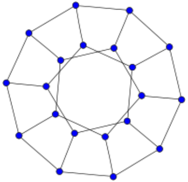
Algoritmos e Estrutura de Dados

Grafos

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Faculdade de Engenharia



Roteiro

Objetivos

Esta aula tem como objetivos:

1. Formalizar a definição de grafo,
2. Introduzir os conceitos de isomorfismo, caminho, circuito, subgrafo, conexão, componente e grafo aleatório.

Motivação

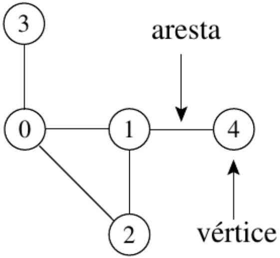
- Muitas aplicações em computação necessitam considerar um conjunto de conexões entre pares de objetos:
  - Existe um caminho para ir de um objeto a outro seguindo as conexões?
  - Qual é a menor distância entre dois objetos?
  - Quantos outros objetos podem ser alcançados a partir de um determinado objeto?
- Existe um tipo abstrato chamado **grafo** que é usado para modelar tais situações.

Aplicações

- Alguns exemplos de problemas práticos que podem ser resolvidos através de uma modelagem em grafos:
  - Ajudar máquinas de busca a localizar informação relevante na Web.
  - Descobrir as melhores combinações entre vagas de emprego e pessoas com enviaram currículo.
  - Descobrir qual é o roteiro mais curto para visitar as principais cidades de uma região turística.

Introdução

- Um grafo é um par(V,A) em que V é um conjunto de vértices e A é o conjunto de arestas.



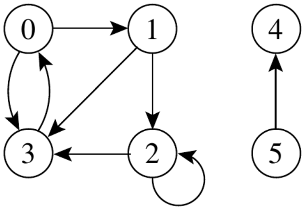
Introdução

- Uma aresta (a, b) será denotada simplesmente por ab ou por ba.
- Dizemos que a aresta ab incide em a e em b, e que a e b são as pontas da aresta.
- Se ab é uma aresta, dizemos que os vértices a e b são vizinhos ou adjacentes.

Definições

Grafos Direcionados

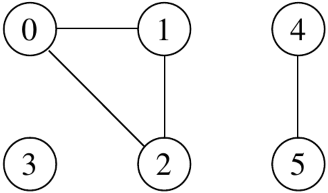
- Um grafo **direcionado** G é um par (V, A), onde V é um conjunto finito de vértices e A é uma relação binária em V.
  - Uma aresta (a, b) sai do vértice a e entra no vértice b. O vértice a é adjacente ao vértice b.
  - Podem existir arestas de um vértice para ele mesmo, chamadas de self-loops.



Definições

Grafos não-direcionados

- Um grafo **não-direcionado** G é um par (V, A), onde o conjunto de arestas A é constituído de pares de vértices não ordenados.
  - As arestas (a, b) e (b, a) são consideradas como uma única aresta.
  - A relação de adjacência é simétrica.
  - Self-loops não são permitidos.

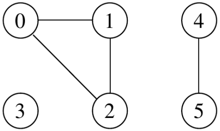


Definições

Grau de um Vértice

Em grafos não-direcionados:

- O **grau** de um vértice é o número de arestas que incidem nele.
- Um vértice de grau zero é chamado de vértice isolado ou não-conectado.
- Ex.: O vértice 1 tem grau 2 e o vértice 3 é isolado.

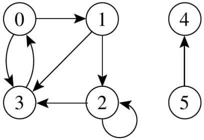


Definições

Grau de um Vértice

Em grafos direcionados

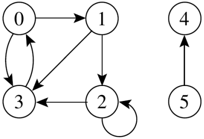
- O **grau** de um vértice é o número de arestas que saem dele (grau de saída) mais o número de arestas que chegam nele (grau de entrada).
- Ex.: O vértice 2 tem grau de entrada 2 e grau de saída 2, portanto possui grau 4.



Definições

Caminho entre Vértices

- Um **caminho** de comprimento k de um vértice x a um vértice y em um grafo $G = (V, A)$ é uma sequência de vértices $(v_0, v_1, v_2, ..., v_k)$ tal que $x = v_0$ e $y = v_k$, e $(v_{i-1}, v_i) \in A$ para $i = 1, 2, ..., k$.
- O comprimento de um caminho é o número de arestas nele, isto é, o caminho contém os vértices $v_0, v_1, v_2, ..., v_k$ e as arestas $(v_0, v_1), (v_1, v_2), ..., (v_{k-1}, v_k)$.
- Se existir um caminho c de x a y então y é alcançável a partir de x via c.
- Um caminho é simples se todos os vértices do caminho são distintos.
- Ex.: O caminho $(0, 1, 2, 3)$ é simples e tem comprimento 3. O caminho $(1, 3, 0, 3)$ não é simples.

