Algoritmos e Estruturas de Dados II
Aula 07a – Grafos: Busca em Profundidade - Aplicações

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

Aplicações da Busca em Profundidade?

Grafos – Busca em Profundidade

Aplicações da Busca em Profundidade?

A Busca em Profundidade por ser utilizada (ou adaptada) para ajudar a resolver uma série de problemas:

- **Encontrar soluções para labirintos**
- Verificar se um grafo é acíclico
- Realizar a ordenação topológica
- Encontrar componentes conexos/conectados
- Encontrar componentes fortemente conexos/conectados
- ...

Encontrar uma Solução para um Labirinto

Representação:

Encontrar uma Solução para um Labirinto

Representação:

- Cada posição do labirinto corresponderá a um **vértice**.

Encontrar uma Solução para um Labirinto

Representação:

- Cada posição do labirinto corresponderá a um **vértice**.
- **Arestas** são criadas para ligar os vértices que correspondem a posições adjacentes do labirinto.

Encontrar uma Solução para um Labirinto

Como vimos, a busca em profundidade navega, recursivamente, **a partir de um até todos os nós alcançáveis**/atingíveis a partir dele.

Encontrar uma Solução para um Labirinto

Como vimos, a busca em profundidade navega, recursivamente, **a partir de um até todos os nós alcançáveis**/atingíveis a partir dele.

Assim, para verificarmos se é possível chegar na saída de um labirinto, basta **chamarmos a busca a partir do início**.

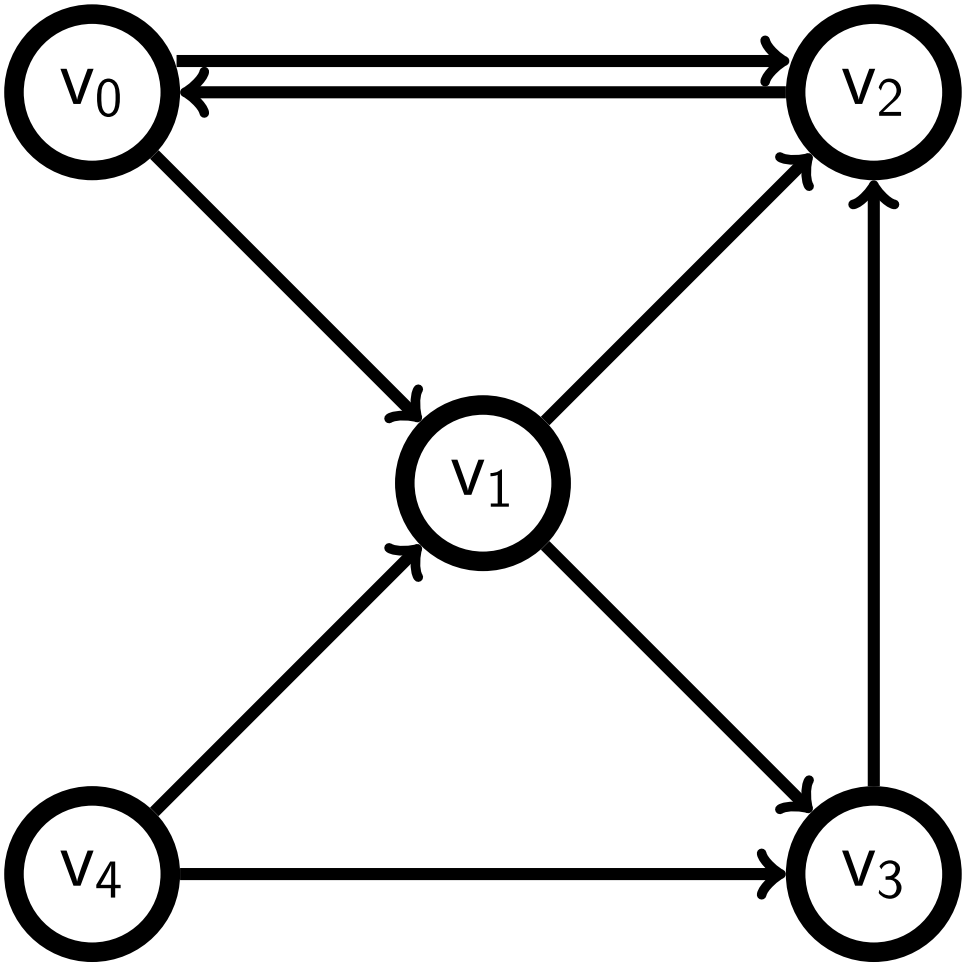
Encontrar uma Solução para um Labirinto

Como vimos, a busca em profundidade navega, recursivamente, **a partir de um até todos os nós alcançáveis**/atingíveis a partir dele.

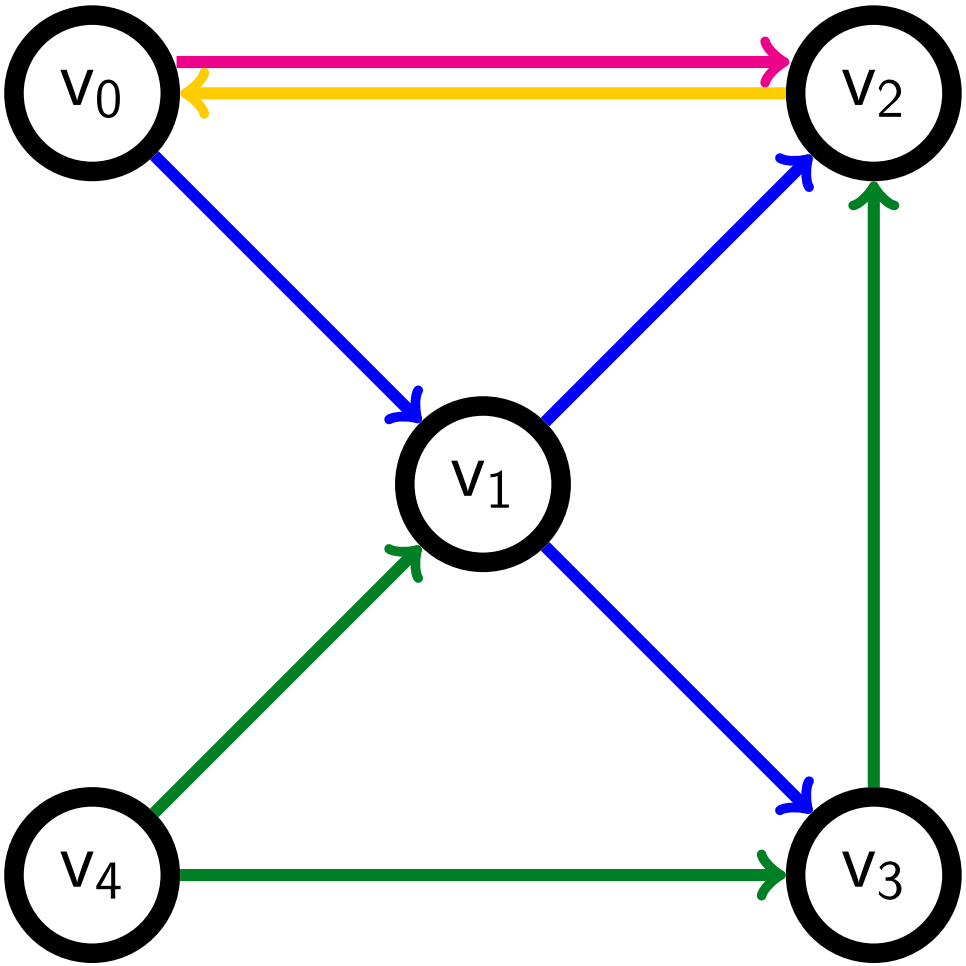
Assim, para verificarmos se é possível chegar na saída de um labirinto, basta **chamarmos a busca a partir do início**.

Faremos algumas alterações para que a busca receba a o vértice de **saída** /destino, bem como para retornar o **caminho percorrido**.

Grafos – Busca em Profundidade



Grafos – Busca em Profundidade

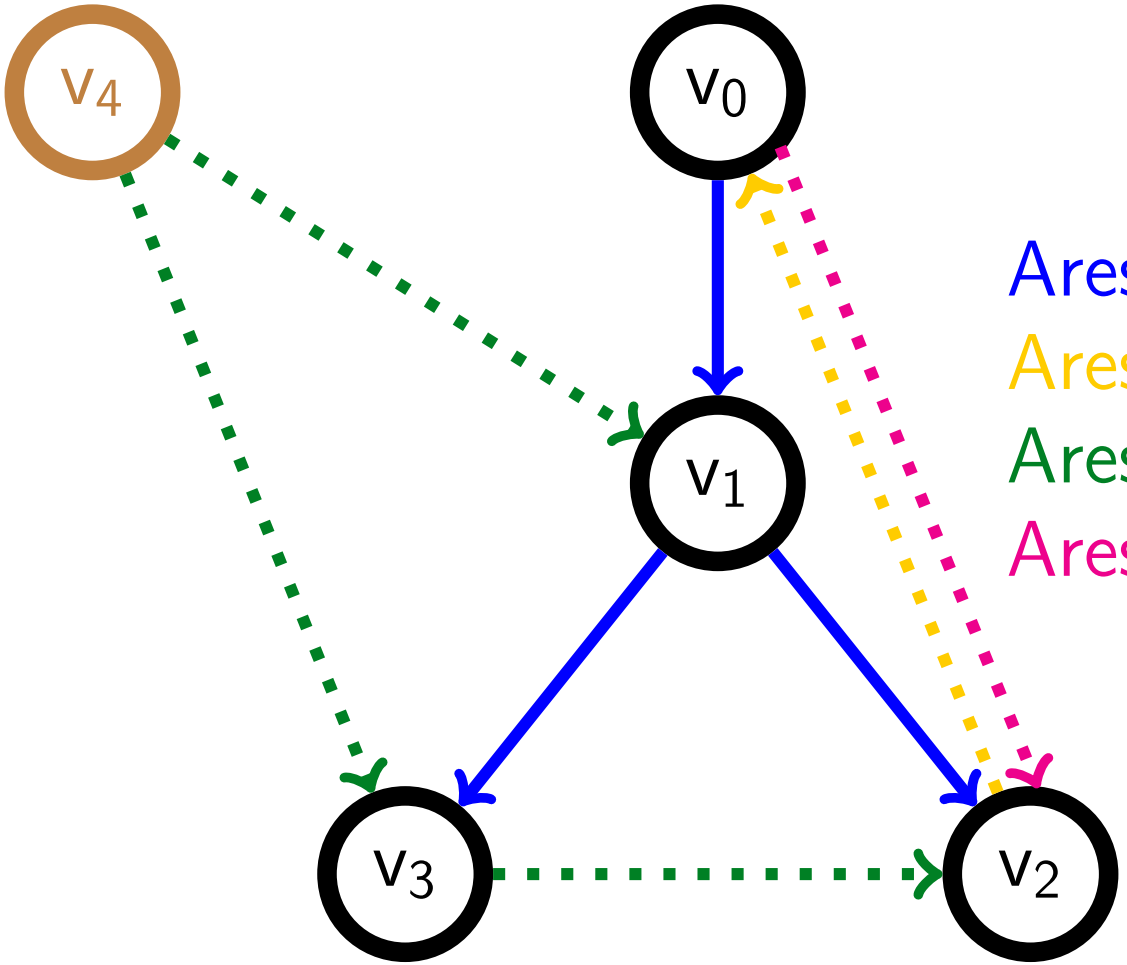


Resumo da Busca em Profundidade:

Nó	Anterior	Descoberta	Término	Cor:
0	-1	1	8	2
1	0	2	7	2
2	1	3	4	2
3	1	5	6	2
4	-1	9	10	2

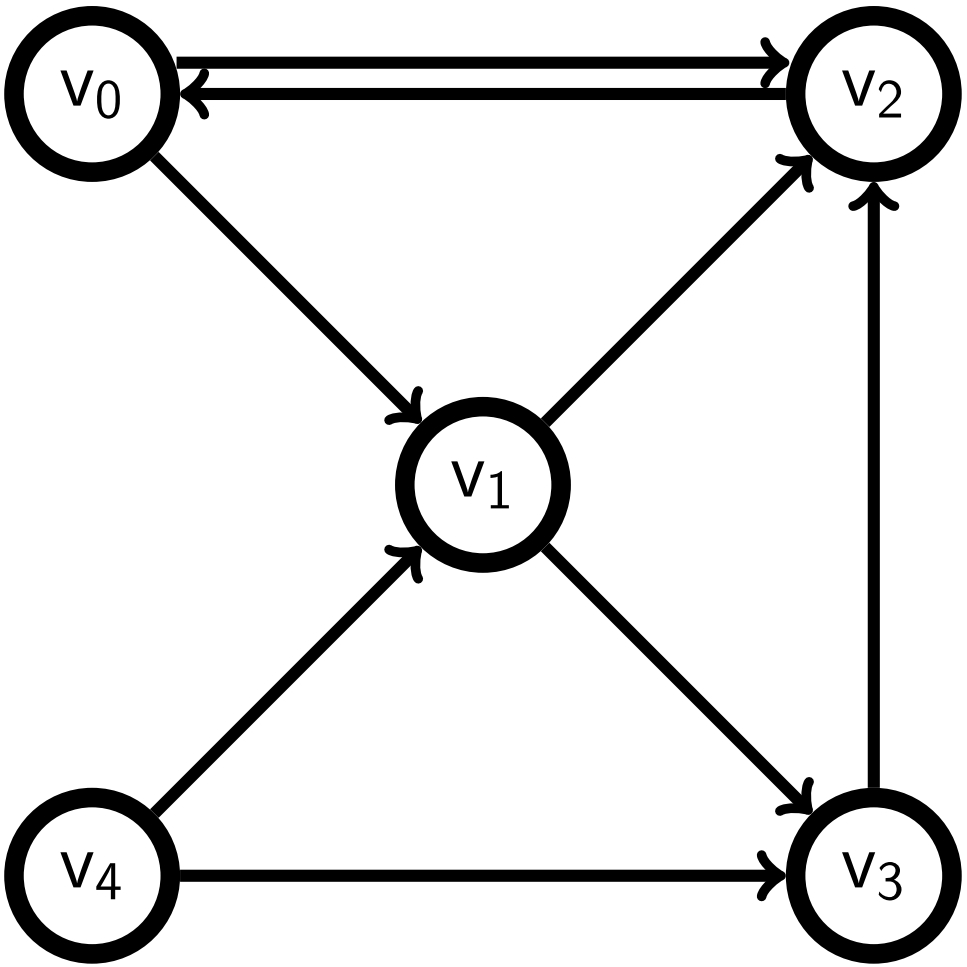
Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Árvores de Busca em Profundidade



