Técnicas para Programação Competitiva

Algoritmos Envolvendo Grafos

Prof. Andrei Braga

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

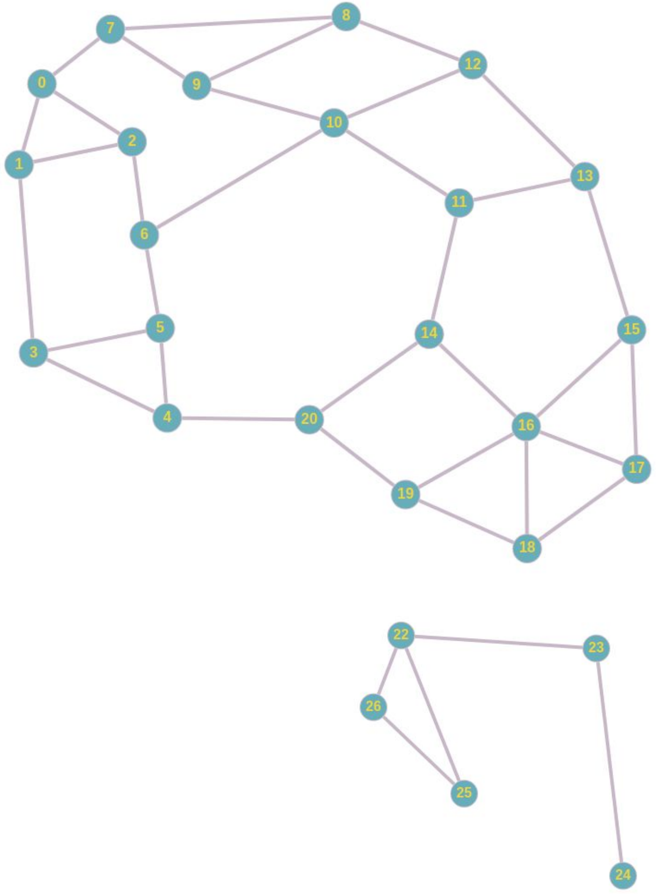
Conteúdo

Conteúdo

- **Grafos - Conceitos básicos**
- Representação computacional
- Busca em profundidade
- Busca em largura
- Grafos dirigidos - Conceitos básicos
- Grafos dirigidos - Representação computacional
- Grafos dirigidos - Busca em profundidade e em largura
- Referências

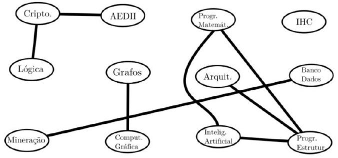
Motivação

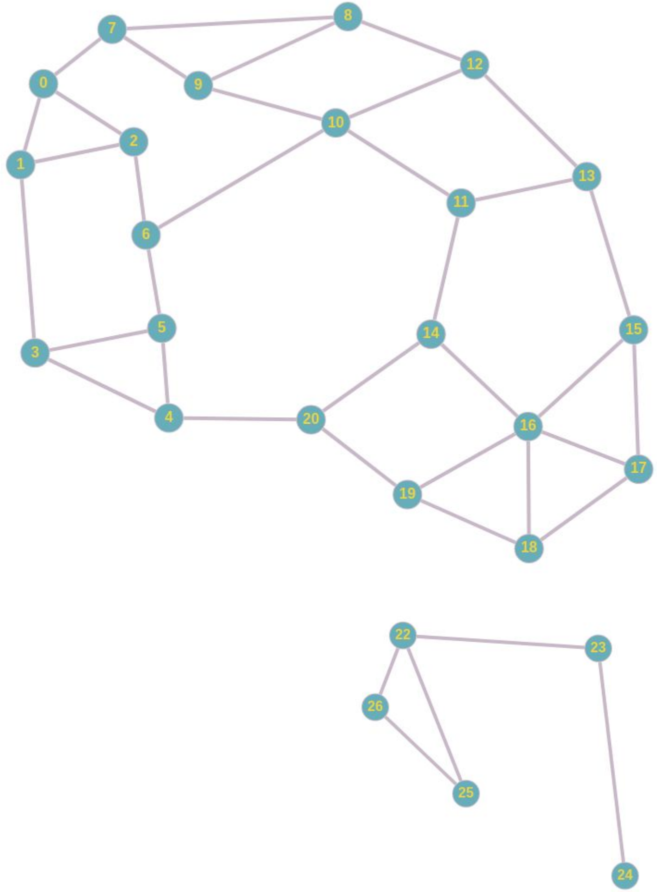
- Muitas aplicações computacionais envolvem
  - Itens (dados ou conjuntos de dados)
  - Conexões entre os itens
- Para modelar situações como estas, usamos uma estrutura matemática (ou uma estrutura de dados) chamada de **grafos**



Motivação

- Exemplos de aplicações:
  - Problemas de roteamento
  - Estudo de redes sociais
  - Problemas de topologia em redes
  - Problemas de alocação

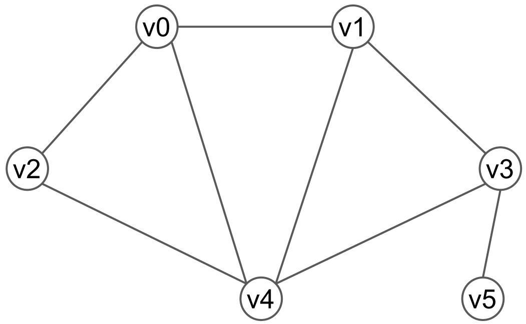




Grafo

- Um **grafo** G é um par ordenado (V, E) composto por
  - um conjunto de **vértices** V e
  - um conjunto de **arestas** E, sendo cada aresta um conjunto $\{ v_i , v_j \}$ de dois vértices de G
    - note que $\{ v_i , v_j \} = \{ v_j , v_i \}$, ou seja, não consideramos uma direção para a aresta

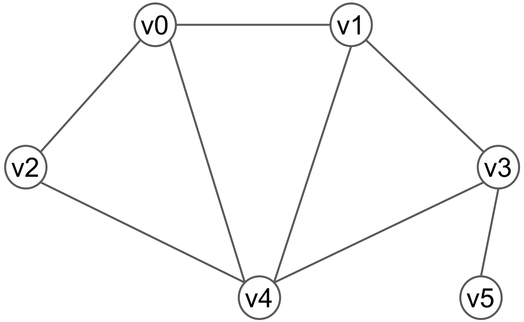
- Exemplo:
  - $G = (V, E)$, onde
    - $V = \{ v_0, v_1, v_2, v_3, v_4, v_5 \}$ e
    - $E = \{ \{ v_0, v_1 \}, \{ v_0, v_2 \}, \{ v_0, v_4 \},$
      $\{ v_1, v_3 \}, \{ v_1, v_4 \}, \{ v_2, v_4 \},$
      $\{ v_3, v_4 \}, \{ v_3, v_5 \} \}$

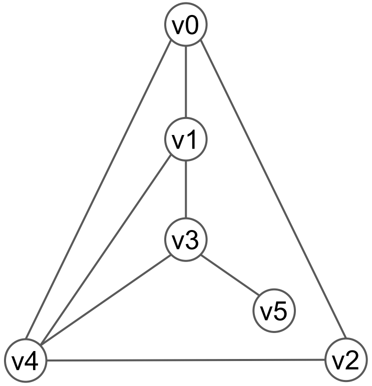


Desenho de um grafo

- Um **desenho** de um grafo é uma representação gráfica do grafo onde
  - **pontos** (ou **círculos**) representam os vértices do grafo e
  - **linhas** conectando os pontos (ou círculos) representam as arestas do grafo
- Um desenho nos dá uma intuição sobre a estrutura do grafo, mas devemos usar esta intuição com cautela, porque o grafo é definido independentemente das suas representações gráficas

- Exemplo:
  - $G = (V, E)$, onde
    - $V = \{ v_0, v_1, v_2, v_3, v_4, v_5 \}$ e
    - $E = \{ \{ v_0, v_1 \}, \{ v_0, v_2 \}, \{ v_0, v_4 \}, \{ v_1, v_3 \}, \{ v_1, v_4 \}, \{ v_2, v_4 \}, \{ v_3, v_4 \}, \{ v_3, v_5 \} \}$

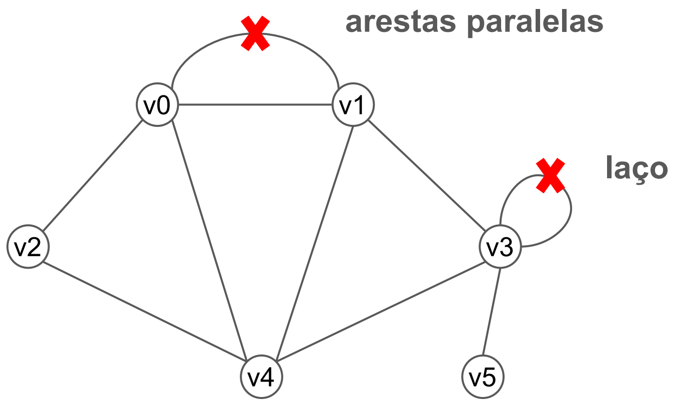




Grafo (simples)

- Em um grafo **simples**,
  - não podem existir duas ou mais arestas conectando um mesmo par de vértices e
  - não podem existir arestas que conectam um vértice a ele mesmo

- Exemplo:



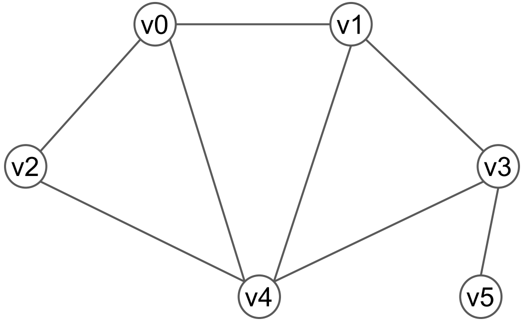
- A não ser que seja dito o contrário, os grafos que vamos considerar são simples

Vizinhança

- Por simplicidade, também denotamos uma aresta { v_i , v_j } como v_iv_j
- Dada uma aresta v_iv_j, os vértices v_i e v_j são os **extremos** desta aresta

- Se v_iv_j é uma aresta de um grafo G, então
  - os vértices v_i e v_j são **vizinhos** ou **adjacentes** em G,
  - v_i é **vizinho** de v_j em G (e vice-versa),
  - v_i é **adjacente** a v_j em G (e vice-versa) e
  - a aresta v_iv_j **incide** em v_i e incide em v_j

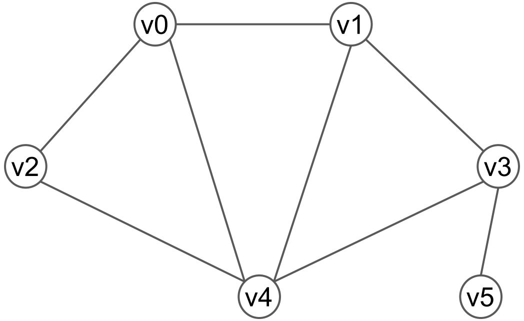
- Exemplo:
  - No grafo ao lado, v_0 é vizinho de (ou adjacente a) v_2 (e vice-versa), os vizinhos de v_3 são v_1, v_4 e v_5 e a aresta v_1v_4 incide em v_1 e em v_4



Passeio

- Um **passeio** em um grafo *G* é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em *G* do seu antecessor

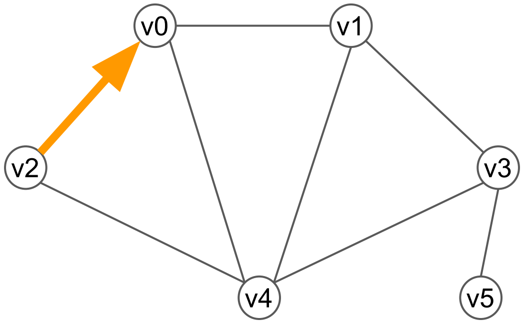
- Exemplo:
  - A sequência $v_2v_0v_4v_1v_0v_4v_3$ é um passeio no grafo ao lado



Passeio

- Um **passeio** em um grafo G é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de G tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em G do seu antecessor

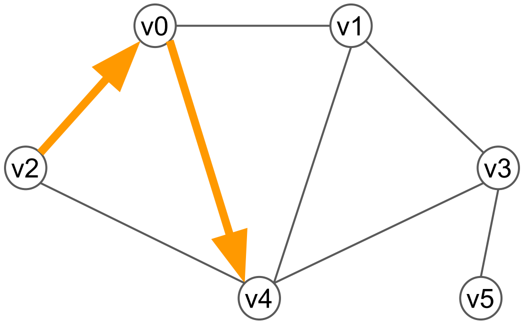
- Exemplo:
    - A sequência $v_2v_0v_4v_1v_0v_4v_3$ é um passeio no grafo ao lado



Passeio

- Um **passeio** em um grafo G é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de G tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em G do seu antecessor

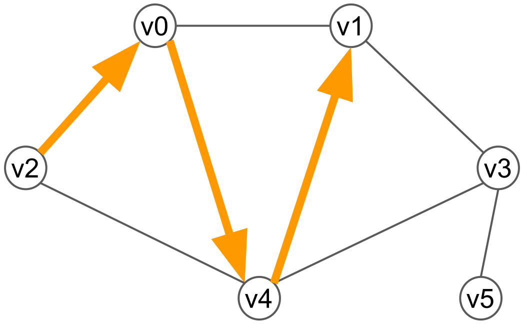
- Exemplo:
  - A sequência $v_2v_0v_4v_1v_0v_4v_3$ é um passeio no grafo ao lado



Passeio

- Um **passeio** em um grafo *G* é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em *G* do seu antecessor

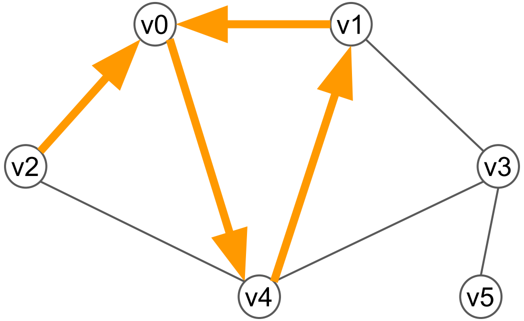
- Exemplo:
  - A sequência $v_2v_0v_4v_1v_0v_4v_3$ é um passeio no grafo ao lado



Passeio

- Um **passeio** em um grafo *G* é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em *G* do seu antecessor

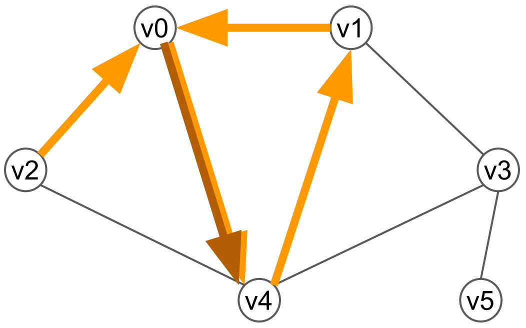
- Exemplo:
  - A sequência $v_2v_0v_4v_1v_0v_4v_3$ é um passeio no grafo ao lado



Passeio

- Um **passeio** em um grafo *G* é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em *G* do seu antecessor

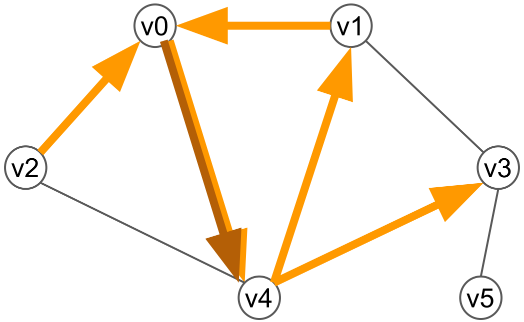
- Exemplo:
  - A sequência $v_2v_0v_4v_1v_0v_4v_3$ é um passeio no grafo ao lado



Passeio

- Um **passeio** em um grafo *G* é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em *G* do seu antecessor

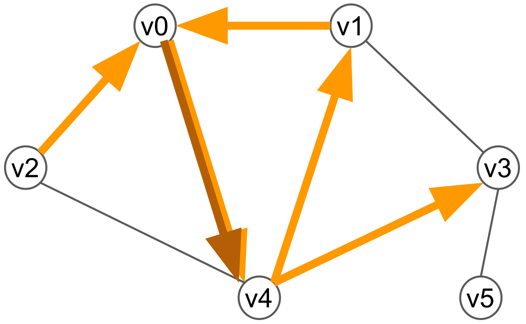
- Exemplo:
  - A sequência $v_2v_0v_4v_1v_0v_4v_3$ é um passeio no grafo ao lado



Passeio

- Um **passeio** em um grafo *G* é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em *G* do seu antecessor

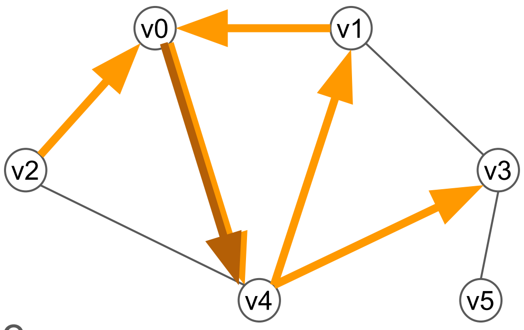
- Exemplo:
  - A sequência $v_2v_0v_4v_1v_0v_4v_3$ é um passeio no grafo ao lado



- Em um passeio, especificamos os vértices, mas as arestas envolvidas também estão implicitamente especificadas
- Por isso, podemos nos referir às **arestas de um passeio**

Passeio

- Um **passeio** em um grafo *G* é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho em *G* do seu antecessor

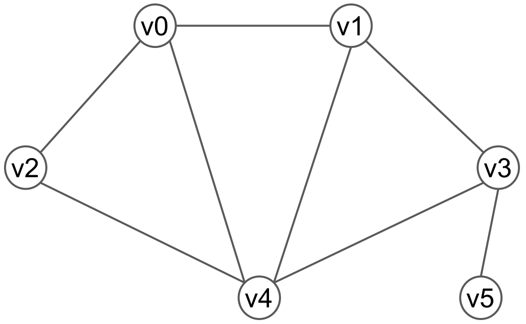


- Dado um passeio $v_{i0}v_{i1}\ldots v_{ik-1}v_{ik}$, dizemos que
  - v_{i0} e v_{ik} são os **extremos** do passeio;
  - $v_{i1}, \ldots, v_{ik-1}$ são os **vértices internos** do passeio;
  - o **comprimento** do passeio é *k*, ou seja, a quantidade de arestas percorridas e
  - o passeio é **fechado** se $v_{i0} = v_{ik}$ e é **aberto** caso contrário

Caminho

- Um **caminho** em um grafo *G* é um passeio em *G* onde não existem vértices repetidos

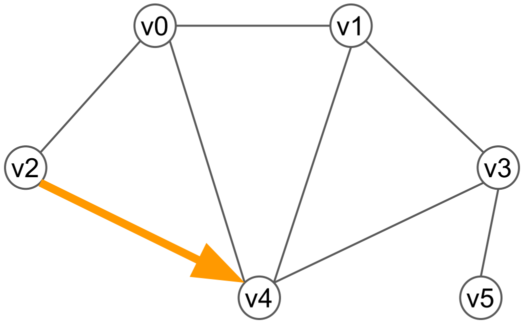
- Exemplo:
  - A sequência $v_2v_4v_0v_1v_3$ é um caminho no grafo ao lado



Caminho

- Um **caminho** em um grafo *G* é um passeio em *G* onde não existem vértices repetidos

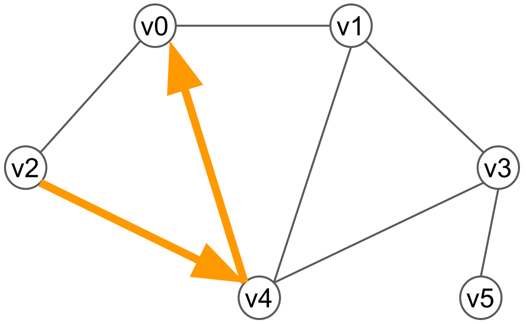
- Exemplo:
    - A sequência $v_2v_4v_0v_1v_3$ é um caminho no grafo ao lado