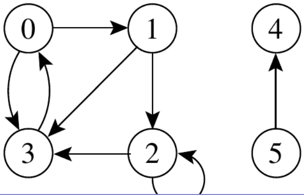


Definições

Ciclos

Em um grafo direcionado:

- Um caminho $(v_0, v_1, ..., v_k)$ forma um **ciclo** se $v_0 = v_k$ e o caminho contém pelo menos uma aresta.
- O ciclo é simples se os vértices $v_1, v_2, ..., v_k$ são distintos.
- O self-loop é um ciclo de tamanho 1.
- Dois caminhos $(v_0, v_1, ..., v_k)$ e $(v'_0, v'_1, ..., v'_k)$ formam o mesmo ciclo se existir um inteiro j tal que $v'_i = v_{(i+j) \bmod k}$ para $i = 0, 1, ..., k-1$.
- Ex.: O caminho $(0, 1, 2, 3, 0)$ forma um ciclo.
- O caminho $(0, 1, 3, 0)$ forma o mesmo ciclo que os caminhos $(1, 3, 0, 1)$ e $(3, 0, 1, 3)$.

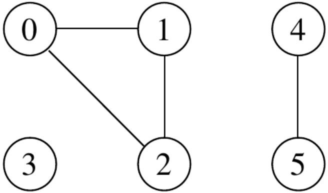


Definições

Ciclos

Em um grafo não-direcionado:

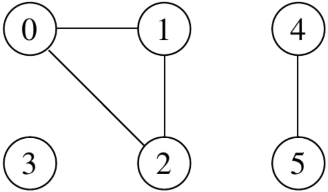
- Um caminho $(v_0, v_1, ..., v_k)$ forma um **ciclo** se $v_0 = v_k$ e o caminho contém pelo menos três arestas.
- O ciclo é simples se os vértices $v_1, v_2, ..., v_k$ são distintos.
- Ex.: O caminho $(0, 1, 2, 0)$ é um ciclo.



Definições

Componentes Conectados

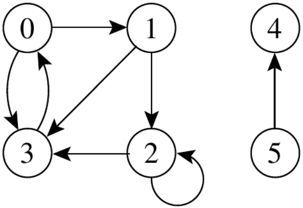
- Um grafo não-direcionado é **conectado** se cada par de vértices está conectado por um caminho.
- Os componentes conectados são as porções conectadas de um grafo.
- Um grafo não-direcionado é conectado se ele tem exatamente um componente conectado.
- Ex.: Os componentes são: $\{0, 1, 2\}, \{4, 5\}$ e 3.



Definições

Componentes Fortemente Conectados

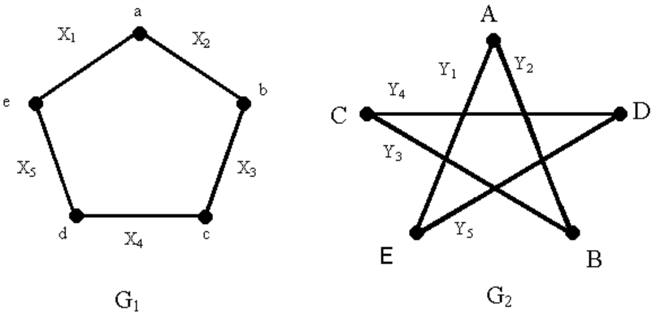
- Um grafo direcionado $G = (V, A)$ é **fortemente conectado** se dois vértices quaisquer são alcançáveis um a partir do outro.
- Os componentes fortemente conectados de um grafo direcionado são conjuntos de vértices sob a relação "são mutuamente alcançáveis".
- Um grafo direcionado fortemente conectado tem apenas um componente fortemente conectado.
- Ex.: $\{0, 1, 2, 3\}$, $\{4\}$ e $\{5\}$ são os componentes fortemente conectados, $\{4, 5\}$ não o é, pois o vértice 5 não é alcançável a partir do vértice 4.



