Aula 7 - DFS em grafos direcionados

Luís Felipe

UFF

02 de Abril de 2026

Luís Felipe

02/04/26

Busca em profundidade p/ dígrafos

Objetivo: percorrer um dígrafo G usando busca em profundidade (DFS), registrando:

- o instante em que cada vértice é descoberto;
- o instante em que cada vértice tem sua exploração encerrada;
- a relação de pai entre os vértices da floresta de profundidade;
- a classificação das arestas do dígrafo.

Luís Felipe

02/04/26

Busca em profundidade p/ dígrafos

Objetivo: percorrer um dígrafo G usando busca em profundidade (DFS), registrando:

- o instante em que cada vértice é descoberto;
- o instante em que cada vértice tem sua exploração encerrada;
- a relação de pai entre os vértices da floresta de profundidade;
- a classificação das arestas do dígrafo.

Ideia central:

- ao entrar em um vértice v, registramos seu tempo de entrada $PE(v)$;
- exploramos todos os seus sucessores (vértices alcançáveis a partir de v);
- ao terminar, registramos seu tempo de saída $PS(v)$.

Luís Felipe

02/04/26

Busca em profundidade p/ dígrafos

Inicialização:

$t \leftarrow 0$ // relógio ou tempo glogal

Para todo vértice $v \in V(G)$, fazemos:

$PE(v) \leftarrow 0$ // relógio de entrada de v
$PS(v) \leftarrow 0$ // tempo de saída de v
$pai(v) \leftarrow null$ // define a floresta de profundidade

Luís Felipe

02/04/26

Busca em profundidade p/ dígrafos

Inicialização:

$t \leftarrow 0$ // relógio ou tempo glogal

Para todo vértice $v \in V(G)$, fazemos:

$PE(v) \leftarrow 0$ // relógio de entrada de v
$PS(v) \leftarrow 0$ // tempo de saída de v
$pai(v) \leftarrow null$ // define a floresta de profundidade

Ou seja:

- $PE(v) = 0$ significa que v ainda não foi visitado;
- $PS(v) = 0$ significa que a exploração de v ainda não terminou;
- $pai(v)$ indicará de qual vértice v foi alcançado pela primeira vez.

Busca em profundidade p/ dígrafos

Chamada externa

Enquanto existir $v \in V(G)$ tal que $PE(v) = 0$, faça:

executar $P(v)$ // nova raiz da busca

Obs.: Por que essa etapa é necessária?

Luís Felipe

02/04/26

Busca em profundidade p/ dígrafos

Chamada externa

Enquanto existir $v \in V(G)$ tal que $PE(v) = 0$, faça:

executar $P(v)$ // nova raiz da busca

Obs.: Por que essa etapa é necessária?

- Em um dígrafo, pode acontecer de a busca iniciada em um vértice não alcançar todos os demais.
- Por isso, a DFS completa pode gerar uma floresta de profundidade, e não apenas uma única árvore.
- Cada vez que encontramos um vértice ainda não visitado, iniciamos uma nova raiz.

Ou seja:
A chamada externa garante que todos os vértices do dígrafo serão processados.

Procedimento recursivo de busca p/ dígrafos

Procedimento $P(v)$

Ao entrar no vértice v:

$$\begin{aligned} t &\leftarrow t+1 \\ PE(v) &\leftarrow t \end{aligned}$$

Ou seja:

- incrementamos o relógio global;
- registramos o instante em que v foi descoberto;
- a partir desse momento, v passa a estar ativo na recursão.

Procedimento recursivo de busca p/ dígrafos

Procedimento $P(v)$

Ao entrar no vértice v:

$$\begin{aligned} t &\leftarrow t+1 \\ PE(v) &\leftarrow t \end{aligned}$$

Ou seja:

- incrementamos o relógio global;
- registramos o instante em que v foi descoberto;
- a partir desse momento, v passa a estar ativo na recursão.

Depois disso, percorremos todos os sucessores de v ($N_{out}(v)$ são os vizinhos de saída de v, i.e. vértices que v apontam):

para todo vértice $w \in N_{out}(v)$ faça

Analisamos cada arco vw que sai de v.

Luís Felipe

02/04/26

Classificação das arestas: aresta de árvore

Ao analisar um sucessor w de v, primeiro verificamos:

$$PE(w) = 0$$

Se isso acontecer, então w ainda não foi alcançado pela busca.

Nessa situação:

- marcamos vw como aresta de árvore;
- definimos $pai(w) \leftarrow v$
- executamos recursivamente $P(w)$

Obs.:

- a aresta vw é a responsável por descobrir w pela primeira vez;
- por isso, ela passa a fazer parte da floresta de profundidade.

Luís Felipe

02/04/26

Classificação das arestas: aresta de retorno

Se w já foi descoberto, então $PE(w) \neq 0$.

Luís Felipe

02/04/26

Classificação das arestas: aresta de retorno

Se w já foi descoberto, então $PE(w) \neq 0$.

O próximo teste é:

$$PS(w) = 0$$

Luís Felipe

02/04/26

Classificação das arestas: aresta de retorno

Se w já foi descoberto, então $PE(w) \neq 0$.

O próximo teste é:

$$PS(w) = 0$$

Se isso ocorrer, então w ainda não saiu da busca.
Ou seja, w ainda está ativo na pilha recursiva.

Nesse caso, marcamos vw como aresta de retorno.

Luís Felipe

02/04/26

Classificação das arestas: aresta de retorno

Se w já foi descoberto, então $PE(w) \neq 0$.

O próximo teste é:

$$PS(w) = 0$$

Se isso ocorrer, então w ainda não saiu da busca.
Ou seja, w ainda está ativo na pilha recursiva.

Nesse caso, marcamos vw como aresta de retorno.

Obs.:

- a aresta volta para um vértice que ainda está em processamento;
- em geral, isso significa que w é ancestral de v na árvore DFS;
- essas arestas são importantes para detectar ciclos e estruturas de conectividade.

Luís Felipe

02/04/26

Classificação das arestas: aresta de avanço

Se w já foi descoberto e já saiu da busca, então $PS(w) \neq 0$.
Nesse caso, verificamos:

$$PE(v) < PE(w)$$

Se essa desigualdade for verdadeira, marcamos vw como aresta de avanço.

Luís Felipe

02/04/26

Classificação das arestas: aresta de avanço

Se w já foi descoberto e já saiu da busca, então $PS(w) \neq 0$.
Nesse caso, verificamos:

$$PE(v) < PE(w)$$

Se essa desigualdade for verdadeira, marcamos vw como aresta de avanço.

Obs.:

- w foi descoberto depois de v;
- portanto, w pertence à região descendente de v na floresta DFS;
- porém, a aresta vw não foi a aresta usada para descobrir w.

Assim, a aresta aponta para um descendente já finalizado.

Luís Felipe
02/04/26

Classificação das arestas: aresta de avanço

Se w já foi descoberto e já saiu da busca, então $PS(w) \neq 0$.
Nesse caso, verificamos:

$$PE(v) < PE(w)$$

Se essa desigualdade for verdadeira, marcamos vw como aresta de avanço.

Obs.:

- w foi descoberto depois de v;
- portanto, w pertence à região descendente de v na floresta DFS;
- porém, a aresta vw não foi a aresta usada para descobrir w.

Assim, a aresta aponta para um descendente já finalizado.

Ou seja, uma aresta de avanço encurta um caminho já existente entre v e w.

Luís Felipe

02/04/26

Classificação das arestas: aresta de cruzamento

Se os casos anteriores não ocorrerem, então a aresta vw é classificada como aresta de cruzamento.

Isso acontece quando:

- w já foi descoberto ($PE(w) \neq 0$);
- w já foi finalizado ($PS(w) \neq 0$);
- e $PE(v) > PE(w)$.

Obs.:

- a aresta não liga v a um ancestral ativo;
- também não liga v a um descendente descoberto depois;
- ela conecta vértices de ramos diferentes da floresta DFS, ou subárvores já encerradas.

Luís Felipe

02/04/26

Classificação das arestas: aresta de cruzamento

Se os casos anteriores não ocorrerem, então a aresta vw é classificada como aresta de cruzamento.

Isso acontece quando:

- w já foi descoberto ($PE(w) \neq 0$);
- w já foi finalizado ($PS(w) \neq 0$);
- e $PE(v) > PE(w)$.

Obs.:

- a aresta não liga v a um ancestral ativo;
- também não liga v a um descendente descoberto depois;
- ela conecta vértices de ramos diferentes da floresta DFS, ou subárvores já encerradas.

Ou seja, uma aresta de cruzamento conecta partes independentes entre v e w.

Procedimento recursivo: finalização de um vértice

Depois que todos os sucessores de v foram analisados, encerramos sua exploração:

$$\begin{aligned} t &\leftarrow t+1 \\ PS(v) &\leftarrow t \end{aligned}$$

Obs.:

- o vértice v terminou de processar todos os seus arcos de saída;
- a partir desse momento, ele deixa de estar ativo na recursão;
- o valor $PS(v)$ registra o instante em que a exploração de v se encerrou.

Relembrando:

- $PE(v)$ marca a entrada em v;
- $PS(v)$ marca a saída de v.

Luís Felipe

02/04/26

Procedimento recursivo de busca p/ dígrafos

```
1   def P(v):
2       global t
3       t = t + 1
4       PE[v] = t
5
6       for w in Nout[v]:
7           if PE[w] == 0:
8               # aresta de arvore (azul)
9               pai[w] = v
10              P(w)
11
12          elif PS[w] == 0:
13              # aresta de retorno (vermelho)
14              pass
15
16          elif PE[v] < PE[w]:
17              # aresta de avanco (amarelo)
18              pass
19
20          else:
21              # aresta de cruzamento (verde)
22              pass
23
24      t = t + 1
25      PS[v] = t
```

Resumo da classificação das arestas

Ao analisar um arco vw em uma DFS de dígrafo:

- **Aresta de árvore**:

$$PE(w) = 0$$

- **Aresta de retorno**:

$$PE(w) \neq 0 \quad \text{e} \quad PS(w) = 0$$

- **Aresta de avanço**:

$$PS(w) \neq 0 \quad \text{e} \quad PE(v) < PE(w)$$

- **Aresta de cruzamento**:

$$PS(w) \neq 0 \quad \text{e} \quad PE(v) > PE(w)$$

Esses testes permitem classificar todas as arestas do dígrafo durante a execução da DFS.

Resumo da classificação das arestas

Ao analisar um arco vw em uma DFS de dígrafo:

- **Aresta de árvore**:

$$PE(w) = 0$$

- **Aresta de retorno**:

$$PE(w) \neq 0 \quad \text{e} \quad PS(w) = 0$$

- **Aresta de avanço**:

$$PS(w) \neq 0 \quad \text{e} \quad PE(v) < PE(w)$$

- **Aresta de cruzamento**:

$$PS(w) \neq 0 \quad \text{e} \quad PE(v) > PE(w)$$

Esses testes permitem classificar todas as arestas do dígrafo durante a execução da DFS.

Complexidade: $O(n + m)$.

Luís Felipe

02/04/26

DFS em dígrafos

- A DFS atribui a cada vértice um instante de entrada e um instante de saída.

Luís Felipe

DFS em dígrafos

- A DFS atribui a cada vértice um instante de entrada e um instante de saída.
- Esses tempos revelam a posição relativa dos vértices na floresta de profundidade.