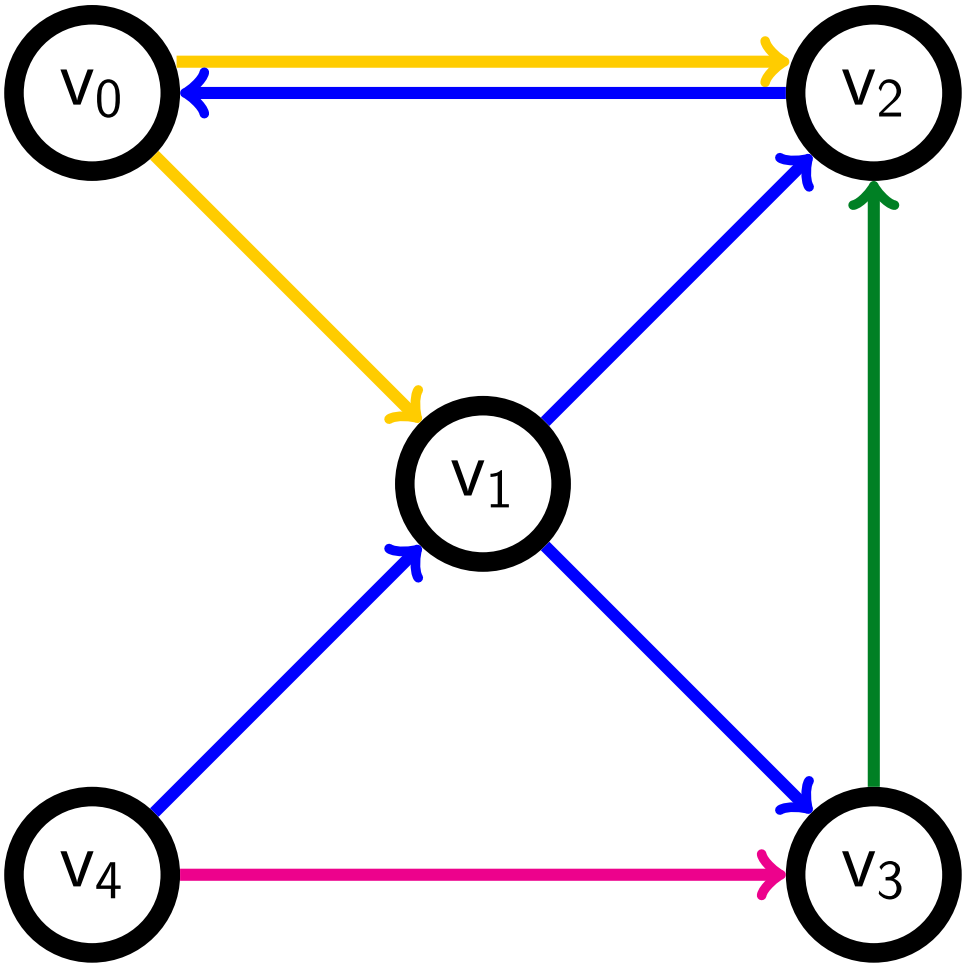
Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Grafos – Busca em Profundidade



E se começarmos do vértice $v4$?

Grafos – Busca em Profundidade

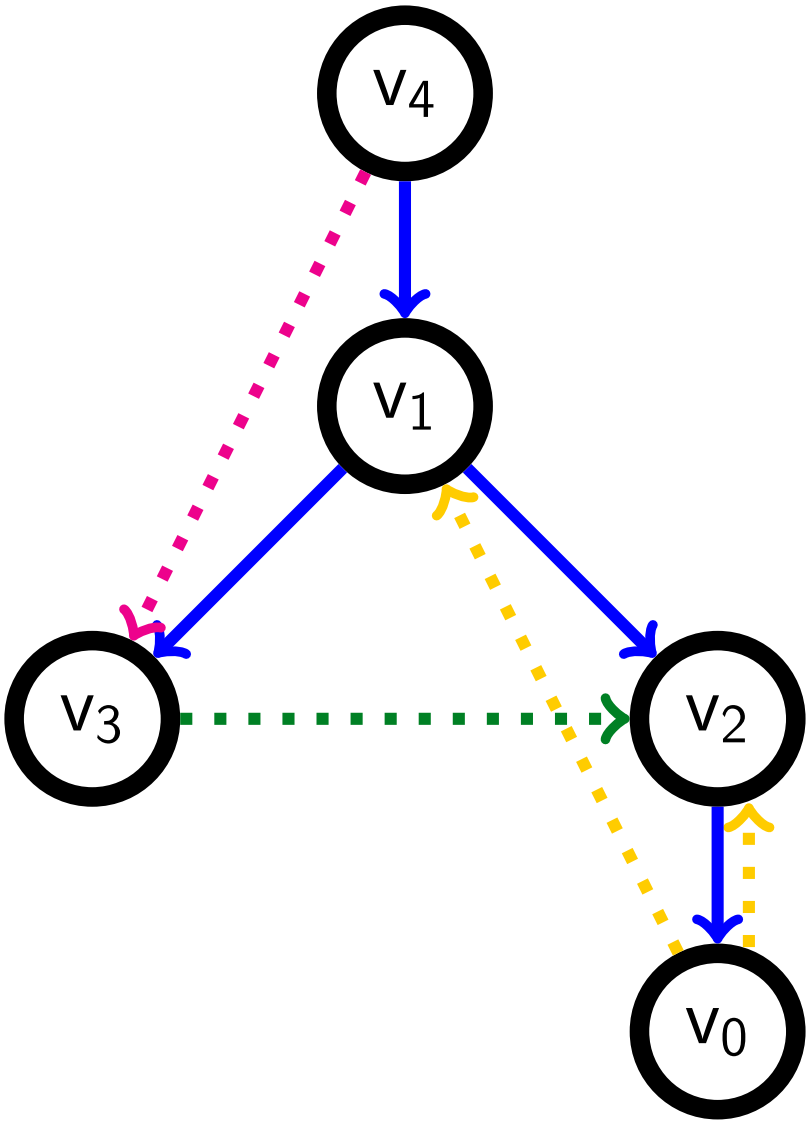


E se começarmos do vértice $v4$?

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        2          4       5   2
 1        4          2       9   2
 2        1          3       6   2
 3        1          7       8   2
 4       -1          1      10   2
```

Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Árvore de Busca em Profundidade



Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Encontrar uma Solução para um Labirinto

Observação:

Encontrar uma Solução para um Labirinto

Observação:

A Busca em Profundidade **não garante** que o caminho encontrado será o de menor tamanho.

Encontrar uma Solução para um Labirinto

Nosso caminho construído como uma lista ligada, criada na volta da recursão (da saída para "trás").

```
ElemLista* novoElemento(int vertice, ElemLista* prox){
  ElemLista* novo = (ElemLista*) malloc(sizeof(ElemLista));
  novo->vertice = vertice;
  novo->prox = prox;
  return novo;
}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){

}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){
  if (atual == destino){
        cabeca->prox = novoElemento(atual, cabeca->prox);
        return true;
  }









}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){
  if (atual == destino){
        cabeca->prox = novoElemento(atual, cabeca->prox);
        return true;
  }
  visitado[atual] = true;









}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){
  if (atual == destino){
        cabeca->prox = novoElemento(atual, cabeca->prox);
        return true;
  }
  visitado[atual] = true;
  ElemLista *end = g->A[atual];
  while (end){









  }

}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){
  if (atual == destino){
          cabeca->prox = novoElemento(atual, cabeca->prox);
          return true;
  }
  visitado[atual] = true;
  ElemLista *end = g->A[atual];
  while (end){
    if (!visitado[end->vertice])
      if (DFSCaminhoAux(g,end->vertice,destino,visitado,cabeca)){



      }

  }

}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){
  if (atual == destino){
          cabeca->prox = novoElemento(atual, cabeca->prox);
          return true;
  }
  visitado[atual] = true;
  ElemLista *end = g->A[atual];
  while (end){
    if (!visitado[end->vertice])
      if (DFSCaminhoAux(g,end->vertice,destino,visitado,cabeca)){
          cabeca->prox = novoElemento(atual, cabeca->prox);
          return true;
      }

  }

}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){
  if (atual == destino){
          cabeca->prox = novoElemento(atual, cabeca->prox);
          return true;
  }
  visitado[atual] = true;
  ElemLista *end = g->A[atual];
  while (end){
    if (!visitado[end->vertice])
      if (DFSCaminhoAux(g,end->vertice,destino,visitado,cabeca)){
          cabeca->prox = novoElemento(atual, cabeca->prox);
          return true;
      }
    end = end->prox;
  }

}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){
  if (atual == destino){
        cabeca->prox = novoElemento(atual, cabeca->prox);
        return true;
  }
  visitado[atual] = true;
  ElemLista *end = g->A[atual];
  while (end){
    if (!visitado[end->vertice])
      if (DFSCaminhoAux(g,end->vertice,destino,visitado,cabeca)){
        cabeca->prox = novoElemento(atual, cabeca->prox);
        return true;
      }
    end = end->prox;
  }
  return false;
}
```

Encontrar uma Solução para um Labirinto

```
bool DFSCaminhoAux(Grafo* g, int atual, int destino,
                   bool* visitado, ElemLista* cabeca){
  if (atual == destino){
        cabeca->prox = novoElemento(atual, cabeca->prox);
        return true;
  }
  visitado[atual] = true;
  ElemLista *end = g->A[atual];
  while (end){
    if (!visitado[end->vertice])
      if (DFSCaminhoAux(g,end->vertice,destino,visitado,cabeca)){
        cabeca->prox = novoElemento(atual, cabeca->prox);
        return true;
      }
    end = end->prox;
  }
  return false;
}
```

$O(V + E)$

Encontrar uma Solução para um Labirinto

```
ElemLista* DFSCaminho(Grafo* g, int origem, int destino){




}
```

Encontrar uma Solução para um Labirinto

```
ElemLista* DFSCaminho(Grafo* g, int origem, int destino){
  ElemLista *inicio;
  ElemLista *cabeca = (ElemLista*) malloc(sizeof(ElemLista));
  cabeca->prox = NULL;









}
```

Encontrar uma Solução para um Labirinto

```
ElemLista* DFSCaminho(Grafo* g, int origem, int destino){
  ElemLista *inicio;
  ElemLista *cabeca = (ElemLista*) malloc(sizeof(ElemLista));
  cabeca->prox = NULL;
  int* visitado = (int*) malloc(sizeof(int)*g->numVertices);
  int x;
  for (x=0;x<g->numVertices;x++) visitado[x] = false;









}
```

Encontrar uma Solução para um Labirinto

```
ElemLista* DFSCaminho(Grafo* g, int origem, int destino){
  ElemLista *inicio;
  ElemLista *cabeca = (ElemLista*) malloc(sizeof(ElemLista));
  cabeca->prox = NULL;
  int* visitado = (int*) malloc(sizeof(int)*g->numVertices);
  int x;
  for (x=0;x<g->numVertices;x++) visitado[x] = false;

  DFSCaminhoAux(g, origem, destino, visitado, cabeca);



}
```

Encontrar uma Solução para um Labirinto

```
ElemLista* DFSCaminho(Grafo* g, int origem, int destino){
  ElemLista *inicio;
  ElemLista *cabeca = (ElemLista*) malloc(sizeof(ElemLista));
  cabeca->prox = NULL;
  int* visitado = (int*) malloc(sizeof(int)*g->numVertices);
  int x;
  for (x=0;x<g->numVertices;x++) visitado[x] = false;

