

---

# Grafos: Busca em Profundidade

---

SCC0216/503 Modelagem Computacional em  
Grafos/Algoritmos e Estruturas de Dados 2

Thiago A. S. Pardo  
Maria Cristina F. Oliveira

---

# Percorrendo um grafo

- Há duas possibilidades
  - Busca em largura (usando uma fila)
  - **Busca em profundidade (usando uma pilha)**

# Percorrendo um grafo

## DFS - *Depth-First Search*

- ❑ Explora-se cada vértice do grafo em profundidade, a medida em que é visitado...
  - Procura-se **avançar na busca** sem olhar para todos os vértices vizinhos 'no mesmo nível'
  - Quando necessário, volta-se aos vizinhos ainda não totalmente processados antes (processo de **backtracking**)

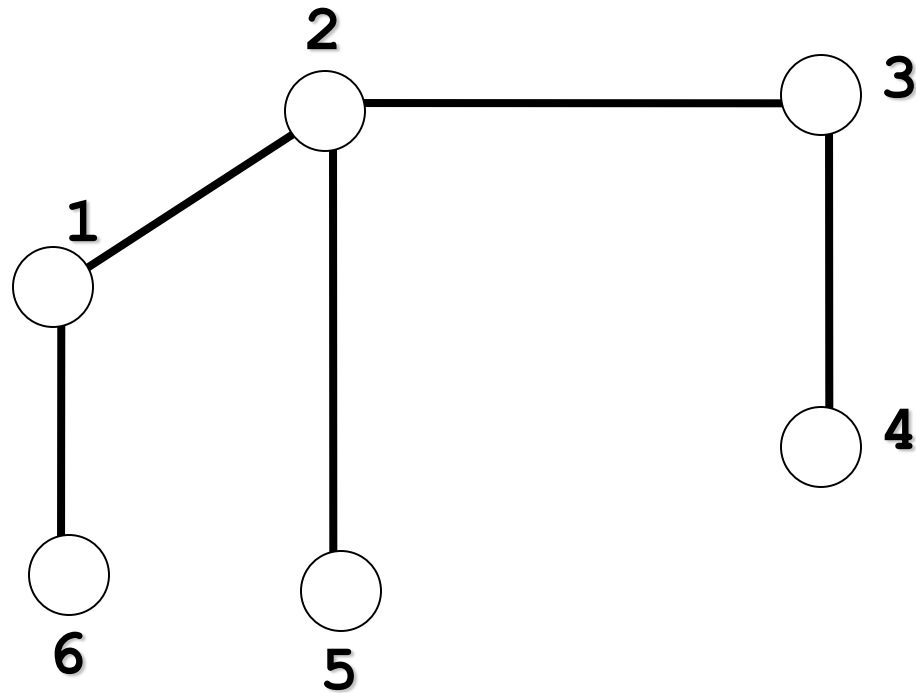
# Percorrendo um grafo

## DFS - *Depth-First Search*

- Pode-se adotar o mesmo esquema de rotulação por cores para guiar a busca
  - Todos os vértices são inicializados **brancos**
  - Quando um vértice  $v$  é ‘descoberto’ pela primeira vez, ele se torna **cinza**
  - Um vértice  $v$  torna-se **preto** quando todos os vértices adjacentes a ele tiverem sido ‘descobertos’