Luís Felipe

DFS em dígrafos

- A DFS atribui a cada vértice um instante de entrada e um instante de saída.
- Esses tempos revelam a posição relativa dos vértices na floresta de profundidade.
- Com base nesses valores, cada arco pode ser classificado como:
  - aresta de árvore;
  - aresta de retorno;
  - aresta de avanço;
  - aresta de cruzamento.

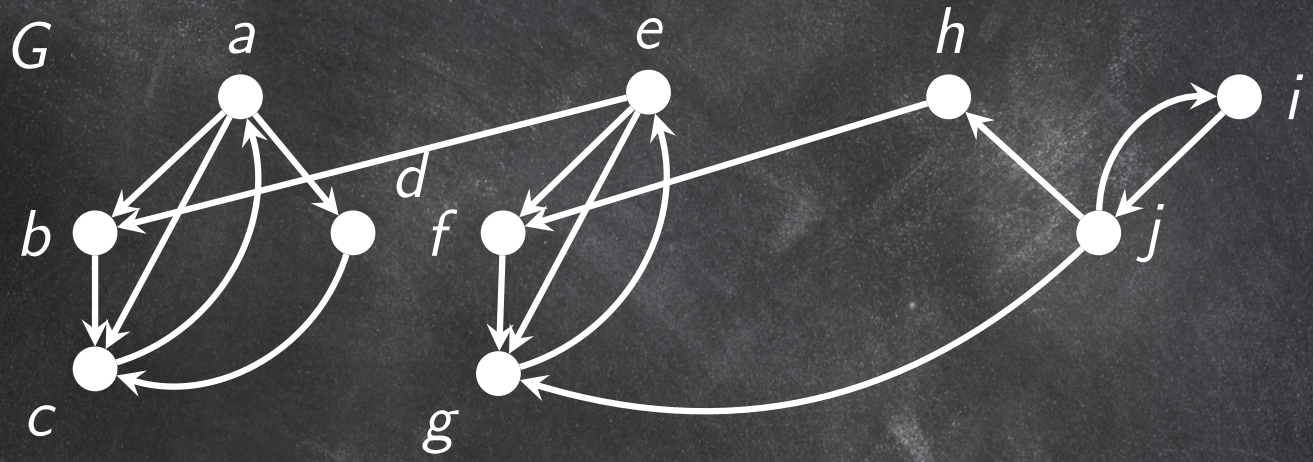
DFS em dígrafos

- A DFS atribui a cada vértice um instante de entrada e um instante de saída.
- Esses tempos revelam a posição relativa dos vértices na floresta de profundidade.
- Com base nesses valores, cada arco pode ser classificado como:
  - aresta de árvore;
  - aresta de retorno;
  - aresta de avanço;
  - aresta de cruzamento.
- Essa classificação é fundamental em vários algoritmos sobre dígrafos, como:
  - detecção de ciclos;
  - ordenação topológica;
  - componentes fortemente conexas.

Luís Felipe

02/04/26

DFS em grafos direcionados

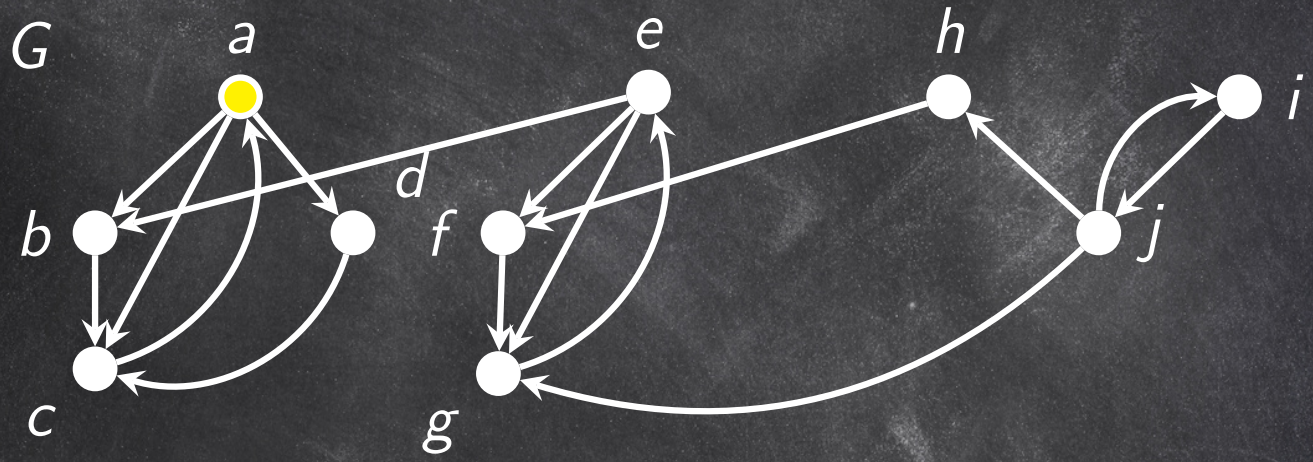


v	a	b	c	d	e	f	g	h	i	j
PE(v)	0	0	0	0	0	0	0	0	0	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

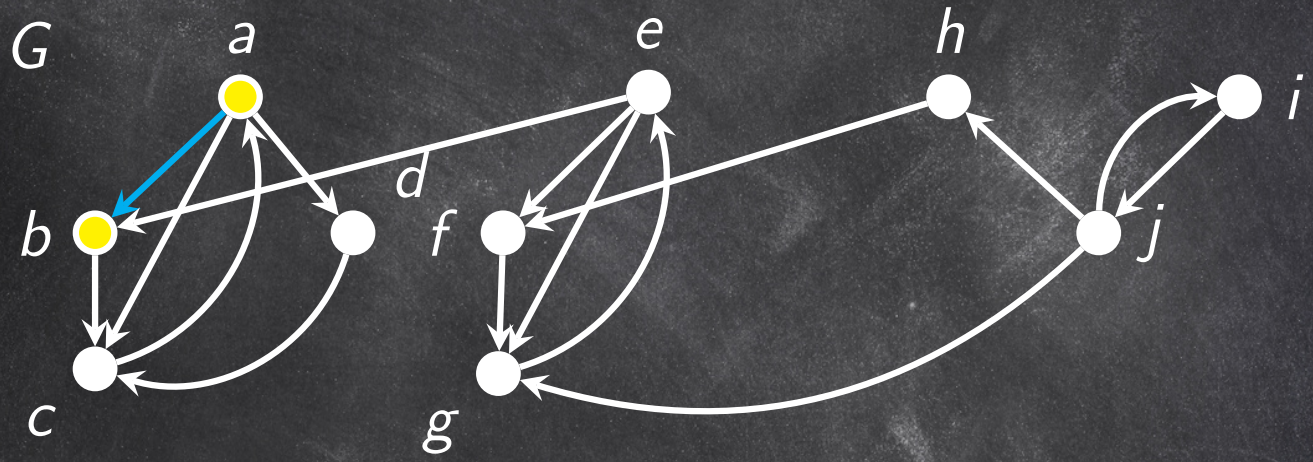


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	0	0	0	0	0	0	0	0	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

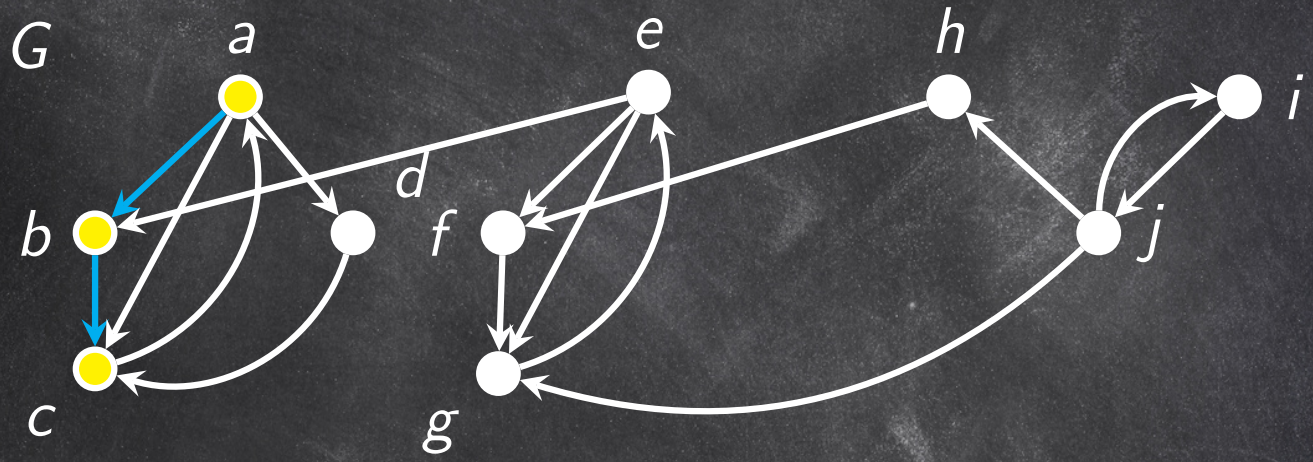


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	0	0	0	0	0	0	0	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

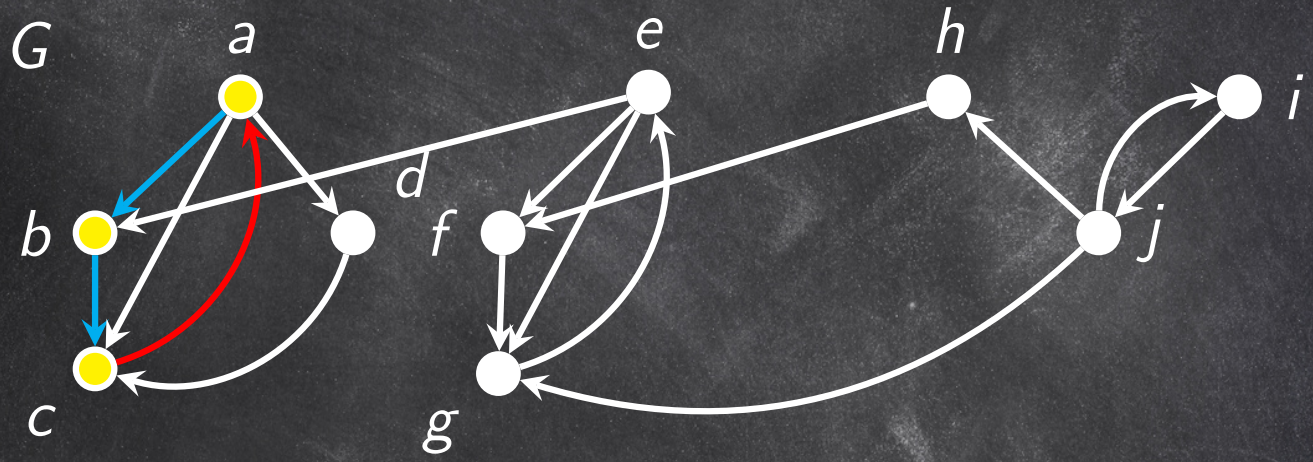


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	0	0	0	0	0	0	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