  DFSCaminhoAux(g, origem, destino, visitado, cabeca);
  inicio = cabeca->prox;
  free(cabeca);
  free(visitado);
  return inicio;
}
```

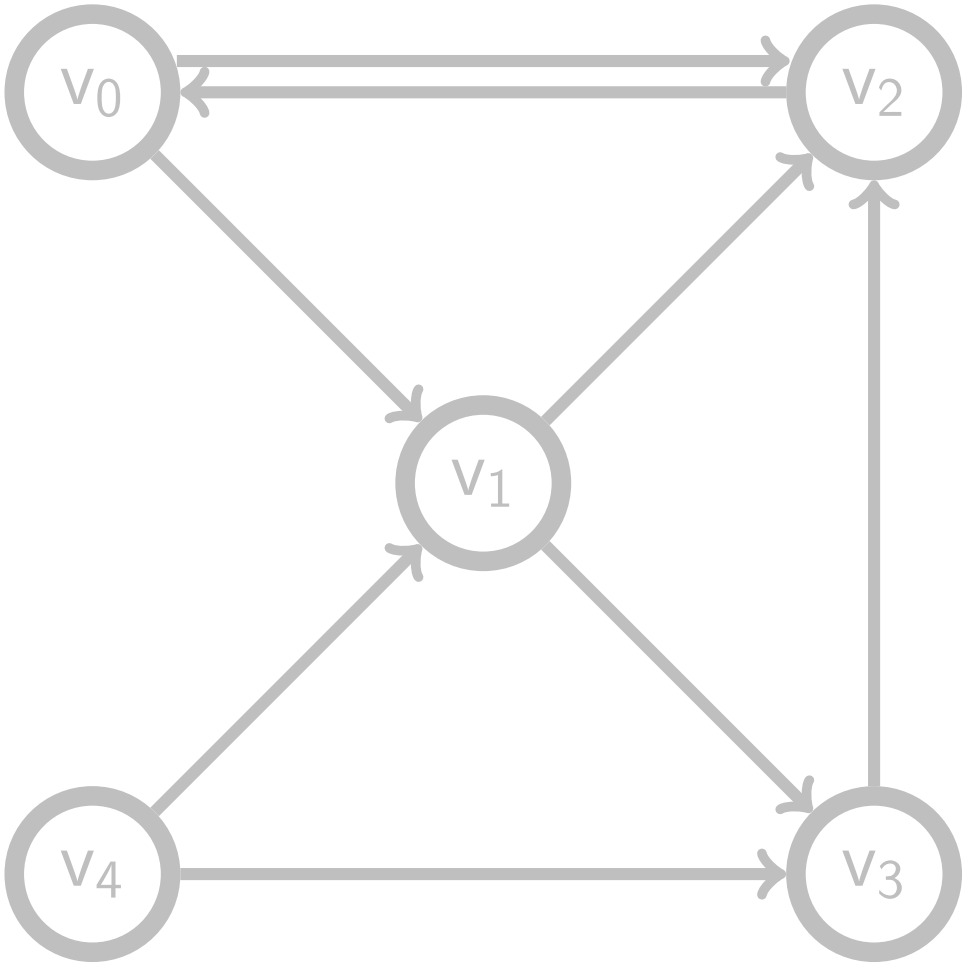
Encontrar uma Solução para um Labirinto

```
ElemLista* DFSCaminho(Grafo* g, int origem, int destino){
  ElemLista *inicio;
  ElemLista *cabeca = (ElemLista*) malloc(sizeof(ElemLista));
  cabeca->prox = NULL;
  int* visitado = (int*) malloc(sizeof(int)*g->numVertices);
  int x;
  for (x=0;x<g->numVertices;x++) visitado[x] = false;