Definições

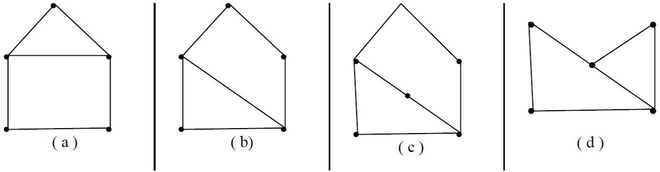
Isomorfismo

- $G = (V, A)$ e $G' = (V', A')$ são **isomorfos** se existir uma bijeção $f : V \to V'$ tal que $(u, v) \in A$ se e somente se $(f(u), f(v)) \in A'$.
- Em outras palavras, é possível re-rotular os vértices de G para serem rótulos de G' mantendo as arestas correspondentes em G e G'.



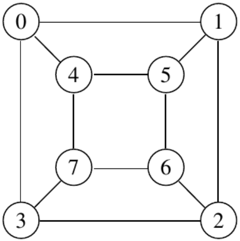
Definições

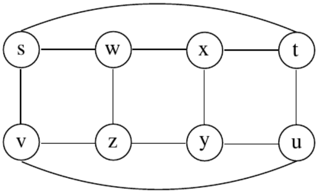
Isomorfismo



Definições

Isomorfismo

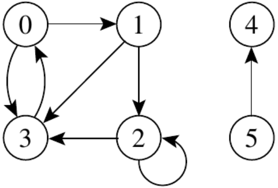


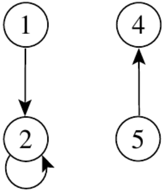


Definições

Subgrafos

- Um grafo $G' = (V', A')$ é um **subgrafo** de $G = (V, A)$ se $V' \subseteq V$ e $A' \subseteq A$.
- Dado um conjunto $V' \subseteq V$, o subgrafo induzido por V' é o grafo $G' = (V', A')$, onde $A' = \{(u, v) \in A | u, v \in V'\}$.
- Ex.: Subgrafo induzido pelo conjunto de vértices $\{1, 2, 4, 5\}$.

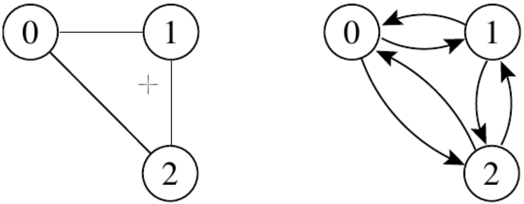




Definições

Versão Direcionada de um Grafo Não-Direcionado

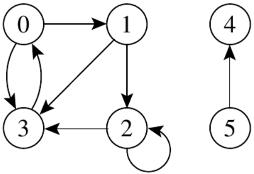
- A **versão direcionada de um grafo não-direcionado** $G = (V, A)$ é um grafo direcionado $G' = (V', A')$ onde $(u, v) \in A'$ se e somente se $(u, v) \in A$.
- Cada aresta não-direcionada (u, v) em G é substituída por duas arestas direcionadas (u, v) e (v, u).
- Em um grafo direcionado, um vizinho de um vértice u é qualquer vértice adjacente a u na versão não-direcionada de G.

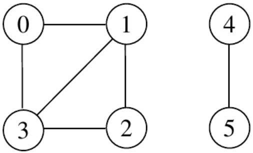


Definições

Versão não-direcionada de um Grafo Direcionado

- A **versão não-direcionada de um grafo direcionado** $G = (V, A)$ é um grafo não-direcionado $G' = (V', A')$ onde $(u, v) \in A'$ se e somente se $u \neq v$ e $(u, v) \in A$.
- A versão não-direcionada contém as arestas de G sem a direção e sem os self-loops.
- Em um grafo não-direcionado, u e v são vizinhos se eles são adjacentes.

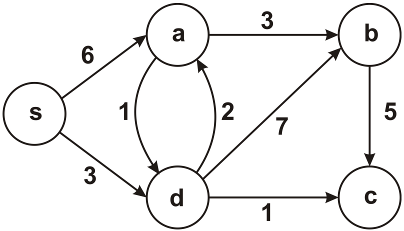




Definições

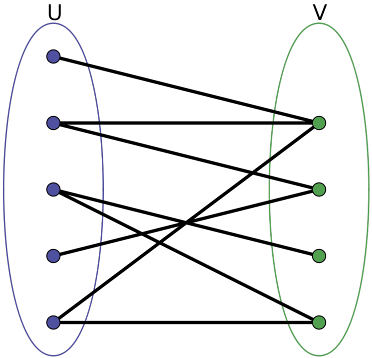
Grafo Ponderado

- Um **grafo ponderado** possui pesos associados às arestas.