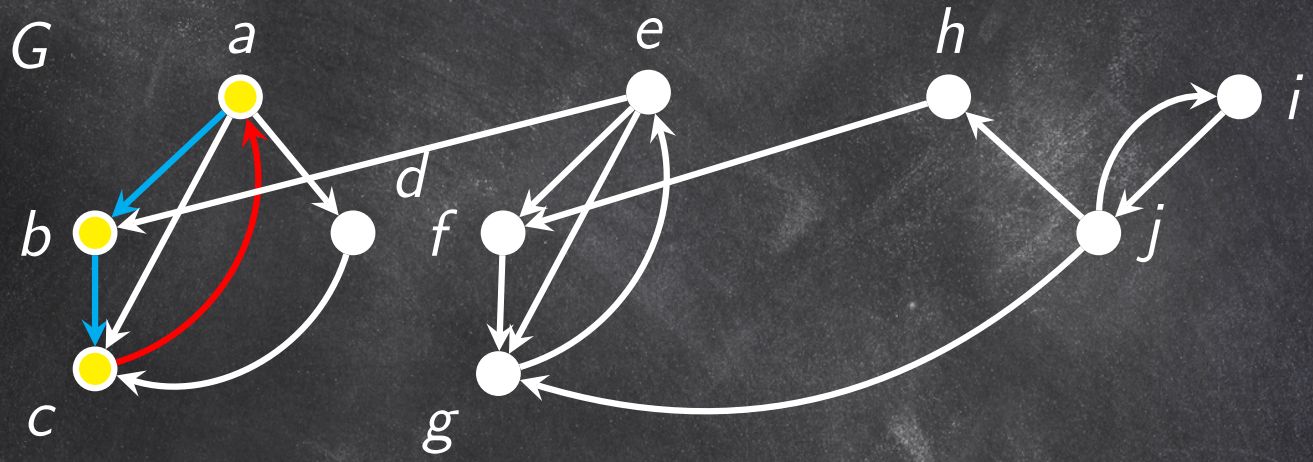


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	0	0	0	0	0	0	0
PS(v)	0	0	0	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

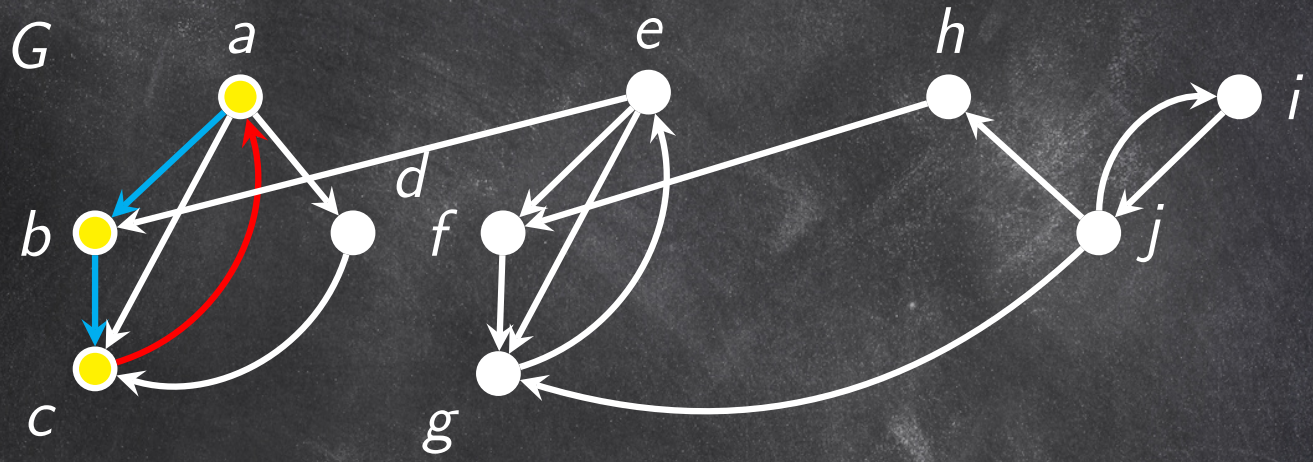


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	0	0	0	0	0	0	0
PS(v)	0	0	4	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

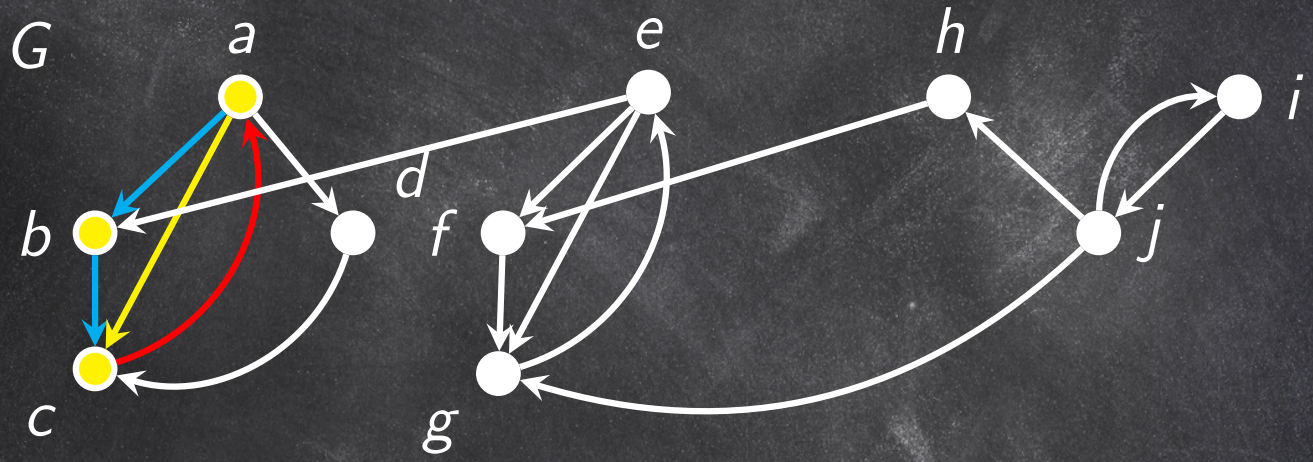


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	0	0	0	0	0	0	0
PS(v)	0	5	4	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

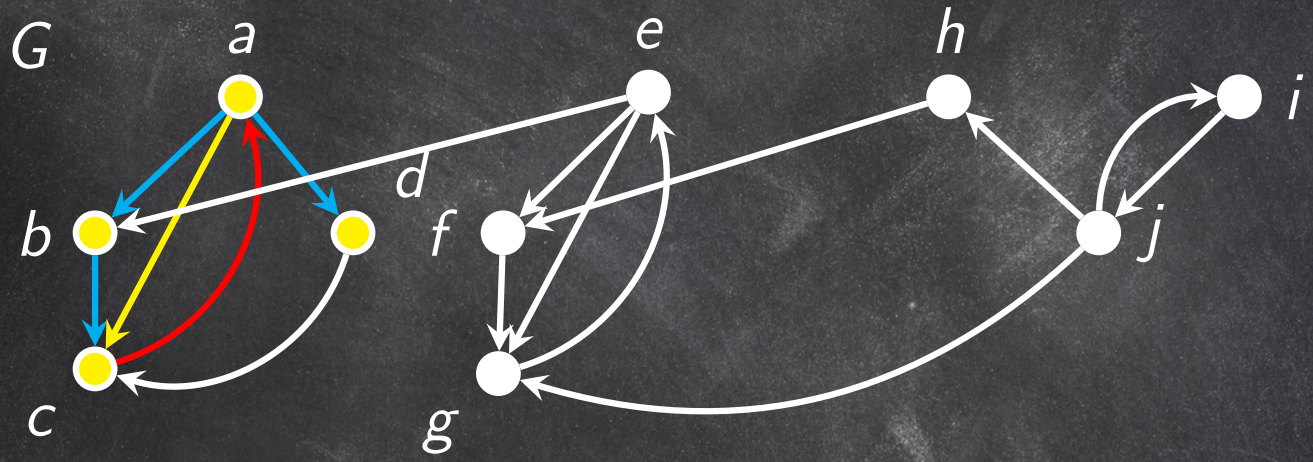


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	0	0	0	0	0	0	0
PS(v)	0	5	4	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

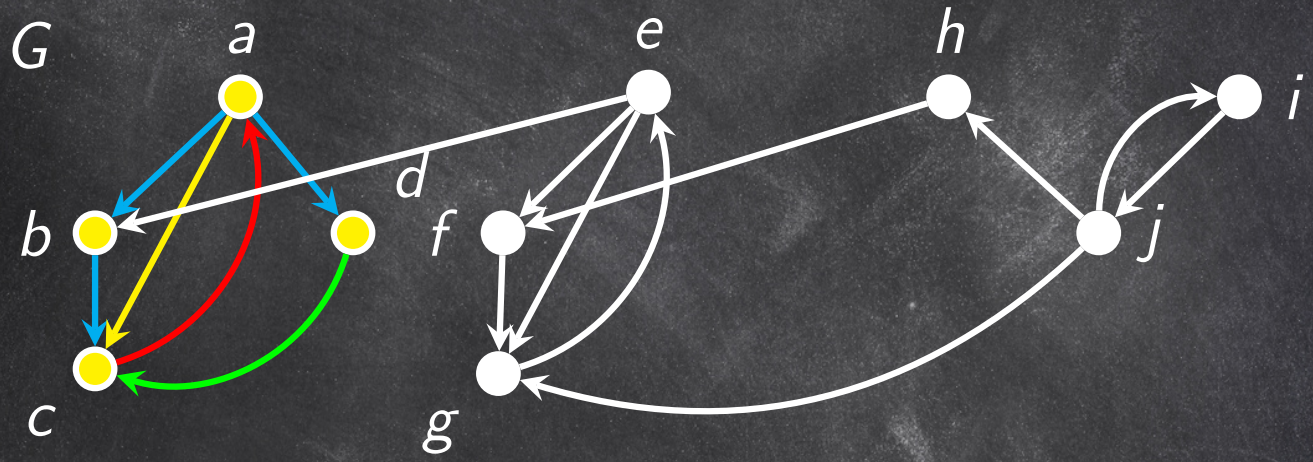


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	0	0	0	0	0	0
PS(v)	0	5	4	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

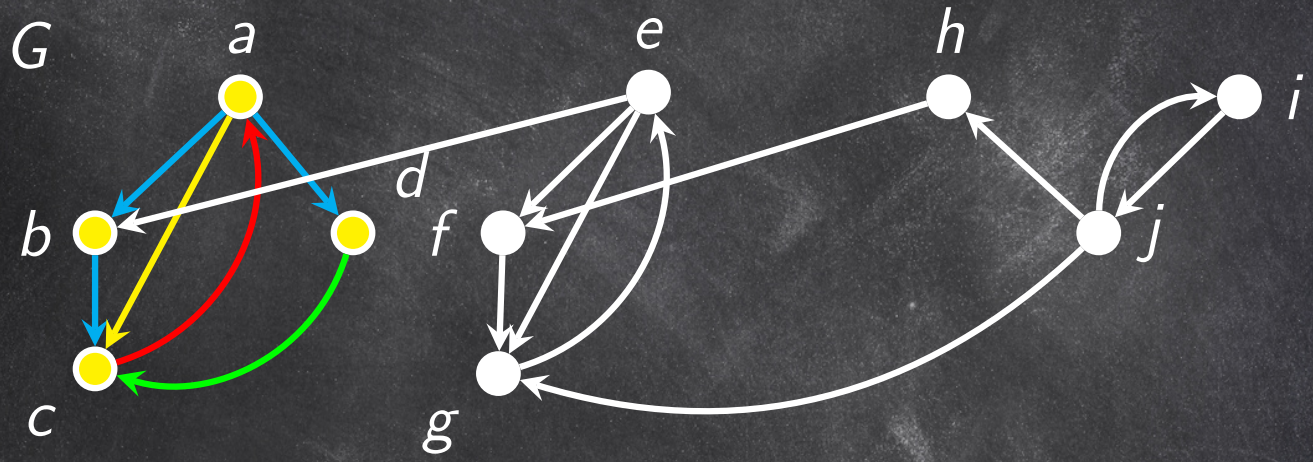


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	0	0	0	0	0	0
PS(v)	0	5	4	0	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