  DFSCaminhoAux(g, origem, destino, visitado, cabeca);
  inicio = cabeca->prox;
  free(cabeca);
  free(visitado);
  return inicio;
}
```

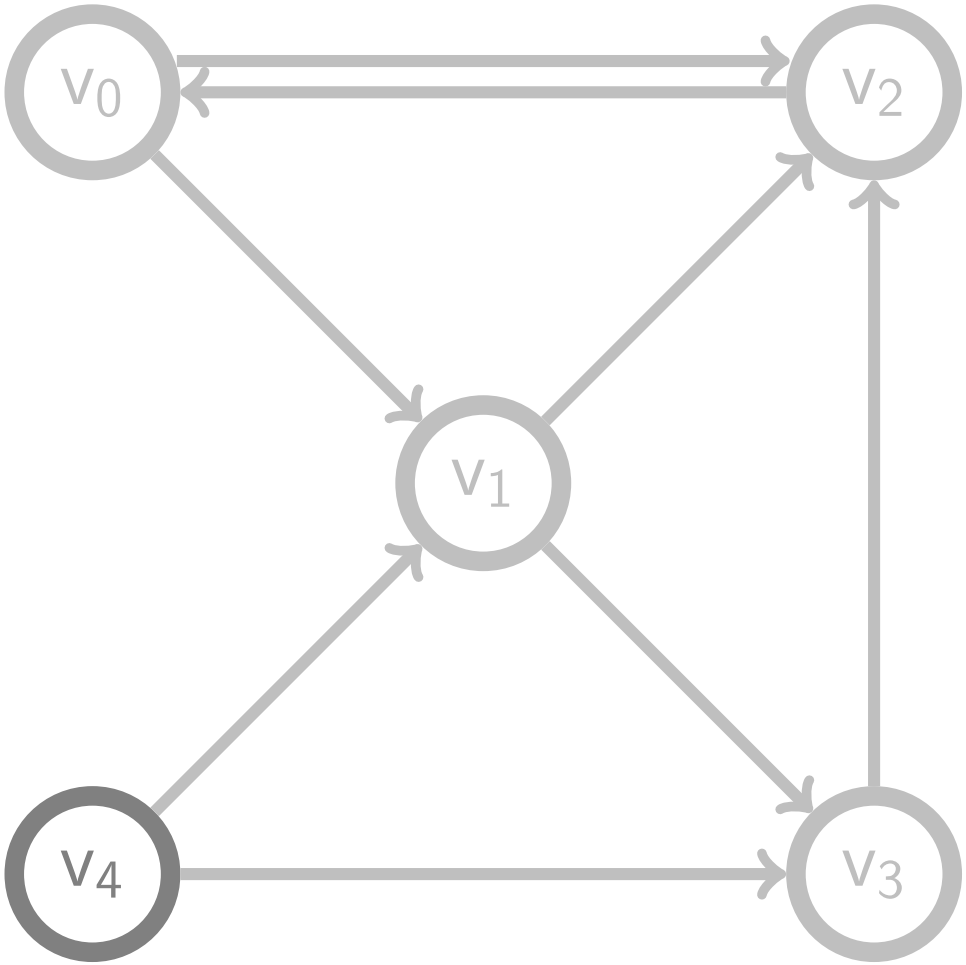
$$O(V + E)$$

Encontrar uma Solução para um Labirinto



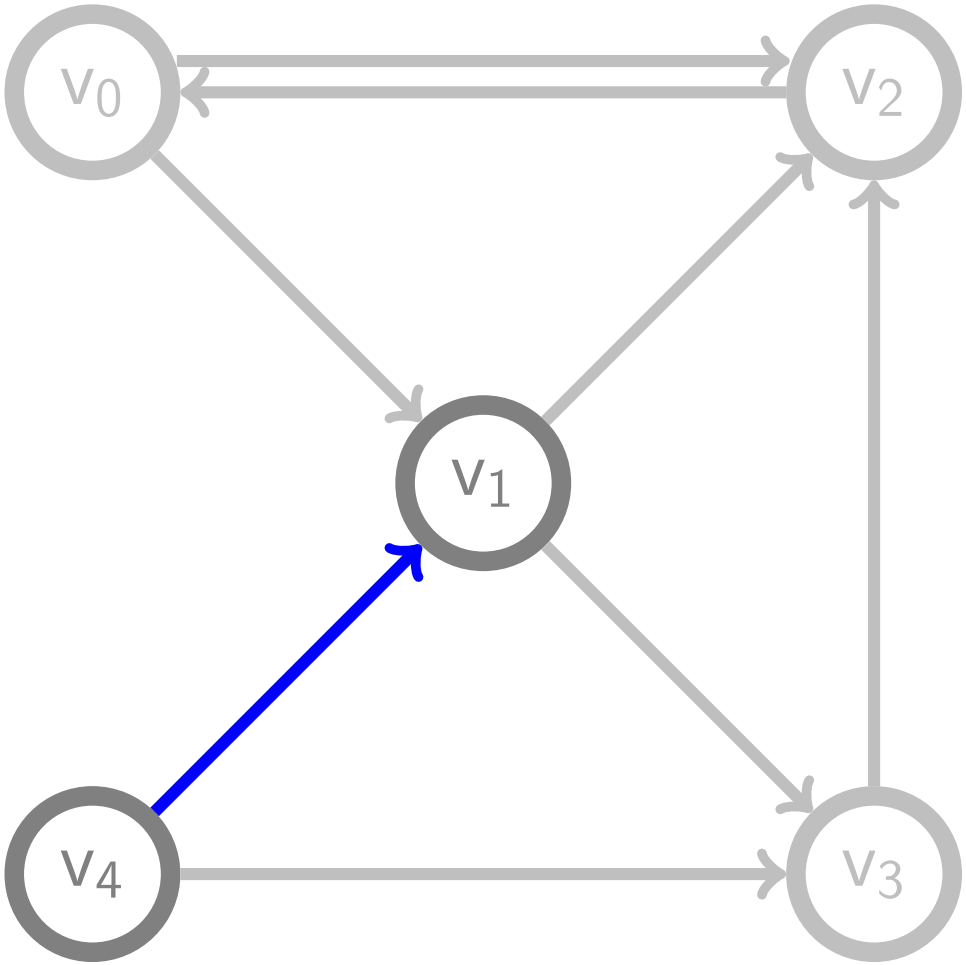
Início $v4$, destino $v3$

Encontrar uma Solução para um Labirinto



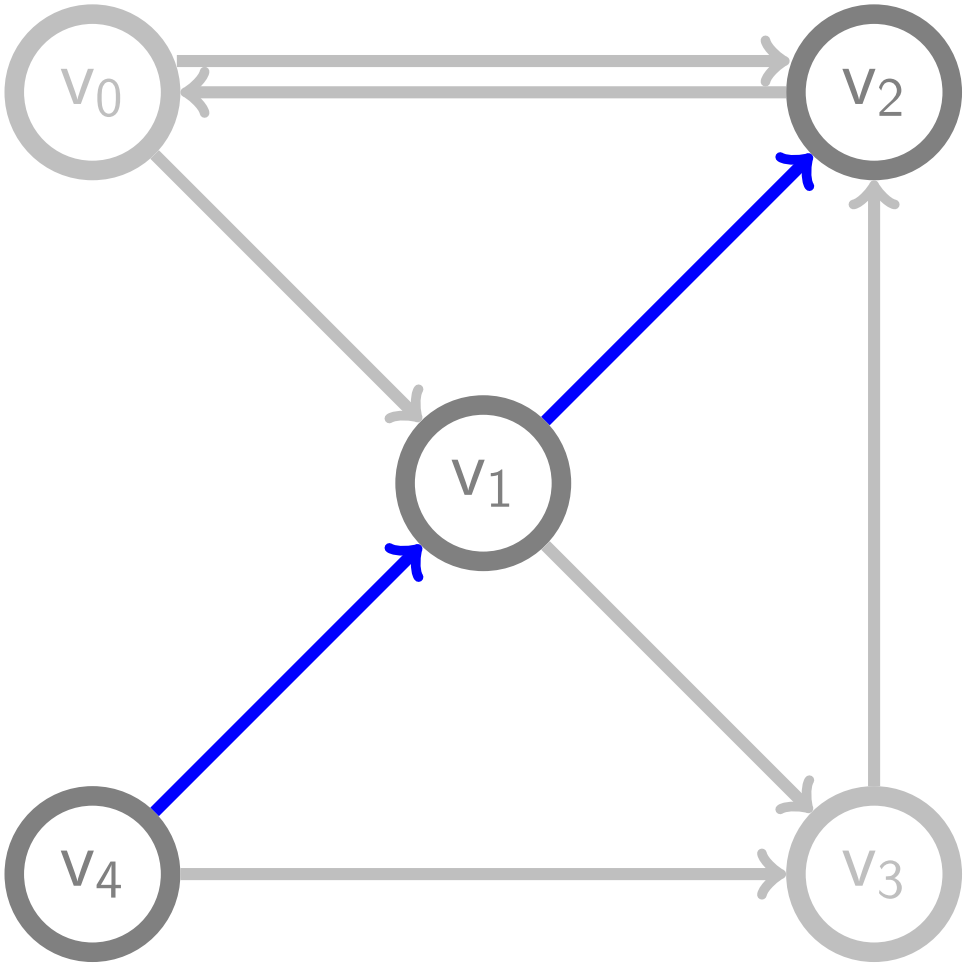
Início $v4$, destino $v3$

Encontrar uma Solução para um Labirinto



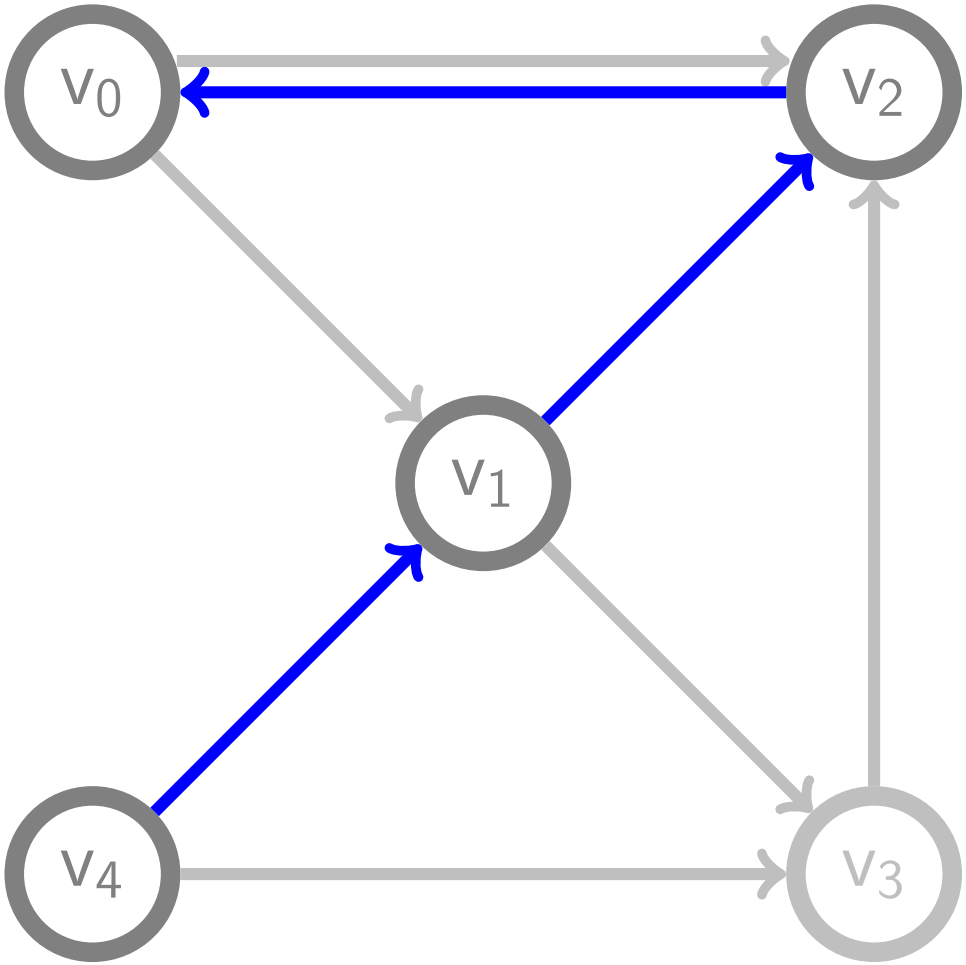
Início $v4$, destino $v3$

Encontrar uma Solução para um Labirinto



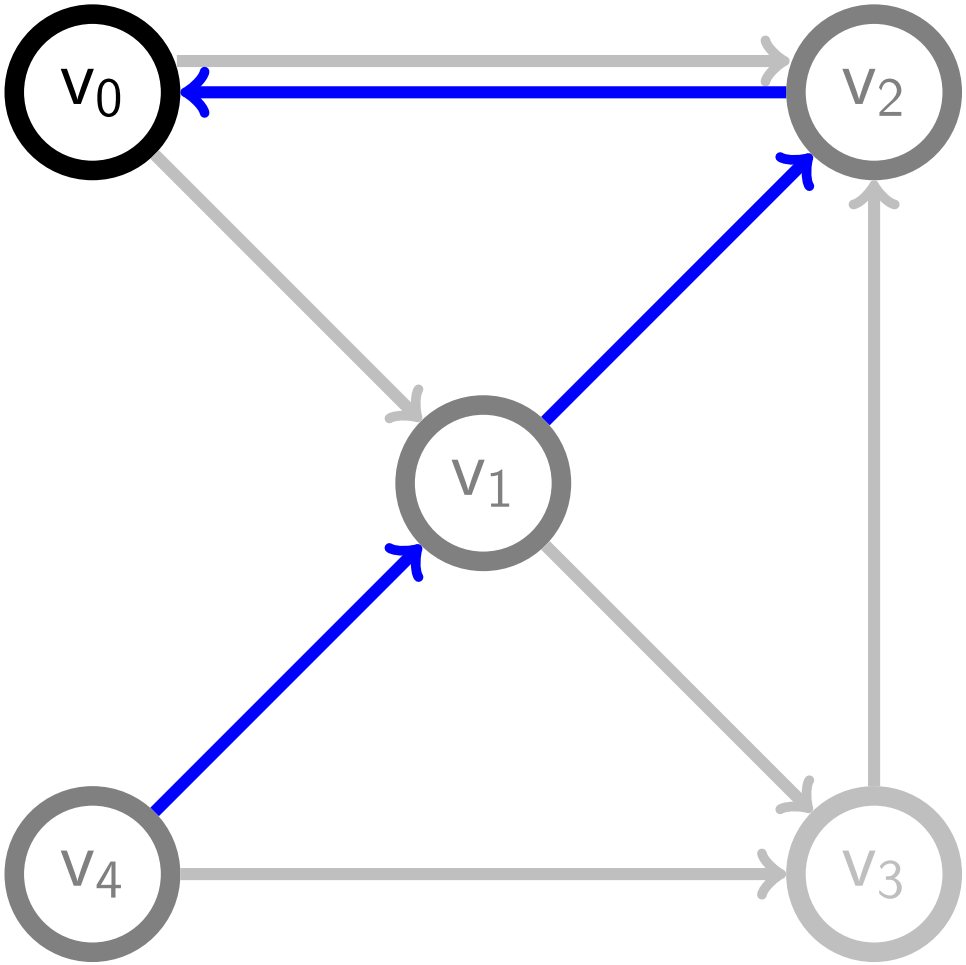
Início $v4$, destino $v3$

Encontrar uma Solução para um Labirinto



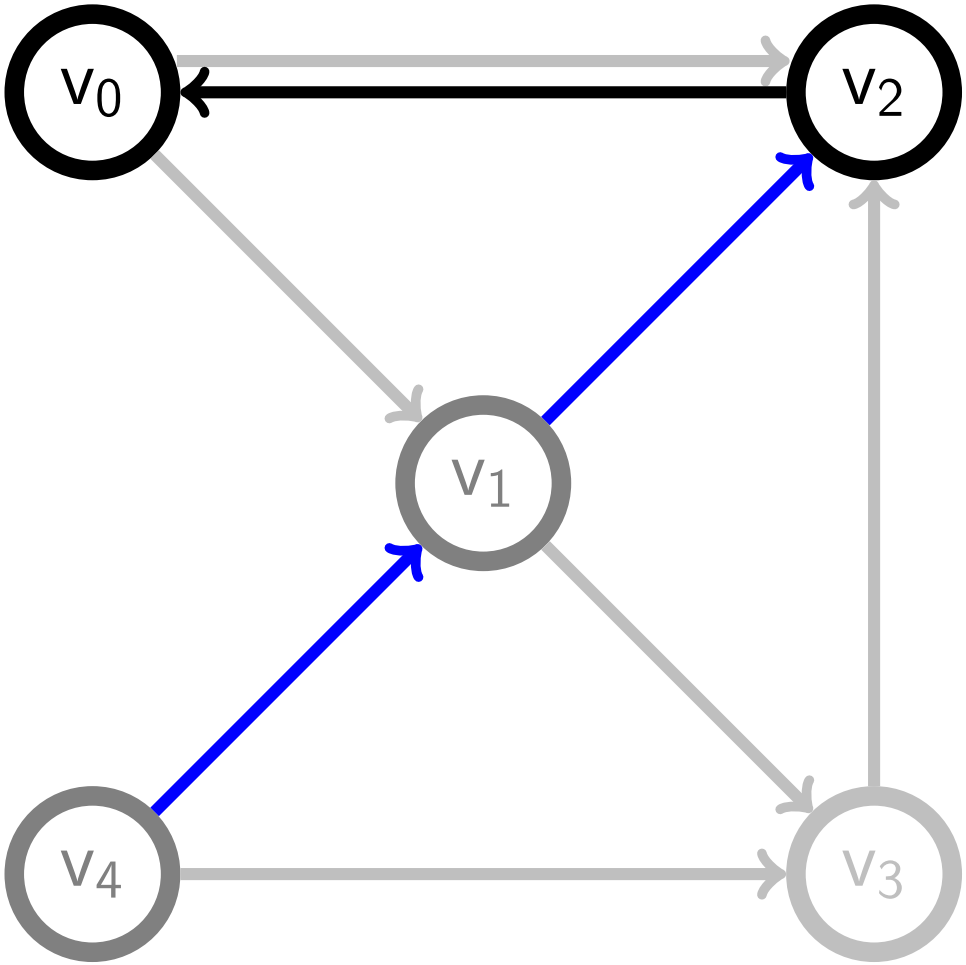
Início $v4$, destino $v3$

Encontrar uma Solução para um Labirinto



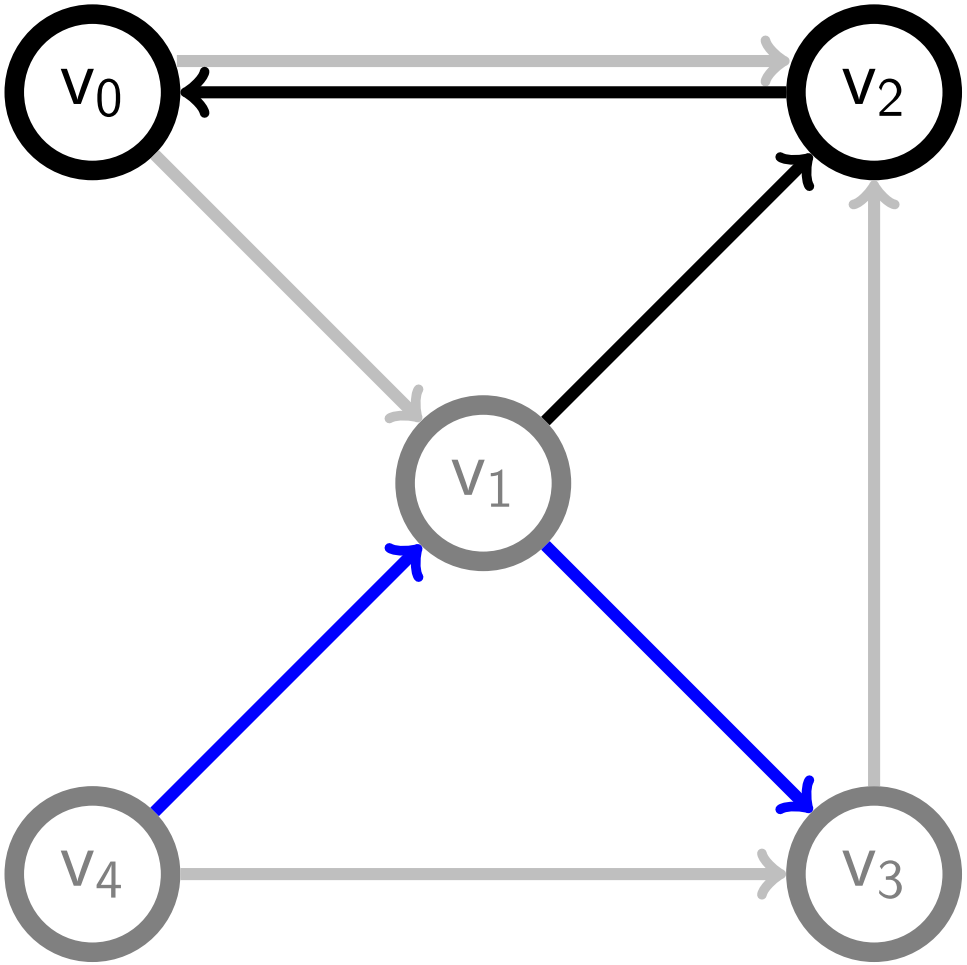
Início $v4$, destino $v3$

Encontrar uma Solução para um Labirinto



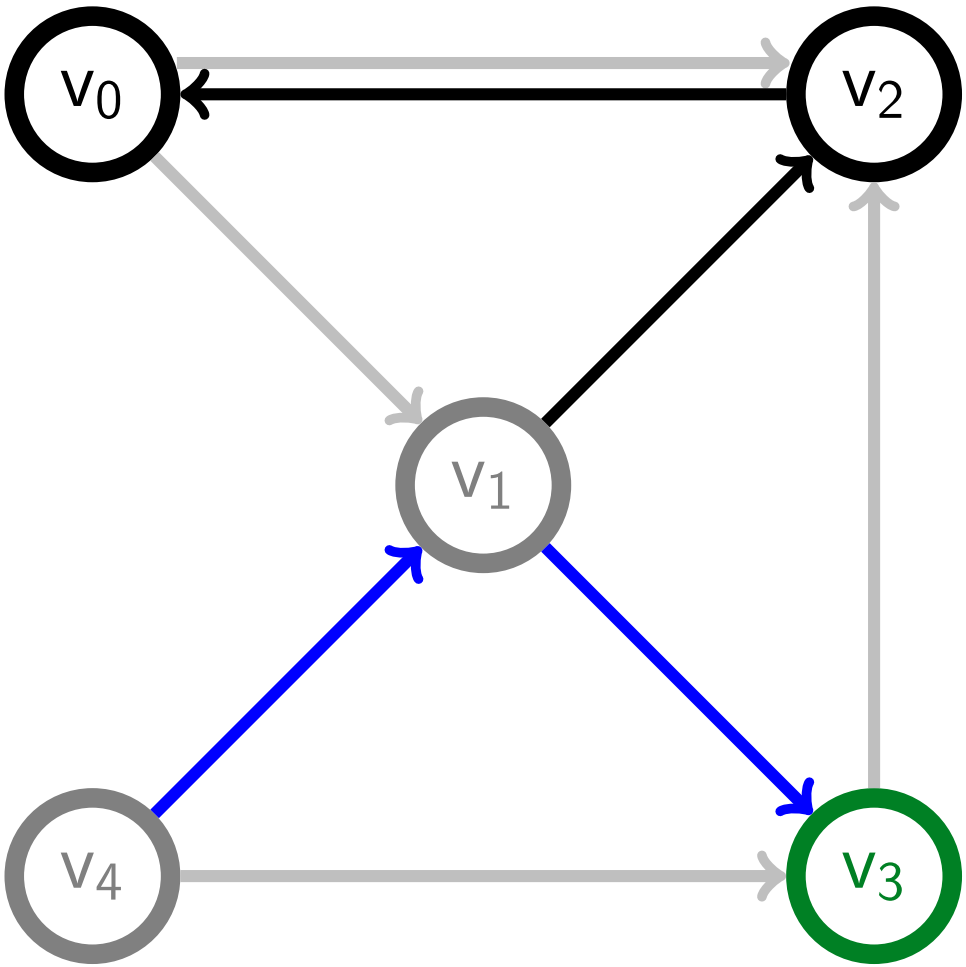
Início $v4$, destino $v3$

Encontrar uma Solução para um Labirinto



Início $v4$, destino $v3$

Encontrar uma Solução para um Labirinto

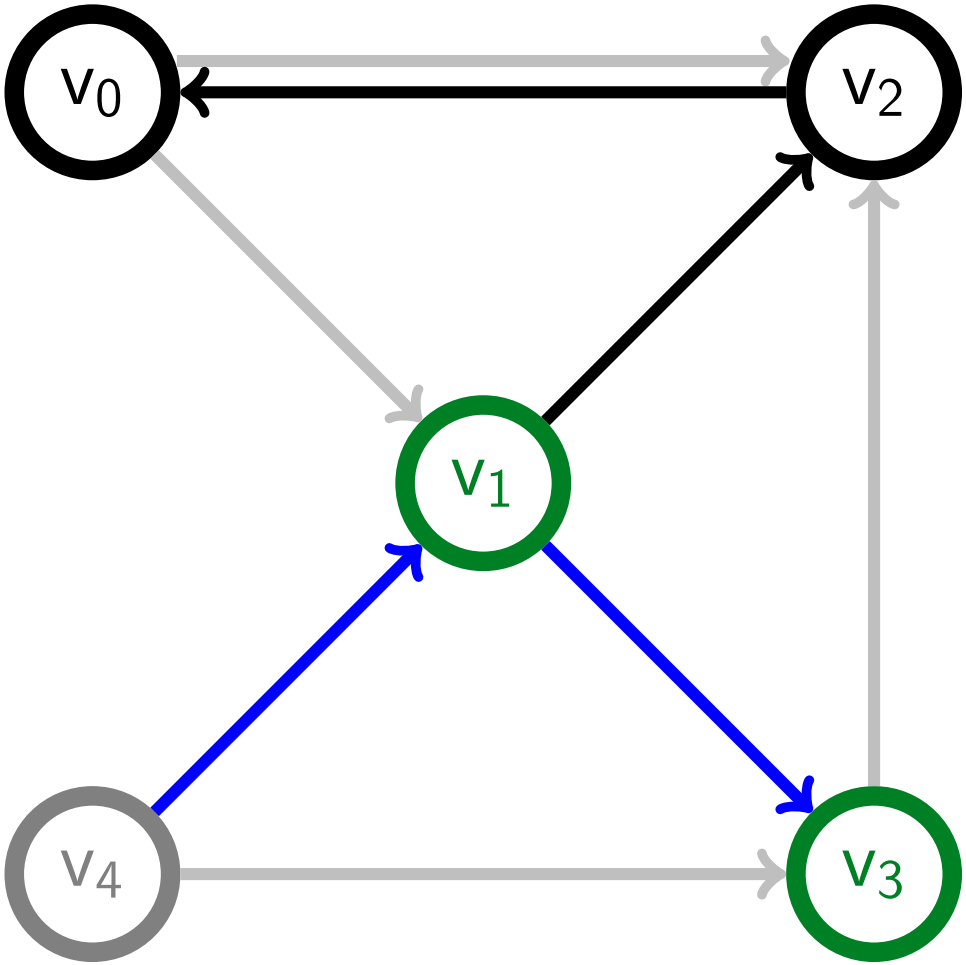


Início $v4$, destino $v3$

Destino encontrado!