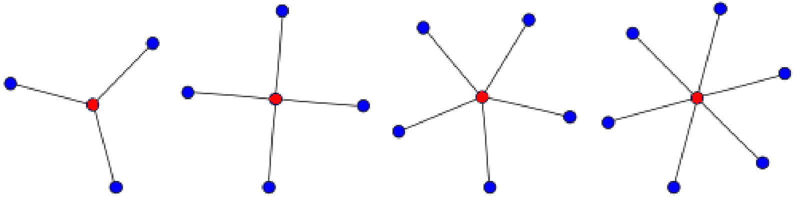
Definições
Grafo Bipartido

- **Grafo Bipartido**: grafo não-direcionado $G = (V, A)$ no qual V *pode ser particionado em dois conjuntos* V_1 e V_2 tal que $(u, v) \in A$ implica que $u \in V_1$ e $v \in V_2$ ou $u \in V_2$ e $v \in V_1$ (todas as arestas ligam os dois conjuntos V_1 e V_2).



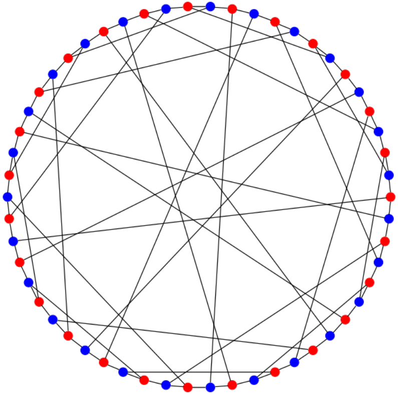
Definições

Grafo Bipartido



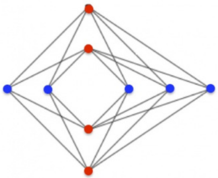
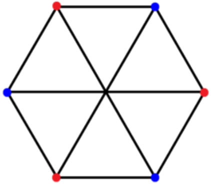
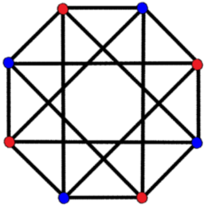
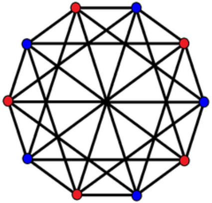
Definições

Grafo Bipartido



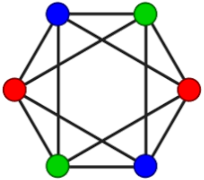
Definições

Grafo Bipartido

Definições

Grafo Tripartido



Definições

Hipergrafo

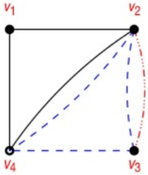
Hipergrafo é uma generalização de um grafo, com suas *arestas ligando quaisquer quantidades positivas de vértices*.

- *Definição*: Um hipergrafo $H(V, F)$ é definido pelo par de conjunto V e F, onde:
    - V é um conjunto não vazio de vértices;
    - F é um conjunto que representa uma "família" e partes não vazias de V.

Definições
Hipergrafo

Um **hipergrafo** é um grafo não dirigido em que cada aresta conecta um número arbitrário de vértices.

- Seja, por exemplo, o grafo $H(V, F)$ dado por:
  - $V = \{v_1, v_2, v_3, v_4\}$
  - $F = \{\{v_1, v_2, v_4\}, \{v_2, v_3, v_4\}, \{v_2, v_3\}\}$



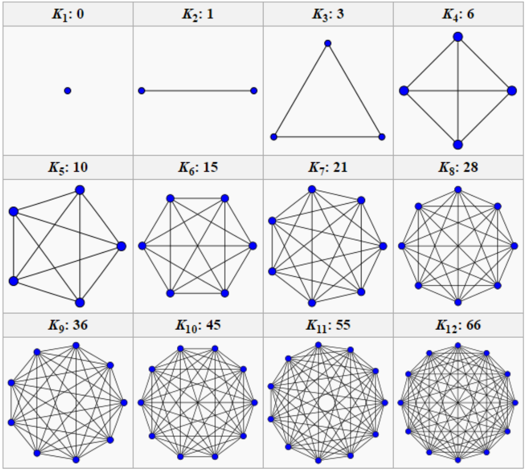
Definições

Grafos Completos

- Um **grafo completo** é um grafo não-direcionado no qual todos os pares de vértices são adjacentes.
- Possui $(|V|^2 - |V|)/2 = |V|(|V| - 1)/2$ arestas, pois do total de $|V|^2$ pares possíveis de vértices devemos subtrair $|V|$ self-loops e dividir por 2 (cada aresta ligando dois vértices é contada duas vezes).
- O número total de grafos diferentes com $|V|$ vértices é $2^{|V|(|V|-1)/2}$ (número de maneiras diferentes de escolher um subconjunto a partir de $|V|(|V| - 1)/2$ possíveis arestas).