Caminho

- Um **caminho** em um grafo *G* é um passeio em *G* onde não existem vértices repetidos

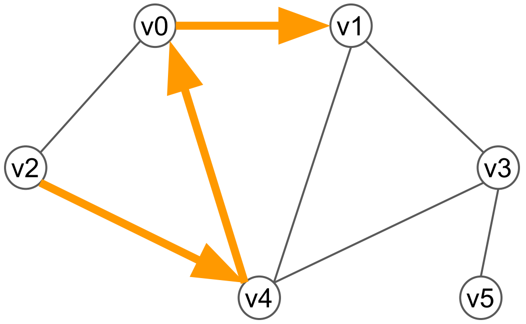
- Exemplo:
  - A sequência $v_2 v_4 v_0 v_1 v_3$ é um caminho no grafo ao lado



Caminho

- Um **caminho** em um grafo *G* é um passeio em *G* onde não existem vértices repetidos

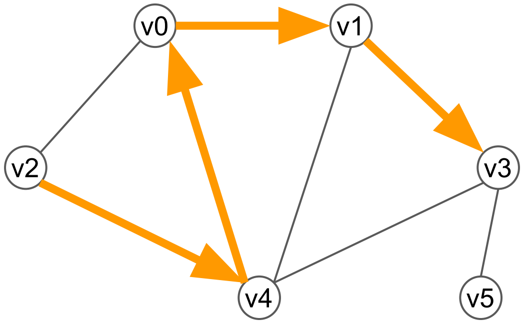
- Exemplo:
  - A sequência $v_2v_4v_0v_1v_3$ é um caminho no grafo ao lado



Caminho

- Um **caminho** em um grafo *G* é um passeio em *G* onde não existem vértices repetidos

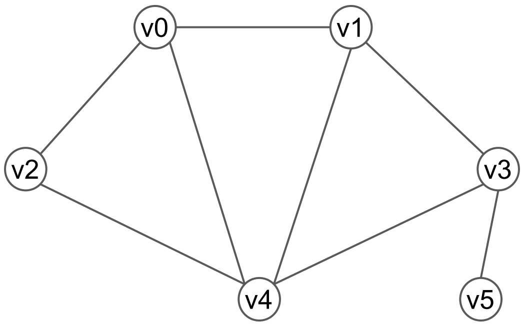
- Exemplo:
  - A sequência $v_2 v_4 v_0 v_1 v_3$ é um caminho no grafo ao lado



Ciclo

- Um **ciclo** em um grafo *G* é um passeio fechado em *G*, com comprimento maior ou igual a 3 e onde não existem vértices internos repetidos

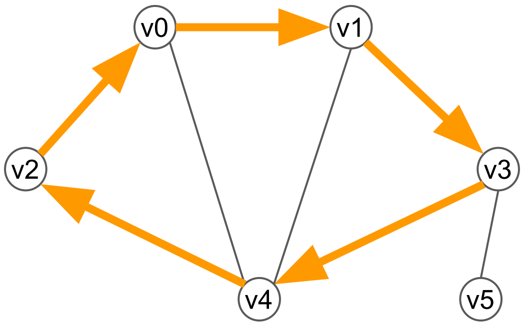
- Exemplo:
  - A sequência $v_2 v_0 v_1 v_3 v_4 v_2$ é um ciclo no grafo ao lado



Ciclo

- Um **ciclo** em um grafo *G* é um passeio fechado em *G*, com comprimento maior ou igual a 3 e onde não existem vértices internos repetidos

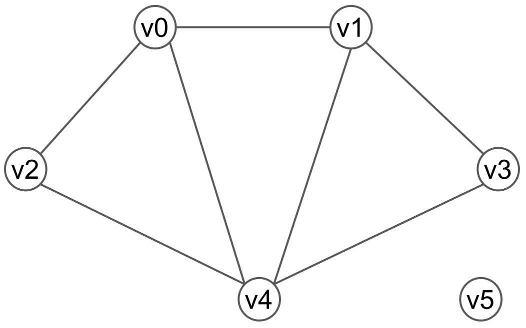
- Exemplo:
  - A sequência $v_2v_0v_1v_3v_4v_2$ é um ciclo no grafo ao lado



Distância

- A **distância** entre dois vértices v_i e v_j em G, denotada por $d(v_i , v_j)$ é
  - o menor comprimento de um caminho entre v_i e v_j em G ou
  - ∞ (infinita) caso não exista um caminho entre v_i e v_j em G

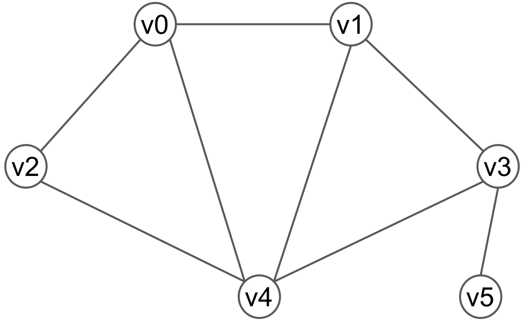
- Exemplo:
  - No grafo ao lado,
    - $d(v_2, v_3) = 2$,
    - $d(v_0, v_1) = 1$,
    - $d(v_4, v_4) = 0$ e
    - $d(v_1, v_5) = \infty$



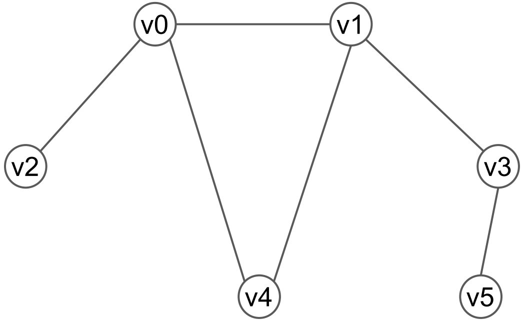
Subgrafo

- Um **subgrafo** de um grafo *G* é um grafo *H* tal que
  - $V(H) \subseteq V(G)$ e
  - $E(H) \subseteq E(G)$
- Exemplo:

G



H



- $V(G) = \{ v_0, v_1, v_2, v_3, v_4, v_5 \}$ e
- $E(G) = \{ \{ v_0, v_1 \}, \{ v_0, v_2 \}, \{ v_0, v_4 \}, \{ v_1, v_3 \}, \{ v_1, v_4 \}, \{ v_2, v_4 \}, \{ v_3, v_4 \}, \{ v_3, v_5 \} \}$

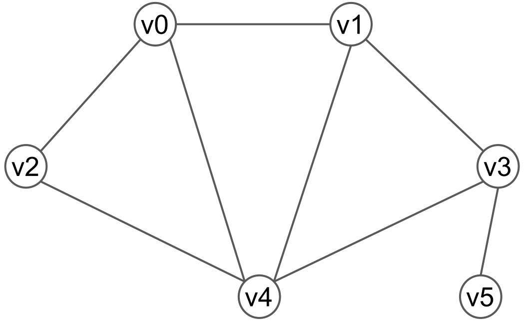
- $V(H) = \{ v_0, v_1, v_2, v_3, v_4, v_5 \}$ e
- $E(H) = \{ \{ v_0, v_1 \}, \{ v_0, v_2 \}, \{ v_0, v_4 \}, \{ v_1, v_3 \}, \{ v_1, v_4 \}, \{ v_3, v_5 \} \}$

Subgrafo

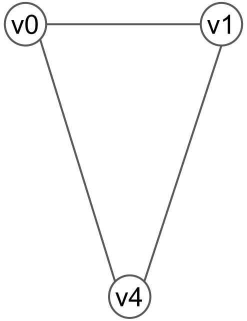
- Um **subgrafo** de um grafo *G* é um grafo *H* tal que
  - $V(H) \subseteq V(G)$ e
  - $E(H) \subseteq E(G)$

- Exemplo:

G



H



  - $V(G) = \{ v_0, v_1, v_2, v_3, v_4, v_5 \}$ e
  - $E(G) = \{ \{ v_0, v_1 \}, \{ v_0, v_2 \}, \{ v_0, v_4 \}, \{ v_1, v_3 \}, \{ v_1, v_4 \}, \{ v_2, v_4 \}, \{ v_3, v_4 \}, \{ v_3, v_5 \} \}$

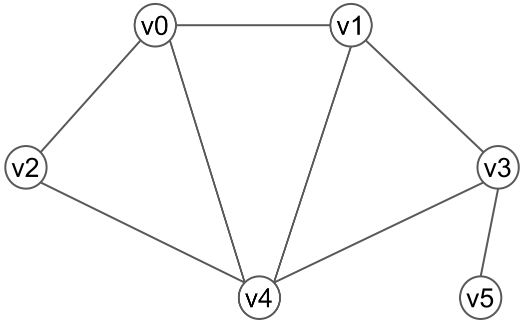
  - $V(H) = \{ v_0, v_1, v_4 \}$ e
  - $E(H) = \{ \{ v_0, v_1 \}, \{ v_0, v_4 \}, \{ v_1, v_4 \} \}$

Conexidade

- Um grafo *G* é **conexo** se, para todo par de vértices v_i , v_j de *G*, existe um caminho em *G* entre v_i e v_j (ou seja, um caminho em *G* cujos extremos são v_i e v_j); *G* é **desconexo** caso contrário

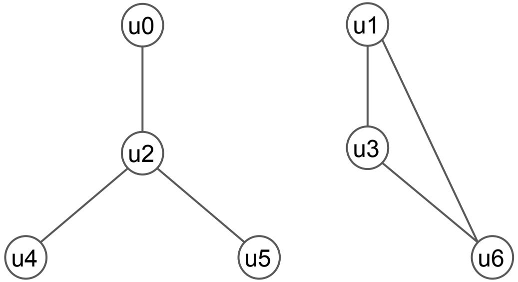
- Exemplo:

G_1



Grafo conexo

G_2



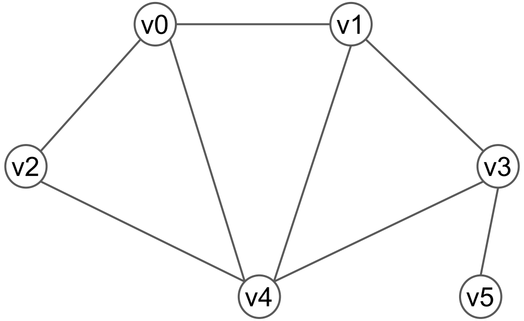
Grafo desconexo

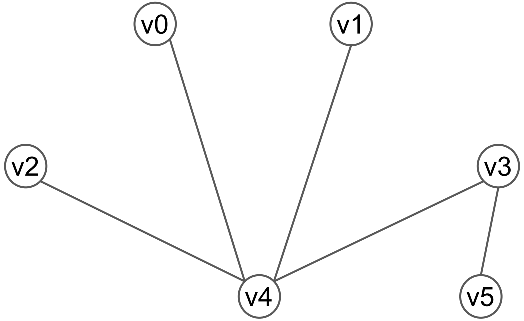
Subgrafo conexo

- Um **subgrafo conexo** de um grafo *G* é um subgrafo de *G* que é conexo

- Exemplo:

G





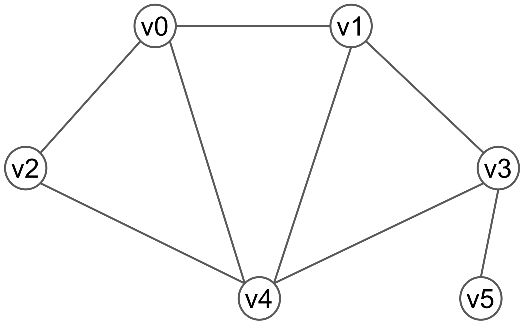
Subgrafo conexo

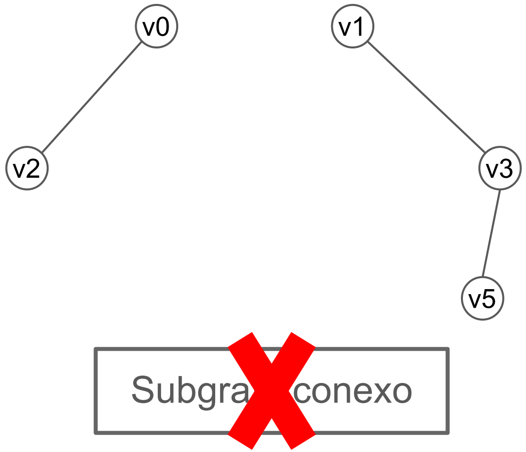
Subgrafo conexo

- Um **subgrafo conexo** de um grafo *G* é um subgrafo de *G* que é conexo

- Exemplo:

G

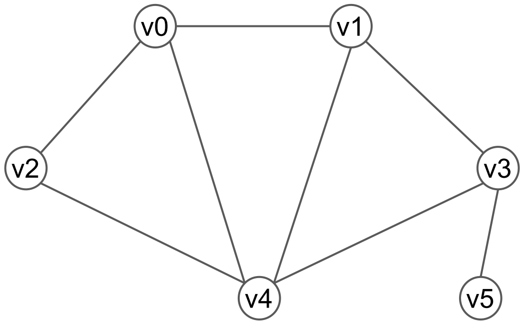


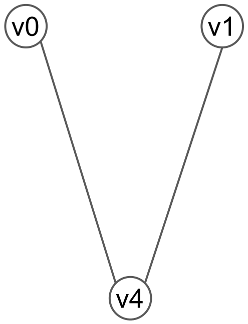


Subgrafo conexo

- Um **subgrafo conexo** de um grafo *G* é um subgrafo de *G* que é conexo
- Exemplo:

G



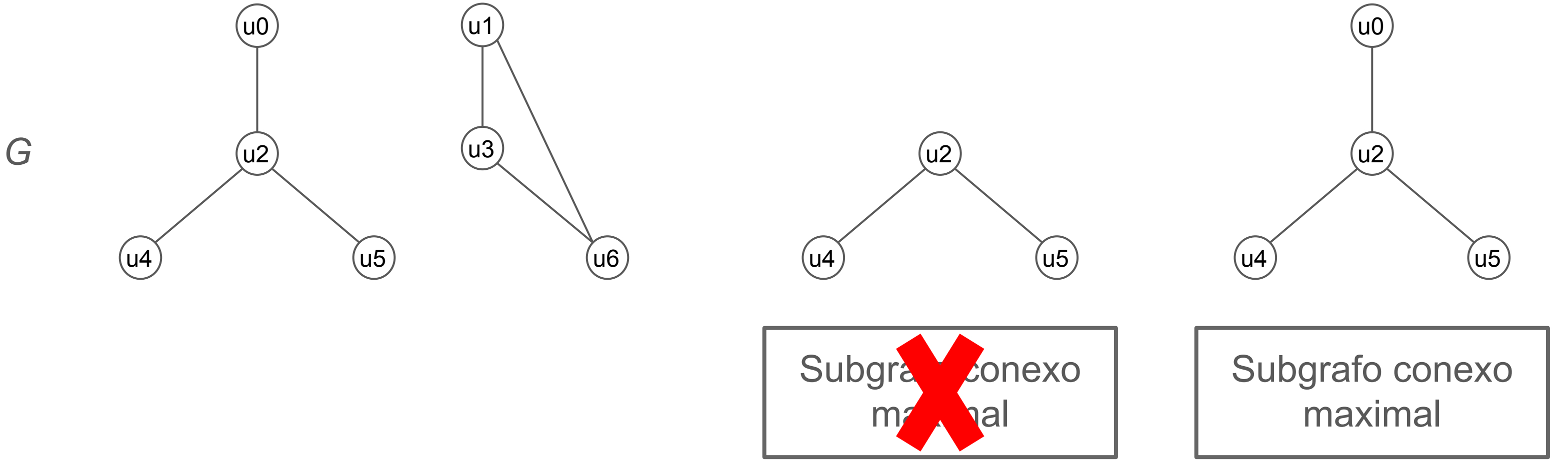


Subgrafo conexo

Subgrafo conexo maximal

- Um **subgrafo conexo maximal** de um grafo *G* é um subgrafo conexo de *G* que não está contido em outro subgrafo conexo de *G*

- Exemplo:

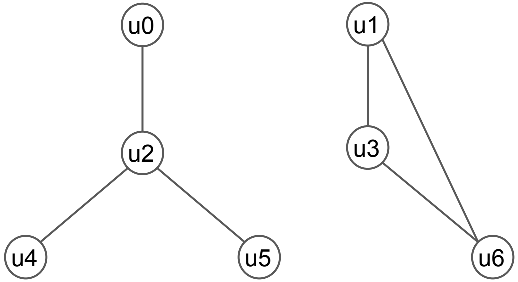


Componentes conexas

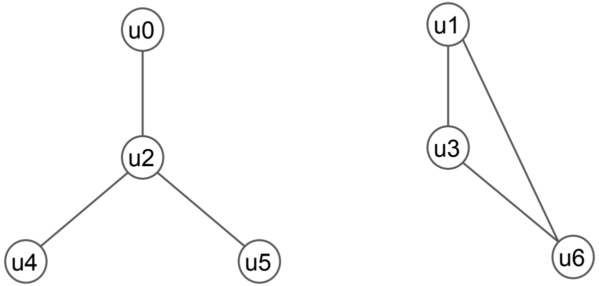
- As **componentes conexas** (ou apenas **componentes**) de um grafo *G* são os subgrafos conexos maximais de *G*
- Denotamos por *c*(*G*) o número de componentes conexas de *G*

- Exemplo:

G



Componentes conexas de *G*



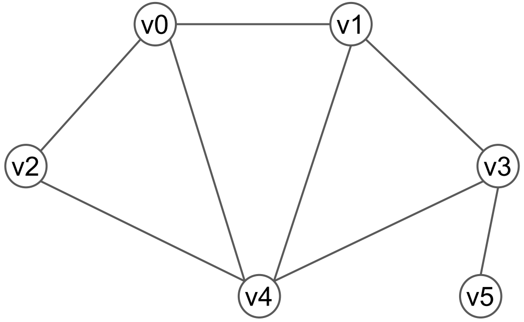
$\boxed{c(G) = 2}$

Componentes conexas

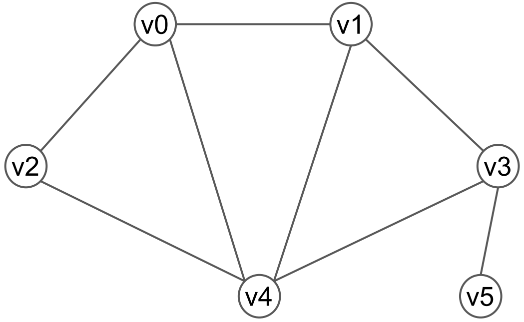
- As **componentes conexas** (ou apenas **componentes**) de um grafo *G* são os subgrafos conexos maximais de *G*
- Denotamos por *c*(*G*) o número de componentes conexas de *G*

- Exemplo:

G



Componentes conexas de *G*



c(*G*) = 1

Componentes conexas

- As **componentes conexas** (ou apenas **componentes**) de um grafo G são os subgrafos conexos maximais de G
- Denotamos por $c(G)$ o número de componentes conexas de G

- Um **grafo conexo** (com pelo menos um vértice) tem exatamente **uma componente**

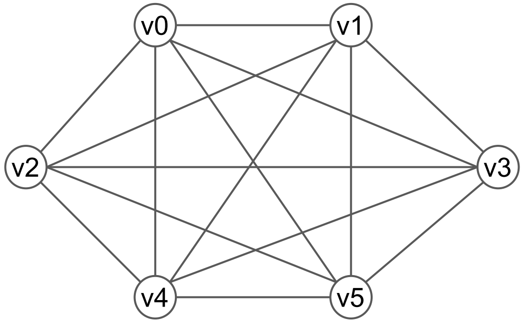
Grafo completo

- Um grafo *G* é **completo** se, para todo par de vértices v_i , v_j de *G*, existe uma **aresta** em *G* entre v_i e v_j

Grafo completo ≠ Grafo conexo

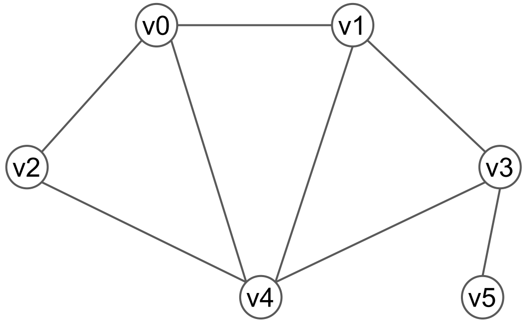
- Exemplo:

G_1



Grafo completo

G_2

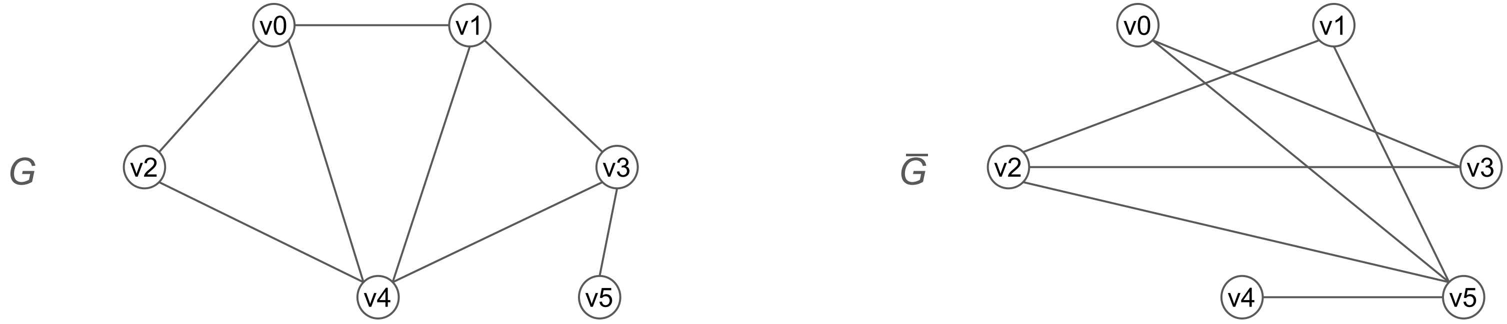


Grafo completo

Complemento de um grafo

- Dado um grafo G, o **complemento de *G*** é o grafo $\overline{G}$ tal que
  - $V(\overline{G}) = V(G)$ e
  - $v_iv_j \in E(\overline{G})$ se e somente se $v_iv_j \notin E(G)$

- Exemplo:

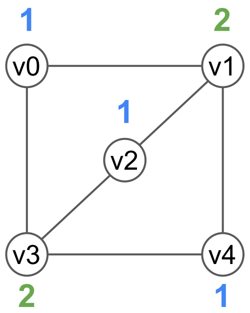


Grafo bipartido

- Um grafo G é **bipartido** se $V(G)$ pode ser particionado em dois conjuntos não-vazios de vértices V_1 e V_2 tal que
  - para todo par de vértices v_i, v_j em V_1, **não** existe uma aresta entre v_i e v_j e
  - para todo par de vértices v_i, v_j em V_2, **não** existe uma aresta entre v_i e v_j

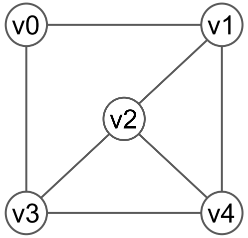
- Exemplo:

G



Grafo bipartido

G



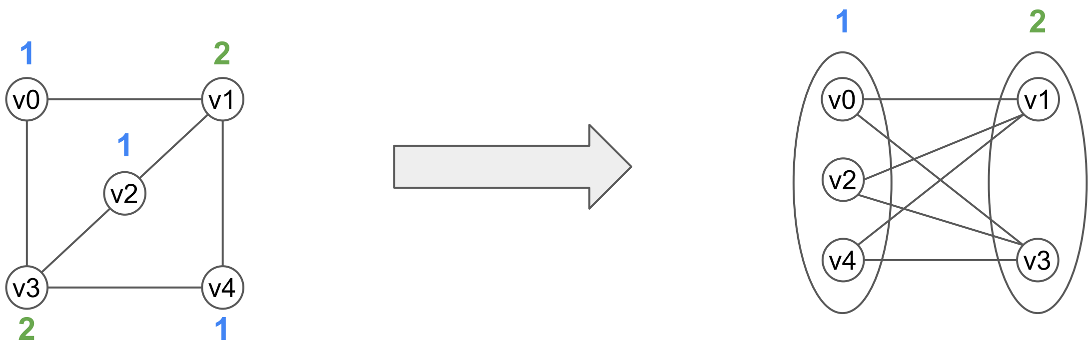
~~Grafo bipartido~~

Grafo bipartido

- Um grafo G é **bipartido** se $V(G)$ pode ser particionado em dois conjuntos não-vazios de vértices V_1 e V_2 tal que
  - para todo par de vértices v_i, v_j em V_1, **não** existe uma aresta entre v_i e v_j e
  - para todo par de vértices v_i, v_j em V_2, **não** existe uma aresta entre v_i e v_j

- Exemplo:

G



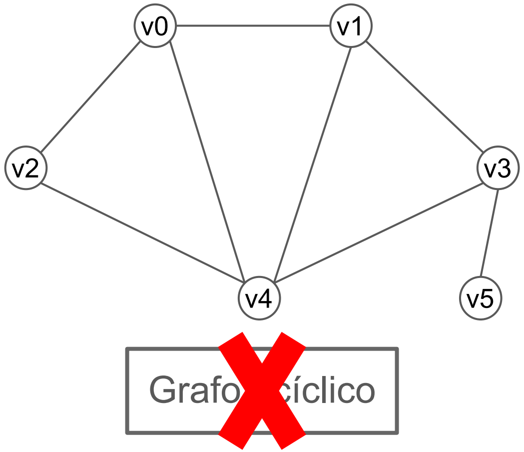
Grafo bipartido

Grafo acíclico

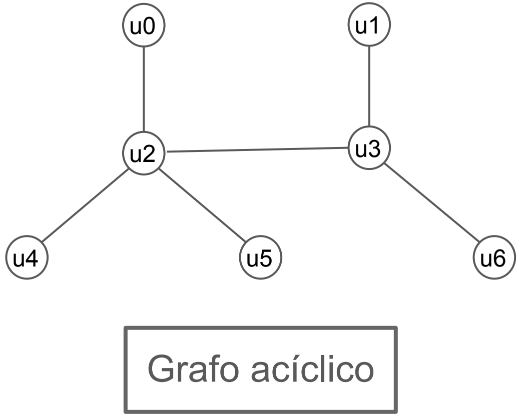
- Um grafo é **acíclico** se não possui ciclos

- Exemplo:

G_1



G_2

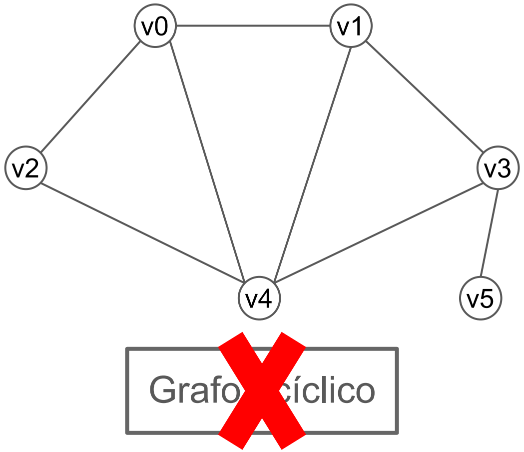


Grafo acíclico

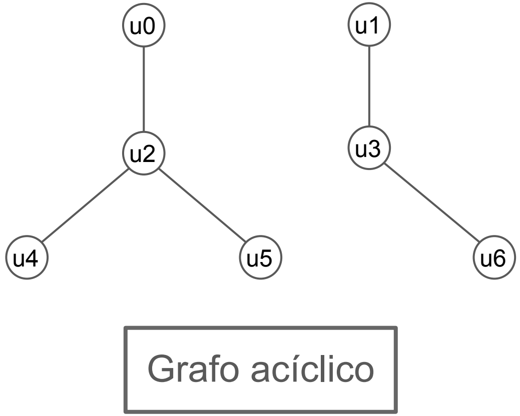
- Um grafo é **acíclico** se não possui ciclos

- Exemplo:

G_1

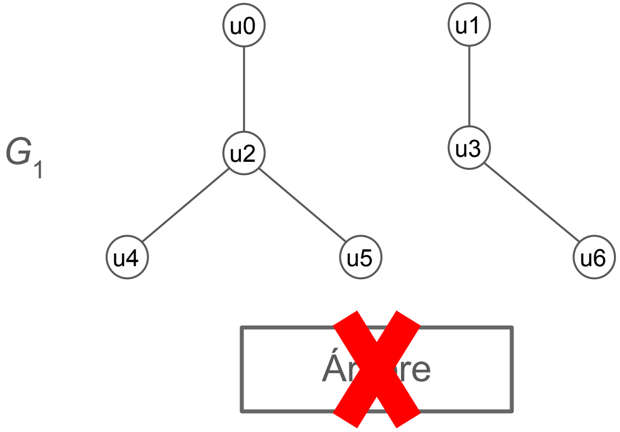


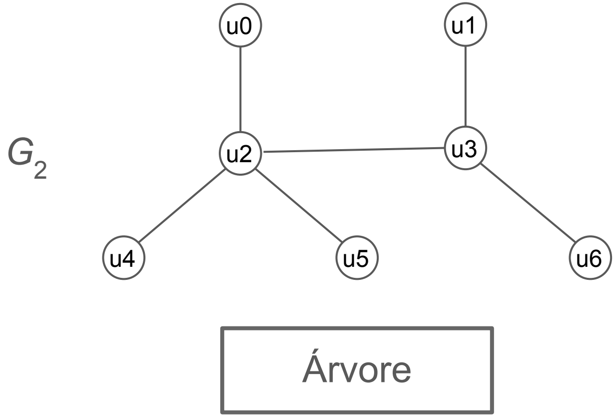
G_2



Árvore

- Uma **árvore** é um grafo conexo acíclico
- Exemplo:





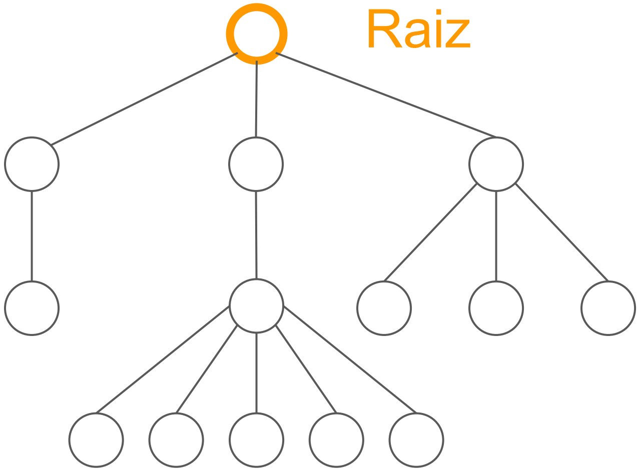
Propriedades de uma árvore

- Dado um grafo *G* com *n* vértices, as seguintes afirmações são equivalentes:
  1. *G* é uma árvore (*G* é um grafo conexo acíclico);
  2. *G* é conexo e possui *n* - 1 arestas;
  3. *G* é acíclico e possui *n* - 1 arestas;
  4. Existe exatamente um caminho entre quaisquer dois vértices de *G*.

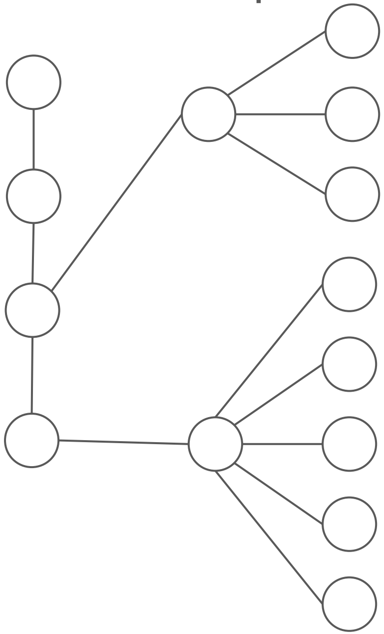
Árvore enraizada

- Uma **árvore enraizada** é uma árvore em que um dos vértices é especificado como a raiz

- Exemplo:



Árvore enraizada

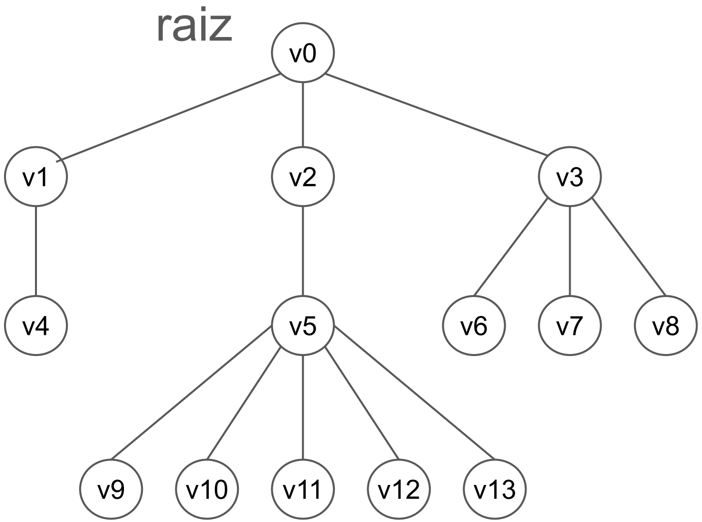


Árvore não enraizada

Árvore enraizada - terminologia

- Dada uma árvore com raiz *r*, se a última aresta do caminho entre o vértice *r* e um vértice *v* na árvore é a aresta *uv*, então dizemos que *u* é o **pai** de *v* e *v* é um **filho** de *u**

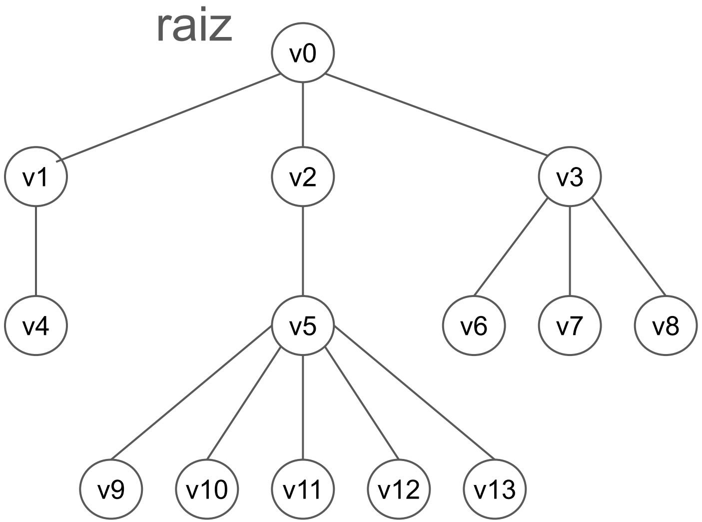
- Exemplo:
    - v2 é   de v0
    - v5 é   de v10
    - v13 é   de v5
    - v6 é   de v7



Árvore enraizada - terminologia

- Dada uma árvore com raiz *r*, se a última aresta do caminho entre o vértice *r* e um vértice *v* na árvore é a aresta *uv*, então dizemos que *u* é o **pai** de *v* e *v* é um **filho** de *u*

- Exemplo:
  - v2 é filho de v0
  - v5 é pai de v10
  - v13 é filho de v5
  - v6 não é pai de v7 (v6 é **irmão** de v7)



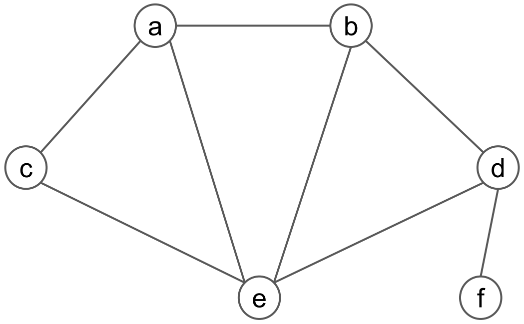
Conteúdo

- Grafos - Conceitos básicos
- **Representação computacional**
- Busca em profundidade
- Busca em largura
- Grafos dirigidos - Conceitos básicos
- Grafos dirigidos - Representação computacional
- Grafos dirigidos - Busca em profundidade e em largura
- Referências

Representação computacional

- A seguir, veremos duas formas comuns de representar um grafo *G*
- Para isso, vamos considerar que fizemos uma associação dos índices 0, 1, … $|V(G)| - 1$ aos vértices de *G*

G

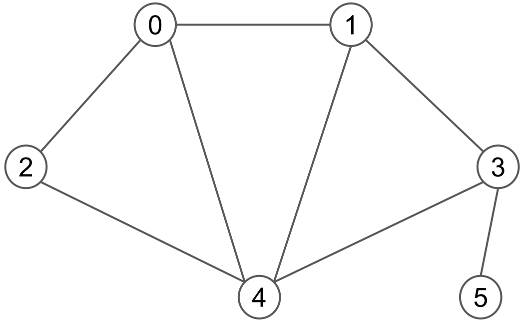


- $0 \rightarrow a$
- $1 \rightarrow b$
- $2 \rightarrow c$
- $3 \rightarrow d$
- $4 \rightarrow e$
- $5 \rightarrow f$

Matriz de adjacências

- A representação de *G* como uma **matriz de adjacências** consiste em uma matriz de |*V*(*G*)| linhas, com índices 0, 1, …, |*V*(*G*)| - 1, e de |*V*(*G*)| colunas, com índices 0, 1, …, |*V*(*G*)| - 1, tal que a célula (*i*, *j*) da matriz é igual a
  - 1 se i j é uma aresta de *G*
  - 0 caso contrário

G



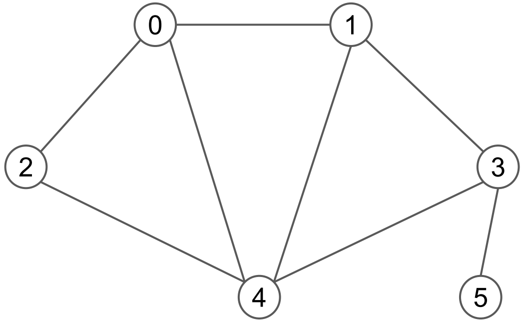
Matriz de adjacências de *G*

	0	1	2	3	4	5
0	0	1	1	0	1	0
1	1	0	0	1	1	0
2	1	0	0	0	1	0
3	0	1	0	0	1	1
4	1	1	1	1	0	0
5	0	0	0	1	0	0

Matriz de adjacências

- Observações:
    - Não é possível representar arestas paralelas
    - Para grafos simples, todas as células da diagonal principal da matriz são iguais a 0
    - Para grafos onde não consideramos uma direção para as arestas, uma aresta *i j* é representada por duas células da matriz: (*i*, *j*) e (*j*, *i*)

G



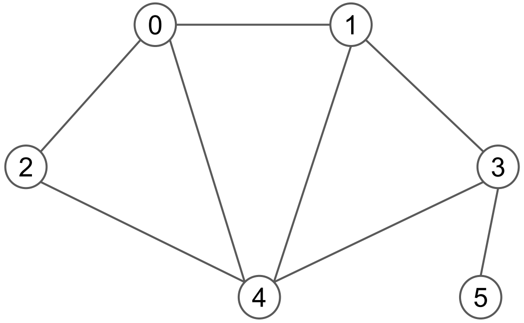
Matriz de adjacências de *G*

	0	1	2	3	4	5
0	0	1	1	0	1	0
1	1	0	0	1	1	0
2	1	0	0	0	1	0
3	0	1	0	0	1	1
4	1	1	1	1	0	0
5	0	0	0	1	0	0

Matriz de adjacências

- Observações:
  - Para grafos onde não consideramos uma direção para as arestas, a matriz é simétrica em relação à diagonal principal

G



Matriz de adjacências de *G*

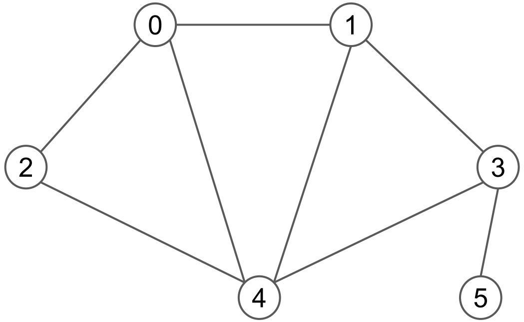
	0	1	2	3	4	5
0	0	1	1	0	1	0
1	1	0	0	1	1	0
2	1	0	0	0	1	0
3	0	1	0	0	1	1
4	1	1	1	1	0	0
5	0	0	0	1	0	0

