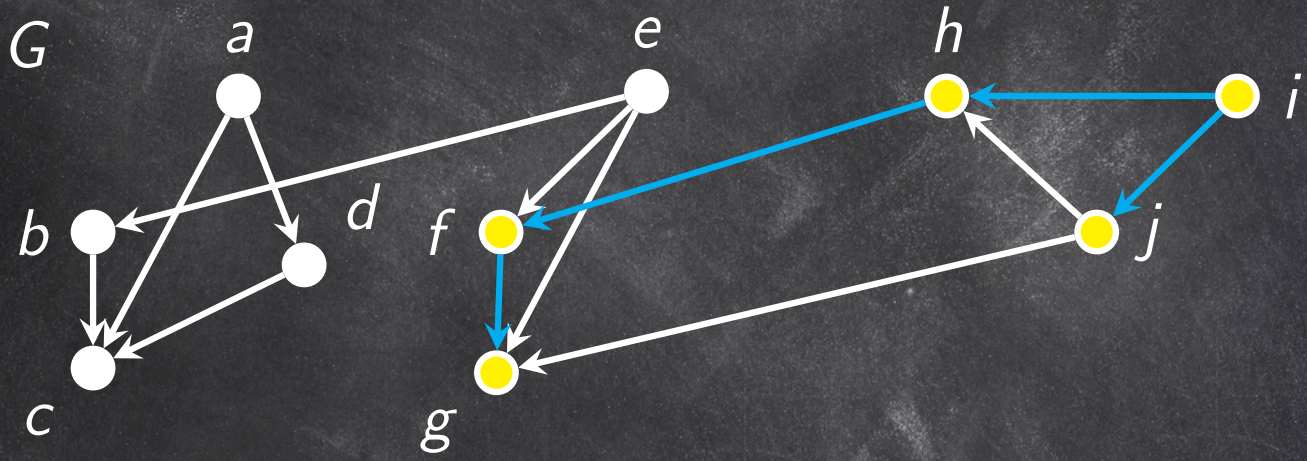


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	0
PS(v)	0	0	0	0	0	6	5	7	0	0