```
Lista do caminho:
  v3
```

Encontrar uma Solução para um Labirinto

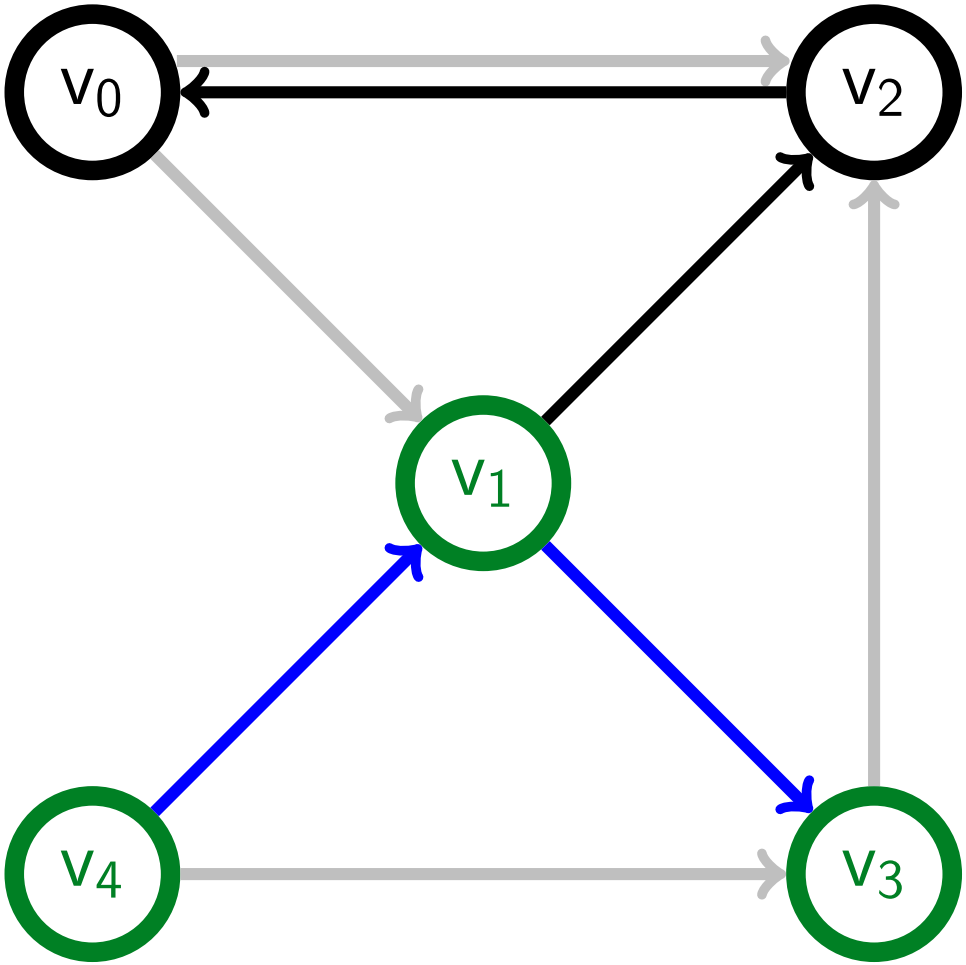


Início $v4$, destino $v3$

Destino encontrado!

```
Lista do caminho:
 v1 -> v3
```

Encontrar uma Solução para um Labirinto



Início $v4$, destino $v3$

Destino encontrado!

```
Lista do caminho:
v4 -> v1 -> v3
```

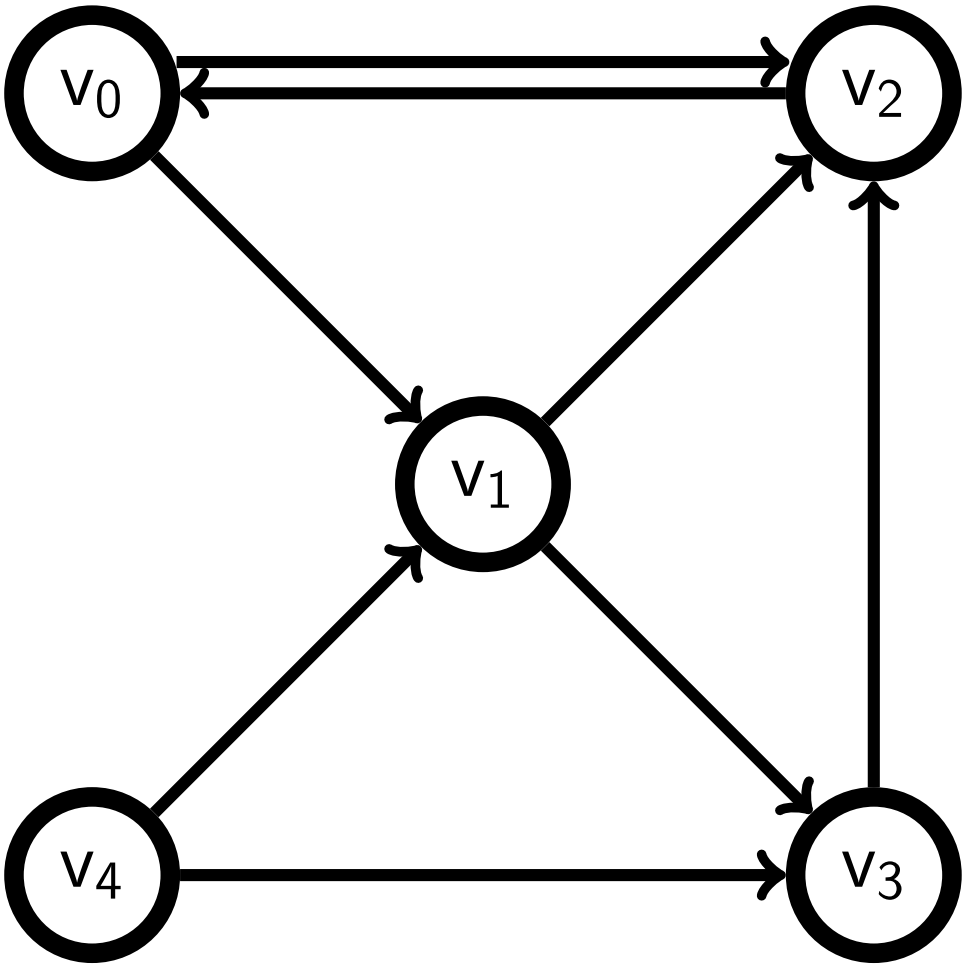
Grafos – Busca em Profundidade

Aplicações da Busca em Profundidade?

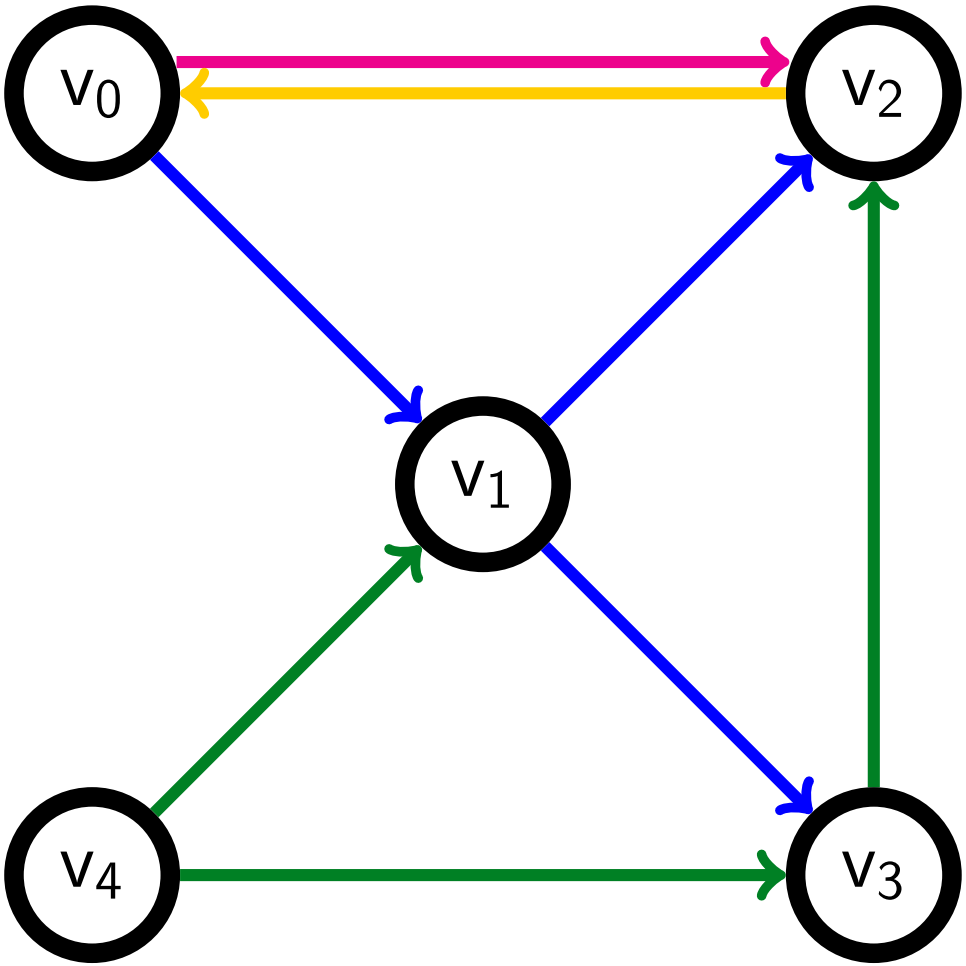
A Busca em Profundidade por ser utilizada (ou adaptada) para ajudar a resolver uma série de problemas:

- Encontrar soluções para labirintos
- **Verificar se um grafo é acíclico**
- Realizar a ordenação topológica
- Encontrar componentes conexos/conectados
- Encontrar componentes fortemente conexos/conectados
- ...

Grafos Cíclicos x Acíclicos



Grafos Cíclicos x Acíclicos



Grafos acíclicos
não possuem
arestas de retorno.

Resumo da Busca em Profundidade:

Nó	Anterior	Descoberta	Término	Cor:
0	-1	1	8	2
1	0	2	7	2
2	1	3	4	2
3	1	5	6	2
4	-1	9	10	2

Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Grafos Cíclicos x Acíclicos

Como sabemos, durante a Busca em Profundidade, que uma aresta é de retorno?

[1] Se o vértice for PRETO, então teremos uma aresta de avanço ou de cruzamento.

Grafos Cíclicos x Acíclicos

Como sabemos, durante a Busca em Profundidade, que uma aresta é de retorno?

Em **Grafos Dirigidos**, uma aresta de retorno é uma **aresta que liga o vértice atual a um vértice CINZA**[1] (vértice já visitado, mas cujo processamento ainda não está concluído).

[1] Se o vértice for PRETO, então teremos uma aresta de avanço ou de cruzamento.

Grafos Cíclicos x Acíclicos

Como sabemos, durante a Busca em Profundidade, que uma aresta é de retorno?

Em **Grafos Dirigidos**, uma aresta de retorno é uma **aresta que liga o vértice atual a um vértice CINZA**[1] (vértice já visitado, mas cujo processamento ainda não está concluído).

Grafos Dirigidos Acíclicos são chamados de **DAG**s (*Directed Acyclic Graphs*).

[1] Se o vértice for PRETO, então teremos uma aresta de avanço ou de cruzamento.

Busca em Profundidade - Verificação de Ciclos - Digrafo

