# Grafos: Busca em Profundidade

SCC0216 Modelagem Computacional em Grafos

Thiago A. S. Pardo
Maria Cristina F. Oliveira

# DFS - algoritmo

```
procedure DFS(G, v) is
    label v as gray
    for all edges (v,w) in G.adjacentEdges(v) do
        if vertex w not labeled as gray then
            call DFS(G, w)
    label v as black
```

# DFS

- Implemente a busca em profundidade

# DFS - algoritmo

- Costuma-se registrar o tempo de descoberta (cinza) e o tempo de término da busca (preto) de cada vértice
- Também é conveniente registrar o antecessor de cada vértice no caminho DFS a partir do vértice inicial

# Implementação DFS

- Implementação a seguir usa 4 vetores de tamanho $n = |V|$
  - **cor**[i]: cor do vértice *i* (*branco* no início, *cinza* assim que descoberto, *preto* assim que processado)
  - **antecessor**[i]: o vértice antecessor de *i* no trajeto DFS a partir do vértice inicial
  - **d**[i]: o momento da descoberta do vértice *i* (cinza)
  - **t**[i]: o momento do processamento do vértice *i* (preto)

```
/* função para busca em profundidade, utiliza função auxiliar visita_dfs
*/

void busca_profundidade(Grafo *G) {
    ...
    int d[MaxNumVertices], t[MaxNumVertices], antecessor[MaxNumVertices];
    TipoCor cor[MaxNumVertices];

                                        .
                                        .
                                        .

}
```

# Implementação DFS

- Usamos a mesma estrutura de dados Grafo que já usamos no BFS

- Idem para as funções auxiliares *ListaAdjVazia()*, *PrimeiroListaAdj()*, *ProxAdj()*, que fazem parte de uma implementação do TAD Grafo...

  - Ver Tópico 10 no Moodle, código BFS

# Implementação DFS

- A função *void busca_profundidade(Grafo *G)* inicializa o processo:
  - A *cor* de todos os vértices é inicializada como *branco*
  - *O antecessor* de todos os vértices é inicializado com *-1*

  - Em seguida (dentro do 2º. laço) chama a função *visita_dfs()*, que implementa o percurso DFS a partir de um vértice inicial, no caso o vértice 0
    - Esse 2º. laço *for (V= 0; ... V++)* garante que todos os vértices serão visitados, mesmo que o grafo tenha mais de uma componente conexa!

```
/* função para busca em profundidade, utiliza função auxiliar visita_dfs
*/

void busca_profundidade(Grafo *G) {
    int V, tempo;
    int d[MaxNumVertices], t[MaxNumVertices], antecessor[MaxNumVertices];
    TipoCor cor[MaxNumVertices];

    printf("*** Sequencia de nos visitados na busca em profundidade ***\n\n");

    tempo= 0;
    for (V= 0; V < G->NumVertices; V++) {
        cor[V]= branco;
        antecessor[V]= -1;
    }

    for (V= 0; V< G->NumVertices; V++)
        if (cor[V] == branco)
            visita_dfs(G, V, &tempo, d, t, cor, antecessor);
}
```

# Implementação DFS

- A função *visita_dfs()* é recursiva, nos moldes do pseudocódigo visto...

```
procedure DFS(G, v) is
    label v as gray
    for all edges (v,w) in G.adjacentEdges(v) do
        if vertex w not labeled as gray then
            call DFS(G, w)
    label v as black
```

  - 'Descobre' o vértice atual, segue adiante no caminho seguindo para o seu primeiro vértice adjacente que ainda não foi descoberto...
  - Quando todos os adjacentes tiverem sido descobertos a execução atual é retomada e o processamento desse vértice é finalizado

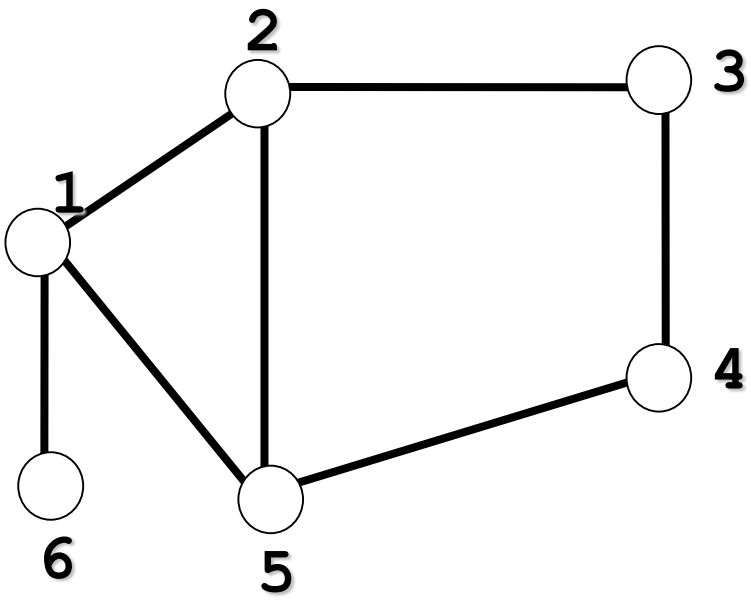
# Implementação DFS

- A função *visita_dfs()* também registra o vértice antecessor e os tempos de descoberta e de processamento de cada vértice
- Parâmetros da função
  - Grafo *G: a estrutura de dados que representa o grafo
  - **int** V: o vértice inicial da busca
  - **int** *tempo: a variável que registra o tempo (passo)
  - **int** d[], **int** t[], **TipoCor** cor[], **int** antecessor[]: os vetores que registram, para cada vértice, o tempo da descoberta, o tempo do processamento, a cor e o antecessor no caminho. Os conteúdos dos vetores cor[] e antecessor[] foram inicializados antes da chamada, e sofrem alterações dentro da função. Os vetores d[] e t[] são alterados dentro da função.

```
void visita_dfs(Grafo *G, int V, int *tempo, int d[], int t[], TipoCor cor[], int antecessor[])
{
    int FimListaAdj, erro;
    no_lista *Adj, *Aux;

    cor[V]= cinza;     (*tempo)++;     d[V]= *tempo;

    if (!ListaAdjVazia(G, V, &erro)) {
       Aux= PrimeiroListaAdj(G, V, &erro);
       FimListaAdj= 0;
       while (!FimListaAdj) {
            ProxAdj(G, &Adj, &Aux, &FimListaAdj);
            if (cor[Adj->v] == branco) {
               antecessor[Adj->v]= V;
               visita_dfs(G, Adj->v, tempo, d, t, cor, antecessor);
            }
       }
    }

    cor[V]=preto;       (*tempo)++;             t[V]=*tempo;

    printf("No %d, descoberta=%d, termino=%d, antecessor=%d\n", V, d[V], t[V], antecessor[V]);
}
```

# Implementação DFS

- Execução *busca_profundidade(Grafo *G)*

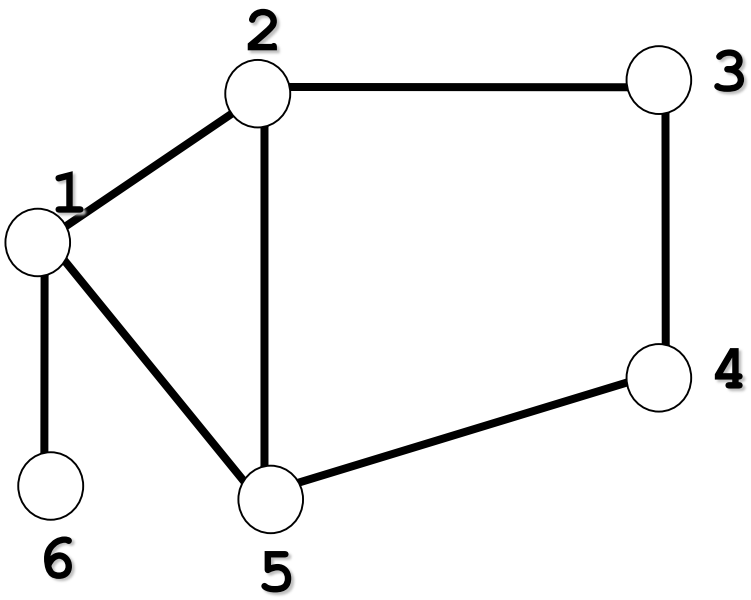



*** Sequencia de nos visitados na busca em profundidade ***

No 5, descoberta=5, termino=6, antecessor=4
No 4, descoberta=4, termino=7, antecessor=3
No 3, descoberta=3, termino=8, antecessor=2
No 2, descoberta=2, termino=9, antecessor=1
No 6, descoberta=10, termino=11, antecessor=1
No 1, descoberta=1, termino=12, antecessor=-1
Pressione qualquer tecla para continuar. . .

# Implementação DFS

- Execução *busca_profundidade(Grafo *G)*




*** Sequencia de nos visitados na busca em profundidade ***

No 5, descoberta=5, termino=6, antecessor=4
No 4, descoberta=4, termino=7, antecessor=3
No 3, descoberta=3, termino=8, antecessor=2
No 2, descoberta=2, termino=9, antecessor=1
No 6, descoberta=10, termino=11, antecessor=1
No 1, descoberta=1, termino=12, antecessor=-1
Pressione qualquer tecla para continuar. . .

