Universidade Federal do Pará
Instituto de Ciências Exatas e Naturais
Programa de Pós-Graduação em Ciência da Computação

GRAFOS

Caminhos em Grafos

Nelson Cruz Sampaio Neto
nelsonneto@ufpa.br

28 de setembro de 2017

Algoritmos de busca

- Um algoritmo de busca (ou varredura) é um algoritmo que segue sistematicamente as arestas e visita os vértices.
- Para justificar a palavra "busca", imagine que o algoritmo percorre o grafo buscando **todos** os vértices acessíveis a partir de um determinado vértice em questão.
- Há muitas maneiras de fazer essa busca. Cada estratégia é caracterizada pela ordem em que os vértices são visitados.
- Assim, a ordem de visita torna-se essencial se desejamos determinar outras propriedades além da mera característica de um determinado vértice ser alcançado a partir de outro.

Algoritmos de busca

- Principais algoritmos usados para caminhar em grafos:

  - **Busca em profundidade**: A idéia desse algoritmo é ir o mais fundo possível no grafo.

  - **Busca em largura**: A estratégia é visitar primeiramente todos os vizinhos inexplorados.

- Servem de base para outros algoritmos importantes, como ordenação topológica, caminho mais curto, verificação de grafos acíclicos, componentes fortemente conexos, etc.

Busca em profundidade

- 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 outros vértices que sucedem o vértice do qual **v** foi descoberto.

- Dois contadores são utilizados: um que marca o tempo de descoberta de **v**; e outro que indica o tempo de término do exame da lista de adjacências de **v**.

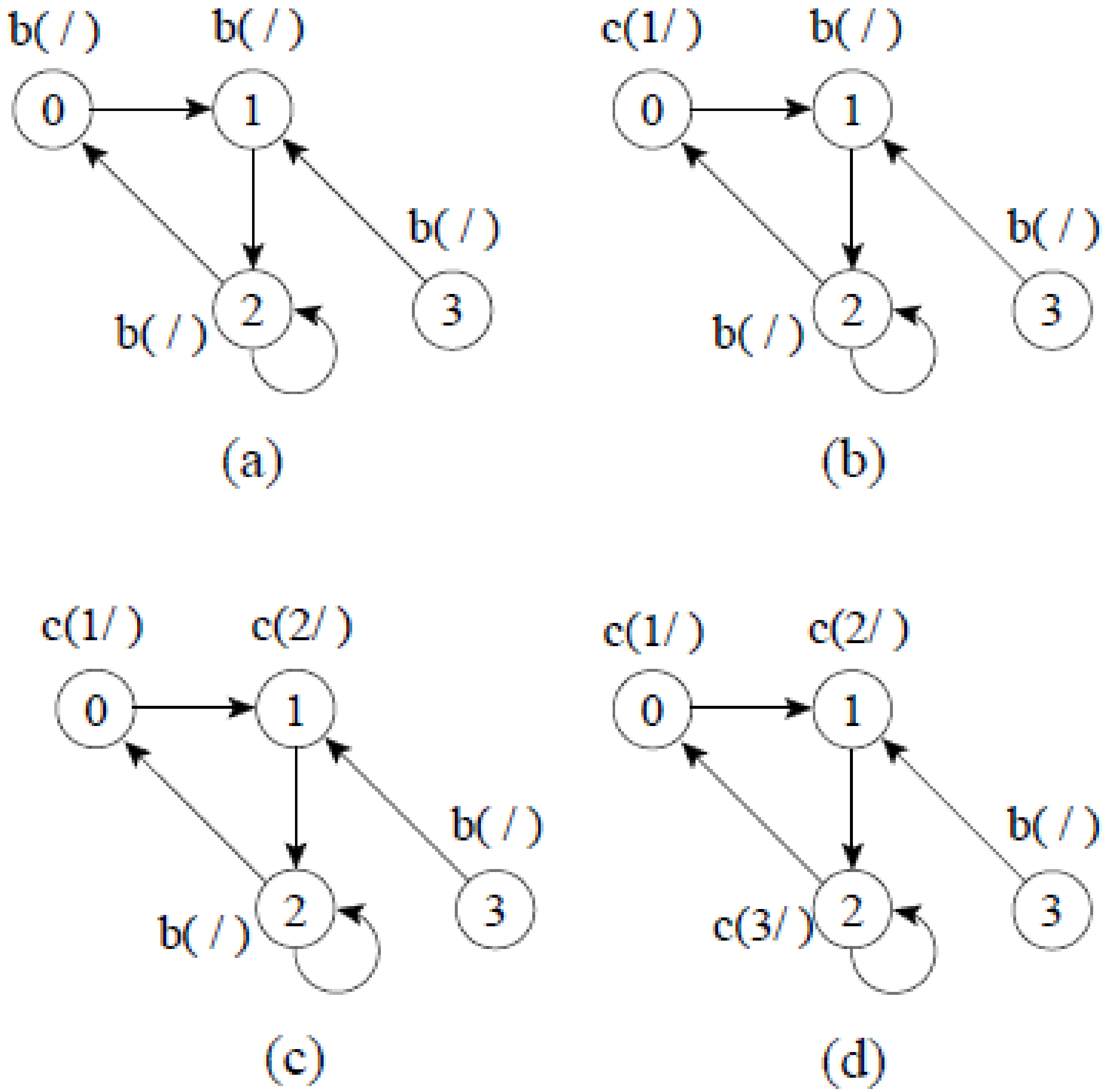
Algoritmo - DFS

```
DFS(G)
for ∀u ∈ V[G]  do
        cor[u] ← BRANCO
        π[u] ← NIL
tempo ← 0
for ∀u ∈ V[G] do
        if cor[u] = BRANCO then
                VisitaDFS(u)
```