Luís Felipe

09/04/26

Exemplo

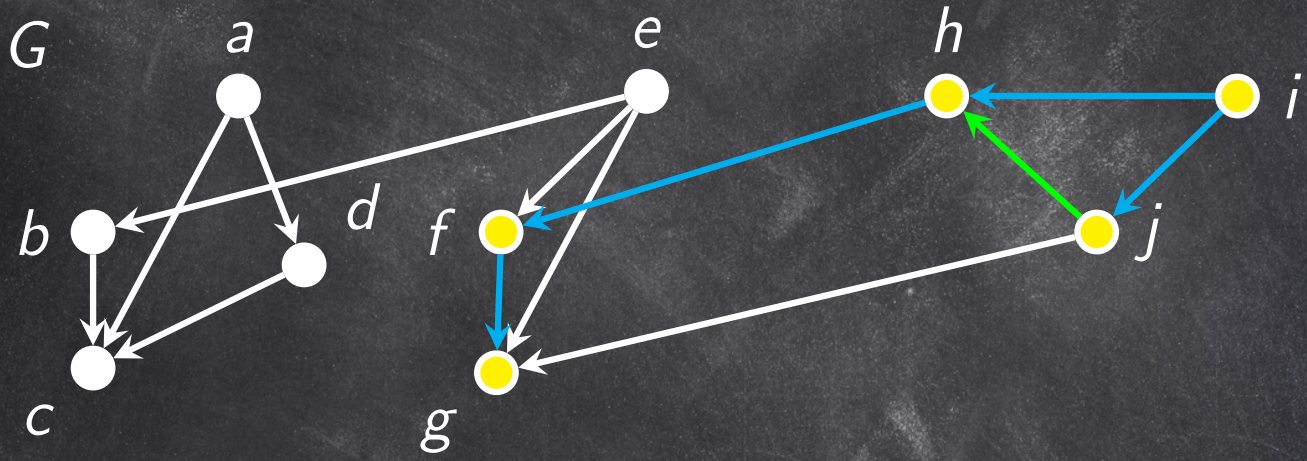


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	0	0

Luís Felipe

09/04/26

Exemplo

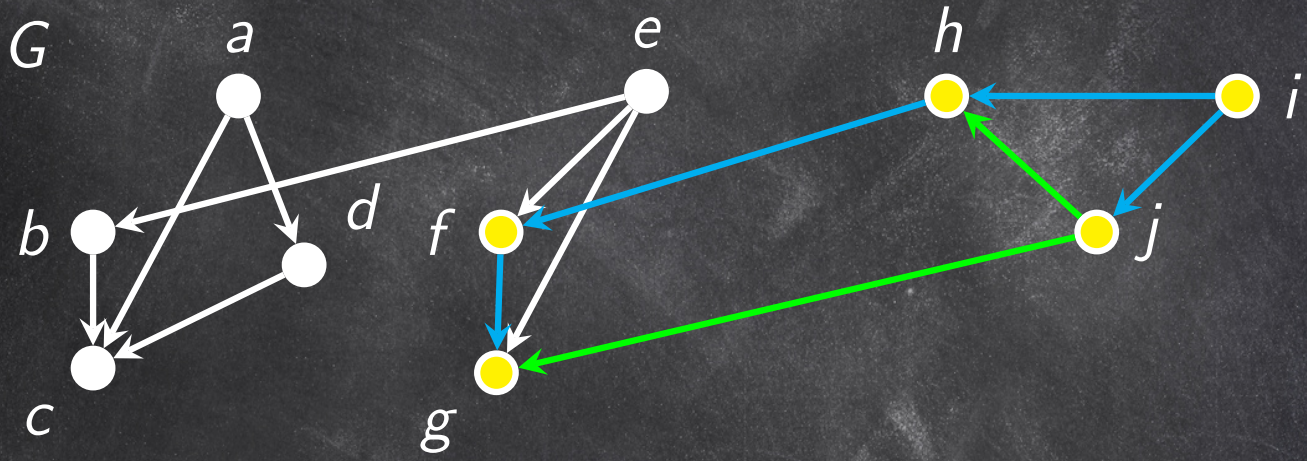


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	0	0

Luís Felipe

09/04/26

Exemplo

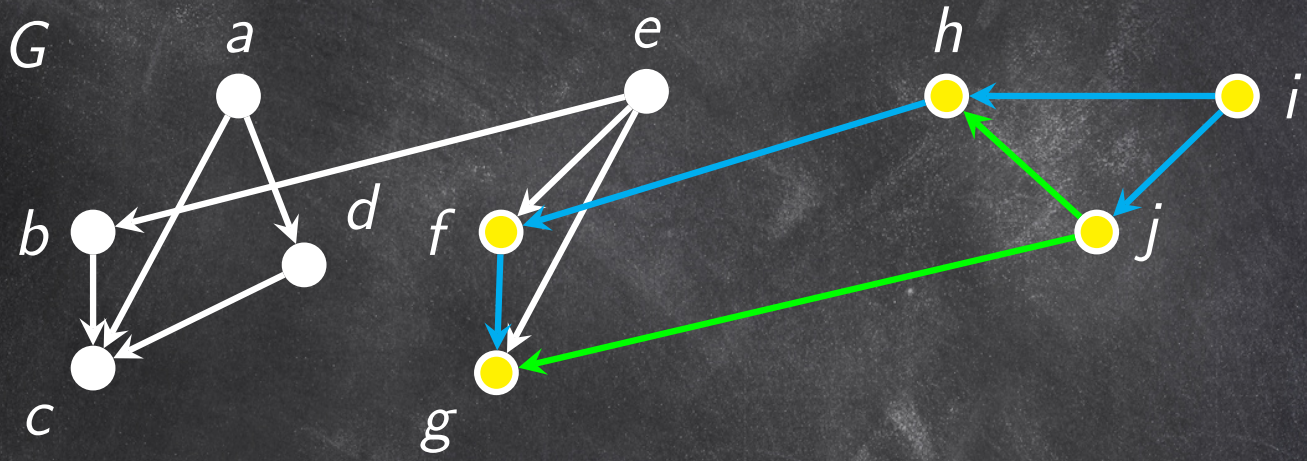


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	0	0

Luís Felipe

09/04/26

Exemplo

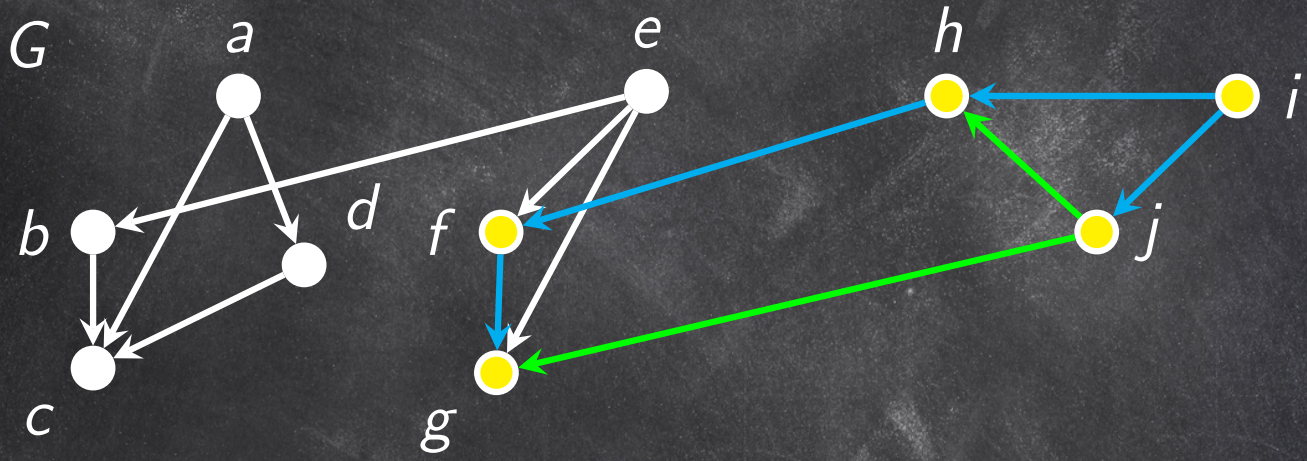


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	0	9

Luís Felipe

09/04/26

Exemplo

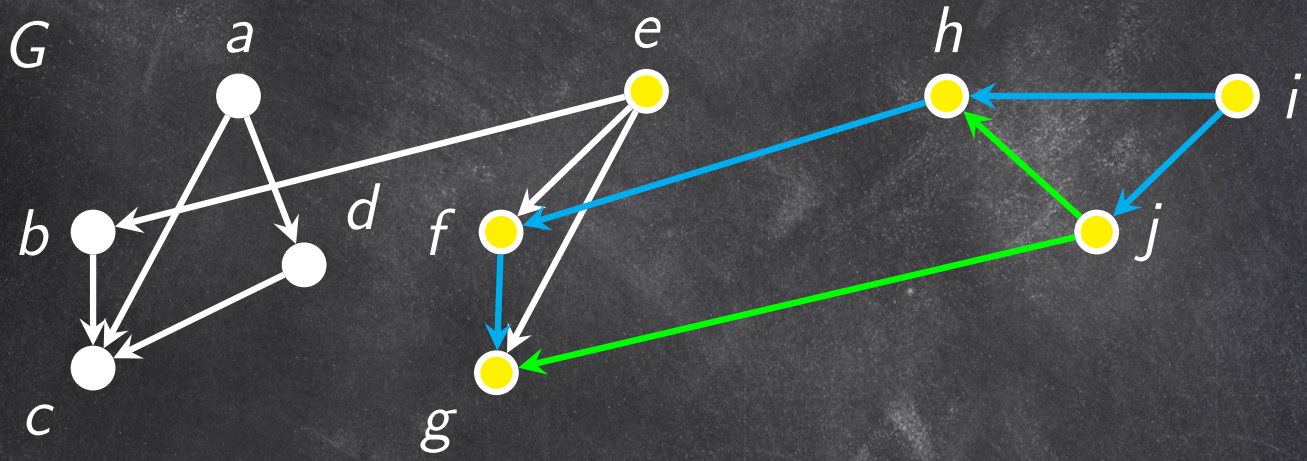


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

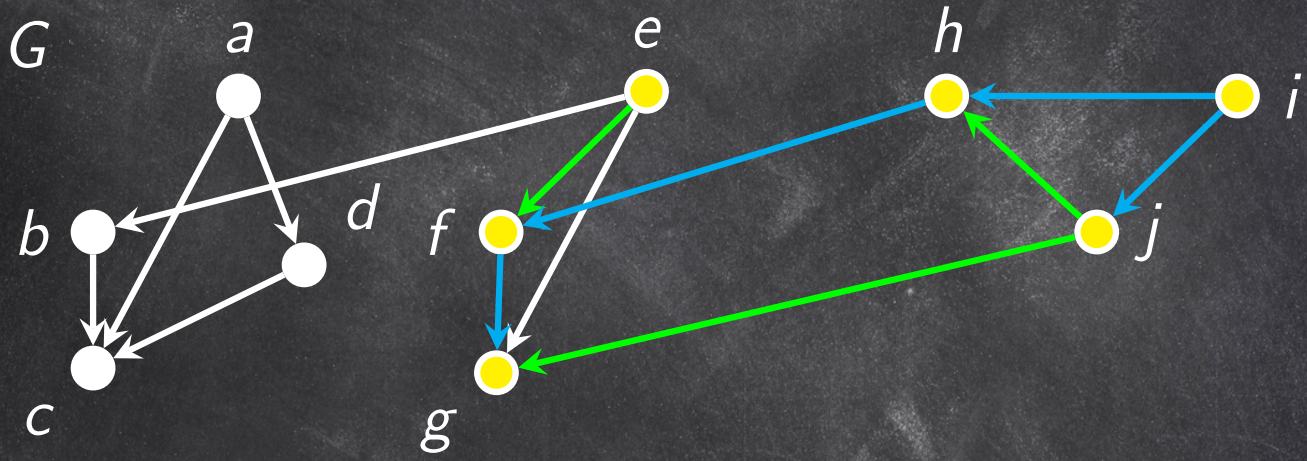


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

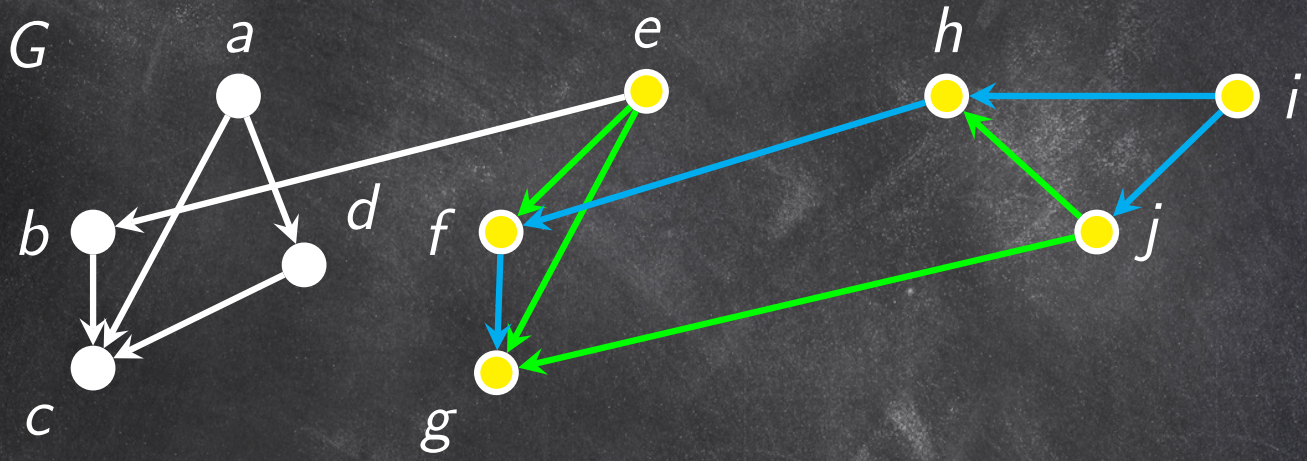


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

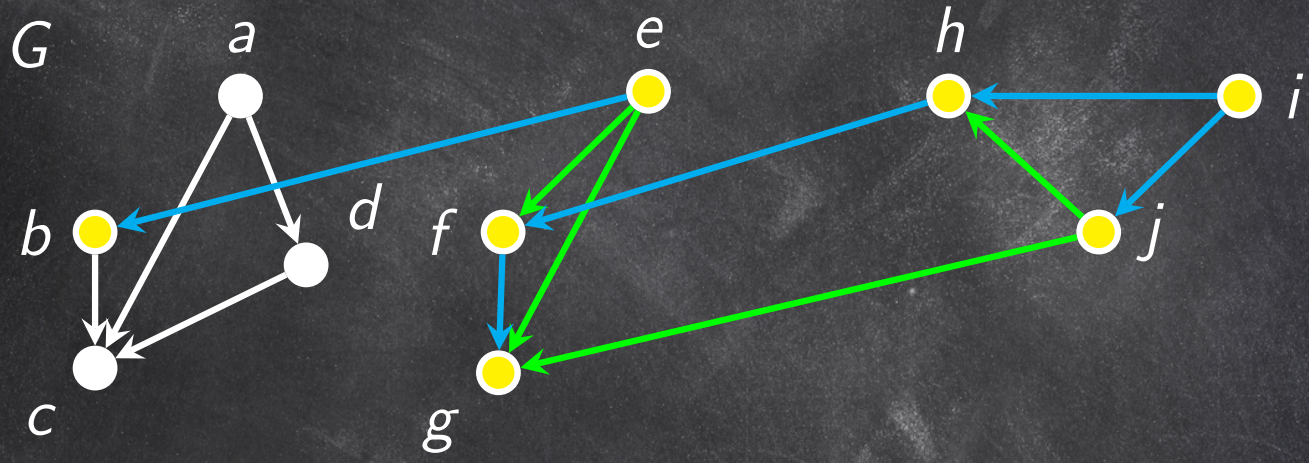


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

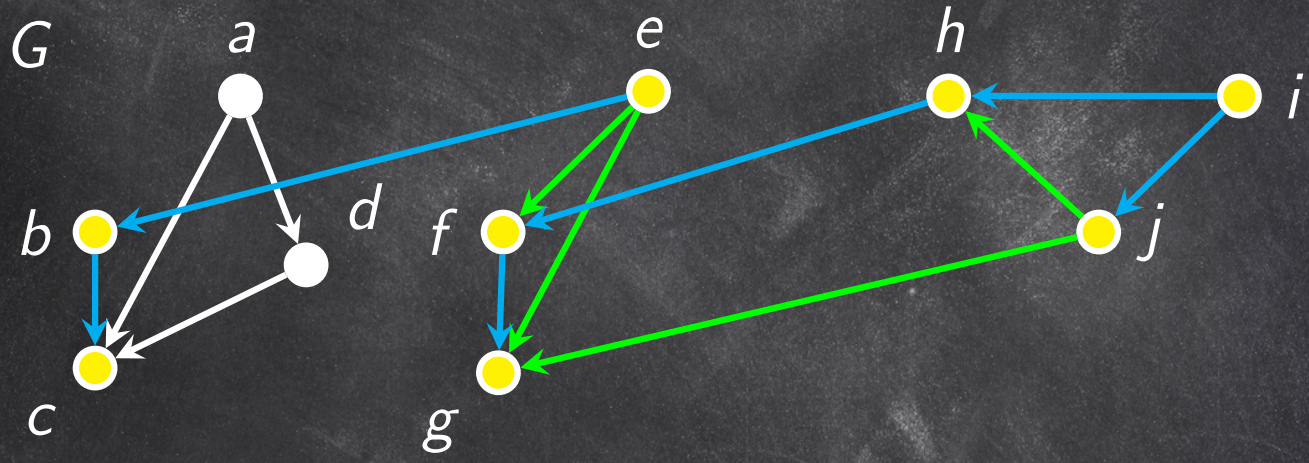


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	12	0	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

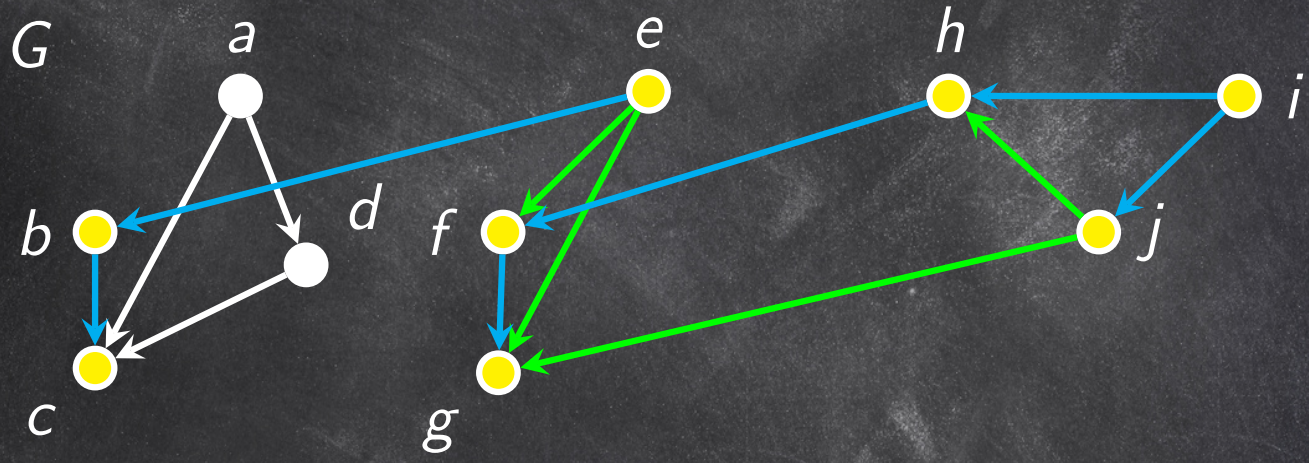


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	12	13	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

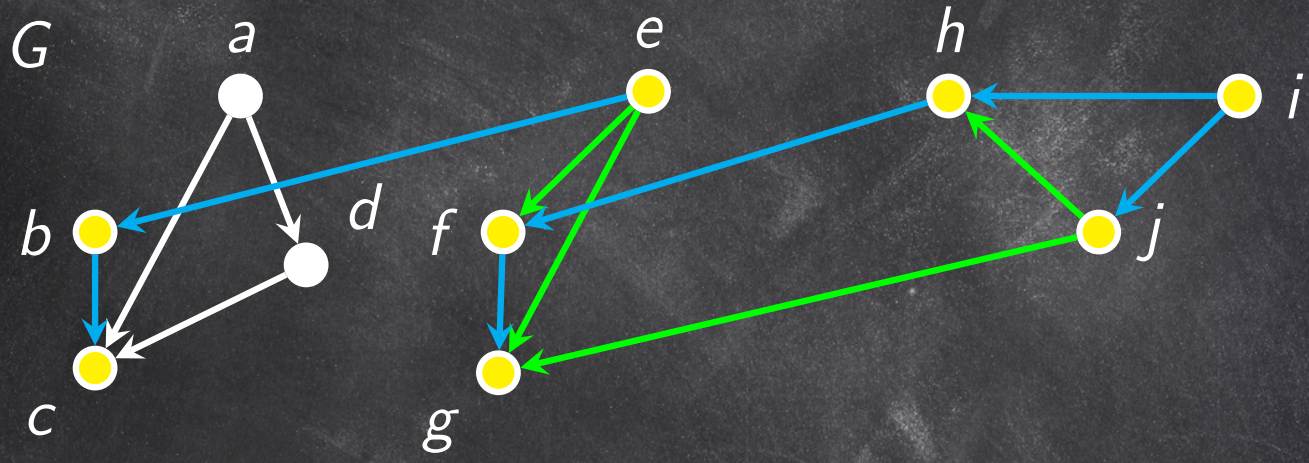


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	12	13	0	11	3	4	2	1	8
PS(v)	0	0	14	0	0	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

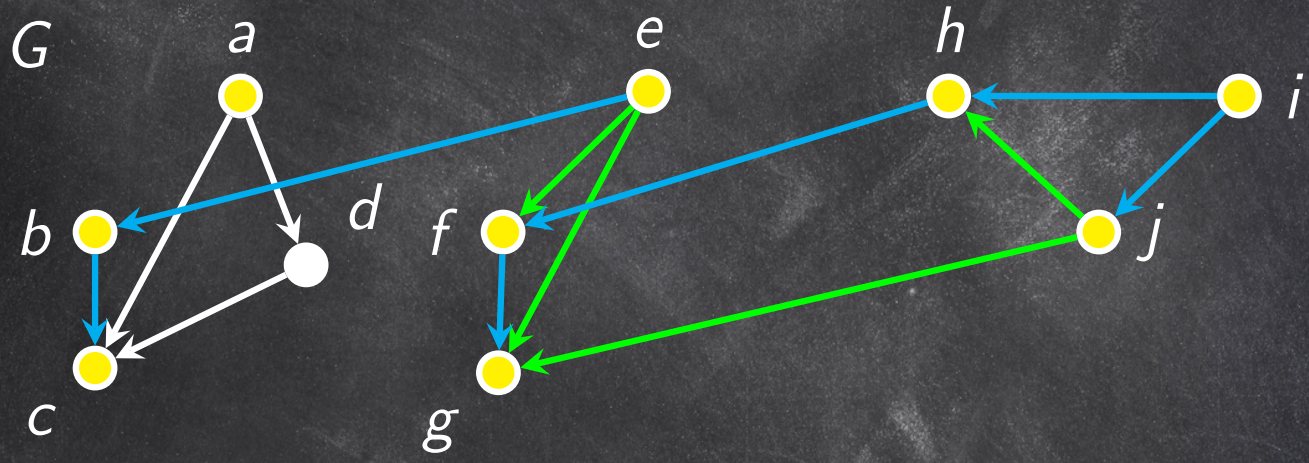


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	12	13	0	11	3	4	2	1	8
PS(v)	0	15	14	0	0	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