Basta olhar a sequencia de antecessores para saber o caminho a partir do vértice inicial até um vértice qualquer, p.ex.:
Caminho DFS de 1 a 5: 5 4 3 2 1
Caminho DFS de 1 a 6: 6 1

# Complexidade do DFS

O(|V| + |A|)

- A função *visita_dfs* é chamada exatamente uma vez para cada vértice de V
- Na função *visita_dfs*, o laço é executado |adj[v]| vezes, i.e., será executado O(|A|) vezes no total

```
void busca_profundidade(Grafo *G) {

            ...

    for (V= 0; V< G->NumVertices; V++)
        if (cor[V] == branco)
            visita_dfs(G, V, &tempo, d, t, cor, antecessor);
}

void visita_dfs(Grafo *G, int V, int *tempo, int d[], int t[], TipoCor cor[], int antecessor[])
{

                        ...

while (!FimListaAdj) {
            ProxAdj(G, &Adj, &Aux, &FimListaAdj);
            if (cor[Adj->v] == branco) {
               antecessor[Adj->v]= V;
               visita_dfs(G, Adj->v, tempo, d, t, cor, antecessor);
            }
      }
   }
                        ...
}
```

# Arestas

- É possível classificar as arestas durante a busca em profundidade

- 4 categorias:

  - Aresta de árvore
  - Aresta de retorno
  - Aresta de avanço
  - Aresta de cruzamento

# Arestas

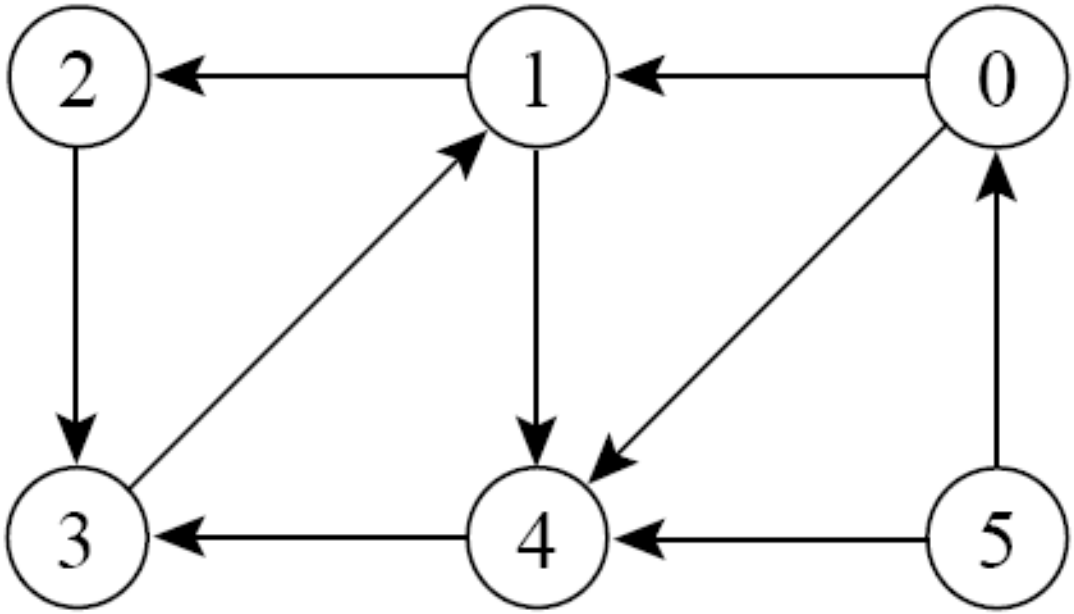
- Tipos de arestas
  - Aresta de árvore: ($u$,$v$) é uma **aresta de árvore** se $v$ é descoberto na busca em profundidade percorrendo-se a aresta ($u$,$v$)
    - $v$ é branco
  - Aresta de retorno: ($u$,$v$) é uma **aresta de retorno** se conecta $u$ com um ancestral $v$ na árvore
    - $v$ é cinza

# Arestas

- Tipos de arestas
  - Aresta de avanço: (*u*,*v*) não pertence à árvore, mas conecta *u* a um descendente *v* na árvore
    - d[u] < d[v] (*u* é descoberto antes de *v*)
  - Aresta de cruzamento: demais arestas, podem conectar vértices na mesma árvore ou em árvores diferentes
    - d[u] > d[v] (*u* é descoberto depois de *v*)