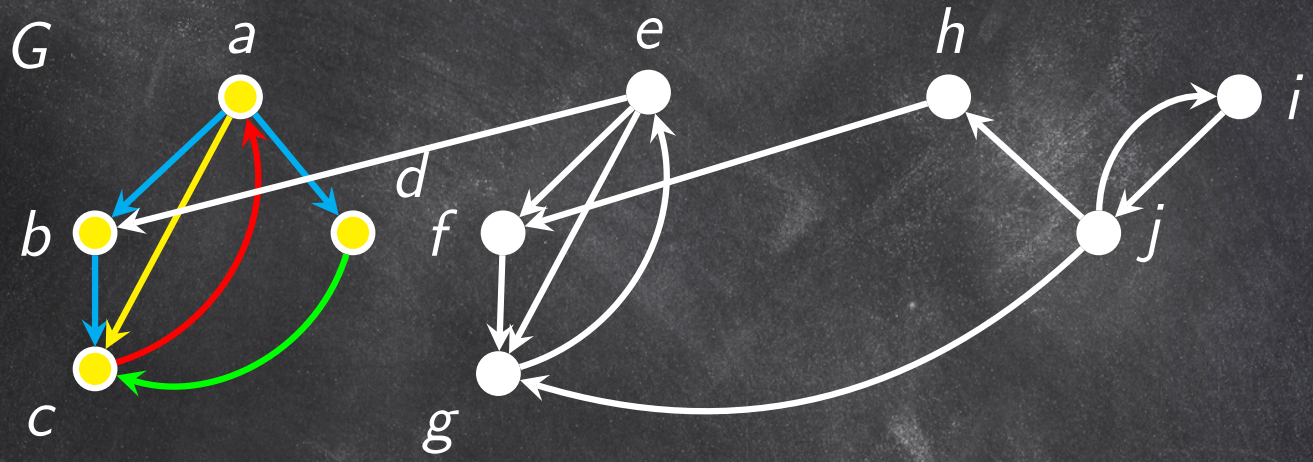


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	0	0	0	0	0	0
PS(v)	0	5	4	7	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

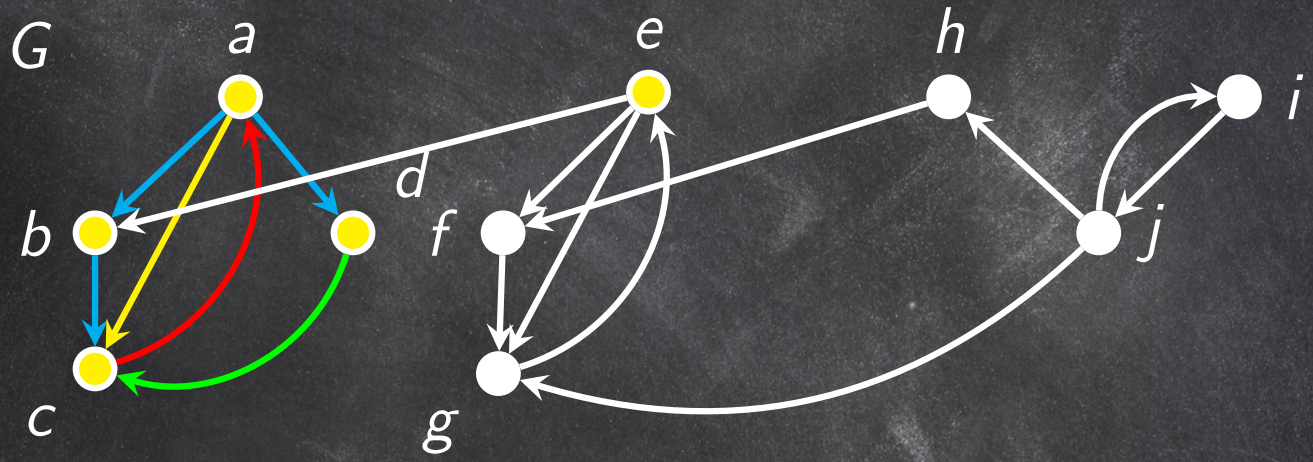


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	0	0	0	0	0	0
PS(v)	8	5	4	7	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

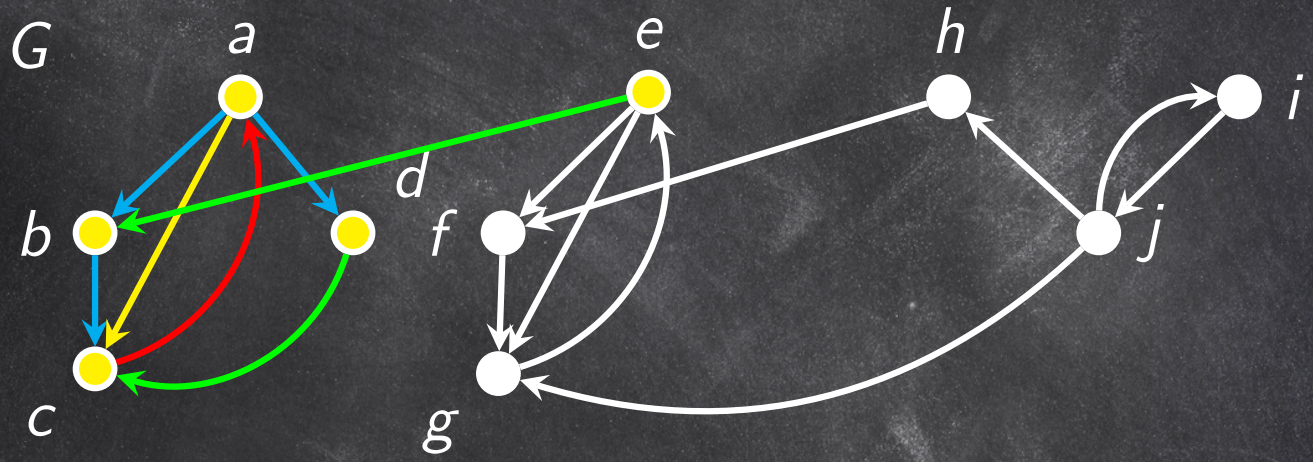


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	0	0	0	0	0
PS(v)	8	5	4	7	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

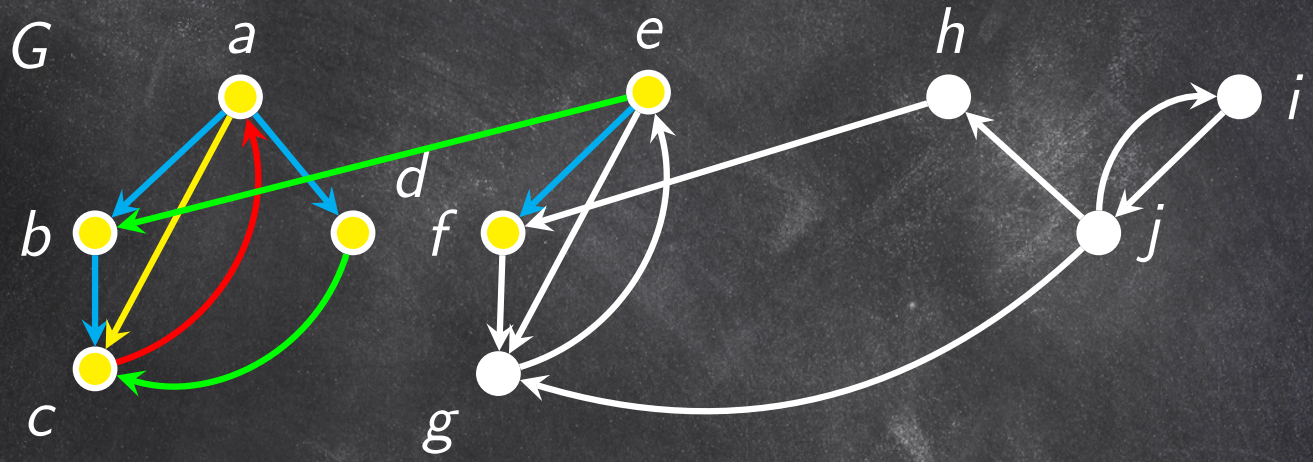


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	0	0	0	0	0
PS(v)	8	5	4	7	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

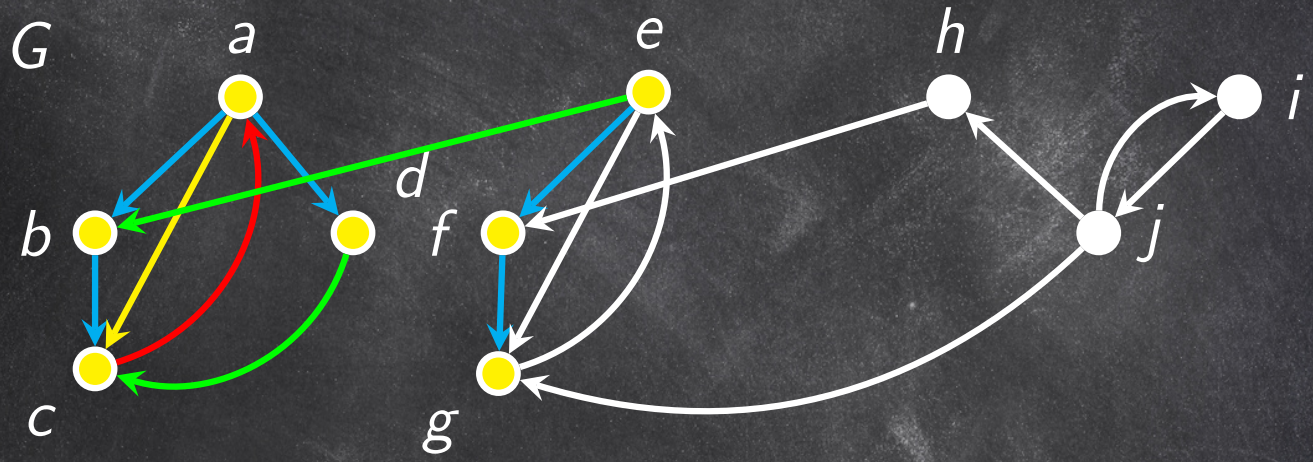


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	0	0	0	0
PS(v)	8	5	4	7	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