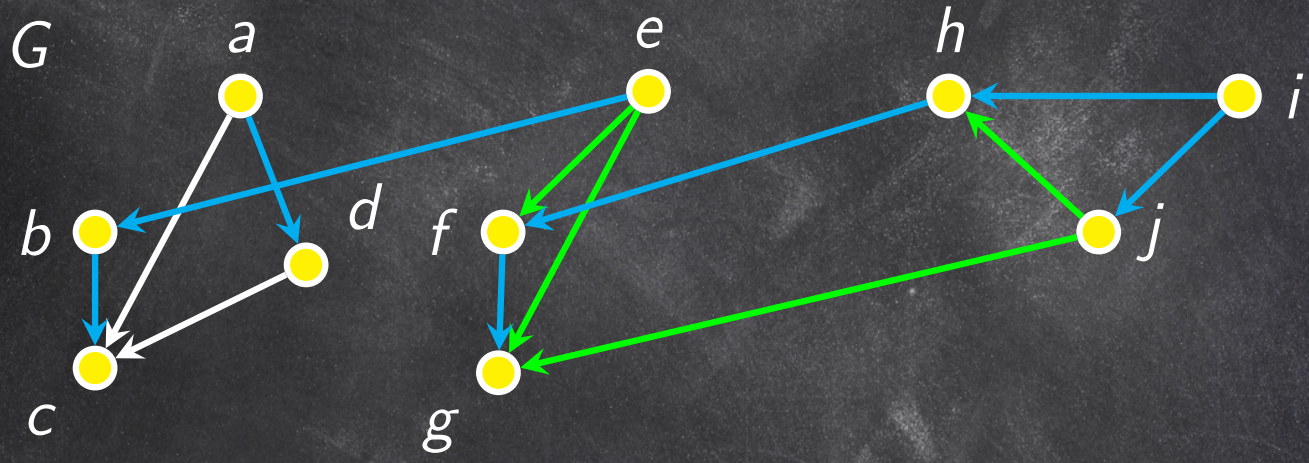


v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	0	11	3	4	2	1	8
PS(v)	0	15	14	0	16	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

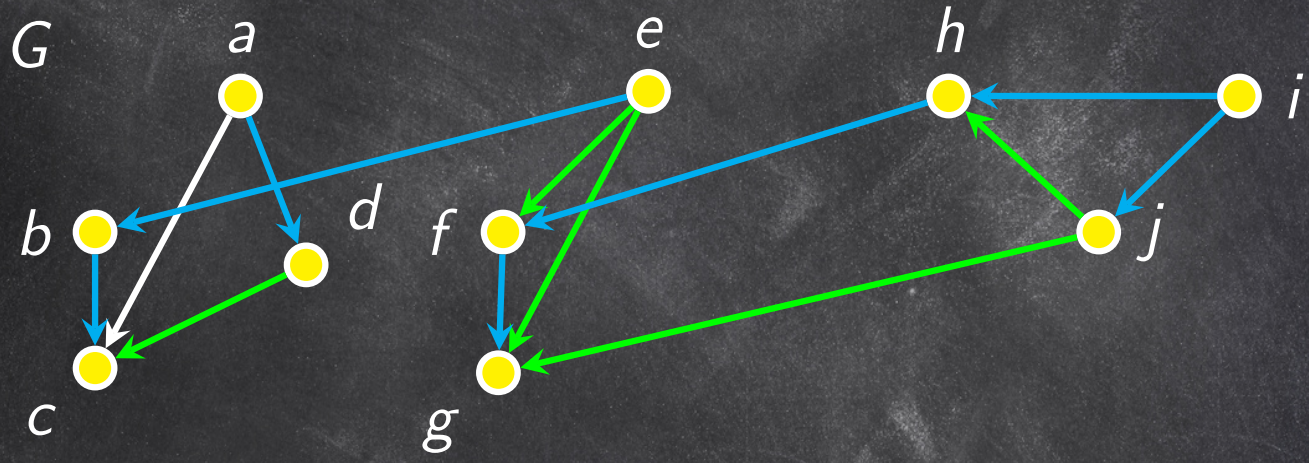


v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	0	15	14	0	16	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

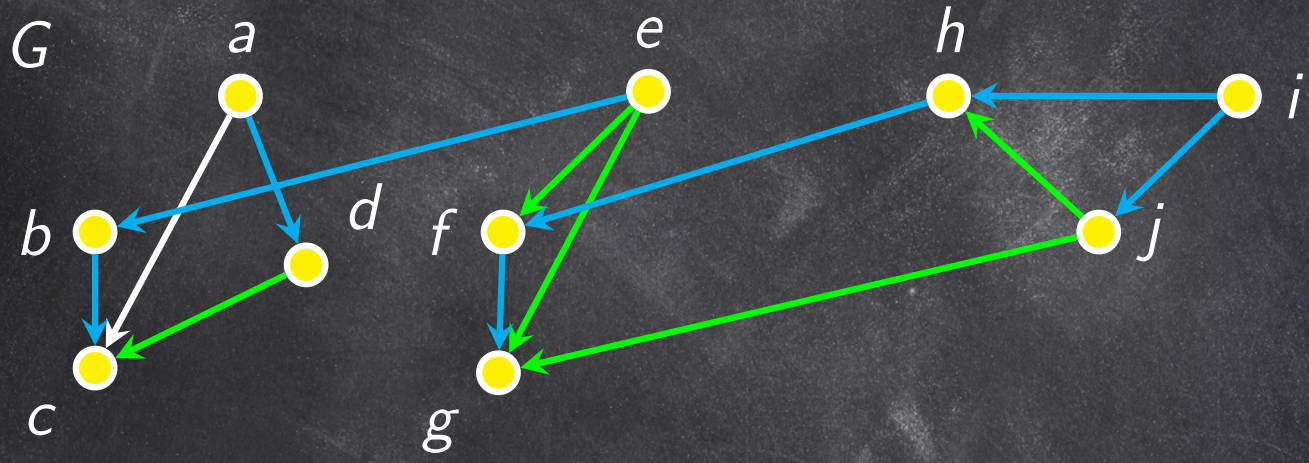


v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	0	15	14	0	16	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

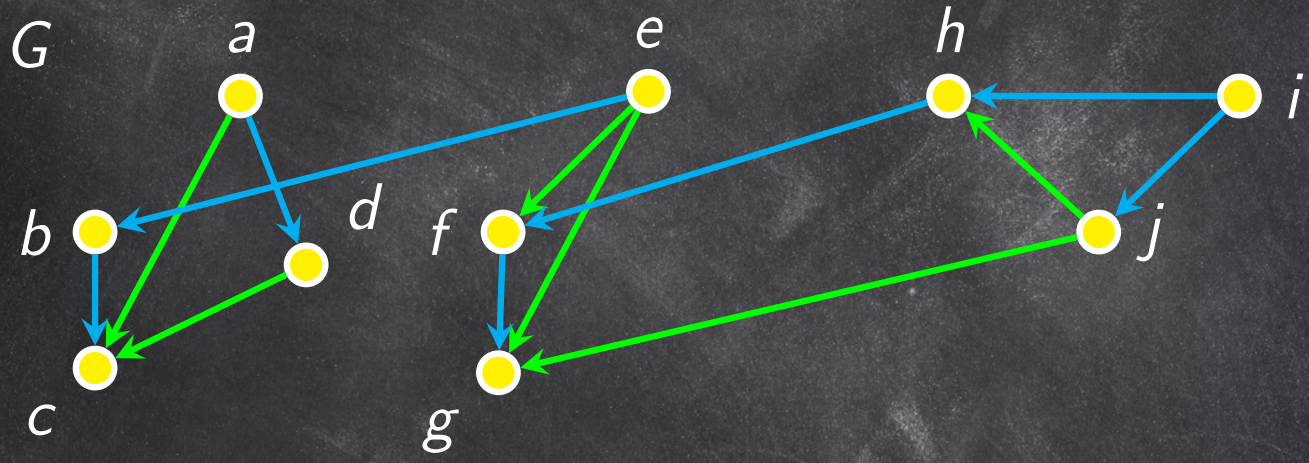


v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	0	15	14	19	16	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

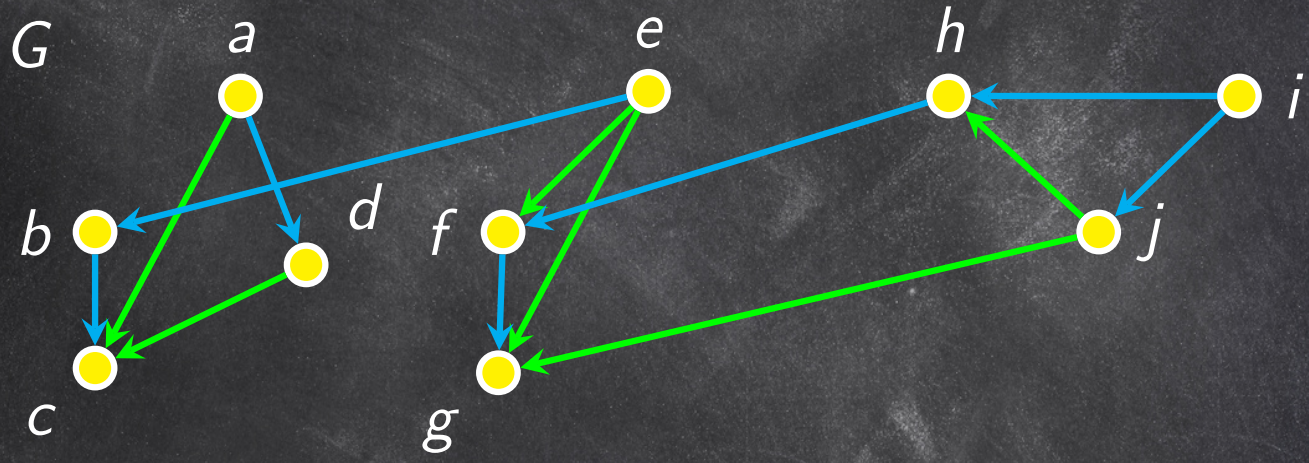


v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	0	15	14	19	16	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

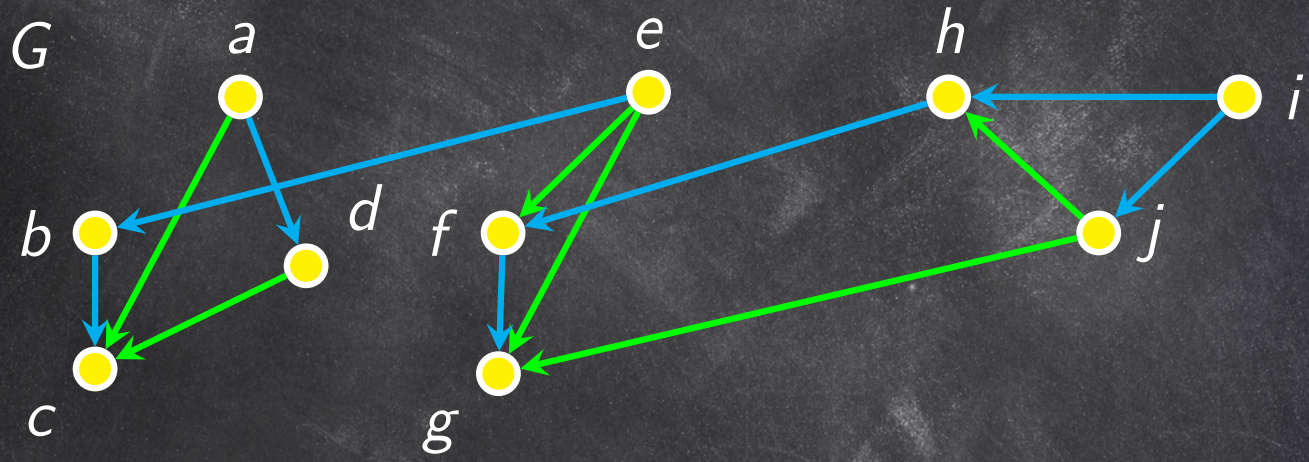


v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	20	15	14	19	16	6	5	7	10	9

Luís Felipe

09/04/26

Exemplo

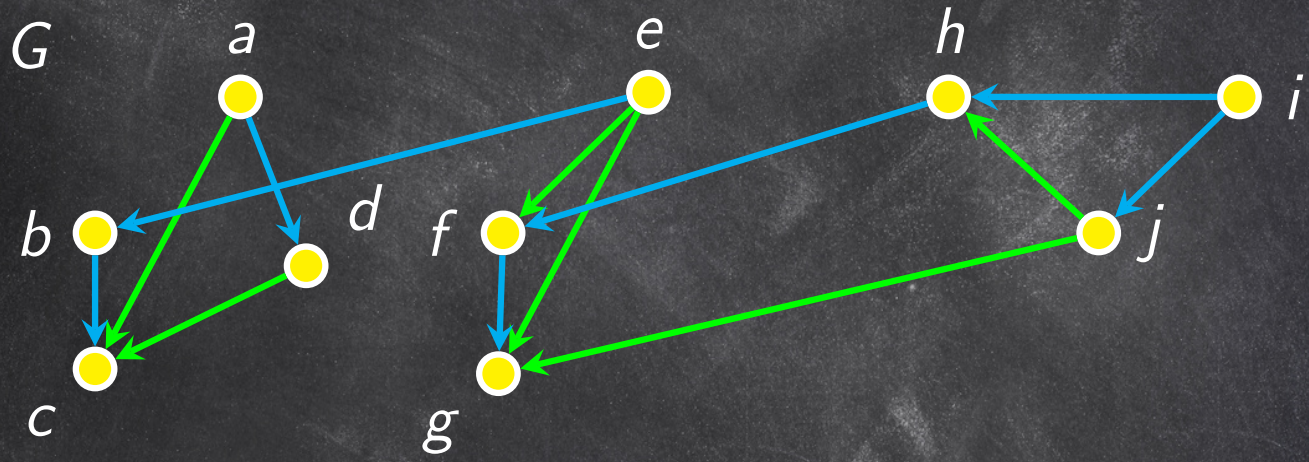


v	a	d	e	b	c	i	j	h	f	g
PS(v)	20	19	16	15	14	10	9	7	6	5

Luís Felipe

09/04/26

Exemplo



v	a	d	e	b	c	i	j	h	f	g
PS(v)	20	19	16	15	14	10	9	7	6	5

Ordenação topológica: *a d e b c i j h f g*

Luís Felipe

09/04/26

Obtenção da ordenação topológica

Corretude do algoritmo:

Seja v_n o vértice com menor profundidade de saída PS.