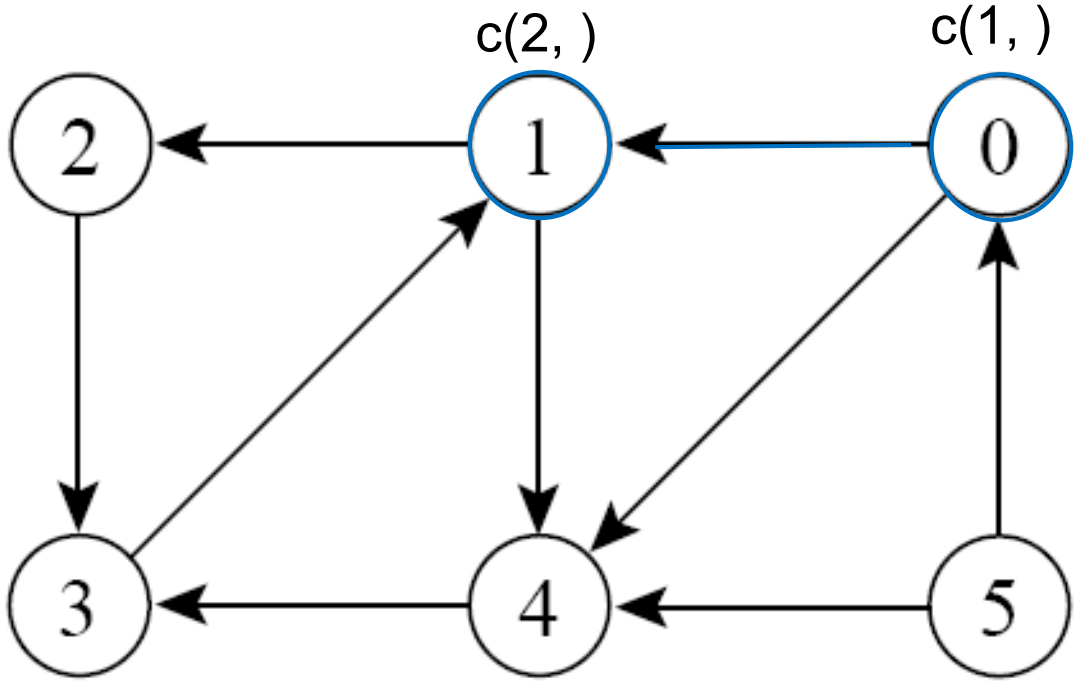
# Arestas

- Exemplo, partindo do vértice 0



# Arestas

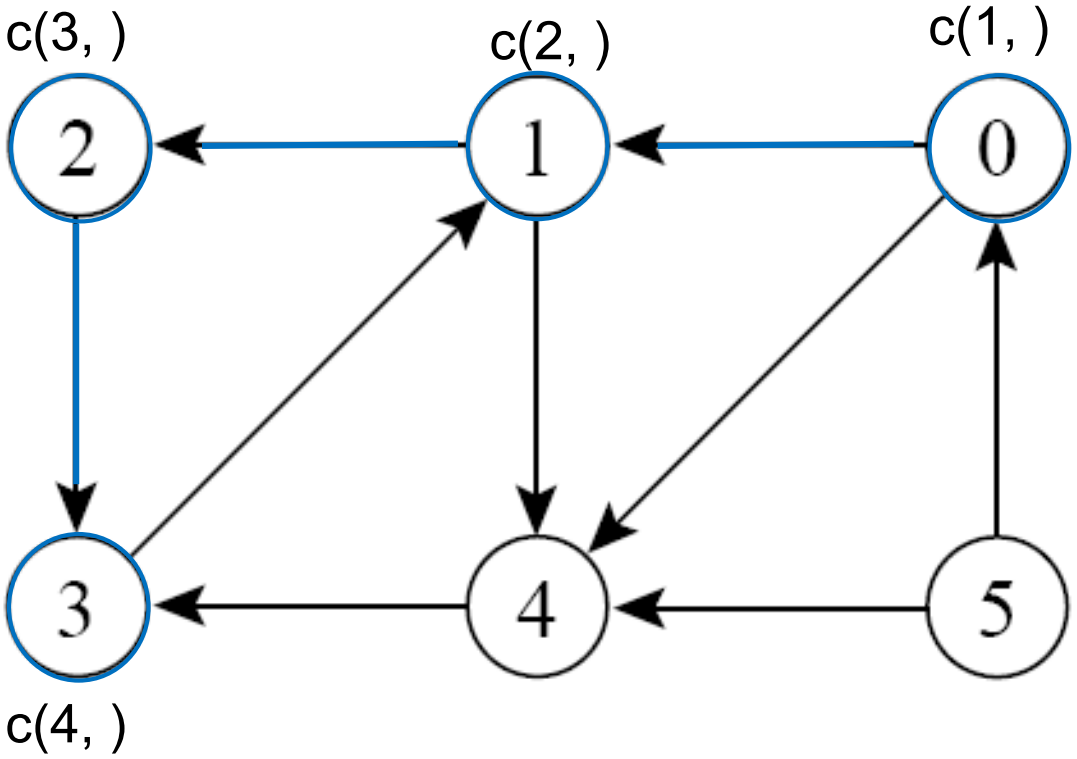
- Exemplo, partindo do vértice 0




aresta (0,1): árvore

# Arestas

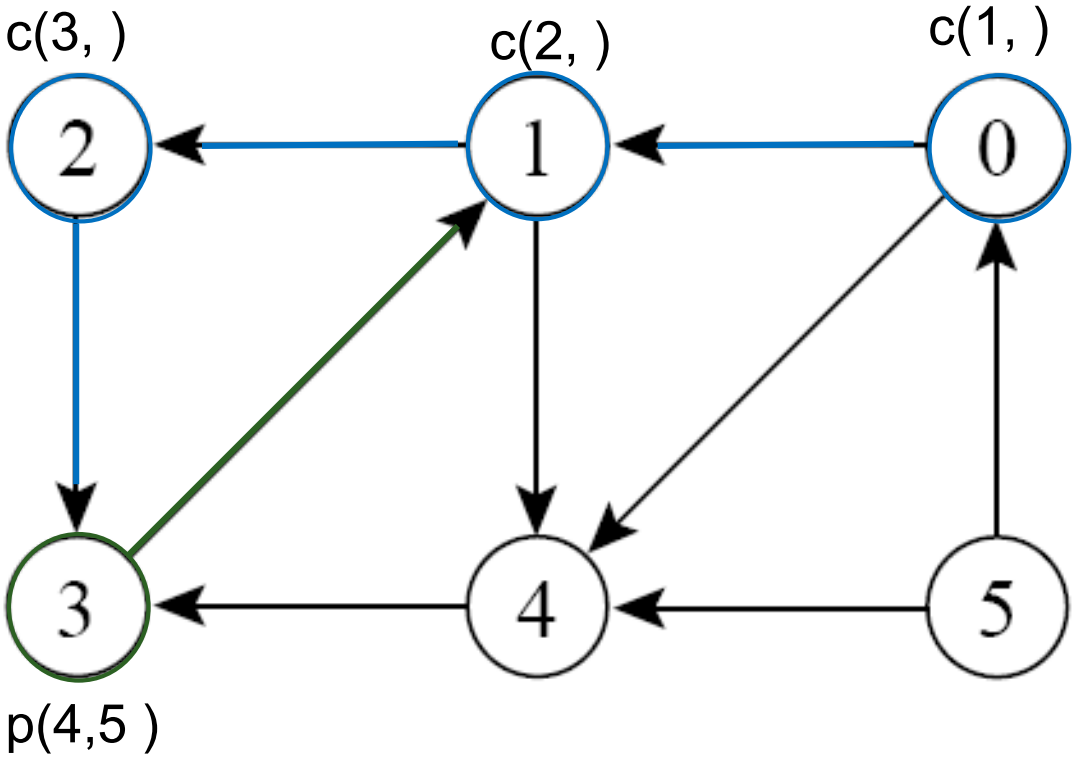
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore

# Arestas

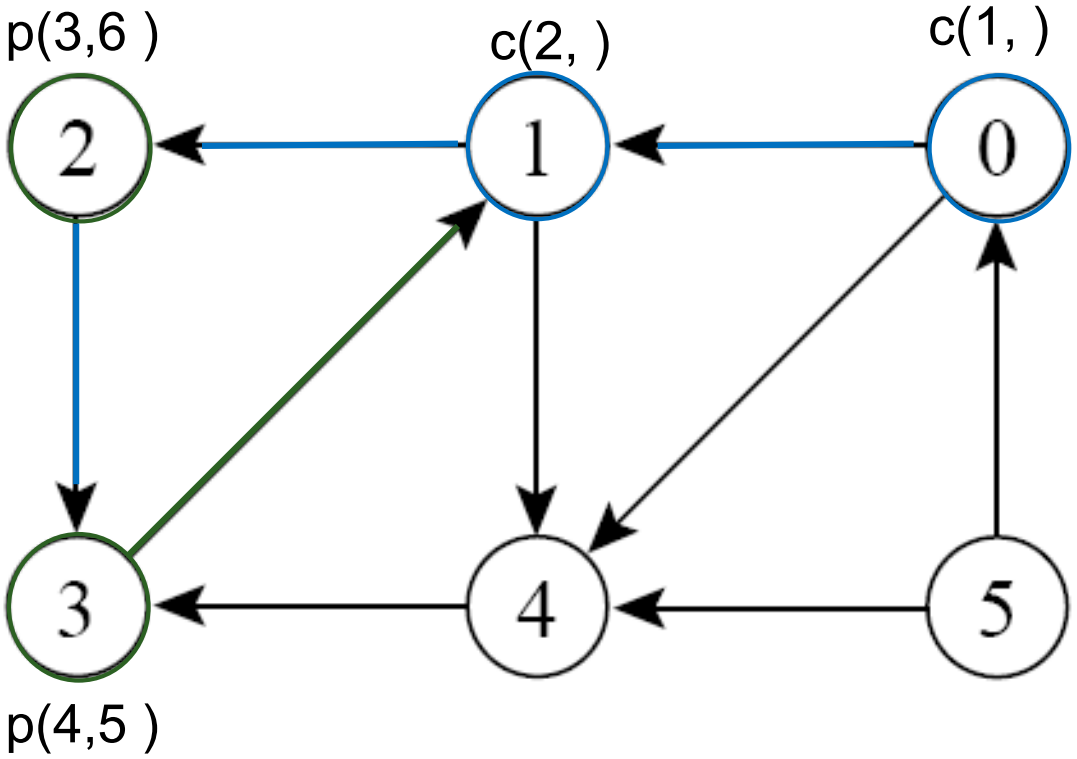
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno

# Arestas

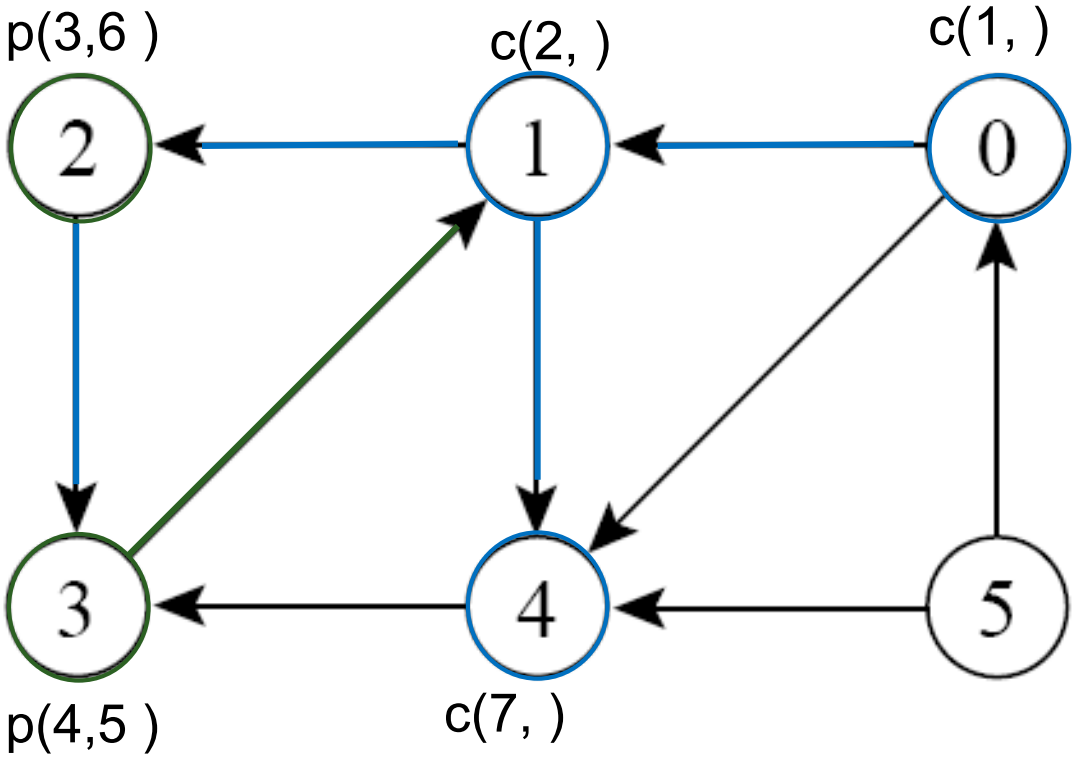
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno

# Arestas

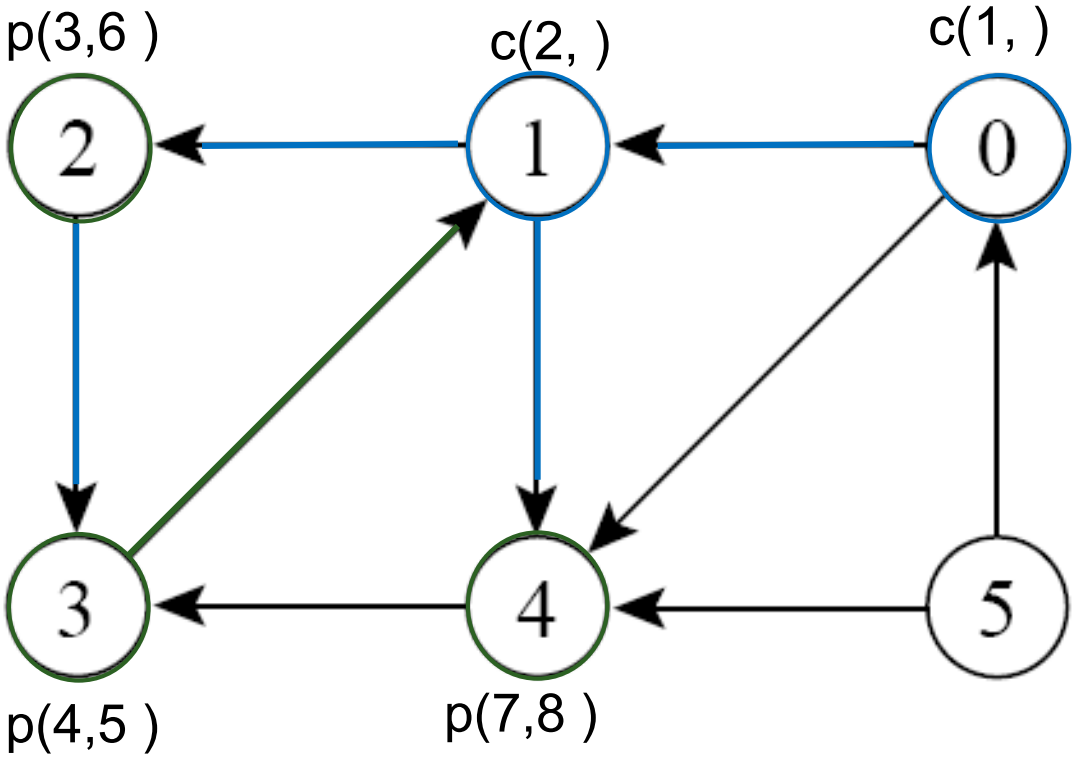
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno
aresta (1,4): árvore