Matriz de adjacências

- Implementação:

```cpp
// n eh o numero de vertices do grafo
vector<vector<int>> matriz_adj(n, vector<int>(n));
```

G



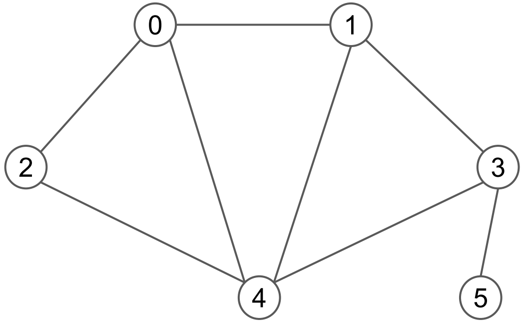
Matriz de adjacências de *G*

	0	1	2	3	4	5
0	0	1	1	0	1	0
1	1	0	0	1	1	0
2	1	0	0	0	1	0
3	0	1	0	0	1	1
4	1	1	1	1	0	0
5	0	0	0	1	0	0

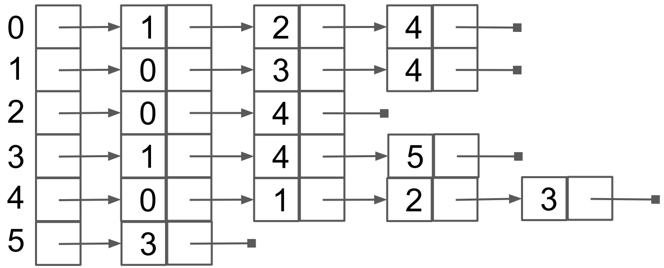
Listas de adjacência

- A representação de G como **listas de adjacência** consiste em um vetor de $|V(G)|$ elementos, com índices 0, 1, …, $|V(G)| - 1$, tal que o elemento i do vetor armazena uma lista com os vértices adjacentes ao vértice i em G

G



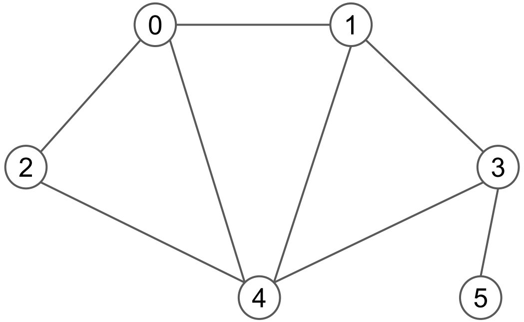
Listas de adjacência de G



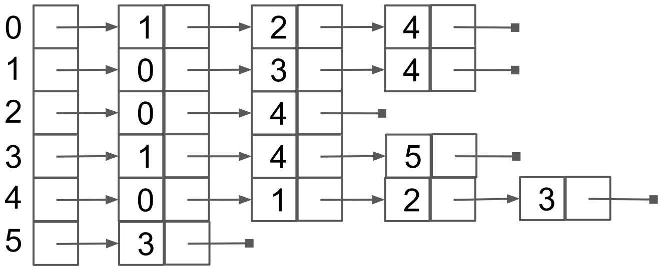
Listas de adjacência

- Observações:
  - Para grafos onde não consideramos uma direção para as arestas, uma aresta *i j* é representada em duas listas de adjacência: o vértice *i* está na lista do vértice *j* e o vértice *j* está na lista do vértice *i*

G



Listas de adjacência de *G*

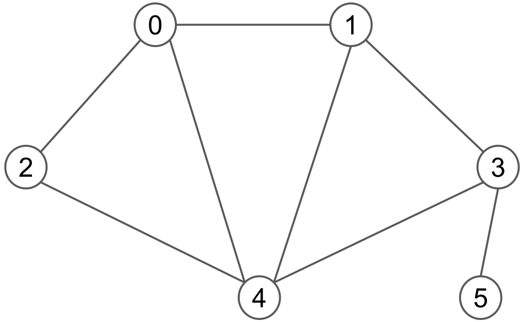


Listas de adjacência

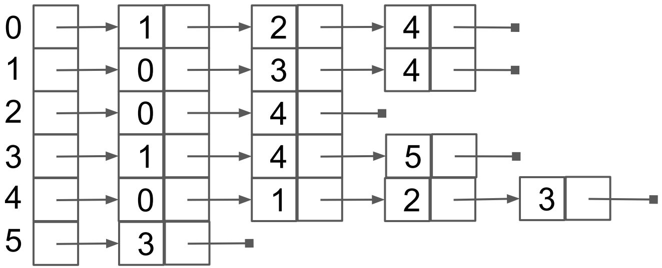
- Implementação:

```
// n eh o numero de vertices do grafo
vector<list<int>> listas_adj(n);
```

G



Listas de adjacência de *G*



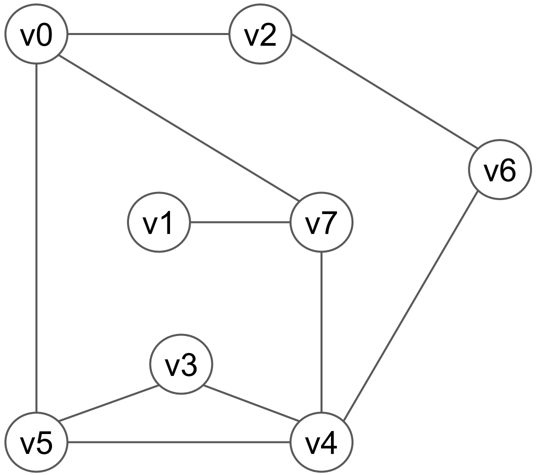
Conteúdo

- Grafos - Conceitos básicos
- Representação computacional
- **Busca em profundidade**
- Busca em largura
- Grafos dirigidos - Conceitos básicos
- Grafos dirigidos - Representação computacional
- Grafos dirigidos - Busca em profundidade e em largura
- Referências

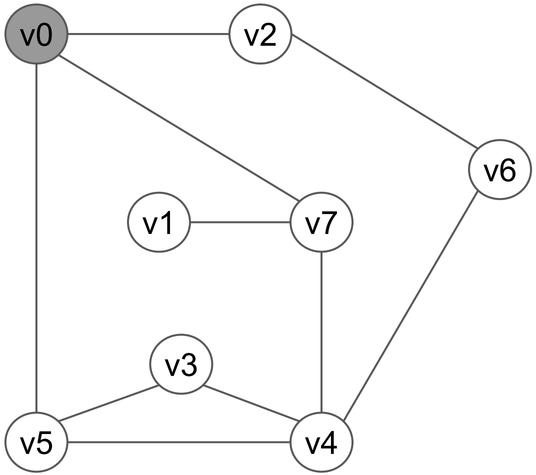
Percorrendo os vértices de um grafo

- Em várias situações, queremos percorrer de maneira eficiente os vértices de um grafo
- Vamos ver um algoritmo que faz isto

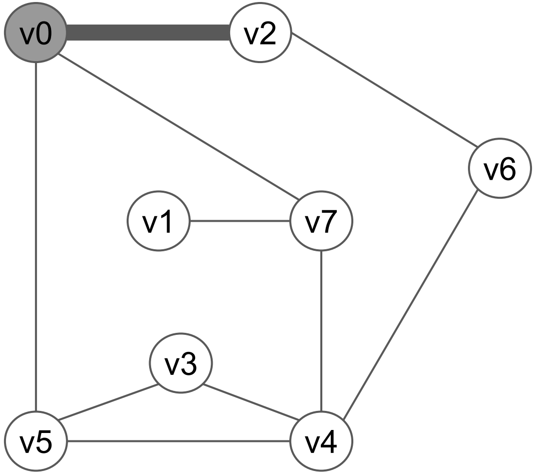
Percorrendo os vértices de um grafo



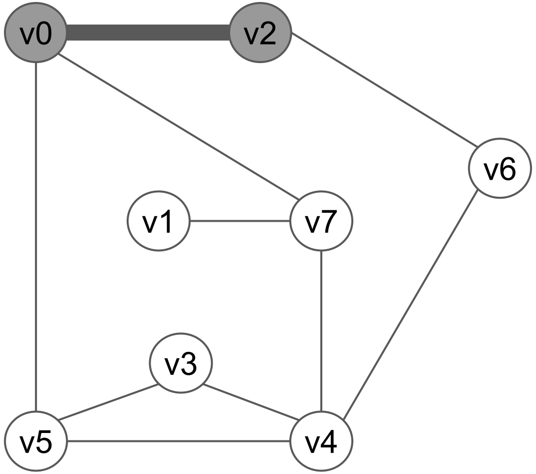
Percorrendo os vértices de um grafo



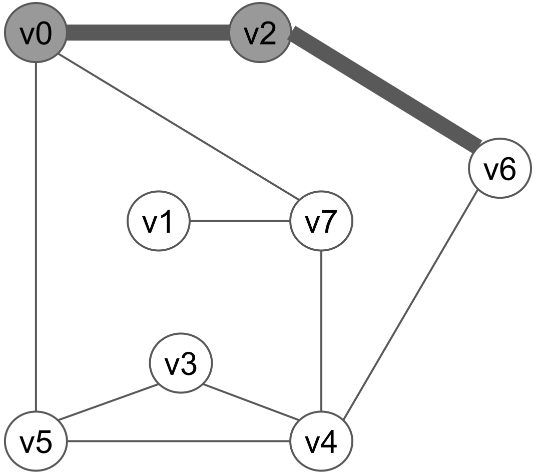
Percorrendo os vértices de um grafo



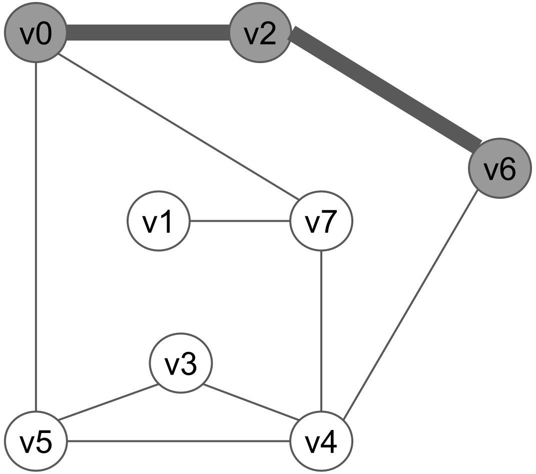
Percorrendo os vértices de um grafo



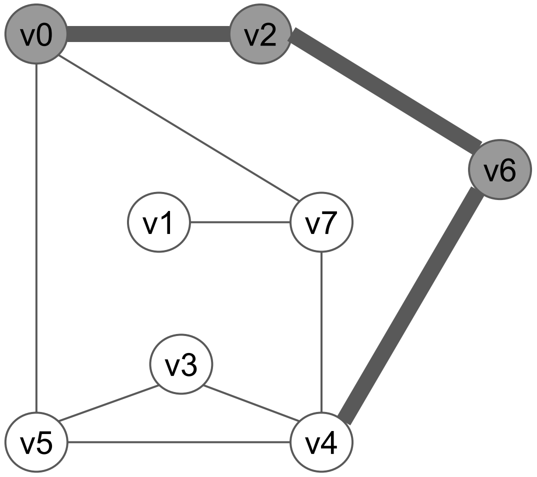
Percorrendo os vértices de um grafo



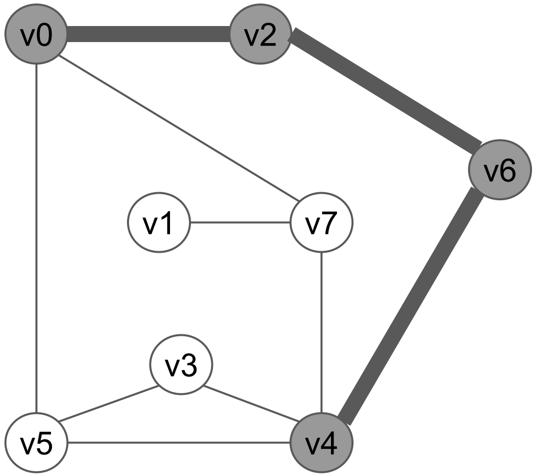
Percorrendo os vértices de um grafo



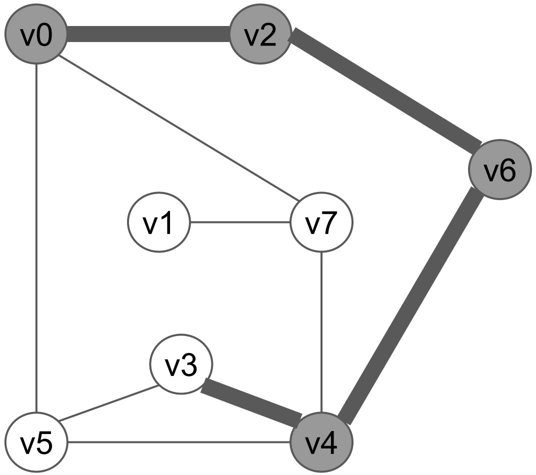
Percorrendo os vértices de um grafo



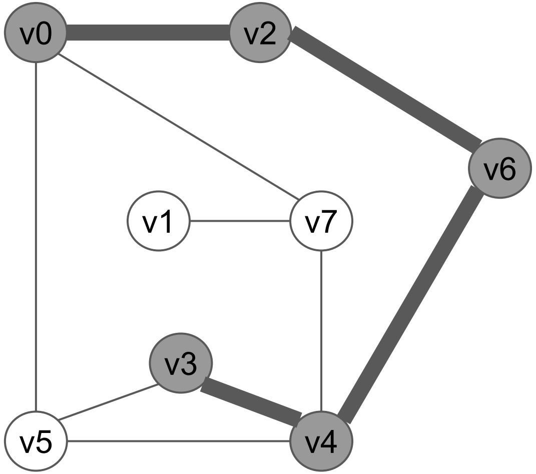
Percorrendo os vértices de um grafo



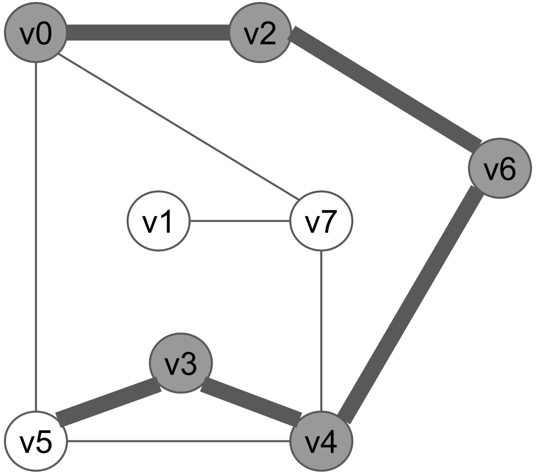
Percorrendo os vértices de um grafo



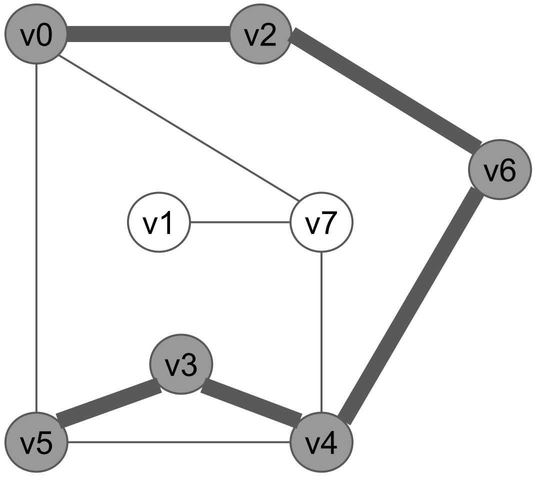
Percorrendo os vértices de um grafo



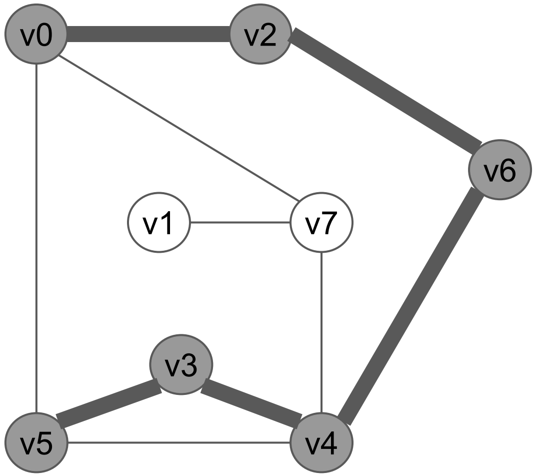
Percorrendo os vértices de um grafo



Percorrendo os vértices de um grafo

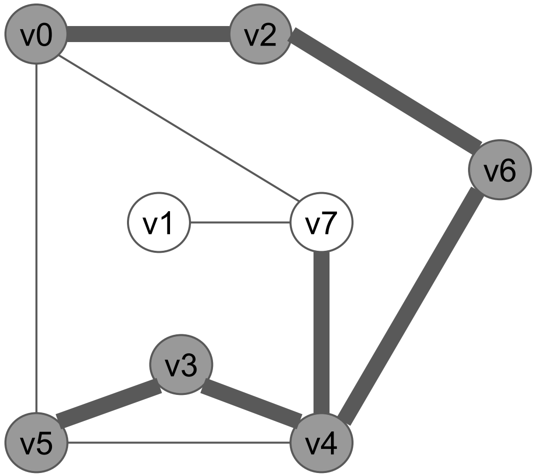


Percorrendo os vértices de um grafo

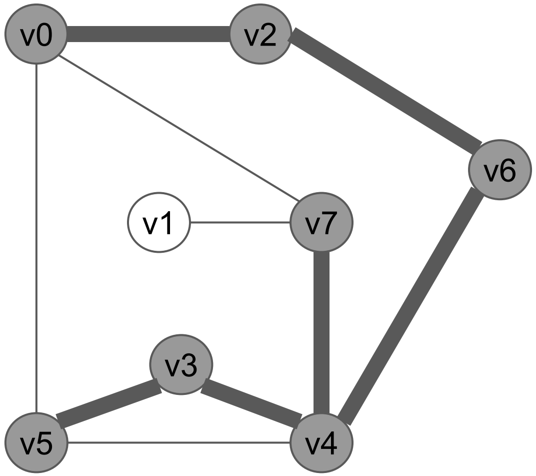


Não é possível ir adiante (sem repetir vértices) a partir daqui. Temos que retroceder (**backtrack**) para conseguir considerar outras possibilidades

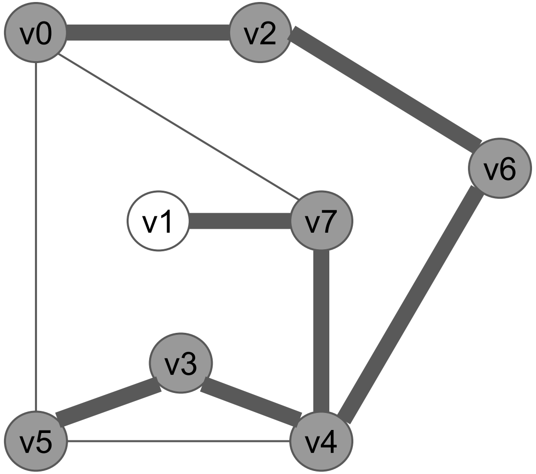
Percorrendo os vértices de um grafo



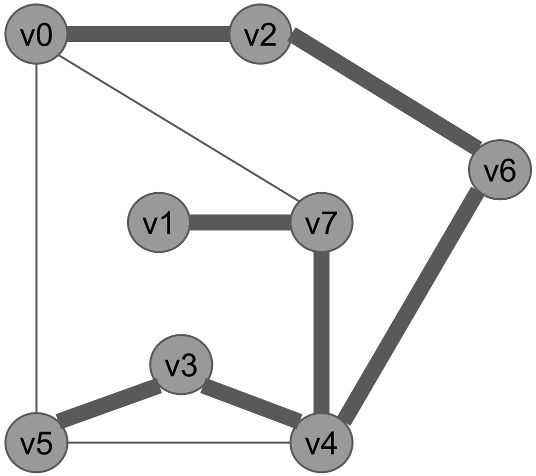
Percorrendo os vértices de um grafo



Percorrendo os vértices de um grafo



Percorrendo os vértices de um grafo



Percorrendo os vértices de um grafo

- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

Percorrendo os vértices de um grafo

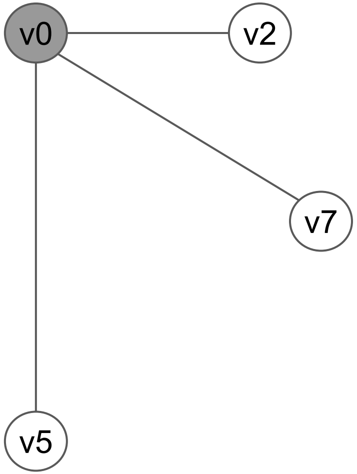
- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