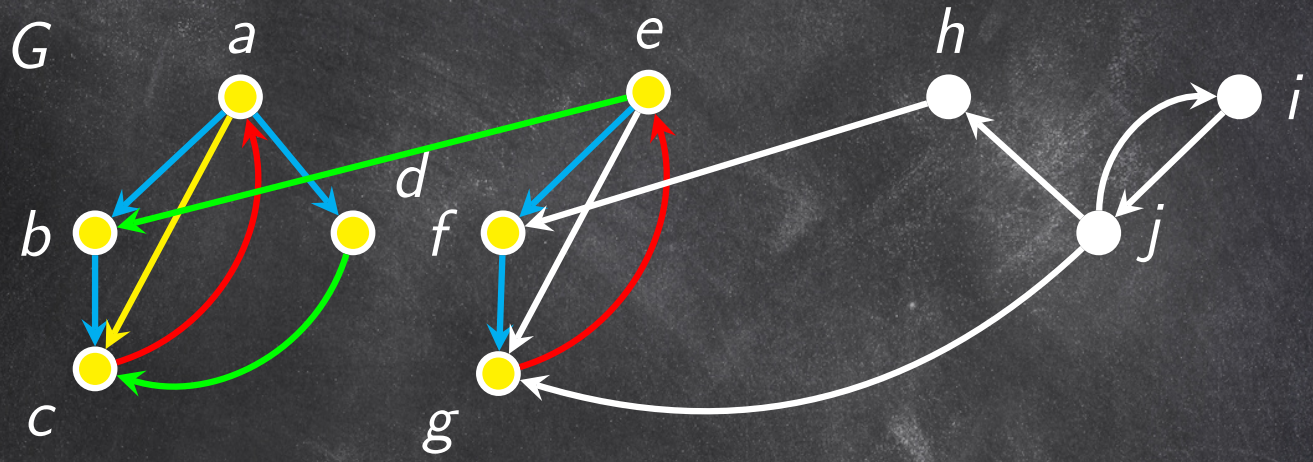


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	0	0	0
PS(v)	8	5	4	7	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

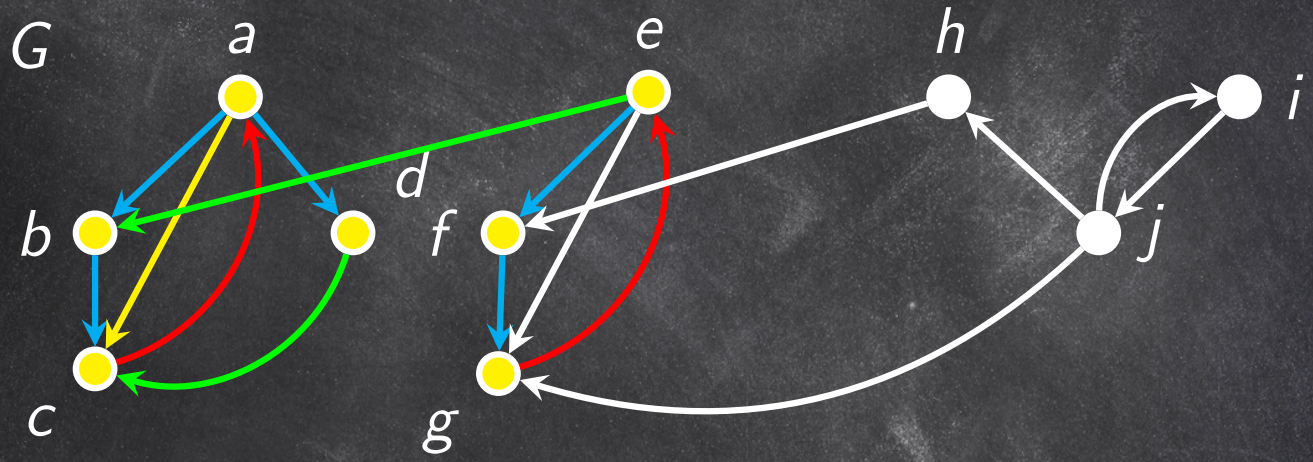


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	0	0	0
PS(v)	8	5	4	7	0	0	0	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

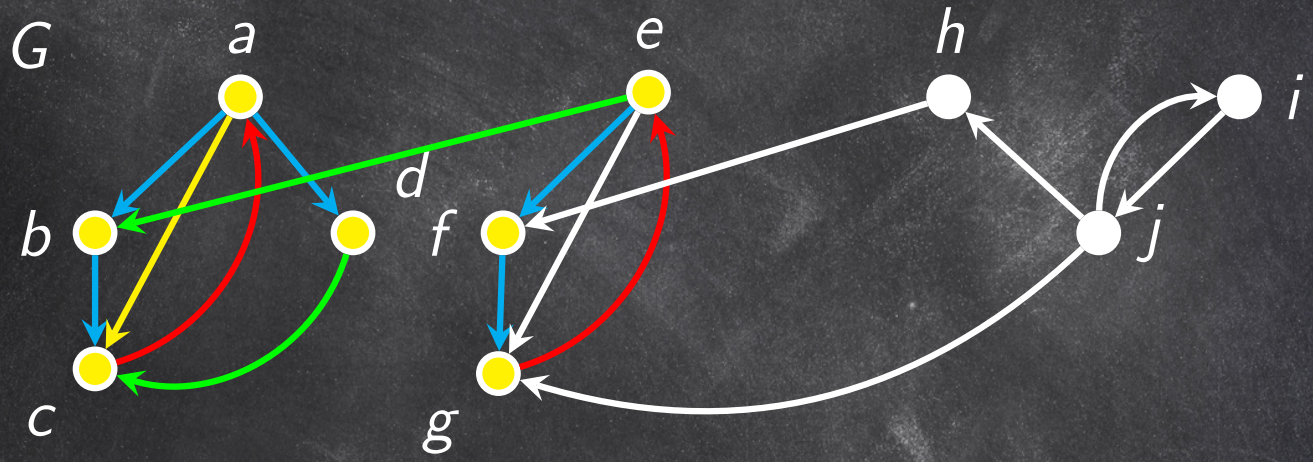


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	0	0	0
PS(v)	8	5	4	7	0	0	12	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

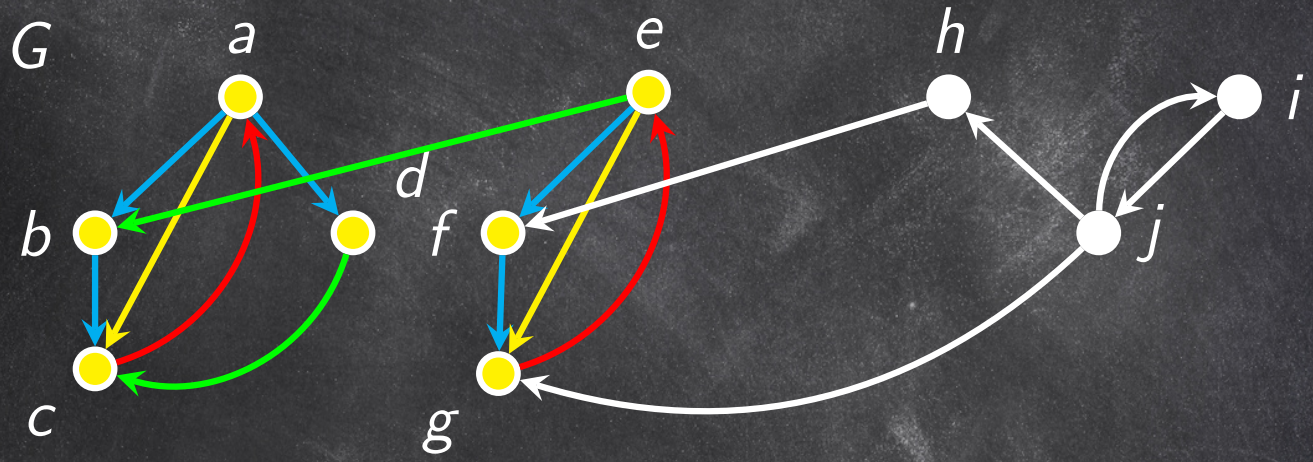


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	0	0	0
PS(v)	8	5	4	7	0	13	12	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

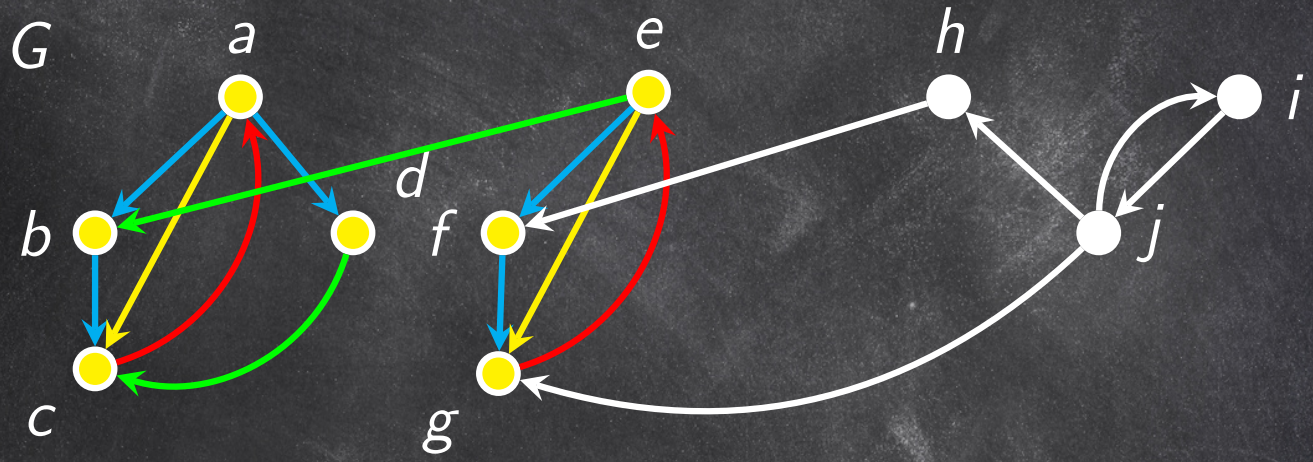


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	0	0	0
PS(v)	8	5	4	7	0	13	12	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

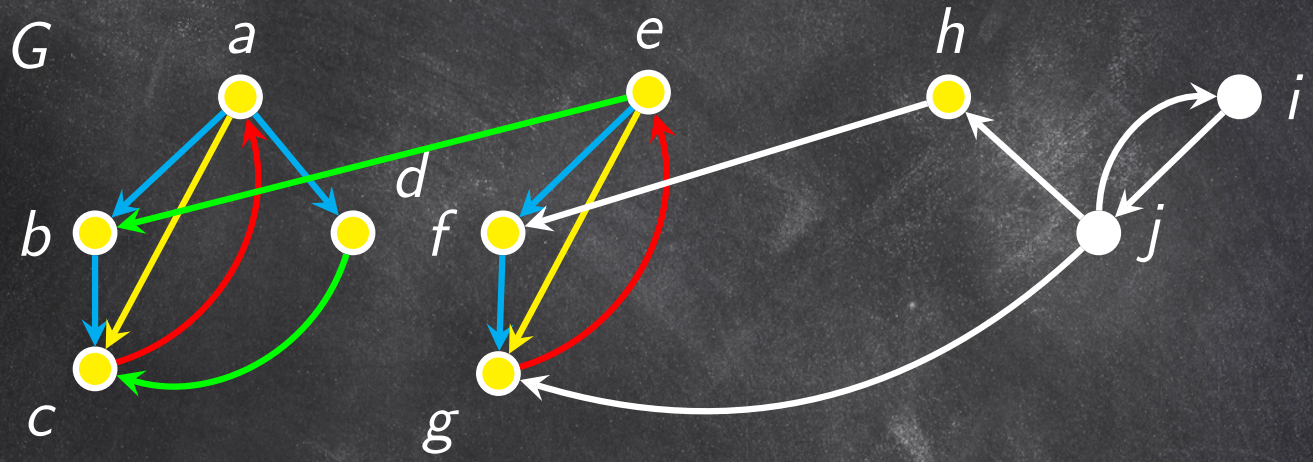


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	0	0	0
PS(v)	8	5	4	7	14	13	12	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