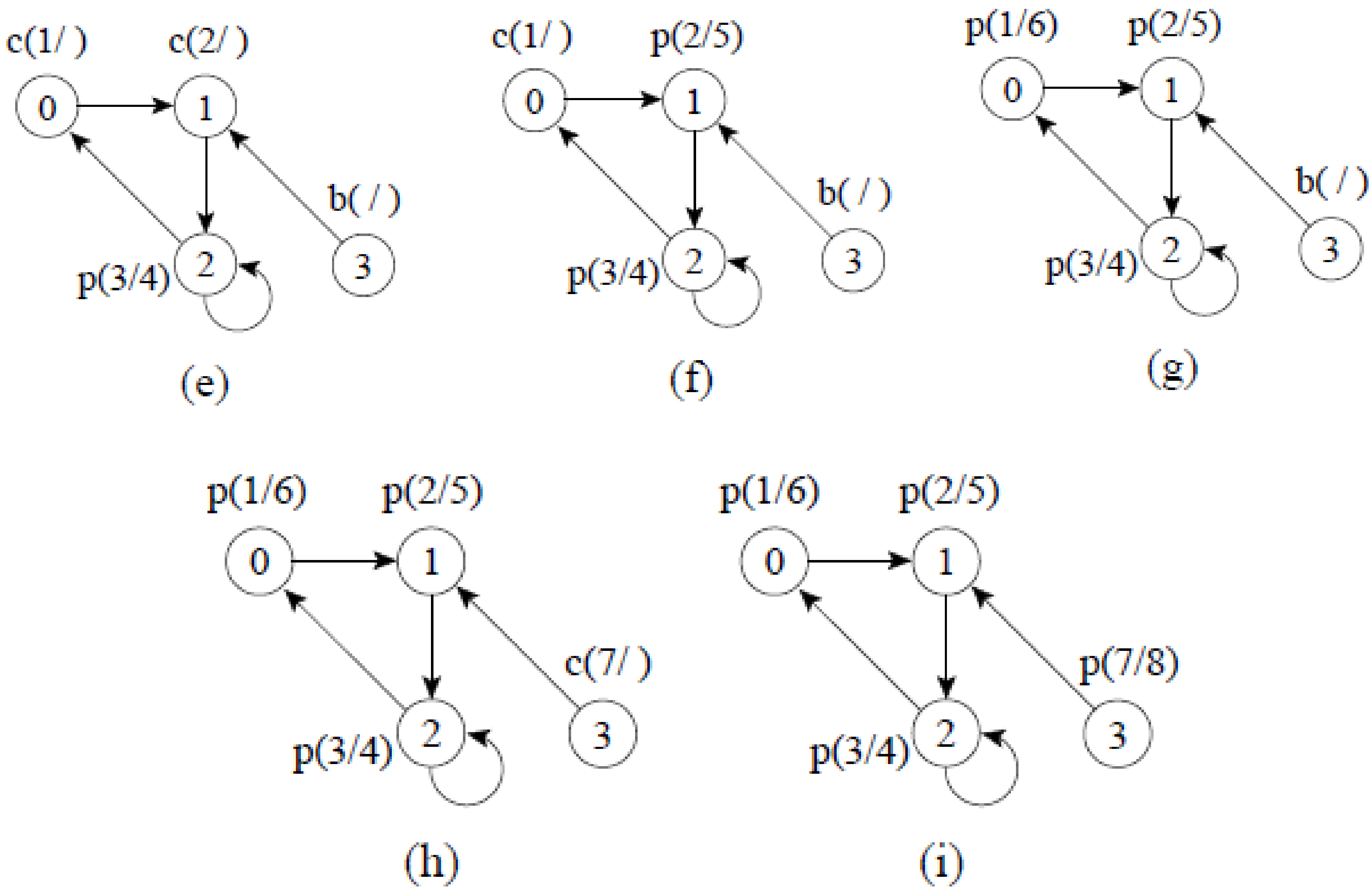
```
VisitaDFS(u)
cor[u] ← CINZA
d[u] ← tempo ← tempo + 1
for ∀v ∈ Adj[u] do
        if cor[v] = BRANCO then
                π[v] ← u
                VisitaDFS(v)
cor[u] ← PRETO
F[u] ← tempo ← tempo + 1
```

Pseudocódigo do Algoritmo de Busca em Profundidade.

Exemplo

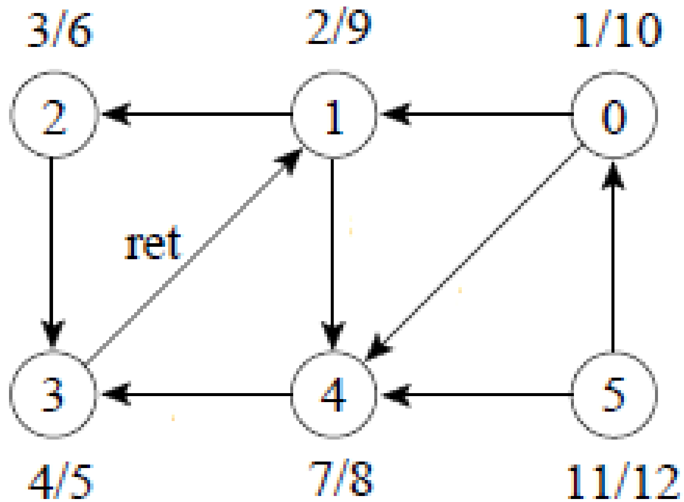


Exemplo


c(1/)
c(2/)
0
1
b(/)
p(3/4)
2
3
(e)
c(1/)
p(2/5)
0
1
b(/)
p(3/4)
2
3
(f)
p(1/6)
p(2/5)
0
1
b(/)
p(3/4)
2
3
(g)
p(1/6)
p(2/5)
0
1
c(7/)
p(3/4)
2
3
(h)
p(1/6)
p(2/5)
0
1
p(7/8)
p(3/4)
2
3
(i)

Considerações

- A complexidade total do método é ***O*(|V| + |A|)**.
- <u>Arestas de retorno</u>: conectam um vértice **u** com um antecessor **v** em uma busca em profundidade (inclui *self-loop).*
- Caso uma aresta de retorno seja encontrada durante a busca em profundidade, então o grafo é <u>cíclico</u>.



Ordenação Topológica

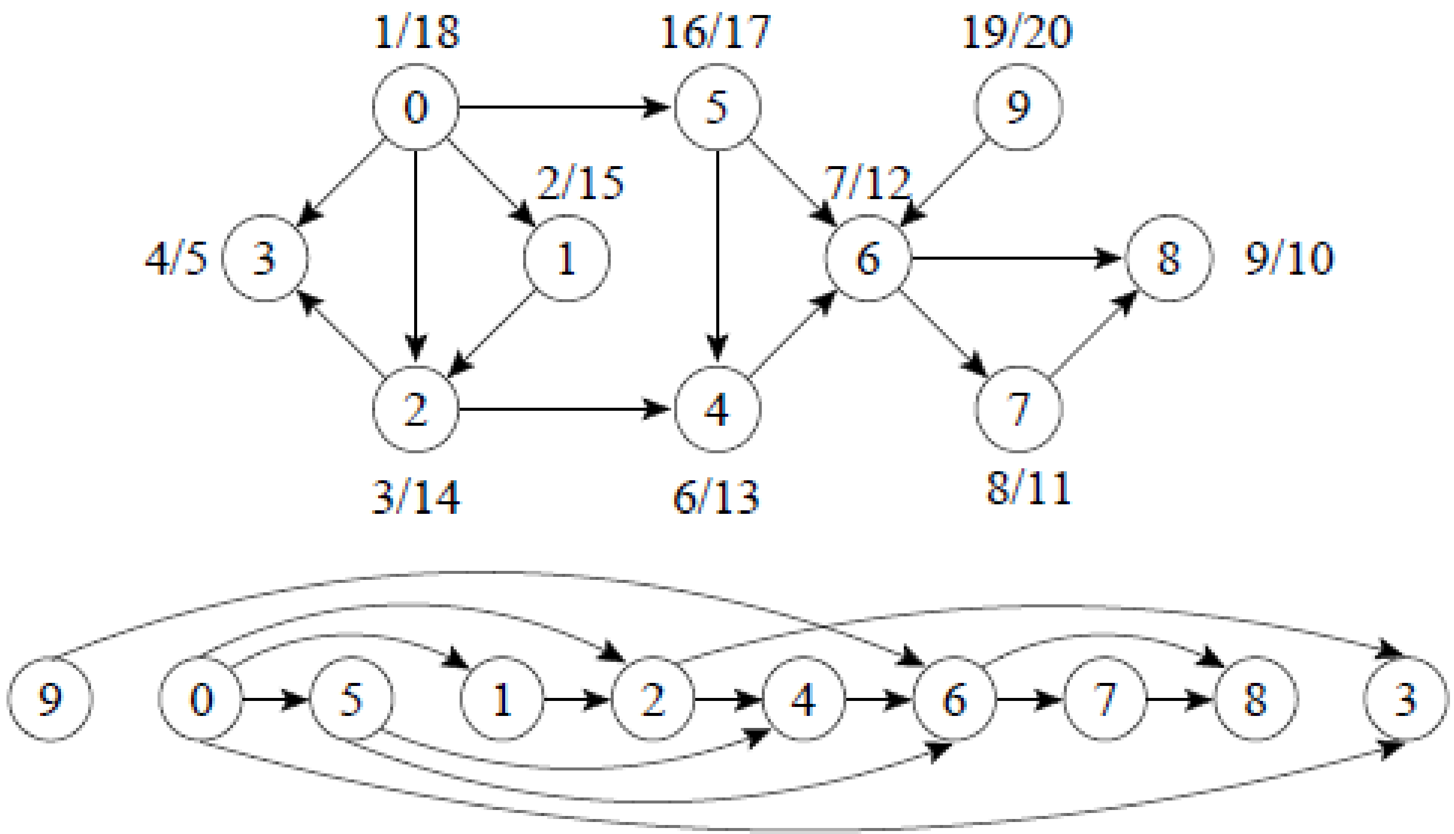
- Os grafos orientados e acíclicos são usados para indicar precedências entre eventos.

- Uma aresta orientada **(u, v)** indica que a atividade **u** tem que ser realizada antes da atividade **v**.

- Ordenação topológica: É uma ordenação linear dos vértices ao longo de uma linha horizontal de tal forma que todas as arestas estão direcionadas da esquerda para a direita.

- Como obter? Busca em profundidade. Ao término de cada vértice insira-o na frente de uma lista linear encadeada.

Exemplo


1/18
16/17
19/20
0
5
9
2/15
7/12
4/5
3
1
6
8
9/10
2
4
7
3/14
6/13
8/11
9
0
5
1
2
4
6
7
8
3

Componentes Fortemente Conexos

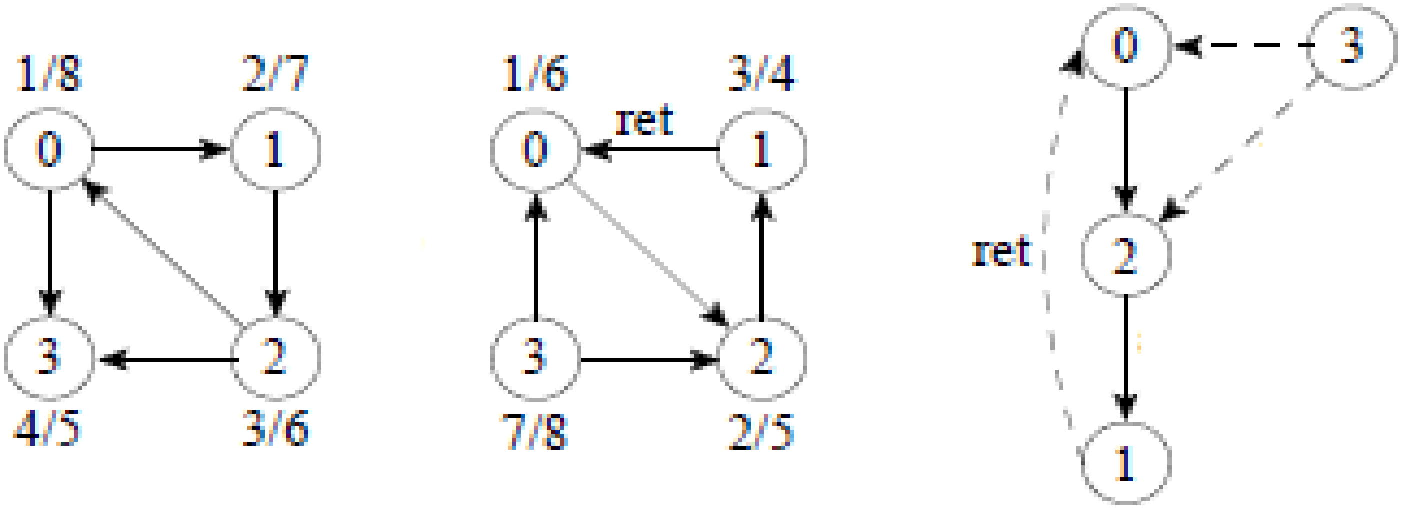
1. Aplicar a busca em profundidade no grafo **G** para obter os tempos de término para cada vértice **u**.

2. Obter o grafo transposto $\mathbf{G}^{\mathbf{T}}$.

3. Aplicar a busca em profundidade no grafo $\mathbf{G}^{\mathbf{T}}$, realizando a busca a partir do vértice de maior tempo obtido na linha 1. Se a busca em profundidade não alcançar todos os vértices, inicie uma nova busca em profundidade a partir do vértice de maior tempo dentre os vértices restantes.

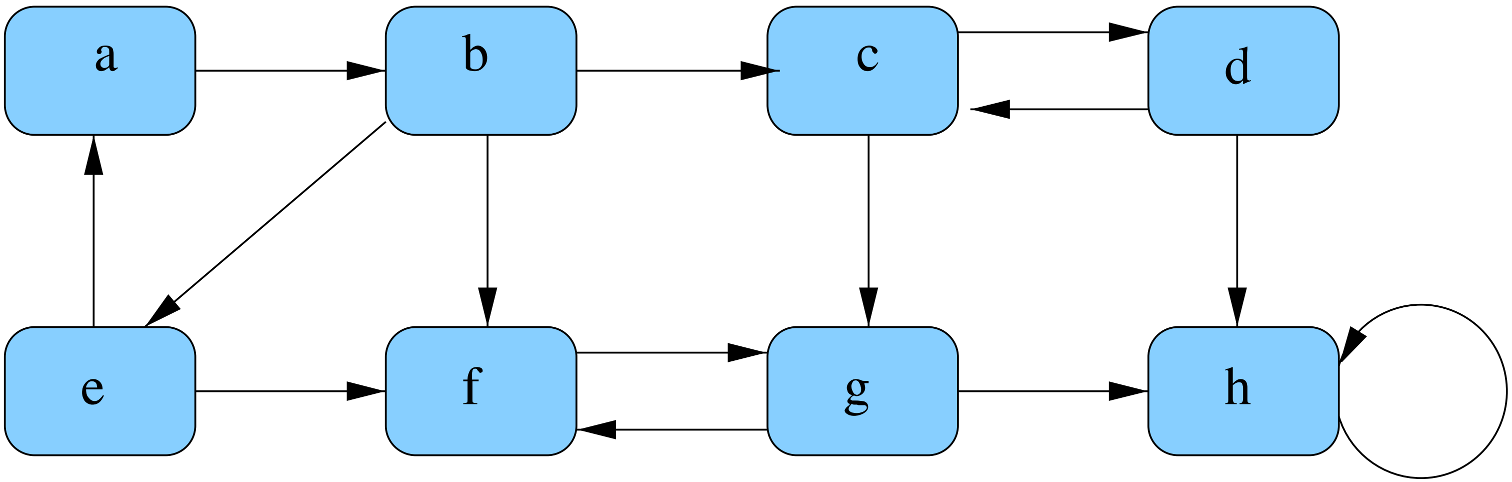
4. Retornar os vértices de cada subgrafo obtido na busca em profundidade na linha 3 como um componente fortemente conectado separado.

Exemplo



(a) (b) (c)

Exemplo



Busca em largura

- O algoritmo descobre todos os vértices a uma distância **k** do vértice origem antes de descobrir qualquer vértice que esteja a uma distância **k + 1**.

- A complexidade total do método é ***O*(|V| + |A|)**.

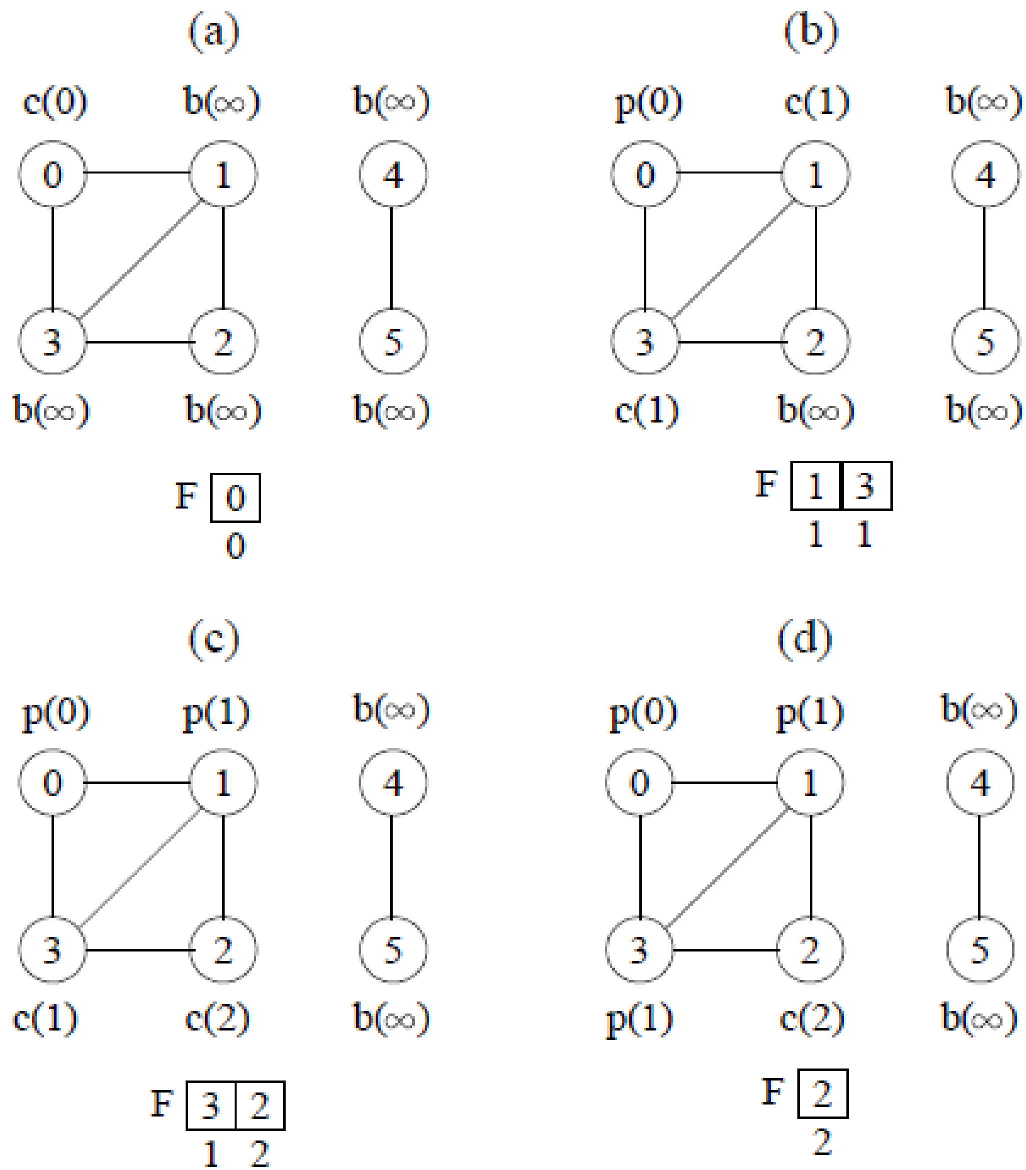
- A busca em largura obtém o <u>caminho mais curto</u> de **u** até **v**, o que não é possível através da busca em profundidade.

Algoritmo - BFS

```
BFS(G, s)
for ∀u ∈ V[G] – {s} do
        cor[u] ← BRANCO
        d[u] ← ∞
        π[u] ← NIL
cor[s] ← CINZA
d[s] ← 0
π[s] ← NIL
Q ← {s}
while Q ≠ Ø do
        u ← Desenfileira[Q]
        for ∀v ∈ Adj[u] do
                if cor[v] = BRANCO then
                        cor[v] ← CINZA
                        d[v] ← d[u] + 1
                        π[v] ← u
                        Enfileira(Q, v)
        cor[u] ← PRETO
```

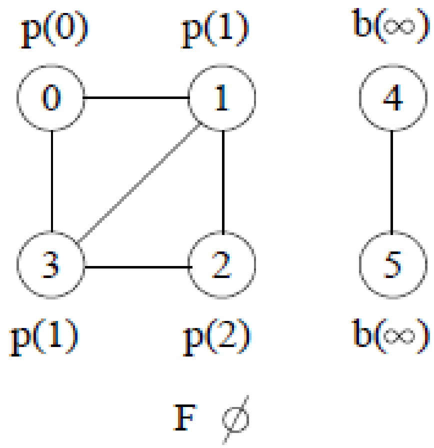
Pseudocódigo do Algoritmo de Busca em Largura.

Exemplo

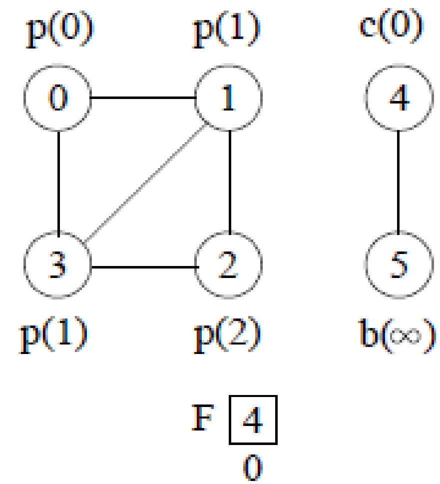

(a)
c(0) b(∞) b(∞)
0 1 4
3 2 5
b(∞) b(∞) b(∞)
F 0
0
(b)
p(0) c(1) b(∞)
0 1 4
3 2 5
c(1) b(∞) b(∞)
F 1 3
1 1
(c)
p(0) p(1) b(∞)
0 1 4
3 2 5
c(1) c(2) b(∞)
F 3 2
1 2
(d)
p(0) p(1) b(∞)
0 1 4
3 2 5
p(1) c(2) b(∞)
F 2
2

Exemplo

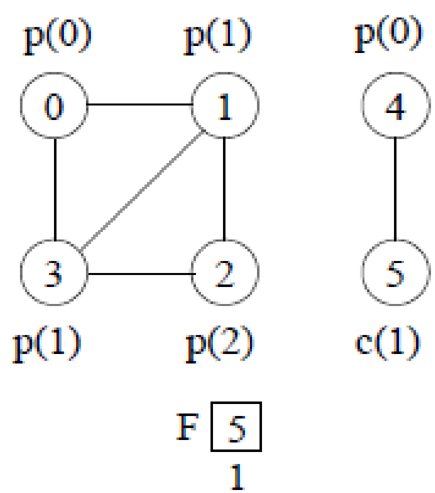
(e)



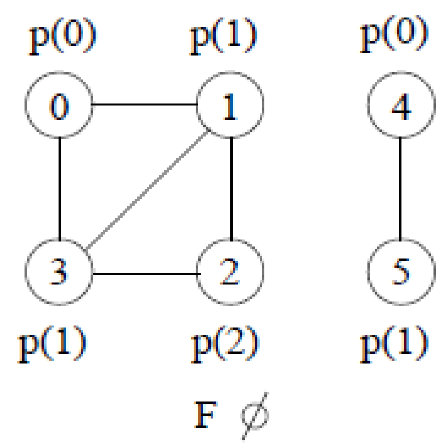
(f)



(g)



(h)



Caminho mais curto

- O procedimento de busca em largura constrói uma árvore de busca que é armazenada na variável antecessor (π).
- O programa abaixo imprime os vértices do caminho mais curto entre o vértice origem e outro vértice qualquer do grafo, a partir do vetor antecessor.

```
public void imprimeCaminho (int origem, int v) {
  if (origem == v) System.out.println (origem);
  else if (this.antecessor[v] == -1)
    System.out.println ("Nao existe caminho de " + origem + " ate " + v);
  else {
    imprimeCaminho (origem, this.antecessor[v]);
    System.out.println (v);
  }
}
```

Caminho mais curto

- No exemplo anterior, o caminho mais curto entre os vértices 0 e 2, dado pela busca em largura realizada, é 0 – 1 – 2.

- O caminho 0 – 3 – 2 não seria informado pela busca em largura realizada porque o predecessor do vértice 2 é o vértice 1, e não o vértice 3.

- Perceba que o vértice 1 entrou na fila antes do vértice 3.

Grafos eulerianos e hamiltonianos

- Caminhos que usam todos os vértices ou todas as arestas de um grafo são geralmente chamados de percursos.

- Inúmeros problemas práticos podem ser vistos como um percurso num grafo: coleta de lixo, caixeiro viajante, etc.

- Eles se dividem em duas categorias:
  - Problemas do tipo euleriano; e
  - Problemas do tipo hamiltoniano.

Grafos eulerianos e hamiltonianos

- Problema do tipo euleriano: requer que <u>todas as arestas</u> do grafo sejam percorridas sem repetição.

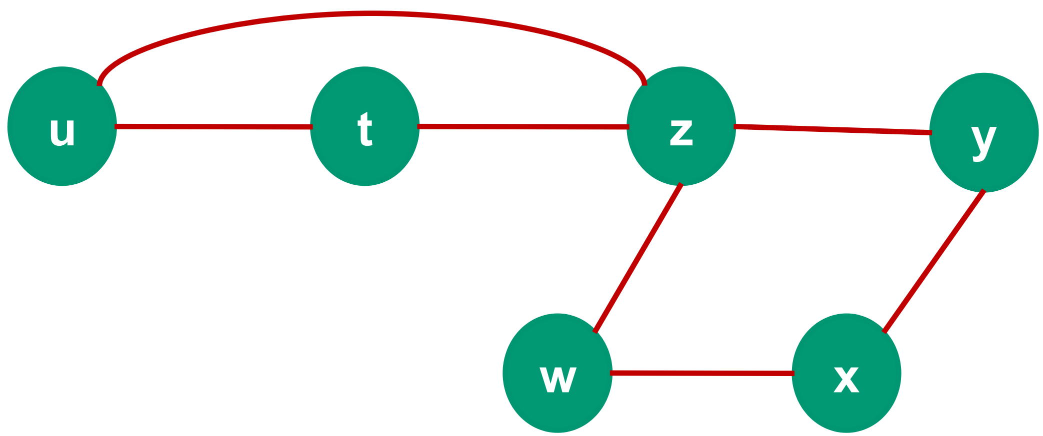
- Os problemas eulerianos podem ser resolvidos em tempo polinomial.

- Problema do tipo hamiltoniano: requer que <u>todos os vértices</u> do grafo sejam visitados uma única vez.

- Os problemas hamiltonianos têm uma solução de custo elevado e são classificados como NP-difícil.

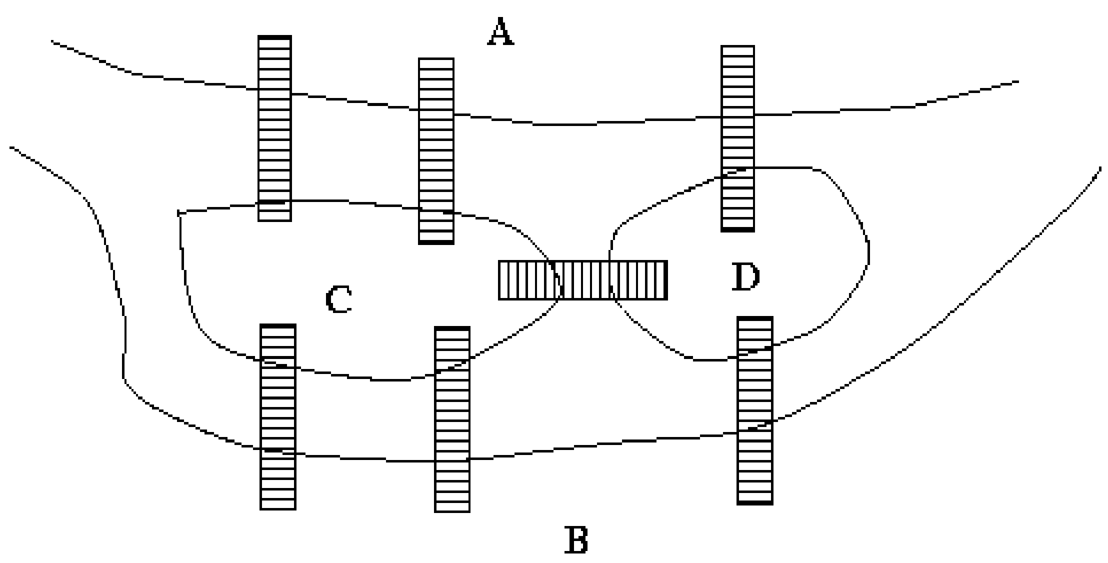
Grafos eulerianos

- Um percurso euleriano passa por todas as arestas do grafo sem repetição.
- Um ciclo euleriano é um percurso euleriano fechado.
- Um grafo é euleriano quando contém um ciclo euleriano.



Grafos eulerianos

- Problema das Sete Pontes de Königsberg (1736): é possível construir um percurso (ou um ciclo) que atravesse cada ponte exatamente uma vez?



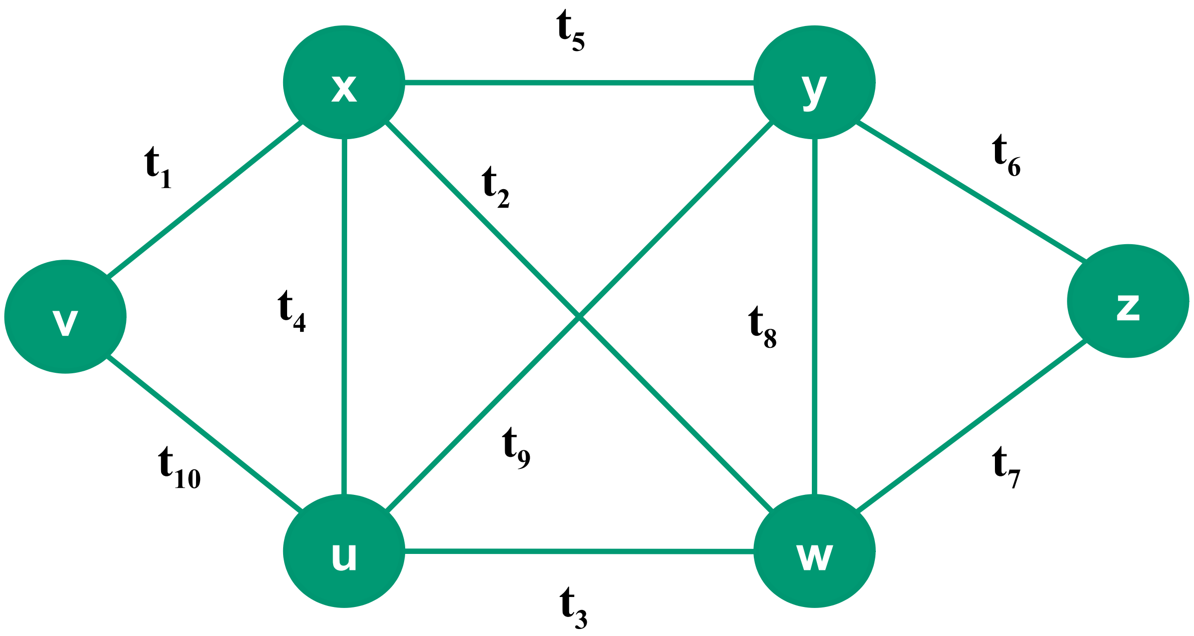
Grafos eulerianos

- Leonhard Euler provou que uma condição necessário para a existência de ciclos eulerianos é que todos os vértices do grafo tenham grau par.

- Euler afirmou ainda, mas, sem prova, que grafos conexos com todos os vértices de grau par possuem um ciclo euleriano.

- Em 1873, Carl Hierholzer conseguiu provar a afirmativa de Euler e estabeleceu que um grafo conexo é euleriano se e somente se <u>todos os seus vértices têm grau par</u>.

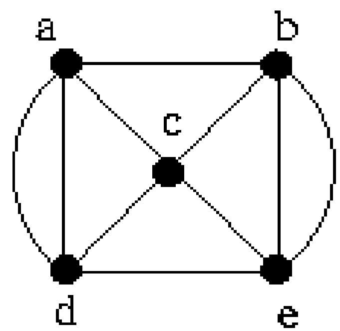
Grafos eulerianos



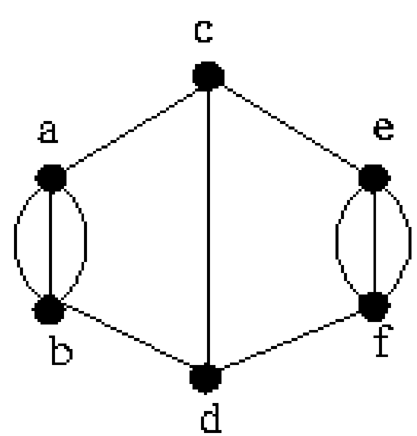
Grafos eulerianos

- Um grafo tem um percurso euleriano aberto se e somente se ele tem exatamente dois vértices de grau ímpar. Também chamado de grafo semi-euleriano.

- O uso do termo euleriano é idêntico para dígrafos, exceto que os percursos são direcionados.



(a)



(b)

Grafos eulerianos

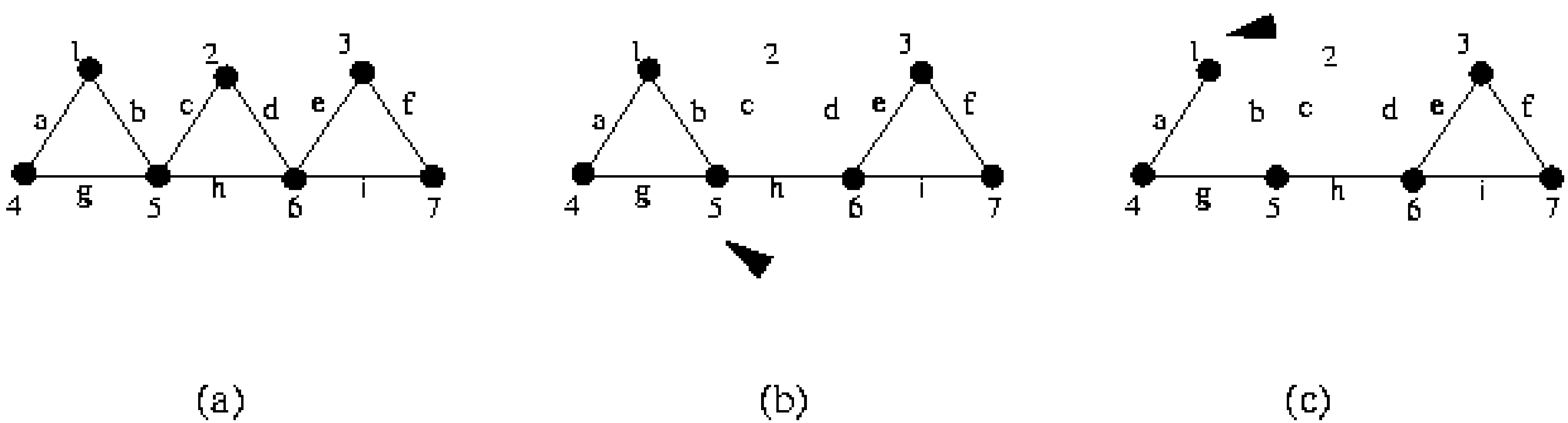
- **Algoritmo de Fleury**: constrói um ciclo euleriano em um grafo euleriano.

- Comece em qualquer vértice **u** e percorra as arestas de forma aleatória, seguindo sempre as seguintes regras:

  I – apague as arestas depois de passar por elas;

  II – se aparecer algum vértice isolado, apague-o também; e

  III – passe por uma ponte somente se não houver outra alternativa.

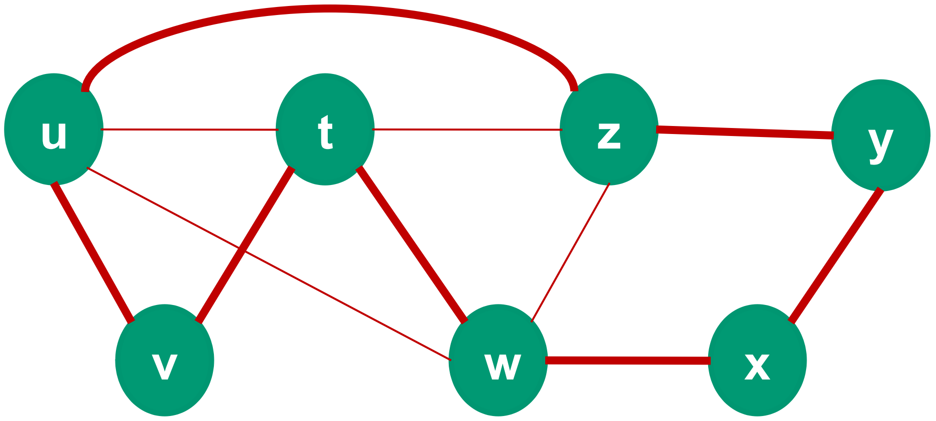


Grafos hamiltonianos

- Um percurso hamiltoniano em um grafo **G** é um <u>caminho simples</u> que contém todos os vértices de **G**.

- Um ciclo hamiltoniano é um percurso hamiltoniano fechado.

- Um grafo é hamiltoniano quando tem um ciclo hamiltoniano.



Grafos hamiltonianos

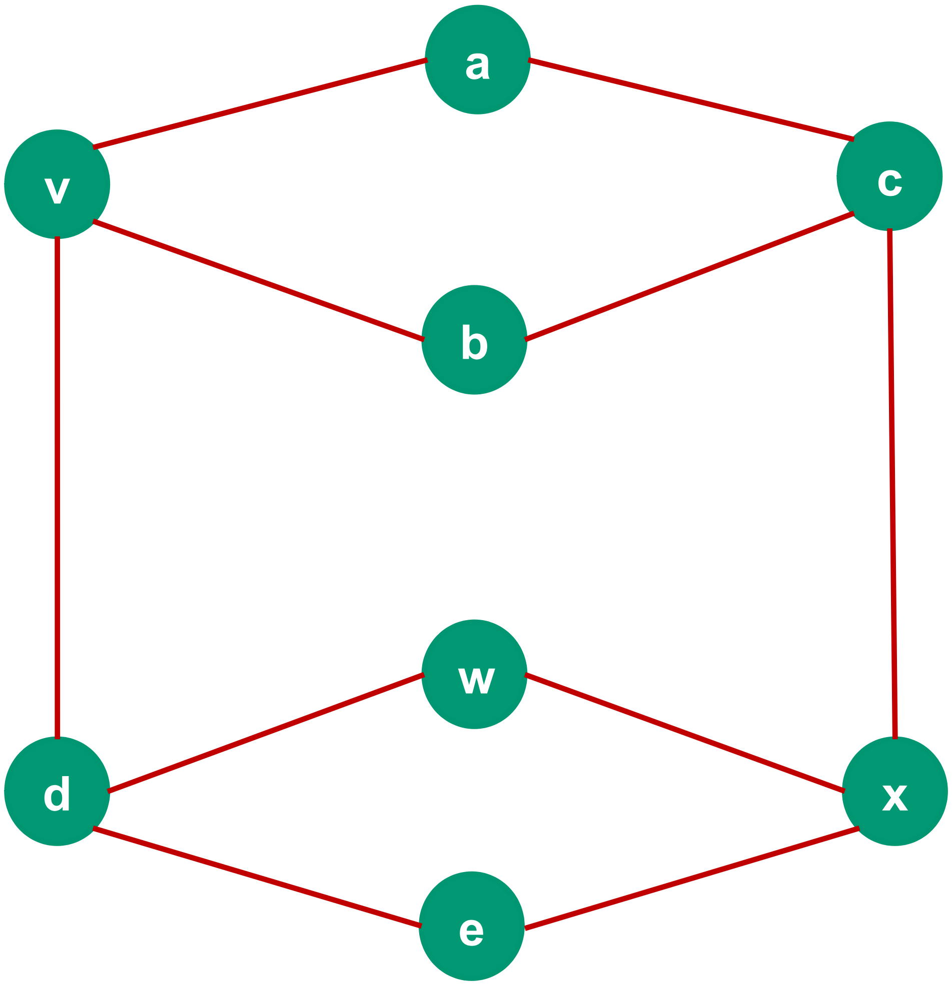
- Não existe um algoritmo que funcione para qualquer grafo.

- Há condições necessárias para um grafo ser hamiltoniano, mas não suficientes.

- Exemplo: Todo grafo hamiltoniano é biconexo, pois existem dois caminhos disjuntos entre cada par de vértices.

- Contudo, a recíproca não é verdadeira!

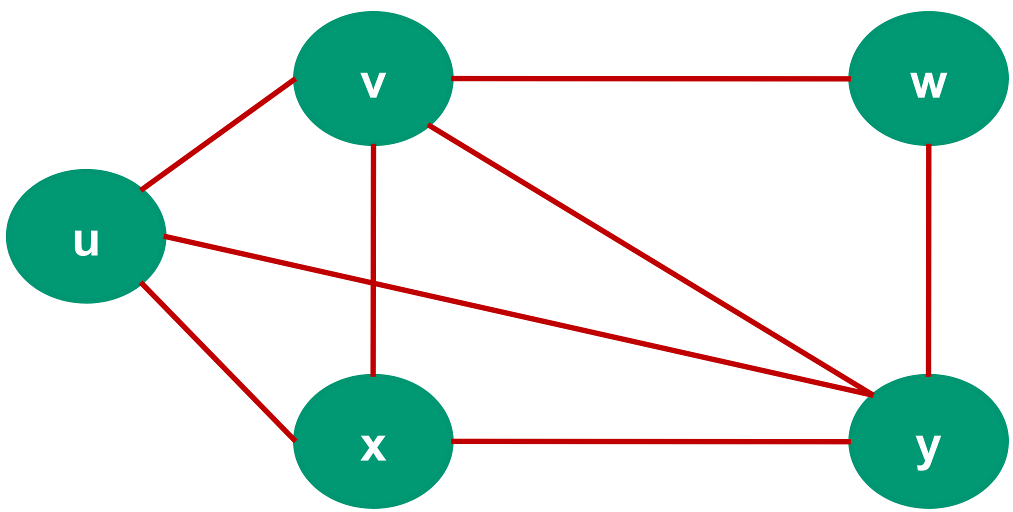
Grafo biconexo não hamiltoniano



Grafos hamiltonianos

- <u>Condição de Ore</u> (1960): suficiente, mas não necessária.

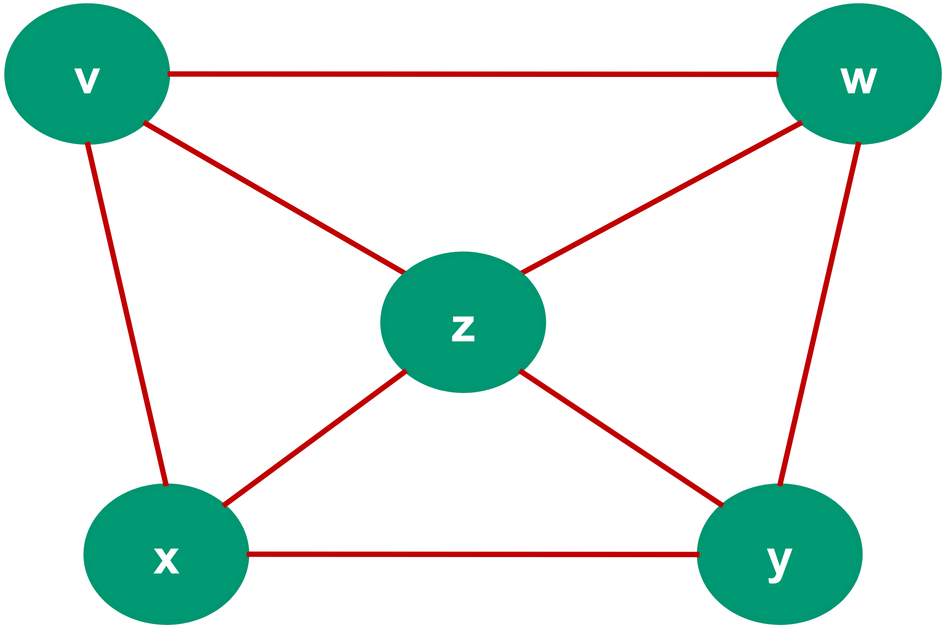
  “Seja **G** um grafo simples de **n** vértices, onde **n ≥ 3**, tal que **deg(x) + deg(y) ≥ n** para cada par de vértices não adjacentes **x** e **y**. Então **G** é hamiltoniano.”



Grafos hamiltonianos

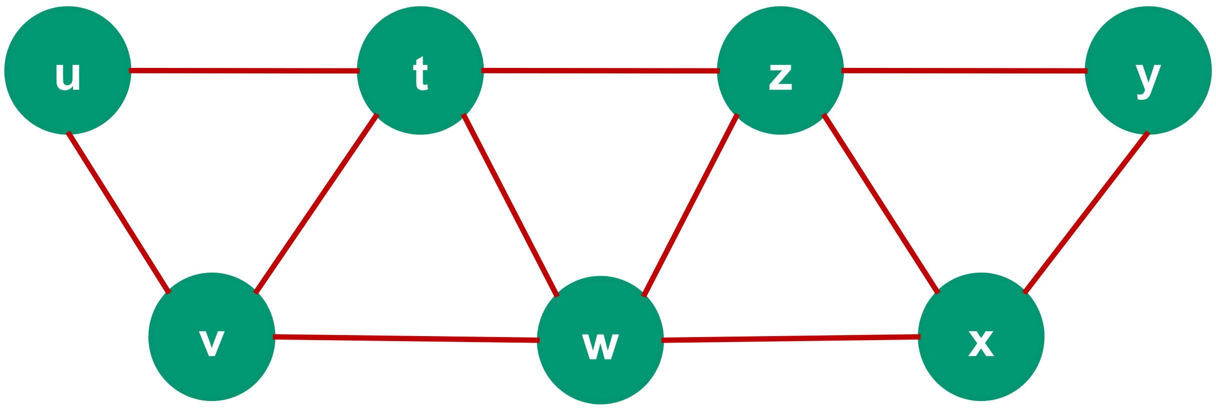
- <u>Condição de Dirac</u> (1952): suficiente, mas não necessária.

  "Seja **G** um grafo simples com **n** vértices, onde **n ≥ 3**, tal que **deg(v) ≥ n/2** para cada vértice **v**. Então **G** é hamiltoniano."



Exercício

- A Condição de Ore é satisfeita para o grafo abaixo? É possível afirmar que o grafo é hamiltoniano? Por quê?



Exercício

- Identifique se o grafo abaixo é ou não hamiltoniano.