Luís Felipe

09/04/26

Obtenção da ordenação topológica

Corretude do algoritmo:

Seja v_n o vértice com menor profundidade de saída PS.
Suponha, por contradição, que exista uma aresta $v_n \to w$.

Luís Felipe

09/04/26

Obtenção da ordenação topológica

Corretude do algoritmo:

Seja v_n o vértice com menor profundidade de saída PS.
Suponha, por contradição, que exista uma aresta $v_n \rightarrow w$.
Pela propriedade da DFS em dígrafos acíclicos, temos que para toda aresta vw vale: PS(v) > PS(w).

Luís Felipe

09/04/26

Obtenção da ordenação topológica

Corretude do algoritmo:

Seja v_n o vértice com menor profundidade de saída PS.
Suponha, por contradição, que exista uma aresta $v_n \to w$.
Pela propriedade da DFS em dígrafos acíclicos, temos que para toda aresta vw vale: PS(v) > PS(w). Logo, $PS(v_n) > PS(w)$, o que contradiz o fato de v_n ter o menor PS.

Luís Felipe

09/04/26

Obtenção da ordenação topológica

Corretude do algoritmo:

Seja v_n o vértice com menor profundidade de saída PS.
Suponha, por contradição, que exista uma aresta $v_n \rightarrow w$.
Pela propriedade da DFS em dígrafos acíclicos, temos que para toda aresta vw vale: PS(v) > PS(w). Logo, $PS(v_n) > PS(w)$, o que contradiz o fato de v_n ter o menor PS.
Portanto, não existe aresta saindo de v_n; logo v_n é um sumidouro do grafo (grau de saída igual a 0). Assim, sua posição final na ordenação topológica está correta.

Luís Felipe

09/04/26

Obtenção da ordenação topológica

Corretude do algoritmo:

Seja v_n o vértice com menor profundidade de saída PS.
Suponha, por contradição, que exista uma aresta $v_n \to w$.
Pela propriedade da DFS em dígrafos acíclicos, temos que para toda aresta vw vale: PS(v) > PS(w). Logo, $PS(v_n) > PS(w)$, o que contradiz o fato de v_n ter o menor PS.
Portanto, não existe aresta saindo de v_n; logo v_n é um sumidouro do grafo (grau de saída igual a 0). Assim, sua posição final na ordenação topológica está correta.

Removendo v_n do grafo, o vértice v_{n-1} torna-se um sumidouro de $G - v_n$.

Luís Felipe

09/04/26

Obtenção da ordenação topológica

Corretude do algoritmo:

Seja v_n o vértice com menor profundidade de saída PS.
Suponha, por contradição, que exista uma aresta $v_n \to w$.
Pela propriedade da DFS em dígrafos acíclicos, temos que para toda aresta vw vale: PS(v) > PS(w). Logo, $PS(v_n) > PS(w)$, o que contradiz o fato de v_n ter o menor PS.
Portanto, não existe aresta saindo de v_n; logo v_n é um sumidouro do grafo (grau de saída igual a 0). Assim, sua posição final na ordenação topológica está correta.

Removendo v_n do grafo, o vértice v_{n-1} torna-se um sumidouro de $G - v_n$.
Aplicando sucessivamente esse raciocínio, cada vértice v_i é um sumidouro do dígrafo $G - \{v_{i+1}, v_{i+2}, \ldots, v_n\}$.

Luís Felipe

09/04/26

Ordenação topológica via DFS ao passo da execução

Ideia fundamental:

Durante a execução da DFS, os vértices são mantidos implicitamente em uma pilha de execução.

Luís Felipe

09/04/26

Ordenação topológica via DFS ao passo da execução

Ideia fundamental:

Durante a execução da DFS, os vértices são mantidos implicitamente em uma pilha de execução.

Quando um vértice v termina sua exploração (ou seja, quando todos os seus vizinhos já foram visitados), ele é removido da pilha.

Luís Felipe

09/04/26

Ordenação topológica via DFS ao passo da execução

Ideia fundamental:

Durante a execução da DFS, os vértices são mantidos implicitamente em uma pilha de execução.

Quando um vértice v termina sua exploração (ou seja, quando todos os seus vizinhos já foram visitados), ele é removido da pilha.

Observação: Esse é exatamente o momento em que v recebe seu tempo de saída $PS(v)$.

Luís Felipe

09/04/26

Ordenação topológica via DFS ao passo da execução

Ideia fundamental:

Durante a execução da DFS, os vértices são mantidos implicitamente em uma pilha de execução.

Quando um vértice v termina sua exploração (ou seja, quando todos os seus vizinhos já foram visitados), ele é removido da pilha.

Observação: Esse é exatamente o momento em que v recebe seu tempo de saída $PS(v)$.

Conclusão:
Podemos construir uma ordenação topológica durante a própria DFS, da seguinte forma:

- sempre que um vértice v sai da pilha, inseri-lo no início da ordenação;

Luís Felipe

09/04/26

Ordenação topológica via DFS ao passo da execução

Ideia fundamental:

Durante a execução da DFS, os vértices são mantidos implicitamente em uma pilha de execução.

Quando um vértice v termina sua exploração (ou seja, quando todos os seus vizinhos já foram visitados), ele é removido da pilha.

Observação: Esse é exatamente o momento em que v recebe seu tempo de saída $PS(v)$.

Conclusão:
Podemos construir uma ordenação topológica durante a própria DFS, da seguinte forma:

- sempre que um vértice v sai da pilha, inseri-lo no início da ordenação;

Um vértice só sai da pilha depois de todos os vértices que dependem dele. Logo, a ordem de remoção da pilha (invertida) é uma ordenação topológica.

Luís Felipe

09/04/26

DFS para ordenação topológica com detecção de ciclo

```python
def dfs_topo(v, adj, pe, ps, pai, t, emPilha, ordem, ciclo):
    t[0] += 1
    pe[v] = t[0]
    emPilha[v] = True

    for w in adj[v]:
        if pe[w] == 0:                  # w ainda nao foi visitado
            pai[w] = v
            dfs_topo(w, adj, pe, ps, pai, t, emPilha, ordem, ciclo)
        elif emPilha[w]:                # aresta de retorno
            ciclo[0] = True

    t[0] += 1
    ps[v] = t[0]
    emPilha[v] = False

    ordem.insert(0, v)                  # insere no inicio da ordenacao

def ordenacao_topologica(adj):
    n = len(adj)
    pe = [0]*n
    ps = [0]*n
    pai = [-1]*n
    emPilha = [False]*n
    ordem = []
    t = [0]
    ciclo = [False]

    for v in range(n):
        if pe[v] == 0:
            dfs_topo(v, adj, pe, ps, pai, t, emPilha, ordem, ciclo)

    if ciclo[0]:
        return None                     # G possui ciclo

    return ordem
```

Luís Felipe

09/04/26

Complexidade

Observações:

- Inserir v ao final da DFS equivale a usar $PS(v)$ em ordem decrescente.

Luís Felipe

09/04/26

Complexidade

Observações:

- Inserir v ao final da DFS equivale a usar $PS(v)$ em ordem decrescente.
- A presença de aresta de retorno implica que o grafo não é uma DAG.

Luís Felipe

09/04/26

Complexidade

Observações:

- Inserir v ao final da DFS equivale a usar $PS(v)$ em ordem decrescente.
- A presença de aresta de retorno implica que o grafo não é uma DAG.
- A ordenação é construída durante a execução da DFS.

Complexidade:

Rodar uma busca em profundidade sobre G tem complexidade $O(n + m)$.

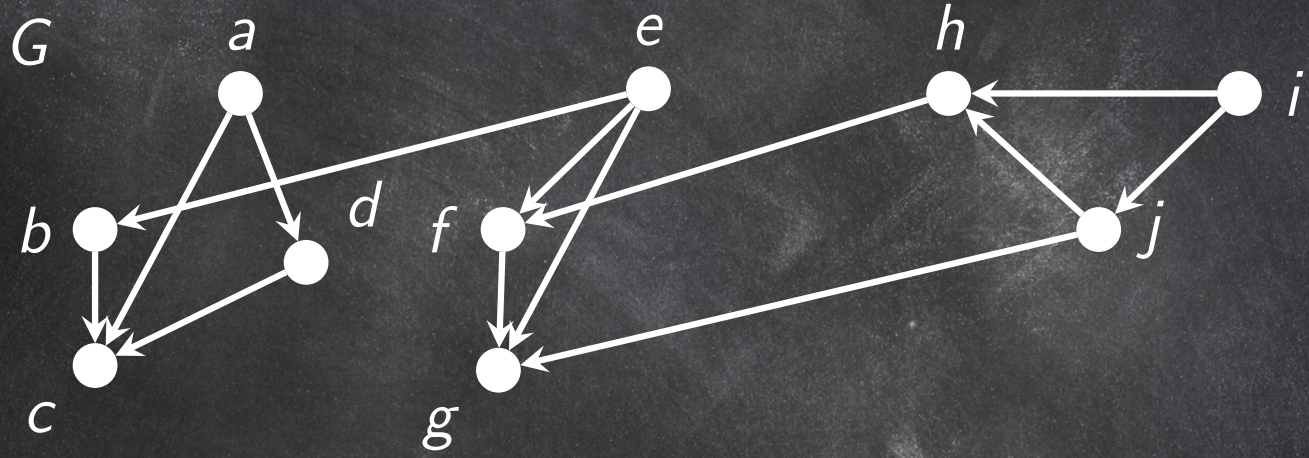
Cada vez que um vértice sai da busca (recebe uma PS diferente de zero), é colocado imediatamente à esquerda da ordenação topológica sendo construída. Isto dispensa o passo de ordenação das PS's.

Logo, a complexidade final é $O(n + m)$.

Luís Felipe

09/04/26

Exemplo



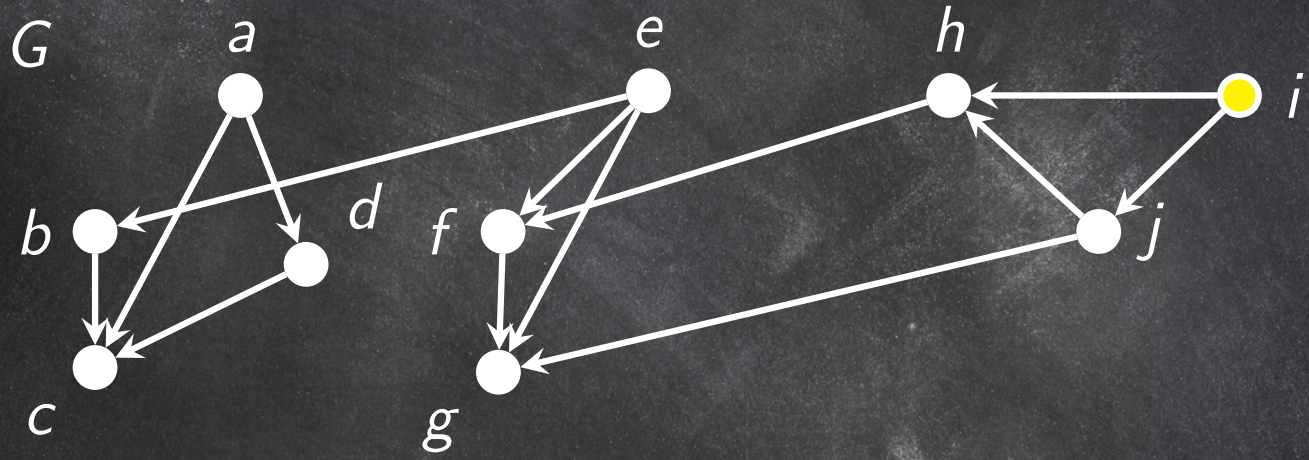
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	0	0	0	0	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Ordenação topológica:

Luís Felipe

09/04/26

Exemplo



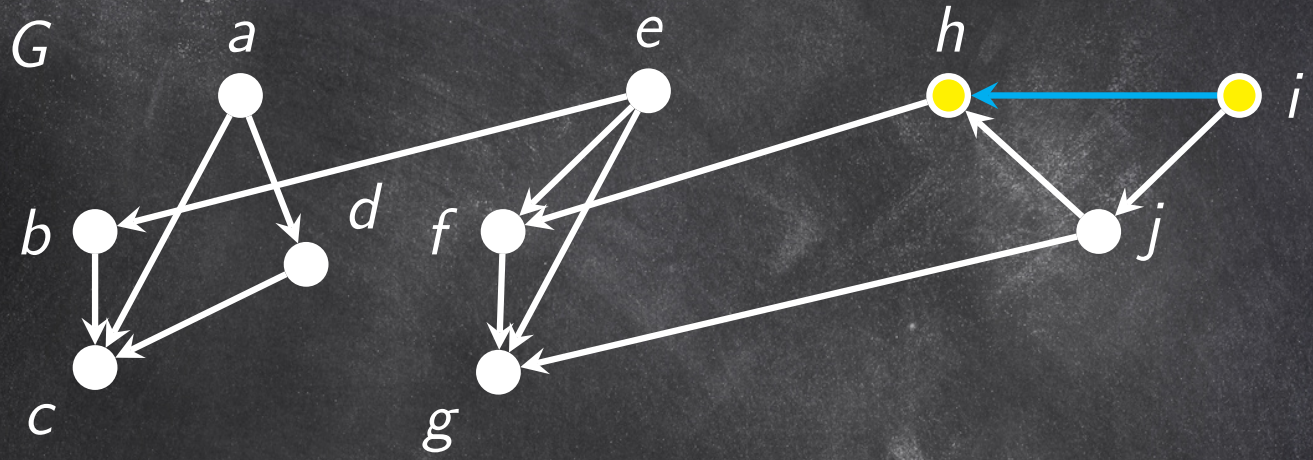
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	0	0	0	1	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Ordenação topológica:

Luís Felipe

09/04/26

Exemplo



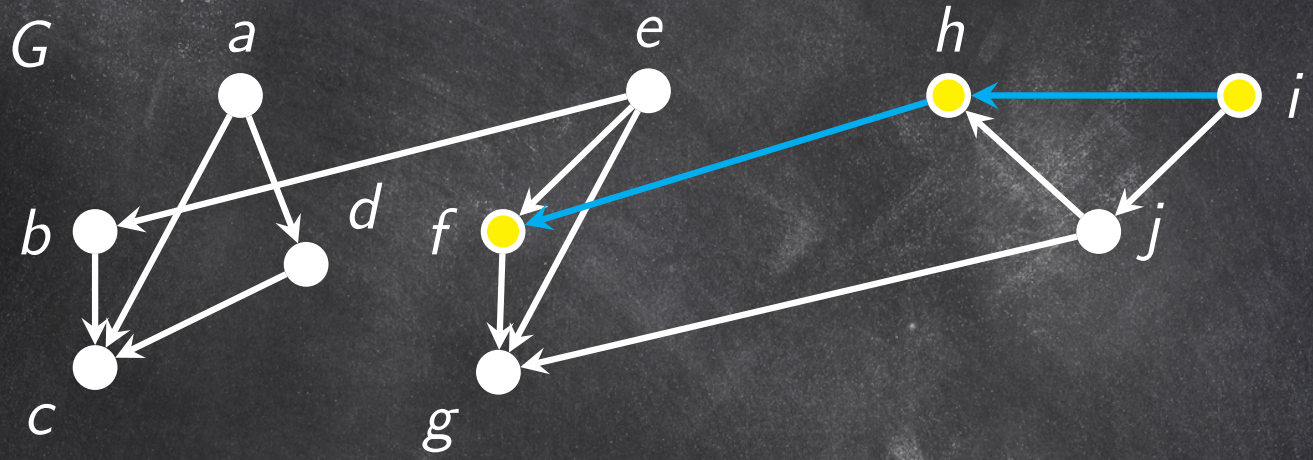
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	0	0	2	1	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Ordenação topológica:

Luís Felipe

09/04/26

Exemplo



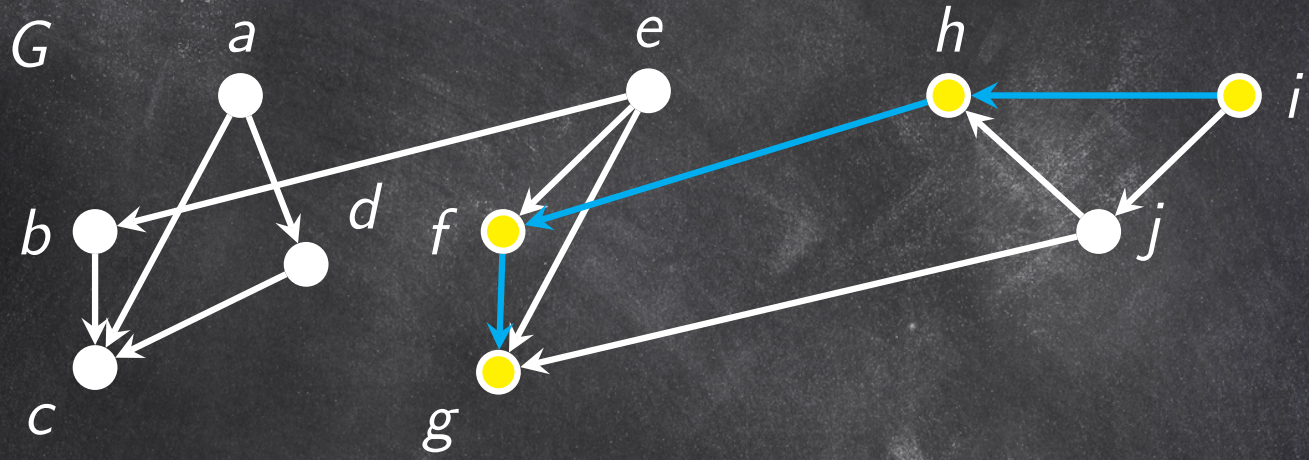
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	0	2	1	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Ordenação topológica:

Luís Felipe

09/04/26

Exemplo



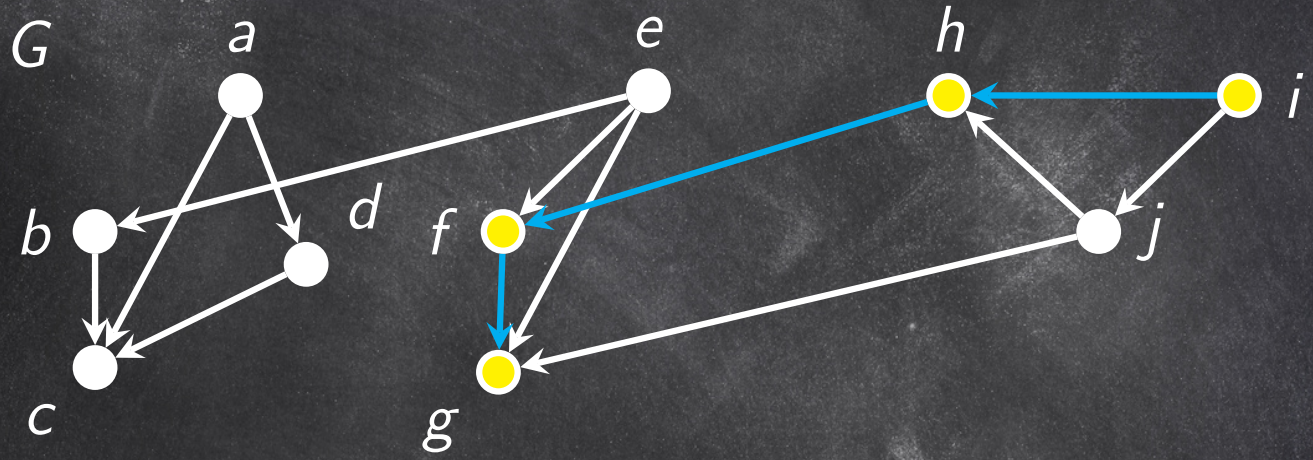
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Ordenação topológica:

Luís Felipe

09/04/26

Exemplo



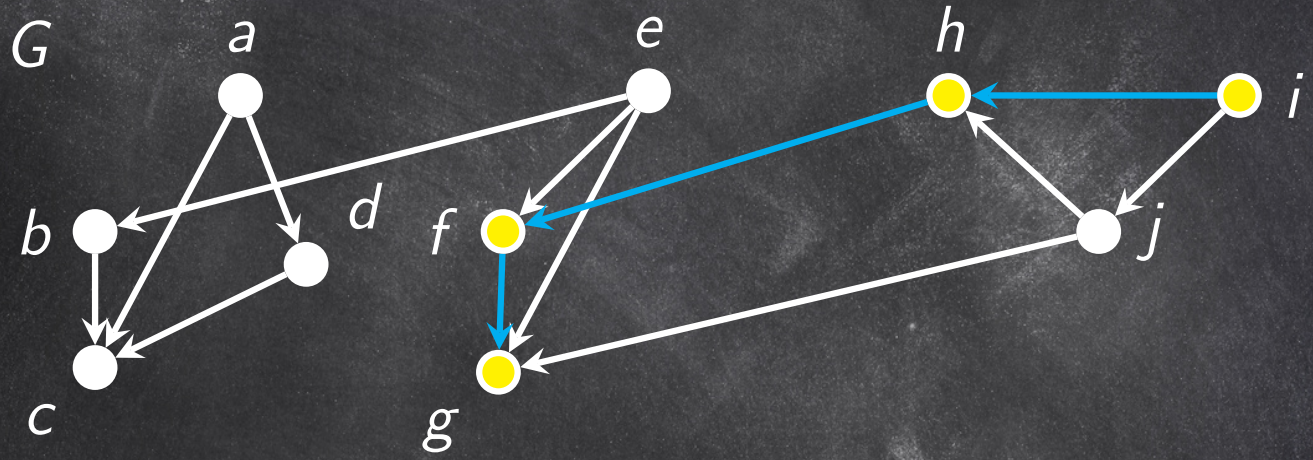
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	0
PS(v)	0	0	0	0	0	0	5	0	0	0

Ordenação topológica: g

Luís Felipe

09/04/26

Exemplo



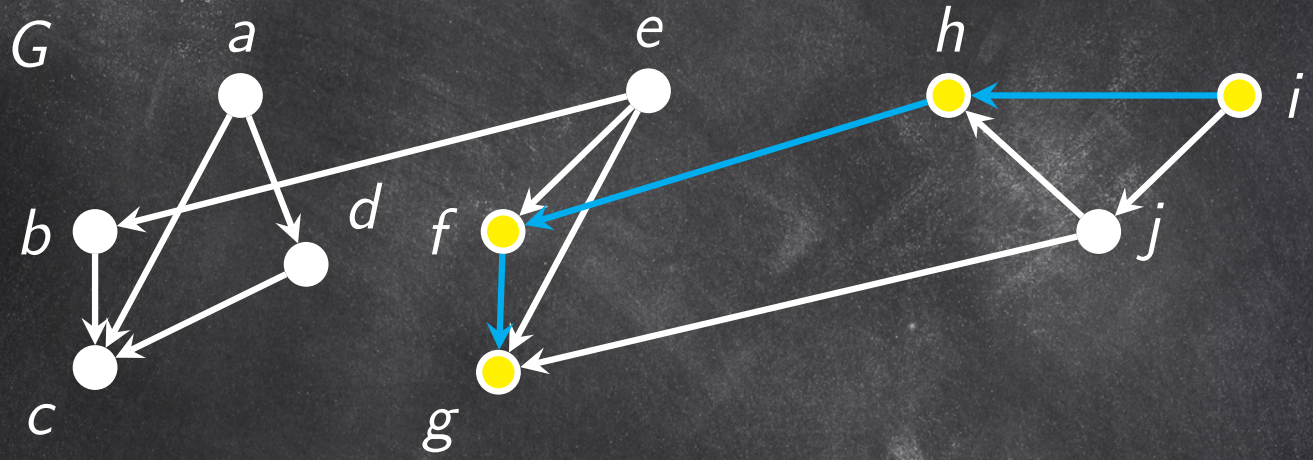
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	0
PS(v)	0	0	0	0	0	6	5	0	0	0

Ordenação topológica: f g

Luís Felipe

09/04/26

Exemplo



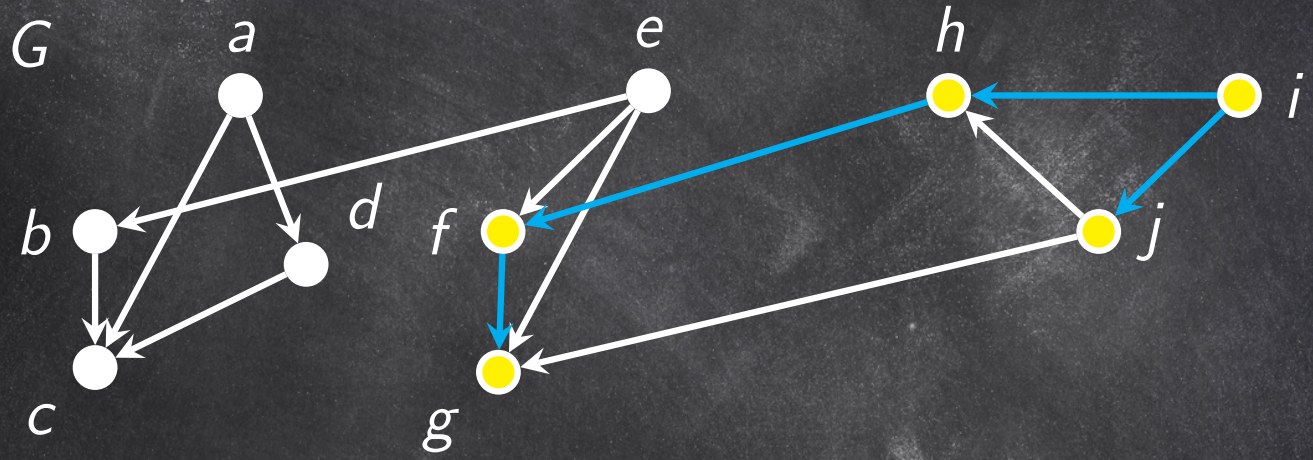
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	0
PS(v)	0	0	0	0	0	6	5	7	0	0

Ordenação topológica: h f g

Luís Felipe

09/04/26

Exemplo



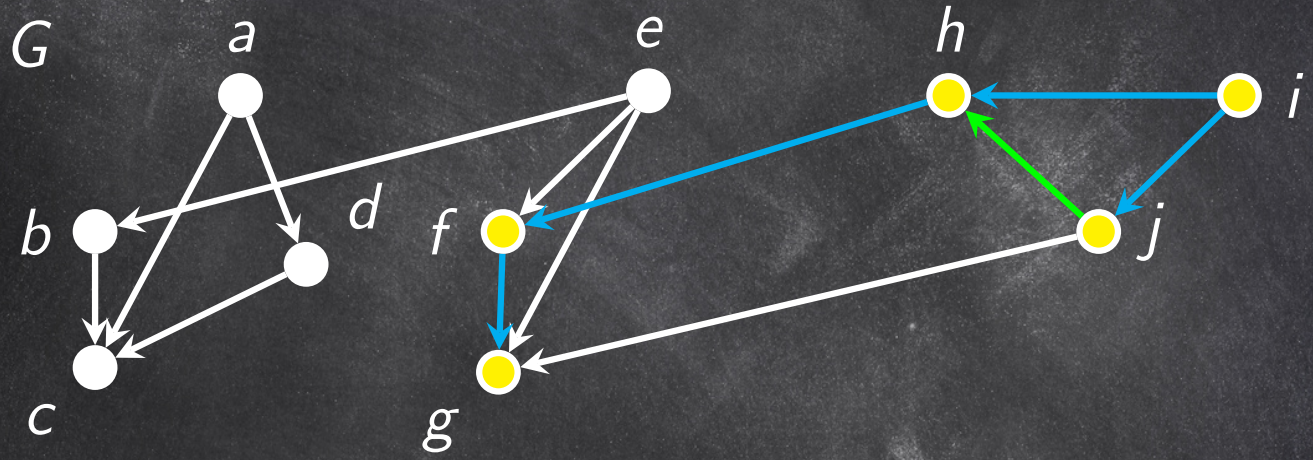
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	0	0

Ordenação topológica: h f g

Luís Felipe

09/04/26

Exemplo



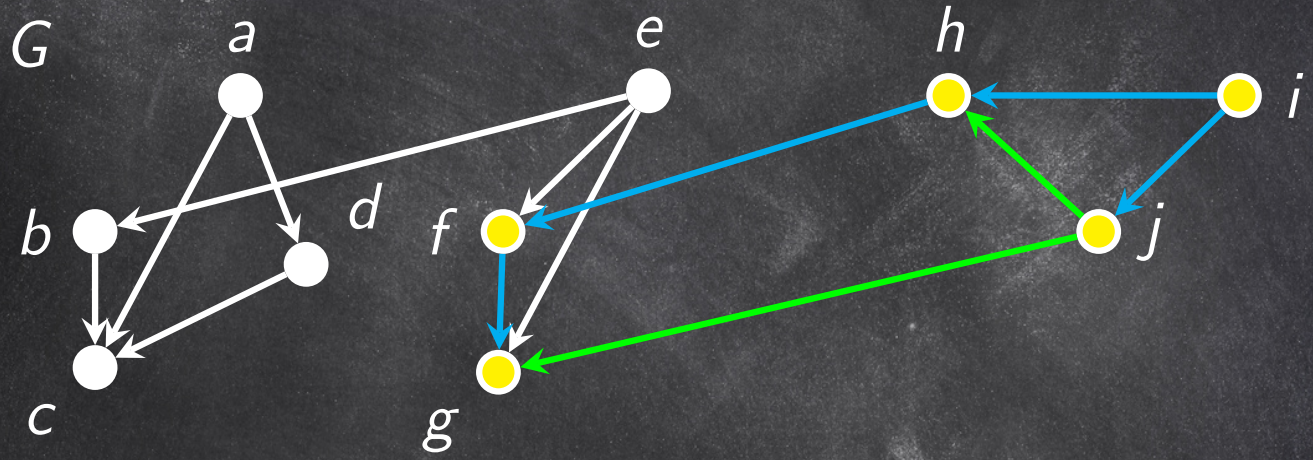
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	0	0

Ordenação topológica: h f g

Luís Felipe

09/04/26

Exemplo



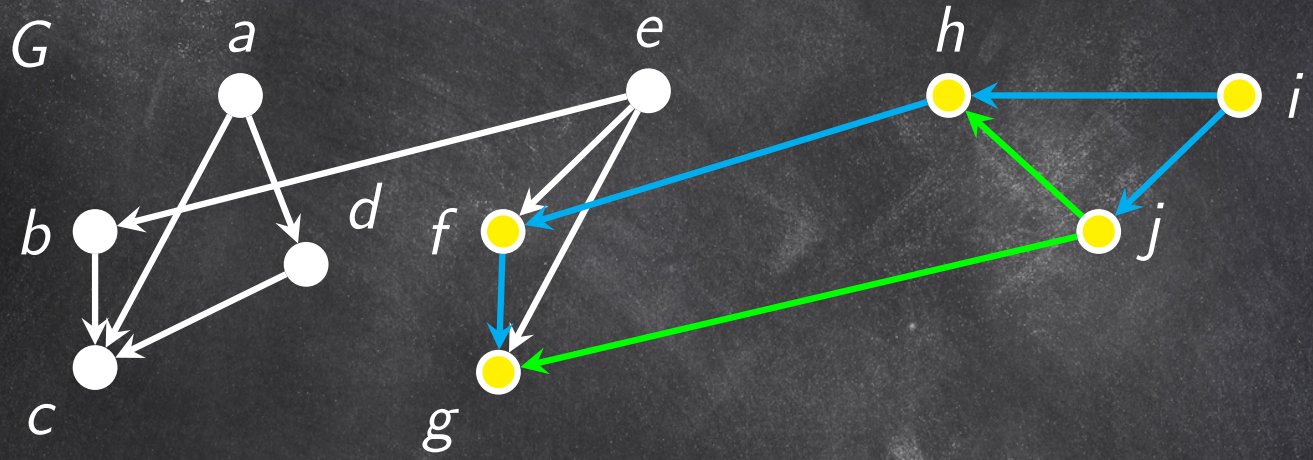
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	0	0

Ordenação topológica: h f g

Luís Felipe

09/04/26

Exemplo



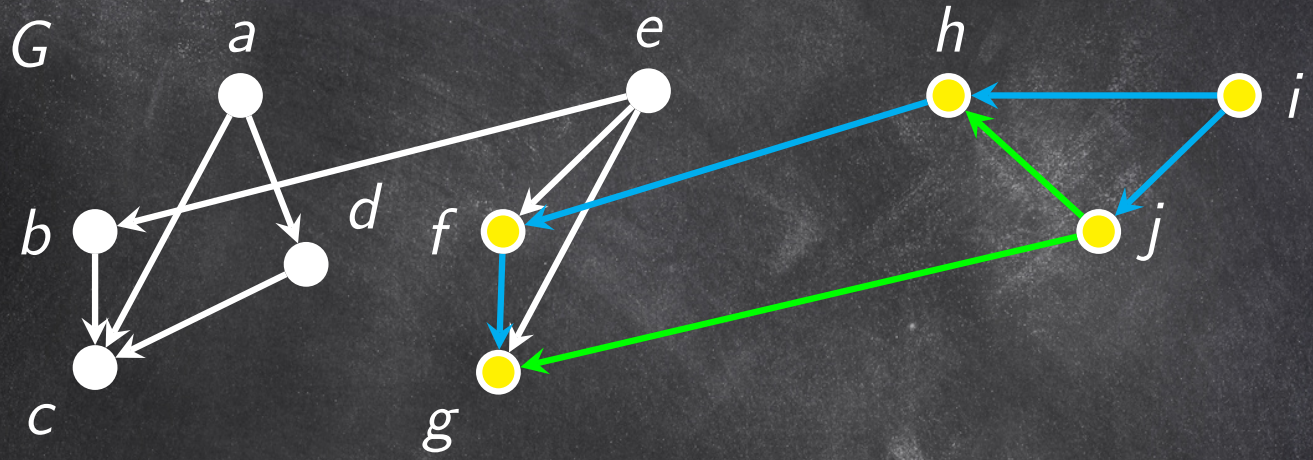
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	0	9

Ordenação topológica: j h f g

Luís Felipe

09/04/26

Exemplo



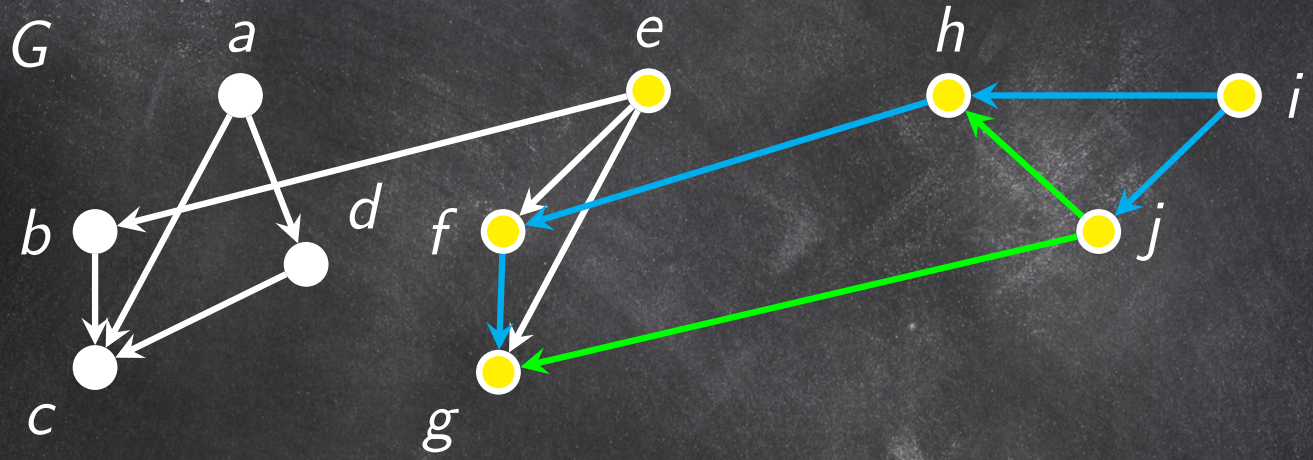
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Ordenação topológica: *i j h f g*

Luís Felipe

09/04/26

Exemplo



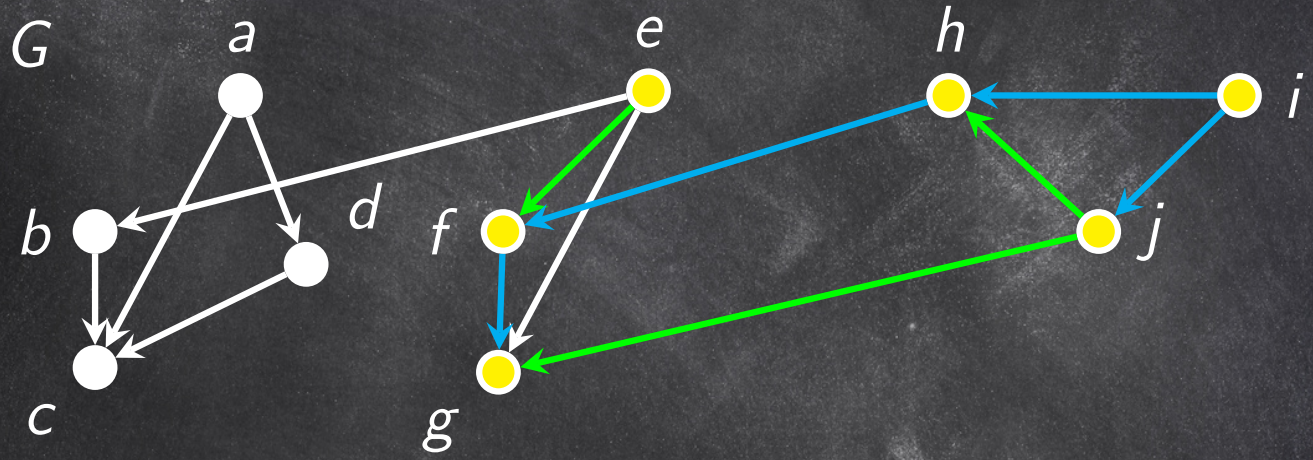
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Ordenação topológica: *i j h f g*

Luís Felipe

09/04/26

Exemplo



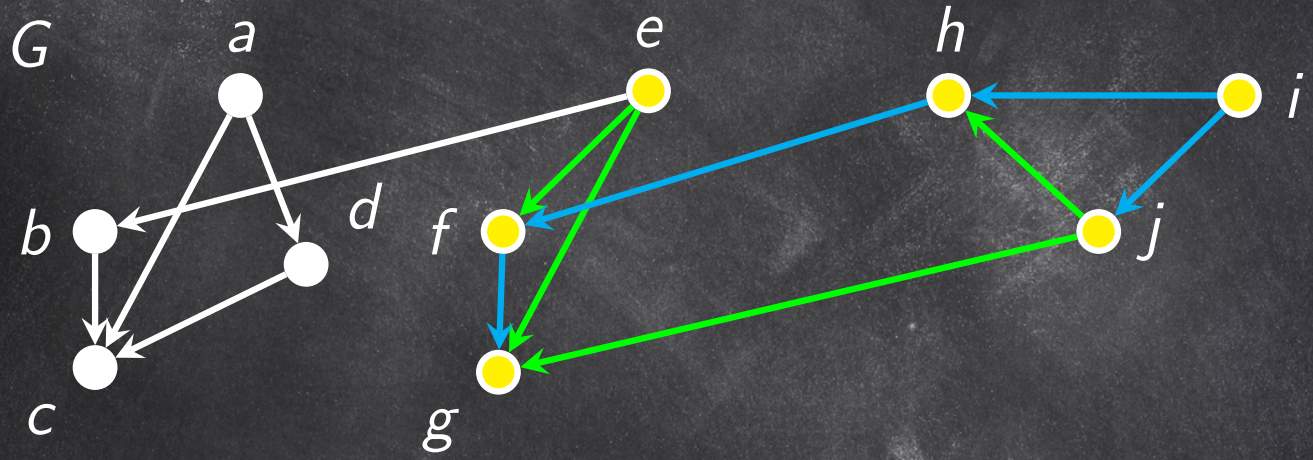
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Ordenação topológica: *i j h f g*

Luís Felipe

09/04/26

Exemplo



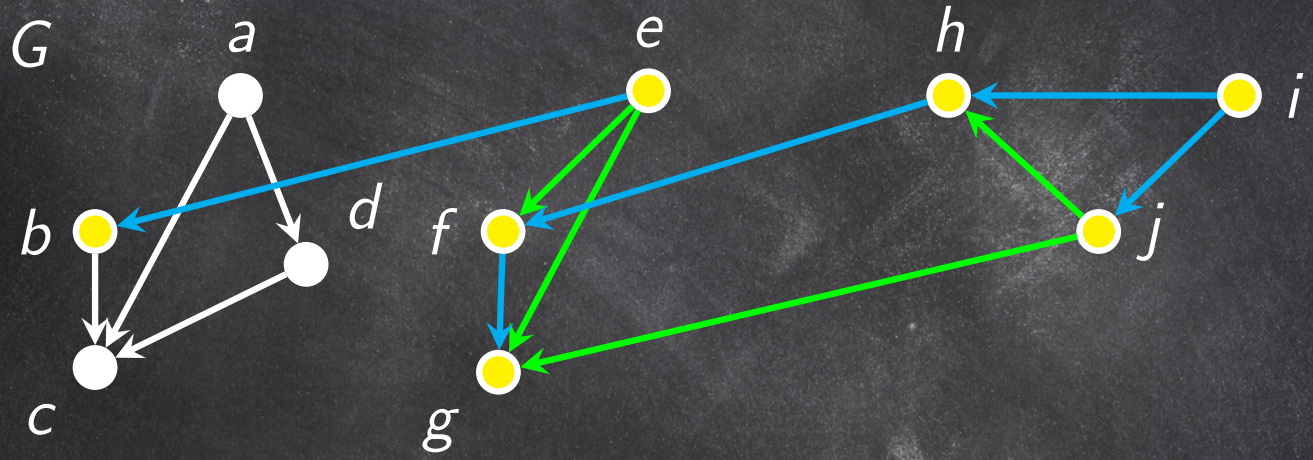
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Ordenação topológica: *i j h f g*

Luís Felipe

09/04/26

Exemplo



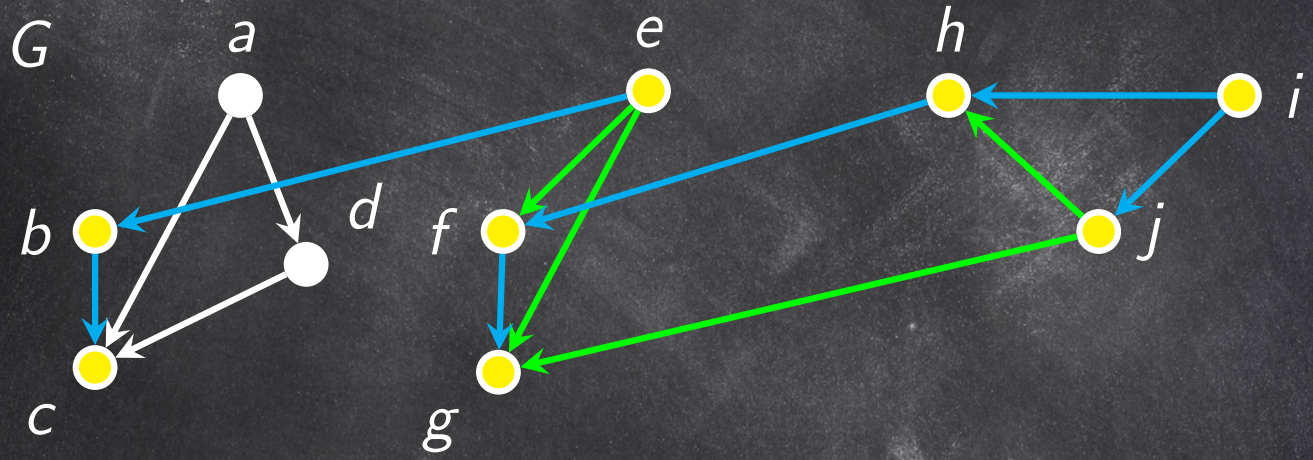
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	12	0	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Ordenação topológica: *i j h f g*

Luís Felipe

09/04/26

Exemplo



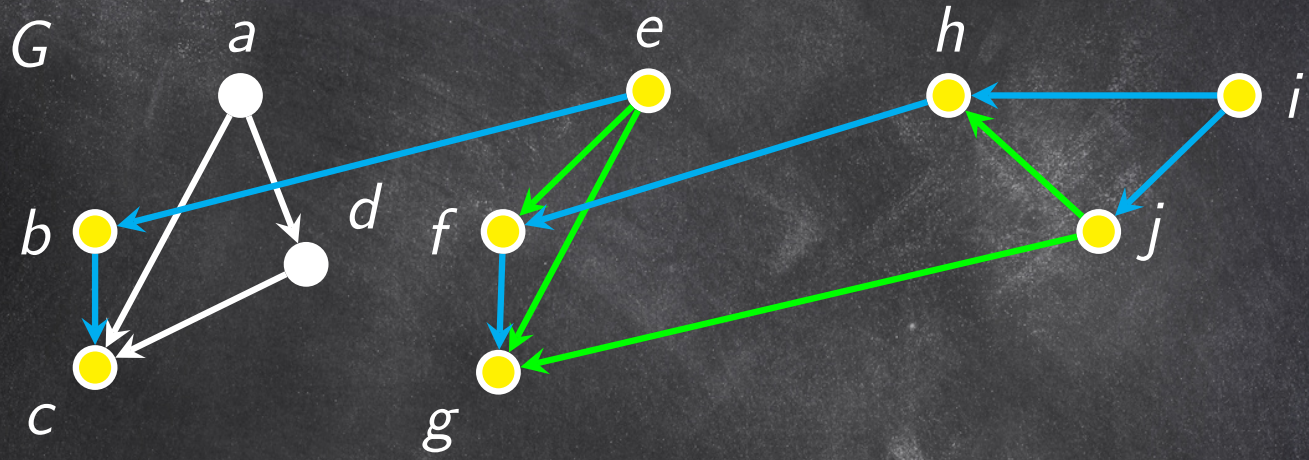
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	12	13	0	11	3	4	2	1	8
PS(v)	0	0	0	0	0	6	5	7	10	9

Ordenação topológica: *i j h f g*

Luís Felipe

09/04/26

Exemplo



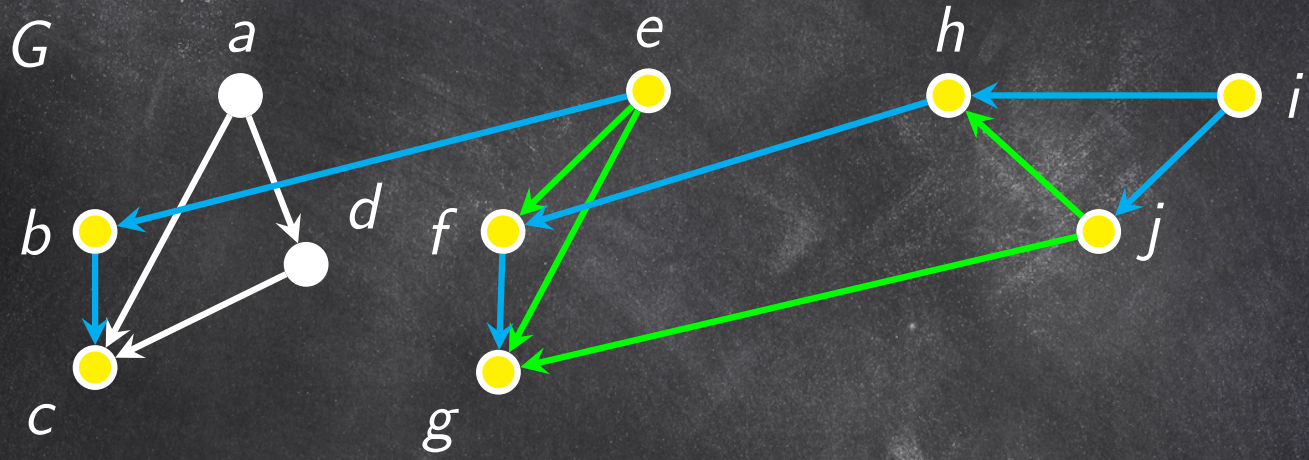
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	12	13	0	11	3	4	2	1	8
PS(v)	0	0	14	0	0	6	5	7	10	9

Ordenação topológica: *c i j h f g*

Luís Felipe

09/04/26

Exemplo



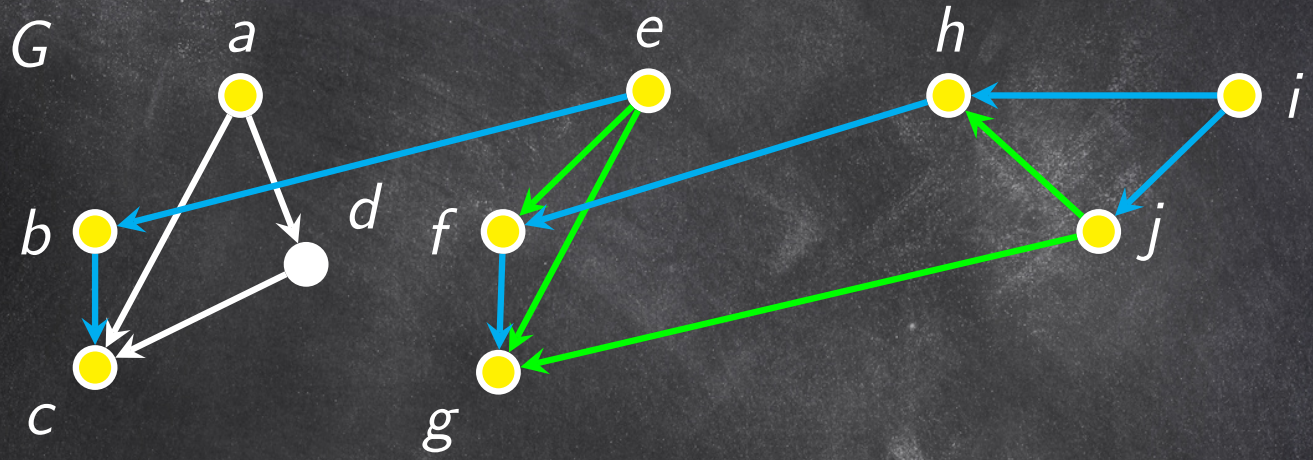
v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	12	13	0	11	3	4	2	1	8
PS(v)	0	15	14	0	0	6	5	7	10	9

Ordenação topológica: *b c i j h f g*

Luís Felipe

09/04/26

Exemplo



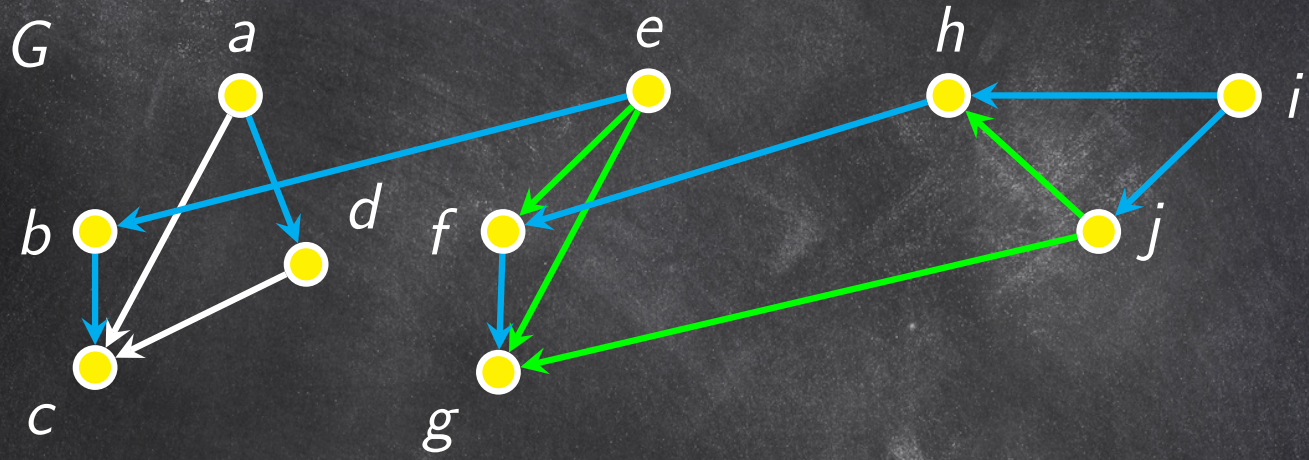
v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	0	11	3	4	2	1	8
PS(v)	0	15	14	0	16	6	5	7	10	9

Ordenação topológica: *e b c i j h f g*

Luís Felipe

09/04/26

Exemplo



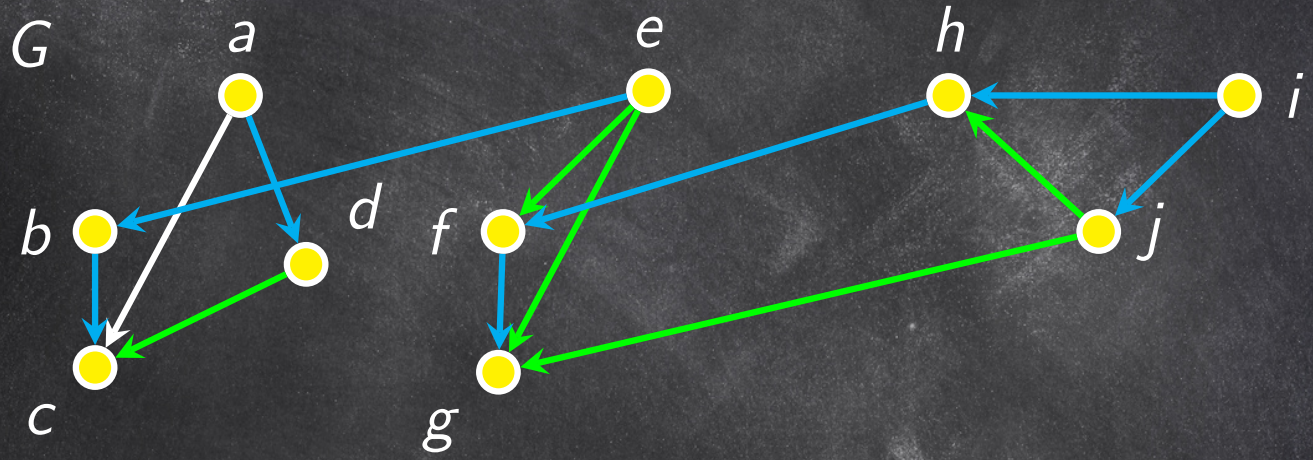
v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	0	15	14	0	16	6	5	7	10	9

Ordenação topológica: *e b c i j h f g*

Luís Felipe

09/04/26

Exemplo



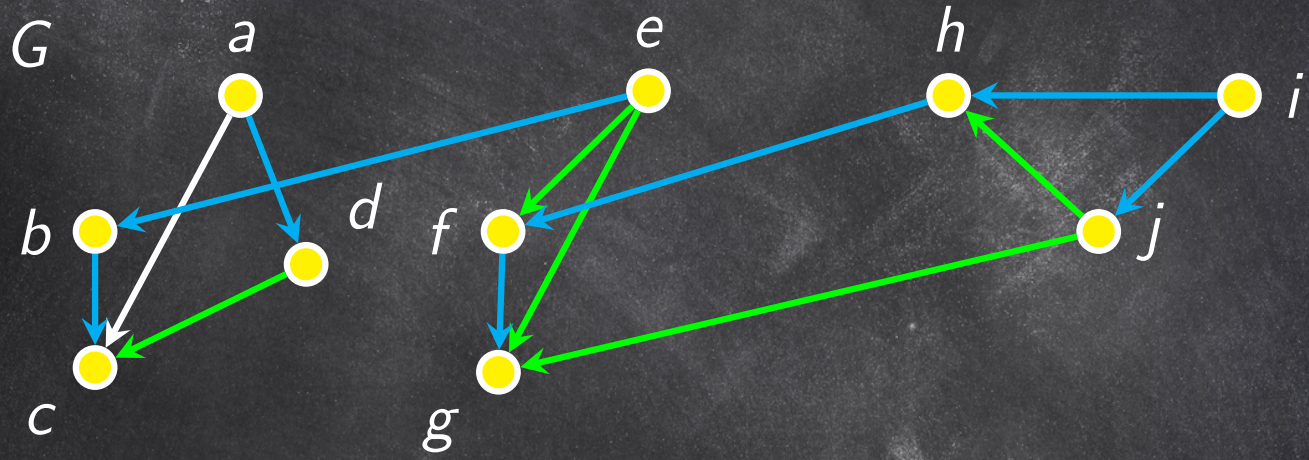
v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	0	15	14	0	16	6	5	7	10	9

Ordenação topológica: *e b c i j h f g*

Luís Felipe

09/04/26

Exemplo



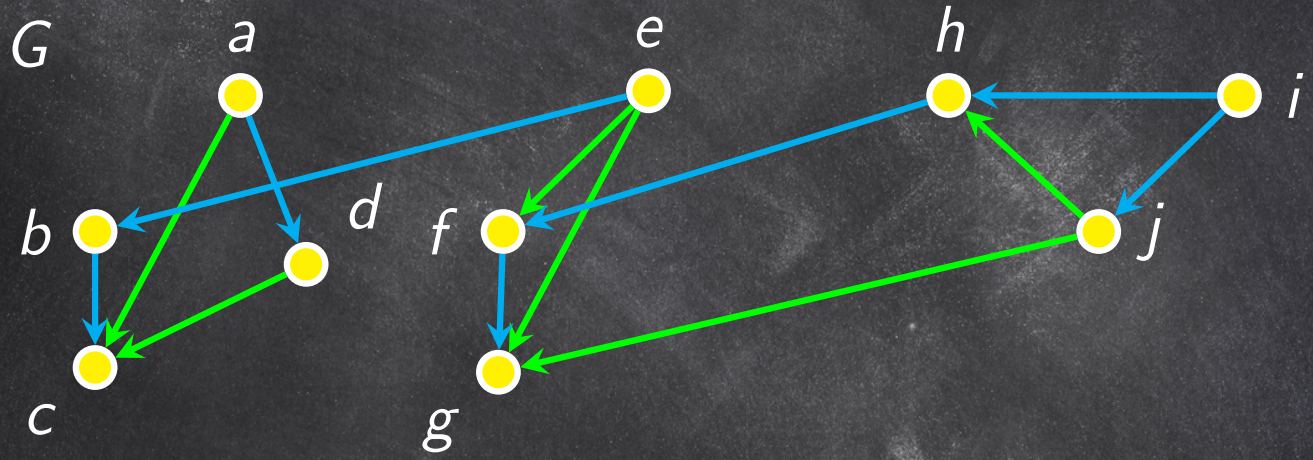
v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	0	15	14	19	16	6	5	7	10	9

Ordenação topológica: *d e b c i j h f g*

Luís Felipe

09/04/26

Exemplo



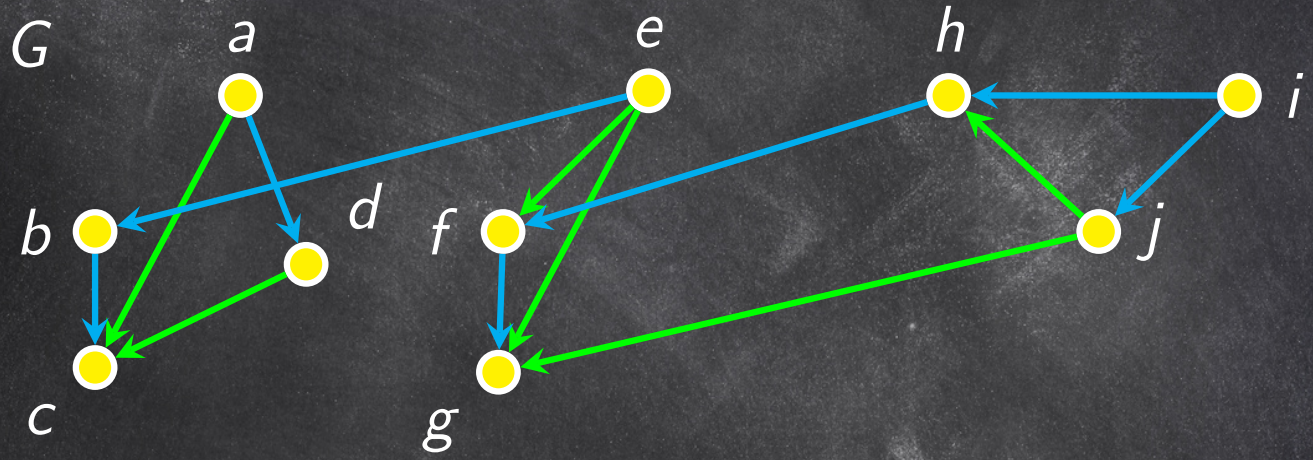
v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	0	15	14	19	16	6	5	7	10	9

Ordenação topológica: *d e b c i j h f g*

Luís Felipe

09/04/26

Exemplo



v	a	b	c	d	e	f	g	h	i	j
PE(v)	17	12	13	18	11	3	4	2	1	8
PS(v)	20	15	14	19	16	6	5	7	10	9

Ordenação topológica: *a d e b c i j h f g*

Luís Felipe

09/04/26

Algoritmo alternativo para ordenação topológica

1. Enquanto ainda existirem vértices no grafo:
   - localizar um sumidouro
   - removê-lo do grafo
   - colocá-lo à esquerda na ordenação sendo construída
2. Repetir o processo até remover todos os vértices.

```
1  def ordenacao_topologica_sumidouros(adj):
2      n = len(adj)        # adj[v] = lista de vizinhos alcançados por arestas que saem de v
3      removido = [False] * n
4      ordenacao = []
5      for _ in range(n):
6          sumidouro = -1
7          for v in range(n):              # procura um vértice não removido com grau de saída 0
8              if removido[v]:
9                  continue                # ignora vértices já removidos
10
11             grau_saida = 0
12             for w in adj[v]:
13                 if not removido[w]:
14                     grau_saida += 1     # ver se há vizinho de saída no grafo atual
15
16             if grau_saida == 0:
17                 sumidouro = v
18                 break
19
20         if sumidouro == -1:
21             return None                 # o grafo não é acíclico
22
23         removido[sumidouro] = True
24         ordenacao.insert(0, sumidouro)  # coloca a esquerda
25
26     return ordenacao
```

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.
- Em cada iteração, procuramos um sumidouro:
  - Percorremos até n vértices não removidos

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.
- Em cada iteração, procuramos um sumidouro:
  - Percorremos até n vértices não removidos
  - Para cada vértice v, verificamos seus vizinhos em adj[v]

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.
- Em cada iteração, procuramos um sumidouro:
  - Percorremos até n vértices não removidos
  - Para cada vértice v, verificamos seus vizinhos em adj[v]
- O custo para testar um vértice é proporcional ao seu grau de saída

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.
- Em cada iteração, procuramos um sumidouro:
  - Percorremos até n vértices não removidos
  - Para cada vértice v, verificamos seus vizinhos em adj[v]
- O custo para testar um vértice é proporcional ao seu grau de saída
- No pior caso, examinamos todas as listas de adjacência

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.
- Em cada iteração, procuramos um sumidouro:
  - Percorremos até n vértices não removidos
  - Para cada vértice v, verificamos seus vizinhos em adj[v]
- O custo para testar um vértice é proporcional ao seu grau de saída
- No pior caso, examinamos todas as listas de adjacência

Conclusão:

$$T(n, m) = O(n \cdot (n + m)) = O(n^2 + nm)$$

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.
- Em cada iteração, procuramos um sumidouro:
  - Percorremos até n vértices não removidos
  - Para cada vértice v, verificamos seus vizinhos em adj[v]
- O custo para testar um vértice é proporcional ao seu grau de saída
- No pior caso, examinamos todas as listas de adjacência

Conclusão:

$$T(n, m) = O(n \cdot (n + m)) = O(n^2 + nm)$$

- O algoritmo recalcula graus de saída a cada iteração

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.
- Em cada iteração, procuramos um sumidouro:
  - Percorremos até n vértices não removidos
  - Para cada vértice v, verificamos seus vizinhos em adj[v]
- O custo para testar um vértice é proporcional ao seu grau de saída
- No pior caso, examinamos todas as listas de adjacência

Conclusão:

$$T(n, m) = O(n \cdot (n + m)) = O(n^2 + nm)$$

- O algoritmo recalcula graus de saída a cada iteração
- Não há reaproveitamento de informação

Luís Felipe

09/04/26

Complexidade do algoritmo por sumidouros

Análise de tempo:

- O algoritmo executa um laço principal de n iterações.
- Em cada iteração, procuramos um sumidouro:
  - Percorremos até n vértices não removidos
  - Para cada vértice v, verificamos seus vizinhos em adj[v]
- O custo para testar um vértice é proporcional ao seu grau de saída
- No pior caso, examinamos todas as listas de adjacência

Conclusão:

$$T(n, m) = O(n \cdot (n + m)) = O(n^2 + nm)$$

- O algoritmo recalcula graus de saída a cada iteração
- Não há reaproveitamento de informação
- Mais lento que DFS.