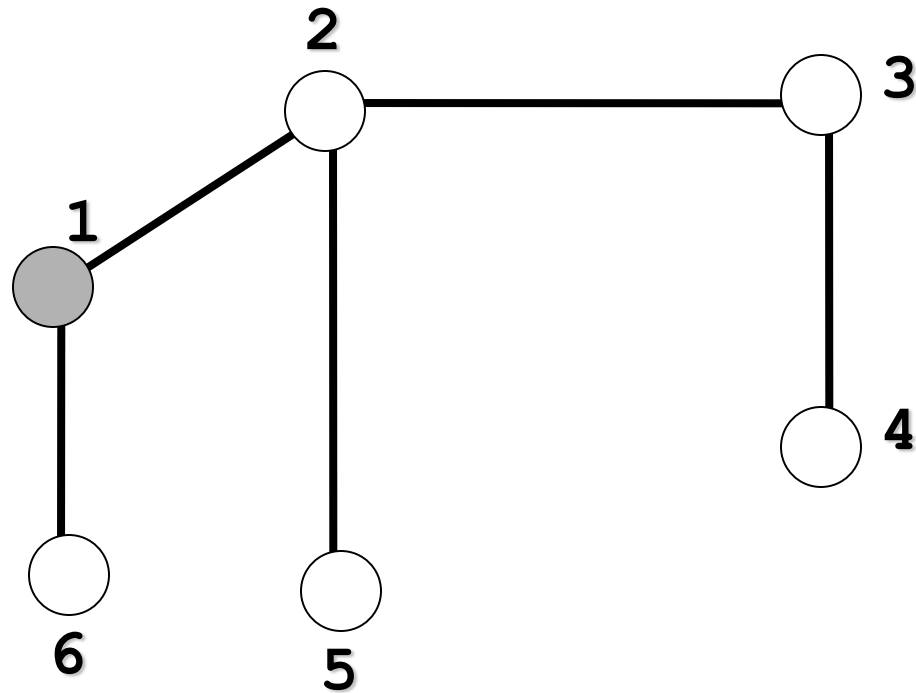
# DFS – exemplo

Percorrendo um Grafo: DFS



# DFS – exemplo

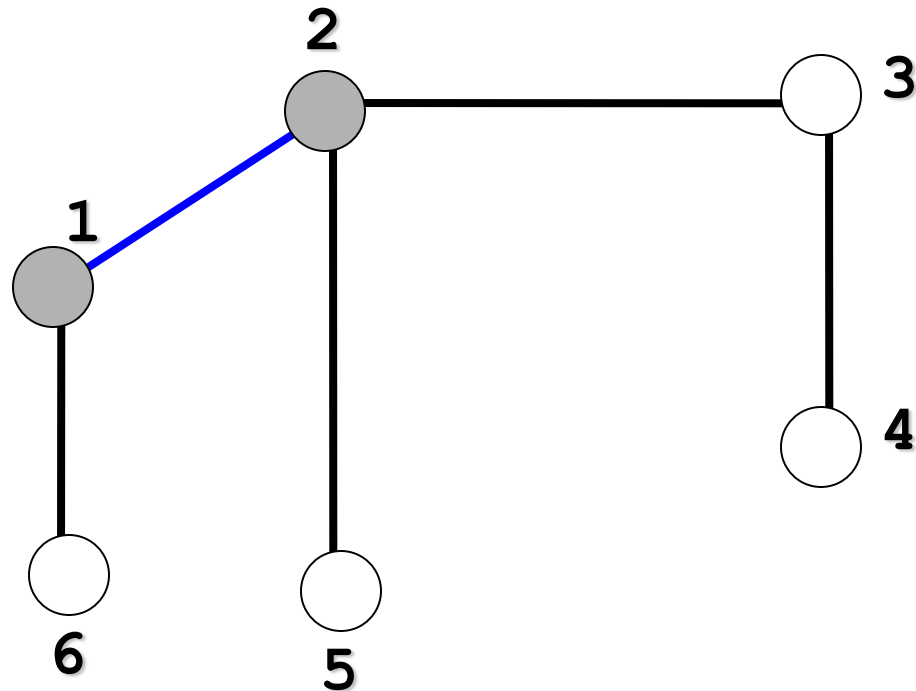
Percorrendo um Grafo: DFS



Vértice inicial: 1

# DFS – exemplo

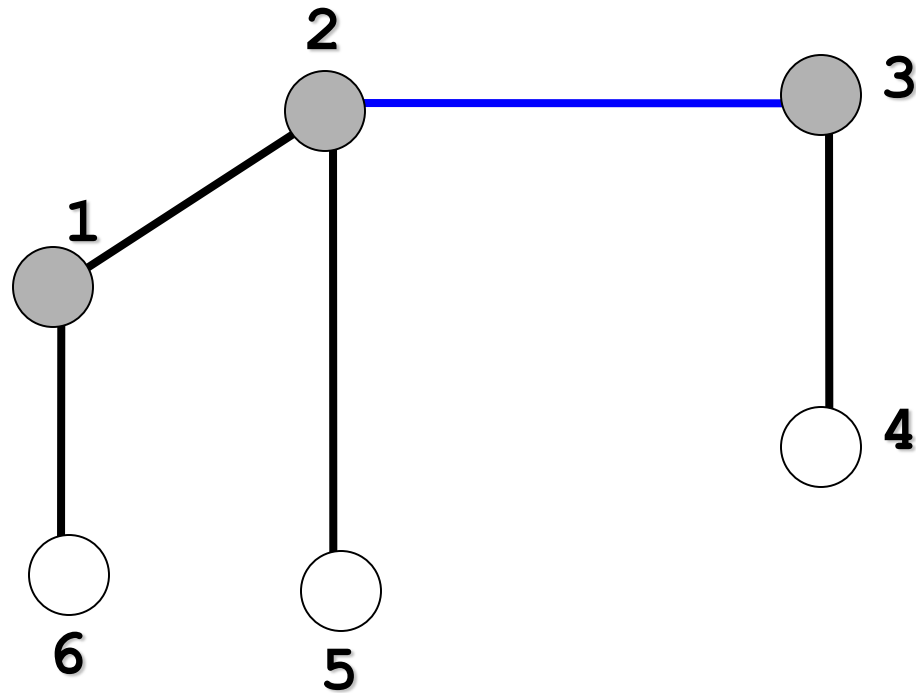
Percorrendo um Grafo: DFS



Busca-se pelo primeiro vértice adjacente a 1: 2

# DFS – exemplo

Percorrendo um Grafo: DFS

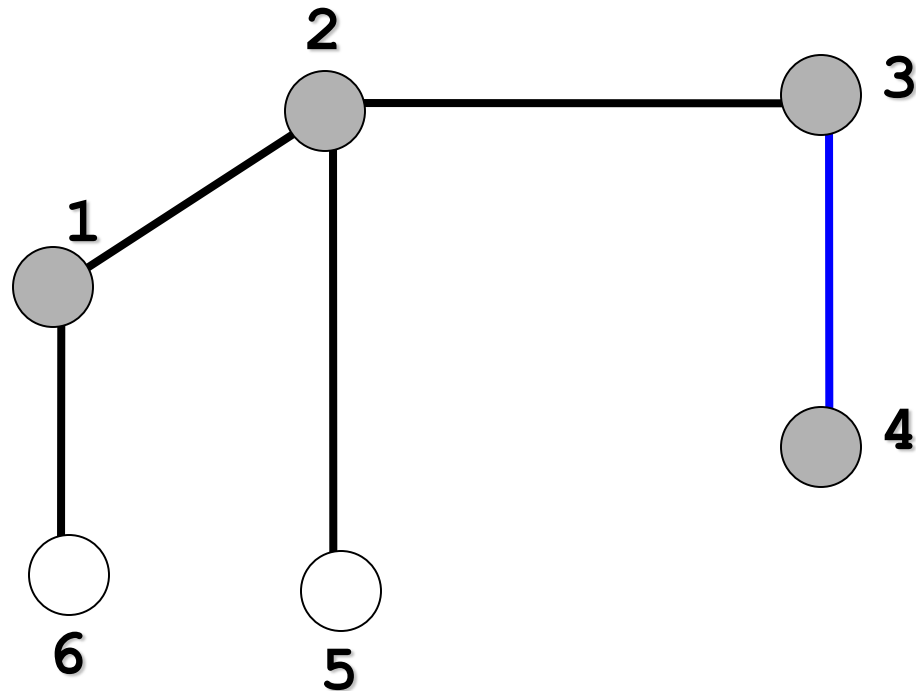


Busca-se pelo primeiro vértice adjacente a 2: 3



# DFS – exemplo

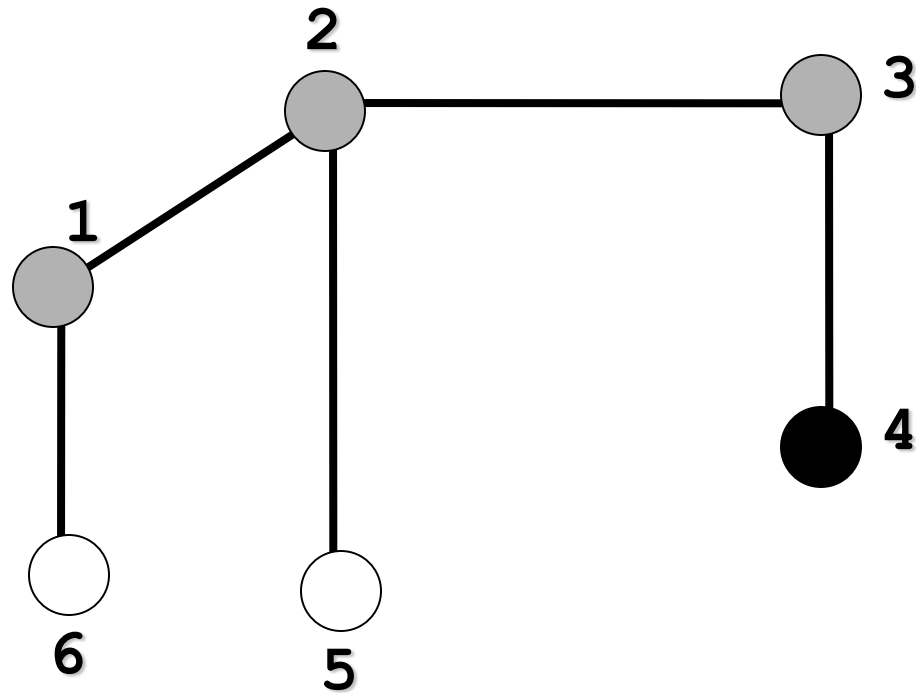
Percorrendo um Grafo: DFS



Busca-se pelo primeiro vértice adjacente a 3: 4

# DFS – exemplo

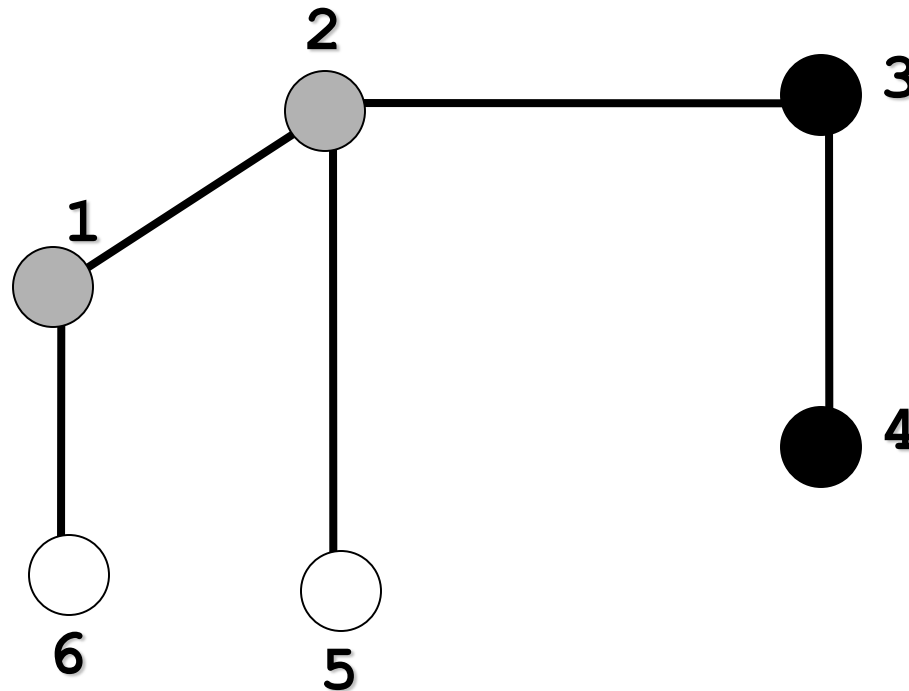
Percorrendo um Grafo: DFS



Vértice 4 não tem mais vértices adjacentes! Retorna-se ao anterior.