Definições

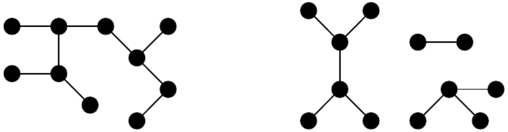
Grafos Completos



Definições

Árvores

- **Árvore livre**: grafo não-direcionado acíclico e conectado.
- É comum dizer apenas que o grafo é uma árvore omitindo o “livre”.
- **Floresta**: grafo não-direcionado acíclico, podendo ou não ser conectado.



Representação

As principais formas para representar grafos são:

- **Matriz de Adjacência**
- **Lista de Adjacência**

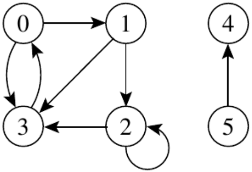
Representação

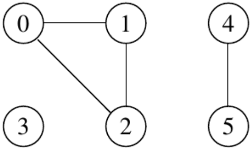
Matriz de Adjacência

- A **Matriz de Adjacência** de um grafo $G = (V, A)$ contendo n vértices é uma matriz $n \times n$ de bits, onde $A[i, j]$ é 1 (ou verdadeiro) se e somente se existe um arco do vértice i para o vértice j.
- Para grafos ponderados $A[i, j]$ contém o rótulo ou peso associado com a aresta e, neste caso, a matriz não é de bits.
- Se não existir uma aresta de i para j então é necessário utilizar um valor que não possa ser usado como rótulo ou peso.

Representação

Matriz de Adjacência





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

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

Matriz de Adjacência

Análise

- Deve ser utilizada para grafos densos, onde $|A|$ é próximo de $|V|^2$.
- Vantagens:
  - O tempo necessário para acessar um elemento é independente de $|V|$ ou $|A|$.
  - É muito útil para algoritmos em que necessitamos saber com rapidez se existe uma aresta ligando dois vértices.
- Desvantagens:
  - A maior desvantagem é que a matriz necessita $\omega(|V|^2)$ de espaço.
  - Ler ou examinar a matriz tem complexidade de tempo $O(|V|^2)$.

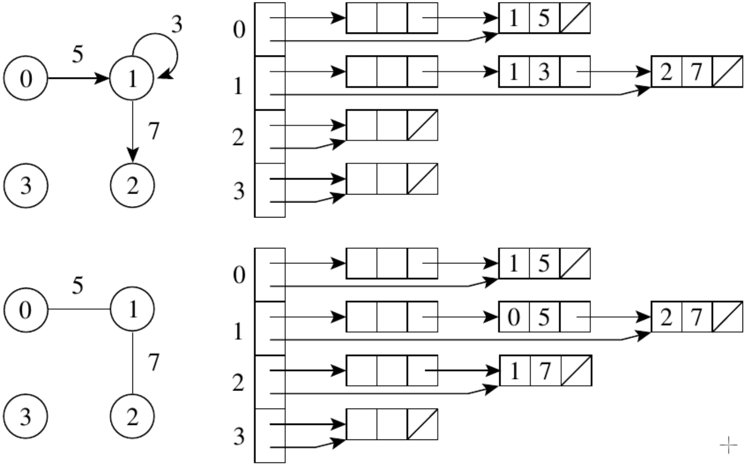
Representação

Lista de Adjacência

- A **Lista de Adjacência** consiste de um vetor, denominado Adj, contendo $|V|$ listas, uma para cada vértice de V.
- Para cada $u \in V$, Adj[u] contém todos os vértices de G adjacentes a u.
- Os vértices são armazenados de forma arbitrária na lista.
- Também pode ser utilizada para representar grafos dirigidos.

Representação

Lista de Adjacência



Lista de Adjacência
Análise

- Os vértices de uma lista de adjacência são em geral armazenados em uma ordem arbitrária.
- Indicada para grafos esparsos, onde $|A|$ é muito menor do que $|V|^2$.
- Vantagens:
  - É compacta e usualmente utilizada na maioria das aplicações.
  - Possui uma complexidade de espaço $O(|V| + |A|)$.
- Desvantagens:
  - É ineficiente para determinar se uma aresta está no grafo.
  - A principal desvantagem é que ela pode ter tempo $O(|V|)$ para determinar se existe uma aresta entre o vértice i e o vértice j, pois podem existir $O(|V|)$ vértices na lista de adjacentes do vértice i.