```
void visitaDFSCores(Grafo* g, int atual, int* tempo, int* cor,
                    int* tDescoberta, int* tTermino, int* anterior){
  (*tempo)++;
  cor[atual] = 1; // CINZA
  tDescoberta[atual] = *tempo;
  int w;
  ElemLista* end = g->A[atual];
  while (end){
    w = end->vertice;
    if(cor[w]==0){ // BRANCO
      anterior[w] = atual;
      visitaDFSCores(g,w,tempo,cor,tDescoberta,tTermino,anterior);
    } else if( cor[w]==1 ) printf("Este grafo não é acíclico.\n");
    end = end->prox;
  }
  cor[atual] = 2; // PRETO
  (*tempo)++;
  tTermino[atual] = *tempo;
}
```

$O(V + E)$

Grafos Cíclicos x Acíclicos

Como sabemos, durante a Busca em Profundidade, que uma aresta é de retorno?

Em **Grafos Não Dirigidos**, uma aresta de retorno é uma **aresta que liga o vértice atual a um vértice CINZA** *desde que esta aresta não tenha sido usada para chegar no vértice atual*[2].

[2] Lembre-se que em grafos não dirigidos poderíamos caminhar nas duas direções da aresta, mas só devemos usar cada aresta uma única vez.

Grafos Cíclicos x Acíclicos

Como sabemos, durante a Busca em Profundidade, que uma aresta é de retorno?

Em **Grafos Não Dirigidos**, uma aresta de retorno é uma **aresta que liga o vértice atual a um vértice CINZA** *desde que esta aresta não tenha sido usada para chegar no vértice atual*[2].

Em Grafos Não Dirigidos a Busca em Profundidade produz apenas arestas de árvore e de retorno.

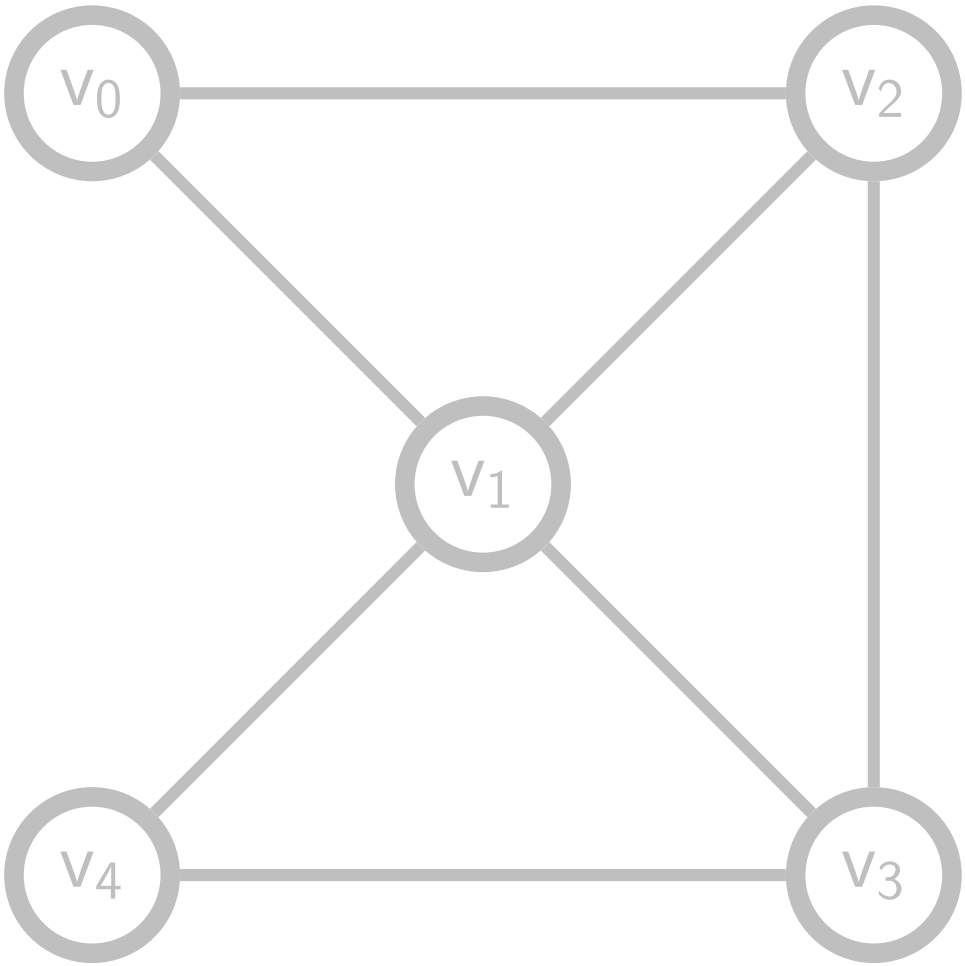
[2]Lembre-se que em grafos não dirigidos poderíamos caminhar nas duas direções da aresta, mas só devemos usar cada aresta uma única vez.

Busca em Profundidade - Verificação de Ciclos - Grafo

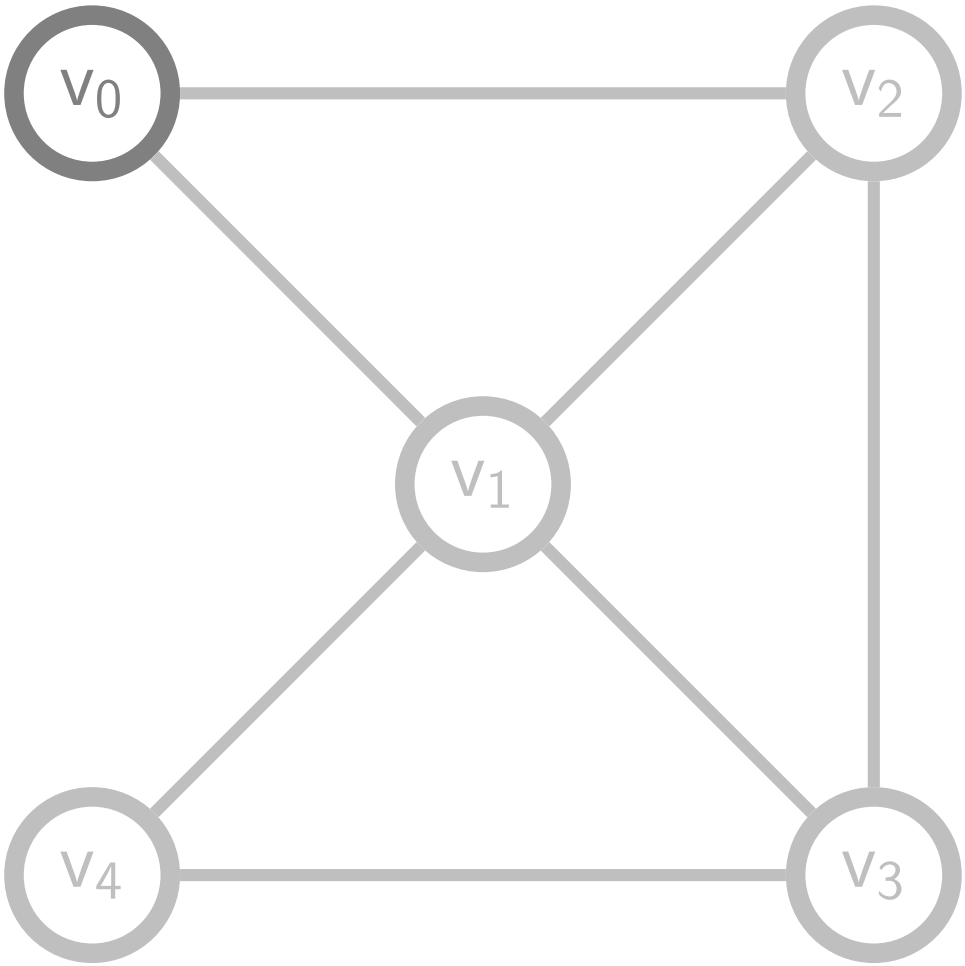
```
void visitaDFSCores(Grafo* g, int atual, int* tempo, int* cor,
                    int* tDescoberta, int* tTermino, int* anterior){
  (*tempo)++;
  cor[atual] = 1; // CINZA
  tDescoberta[atual] = *tempo;
  int w;
  ElemLista* end = g->A[atual];
  while (end){
    w = end->vertice;
    if(cor[w]==0){ // BRANCO
      anterior[w] = atual;
      visitaDFSCores(g,w,tempo,cor,tDescoberta,tTermino,anterior);
    } else if(anterior[atual] != w)
              printf("Este grafo não é acíclico.\n");
    end = end->prox; }
  cor[atual] = 2; // PRETO
  (*tempo)++;
  tTermino[atual] = *tempo;
}
```

$$O(V + E)$$

Grafo Não Dirigido



Grafo Não Dirigido



Grafos com ciclos

possuem arestas de retorno.

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        -1         1       10   2
 1         0         2        9   2
 2         1         3        8   2
 3         2         4        7   2
 4         3         5        6   2
```

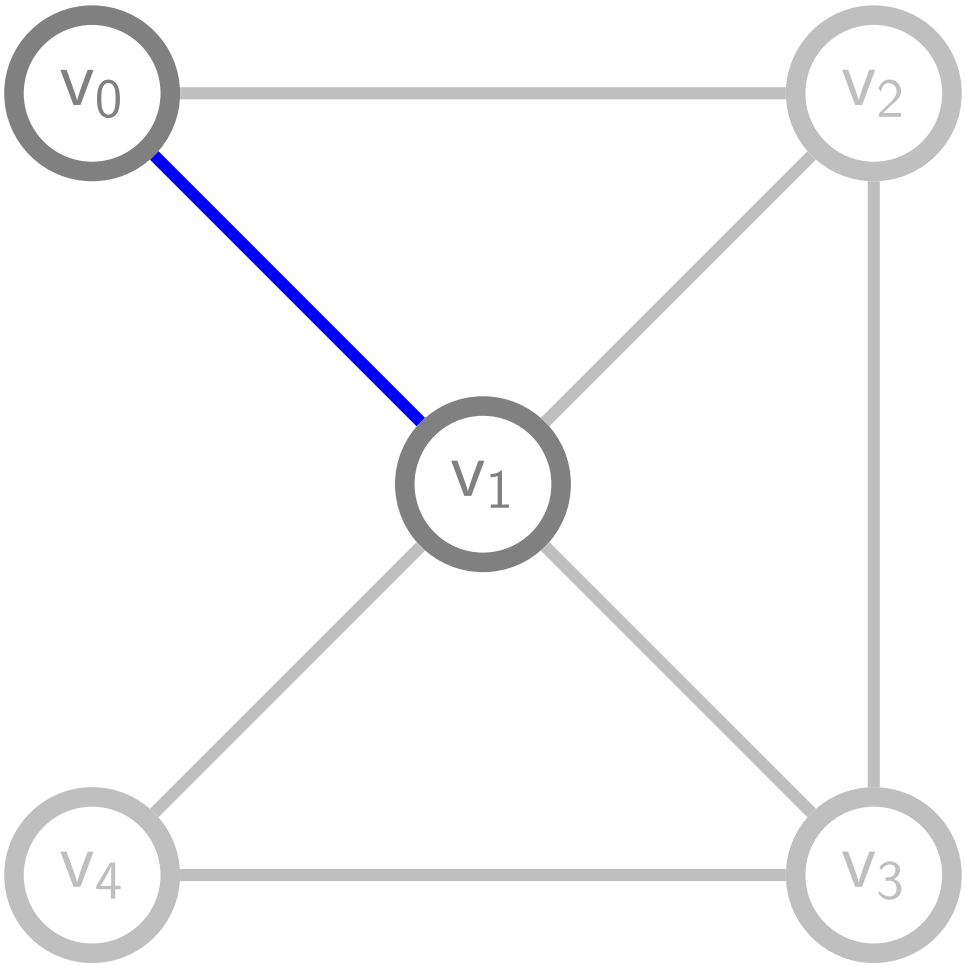
Arestas de Árvore

Arestas de Retorno

Arestas de Cruzamento

Arestas de Avanço

Grafo Não Dirigido



Grafos com ciclos

possuem arestas de retorno.