# DFS – exemplo

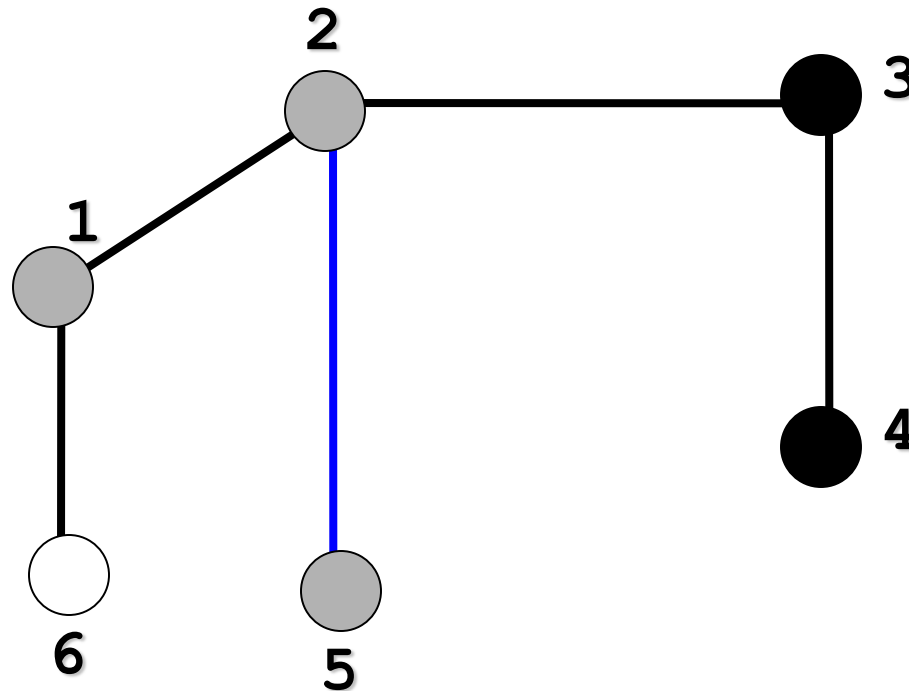
Percorrendo um Grafo: DFS



Vértice 3 não tem mais vértices adjacentes! Retorna-se ao anterior: 2

# DFS – exemplo

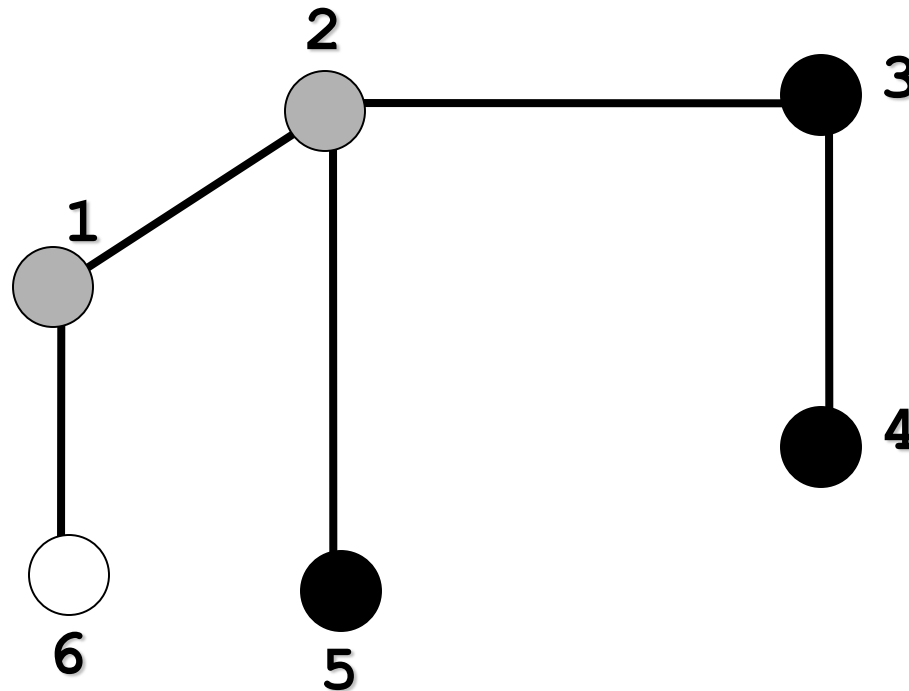
Percorrendo um Grafo: DFS



Busca-se pelo outro vértice adjacente a 2: 5

# DFS – exemplo

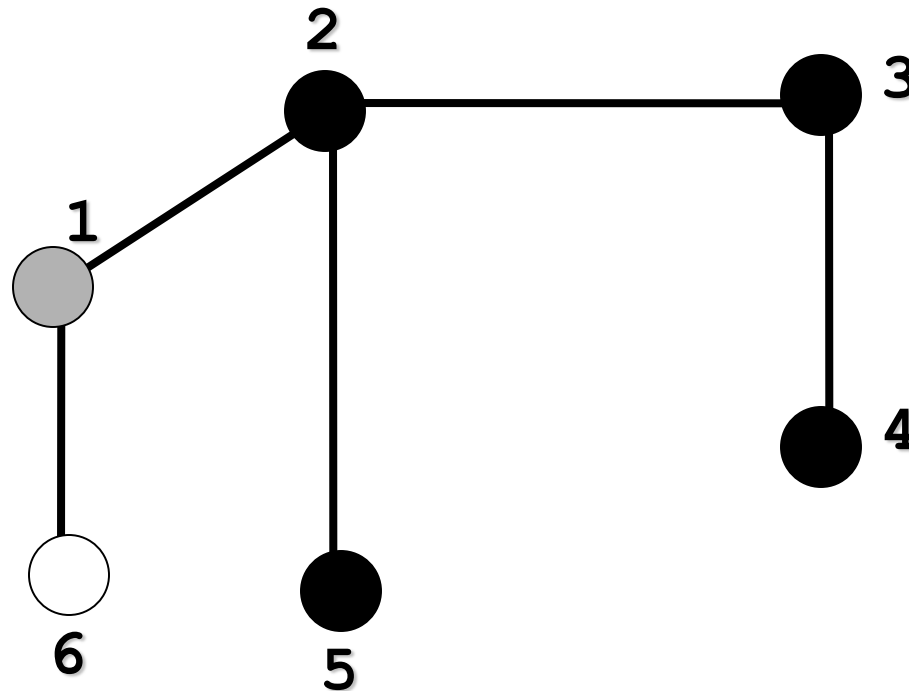
Percorrendo um Grafo: DFS



Vértice 5 não tem mais nós adjacentes! Retorna-se ao anterior: 2

# DFS – exemplo

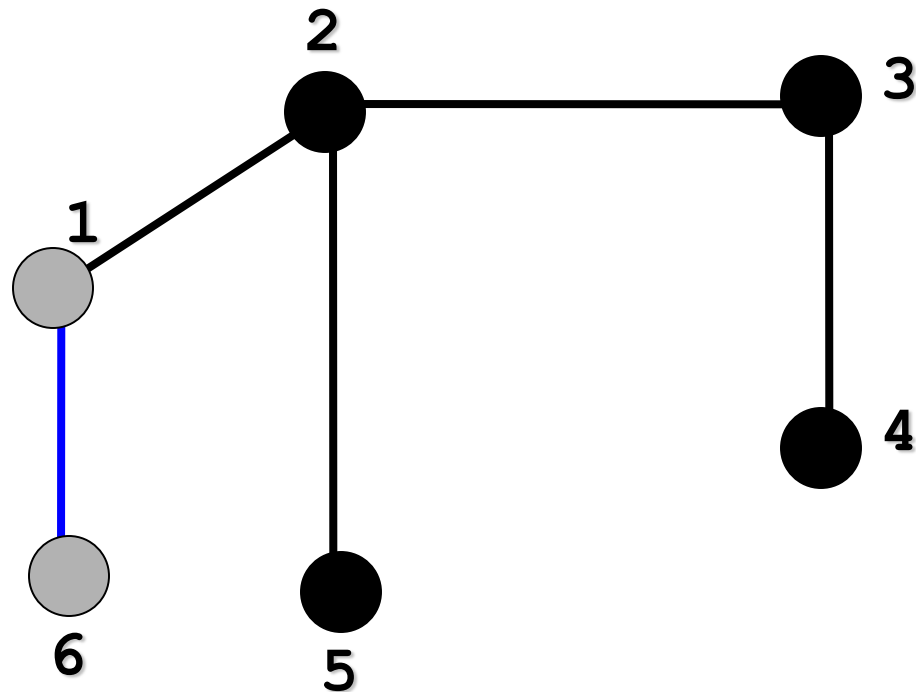
Percorrendo um Grafo: DFS



Vérrice 2 não tem mais vértices adjacentes! Retorna-se ao anterior: 1

# DFS – exemplo

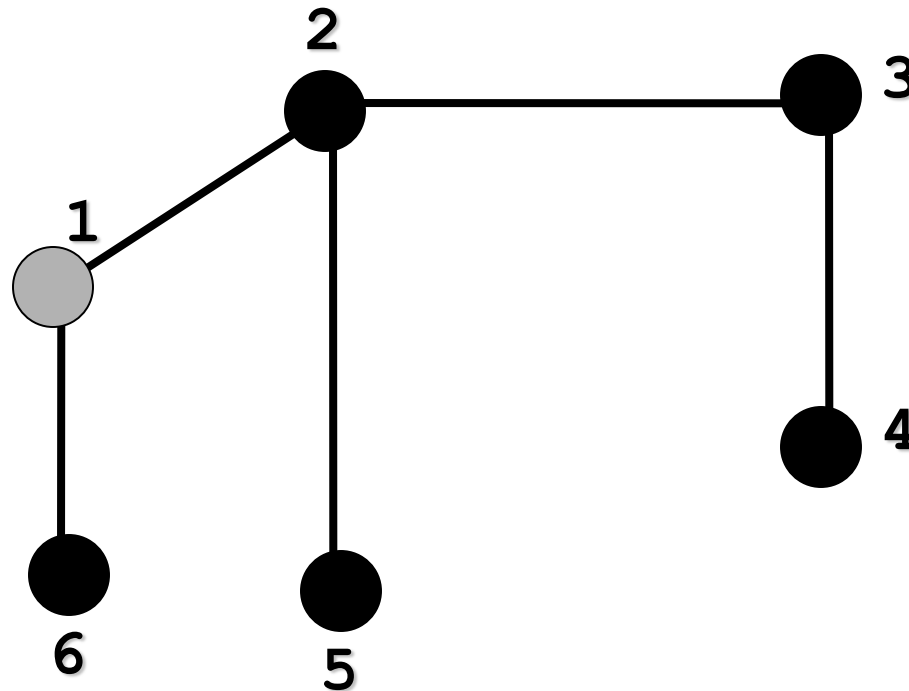
Percorrendo um Grafo: DFS



Busca-se pelo outro vértice adjacente a 1: 6

# DFS – exemplo

Percorrendo um Grafo: DFS

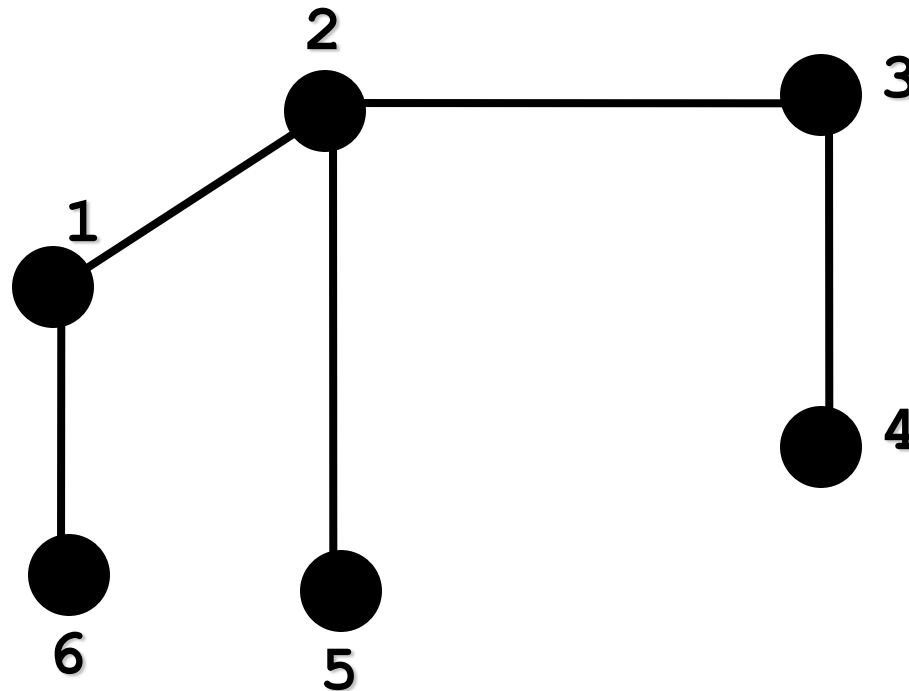


Vértice 6 não tem mais vértices adjacentes! Retorna-se ao anterior: 1



# DFS – exemplo

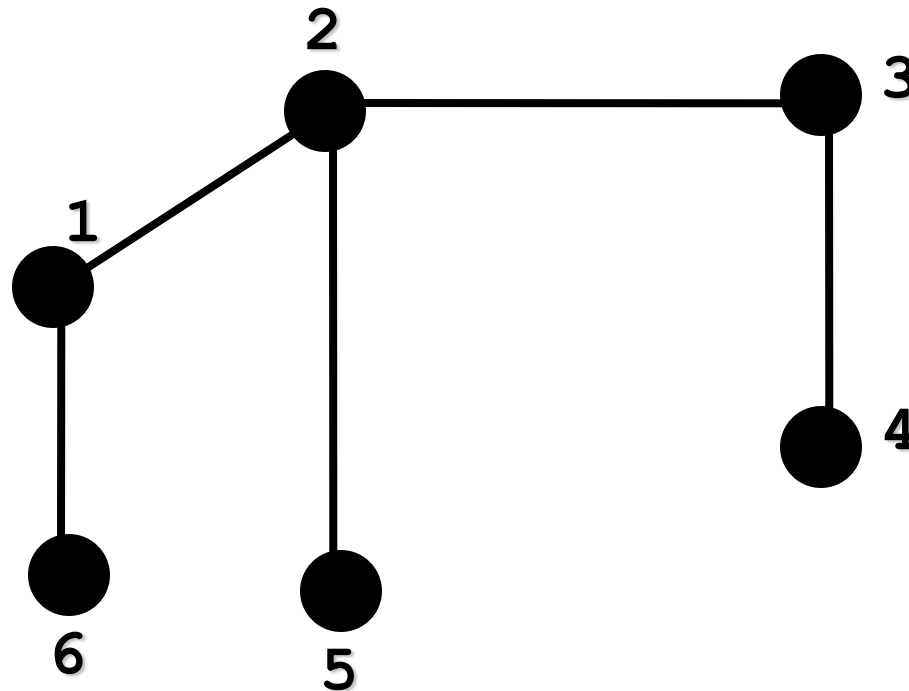
Percorrendo um Grafo: DFS



Vértice 1 não tem mais vértices adjacentes! Fim da busca!

# DFS – exemplo

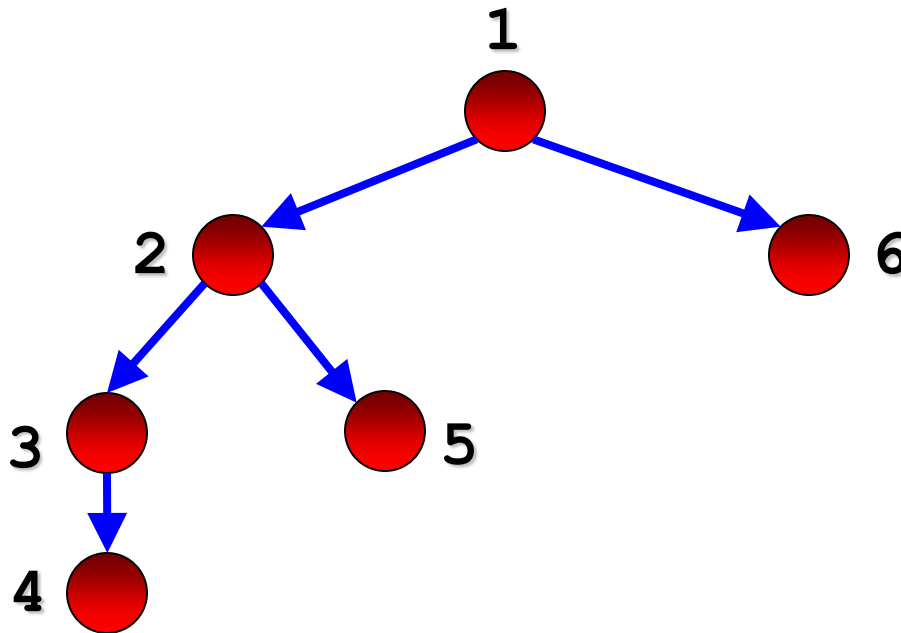
Percorrendo um Grafo: DFS



Sequencia dos vértices (processados): 4, 3, 5, 2, 6, 1

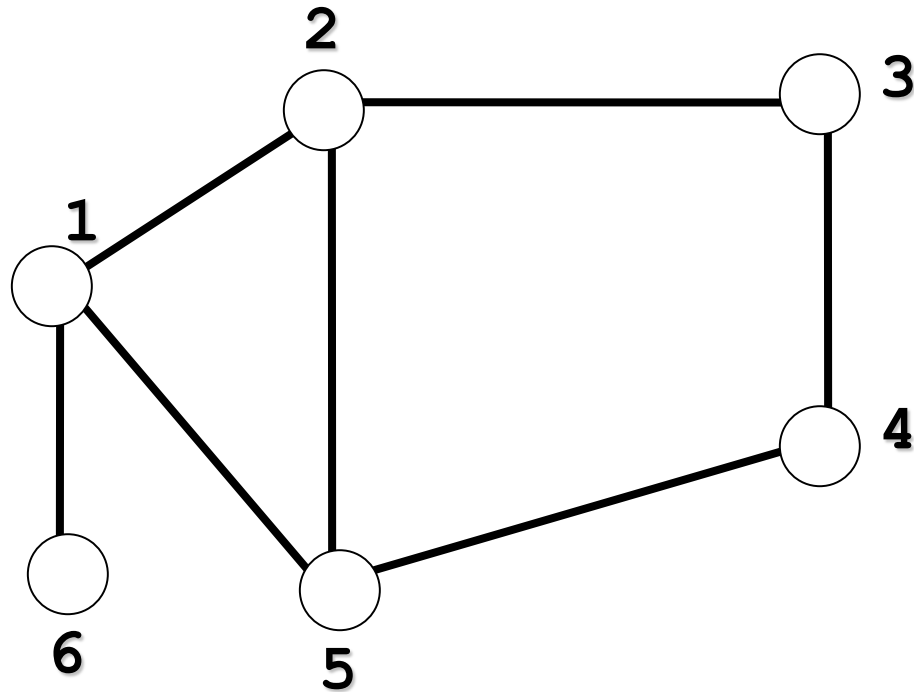
# Árvore de busca em profundidade

Percorrendo um Grafo: **Árvore de Busca em Profundidade**



Todas as arestas foram percorridas: é um mero acaso!

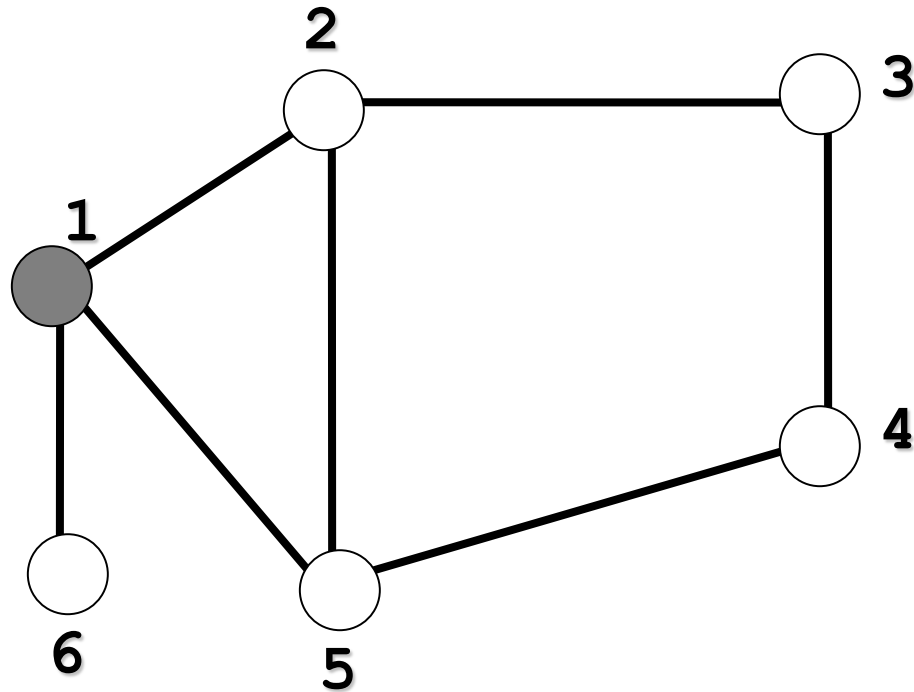
# DFS – exemplo 2



Nós descobertos (cinza)

Nós processados (preto)

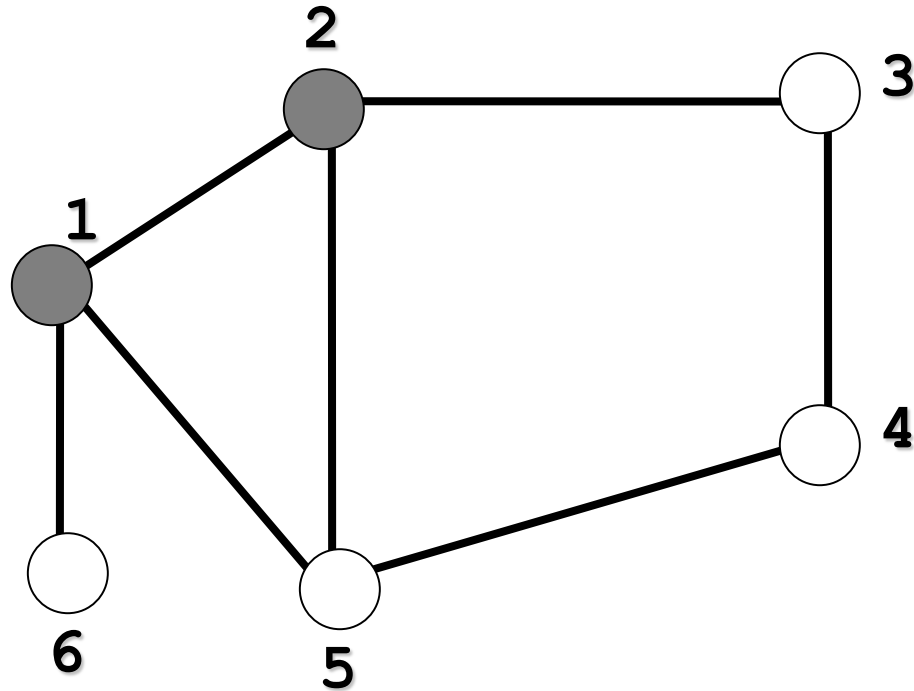
# DFS – exemplo 2



Nós descobertos (cinza): 1

Nós processados (preto):

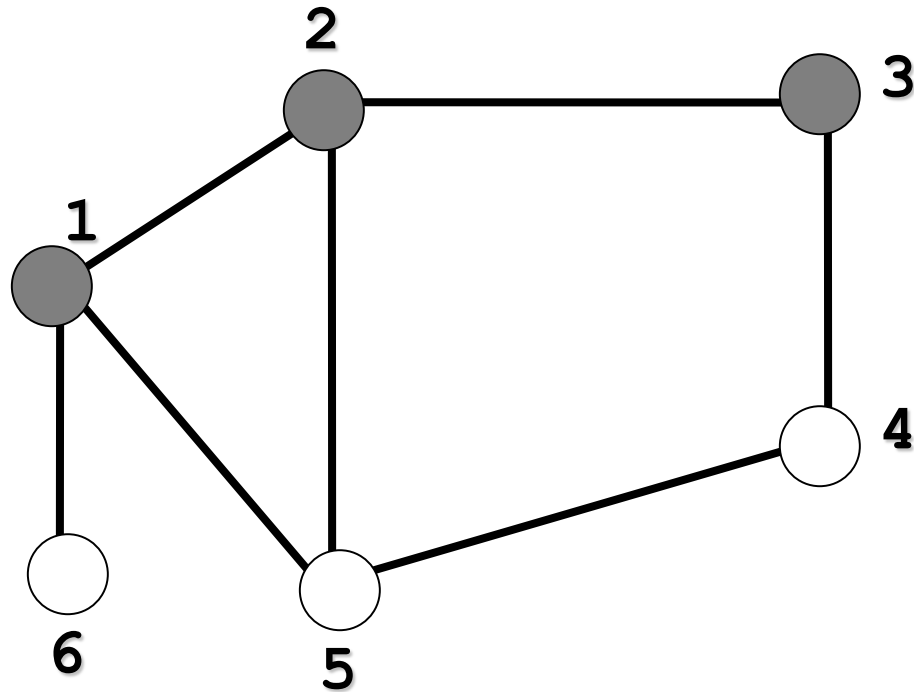
# DFS – exemplo 2



Nós descobertos (cinza): 1 2

Nós processados (preto):

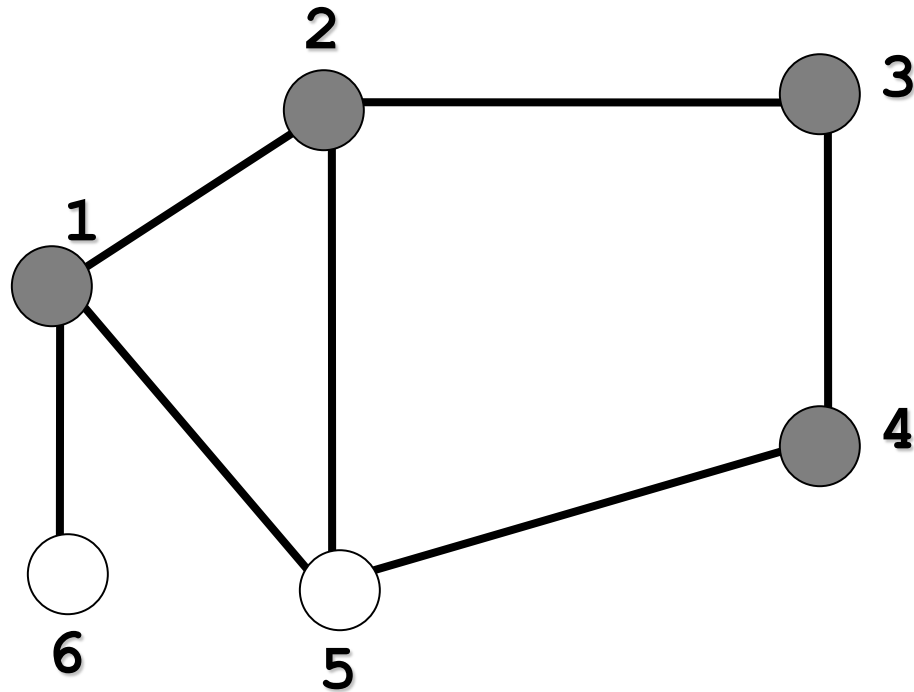
# DFS – exemplo 2



Nós descobertos (cinza): 1 2 3

Nós processados (preto):

# DFS – exemplo 2

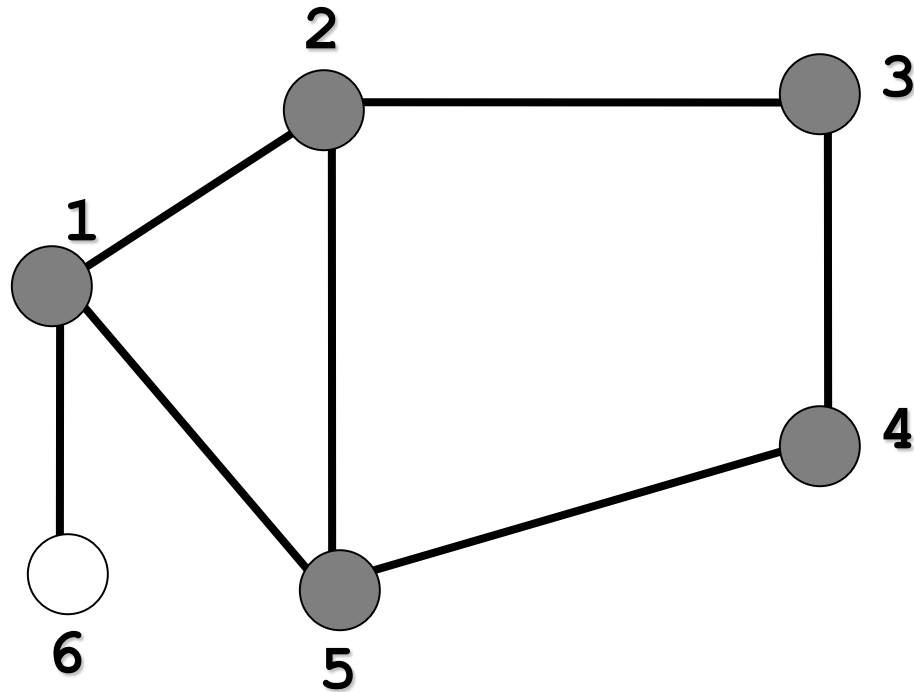


Nós descobertos (cinza): 1 2 3 4

Nós processados (preto):



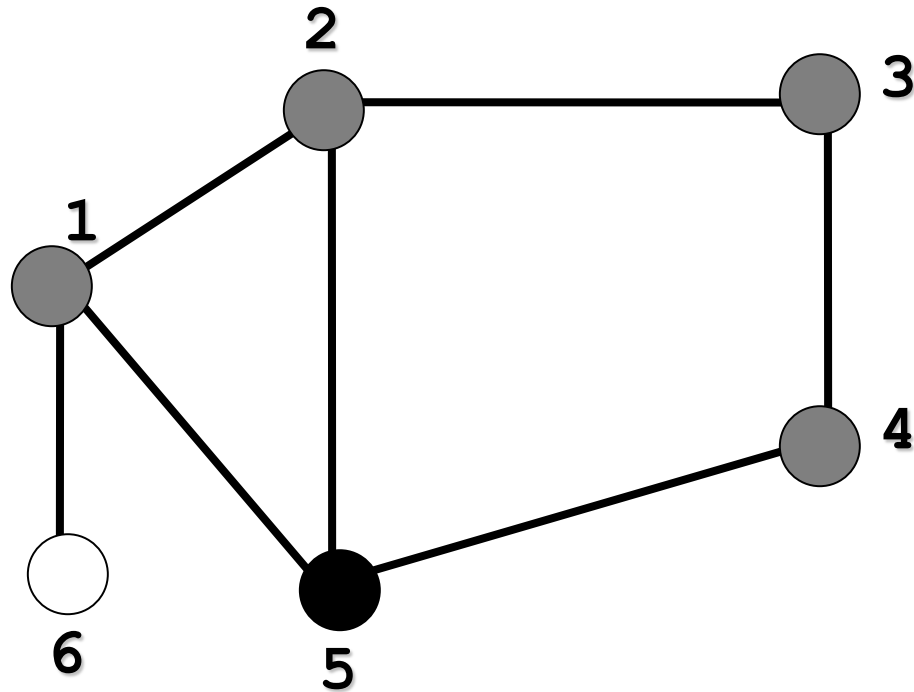
# DFS – exemplo 2



Nós descobertos (cinza): 1 2 3 4 5

Nós processados (preto):

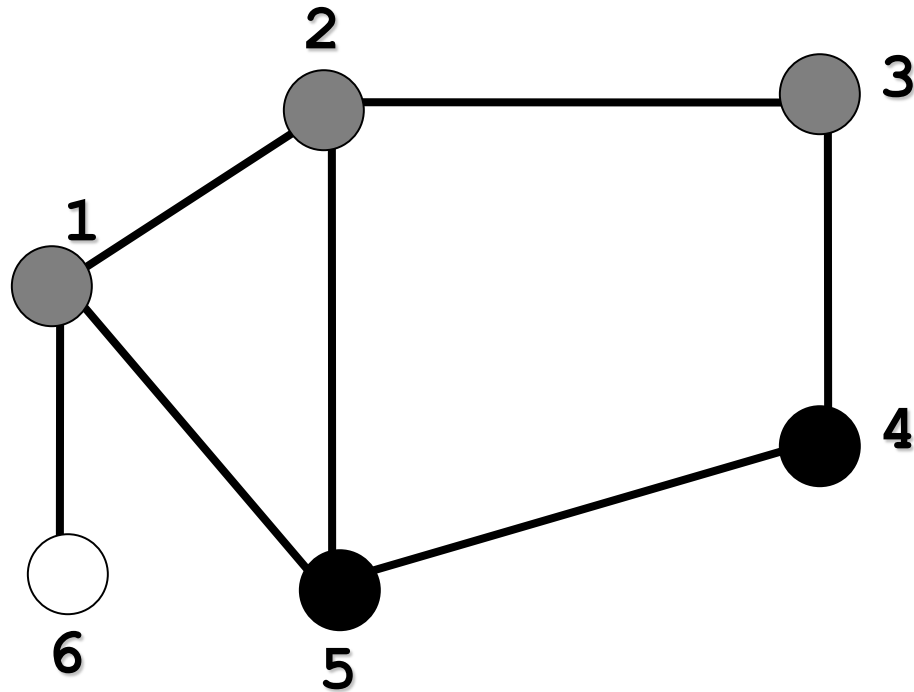
# DFS – exemplo 2



Nós descobertos (cinza): 1 2 3 4 5

Nós processados (preto): 5

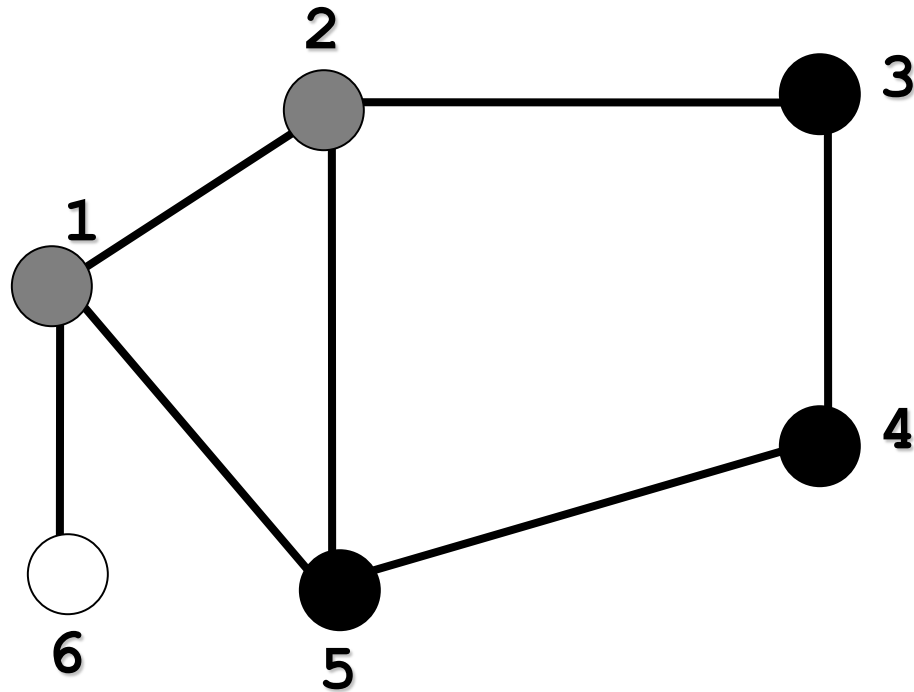
# DFS – exemplo 2



Nós descobertos (cinza): 1 2 3 4 5

Nós processados (preto): 5 4

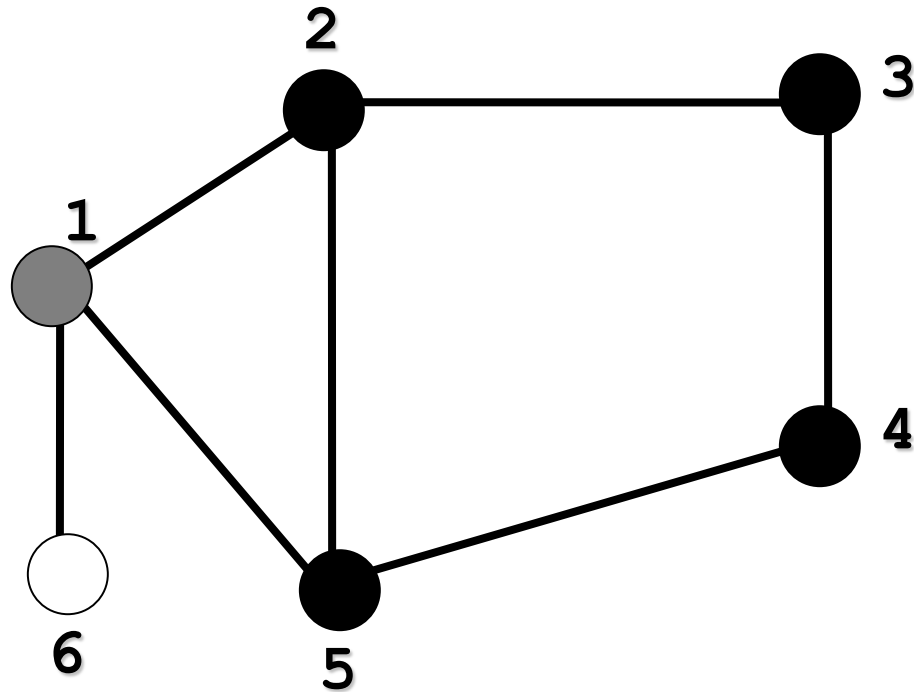
# DFS – exemplo 2



Nós descobertos (cinza): 1 2 3 4 5

Nós processados (preto): 5 4 3

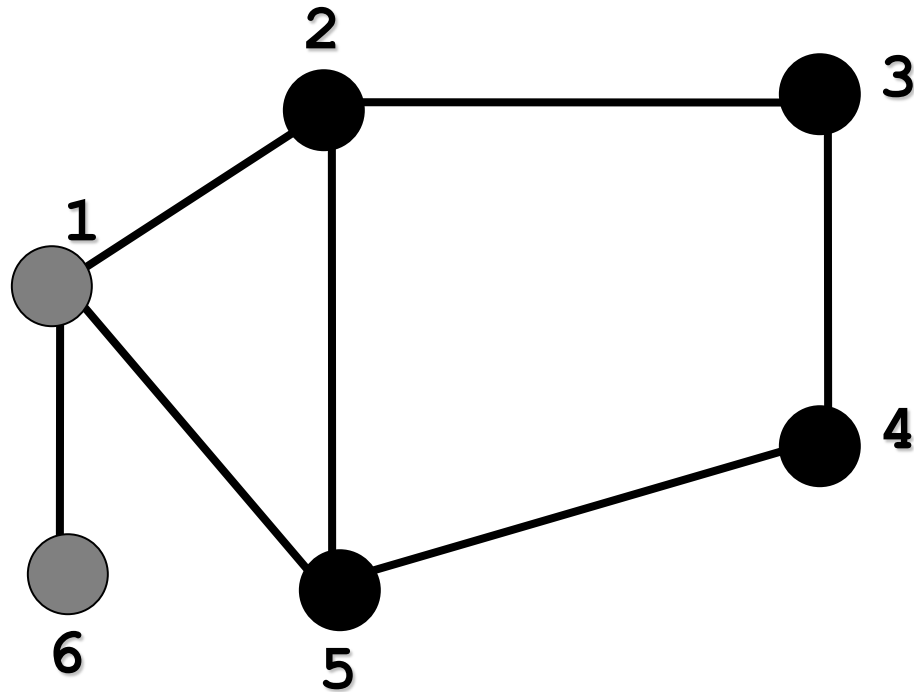
# DFS – exemplo 2



Nós descobertos (cinza): 1 2 3 4 5

Nós processados (preto): 5 4 3 2

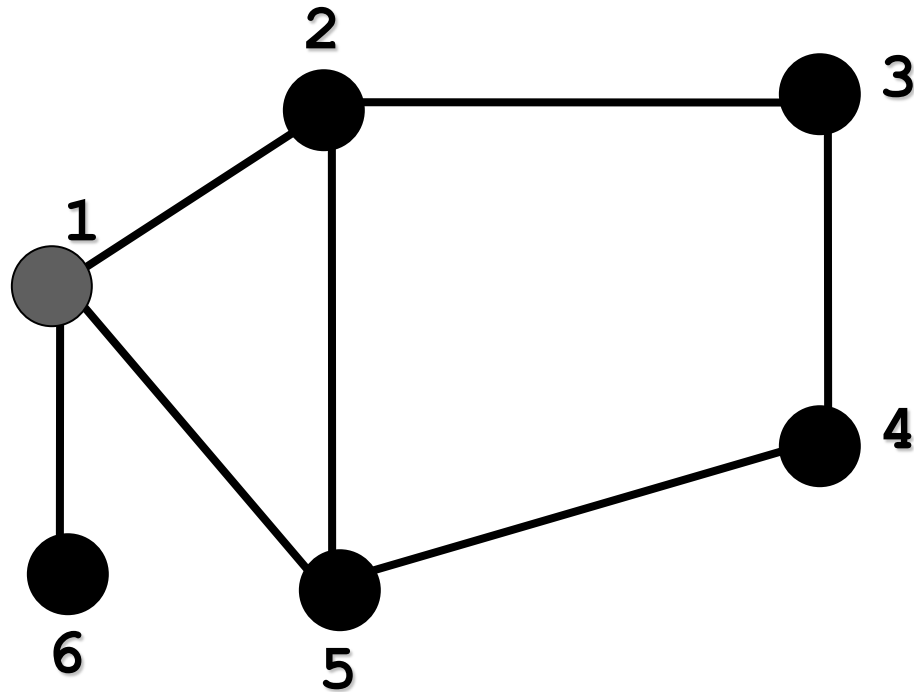
# DFS – exemplo 2



Nós descobertos (cinza): 1 2 3 4 5 6

Nós processados (preto): 5 4 3 2

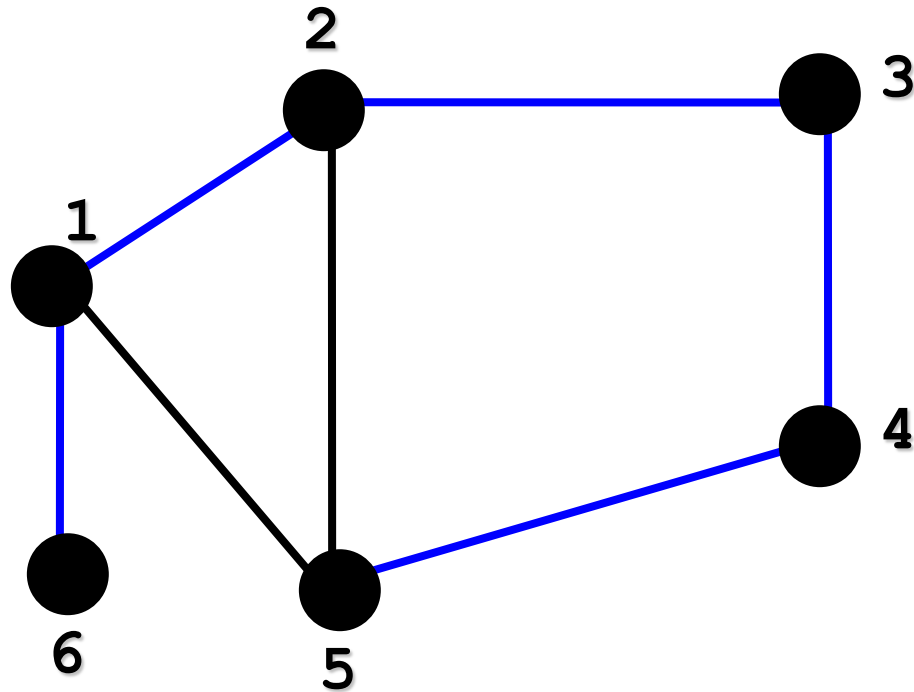
# DFS – exemplo 2



Nós descobertos (cinza): 1 2 3 4 5 6

Nós processados (preto): 5 4 3 2 6

# DFS – exemplo 2



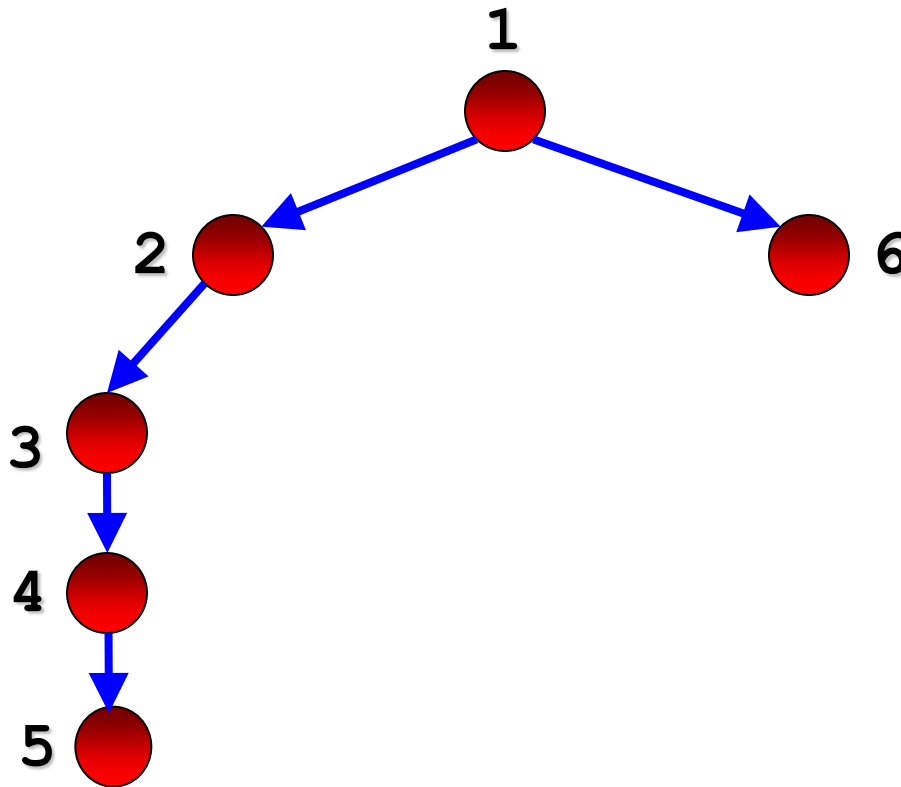
Nós descobertos (cinza): 1 2 3 4 5 6

Nós processados (preto): 5 4 3 2 6 1



# Árvore de busca em profundidade

Percorrendo um Grafo: **Árvore de Busca em Profundidade**



Nem todas as arestas foram percorridas!

---

# DFS

## ■ Algoritmo

- Usa uma **pilha** para organizar os vértices a serem percorridos
  - Pilha pode ser implícita (via recursão) ou explícita

---

# DFS - algoritmo

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

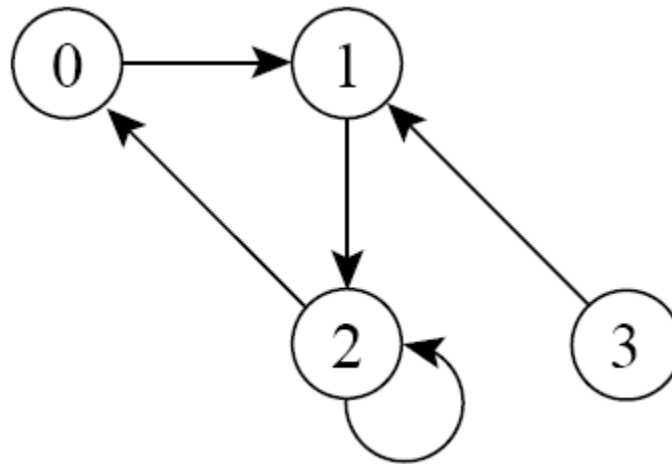
# DFS

## ■ Algoritmo

- Usa uma **pilha** para organizar os vértices a serem percorridos
  - Pilha pode ser implícita (via recursão) ou explícita
- 1. A cada escolha de caminho a ser desenvolvido, empilha-se o vértice original e segue-se o caminho
- 2. Cada vez que o caminho acaba, retorna-se ao vértice anterior empilhado
- Costuma-se registrar o tempo de **descoberta** (cinza) e o tempo de **término** da busca (preto) de cada vértice

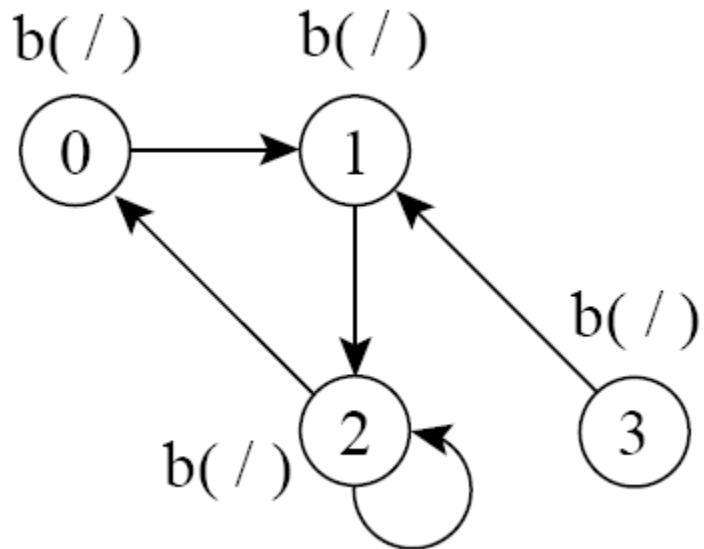
# DFS

- Exemplo detalhado
  - Status de cada vértice, tempo de descoberta e tempo de término



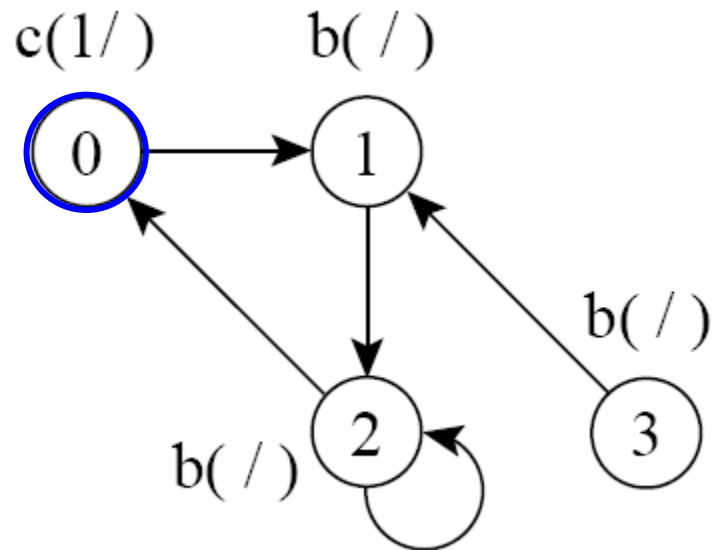
# DFS

- Exemplo detalhado



# DFS

- Exemplo detalhado

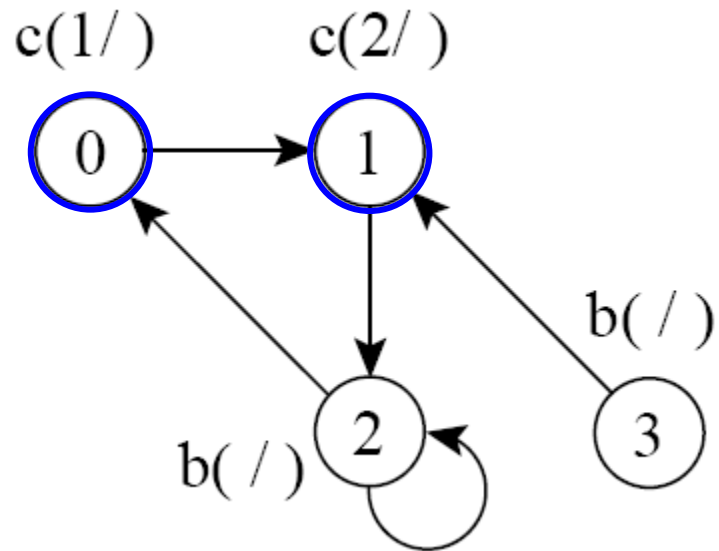


---

b=branco, c=cinza, p=preto, (tempo de descoberta / tempo de término do vértice)

# DFS

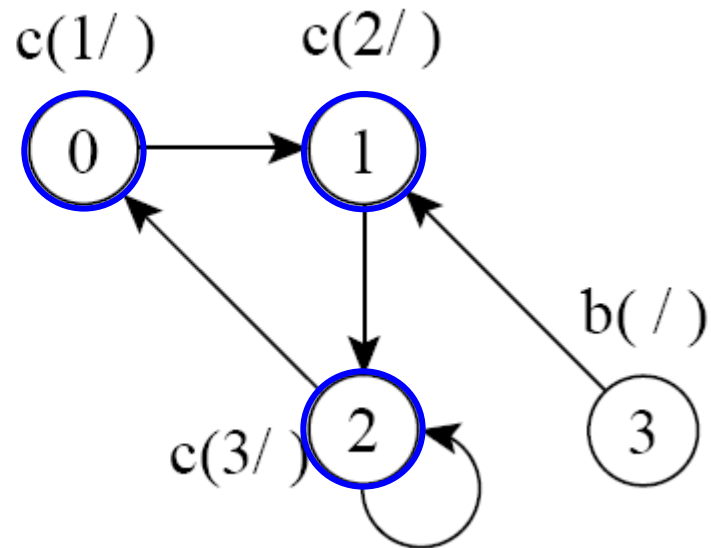
- Exemplo detalhado





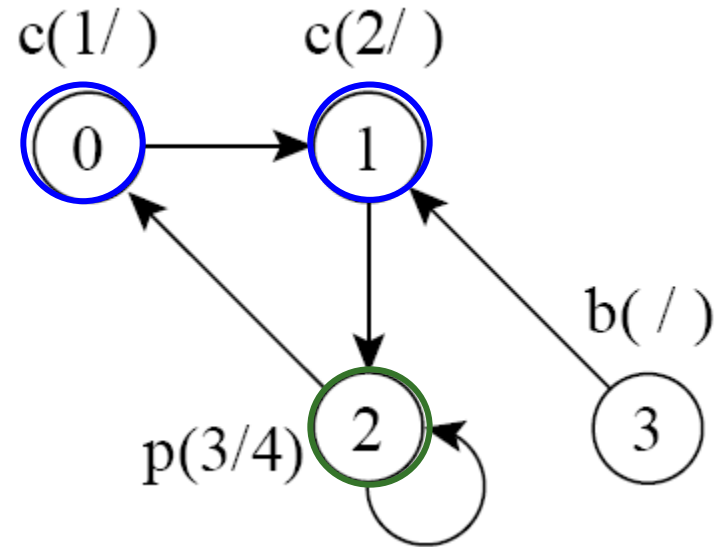
# DFS

- Exemplo detalhado



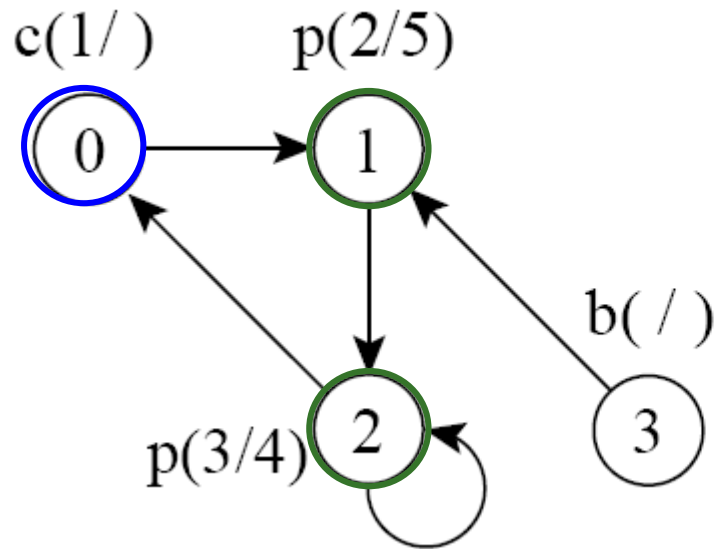
# DFS

- Exemplo detalhado



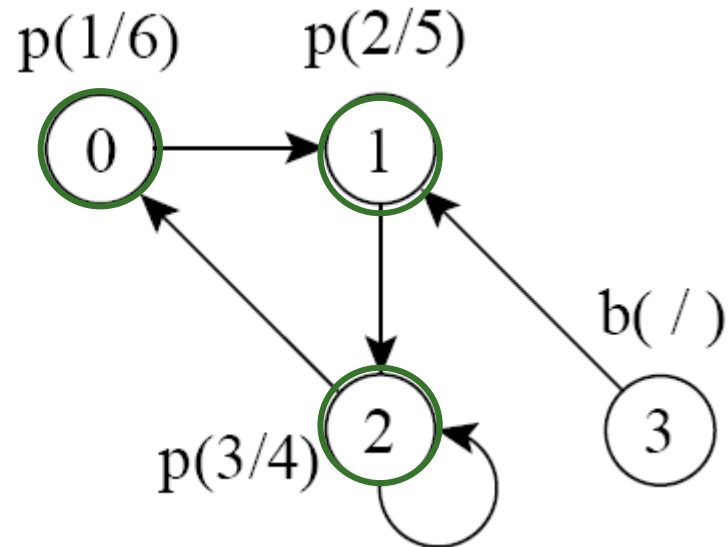
# DFS

- Exemplo detalhado



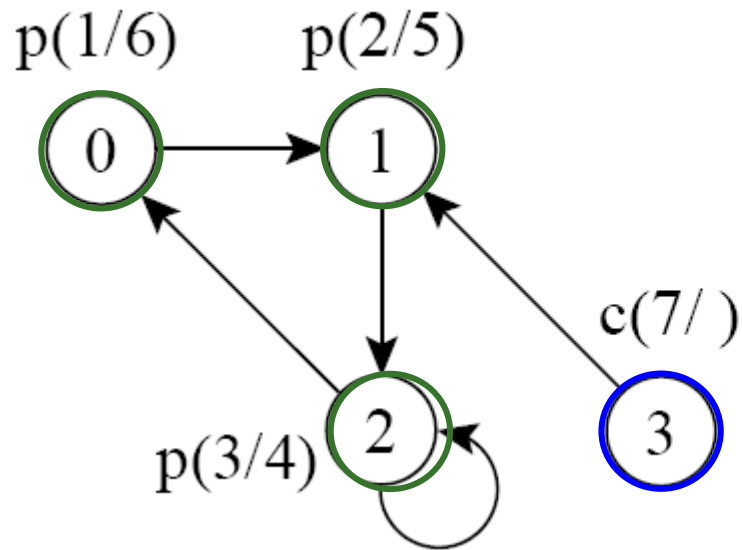
# DFS

- Exemplo detalhado



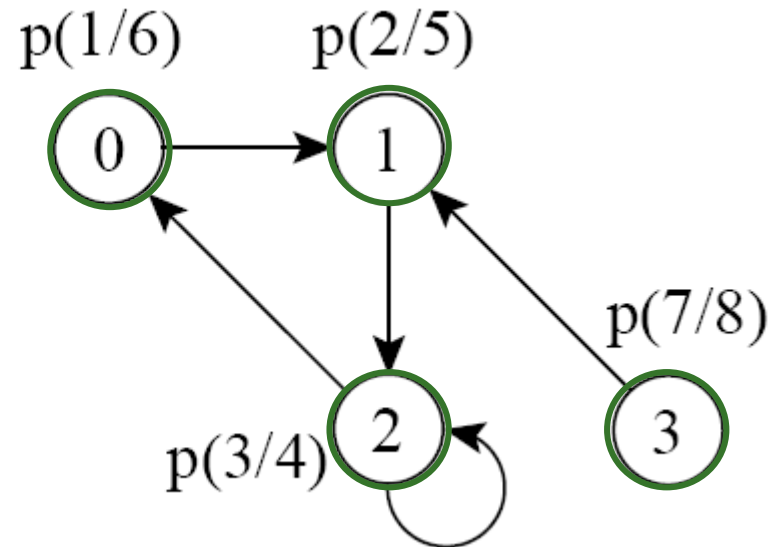
# DFS

- Exemplo detalhado



# DFS

- Exemplo detalhado



# Complexidade do DFS

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

- A função DFS é chamada exatamente uma vez para cada vértice de  $V$
- A cada chamada da função, o laço é executado  $|\text{adj}[v]|$  vezes, i.e., ele será executado  $O(|A|)$  vezes no total

---

# DFS - algoritmo

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



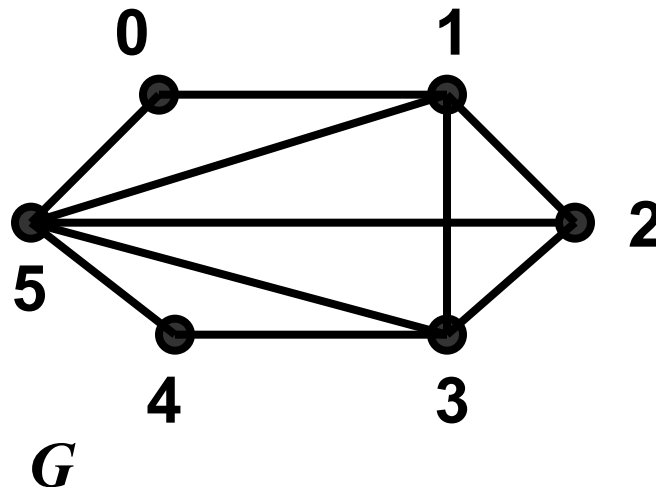
---

# Exercícios

---

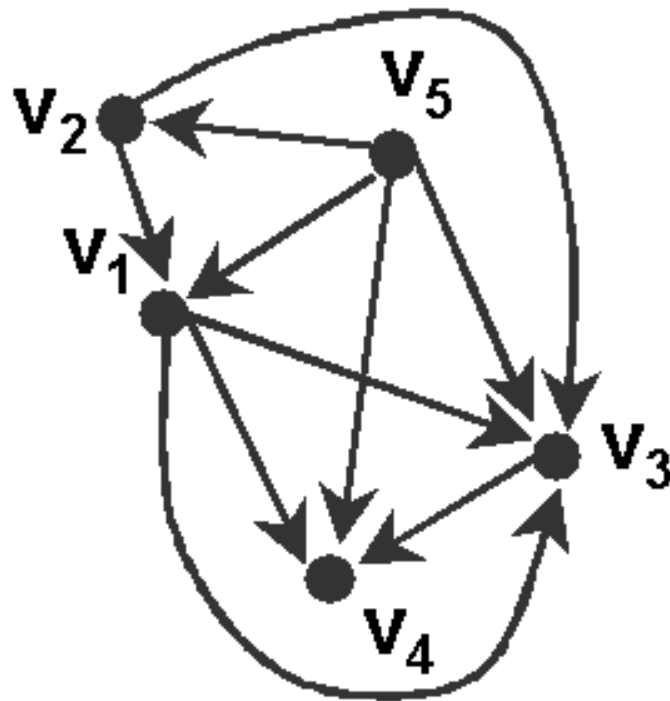
# DFS

- Faça a busca em profundidade no grafo abaixo



# DFS

- Faça a busca em profundidade no grafo abaixo



---

# 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 no Moodle o arquivo com códigos BFS

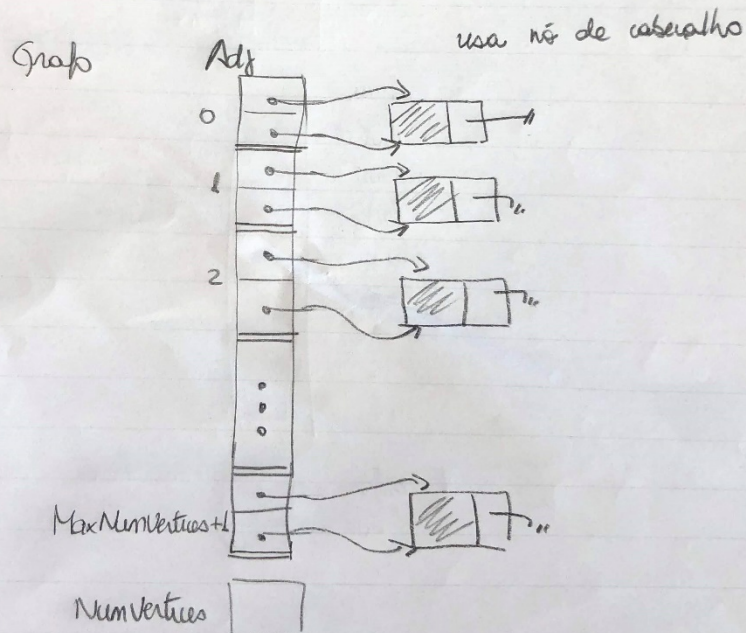
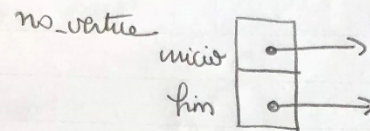
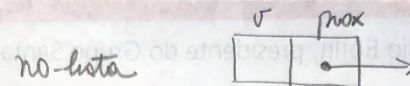


# Lembrando...

```
typedef struct no_lista {  
    elem v;  
    struct no_lista *prox;  
} no_lista;
```

```
typedef struct {  
    no_lista *inicio, *fim;  
} no_vertice;
```

```
typedef struct {  
    no_vertice Adj[MaxNumVertices];  
    int NumVertices;  
} Grafo;
```



# 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);
            }
        }
    }

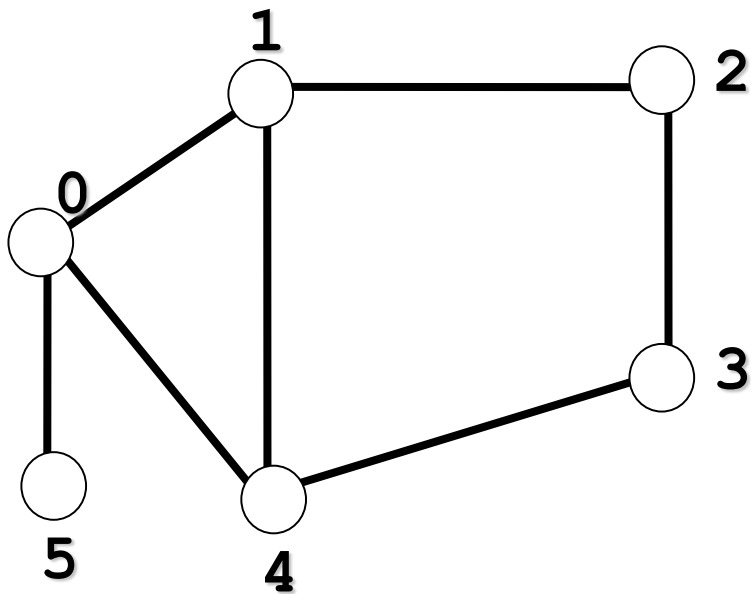
    cor[V]=preto;    (*tempo)++;    t[V]=*tempo;

    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 4, descoberta=5, termino=6, antecessor=3  
No 3, descoberta=4, termino=7, antecessor=2  
No 2, descoberta=3, termino=8, antecessor=1  
No 1, descoberta=2, termino=9, antecessor=0  
No 5, descoberta=10, termino=11, antecessor=0  
No 0, descoberta=1, termino=12, antecessor=-1  
Pressione qualquer tecla para continuar. . .

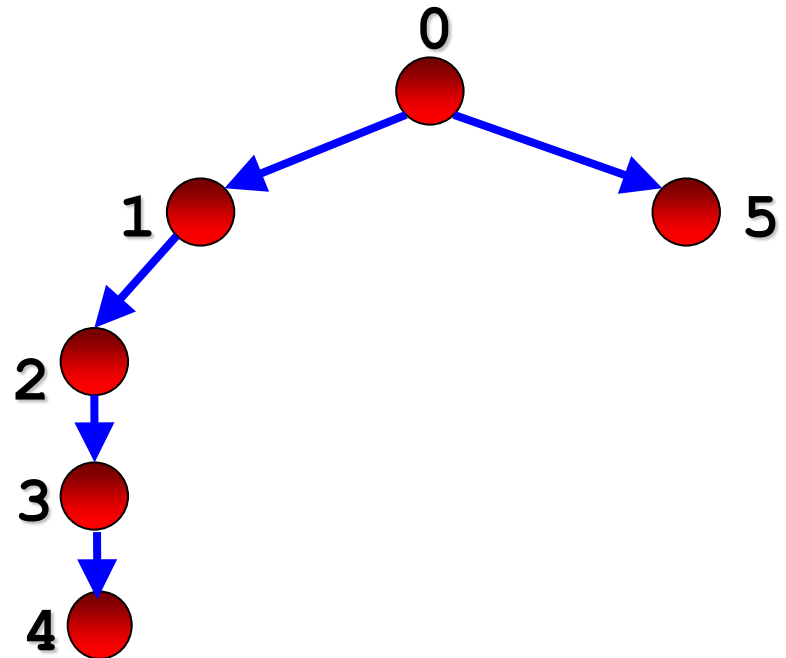
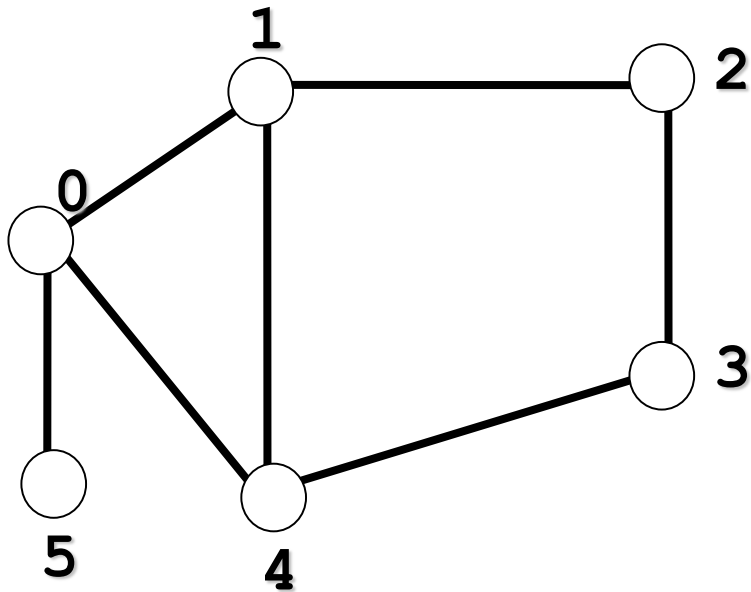
Basta olhar a sequencia de antecessores para saber o caminho a partir do vértice inicial até um vértice qualquer, p.ex.:

Caminho DFS de 0 a 4: 4 3 2 1 0

Caminho DFS de 0 a 5: 5 0

# Implementação DFS

- Execução *busca\_profundidade(Grafo \*G)*



Basta olhar a sequencia de antecessores para saber o caminho a partir do vértice inicial até um vértice qualquer, p.ex.:

Caminho DFS de 0 a 4: 4 3 2 1 0

Caminho DFS de 0 a 5: 5 0



---

# 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  $|\text{adj}[v]|$  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);
```

```
    }
```

```
    }
```

```
}
```

```
    ...
```

```
}
```