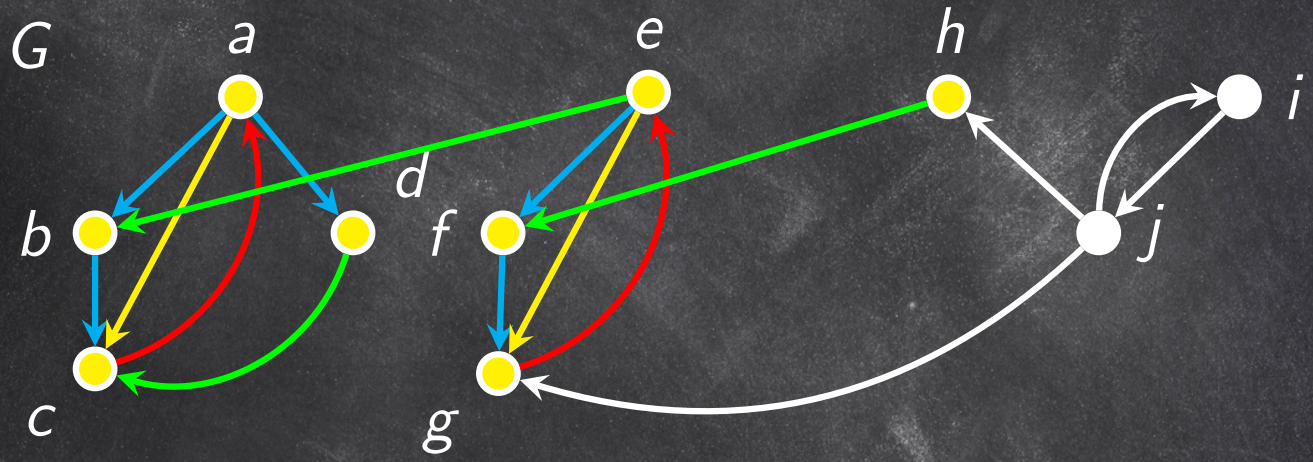


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	0	0
PS(v)	8	5	4	7	14	13	12	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

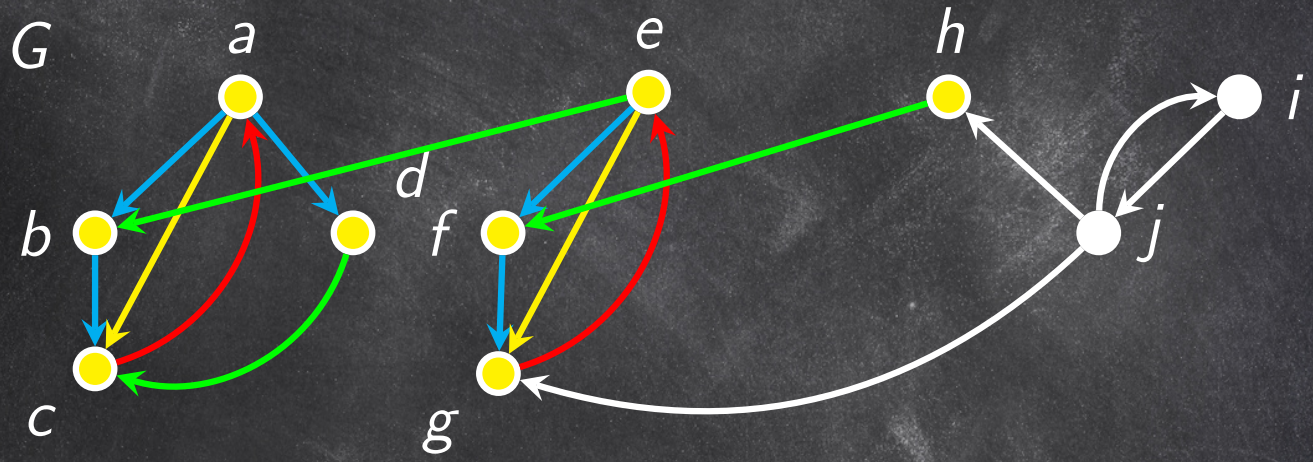


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	0	0
PS(v)	8	5	4	7	14	13	12	0	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

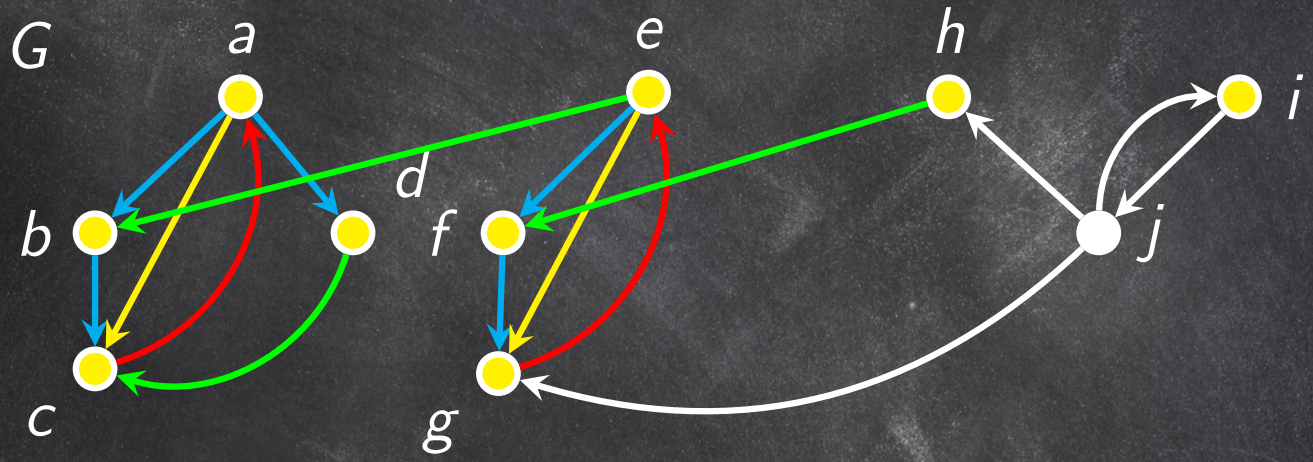


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	0	0
PS(v)	8	5	4	7	14	13	12	16	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

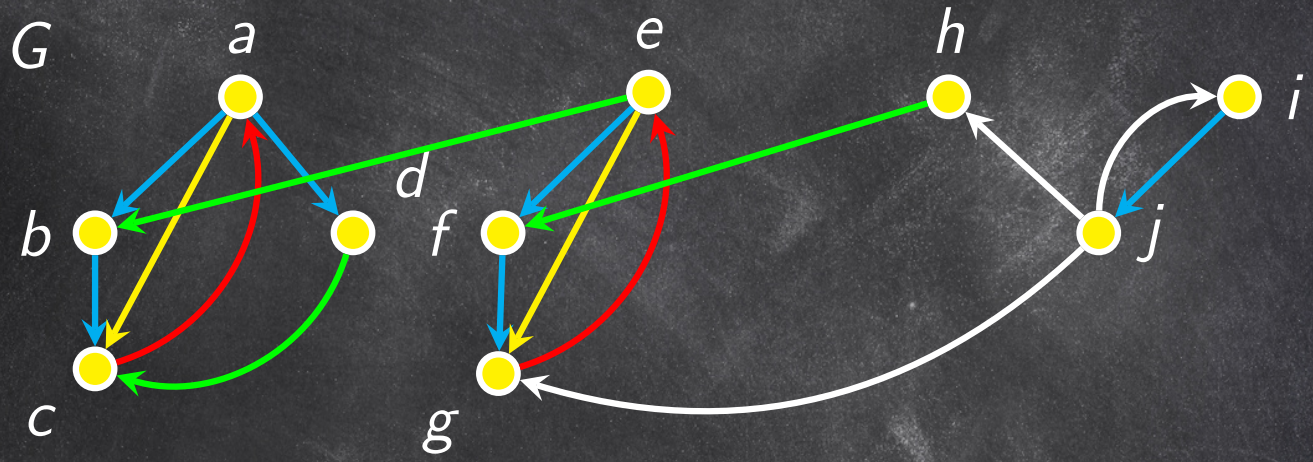


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	17	0
PS(v)	8	5	4	7	14	13	12	16	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

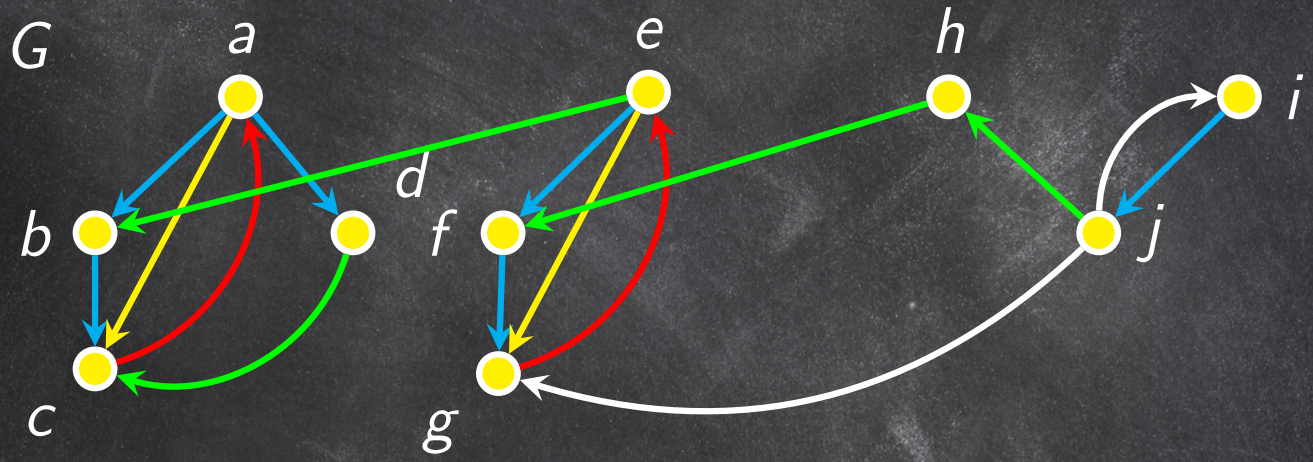


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	17	18
PS(v)	8	5	4	7	14	13	12	16	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

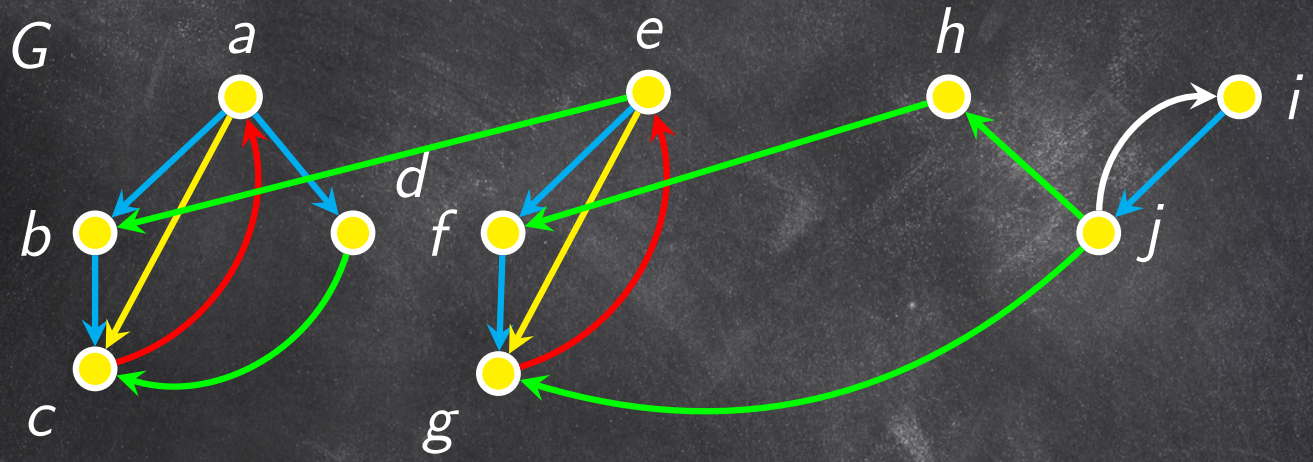


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	17	18
PS(v)	8	5	4	7	14	13	12	16	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

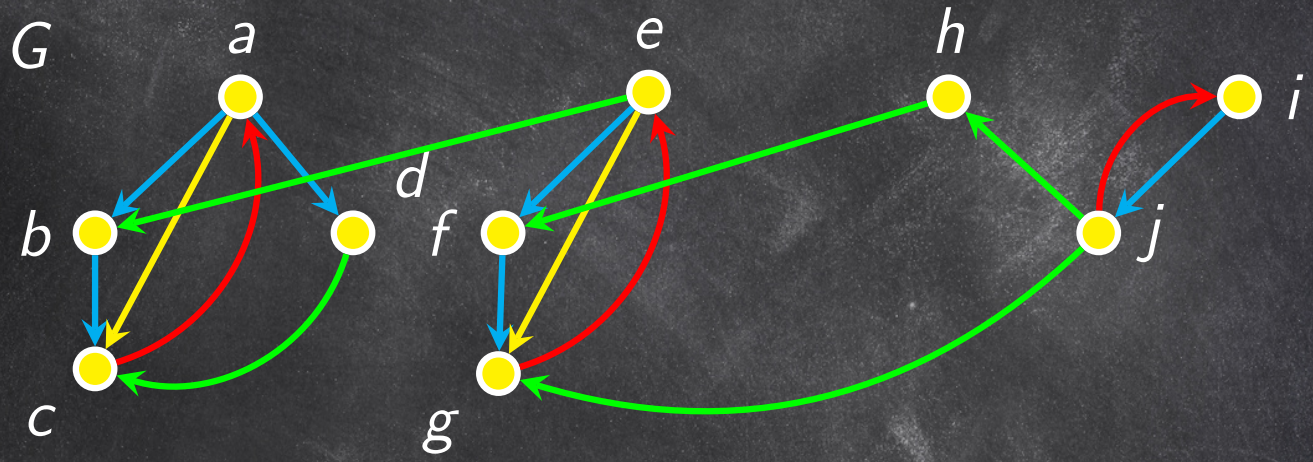


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	17	18
PS(v)	8	5	4	7	14	13	12	16	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

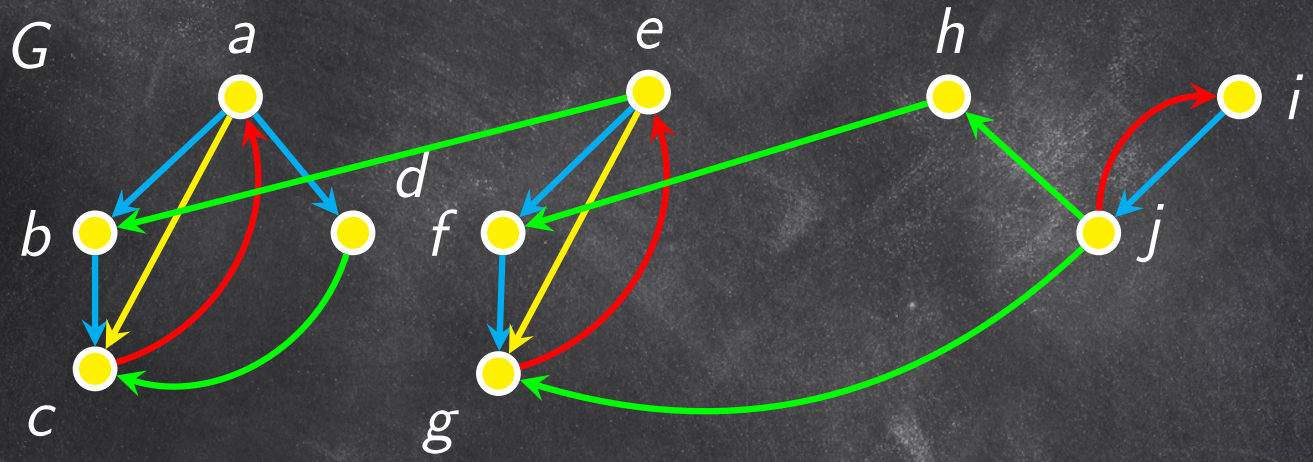


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	17	18
PS(v)	8	5	4	7	14	13	12	16	0	0

Luís Felipe

02/04/26

DFS em grafos direcionados

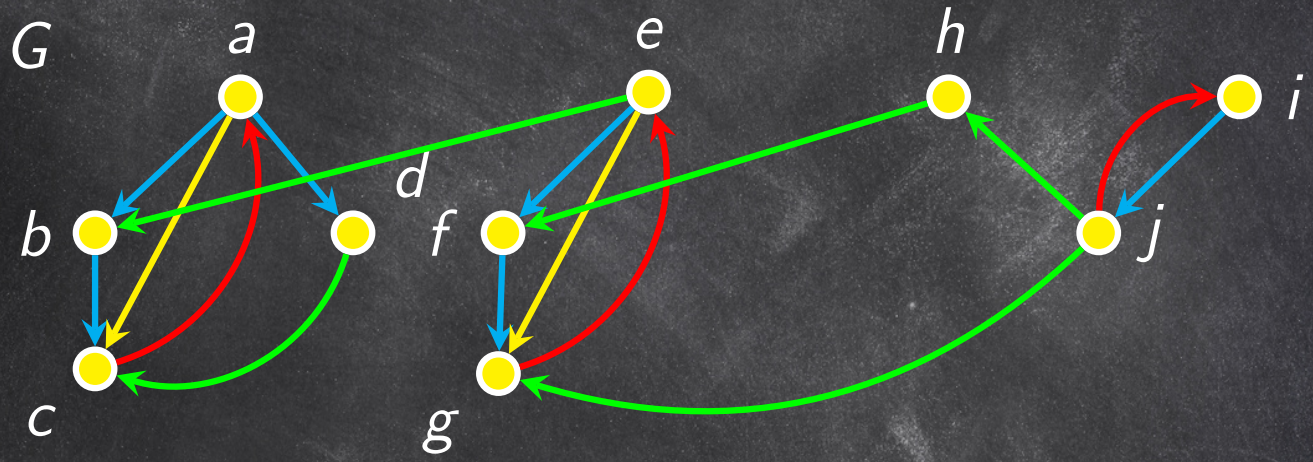


v	a	b	c	d	e	f	g	h	i	j
PE(v)	1	2	3	6	9	10	11	15	17	18
PS(v)	8	5	4	7	14	13	12	16	0	19

Luís Felipe

02/04/26

DFS em grafos direcionados



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

Luís Felipe

02/04/26

Floresta geradora

- A floresta (direcionada) de profundidade T é uma floresta geradora de G, ou seja, componentes conexas são árvores. Neste caso, todos os vértices de G são alcançados pela busca e ficam com uma PE diferente de zero no final do procedimento.

Luís Felipe

02/04/26

Floresta geradora

- A floresta (direcionada) de profundidade T é uma floresta geradora de G, ou seja, componentes conexas são árvores. Neste caso, todos os vértices de G são alcançados pela busca e ficam com uma PE diferente de zero no final do procedimento.
- Somente as arestas azuis (ligando pai e filho) pertencem à flores de profundidade T. As arestas de retorno (arestas vermelhas), de avanço (amarelas) e de cruzamento (verdes) não pertencem a T.

Luís Felipe

02/04/26

Floresta geradora

- A floresta (direcionada) de profundidade T é uma floresta geradora de G, ou seja, componentes conexas são árvores. Neste caso, todos os vértices de G são alcançados pela busca e ficam com uma PE diferente de zero no final do procedimento.

- Somente as arestas azuis (ligando pai e filho) pertencem à flores de profundidade T. As arestas de retorno (arestas vermelhas), de avanço (amarelas) e de cruzamento (verdes) não pertencem a T.

Propriedade:

- Toda aresta de retorno fecha um ciclo direcionado.
- Equivalentemente, toda aresta de retorno liga um vértice v a um de seus ancestrais na floresta de profundidade T.