Percurso

- Fazer buscas em um grafo significa percorrer suas arestas sistematicamente, de modo a visitar seus vértices.
- As principais formas para percorrer grafos são:
  - **Busca em Largura** - em inglês *Breadth-First Search* ou BFS
  - **Busca em Profundidade** - em inglês *Depth-First Search* ou DFS

- A **Busca em Largura** é um dos algoritmos mais simples para exploração de um grafo.
  - Dados um grafo G = (V, E) e um vértice s, chamado de fonte, a busca em largura sistematicamente explora as arestas de G de maneira a visitar todos os vértices alcançáveis a partir de s.
- Expande a fronteira entre vértices descobertos e não-descobertos uniformemente através da largura da fronteira.
  - O algoritmo descobre todos os vértices a uma distância k do vértice de origem s antes de descobrir qualquer vértice a uma distância $k + 1$.
- O grafo pode ser direcionado ou não-direcionado.

Percurso

Busca em Largura

Esse algoritmo é base para muitos outros algoritmos, como por exemplo:

- Achar componentes conectados.
- Achar todos os nós contectados a apenas um nó.
- Achar o menor caminho entre um nó raiz e os outros nós do grafo (algoritmo de Dijsktra).
- Testar bipartição em grafos.

Busca em Largura

Algoritmo

Algoritmo 1: BFS

Entrada: Grafo $G = (V, A)$, vértice inicial s.

início

  Marque s como explorado.

  Imprima(s).

  Enfileire-o em Q.

  enquanto *(Q não estiver vazia)* **faça**

    Desenfileira o 1º vértice u em Q.

    para *cada vértice v vizinho de u* **faça**

      se *v não foi explorado* **então**

        Marque v como explorado.

        Imprima(u).

        Enfileira v em Q.

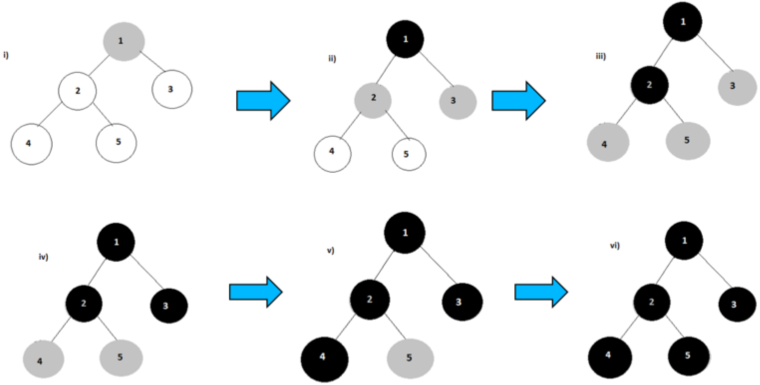
Percurso

Busca em Largura

Algoritmo:

- Para controlar a busca, o algoritmo da Busca em Largura pinta cada vértice na cor branca, cinza ou preto.
- Todos os vértices iniciam com a cor branca e podem, mais tarde, se tornar cinza e depois preto.
    - Branca: não visitado;
    - Cinza: visitado;
    - Preta: visitado e seus nós adjacentes visitados.

Percurso
Busca em Largura



Percurso

Busca em Largura

- Dado um nó inicial s, a busca em largura determina a distância (em número de arestas) de cada vértice atingível a partir de s.
  - Vértices a uma distância de $k + 1$ arestas de s só são visitados após todos os vértices a uma distância de k terem sido visitados.
- Para isso, armazena em cada vértice com a distância d do vértice inicial s.
- Também armazena em cada vértice o seu predecessor π, exceto o vértice inicial, que não possui predecessor.

Percurso

Busca em Largura

Algoritmo 2: BuscaEmLargura

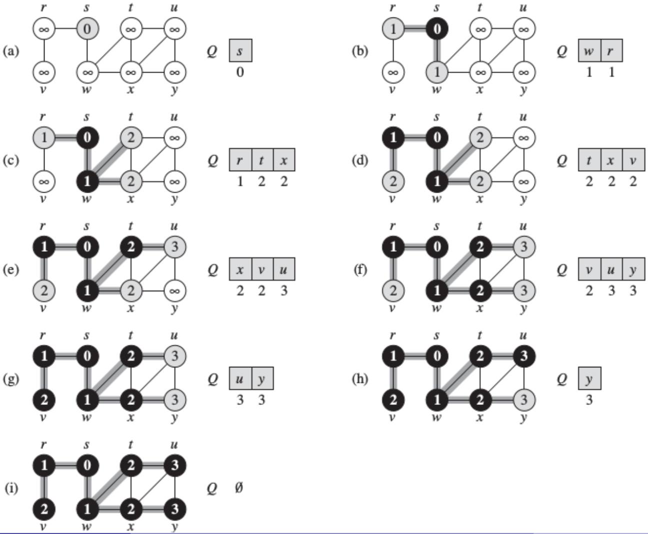
Entrada: Grafo $G = (V, A)$, vértice inicial s.
Saída: Percurso armazenado no campo "predecessor" presente em cada vértice $v \in V$.