# Arestas

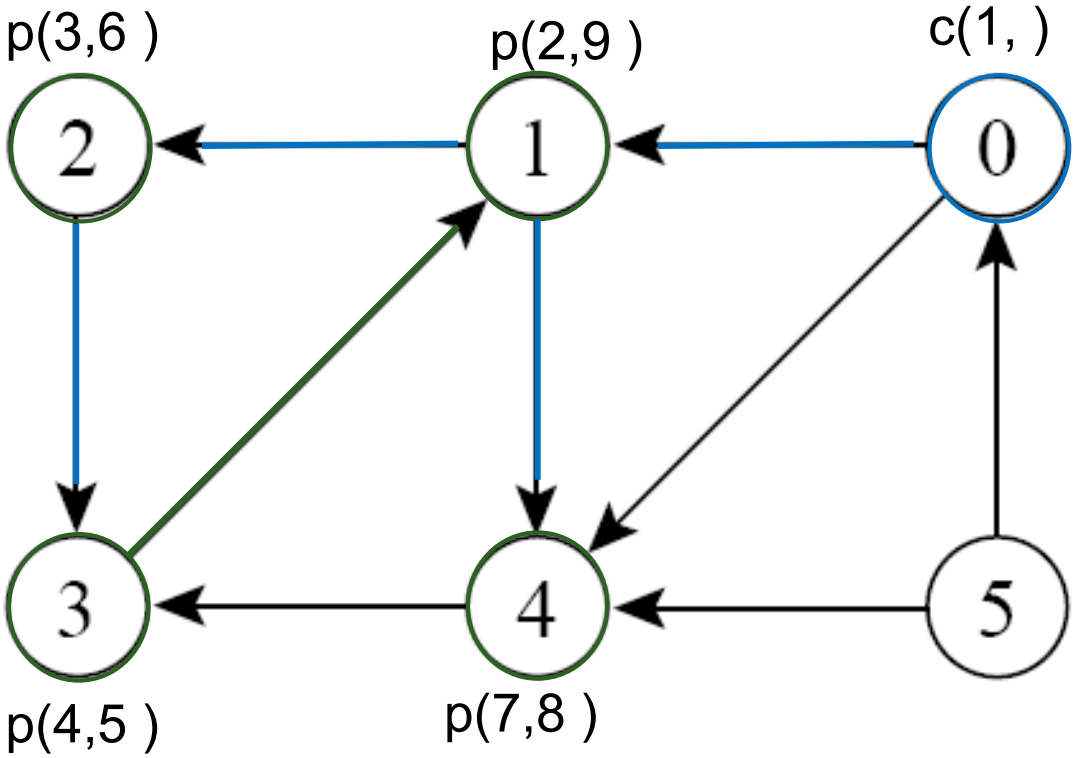
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno
aresta (1,4): árvore

# Arestas

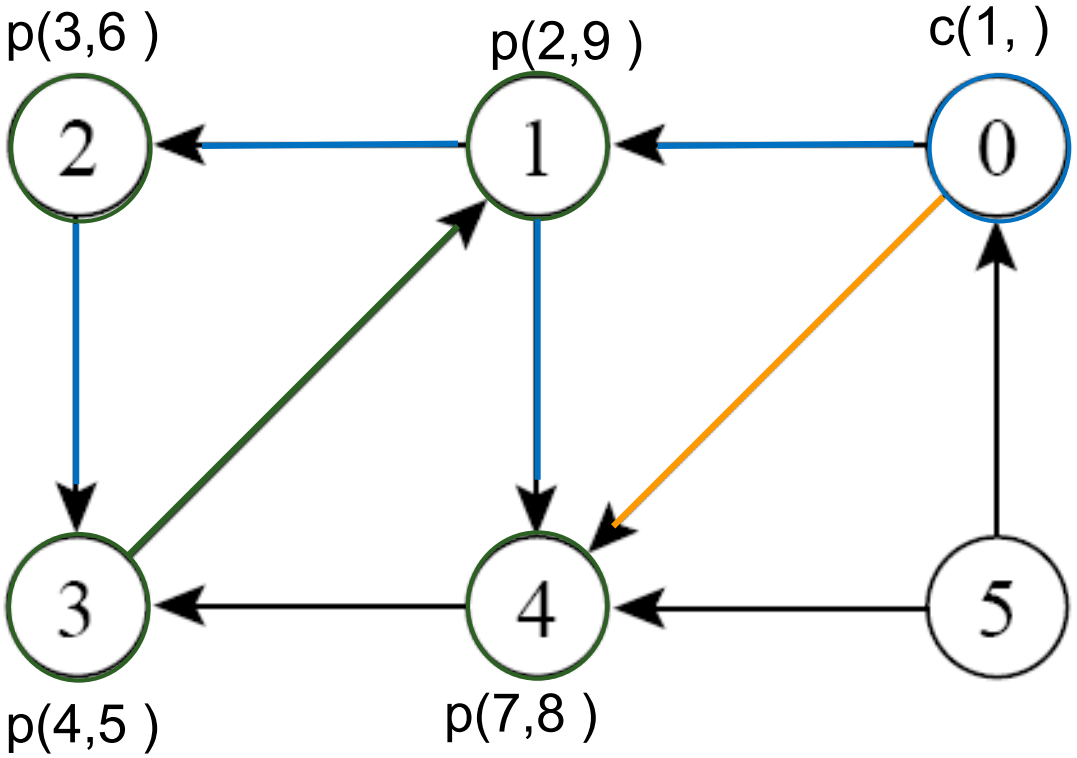
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno
aresta (1,4): árvore
aresta (4,3): cruzam

# Arestas

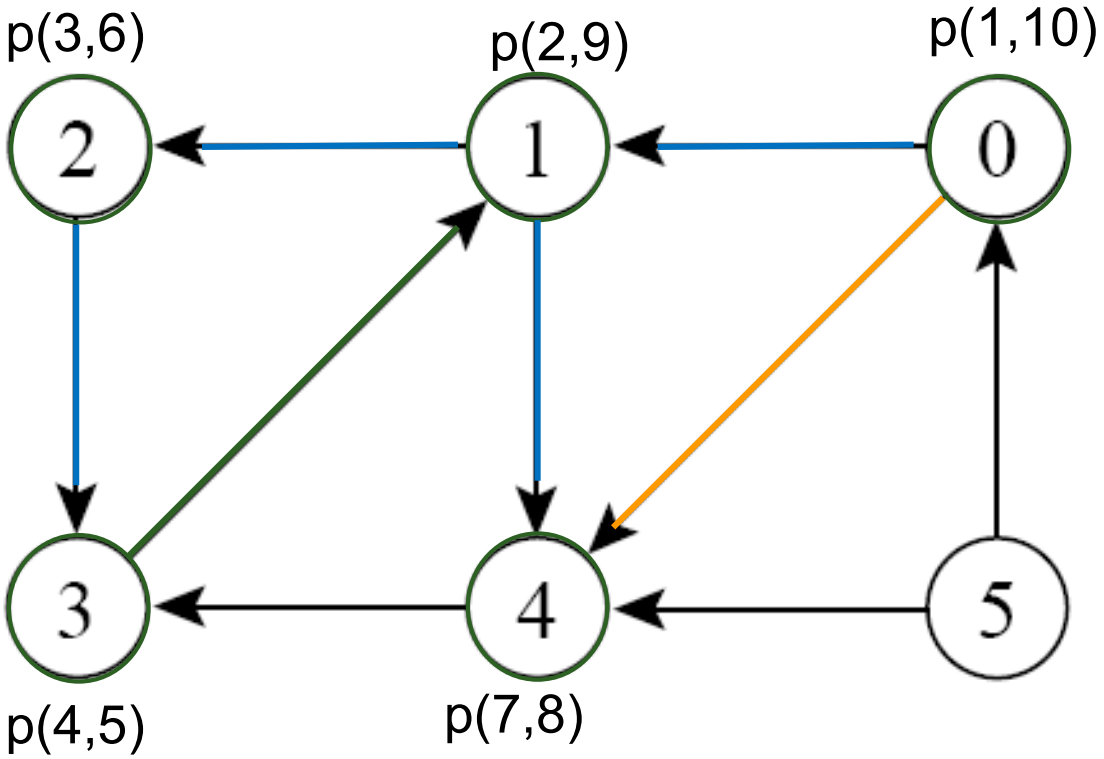
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno
aresta (1,4): árvore
aresta (4,3): cruzam
aresta (0,4): avanço