1. Comece em v_0

Percorrendo os vértices de um grafo

- Como podemos descrever a estratégia vista antes de uma forma geral?
    - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

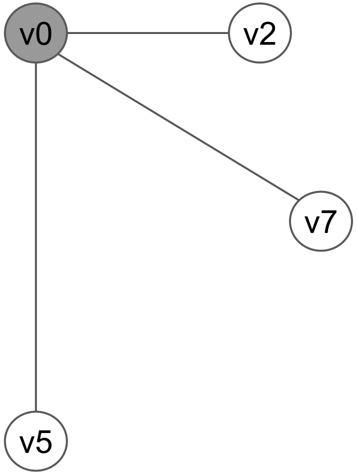
1. Comece em v_0
2. Considere os vizinhos de v_0



Percorrendo os vértices de um grafo

- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

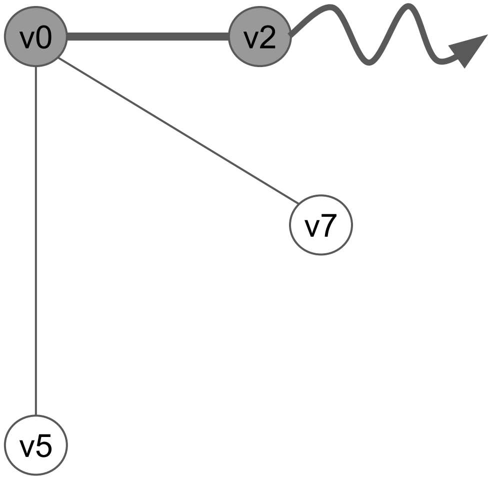
1. Comece em v_0
2. Considere os vizinhos de v_0
   Observe que, para percorrer os demais vértices do grafo, podemos



Percorrendo os vértices de um grafo

- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

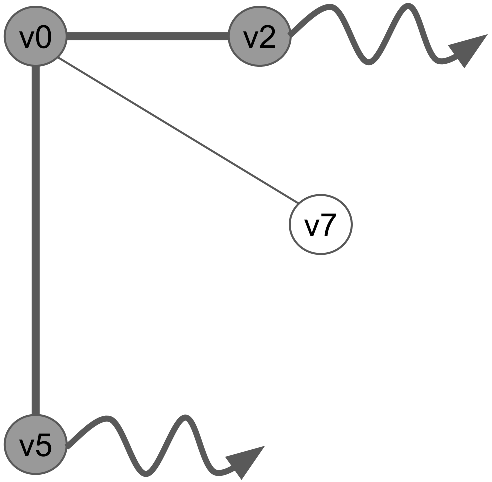
1. Comece em v_0
2. Considere os vizinhos de v_0
   Observe que, para percorrer os demais vértices do grafo, podemos percorrer os vértices partindo de v_2



Percorrendo os vértices de um grafo

- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

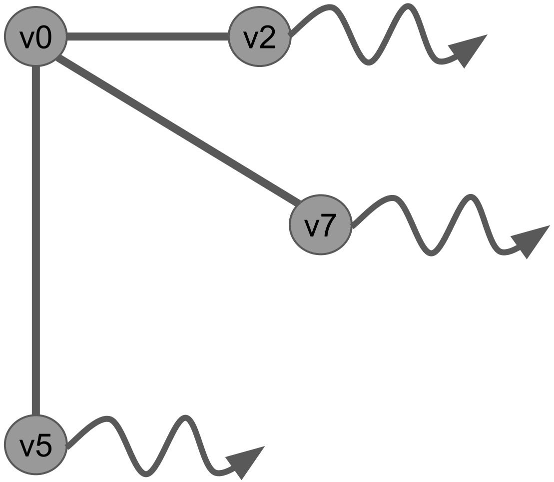
1. Comece em v_0
2. Considere os vizinhos de v_0
   Observe que, para percorrer os demais vértices do grafo, podemos percorrer os vértices partindo de v_2, percorrer os vértices partindo de v_5



Percorrendo os vértices de um grafo

- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

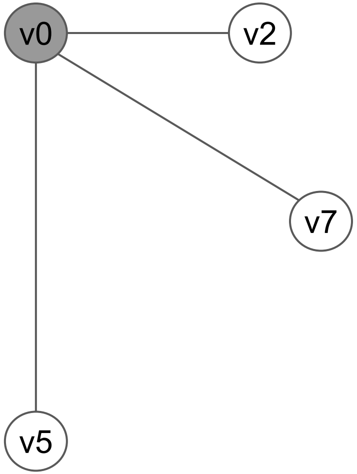
1. Comece em v_0
2. Considere os vizinhos de v_0
   Observe que, para percorrer os demais vértices do grafo, podemos percorrer os vértices partindo de v_2, percorrer os vértices partindo de v_5 e percorrer os vértices partindo de v_7



Percorrendo os vértices de um grafo

- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

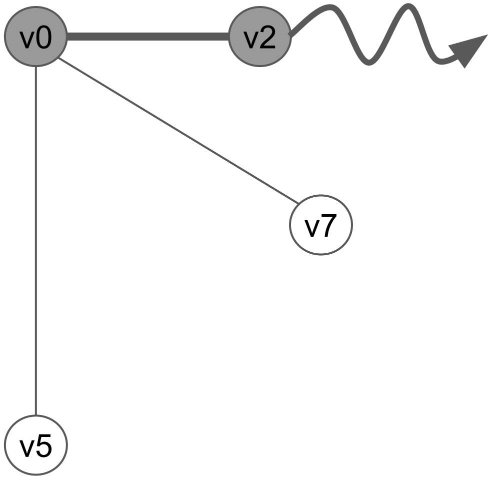
1. Comece em v_0
2. Considere os vizinhos de v_0



Percorrendo os vértices de um grafo

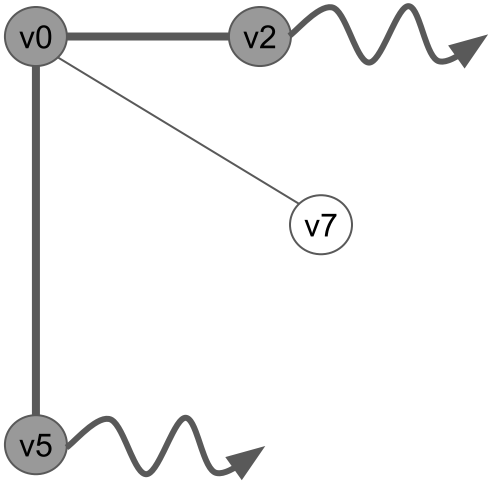
- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

1. Comece em v_0
2. Considere os vizinhos de v_0
3. Percorra recursivamente os vértices do grafo partindo de v_2



Percorrendo os vértices de um grafo

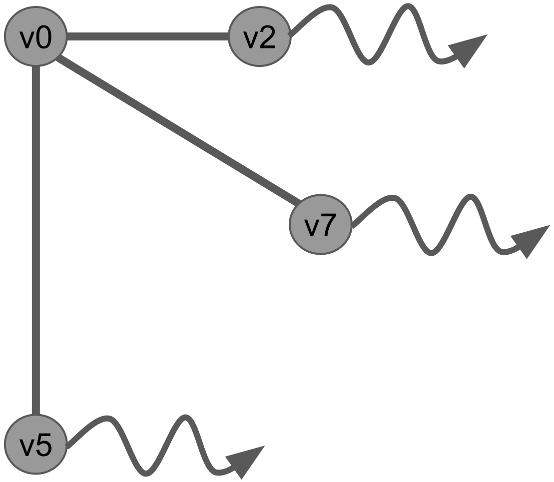
- Como podemos descrever a estratégia vista antes de uma forma geral?
    - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
    1. Comece em v_0
    2. Considere os vizinhos de v_0
    3. Percorra recursivamente os vértices do grafo partindo de v_2
    4. Percorra recursivamente os vértices do grafo partindo de v_5



Percorrendo os vértices de um grafo

- Como podemos descrever a estratégia vista antes de uma forma geral?
  - Queremos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

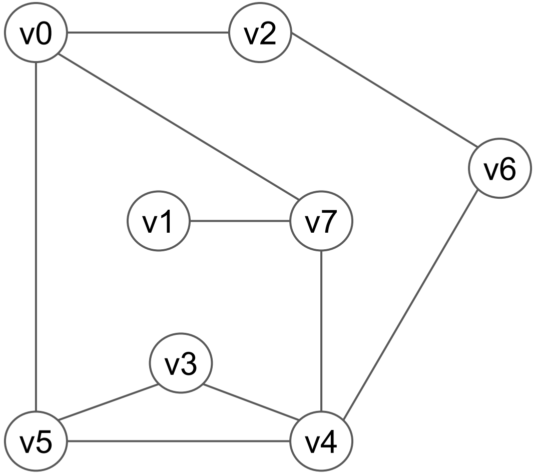
1. Comece em v_0
2. Considere os vizinhos de v_0
3. Percorra recursivamente os vértices do grafo partindo de v_2
4. Percorra recursivamente os vértices do grafo partindo de v_5 e
5. Percorra recursivamente os vértices do grafo partindo de v_7



Não queremos visitar novamente vértices já visitados. Por isso, vamos marcar os vértices que vão sendo visitados

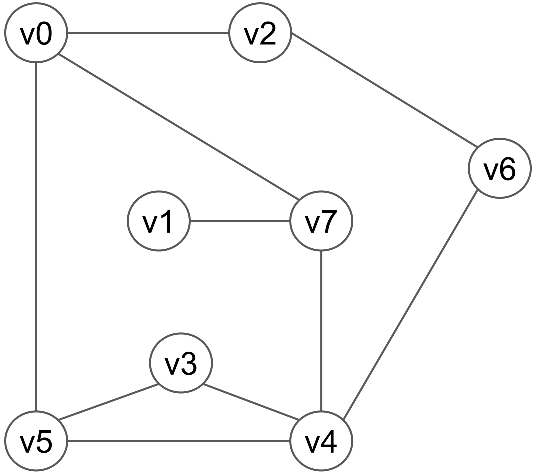
Percorrendo os vértices de um grafo - Implementação

```
void percorre(int v) {




}
```



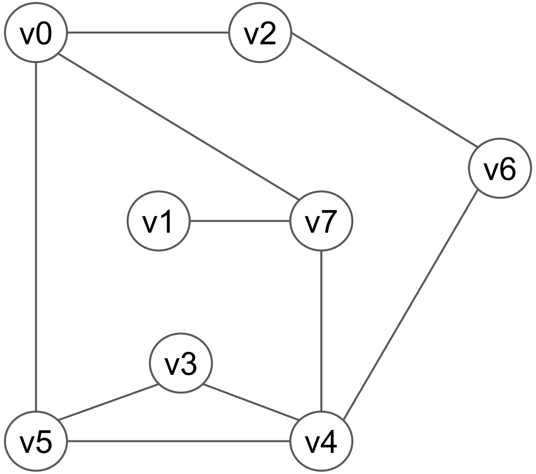
Percorrendo os vértices de um grafo - Implementação

```
void percorre(int v) {

    for (auto u : listas_adj[v])


}
```



Percorrendo os vértices de um grafo - Implementação

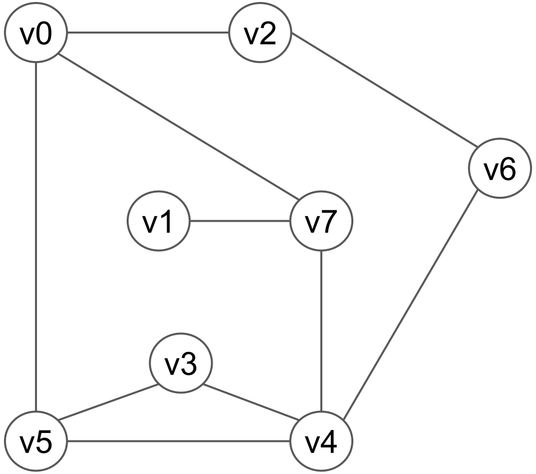
```
void percorre(int v) {

    for (auto u : listas_adj[v])
        percorre(u);

}
```



Percorrendo os vértices de um grafo - Implementação

```
// O vetor marcado eh criado e inicializado antes
// da funcao percorre ser chamada
```

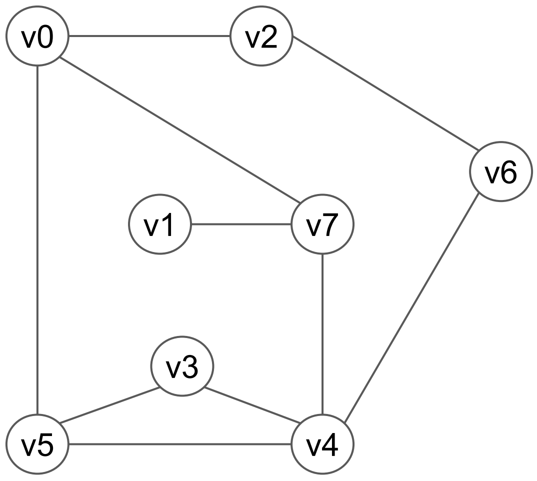
```
void percorre(int v, int marcado[]) {

    for (auto u : listas_adj[v])
        percorre(u, marcado);

}
```



Percorrendo os vértices de um grafo - Implementação

```
// O vetor marcado eh criado e inicializado antes
// da funcao percorre ser chamada
```

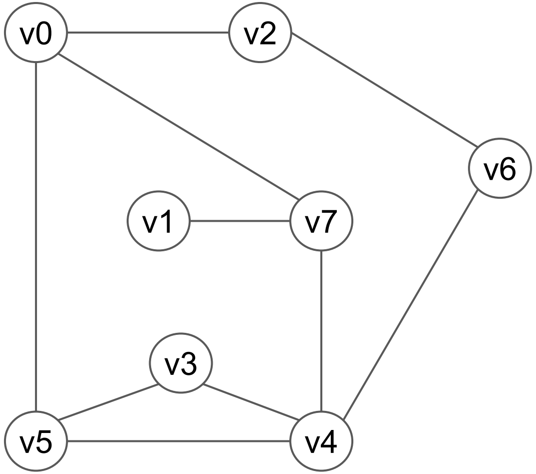
```
void percorre(int v, int marcado[]) {
    marcado[v] = 1;
    for (auto u : listas_adj[v])
        if (marcado[u] == 0)
            percorre(u, marcado);
}
```



Percorrendo os vértices de um grafo - Implementação

```
// O vetor marcado eh criado e inicializado antes
// da funcao percorre ser chamada
```

```
void percorre(int v, int marcado[]) {
    marcado[v] = 1;
    for (auto u : listas_adj[v])
        if (marcado[u] == 0)
            percorre(u, marcado);
}
```

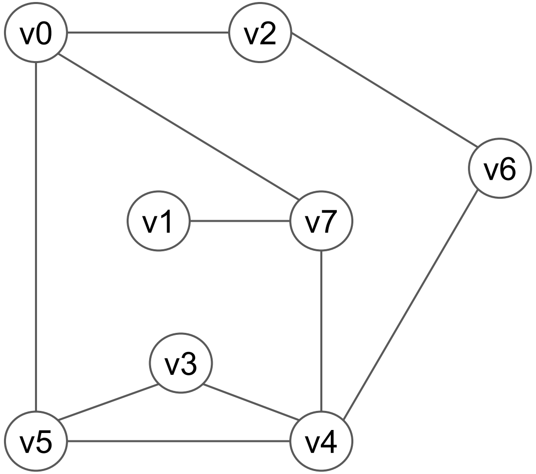


O processo de percorrer um grafo também é chamado de **busca**

Percorrendo os vértices de um grafo - Implementação

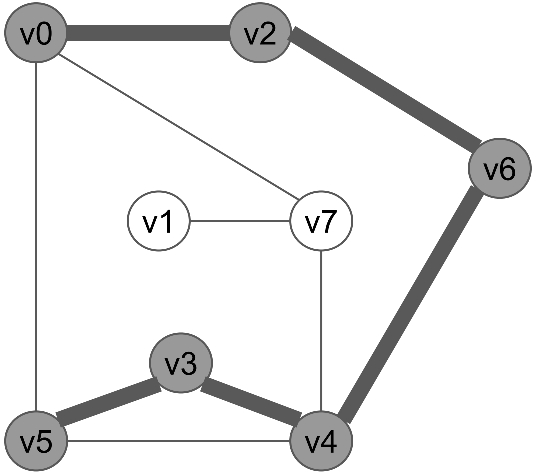
```
// O vetor marcado eh criado e inicializado antes
// da funcao busca ser chamada
```

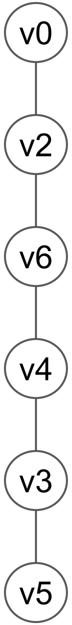
```
void busca(int v, int marcado[]) {
    marcado[v] = 1;
    for (auto u : listas_adj[v])
        if (marcado[u] == 0)
            busca(u, marcado);
}
```



O processo de percorrer um grafo também é chamado de **busca**

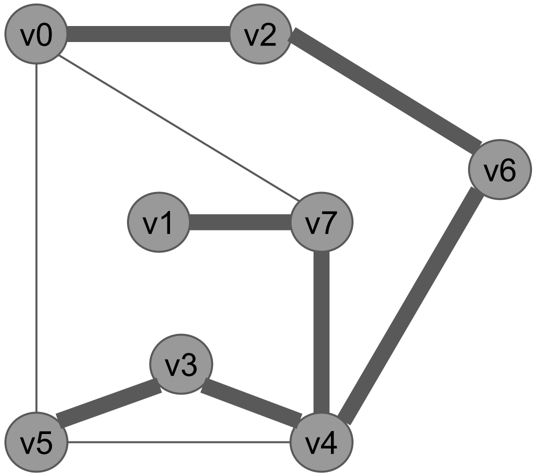
Percorrendo os vértices de um grafo - Dinâmica

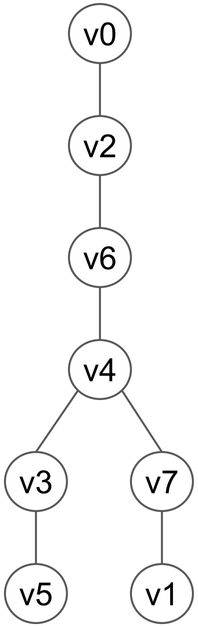




A busca segue em **profundidade** até não ser mais possível, para depois retornar

Percorrendo os vértices de um grafo - Dinâmica

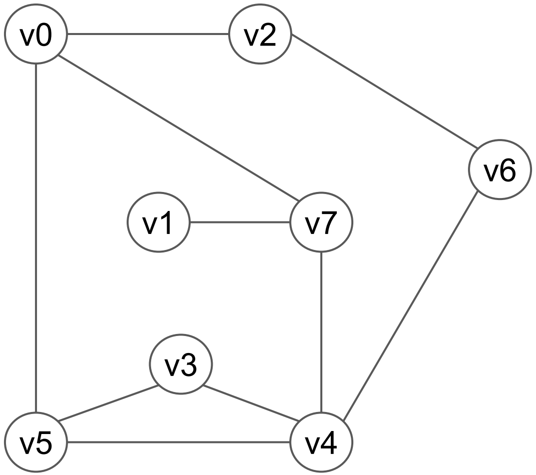




Percorrendo os vértices de um grafo - Implementação

```
// O vetor marcado eh criado e inicializado antes
// da funcao busca ser chamada
```

```
void busca(int v, int marcado[]) {
    marcado[v] = 1;
    for (auto u : listas_adj[v])
        if (marcado[u] == 0)
            busca(u, marcado);
}
```

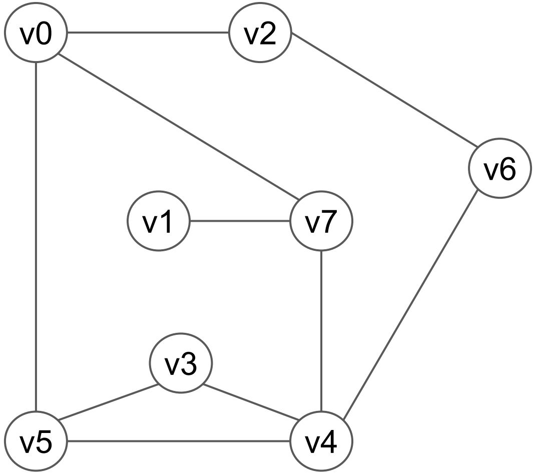


A busca segue em **profundidade** até não ser mais possível, para depois retornar

Percorrendo os vértices de um grafo - Implementação

```
// O vetor marcado eh criado e inicializado antes
// da funcao busca_prof ser chamada
```

```
void busca_prof(int v, int marcado[]) {
    marcado[v] = 1;
    for (auto u : listas_adj[v])
        if (marcado[u] == 0)
            busca_prof(u, marcado);
}
```



A busca segue em **profundidade** até não ser mais possível, para depois retornar

Busca em um grafo

- A estratégia de busca em um grafo vista nos slides anteriores é conhecida como o algoritmo de **busca em profundidade**
- Ao realizar uma busca em um grafo, conseguimos visitar **todo vértice** *w* do grafo tal que existe um **caminho** entre o **vértice inicial** da busca e *w*
- Em outras palavras, conseguimos visitar todos os vértices da **componente conexa** do grafo que contém o vértice inicial da busca
- Se o grafo é conexo, então conseguimos visitar todos os seus vértices

Aplicações

- Usando o algoritmo de busca em profundidade, como podemos verificar se um grafo é conexo?

Aplicações

- Usando o algoritmo de busca em profundidade, como podemos verificar se um grafo é conexo?
  Podemos realizar a busca e em seguida verificar se algum vértice do grafo não foi marcado como visitado

Aplicações

- Usando o algoritmo de busca em profundidade, como podemos determinar o número de componentes conexas de um grafo?

Aplicações

- Usando o algoritmo de busca em profundidade, como podemos determinar o número de componentes conexas de um grafo?

```c
int conta_comps_conexas() {
    // Criacao e inicializacao do vetor marcado
    int cont = 0;
    for (int v = 0; v < n; v++)
        if (marcado[v] == 0) {
            busca_prof(v, marcado);
            cont++;
        }
    return cont;
}
```

Conteúdo

- Grafos - Conceitos básicos
- Representação computacional
- Busca em profundidade
- **Busca em largura**
- Grafos dirigidos - Conceitos básicos
- Grafos dirigidos - Representação computacional
- Grafos dirigidos - Busca em profundidade e em largura
- Referências

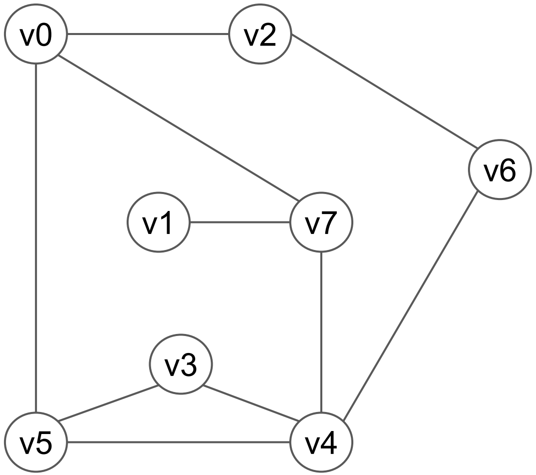
Busca em largura

- Considere o seguinte objetivo:
  Dado um grafo, queremos determinar a distância entre um certo vértice e cada um dos vértices do grafo
- Podemos atingir este objetivo através de uma estratégia de busca chamada **busca em largura**
- Para este objetivo, o algoritmo de busca em profundidade não é útil, pois a estratégia utilizada não tem relação com calcular distâncias

- Em uma busca em largura, vamos percorrer o grafo da seguinte maneira: vamos **visitar primeiro** os vértices **mais próximos** do vértice inicial

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

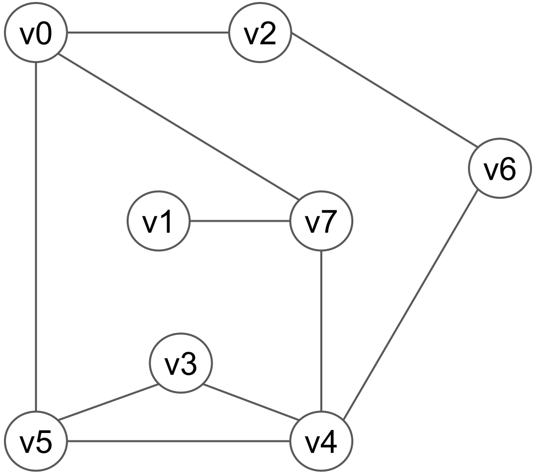


Estrutura de dados:

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

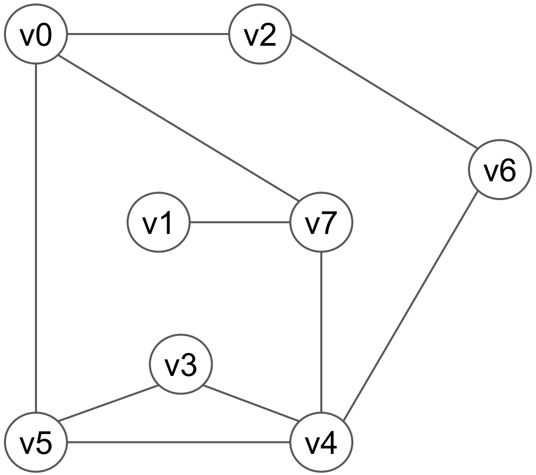


Estrutura de dados:

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

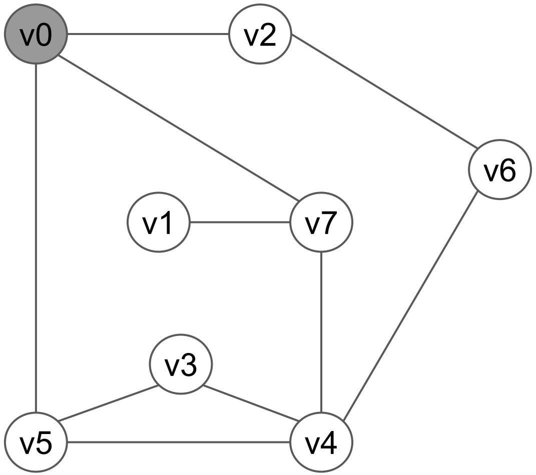


Estrutura de dados: v0

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

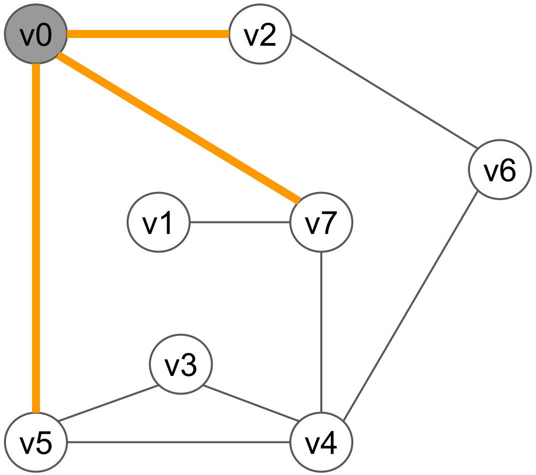


Estrutura de dados:

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

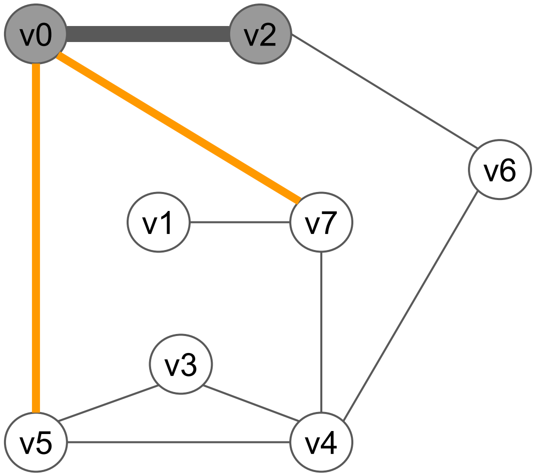


Estrutura de dados: v2 v5 v7

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

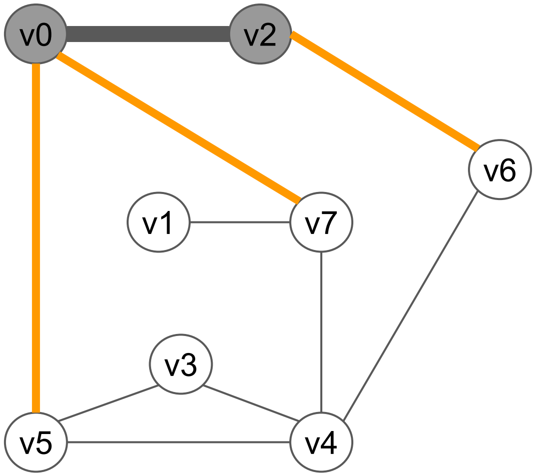


Estrutura de dados: v5 v7

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

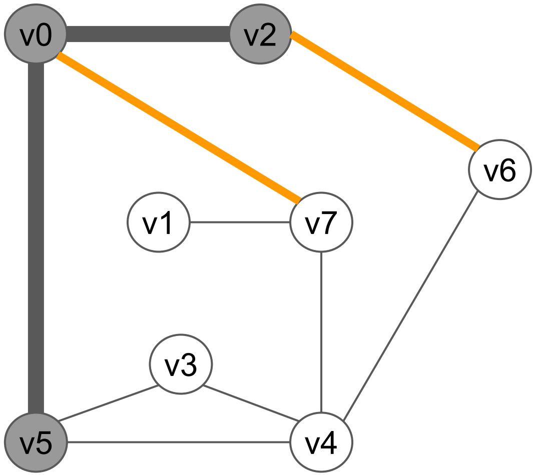


Estrutura de dados: v5 v7 v6

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

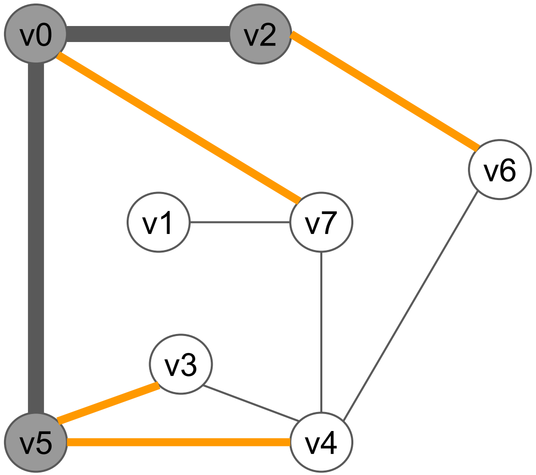


Estrutura de dados: v7 v6

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

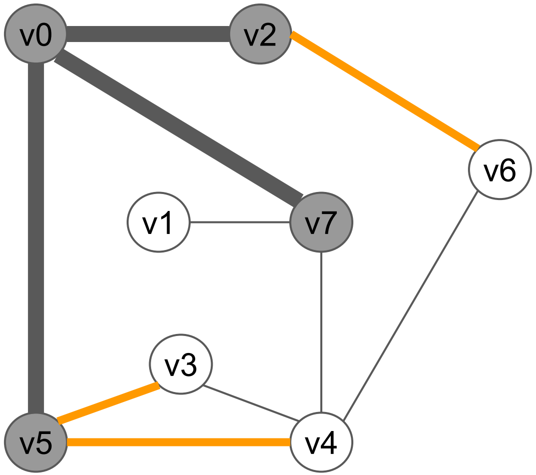


Estrutura de dados: v7 v6 v3 v4

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

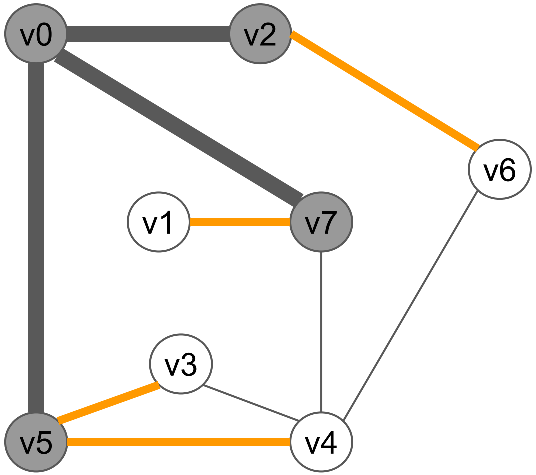


Estrutura de dados: v6 v3 v4

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



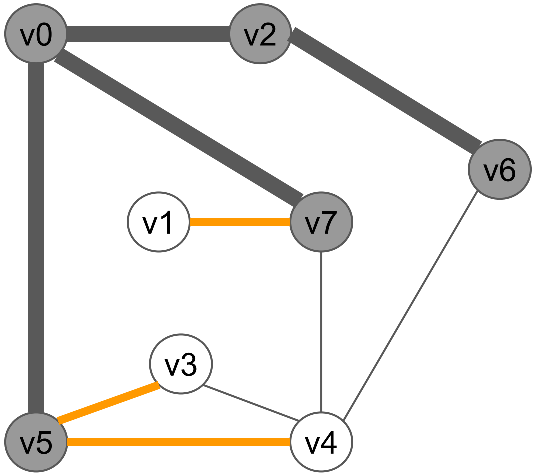
Estrutura de dados:

v6 v3 v4 v1

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

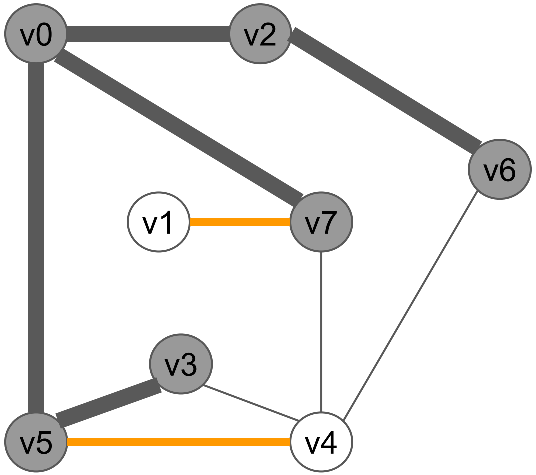


Estrutura de dados: v3 v4 v1

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**

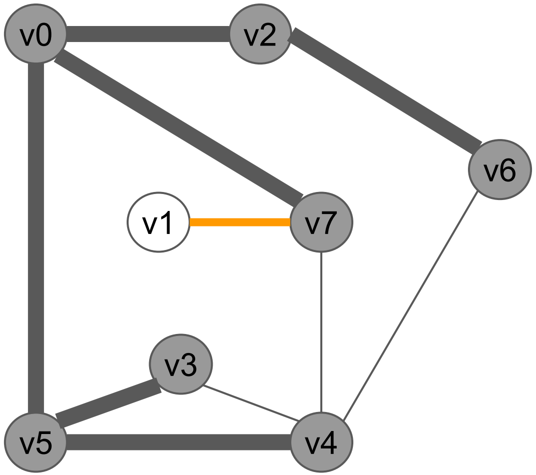


Estrutura de dados: v4 v1

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



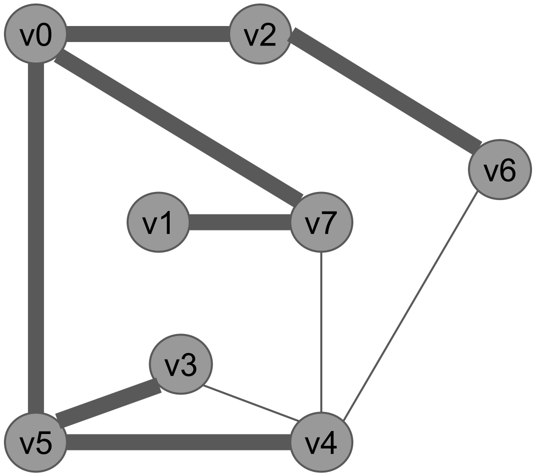
Estrutura de dados:

v1

Busca em largura

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

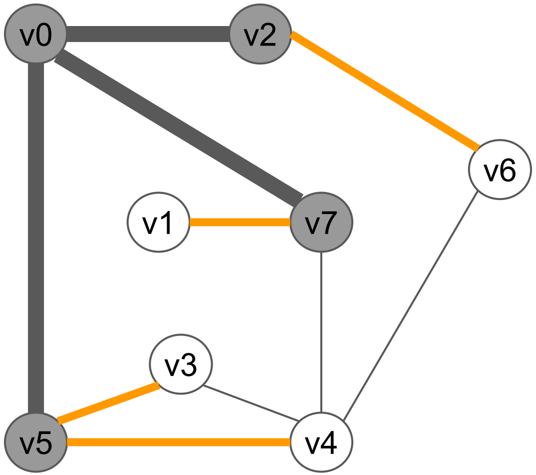
Partindo do **v0**

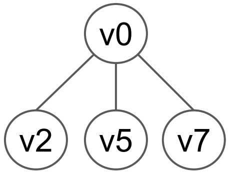


Estrutura de dados:

Busca em largura - Dinâmica

Partindo do **v0**



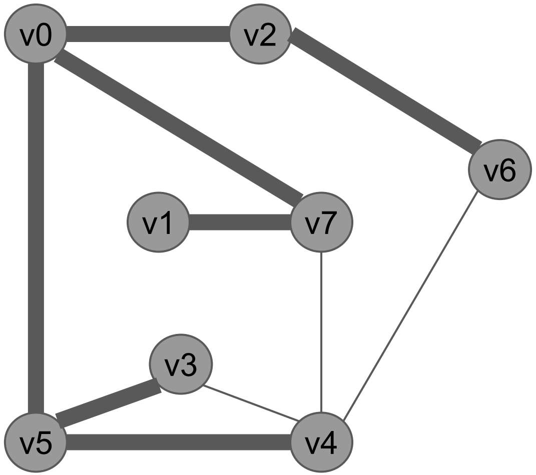


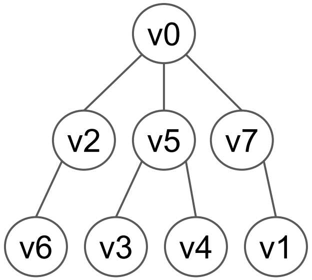
A busca segue em **largura** até não ser mais possível, para depois se aprofundar

Estrutura de dados: v6 v3 v4 v1

Busca em largura - Dinâmica

Partindo do **v0**





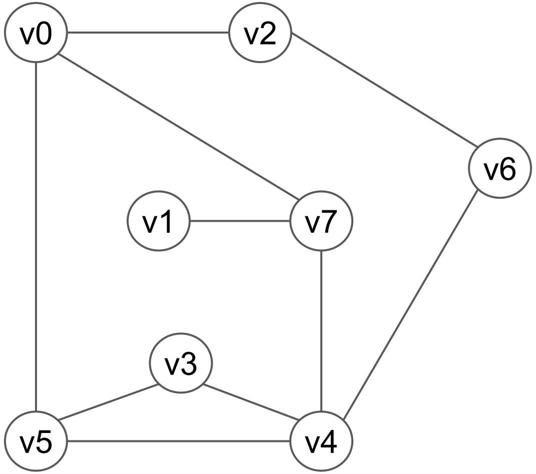
Estrutura de dados:

Busca em largura - Implementação

- O que podemos dizer da lógica através da qual os vértices são visitados no algoritmo de busca em largura?
  - É uma lógica de **fila**
- Então, vamos implementar este algoritmo usando uma fila

Busca em largura - Implementação

```
void busca_larg(int v) {
    // Criacao e inicializacao do vetor marcado
    queue<int> fila;
    marcado[v] = 1;
    fila.push(v);
    while (!fila.empty()) {
        int w = fila.front();
        fila.pop();
        for (auto u : listas_adj[w])
            if (marcado[u] == 0) {
                marcado[u] = 1;
                fila.push(u);
            }
    }
}
```

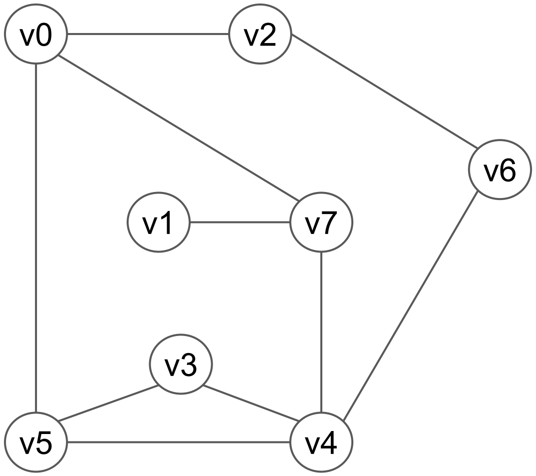


Busca em largura

- Usando o algoritmo de busca em largura, podemos determinar a distância entre o **vértice inicial da busca** e cada um dos vértices do grafo

Busca em largura - Implementação

```
void busca_larg(int v, int dist[]) {
    // Criacao e inicializacao do vetor marcado
    // Inicializacao do vetor dist
    queue<int> fila;
    marcado[v] = 1;
    dist[v] = 0;
    fila.push(v);
    while (!fila.empty()) {
        int w = fila.front();
        fila.pop();
        for (auto u : listas_adj[w])
            if (marcado[u] == 0) {
                marcado[u] = 1;
                dist[u] = dist[w] + 1;
                fila.push(u);
            }
    }
}
```



Conteúdo

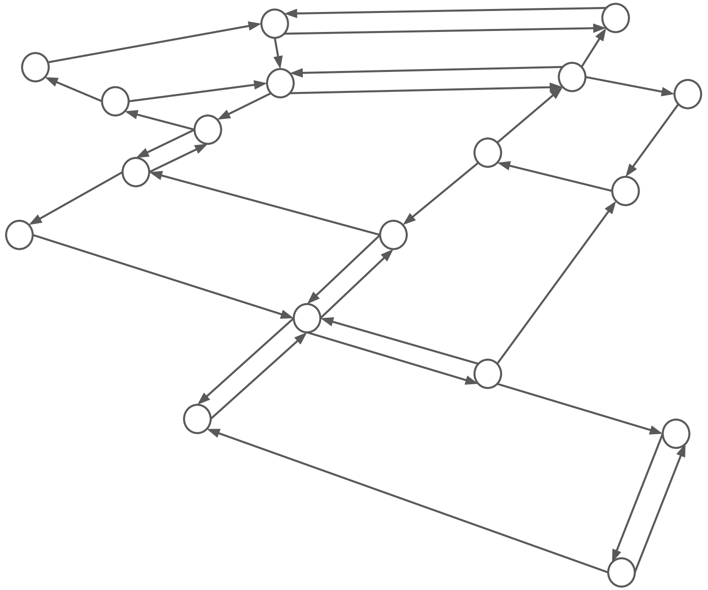
- Grafos - Conceitos básicos
- Representação computacional
- Busca em profundidade
- Busca em largura
- **Grafos dirigidos - Conceitos básicos**
- Grafos dirigidos - Representação computacional
- Grafos dirigidos - Busca em profundidade e em largura
- Referências

Motivação

- Em várias situações que podemos modelar com grafos, faz sentido considerarmos que as arestas têm uma **direção** (ou **orientação** ou **sentido**)
- Exemplo:
    - Temos um mapa de vias (ruas ou rodovias) e estamos interessados nos caminhos que podemos percorrer neste mapa
    - Uma via que conecta um ponto *x* a um ponto *y* pode ter apenas a mão de *x* para *y*, apenas a mão de *y* para *x* ou ambas as mãos
    - Podemos representar este mapa como um grafo onde cada **aresta** tem uma direção e representa **uma mão de uma via**

Motivação

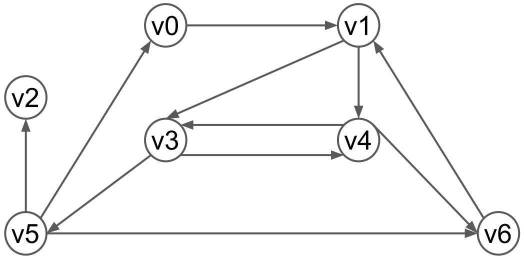
- Em várias situações que podemos modelar com grafos, faz sentido considerarmos que as arestas têm uma **direção** (ou **orientação** ou **sentido**)
- Exemplo:
  - Temos um mapa de vias (ruas ou rodovias) e estamos interessados nos caminhos que podemos percorrer neste mapa
  - Uma via que conecta um ponto *x* a um ponto *y* pode ter apenas a mão de *x* para *y*, apenas a mão de *y* para *x* ou ambas as mãos
  - Podemos representar este mapa como um grafo onde cada **aresta** tem uma direção e representa **uma mão de uma via**



Grafo dirigido – Digrafo

- Um **grafo dirigido** ou **digrafo** G é um par ordenado (V, E) composto por
  - um conjunto de **vértices** V e
  - um conjunto de **arestas** E, sendo cada aresta um par ordenado (v_i, v_j) de vértices de G
    - note que (v_i, v_j) ≠ (v_j, v_i)

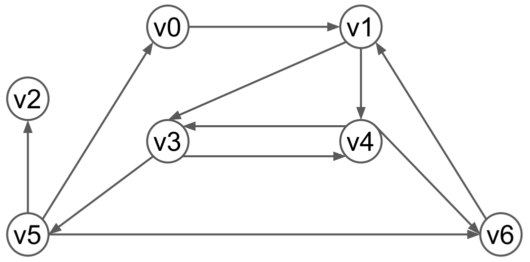
- Exemplo:
  - G = (V, E), onde
    - $V = \{ v_0, v_1, v_2, v_3, v_4, v_5, v_6 \}$ e
    - $E = \{ (v_0, v_1), (v_1, v_3), (v_1, v_4),$
      $(v_3, v_4), (v_3, v_5), (v_4, v_3),$
      $(v_4, v_6), (v_5, v_0), (v_5, v_2),$
      $(v_5, v_6), (v_6, v_1) \}$



Grafo dirigido – Digrafo

- Um **grafo dirigido** ou **digrafo** G é um par ordenado (V, E) composto por
  - um conjunto de **vértices** V e
  - um conjunto de **arestas** E, sendo cada aresta um par ordenado (v_i, v_j) de vértices de G
    - note que (v_i, v_j) ≠ (v_j, v_i);
    - denominamos v_i a **cauda** da aresta e v_j a **cabeça** da aresta

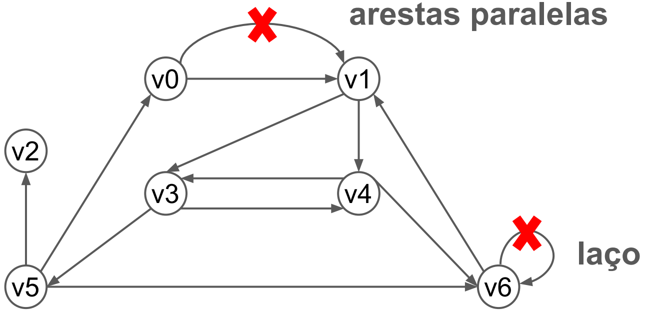
- Exemplo:
  - G = (V, E), onde
    - $V = \{ v_0, v_1, v_2, v_3, v_4, v_5, v_6 \}$ e
    - $E = \{ (v_0, v_1), (v_1, v_3), (v_1, v_4),$
      $(v_3, v_4), (v_3, v_5), (v_4, v_3),$
      $(v_4, v_6), (v_5, v_0), (v_5, v_2),$
      $(v_5, v_6), (v_6, v_1) \}$



Digrafo (simples)

- Em um digrafo **simples**,
  - não podem existir duas ou mais arestas com a mesma cauda e a mesma cabeça e
  - não podem existir arestas que conectam um vértice a ele mesmo

- Exemplo:



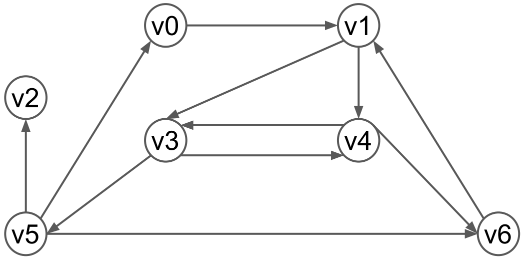
- A não ser que seja dito o contrário, os digrafos que vamos considerar são simples

Vizinhança

- Por simplicidade, também denotamos uma aresta (v_i , v_j) como $v_i v_j$
- Dada uma aresta $v_i v_j$, os vértices v_i e v_j são os **extremos** desta aresta

- Se $v_i v_j$ é uma aresta de um digrafo G, então
  - a aresta $v_i v_j$ **sai** de v_i e **entra** em v_j ,
  - v_i é **vizinho de entrada** de v_j em G e
  - v_j é **vizinho de saída** de v_i em G

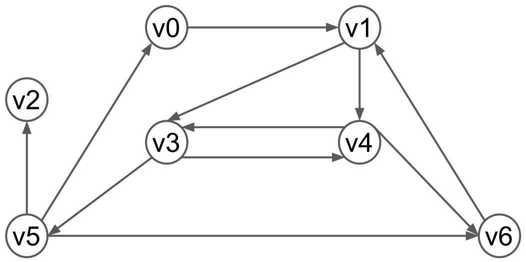
- Exemplo:
  - No digrafo ao lado, v_1 é vizinho de saída de v_0 e v_0 é vizinho de entrada de v_1. Os vizinhos de saída de v_5 são v_0, v_2 e v_6. A aresta $v_1 v_4$ sai de v_1 e entra em v_4



Passeio

- Um **passeio** em um digrafo *G* é uma sequência de vértices $v_{i0}v_{i1}\dots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho de saída em *G* do seu antecessor

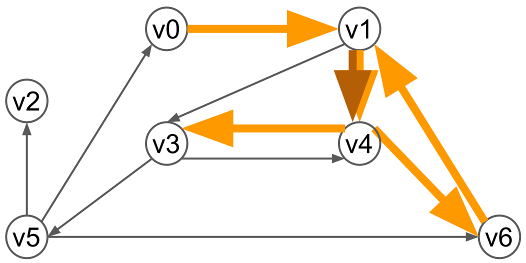
- Exemplo:
  - A sequência $v_0v_1v_4v_6v_1v_4v_3$ é um passeio no digrafo ao lado



Passeio

- Um **passeio** em um digrafo *G* é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de *G* tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho de saída em *G* do seu antecessor

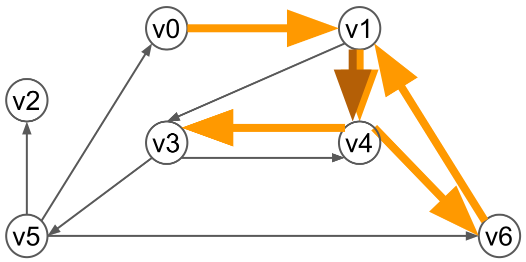
- Exemplo:
  - A sequência $v_0v_1v_4v_6v_1v_4v_3$ é um passeio no digrafo ao lado



Passeio

- Um **passeio** em um digrafo G é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de G tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho de saída em G do seu antecessor

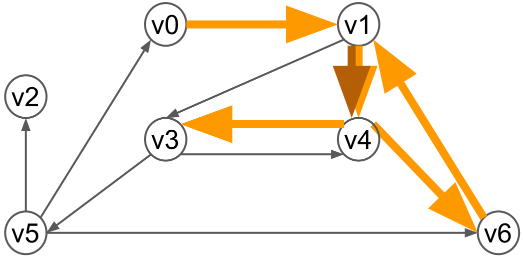
- Exemplo:
  - A sequência $v_0v_1v_4v_6v_1v_4v_3$ é um passeio no digrafo ao lado



- Em um passeio, especificamos os vértices, mas as arestas envolvidas também estão implicitamente especificadas
- Por isso, podemos nos referir às **arestas de um passeio**

Passeio

- Um **passeio** em um digrafo G é uma sequência de vértices $v_{i0}v_{i1}\ldots v_{ik}$ de G tal que, com exceção do primeiro vértice, cada vértice da sequência é vizinho de saída em G do seu antecessor

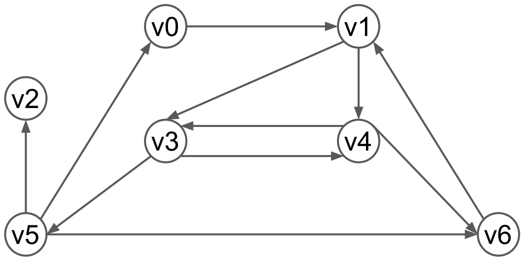


- Chamamos um passeio $v_{i0}v_{i1}\ldots v_{ik-1}v_{ik}$ de um $v_{i0}v_{ik}$-passeio e dizemos que
  - v_{i0} e v_{ik} são os **extremos** do passeio;
  - v_{i0} é a **origem** do passeio e v_{ik} é o **destino** do passeio;
  - $v_{i1}, \ldots, v_{ik-1}$ são os **vértices internos** do passeio;
  - o **comprimento** do passeio é k, ou seja, a quantidade de arestas percorridas e
  - o passeio é **fechado** se $v_{i0} = v_{ik}$ e é **aberto** caso contrário

Caminho

- Um **caminho** em um digrafo *G* é um passeio em *G* onde não existem vértices repetidos

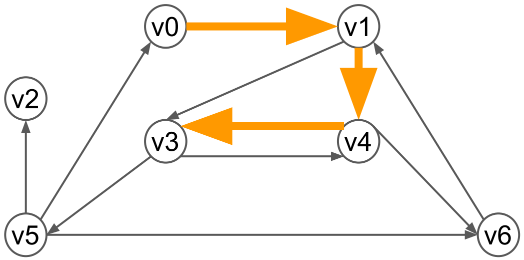
- Exemplo:
    - A sequência $v_0v_1v_4v_3$ é um caminho no digrafo ao lado



Caminho

- Um **caminho** em um digrafo *G* é um passeio em *G* onde não existem vértices repetidos

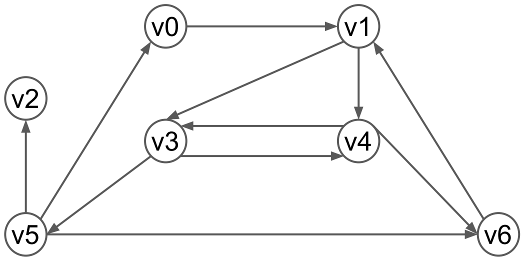
- Exemplo:
  - A sequência $v_0v_1v_4v_3$ é um caminho no digrafo ao lado



Caminho

- Um **caminho** em um digrafo G é um passeio em G onde não existem vértices repetidos

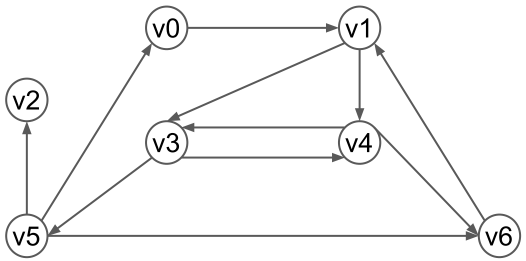
- Exemplo:
  - A sequência $v_0v_1v_6$
    não é um caminho no digrafo ao lado



Ciclo

- Um **ciclo** em um digrafo *G* é um passeio fechado em *G*, com comprimento maior ou igual a 1 e onde não existem vértices internos repetidos

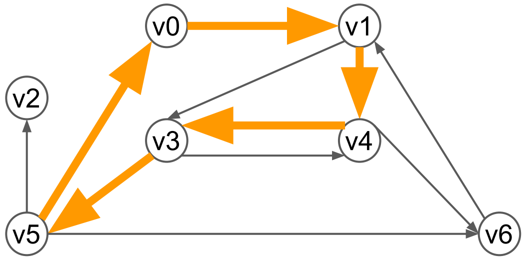
- Exemplo:
  - A sequência $v_0v_1v_4v_3v_5v_0$ é um ciclo no digrafo ao lado



Ciclo

- Um **ciclo** em um digrafo *G* é um passeio fechado em *G*, com comprimento maior ou igual a 1 e onde não existem vértices internos repetidos

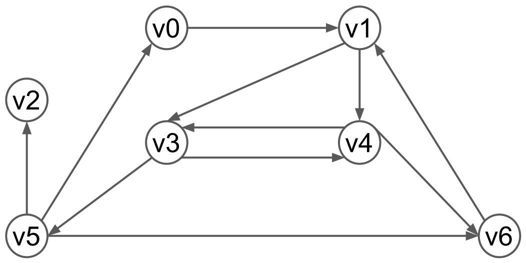
- Exemplo:
  - A sequência $v_0v_1v_4v_3v_5v_0$ é um ciclo no digrafo ao lado



Ciclo

- Um **ciclo** em um digrafo *G* é um passeio fechado em *G*, com comprimento maior ou igual a 1 e onde não existem vértices internos repetidos

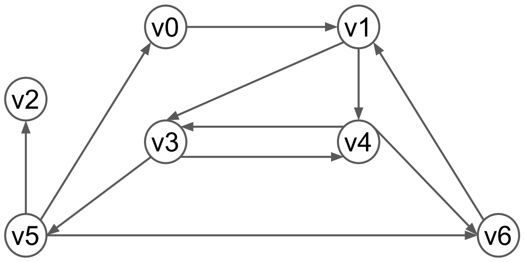
- Exemplo:
    - A sequência $v_1v_4v_3v_1$
      não é um ciclo no digrafo ao lado



Distância

- A **distância** de um vértice v_i para um vértice v_j em um digrafo G, denotada por $d(v_i, v_j)$, é
  - o menor comprimento de um v_iv_j-caminho em G ou
  - ∞ (infinita) caso não exista um v_iv_j-caminho em G
- Note que, em geral, $d(v_i, v_j) \neq d(v_j, v_i)$

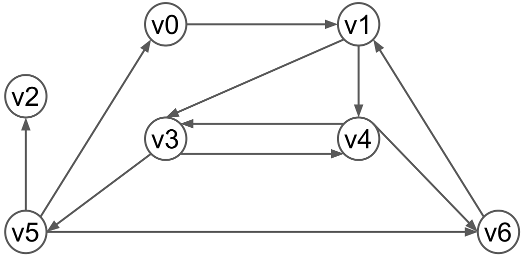
- Exemplo:
  - No digrafo ao lado,
    - $d(v_0, v_4) = 2$ e $d(v_4, v_0) = 3$,
    - $d(v_5, v_6) = 1$ e $d(v_6, v_5) = 3$,
    - $d(v_4, v_4) = 0$ e
    - $d(v_3, v_2) = 2$ e $d(v_2, v_3) = \infty$



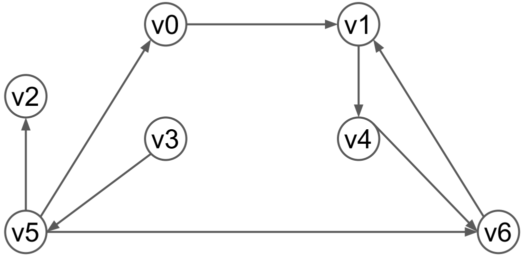
Subgrafo

- Um **subgrafo** de um digrafo *G* é um digrafo *H* tal que
  - $V(H) \subseteq V(G)$ e
  - $E(H) \subseteq E(G)$
- Exemplo:

G



H



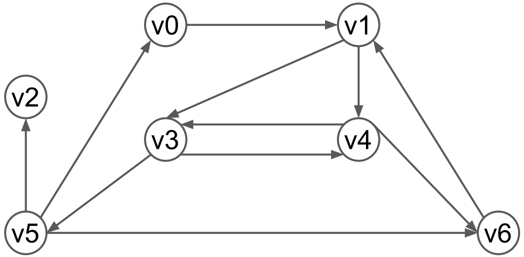
  - $V(G) = \{ v_0, v_1, v_2, v_3, v_4, v_5, v_6 \}$ e
  - $E(G) = \{ (v_0, v_1), (v_1, v_3), (v_1, v_4), (v_3, v_4), (v_3, v_5), (v_4, v_3), (v_4, v_6), (v_5, v_0), (v_5, v_2), (v_5, v_6), (v_6, v_1) \}$

  - $V(H) = \{ v_0, v_1, v_2, v_3, v_4, v_5, v_6 \}$ e
  - $E(H) = \{ (v_0, v_1), (v_1, v_4), (v_3, v_5), (v_4, v_6), (v_5, v_0), (v_5, v_2), (v_5, v_6), (v_6, v_1) \}$

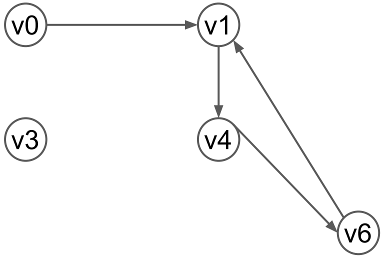
Subgrafo

- Um **subgrafo** de um digrafo G é um digrafo H tal que
  - $V(H) \subseteq V(G)$ e
  - $E(H) \subseteq E(G)$
- Exemplo:

G



H



- $V(G) = \{ v_0, v_1, v_2, v_3, v_4, v_5, v_6 \}$ e
- $E(G) = \{ (v_0, v_1), (v_1, v_3), (v_1, v_4), (v_3, v_4), (v_3, v_5), (v_4, v_3), (v_4, v_6), (v_5, v_0), (v_5, v_2), (v_5, v_6), (v_6, v_1) \}$

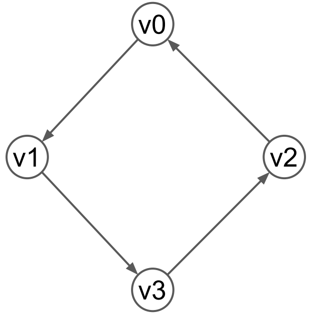
- $V(H) = \{ v_0, v_1, v_3, v_4, v_6 \}$ e
- $E(H) = \{ (v_0, v_1), (v_1, v_4), (v_4, v_6), (v_6, v_1) \}$

Conexidade (forte)

- Um digrafo G é **fortemente conexo** se, para todo par de vértices v_i , v_j de G, existe em G um v_iv_j-caminho (um caminho cuja origem é v_i e cujo destino é v_j) e um v_jv_i-caminho (um caminho cuja origem é v_j e cujo destino é v_i)

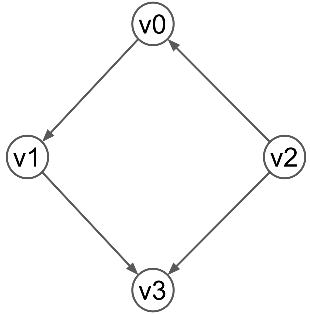
- Exemplo:

G_1



Digrafo fortemente conexo

G_2



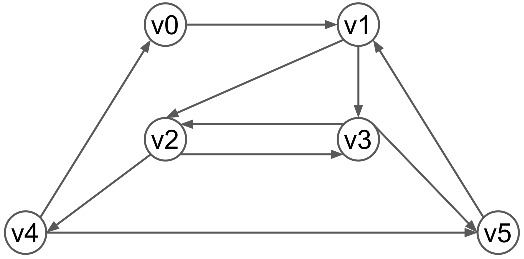
~~Digrafo fortemente conexo~~

Conexidade (forte)

- Um digrafo G é **fortemente conexo** se, para todo par de vértices v_i , v_j de G, existe em G um v_iv_j-caminho (um caminho cuja origem é v_i e cujo destino é v_j) e um v_jv_i-caminho (um caminho cuja origem é v_j e cujo destino é v_i)

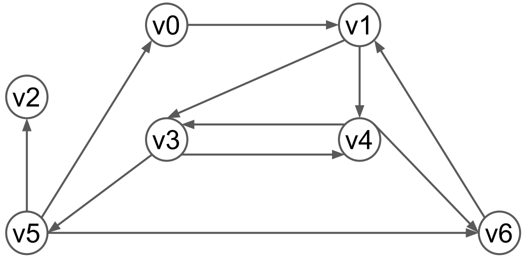
- Exemplo:

G_1



Digrafo fortemente conexo

G_2



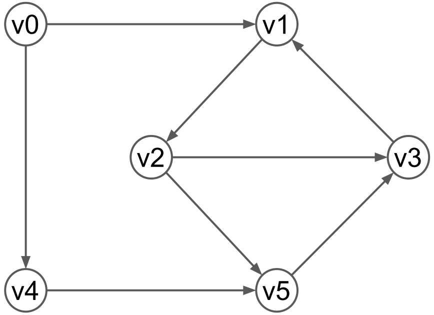
~~Digrafo fortemente conexo~~

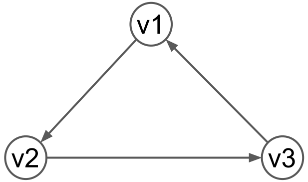
Conexidade (forte)

- Um **subgrafo fortemente conexo** de um digrafo *G* é um subgrafo de *G* que é fortemente conexo

- Exemplo:

G





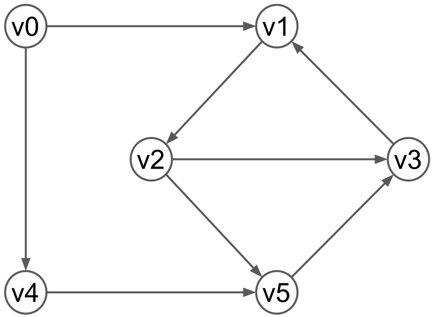
Subgrafo fortemente conexo

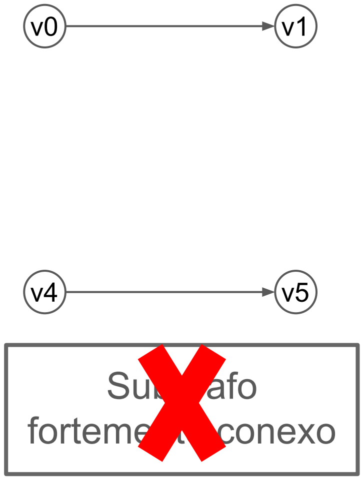
Conexidade (forte)

- Um **subgrafo fortemente conexo** de um digrafo *G* é um subgrafo de *G* que é fortemente conexo

- Exemplo:

G



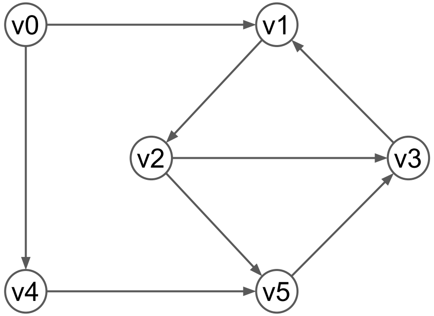


Conexidade (forte)

- Um **subgrafo fortemente conexo** de um digrafo *G* é um subgrafo de *G* que é fortemente conexo

- Exemplo:

G



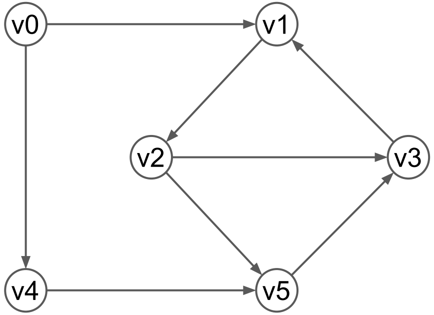
~~Subgrafo fortemente conexo~~

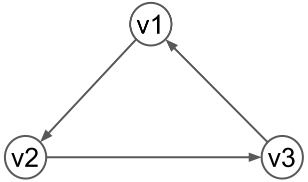
Conexidade (forte)

- Um **subgrafo fortemente conexo maximal** de um digrafo *G* é um subgrafo fortemente conexo de *G* que não está contido em outro subgrafo fortemente conexo de *G*

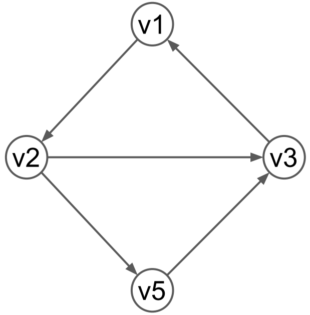
- Exemplo:

G





~~Subgrafo fortemente conexo maximal~~



Subgrafo fortemente conexo maximal

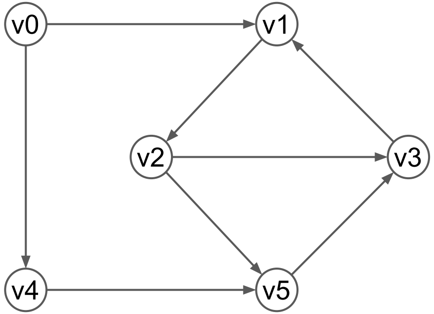
Conexidade (forte)

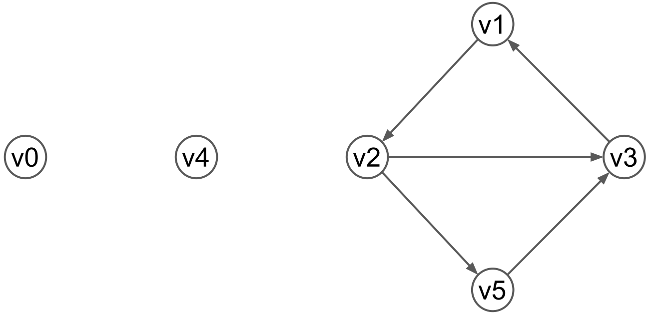
- As **componentes fortemente conexas** de um digrafo *G* são os subgrafos fortemente conexos maximais de *G*

- Exemplo:

Componentes fortemente conexas de *G*

G





3 componentes fortemente conexas

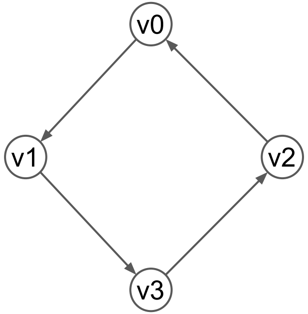
Conexidade (forte)

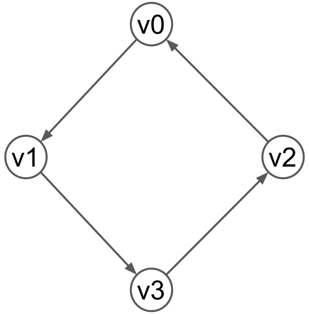
- As **componentes fortemente conexas** de um digrafo *G* são os subgrafos fortemente conexos maximais de *G*

- Exemplo:

Componentes fortemente conexas de *G*

G





1 componente fortemente conexa

Conexidade (forte)

- As **componentes fortemente conexas** de um digrafo *G* são os subgrafos fortemente conexos maximais de *G*

- Um **grafo fortemente conexo** (com pelo menos um vértice) tem exatamente **uma componente fortemente conexa**

Conteúdo

- Grafos - Conceitos básicos
- Representação computacional
- Busca em profundidade
- Busca em largura
- Grafos dirigidos - Conceitos básicos
- **Grafos dirigidos - Representação computacional**
- Grafos dirigidos - Busca em profundidade e em largura
- Referências

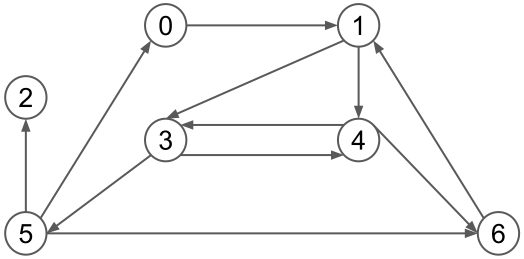
Representação computacional

- Anteriormente, vimos duas formas comuns de representar computacionalmente um grafo não-dirigido: matriz de adjacências e listas de adjacência
- A seguir, veremos formas equivalentes de representar computacionalmente um digrafo

Matriz de adjacências

- A representação de um digrafo *G* como uma **matriz de adjacências** consiste em uma matriz de |*V*(*G*)| linhas, com índices 0, 1, …, |*V*(*G*)| - 1, e de |*V*(*G*)| colunas, com índices 0, 1, …, |*V*(*G*)| - 1, tal que a célula (*i*, *j*) da matriz é igual a
  - 1 se i j é uma aresta de *G*
  - 0 caso contrário

G



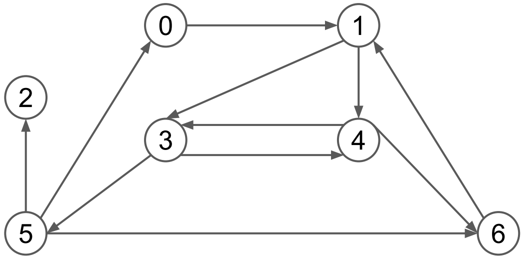
Matriz de adjacências de *G*

	0	1	2	3	4	5	6
0	0	1	0	0	0	0	0
1	0	0	0	1	1	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0
4	0	0	0	1	0	0	1
5	1	0	1	0	0	0	1
6	0	1	0	0	0	0	0

Matriz de adjacências

- Observações:
  - Não é possível representar arestas paralelas
  - Para digrafos simples, todas as células da diagonal principal da matriz são iguais a 0
  - Uma aresta *i j* é representada por **apenas uma** célula da matriz: (*i*, *j*) – a célula (*j*, *i*) representa uma aresta diferente, a aresta *j i*

G

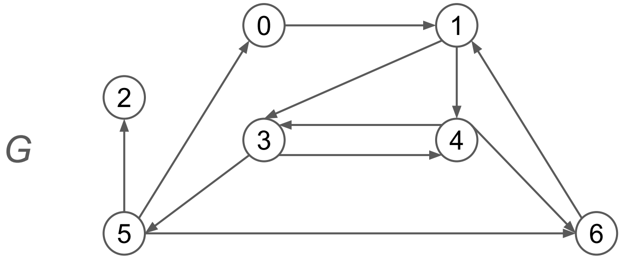


Matriz de adjacências de *G*

	0	1	2	3	4	5	6
0	0	1	0	0	0	0	0
1	0	0	0	1	1	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0
4	0	0	0	1	0	0	1
5	1	0	1	0	0	0	1
6	0	1	0	0	0	0	0

Matriz de adjacências

- Observações:
    - Em geral, a matriz não é simétrica em relação à diagonal principal



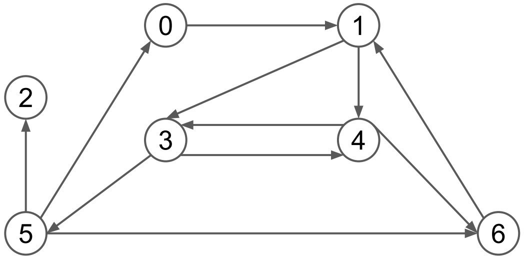
Matriz de adjacências de *G*

	0	1	2	3	4	5	6
0	0	1	0	0	0	0	0
1	0	0	0	1	1	0	0
2	0	0	0	0	0	0	0
3	0	0	0	0	1	1	0
4	0	0	0	1	0	0	1
5	1	0	1	0	0	0	1
6	0	1	0	0	0	0	0

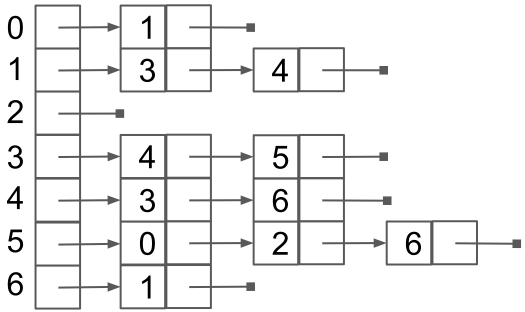
Listas de adjacência

- A representação de um digrafo *G* como **listas de adjacência** consiste em um vetor de $|V(G)|$ elementos, com índices 0, 1, …, $|V(G)| - 1$, tal que o elemento *i* do vetor armazena uma lista com os vizinhos de saída do vértice *i* em *G*

G



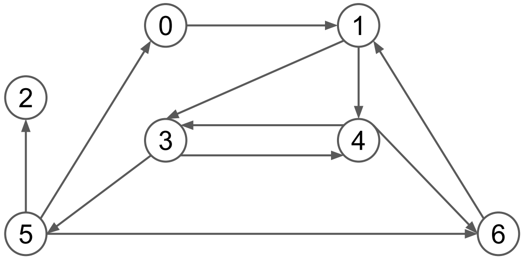
Listas de adjacência de *G*



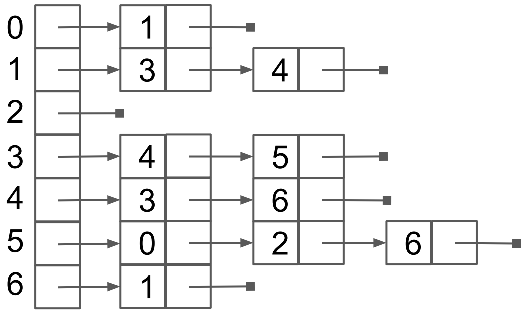
Listas de adjacência

- Observações:
  - Uma aresta *i j* é representada em **apenas uma** lista de adjacência: o vértice *j* está na lista do vértice *i*

G



Listas de adjacência de *G*



Conteúdo

- Grafos - Conceitos básicos
- Representação computacional
- Busca em profundidade
- Busca em largura
- Grafos dirigidos - Conceitos básicos
- Grafos dirigidos - Representação computacional
- **Grafos dirigidos - Busca em profundidade e em largura**
- Referências

Busca em profundidade e busca em largura

- Os algoritmos de busca em profundidade e em largura para digrafos são definidos da mesma forma que para grafos não-dirigidos com uma adaptação: os vizinhos considerados nos algoritmos são sempre **vizinhos de saída**

Aplicação

- Vimos anteriormente um algoritmo para determinar o número de componentes conexas de um grafo não-dirigido G
- Como podemos fazer para determinar o número de componentes fortemente conexas de um digrafo G?

Aplicação

- Como podemos fazer para determinar o número de componentes fortemente conexas de um digrafo *G*?
- Ideia:
  1. Faça *i* = 0
  2. Enquanto houver vértices não visitados no digrafo *G*:
  3. Execute o algoritmo de busca em profundidade no digrafo *G* começando por um vértice não visitado; quando um vértice *v* e seus vizinhos de saída tiverem sido visitados, faça *fin*(*v*) = *i* e *i* = *i* + 1
  4. Construa o digrafo *G'* dado pelo digrafo *G* com as direções das arestas de *G* invertidas
  5. Enquanto houver vértices não visitados no digrafo *G'*:
  6. Execute o algoritmo de busca em profundidade no digrafo *G'* começando por um vértice não visitado *v* para o qual *fin*(*v*) seja máximo
- Cada execução do algoritmo de busca em profundidade realizada nos Passos 5-6 determina uma componente fortemente conexa do digrafo *G*

Aplicação

- Como podemos fazer para determinar o número de componentes fortemente conexas de um digrafo *G*?
- Ver
    - a Seção 22.5 do livro
      Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. 3rd. ed. MIT Press, 2009.
    - a Seção 19.8 do livro
      Sedgewick, R. Algorithms in C++ – Part 5. Graph Algorithms. 3rd. ed. Addison-Wesley, 2002.

Referências

- Esta apresentação é baseada nos seguintes materiais:

1. Capítulo 22 do livro
   Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. 3rd. ed. MIT Press, 2009.
2. Capítulos 17 a 19 do livro
   Sedgewick, R. Algorithms in C++ – Part 5. Graph Algorithms. 3rd. ed. Addison-Wesley, 2002.