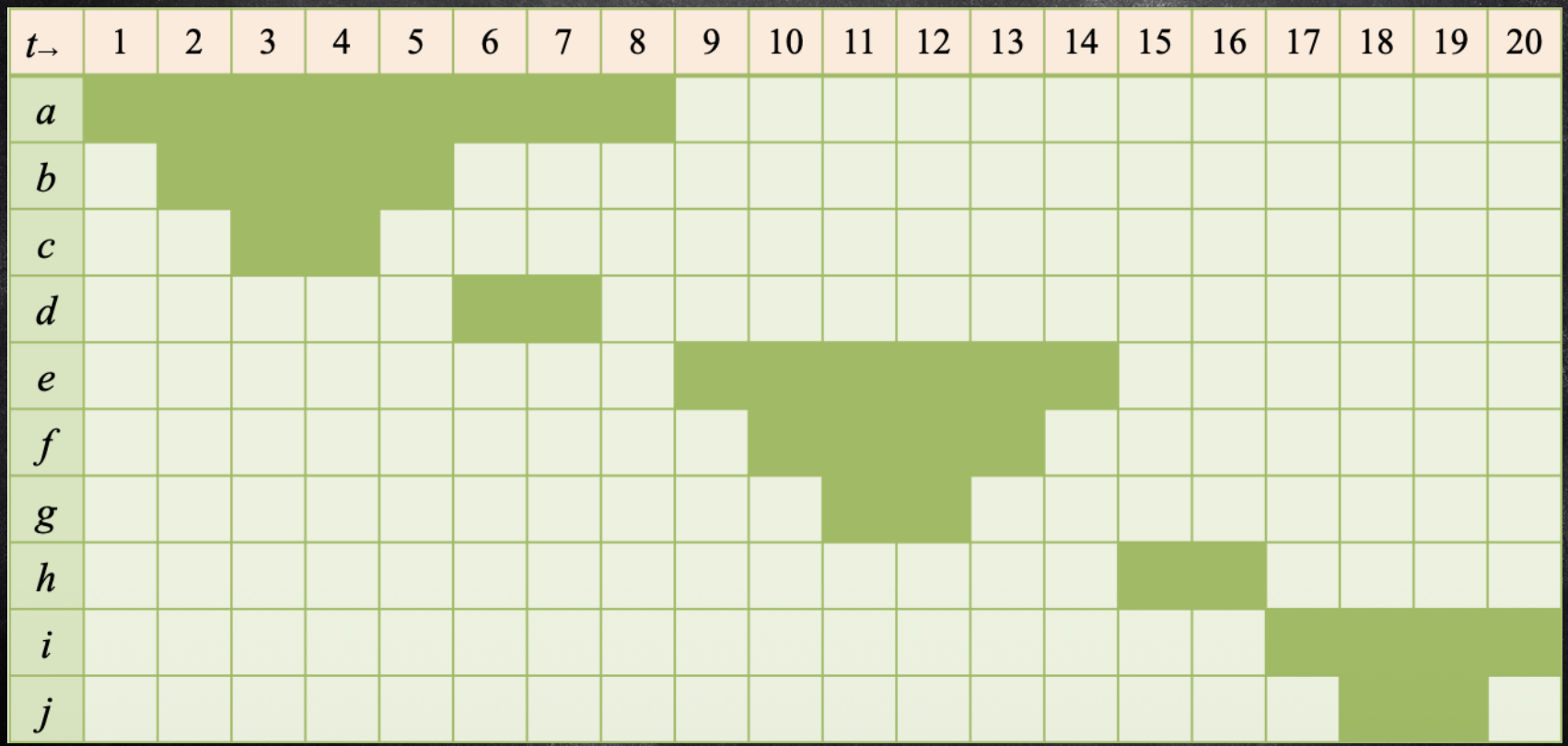
Luís Felipe

02/04/26

Intervalo de vida

Intervalo de vida de um vértice v: $I(v) = [PE(v), PS(v)]$

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



Luís Felipe

02/04/26

Caracterização das arestas da busca

Seja vw aresta de G e T sua árvore de profundidade. Temos que:

- Arestas de T (azuis): vw é uma aresta de T sse $I(v)$ contém $I(w)$ e, no momento da visita, $PE(w) = 0$.

- Arestas de avanço (amarelas): vw é uma aresta de avanço sse $I(v)$ contém $I(w)$ e, no momento da visita, $PE(w) \neq 0$.

- Arestas de retorno (vermelhas): vw é uma aresta de retorno sse $I(v)$ está contido em $I(w)$.

- Arestas de cruzamento (verdes): vw é uma aresta de cruzamento sse $I(v)$ está totalmente à direita de $I(w)$.

Luís Felipe

02/04/26

DFS iterativa em dígrafos

Até aqui, vimos a DFS em sua forma recursiva.

Luís Felipe

02/04/26

DFS iterativa em dígrafos

Até aqui, vimos a DFS em sua forma recursiva.
Agora, vamos reescrever a busca em profundidade de forma iterativa, usando uma pilha explícita.

Luís Felipe

02/04/26

DFS iterativa em dígrafos

Até aqui, vimos a DFS em sua forma recursiva.
Agora, vamos reescrever a busca em profundidade de forma iterativa, usando uma pilha explícita.
Ideia:

- a pilha de chamadas recursivas será substituída por uma pilha de vértices;
- o vértice no topo da pilha é o que está sendo processado no momento;
- os vértices da pilha representam o caminho ativo da busca.

Luís Felipe

02/04/26

Estruturas usadas

Nesta versão, manteremos:

- `marca[v]`: indica se v já foi visitado;
- `pilha`: contém os vértices ativos da DFS;
- `n_out[v]`: lista de sucessores de v ainda não examinados.

Luís Felipe

02/04/26

Estruturas usadas

Nesta versão, manteremos:

- `marca[v]`: indica se v já foi visitado;
- `pilha`: contém os vértices ativos da DFS;
- `n_out[v]`: lista de sucessores de v ainda não examinados.

Observação:

- aqui não precisamos guardar todos os tempos PE e PS;
- o foco passa a ser o andamento da busca e a detecção de ciclo.

Passo iterativo da DFS

Enquanto a pilha não estiver vazia:

1. olhamos o vértice v no topo;
2. se ainda houver sucessor não examinado de v, escolhemos um;
3. se esse sucessor w não foi visitado, empilhamos w;
4. caso contrário, seguimos examinando os demais sucessores de v;
5. se não restar mais nenhum sucessor, desempilhamos v.

Passo iterativo da DFS

Enquanto a pilha não estiver vazia:

1. olhamos o vértice v no topo;
2. se ainda houver sucessor não examinado de v, escolhemos um;
3. se esse sucessor w não foi visitado, empilhamos w;
4. caso contrário, seguimos examinando os demais sucessores de v;
5. se não restar mais nenhum sucessor, desempilhamos v.

Obs.: a busca desce quando empilha e sobe quando desempilha.

Por que alterar a lista de sucessores?

Na implementação recursiva, ao voltar de uma chamada, o algoritmo já sabe de qual sucessor deve continuar.

Por que alterar a lista de sucessores?

Na implementação recursiva, ao voltar de uma chamada, o algoritmo já sabe de qual sucessor deve continuar.
Na implementação iterativa, precisamos controlar isso explicitamente.

Por que alterar a lista de sucessores?

Na implementação recursiva, ao voltar de uma chamada, o algoritmo já sabe de qual sucessor deve continuar.
Na implementação iterativa, precisamos controlar isso explicitamente.

Uma forma simples é:

- manter, para cada vértice v, a lista de sucessores ainda não examinados;
- remover um sucessor dessa lista cada vez que ele for considerado.

Por que alterar a lista de sucessores?

Na implementação recursiva, ao voltar de uma chamada, o algoritmo já sabe de qual sucessor deve continuar.
Na implementação iterativa, precisamos controlar isso explicitamente.

Uma forma simples é:

- manter, para cada vértice v, a lista de sucessores ainda não examinados;
- remover um sucessor dessa lista cada vez que ele for considerado.

Logo:

$$n\_out[v] = \emptyset \quad \Longrightarrow \quad v \text{ terminou.}$$

Luís Felipe

02/04/26

Detecção de ciclo

Durante a DFS, pode acontecer de uma aresta vw apontar para um vértice w que ainda está na pilha.

Luís Felipe

02/04/26

Detecção de ciclo

Durante a DFS, pode acontecer de uma aresta vw apontar para um vértice w que ainda está na pilha.
Isso significa que w é um ancestral ativo de v.

Luís Felipe

02/04/26

Detecção de ciclo

Durante a DFS, pode acontecer de uma aresta vw apontar para um vértice w que ainda está na pilha.
Isso significa que w é um ancestral ativo de v.
Portanto, já existe um caminho de w até v na árvore DFS, e a aresta vw fecha um ciclo.

Luís Felipe

02/04/26

Detecção de ciclo

Durante a DFS, pode acontecer de uma aresta vw apontar para um vértice w que ainda está na pilha.
Isso significa que w é um ancestral ativo de v.
Portanto, já existe um caminho de w até v na árvore DFS, e a aresta vw fecha um ciclo.

$$w \in \texttt{pilha} \implies vw \text{ é aresta de retorno} \implies \text{há ciclo.}$$

Luís Felipe

02/04/26

Pseudocódigo da etapa iterativa

```
1   while pilha != []:                      # enquanto ainda há vértices ativos
2       v = pilha[-1]                       # topo da pilha
3       marca[v] = 1                        # marca v como visitado
4
5       if n_out[v] != []:                  # ainda existem sucessores de v
6           w = n_out[v][0]                 # escolhe um sucessor
7           del n_out[v][0]                 # remove da lista
8
9           if marca[w] == 0:               # se w não foi visitado
10              pilha.append(w)             # empilha w
11              na_pilha[w] = 1             # marca w como ativo
12          elif na_pilha[w] == 1:          # se w ainda está na pilha
13              ciclo = True                # aresta de retorno, implica em ciclo
14      else:
15          na_pilha[v] = 0                 # v deixa de estar ativo
16          pilha.pop()                     # desempilha v
```

Luís Felipe

02/04/26

Pseudocódigo da etapa iterativa

```
1   while pilha != []:                  # enquanto ainda há vértices ativos
2       v = pilha[-1]                   # topo da pilha
3       marca[v] = 1                    # marca v como visitado
4
5       if n_out[v] != []:              # ainda existem sucessores de v
6           w = n_out[v][0]             # escolhe um sucessor
7           del n_out[v][0]             # remove da lista
8
9           if marca[w] == 0:           # se w não foi visitado
10              pilha.append(w)         # empilha w
11              na_pilha[w] = 1         # marca w como ativo
12          elif na_pilha[w] == 1:      # se w ainda está na pilha
13              ciclo = True            # aresta de retorno, implica em ciclo
14      else:
15          na_pilha[v] = 0             # v deixa de estar ativo
16          pilha.pop()                 # desempilha v
```

O topo da pilha é sempre o vértice atualmente em processamento.

Luís Felipe

02/04/26

DFS completa

Como o dígrafo pode não ser conexo, repetimos o processo para cada vértice ainda não visitado.

```
1   na_pilha = [0]*n                        # indica se o vértice está atualmente na pilha
2
3   for a in range(n):                      # percorre todos os vértices (DFS completa)
4       if marca[a] == 0:                   # se ainda não foi visitado
5           pilha = [a]                     # inicia nova árvore da floresta
6           na_pilha[a] = 1                 # marca a como ativo (na pilha)
7
8           while pilha != []:              # enquanto houver vértices ativos
9               v = pilha[-1]               # topo da pilha (vértice atual)
10              marca[v] = 1                # marca v como visitado
11
12              if n_out[v] != []:          # ainda há sucessores de v não examinados
13                  w = n_out[v][0]         # escolhe um sucessor w
14                  del n_out[v][0]         # remove w da lista (evita reprocessar)
15
16                  if marca[w] == 0:       # se w ainda não foi visitado
17                      pilha.append(w)     # desce na DFS (empilha w)
18                      na_pilha[w] = 1     # marca w como ativo
19
20                  elif na_pilha[w] == 1:  # se w ainda está na pilha (ativo)
21                      ciclo = True        # aresta de retorno, implica em ciclo
22
23              else:                       # não restam sucessores
24                  na_pilha[v] = 0         # v deixa de estar ativo
25                  pilha.pop()             # sobe na DFS (desempilha v)
```

Luís Felipe

02/04/26

Correspondência com a versão recursiva

Recursiva	Iterativa
chamar $P(w)$	empilhar w
retornar de $P(w)$	desempilhar w
vértices ativos	vértices na pilha
aresta de retorno	aresta para vértice na pilha

A lógica da DFS é a mesma; muda apenas a forma de controlar a execução.