# Arestas

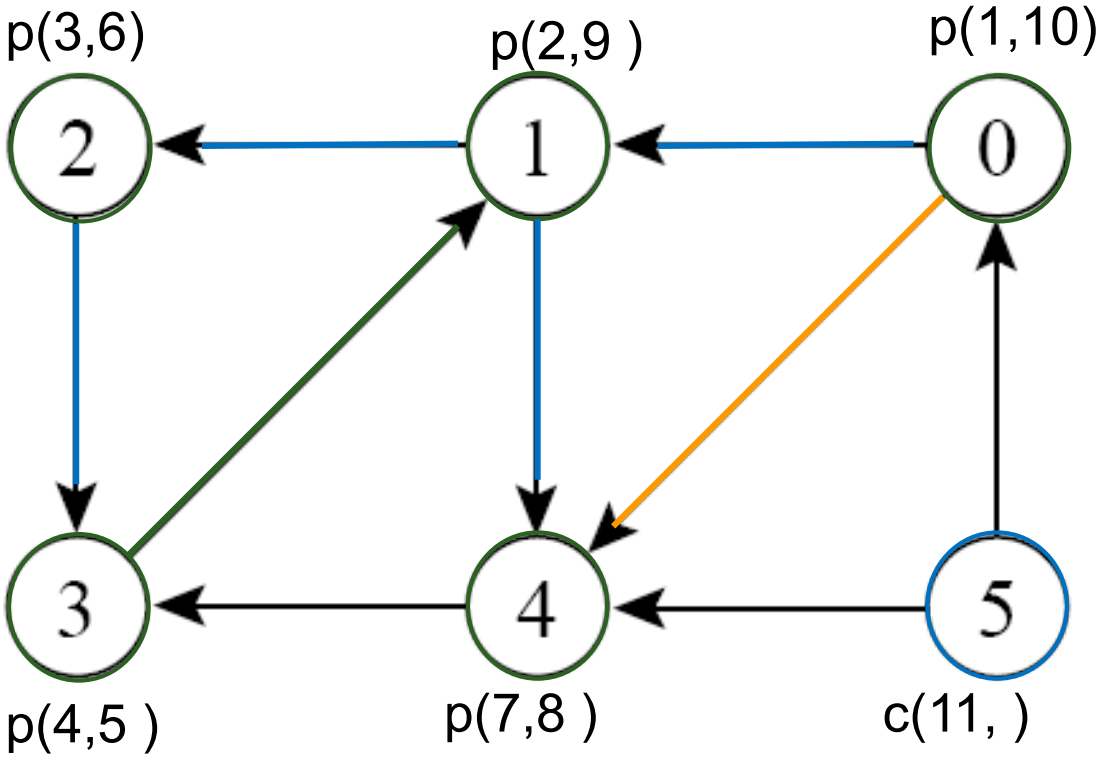
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno
aresta (1,4): árvore
aresta (4,3): cruzam
aresta (0,4): avanço

# Arestas

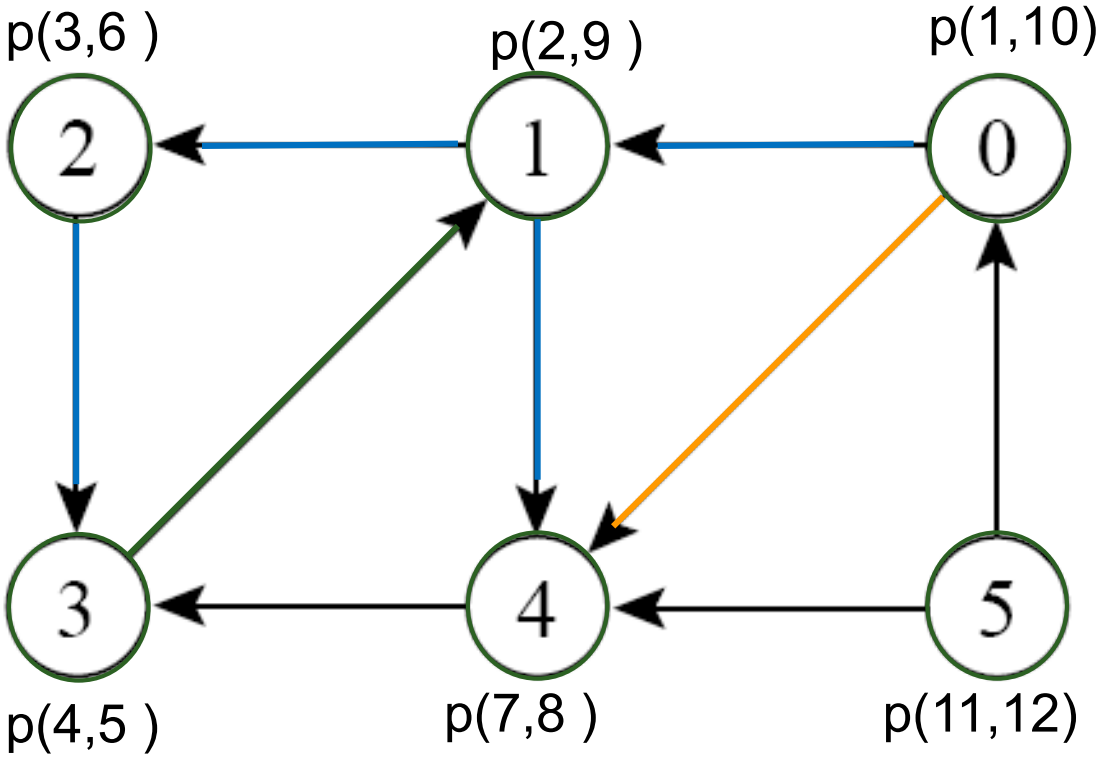
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno
aresta (1,4): árvore
aresta (0,4): avanço
aresta (4,3): cruzam
aresta (5,0): cruzam