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        -1          1       10   2
 1         0          2        9   2
 2         1          3        8   2
 3         2          4        7   2
 4         3          5        6   2
```

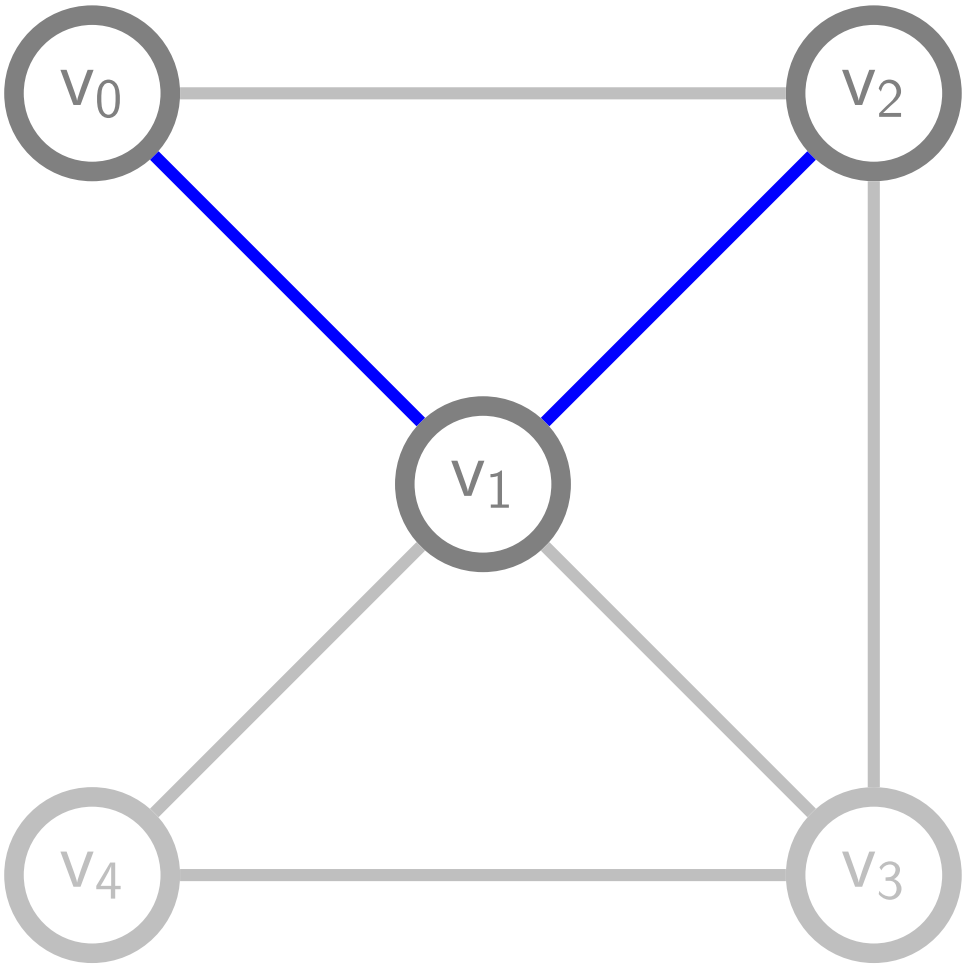
Arestas de Árvore

Arestas de Retorno

Arestas de Cruzamento

Arestas de Avanço

Grafo Não Dirigido



Grafos com ciclos

possuem arestas de retorno.

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        -1          1       10   2
 1         0          2        9   2
 2         1          3        8   2
 3         2          4        7   2
 4         3          5        6   2
```

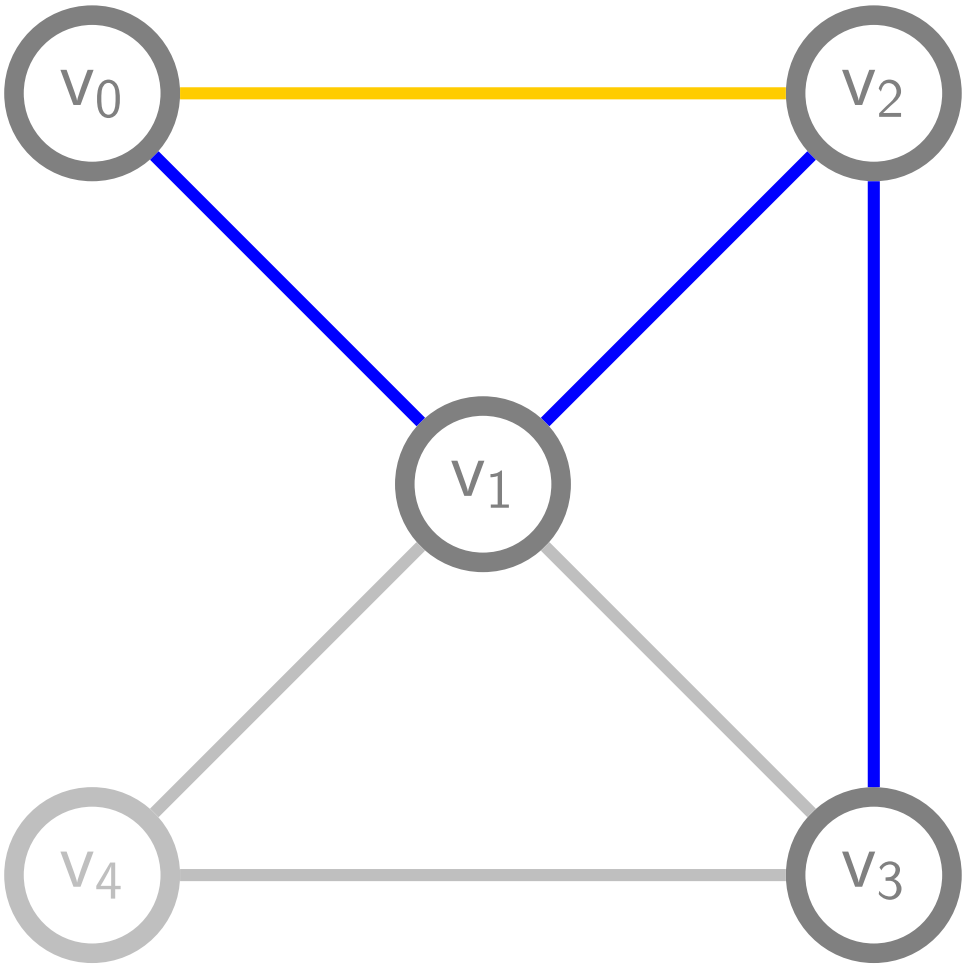
Arestas de Árvore

Arestas de Retorno

Arestas de Cruzamento

Arestas de Avanço

Grafo Não Dirigido

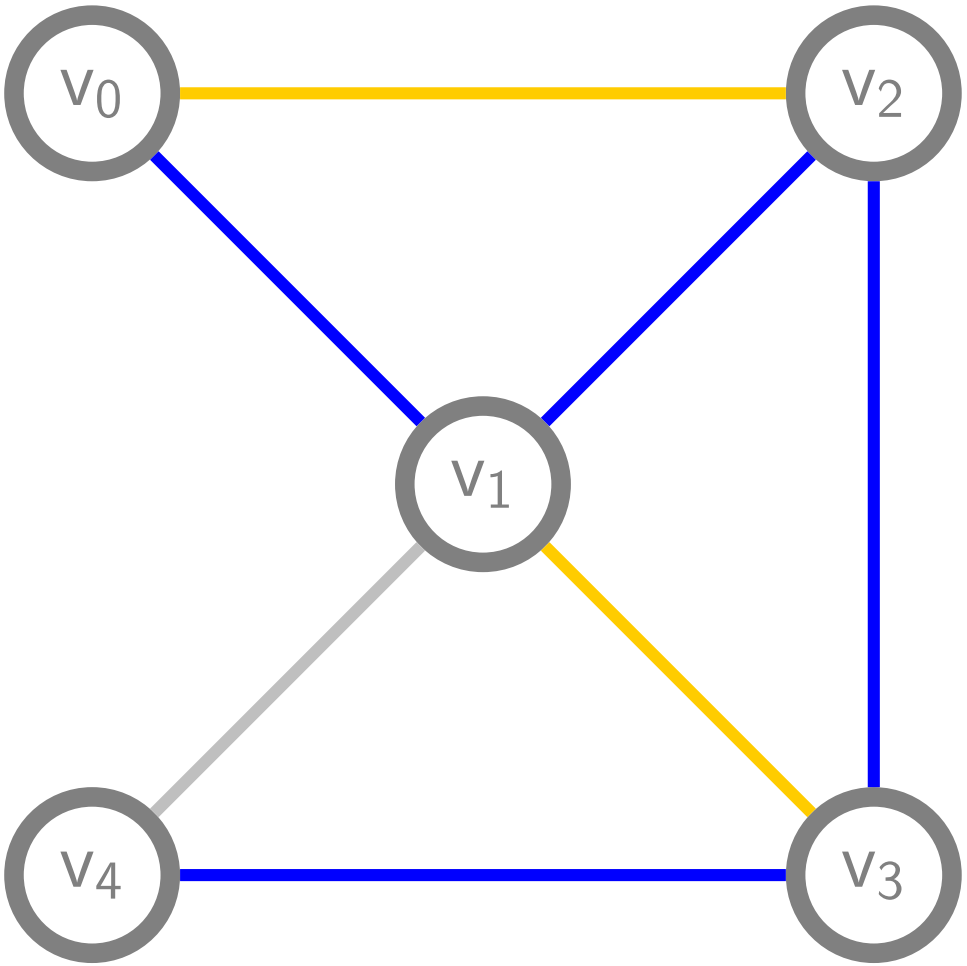


Grafos com ciclos possuem arestas de retorno.

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        -1         1      10   2
 1         0         2       9   2
 2         1         3       8   2
 3         2         4       7   2
 4         3         5       6   2
```

Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Grafo Não Dirigido



Grafos com ciclos

possuem arestas de retorno.

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        -1         1       10   2
 1         0         2        9   2
 2         1         3        8   2
 3         2         4        7   2
 4         3         5        6   2
```