início
    para *cada vértice* $v \in V$ **faça**
        v.$cor \leftarrow$ Branco
        v.$d \leftarrow \infty$
        v.$\pi \leftarrow$ NULL
    s.$cor \leftarrow$ Cinza
    s.$d \leftarrow 0$
    s.$\pi \leftarrow$ NULL
    CriaFilaVazia(Q)
    Enfileira(Q, s)
    enquanto $(Q \neq \emptyset)$ **faça**
        v $\leftarrow$ Desenfileira(Q)
        para *cada vértice* $u \in v.ListaAdj$ **faça**
            se $u.cor = Branco$ **então**
                u.cor $\leftarrow$ Cinza
                u.d $\leftarrow$ v.d + 1
                u.$\pi \leftarrow$ v
                Enfileira(Q, u)
        v.cor $\leftarrow$ Preto

Adaptado de Cormen et al. (2012).

Percurso

Busca em Largura



Busca em Largura

Análise

- Uma característica do BFS é que sua árvore de busca resultante corresponde ao *caminho mais curto* do vértice inicial a qualquer vértice do grafo.
- Cada vértice de V é colocado na fila Q no máximo uma vez: $O(V)$;
- A lista de adjacência de um vértice qualquer de u é percorrida somente quando o vértice é removido da fila;
- A soma de todas as listas de adjacentes é $O(A)$, então o tempo total gasto com as listas de adjacentes é $O(A)$;
- Enfileirar e desenfileirar tem custo $O(1)$;
- Complexidade: $O(V + A)$.

- Na **Busca em Profundidade**, a estratégia é buscar o vértice mais profundo no grafo sempre que possível:
  - As arestas são exploradas a partir do vértice v mais recentemente descoberto que ainda possui arestas não exploradas saindo dele.
- Quando todas as arestas adjacentes a v tiverem sido exploradas a busca anda para trás para explorar vértices que saem do vértice do qual v foi descoberto (*backtracking*).

Percurso

Busca em Profundidade

O algoritmo é a base para muitos outros algoritmos importantes, tais como:

- Verificação de grafos acíclicos,
- Ordenação topológica e
- Componentes fortemente conectados.
- Resolução de quebra-cabeças como labirinto.

Busca em Profundidade

Algoritmo

Algoritmo 3: DFS

Entrada: Grafo $G = (V, A)$, vértice inicial v.
Saída: Imprime o conteúdo de todos os vértices.

```
início
    Marque v como explorado.
    Imprima(v).
    para cada vértice u vizinho de v faça
        se u não foi explorado então
            DFS(G,u).
```

Adaptado de Oliveira (2011).

Busca em Profundidade

Algoritmo

Algoritmo 4: DFS-Não-Recursivo

Entrada: Grafo $G = (V, A)$, vértice inicial v.

Saída: Imprime o conteúdo de todos os vértices.

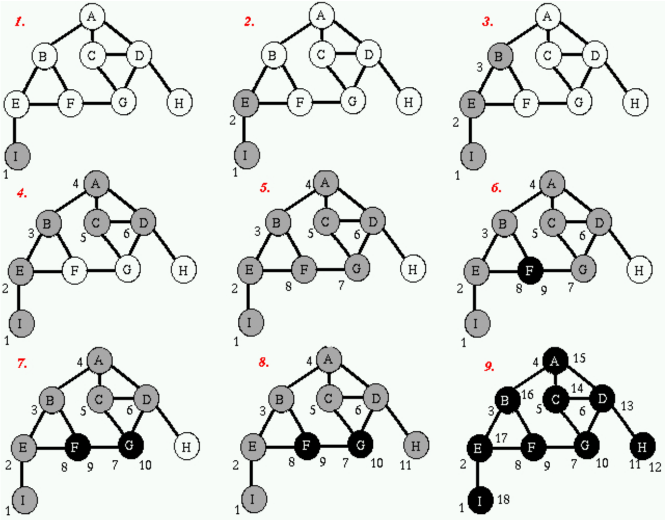
```
início
    Empilhe v na pilha S.
    enquanto S não estiver vazia faça
        Desempilhe v de S.
        Marque v como explorado.
        para cada vértice u vizinho de v faça
            se u não foi explorado então
                Marque u como explorado.
                Empilhe u em S.
        se v não foi explorado então
            Imprima(v).
```

Algoritmo

- Para controlar a busca, o algoritmo da Busca em Profundidade pinta cada vértice na cor branca, cinza ou preto.
- Todos os vértices iniciam com a cor branca e podem, mais tarde, se tornar cinza e depois preto.
  - Branca: não visitado;
  - Cinza: visitado;
  - Preta: visitado e seus nós adjacentes visitados.

Percurso

Busca em Profundidade



Algoritmo

- A busca em profundidade também marca cada vértice com um *timestamp*.
- Cada vértice tem dois *timestamp*:
  - v.d: indica o instante em que v foi visitado (pintado com cinza);
  - v.f: indica o instante em que a busca pelos vértices na lista de adjacência de v foi completada (pintado de preto).

Algoritmo 5: BuscaEmProfundidade

Entrada: Grafo $G = (V, A)$.
Saída: Percurso armazenado no campo "predecessor" presente em cada vértice $v \in V$.

```
início
    para cada vértice u ∈ V faça
        u.cor ← Branco
        u.π ← NULL
    tempo ← 0
    para cada vértice u ∈ V faça
        se u.cor = Branco então
            DFS-Visita(u)
```

Algoritmo 6: DFS-Visita

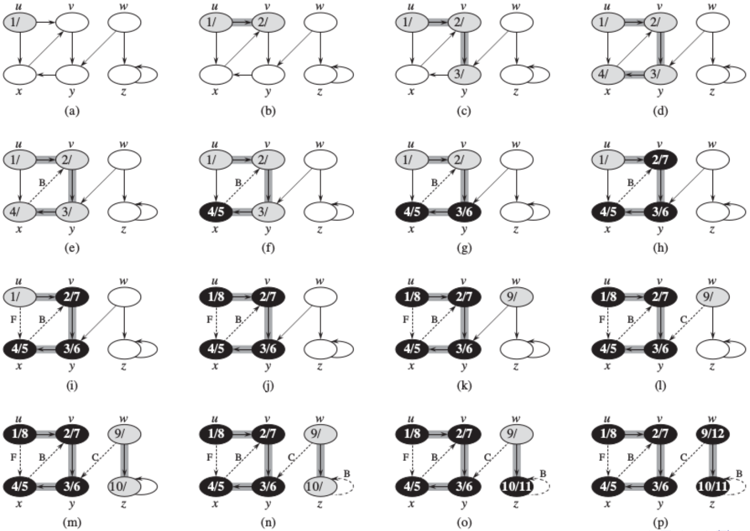
Entrada: Grafo $G = (V, A)$, vértice inicial s.

```
início
    tempo ← tempo + 1
    u.d ← tempo
    u.cor ← Cinza
    para cada vértice v ∈ u.ListaAdj faça
        se v.cor = Branco então
            v.π ← u
            DFS-Visita(G, v)
    u.cor ← Preto
    tempo ← tempo + 1
    u.f ← tempo
```

Adaptado de Cormen et al. (2012).

Percurso

Busca em Profundidade



Busca em Profundidade

Análise

- O procedimento DFS-Visita é chamado exatamente uma vez para cada vértice $u \in V$.
- Isso ocorre porque DFS-Visita é chamado apenas para vértices brancos e a primeira ação é pintar o vértice de cinza: $O(V)$.
- O loop principal de DFS-Visita (u) tem complexidade $O(A)$.
- Complexidade: $O(V + A)$.

Referências bibliográficas

CORMEN, T. H. et al. *Algoritmos: Teoria e Prática*. 3. ed. São Paulo: Campus, 2012. ISBN 978-0-262-03384-8.

OLIVEIRA, S. L. G. *Algoritmos e seus fundamentos*. 1. ed. Lavras: Editora UFLA, 2011.

Material Complementar

- Material IME
  - https://www.ime.usp.br/ pf/algoritmos_para_grafos/aulas/bfs.html
  - https://www.ime.usp.br/ pf/algoritmos_para_grafos/aulas/dfs.html
- Animação Breadth-First Search (Busca em Largura)
  - https://www.cs.usfca.edu/ galles/visualization/BFS.html
- Animação Depth-First Search (Busca em Profundidade)
  - https://www.cs.usfca.edu/ galles/visualization/DFS.html

Material Complementar

- Youtube
  - Estrutura de Dados Descomplicada - Busca em Largura
  - Estrutura de Dados Descomplicada - Busca em Profundidade
  - Univesp - Grafos
  - Univesp - Busca em Largura
  - Univesp - Busca em Profundidade