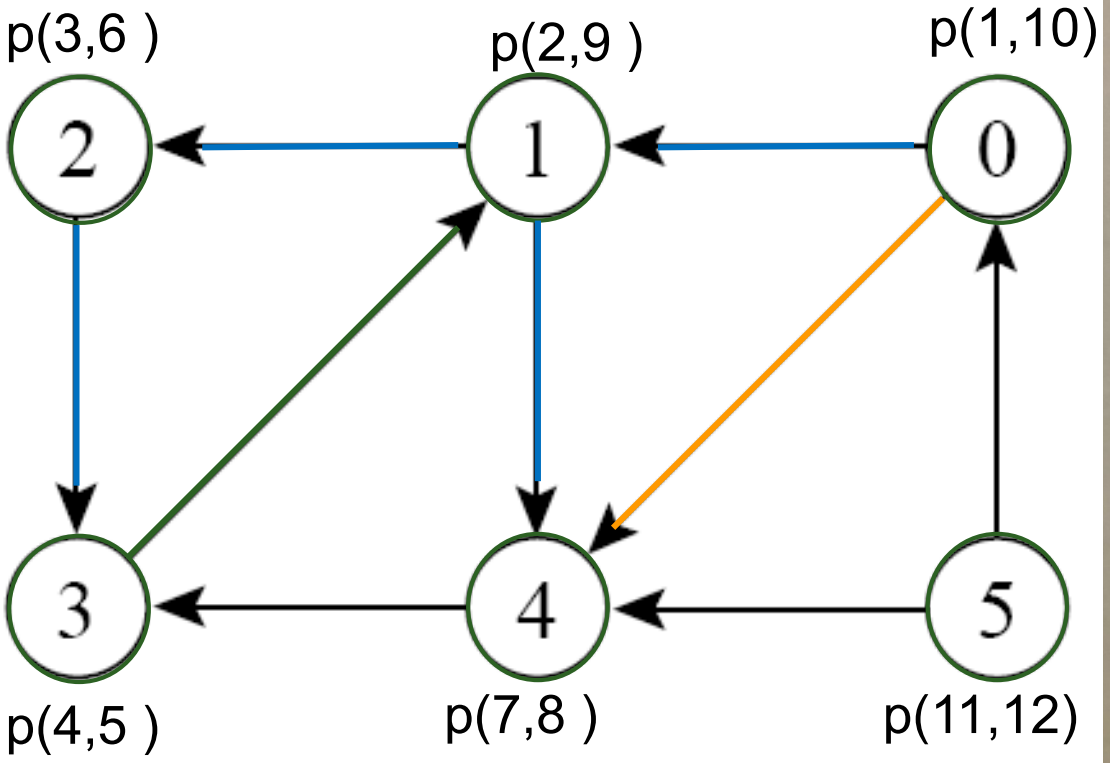
# Arestas

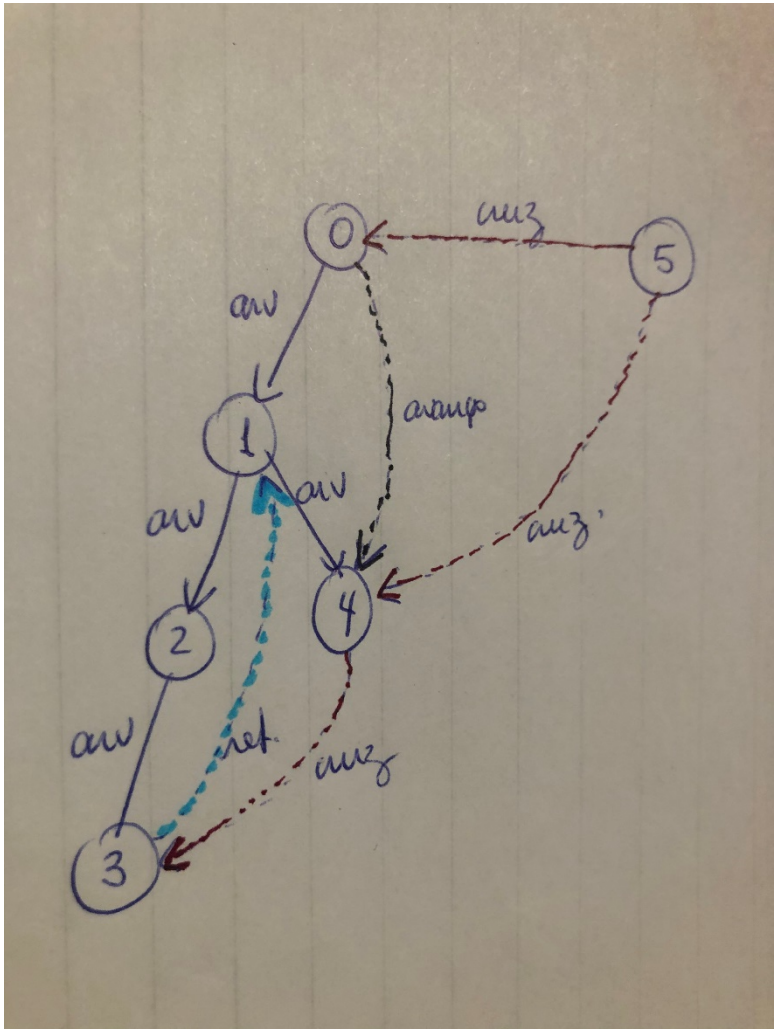
- Exemplo, partindo do vértice 0




aresta (0,1): árvore
aresta (1,2): árvore
aresta (2,3): árvore
aresta (3,1): retorno
aresta (1,4): árvore
aresta (0,4): avanço
aresta (4,3): cruzam
aresta (5,0): cruzam
aresta (5,4): cruzam

# Arestas e árvores DFS



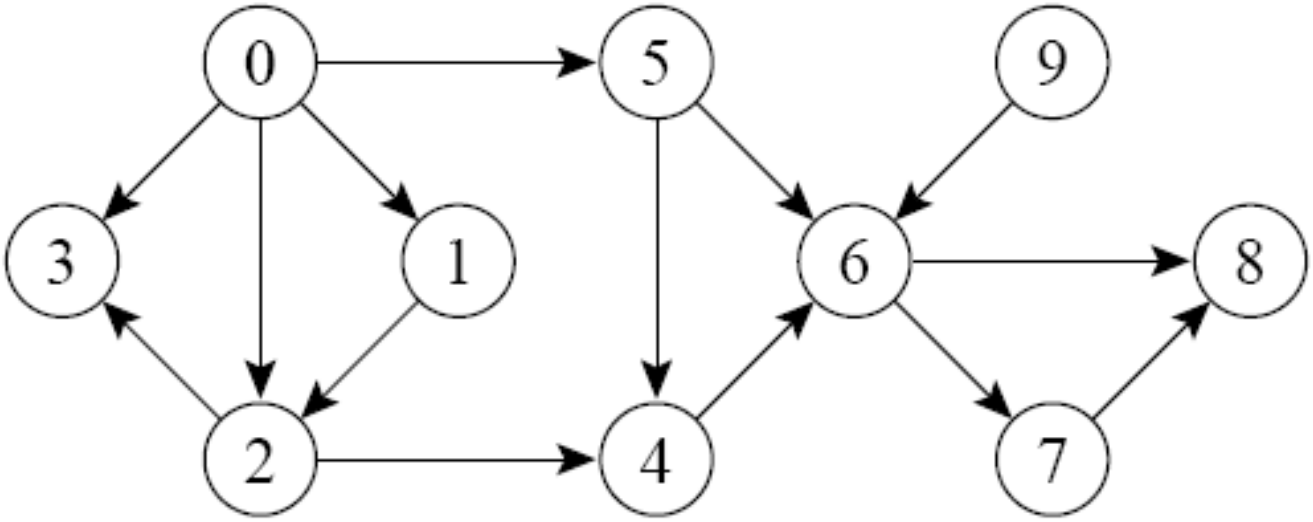



# Arestas

- Exercício: simule a busca DFS como fizemos no grafo anterior, mas partindo do vértice 5. Classifique as arestas e desenhe a árvore DFS resultante.

# Arestas

- Exercício: execute a busca DFS e classifique as arestas



# Grafo cíclico

- Se existe uma aresta de retorno, então o grafo é cíclico

- No algoritmo de busca em profundidade
  - Se encontrar um vértice *v* adjacente cinza, então aresta de retorno foi encontrada, e portanto há ciclo!

# Desafio!

# Ordenação topológica

- Ordenação topológica de um grafo direcionado acíclico
  - Ordenação linear dos vértices do grafo tal que um vértice *u* precede um vértice *v* se existe uma aresta (*u*,*v*)
    - Qual a utilidade da ordenação topológica? Exemplos?

# Ordenação topológica

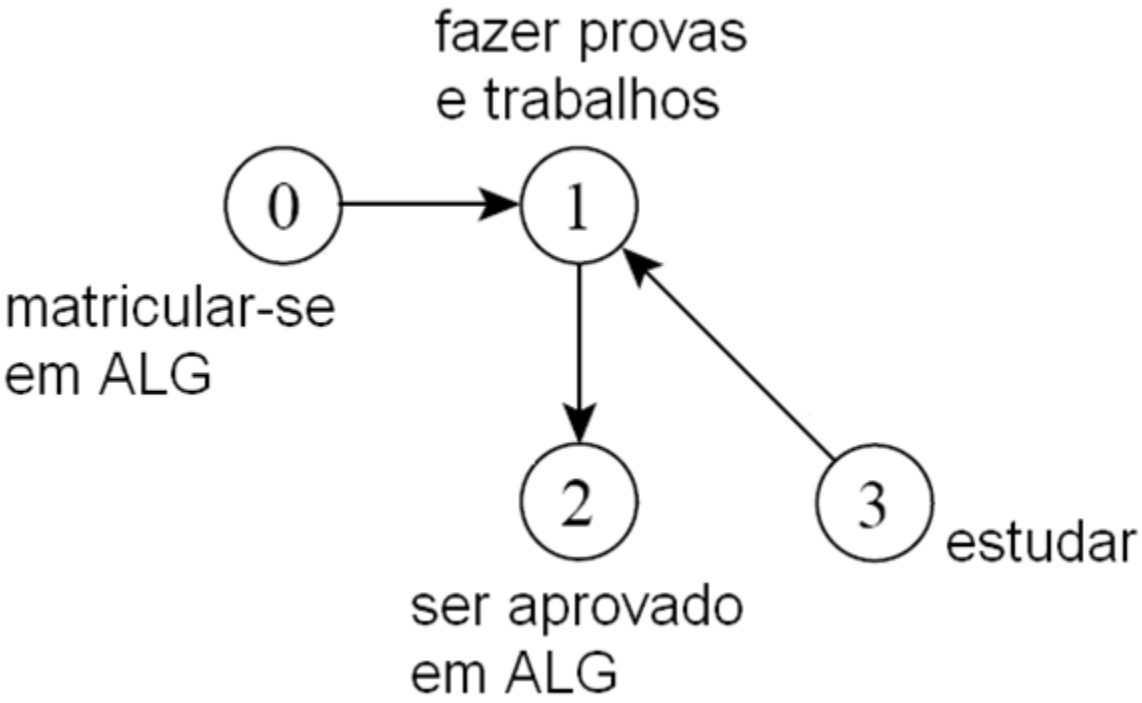
- Ordenação topológica de um grafo direcionado acíclico
  - Ordenação linear dos vértices do grafo tal que um vértice *u* precede um vértice *v* se existe uma aresta (*u*,*v*)
    - Qual a utilidade da ordenação topológica? Exemplos?

# Ordenação topológica

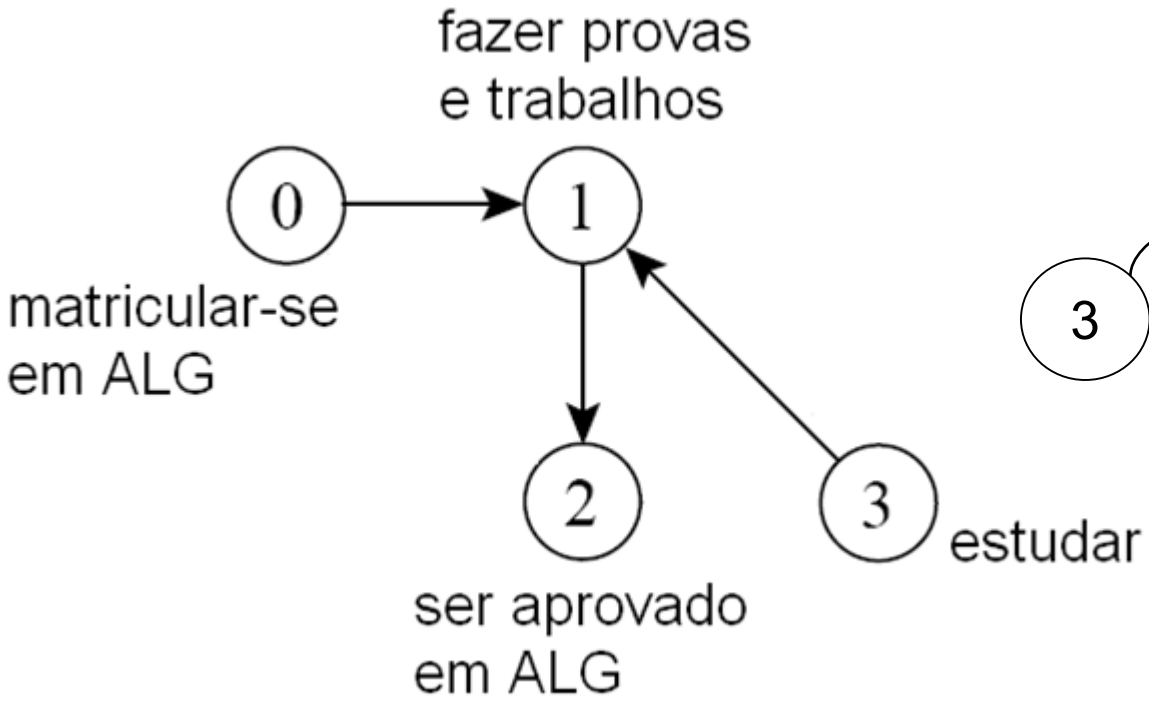
- Ordenação topológica de um grafo direcionado acíclico
  - Ordenação linear dos vértices do grafo tal que um vértice *u* precede um vértice *v* se há uma aresta (*u*,*v*)

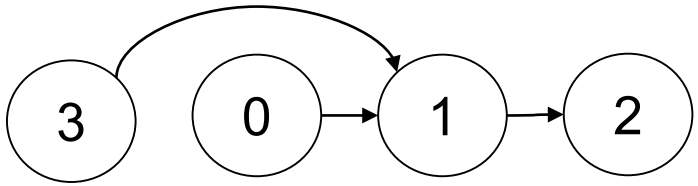
- Útil para programar a execução de uma sequencia de muitas tarefas em que algumas dependem de outras...
  - p. ex., ao construir um edifício, ao determinar a sequencia de disciplinas de um curso
    - certas tarefas/disciplinas podem ser executadas/cursadas simultaneamente, outras não...

# Exemplo

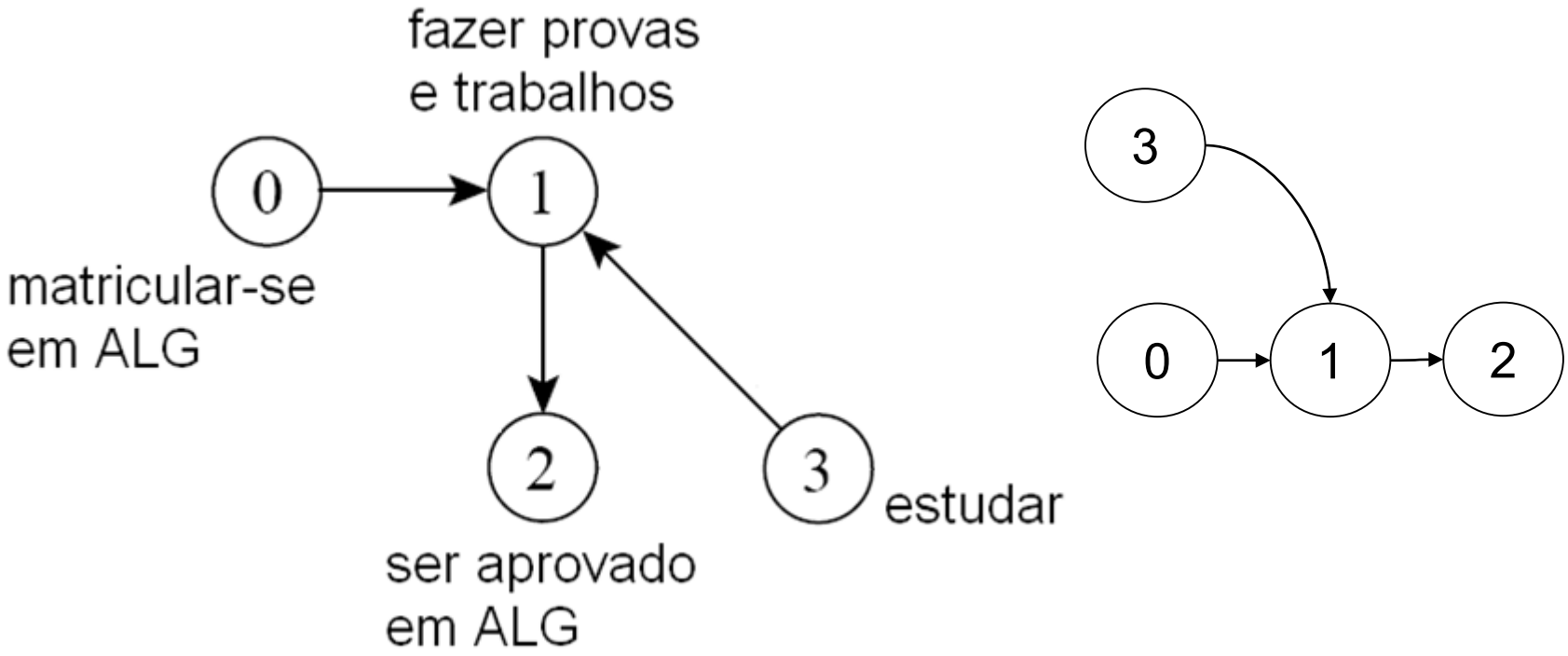

fazer provas
e trabalhos
0
1
matricular-se
em ALG
2
3
estudar
ser aprovado
em ALG

# Exemplo


fazer provas
e trabalhos
0
1
matricular-se
em ALG
2
3
estudar
ser aprovado
em ALG


3
0
1
2

# Exemplo


fazer provas
e trabalhos
0
1
matricular-se
em ALG
2
3
estudar
ser aprovado
em ALG
3
0
1
2

# Exercício

- Implementar uma função para fazer ordenação topológica!

- Grafo de dependências como entrada
  - Direcionado!

# Ordenação topológica

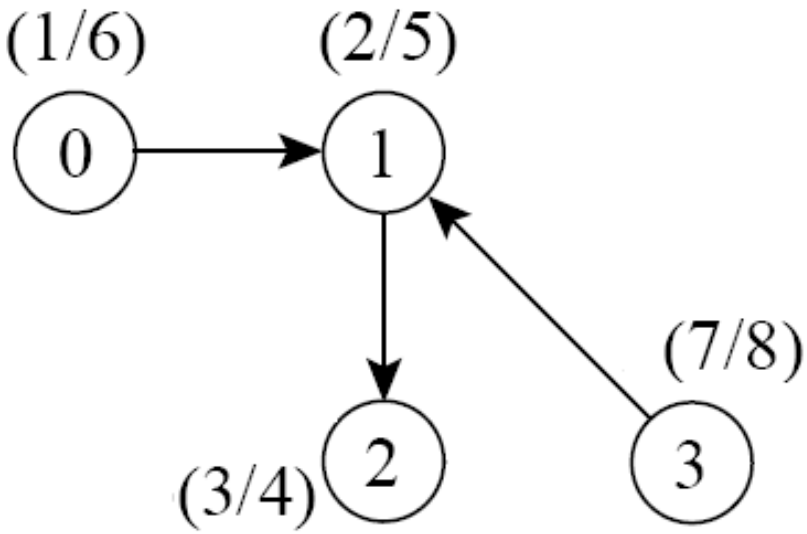
- Algoritmo

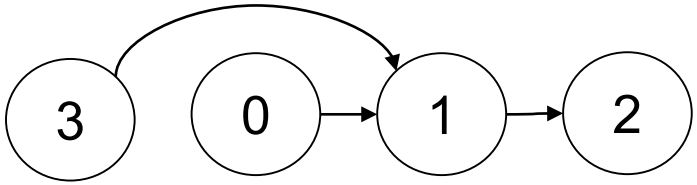
1. Faça a busca em profundidade no grafo
   - Ao término do processamento de cada vértice, insira o vértice no início de uma lista linear (inicialmente vazia)

2. Ao percorrer a lista do começo ao final, tem-se a ordenação topológica do grafo!

# Ordenação topológica

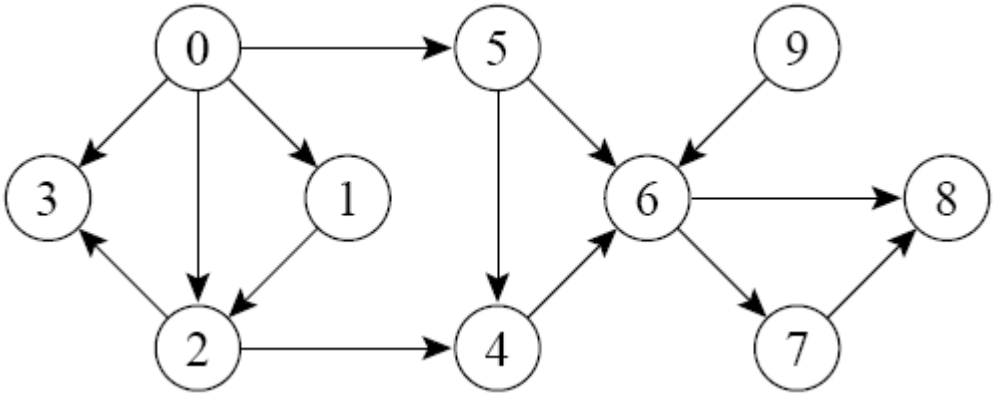
- Ordenação topológica: exemplo



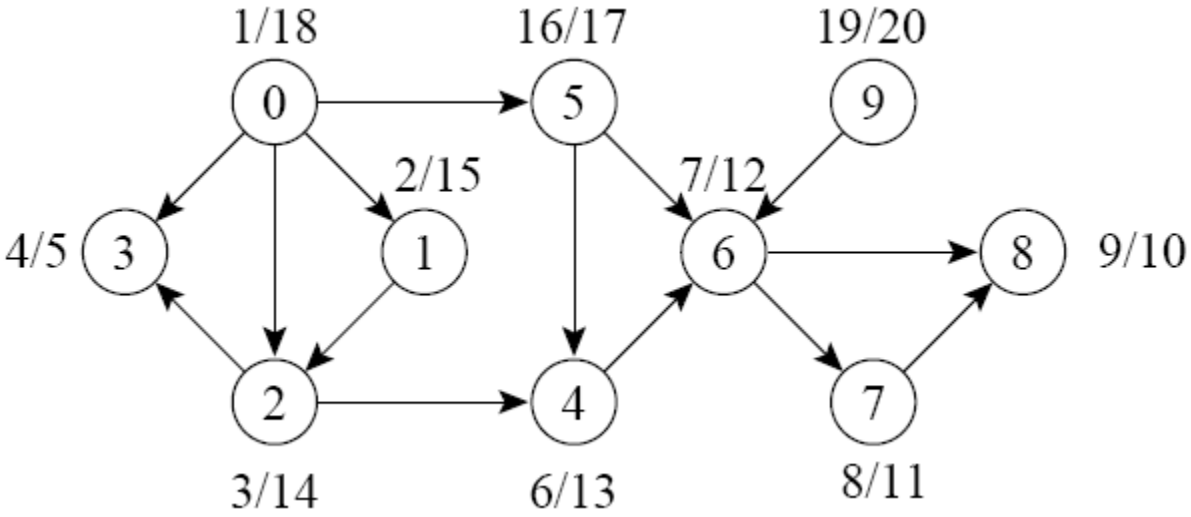
# Ordenação topológica

- Faça a ordenação topológica do grafo abaixo



# Ordenação topológica

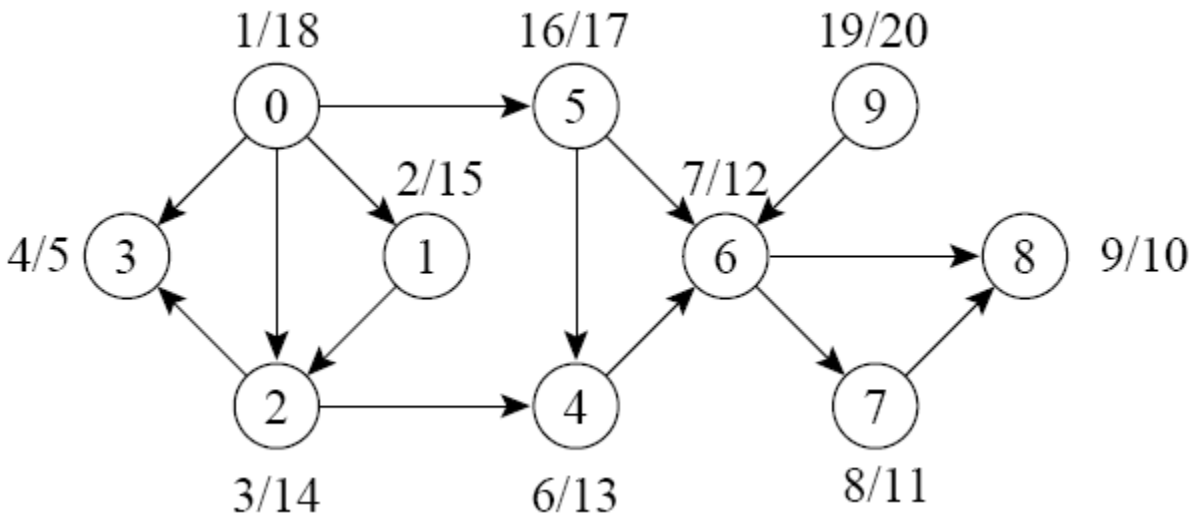
- Faça a ordenação topológica do grafo abaixo

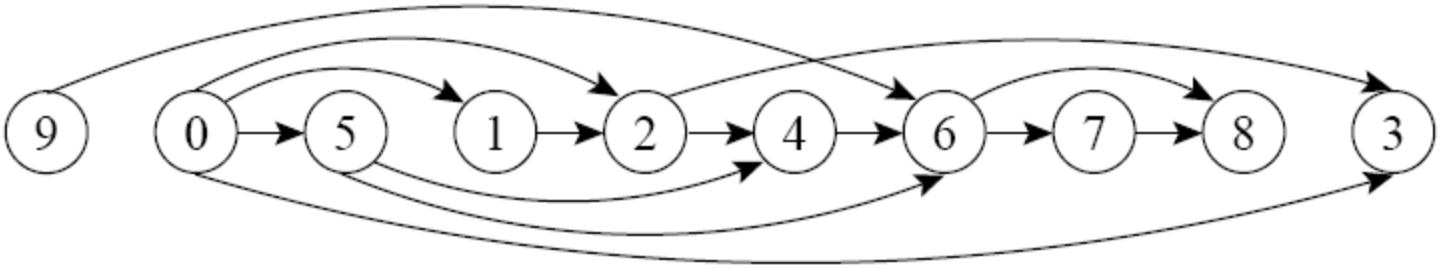



Visita DFS: 3 8 7 6 4 2 1 5 0 9

# Ordenação topológica

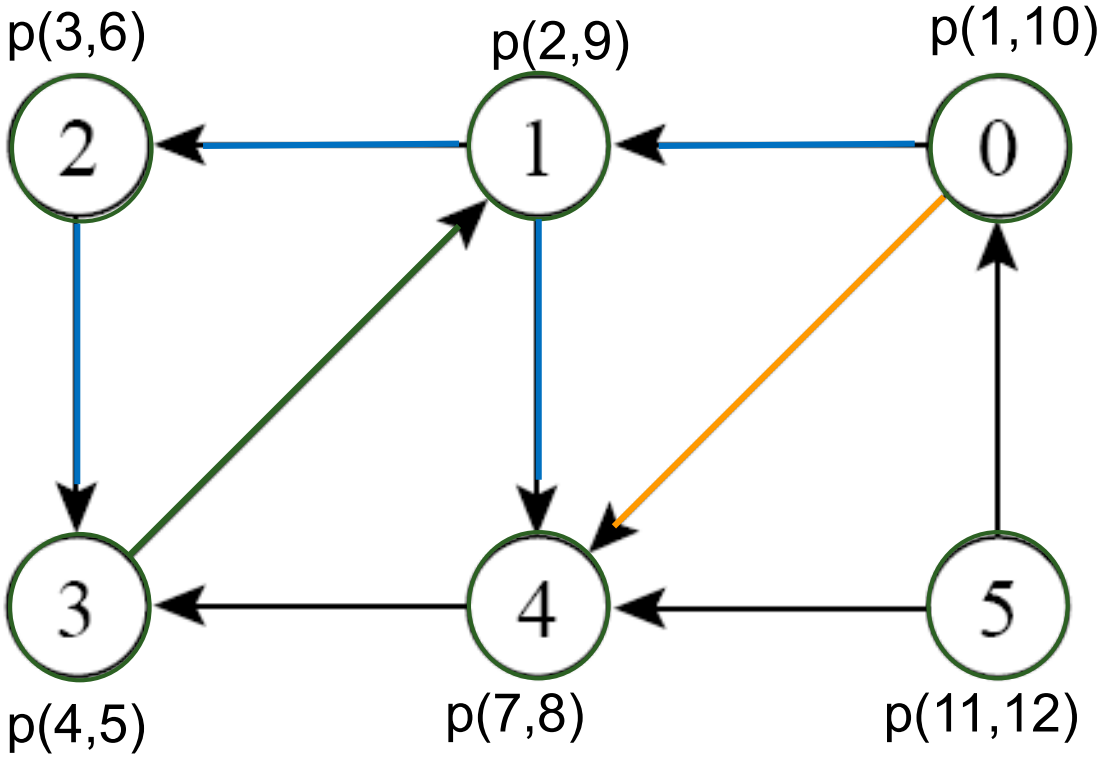
- Faça a ordenação topológica do grafo abaixo



# Ordenação topológica

- Pergunta: é possível gerar uma ordenação topológica deste grafo?




Visita DFS: 3 2 4 1 0 5

# Ordenação topológica

- Ordenação topológica de um grafo direcionado **acíclico**
  - Como alterar o algoritmo de busca em profundidade para realizar a ordenação topológica?
  - Qual a complexidade de tempo do algoritmo?

# Ordenação topológica

- Ordenação topológica de um grafo direcionado **acíclico**
  - Como alterar o algoritmo de busca em profundidade para realizar a ordenação topológica?
    - Após o vértice ficar preto, insere-se o vértice no início da lista
  - Qual a complexidade de tempo do algoritmo?
    - O(|V|+|A|)

# Ordenação topológica

- Atenção
  - Ordenação topológica não é necessariamente única
  - **Não é possível** gerar uma ordenação topológica em grafos com ciclos