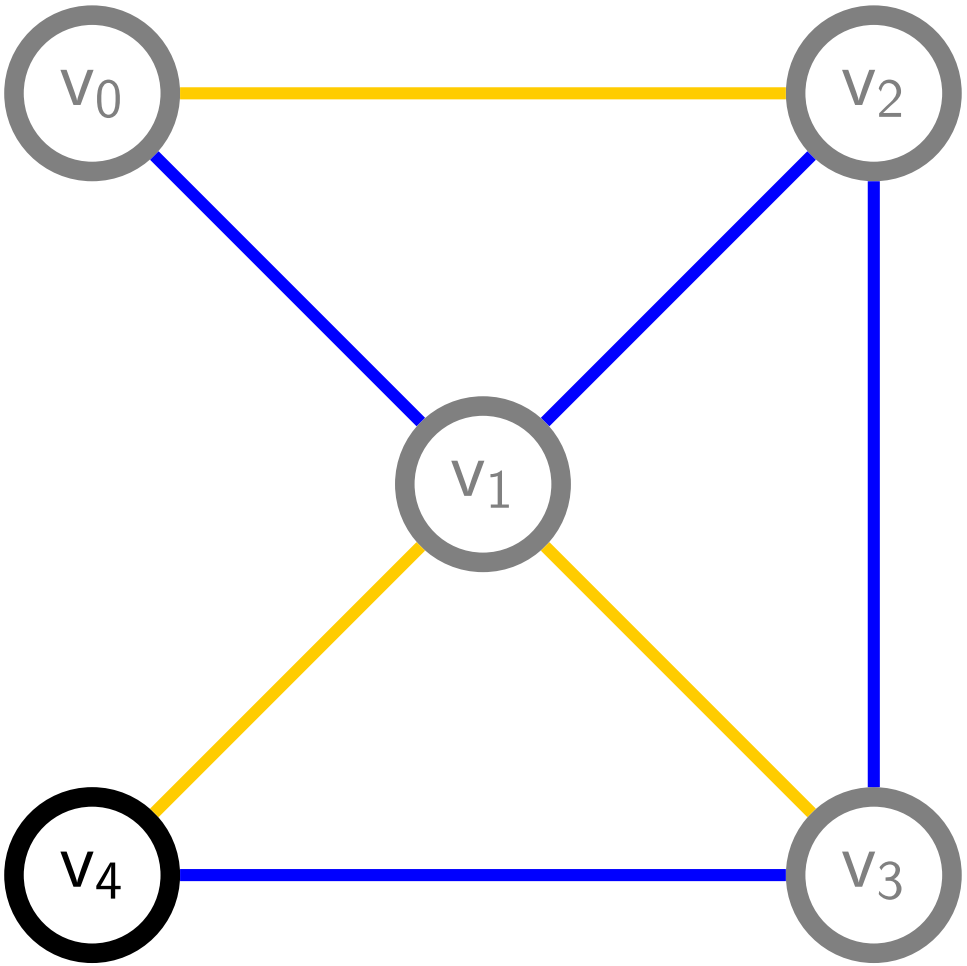
Arestas de Árvore

Arestas de Retorno

Arestas de Cruzamento

Arestas de Avanço

Grafo Não Dirigido

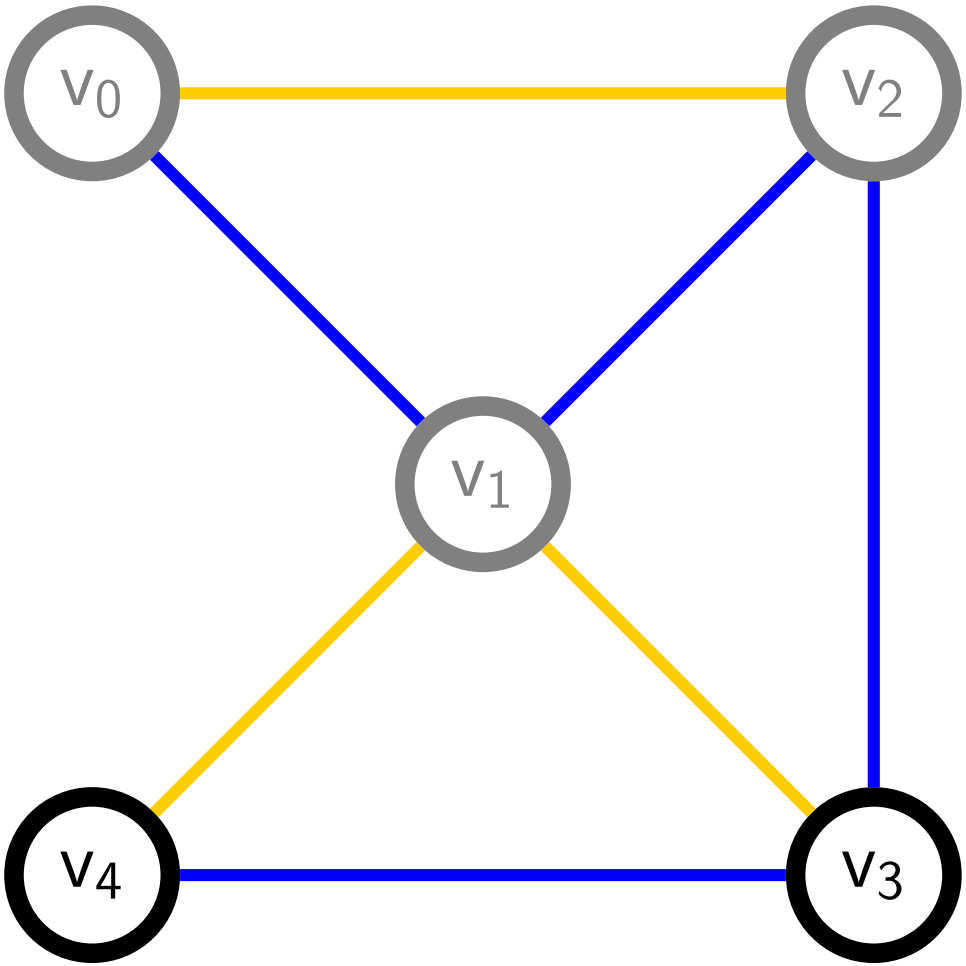


Grafos com ciclos possuem arestas de retorno.

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        -1         1      10   2
 1         0         2       9   2
 2         1         3       8   2
 3         2         4       7   2
 4         3         5       6   2
```

Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Grafo Não Dirigido



Grafos com ciclos

possuem arestas de retorno.

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        -1          1       10   2
 1         0          2        9   2
 2         1          3        8   2
 3         2          4        7   2
 4         3          5        6   2
```

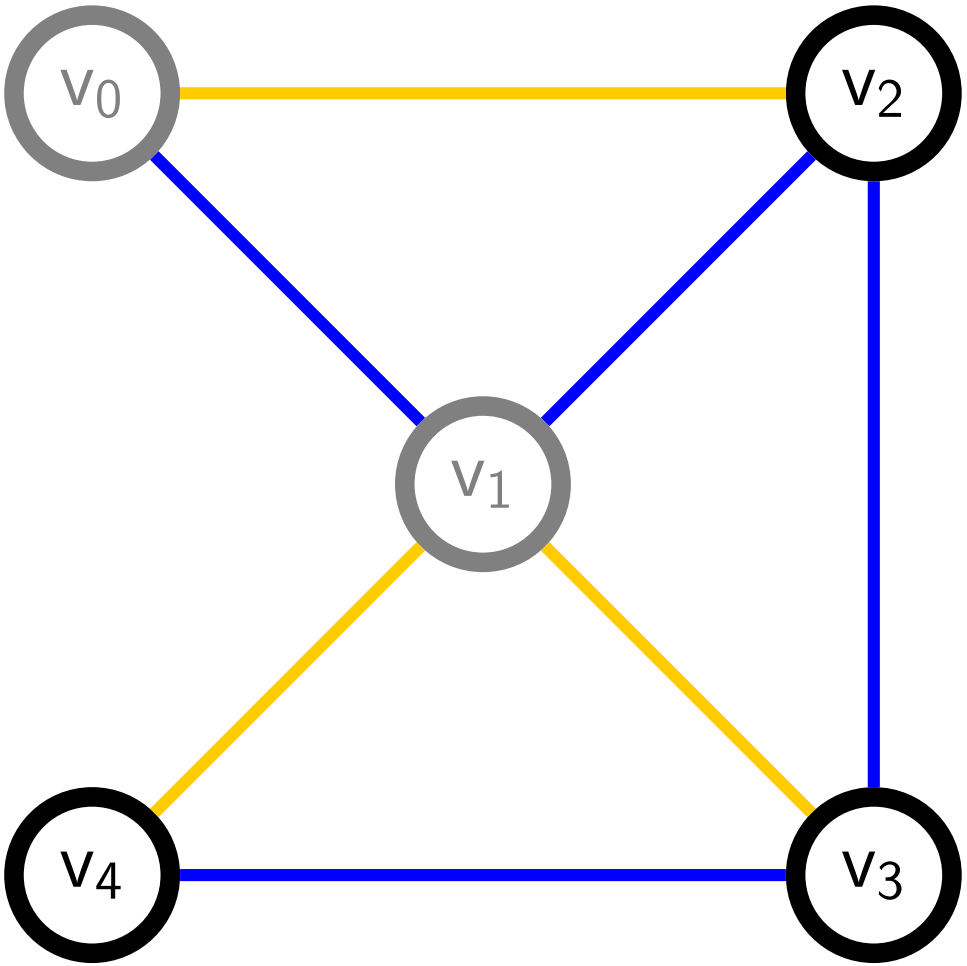
Arestas de Árvore

Arestas de Retorno

Arestas de Cruzamento

Arestas de Avanço

Grafo Não Dirigido



Grafos com ciclos

possuem arestas de retorno.

```
Resumo da Busca em Profundidade:
No Anterior Descoberta Termino Cor:
 0        -1          1       10   2
 1         0          2        9   2
 2         1          3        8   2
 3         2          4        7   2
 4         3          5        6   2
```

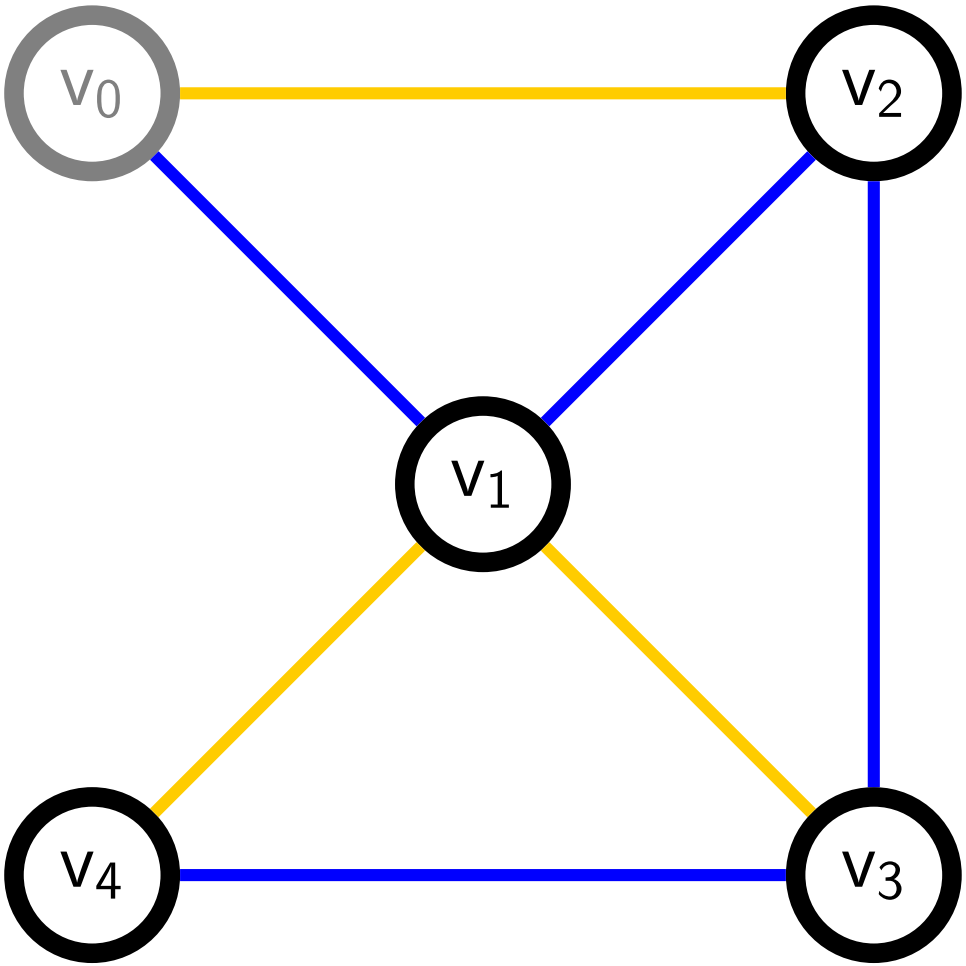
Arestas de Árvore

Arestas de Retorno

Arestas de Cruzamento

Arestas de Avanço

Grafo Não Dirigido



Grafos com ciclos

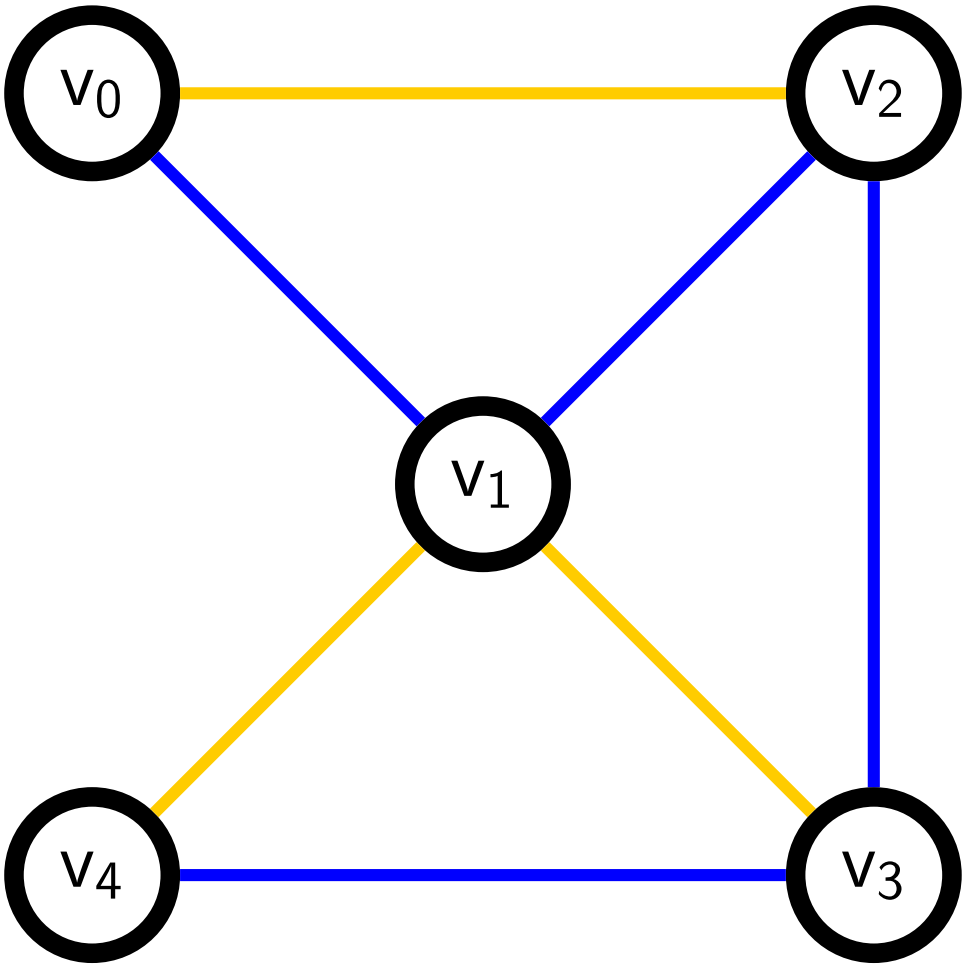
possuem arestas de retorno.

Resumo da Busca em Profundidade:

No	Anterior	Descoberta	Termino	Cor:
0	-1	1	10	2
1	0	2	9	2
2	1	3	8	2
3	2	4	7	2
4	3	5	6	2

Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Grafo Não Dirigido



Grafos com ciclos

possuem arestas de retorno.

Resumo da Busca em Profundidade:

No	Anterior	Descoberta	Termino	Cor:
0	-1	1	10	2
1	0	2	9	2
2	1	3	8	2
3	2	4	7	2
4	3	5	6	2

Arestas de Árvore
Arestas de Retorno
Arestas de Cruzamento
Arestas de Avanço

Algoritmos e Estruturas de Dados II
Aula 07a – Grafos: Busca em Profundidade - Aplicações

Prof. Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri