BCM0506 – Comunicação e Redes

Aulas 4 e 5

Introdução a Grafos

(Parte II)

Prof. Dr. Aritanan Gruber

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

Quadrimestre Suplementar 2020

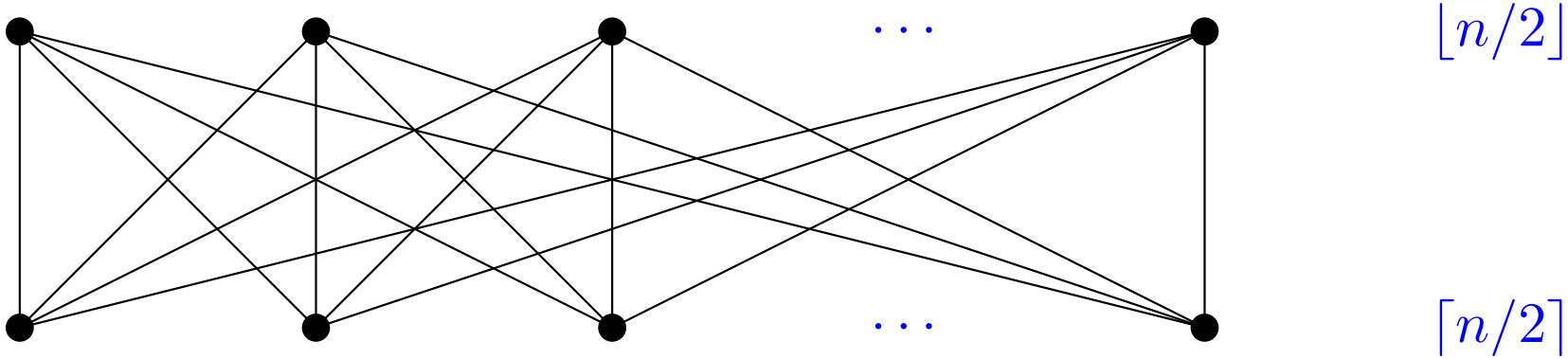
Grafos Livres de Triângulos

Na aula passada, enunciamos o Teorema de Mantel.

Teorema (Mantel, 1910)

Se G é um grafo com n vértices e $m > \lfloor n^2/4 \rfloor$ arestas, então G possui um triângulo como subgrafo.

E mostramos, através do exemplo abaixo, que o limitante inferior no número de arestas é justo.



Grafos Livres de Triângulos

Deixamos, ainda, um guia para a produção de uma prova para o teorema.

- Suponha que G não tem triângulos; analise as vizinhanças dos vértices e note que elas formam conjuntos estáveis;
- Tome S estável máximo e mostre que $m \leq |S| \cdot (V \setminus S)$;
- Use a desigualdade entre as médias aritmética e geométrica

$$\sqrt{xy} \leq \frac{x+y}{2}$$

  para concluir o resultado.

Sem mais delongas, vamos a ela!

Primeira Prova do Teorema de Mantel

Prova. Seja $G = (V, E)$ um grafo com n vértices, m arestas e livre de triângulos.

Seja $S \subseteq V$ um conjunto estável de cardinalidade máxima e defina $T := V \setminus S$.

Como G não tem triângulos, $N(v)$ é um conjunto estável para todo $v \in V$ e assim, $d(v) \leq |S|$.

Já que S é estável, todas as arestas em E possuem ao menos uma ponta em T. Logo,

$$m = \sum_{e \in E} 1 \leq \sum_{v \in T} d(v) \leq \sum_{v \in T} |S| = |S| \cdot |T|.$$

Pela desigualdade entre as médias aritmética e geométrica, $|S| \cdot |T| \leq (|S| + |T|)^2/4 = n^2/4$, concluindo a prova. $\square$

Grafos Livres de Triângulos

Antes de generalizarmos o resultado para cliques maiores, vamos apresentar a prova original desenvolvida por Mantel. Ela é igualmente instrutiva.

Vamos usar o resultado do exercício que provamos na aula passada,

$$\sum_{v \in V} d_G(v)^2 = \sum_{\{u,v\} \in E} (d_G(u) + d_G(v)),$$

e a desigualdade de Cauchy-Schwartz:

$$\left(\sum_{i=1}^{n} x_i y_i \right)^2 \leq \left(\sum_{i=1}^{n} x_i^2 \right) \left(\sum_{i=1}^{n} y_i^2 \right).$$

Digressão: Prova de Cauchy-Schwarz

Lema (desigualdade de Cauchy-Schwarz)

Para quaisquer vetores $\boldsymbol{x}, \boldsymbol{y} \in \mathbb{R}^n$, vale que $|\langle \boldsymbol{x}, \boldsymbol{y} \rangle| \leq \|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|$.

Prova. Podemos supor que $\boldsymbol{x} \neq \boldsymbol{0}$. Defina $\boldsymbol{u} = \lambda \boldsymbol{x} - \boldsymbol{y}$, para algum $\lambda \in \mathbb{R}$. Temos que

$$\langle \boldsymbol{u}, \boldsymbol{u} \rangle = \langle \lambda \boldsymbol{x} - \boldsymbol{y}, \lambda \boldsymbol{x} - \boldsymbol{y} \rangle = \lambda^2 \langle \boldsymbol{x}, \boldsymbol{x} \rangle - 2\lambda \langle \boldsymbol{x}, \boldsymbol{y} \rangle + \langle \boldsymbol{y}, \boldsymbol{y} \rangle \geq 0,$$

em que a desigualdade final advém do fato que $\langle \boldsymbol{u}, \boldsymbol{u} \rangle = \|\boldsymbol{u}\|^2 \geq 0$. Tomando $\lambda = \langle \boldsymbol{x}, \boldsymbol{y} \rangle / \langle \boldsymbol{x}, \boldsymbol{x} \rangle$ e substituindo acima,

$$\frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle^2}{\langle \boldsymbol{x}, \boldsymbol{x} \rangle} - 2\frac{\langle \boldsymbol{x}, \boldsymbol{y} \rangle^2}{\langle \boldsymbol{x}, \boldsymbol{x} \rangle} + \langle \boldsymbol{y}, \boldsymbol{y} \rangle \geq 0 \quad \Longleftrightarrow \quad \langle \boldsymbol{x}, \boldsymbol{y} \rangle^2 \leq \langle \boldsymbol{x}, \boldsymbol{x} \rangle \langle \boldsymbol{y}, \boldsymbol{y} \rangle,$$

ou $|\langle \boldsymbol{x}, \boldsymbol{y} \rangle| \leq \langle \boldsymbol{x}, \boldsymbol{x} \rangle^{1/2} \langle \boldsymbol{y}, \boldsymbol{y} \rangle^{1/2} = \|\boldsymbol{x}\| \cdot \|\boldsymbol{y}\|$. □

A igualdade ocorre se e somente se: um dos vetores é igual a zero, ou quando $\boldsymbol{x} = \boldsymbol{y}$.

Segunda Prova do Teorema de Mantel

Prova. Seja $G = (V, E)$ um grafo com n vértices, m arestas e livre de triângulos.

Para cada aresta $\{u, v\} \in E$, temos que $d(u) + d(v) \leq n$.

De fato, por ser livre de triângulos, $N(u) \cap N(v) = \emptyset$ para todo par de vértices u, v de G, resultando em

$$n = |V| \geq |N(u) \cup N(v)| = |N(u)| + |N(v)| = d(u) + d(v).$$

Somando a desigualdade para todas as arestas de G, obtemos

$$nm \geq \sum_{\{u,v\} \in E} \big(d(u) + d(v)\big) = \sum_{v \in V} d(v)^2, \qquad (\bullet)$$

em que a segunda relação advém do exercício que provamos.

Segunda Prova do Teorema de Mantel

Continuando...

Pela desigualdade de Cauchy-Schwartz

$$\sum_{v \in V} d(v)^2 \geq \frac{1}{n}\left(\sum_{v \in V} d(v)\right)^2 = \frac{4m^2}{n},$$

com a igualdade vindo da aplicação do lema dos apertos de mãos.

Aplicando o lado direito em $(\bullet)$ e rearranjando os termos, obtemos que

$$m \leq \frac{n^2}{4}.$$

□

Grafos Livres de Cliques

Pál Turán generalizou o resultado de Mantel para cliques maiores.

Theorem (Turán, 1941)

Para todo inteiro $r \geq 2$, se G é um grafo com n vértices e

$$m \leq \frac{n^2}{2}\left(1 - \frac{1}{r}\right)$$

arestas, então G é K_{r+1}-livre.

Alguma ideia de como provar este resultado?

Antes da prova, observe que

- $\frac{1}{r} \to 0$ à medida que $r \to \infty$, corroborando com nossa intuição de que é preciso mais arestas para forçar a existência de cliques maiores;
- independente do valor $r \geq 2$ desejado, o grafo tem de ser denso.

Prova do Teorema de Turán

Prova. Sejam $r \geq 2$ e G um grafo com n vértices e com o maior número possível de arestas sem que contenha um $(r+1)$-clique.

A prova é por indução em n.

Os casos em que $n \in \{0, 1, 2, \ldots, r\}$ são triviais, pois não há a possibilidade de $(r+1)$-cliques.

Considere então que $n \geq r+1$ e suponha que o resultado é válido para qualquer grafo com menos de n vértices.

Por maximalidade, G tem ao menos um r-clique $R \subset V(G)$ – em caso contrário, arestas poderiam ser adicionadas a G sem formar um $(r+1)$-clique.

Defina $T := V(G) \setminus R$ e $t := |T|$. Cada vértice $v \in T$ tem, no máximo, $r-1$ vizinhos em R.

Prova do Teorema de Turán

Continuando...

Logo,

$$m \leq \binom{r}{2} + (r-1)t + \frac{t^2}{2}\left(1 - \frac{1}{r}\right), \qquad (1)$$

em que o terceiro termo do lado direito vem por hipótese de indução em $G[T]$. Escrevendo

$$\binom{r}{2} = \frac{r^2}{2}\left(1 - \frac{1}{r}\right) \qquad \text{e} \qquad (r-1)t = \frac{2rt}{2}\left(1 - \frac{1}{r}\right)$$

e substituindo em (1), obtemos que

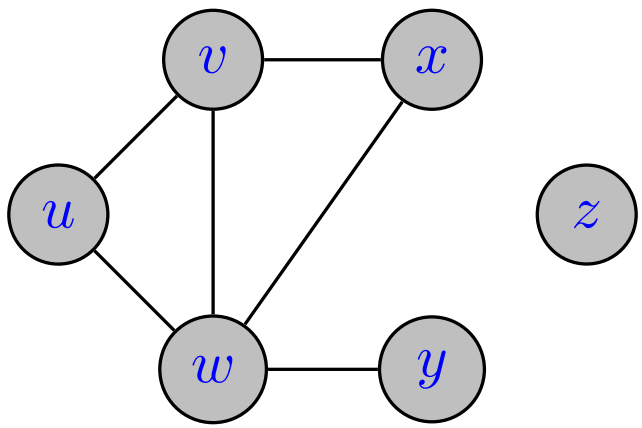
$$m \leq \frac{(r^2 + 2rt + t^2)}{2}\left(1 - \frac{1}{r}\right) = \frac{(r+t)^2}{2}\left(1 - \frac{1}{r}\right) = \frac{n^2}{2}\left(1 - \frac{1}{r}\right). \quad \square$$

Passeios, Trilhas e Caminhos

Um (u, v)-*passeio* P em um grafo G é uma sequência de vértices com início em u e término em v,

$$P = \langle u = v_0, v_1, \ldots, v_k = v \rangle,$$

e tal que vértices adjacentes na sequência são conectados por arestas: $\{v_{j-1}, v_j\} \in E(G)$.



Um passeio no grafo ao lado:

$$P_1 = \langle x, w, y, w, x, v, w \rangle$$

Outro passeio:

$$P_2 = \langle u, v, w, u, w, v, u, w, y \rangle$$

Passeios, Trilhas e Caminhos

O *comprimento* de um passeio P é dado por $|P| = k$, o número de arestas na sequência que define P.

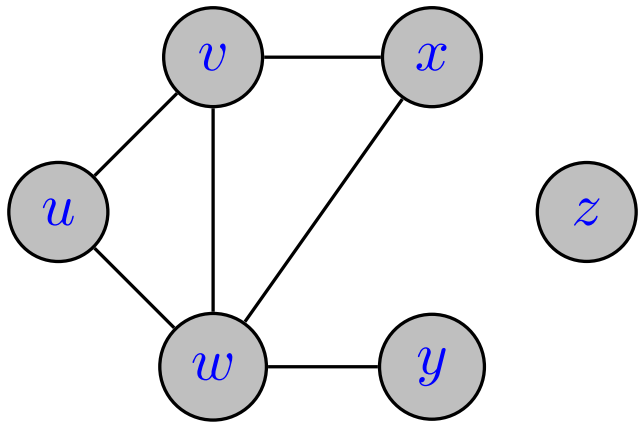
Os vértices e arestas de P formam um subgrafo de G. Logo, abusamos da notação e escrevemos $P \subseteq G$.

Um passeio P é

- uma *trilha* se suas arestas forem distintas;
- um *caminho* se seus vértices forem distintos.

De outra forma, uma trilha é um passeio que não repete arestas e um caminho é uma trilha que não repete vértices (e, claramente, também não repete arestas).

Passeios, Trilhas e Caminhos



Uma trilha no grafo ao lado:

$$T = \langle y, w, x, v, w, u \rangle$$

Um caminho:

$$C = \langle u, v, x, w, y \rangle$$

Exercício

Prove que:

(a) *se existe um (u, v)-passeio em um grafo G, então existe uma (u, v)-trilha em G;*

(b) *se existe uma (u, v)-trilha em G, então existe um (u, v)-caminho em G.*

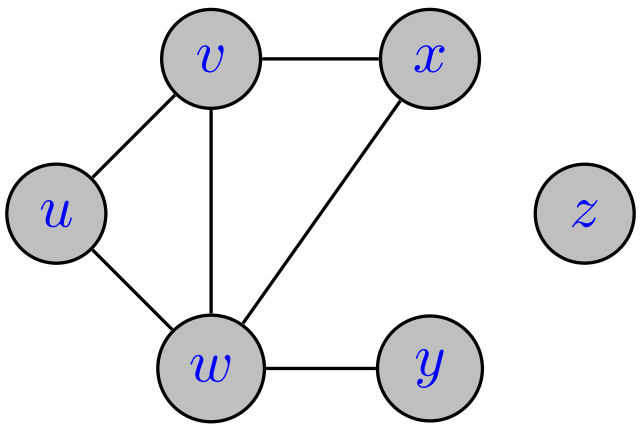
Territórios e Componentes

Escrevemos $u \rightsquigarrow v$ se existe um (u, v)-caminho em G e dizemos que v é *alcançável* à partir de u.

O *território* de um vértice u é definido como

$$\tau(u) := \{v \in V : u \rightsquigarrow v\},$$

o conjunto de vértices alcançáveis à partir de u.



$$\tau(u) = \{u, v, w, x, y\}$$

$$\tau(z) = \{z\}$$

Territórios e Componentes

Note que alcançável ($\rightsquigarrow$) é uma relação de equivalência:

(i) $u \rightsquigarrow u$ para todo $u \in V$,

(ii) se $u \rightsquigarrow v$, então $v \rightsquigarrow u$ para todos $u, v \in V$,

(iii) se $u \rightsquigarrow v$ e $v \rightsquigarrow w$, então $u \rightsquigarrow w$ para todos $u, v, w \in V$.

Assim, o quociente $G/\rightsquigarrow$ é uma classe de equivalências

$$\mathscr{C} = \{[v_1], [v_2], \ldots, [v_\ell]\}$$

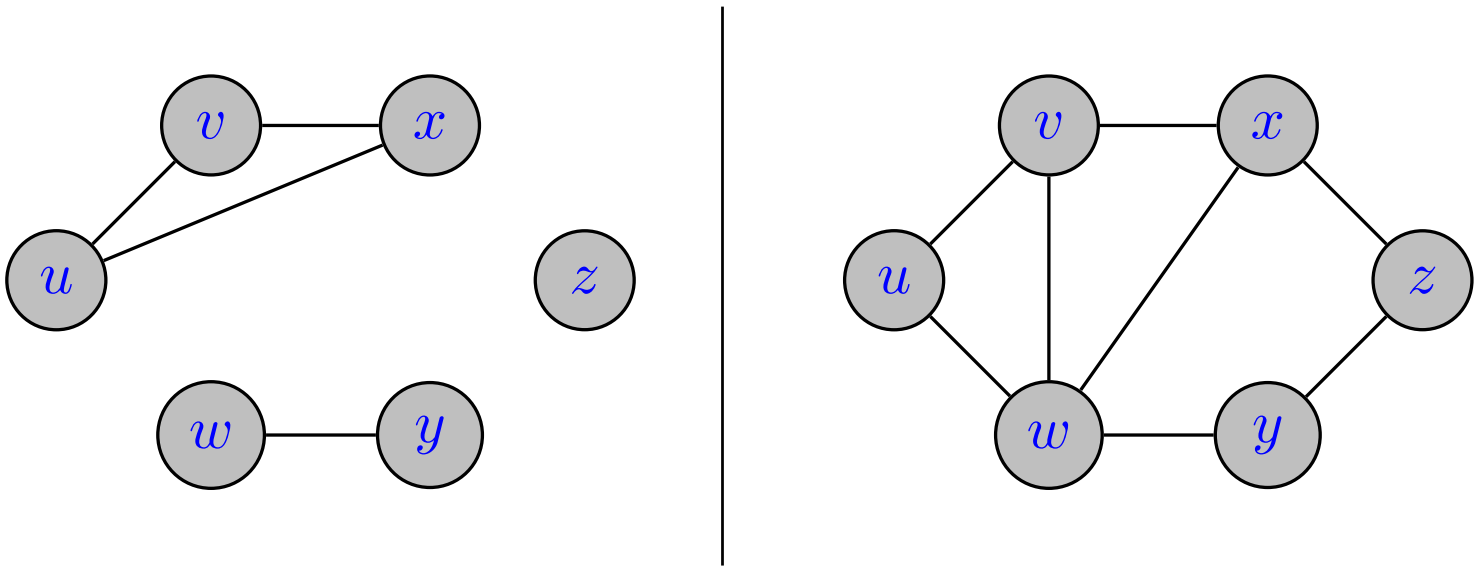
tal que $v_i \in V$, $\tau(v_i) \cap \tau(v_j) = \emptyset$ para $i \neq j$ e $\ell \leq |V|$.

Em palavras, particionamos V em territórios: a união dos territórios é igual a V e os territórios são disjuntos.

(v_i é simplesmente um representante do território i.)

Territórios e Componentes

O subgrafo de G induzido por $\tau(v_i)$, $G[\tau(v_i)]$, é um *componente (conexo)* de G. G é *conexo* se possui um único componente.



O grafo à esquerda é desconexo e possui três componentes; à direita, um grafo conexo.

Territórios e Componentes

Problema

Dado um grafo $G = (V, E)$, determine os componentes de G.

Um algoritmo:

- Inicialmente, $\ell = 0$ e todos os vértices em V são marcados com zero.
- Enquanto algum vértice $x \in V$ tiver marcação igual a zero:
  - incremente o valor de ℓ em uma unidade;
  - marque todos os vértices em $\tau(x)$ com o valor ℓ.
- Quando todos os vértices em V estiverem marcados, devolva ℓ, o número de componentes de G.

Claramente, tal algoritmo identifica a qual componente cada vértice pertence.

Busca em Profundidade (DFS)

A varredura do algoritmo acima pode ser eficientemente obtida via *busca em profundidade* (DFS = *depth-first search*).

Considere uma *pilha* S – estrutura de dados com política de acesso *LIFO* (*last-in, first-out*) a seus elementos – inicialmente vazia.

Para um vértice $x \in V$, defina $\tau(x) = \{x\}$ e empilhe x em S.

Enquanto $S \neq \emptyset$, remova o vértice no topo de S, digamos u, adicione u a $\tau(x)$ e empilhe em S, em qualquer ordem, os vértices em $N(u) \setminus \tau(x)$.

Quando $S = \emptyset$, o algoritmo termina e $\tau(x)$ é o território de x.

A maneira mais simples de codificar uma busca em profundidade é através de funções recursivas: a pilha S passa a fazer parte da pilha de recursão.

Busca em Profundidade (DFS)

Pseudocódigo

```
1 DFS (G)
2   ℓ ← 0
3   foreach u ∈ V(G) do k[u], p[u] ← 0, u
4   foreach u ∈ V(G) do
5     if k[u] = 0 then
6       ℓ ← ℓ + 1
7       DFS-Visit (G, u, ℓ, k, p)
8   return ℓ, k, p
```

```
1 DFS-Visit (G, u, ℓ, k, p)
2   k[u] ← ℓ
3   foreach v ∈ N(u) do
4     if k[v] = 0 then
5       p[v] ← u
6       DSF-Visit (G, v, ℓ, k, p)
```

Busca em Profundidade (DFS)

O pseudocódigo devolve:

- $\ell \in \mathbb{N}$, o número de componentes de G;
- $k \in \mathbb{N}^V$, um vetor que, para cada vértice u, fornece o componente a que u pertence;
- $p \in V^V$, um vetor que indica o predecessor de cada vértice durante a exploração; vértices que serviram de origem durante chamadas à função `DFS-Visit` possuem a sí mesmos como predecessores.

Uma forma conveniente de interpretar o vetor p é como uma floresta geradora maximal para G.

O tempo requerido por DFS é $\Theta(|V|^2)$ com matrizes de adjacências e $\Theta(|V| + |E|)$ com listas de adjacências, pois cada vértice de G é empilhado uma única vez e cada aresta inspecionada duas vezes.

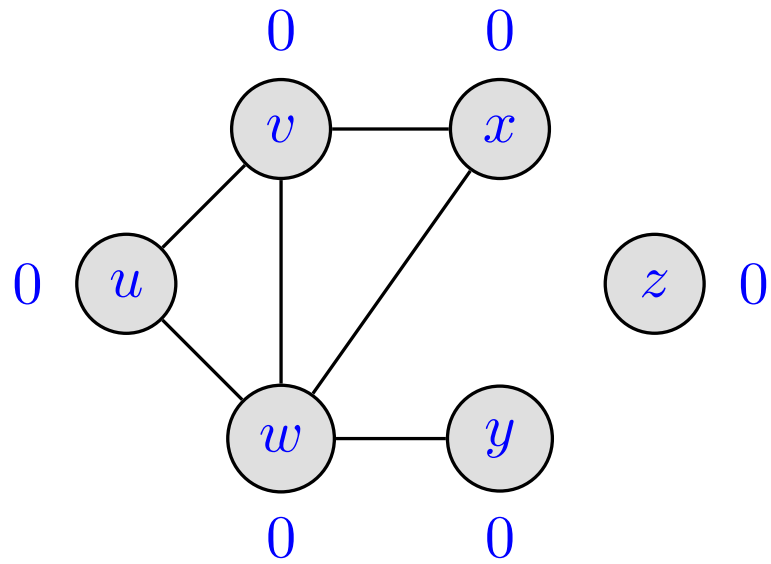
Busca em Profundidade (DFS)

Código em Python para matrizes de adjacências

```
1  def DFS (A: [[int]]) -> (int, [int], [int]):
2    l, n = 0, len (A)
3    k = [0 for u in range (n)]
4    p = [u for u in range (n)]
5
6    def DFS_Visit (u: int, l: int):
7      k[u] = l
8      for v in range (n):
9        if A[l][v] == 1 and k[v] == 0:
10         p[v] = u
11         DFS_Visit (v, l)
12
13   for u in range (n):
14     if A[l][u] == 1 and k[u] == 0:
15       k += 1
16       DFS_Visit (u, l)
17   return (l, k, p)
```

Busca em Profundidade (DFS)

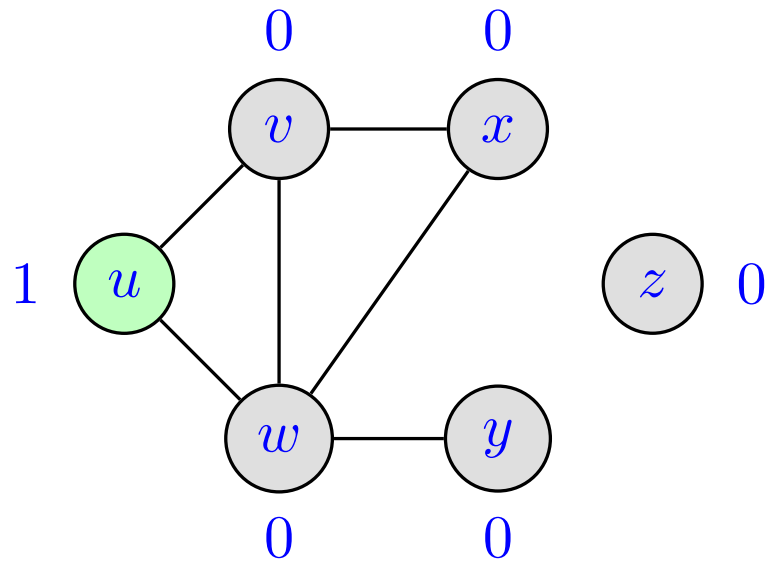
Simulação



$$
\begin{aligned}
S &= \langle\rangle \\
k &= (0,0,0,0,0,0) \\
p &= (u,v,w,x,y,z)
\end{aligned}
$$

Busca em Profundidade (DFS)

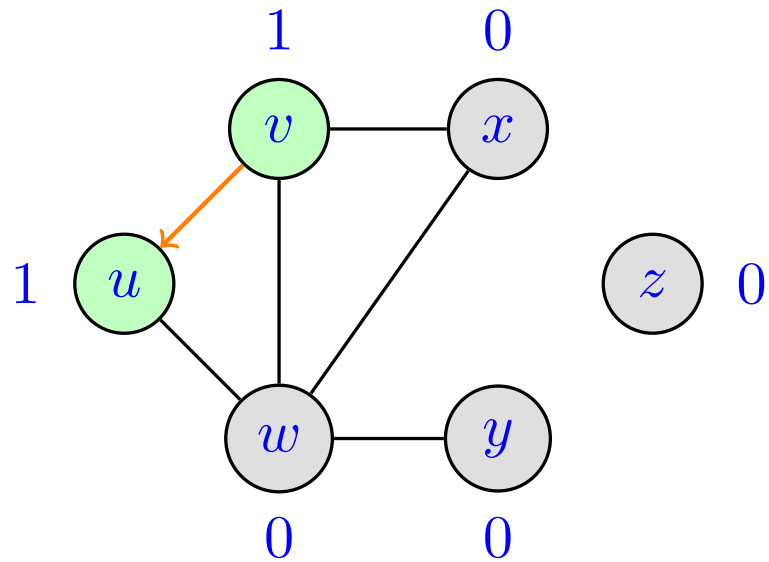
Simulação



$$
\begin{aligned}
S &= \langle u \rangle \\
k &= (\underline{1}, 0, 0, 0, 0, 0) \\
p &= (\underline{u}, v, w, x, y, z)
\end{aligned}
$$

Busca em Profundidade (DFS)

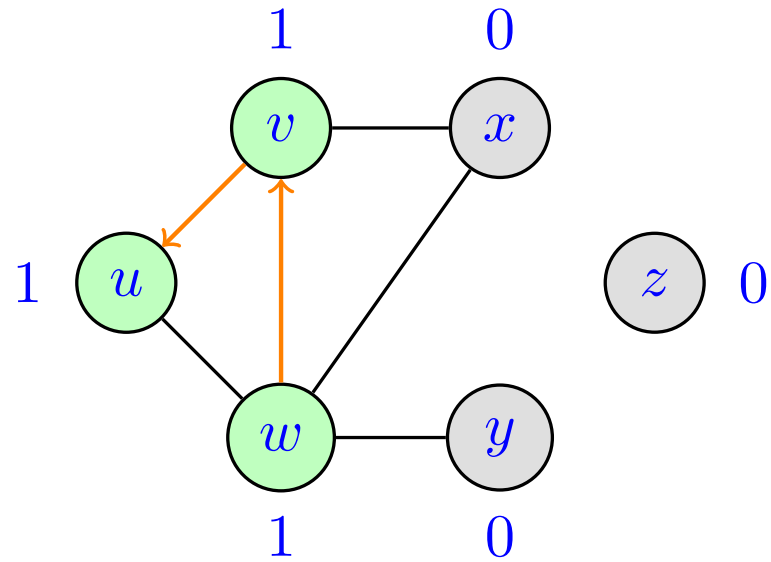
Simulação



$$
\begin{aligned}
S &= \langle u, v \rangle \\
k &= (\underline{1, 1}, 0, 0, 0, 0) \\
p &= (\underline{u, u}, w, x, y, z)
\end{aligned}
$$

Busca em Profundidade (DFS)

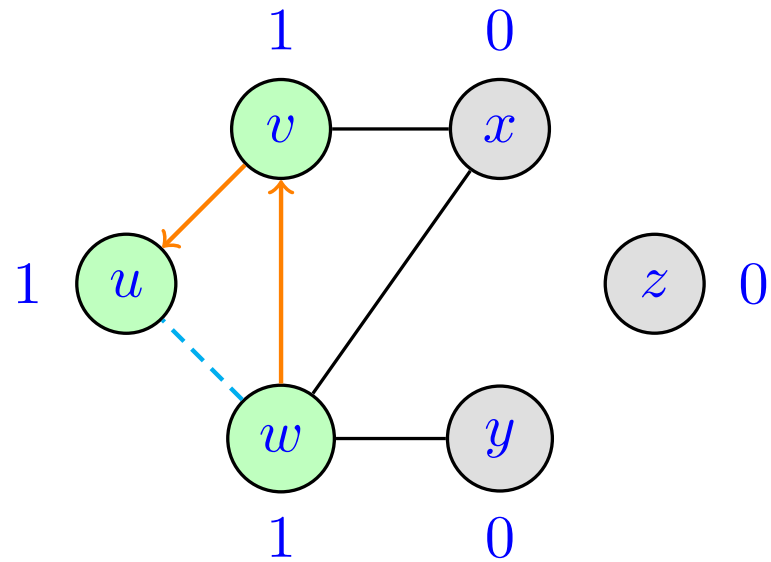
Simulação



$$
\begin{aligned}
S &= \langle u, v, w \rangle \\
k &= (\underline{1, 1, 1}, 0, 0, 0) \\
p &= (\underline{u, u, v}, x, y, z)
\end{aligned}
$$

Busca em Profundidade (DFS)

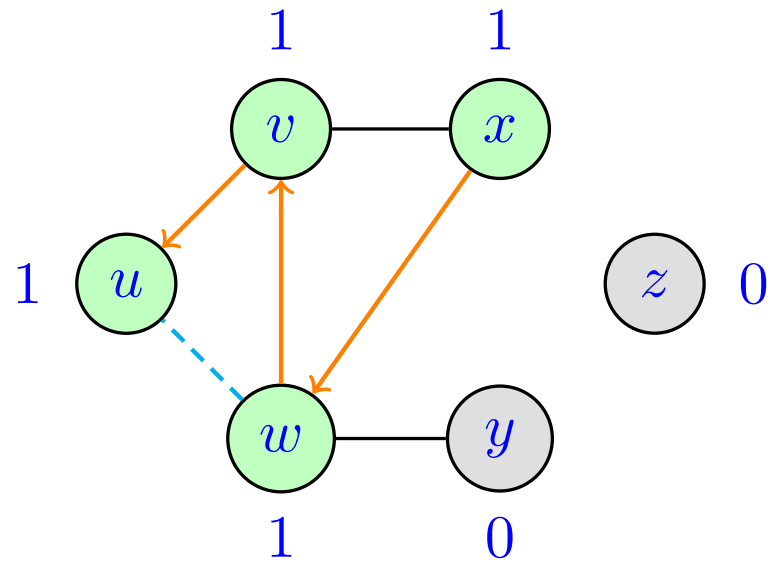
Simulação



$$
\begin{aligned}
S &= \langle u, v, w \rangle \\
k &= (\underline{1, 1, 1}, 0, 0, 0) \\
p &= (\underline{u, u, v}, x, y, z)
\end{aligned}
$$

Busca em Profundidade (DFS)

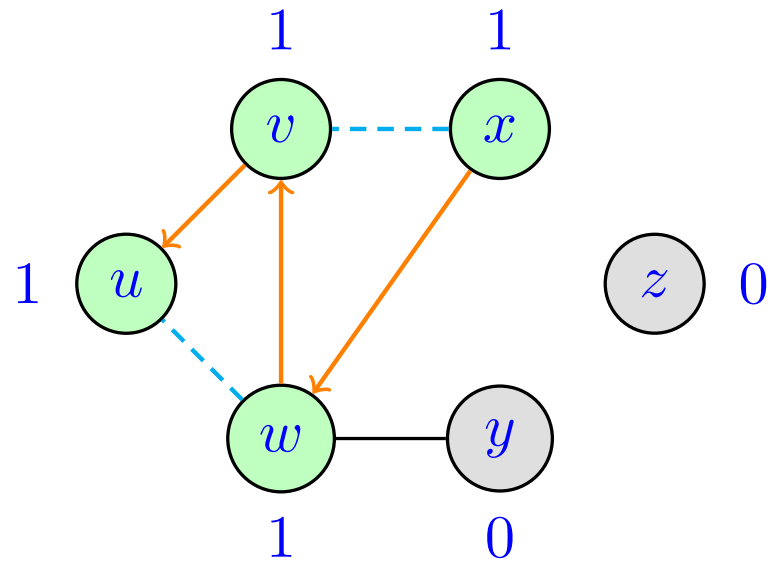
Simulação



$$
\begin{aligned}
S &= \langle u, v, w, x \rangle \\
k &= (\underline{1, 1, 1, 1}, 0, 0) \\
p &= (\underline{u, u, v, w}, y, z)
\end{aligned}
$$

Busca em Profundidade (DFS)

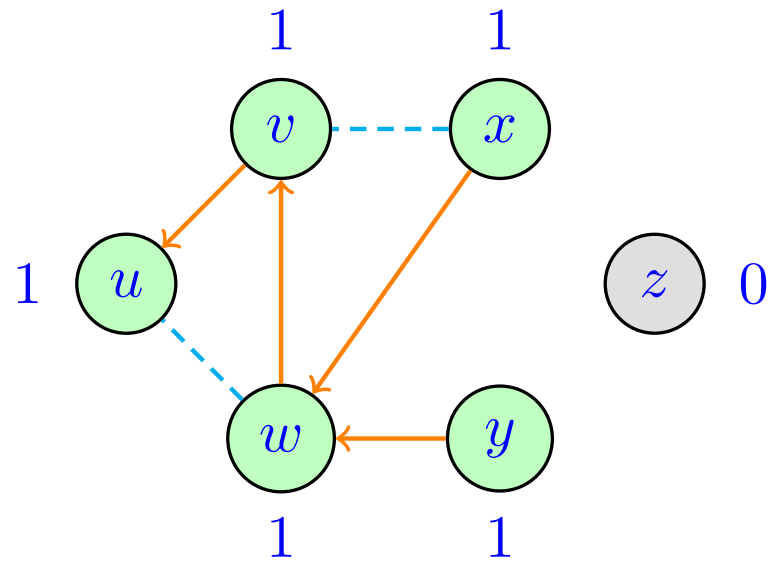
Simulação



$$
\begin{aligned}
S &= \langle u, v, w, x \rangle \\
k &= (\underline{1, 1, 1, 1}, 0, 0) \\
p &= (\underline{u, u, v, w}, y, z)
\end{aligned}
$$

Busca em Profundidade (DFS)

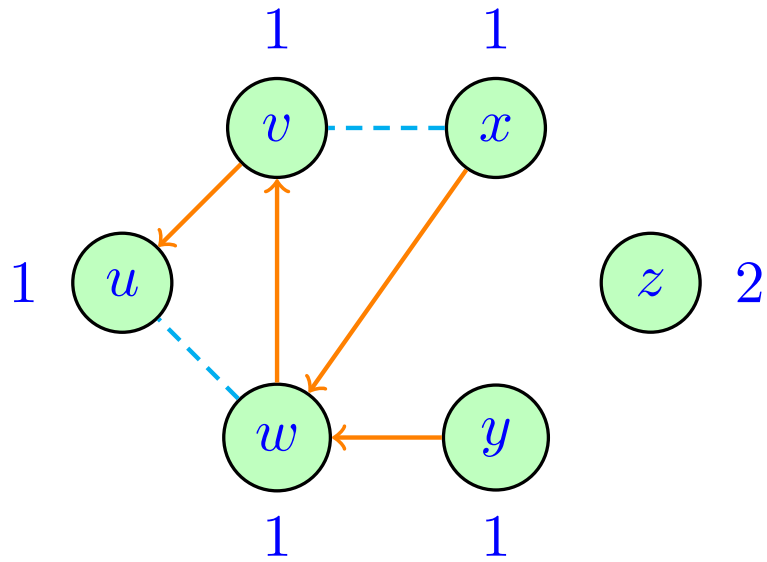
Simulação



$$
\begin{aligned}
S &= \langle u, v, w, y \rangle \\
k &= (\underline{1, 1, 1, 1, 1}, 0) \\
p &= (\underline{u, u, v, w, w}, z)
\end{aligned}
$$

Busca em Profundidade (DFS)

Simulação



$$
\begin{aligned}
S &= \langle z \rangle \\
k &= (\underline{1, 1, 1, 1, 1, 2}) \\
p &= (\underline{u, u, v, w, w, z})
\end{aligned}
$$

Distâncias

Para vértices u, v de um grafo G, a *distância* entre u e v é uma função $\mathrm{dist} : V \times V \to \mathbb{Z}_{\geq}$ definida como

$$\mathrm{dist}(u,v) = \begin{cases} 0 & \text{se } u = v, \\ \min\{|P| : P \in \mathscr{P}(u,v)\} & \text{se } u \neq v, \end{cases}$$

com $\mathscr{P}(u,v)$ o conjunto de todos os (u,v)-caminhos em G.

Em outras palavras, $\mathrm{dist}(u,v)$ é o comprimento de um menor caminho, ou *caminho de comprimento mínimo*, entre u e v.

Caso $u \not\rightsquigarrow v$, temos $\mathrm{dist}(u,v) = \infty$, pois $\min \emptyset = \infty$.

Desta forma, dist é uma *métrica* em V, pois para todos $u, v, w \in V$:
(i) $\mathrm{dist}(u,v) \geq 0$, com a igualdade ocorrendo se e somente se $u = v$;
(ii) $\mathrm{dist}(u,v) = \mathrm{dist}(v,u)$; e (iii) $\mathrm{dist}(u,w) + \mathrm{dist}(w,v) \geq \mathrm{dist}(u,v)$.

Distâncias

Problema (Caminhos de comprimento mínimo)

Dados um grafo $G = (V, E)$ e vértices origem s e destino t, determine dist(s, t) e devolva um (s, t)-caminho mínimo se existir.

Uma solução para o problema acima decorre da própria definição de distância: enumere todos os (s, t)-caminhos em G e escolha o de menor comprimento.

O número de (s, t)-caminhos em G pode ser exponencial em $|V|$.

Lema (Sub-estrutura ótima)

Se $P = \langle s, \ldots, u, \ldots, t \rangle$ é um (s, t)-camínho mínimo, então $\langle s, \ldots, u \rangle$ e $\langle u, \ldots, t \rangle$ são (s, u)- e (u, t)-caminhos mínimos.

Distâncias

Prova. Sejam G, s, t como enunciados e suponha que $\text{dist}(s,t) = k < \infty$. Tome $P(s,t) = \langle s = u_0, u_1, \ldots, u_k = t \rangle$ um (s,t)-caminho mínimo.

Suponha que para algum par $0 \leq i < j \leq k$, o (u_i, u_j)-sub-caminho

$$P(u_i, u_j) = \langle u_i, u_{i+1}, \ldots, u_j \rangle \subset P(s,t)$$

não é mínimo.

Logo, existe um outro (u_i, u_j)-caminho $Q(u_i, u_j)$ com $|Q(u_i, u_j)| < |P(u_i, u_j)|$. Isto implica a existência de um (s,t)-caminho

$$R(s,t) := P(s, u_i) \cup Q(u_i, u_j) \cup P(u_j, t)$$

tal que $|R(s,t)| < |P(s,t)|$, contrariando a hipótese de que $P(s,t)$ é ótimo. □

Distâncias

Usando o lema, podemos exprimir a distância entre s e t como

$$\mathsf{dist}(s,t) = \begin{cases} 0 & \text{se } s = t, \\ 1 + \min_{u \in N(t)} \mathsf{dist}(s,u) & \text{se } s \neq t, \end{cases}$$

e temos, assim, um algoritmo polinomial para o problema.

Avaliando a recorrência para $P(s,t) = \langle s = u_0, u_1, \ldots, u_k = t \rangle$, com $s \neq t$, obtemos que

$$\begin{aligned} \mathsf{dist}(s,t) = 1 + \min_{u_{k-1} \in N(t)} \Big(1 + \min_{u_{k-2} \in N(u_{k-1})} \Big(1 + \cdots \\ + \min_{u_2 \in N(u_3)} \Big(1 + \min_{u_1 \in N(u_2)} \Big(1 + \min_{s \in N(u_1)} 1\Big)\Big) \cdots \Big)\Big) = k, \end{aligned}$$

ou que cada vértice em $P(s,t)$ foi escolhido como a melhor alternativa em um problema local de minimização, e que a sequência de escolhas locais conduz ao ótimo global.

Distâncias

Mais ainda, a sequência de minimizações pode ser utilizada para determinar não apenas a distância entre s e t, mas entre s e todos os demais vértices do grafo, o que é necessário no pior caso.

Temos, assim, um algoritmo *guloso*: varremos os vértices do grafo em ordem crescente de distância à partir do vértice de origem s.

A ordem de varredura dos vértices, que privilegia vizinhança em vez de alcançabilidade, fornece-lhe o nome de *busca em largura* (BFS = *breadth-first search*).

A estrutura de dados adequada para impor a ordem de busca desejada é uma *fila*, que possui política de acesso *FIFO* (first-in, first-out).

Busca em Largura (BFS)

```
1  BFS (G, s)
2  foreach u ∈ V(G) do
3     d[u], p[u] ← ∞, u
4  d[s] ← 0
5  Q ← queue ({s})
6  while Q ≠ ∅ do
7     u ← remove-front (Q)
8     foreach v ∈ N(u) do
9        if d[v] > d[u] + 1 then
10          d[v], p[v] ← d[u] + 1, u
11          append (Q, v)
12 return d, p
```

Busca em Largura (BFS)

O algoritmo BFS recebe um grafo $G = (V, E)$ e um vértice origem $s \in V$.

Devolve:

- um vetor de distâncias $d \in \mathbb{N}^V$, em que $d[u] = \mathsf{dist}(s, u)$;
- um vetor $p \in V^V$ que indica o predecessor de cada vértice durante a exploração.

Vértices que não foram alcançados na busca, em conjunto com s, possuem a sí mesmos como predecessores. O vetor de predecessores também é uma floresta geradora para G, mas não é maximal neste caso.

O tempo requerido por BFS é $O(|V|^2)$ com matrizes de adjacências e $O(|V| + |E|)$ com listas de adjacências, pois cada vértice entra na fila no máximo uma vez e cada aresta é inspecionada uma vez para cada ponta que passe pela fila; logo, 0 ou 2 vezes.

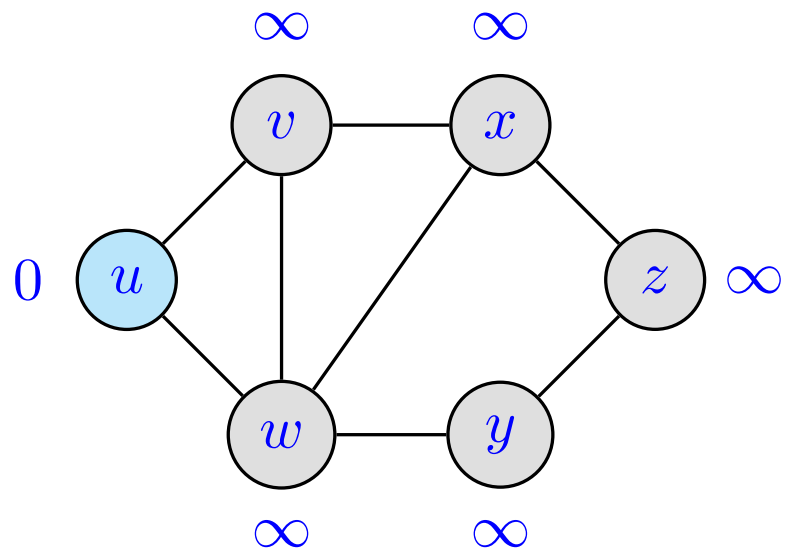
Busca em Largura (BFS)

Código em Python para matrizes de adjacências

```
1  from math import inf as oo
2
3  def BFS (A: [[int]], s: int) -> ([int], [int]):
4    d = [oo] * (n := len (A)); d[s] = 0
5    p = [u for u in range (n)]
6    Q = [s]
7    while Q != []:
8      u = Q.pop (0)
9      for v in range (n):
10       if A[u][v] == 1 and d[u] + 1 < d[v]:
11         d[v] = d[u] + 1; p[v] = u
12         Q.append (v)
13   return (d, p)
```

Busca em Largura (BFS)

Simulação

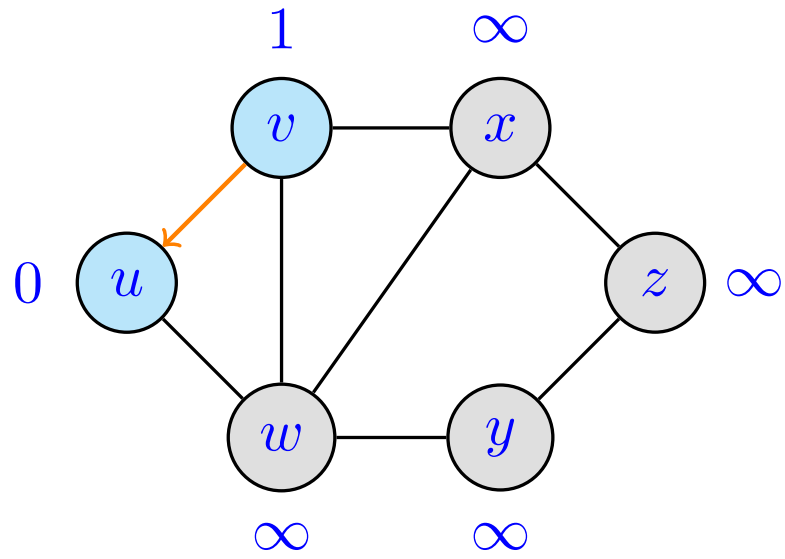


$\mapsto$

$$
\begin{aligned}
Q &= \langle u \rangle \\
d &= (\underline{0}, \infty, \infty, \infty, \infty, \infty) \\
p &= (\underline{u}, v, w, x, y, z)
\end{aligned}
$$

Busca em Largura (BFS)

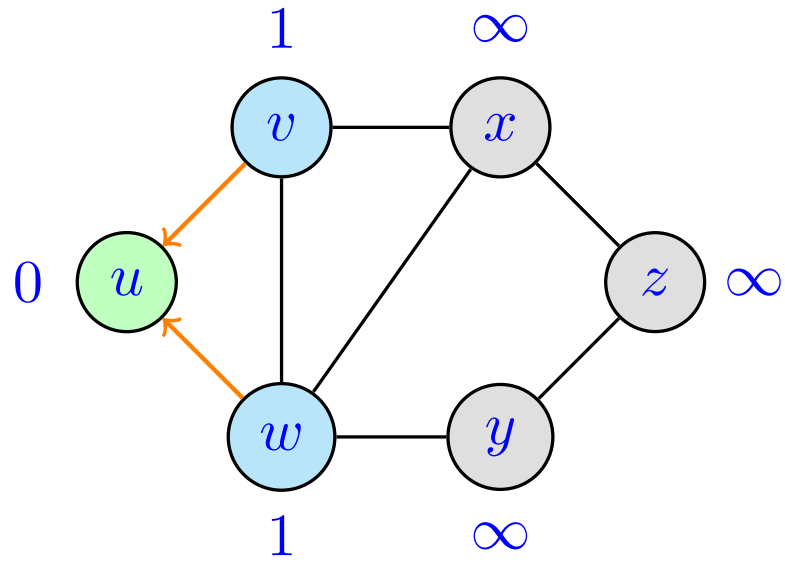
Simulação



$$
\begin{array}{lcl}
\mapsto & u & \\
Q & = & \langle v \rangle \\
d & = & (\underline{0,1}, \infty, \infty, \infty, \infty) \\
p & = & (\underline{u,u}, w, x, y, z)
\end{array}
$$

Busca em Largura (BFS)

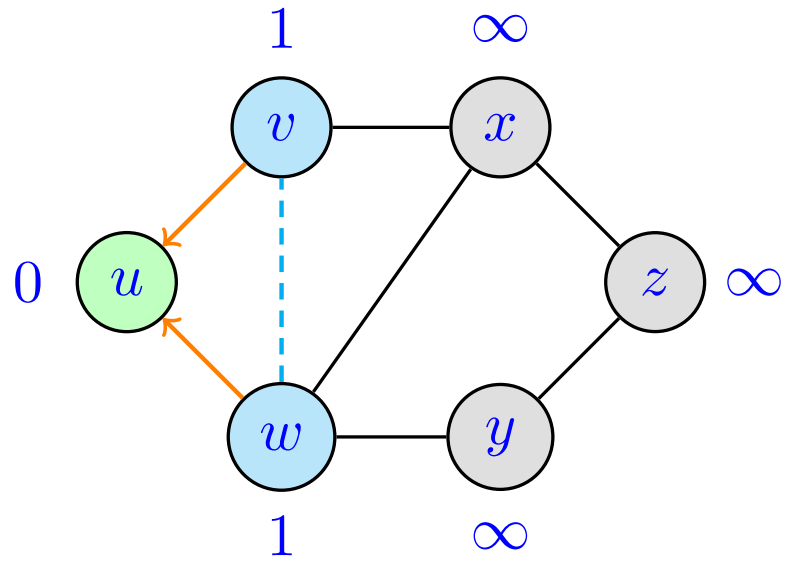
Simulação



$$
\begin{array}{lcl}
\mapsto & u & \\
Q & = & \langle v, w \rangle \\
d & = & (\underline{0, 1, 1}, \infty, \infty, \infty) \\
p & = & (\underline{u, u, u}, x, y, z)
\end{array}
$$

Busca em Largura (BFS)

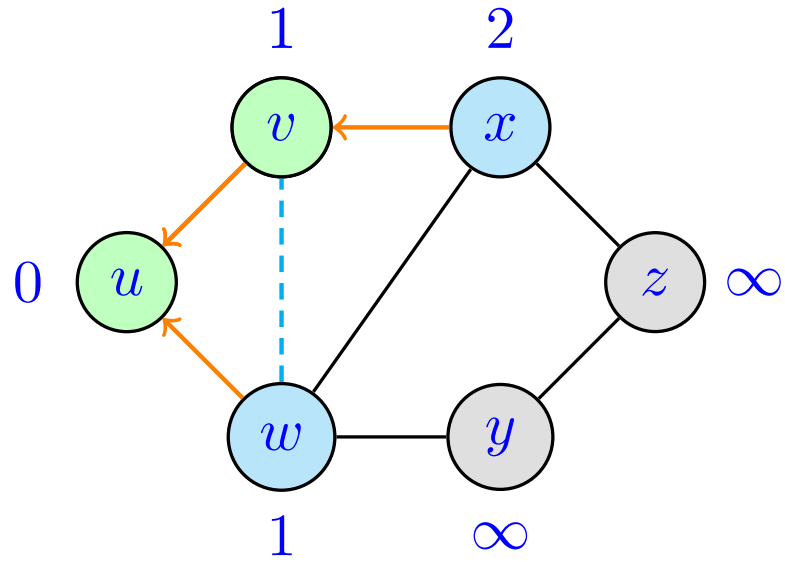
Simulação



$$
\begin{array}{lcl}
\mapsto & v & \\
Q & = & \langle w \rangle \\
d & = & (\underline{1,1,1},\infty,\infty,\infty) \\
p & = & (\underline{u,u,u},x,y,z)
\end{array}
$$

Busca em Largura (BFS)

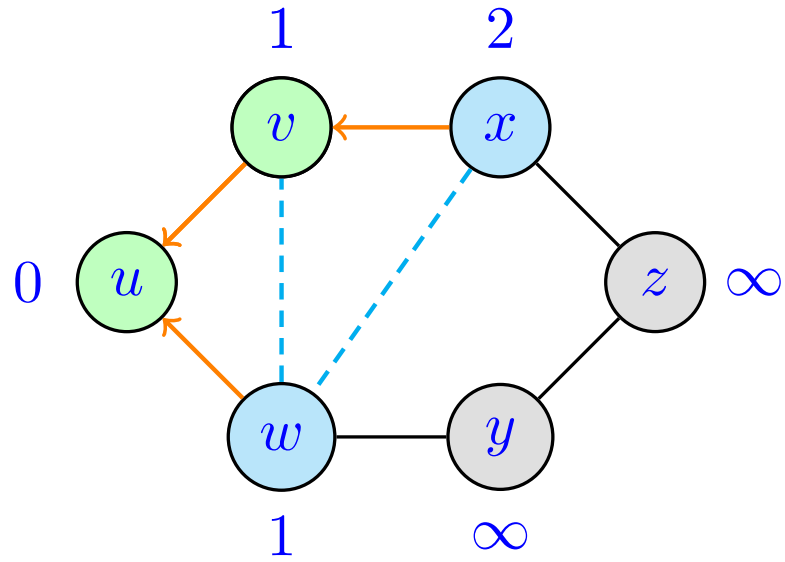
Simulação



$$
\begin{array}{lcl}
\mapsto & v & \\
Q & = & \langle w, x \rangle \\
d & = & (\underline{1, 1, 1, 2}, \infty, \infty) \\
p & = & (\underline{u, u, u, v}, y, z)
\end{array}
$$

Busca em Largura (BFS)

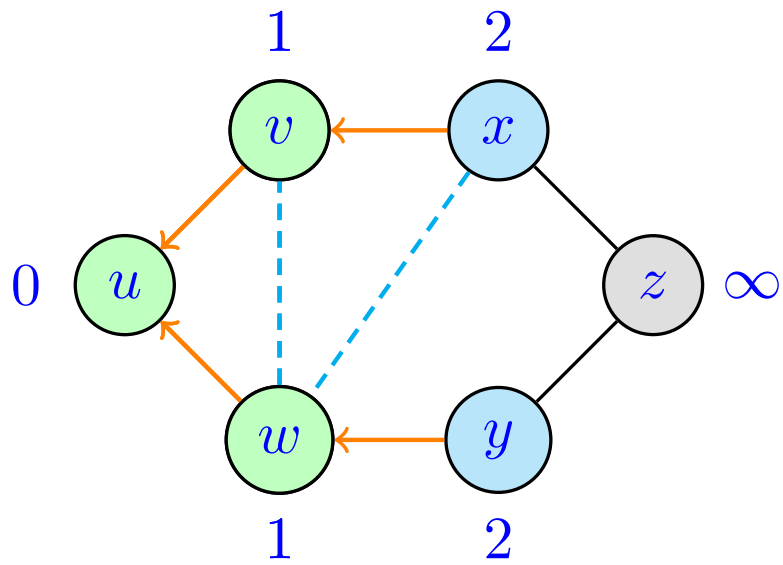
Simulação



$$
\begin{array}{lcl}
\mapsto & w & \\
Q & = & \langle x \rangle \\
d & = & (\underline{1,1,1,2},\infty,\infty) \\
p & = & (\underline{u,u,u,v},y,z)
\end{array}
$$

Busca em Largura (BFS)

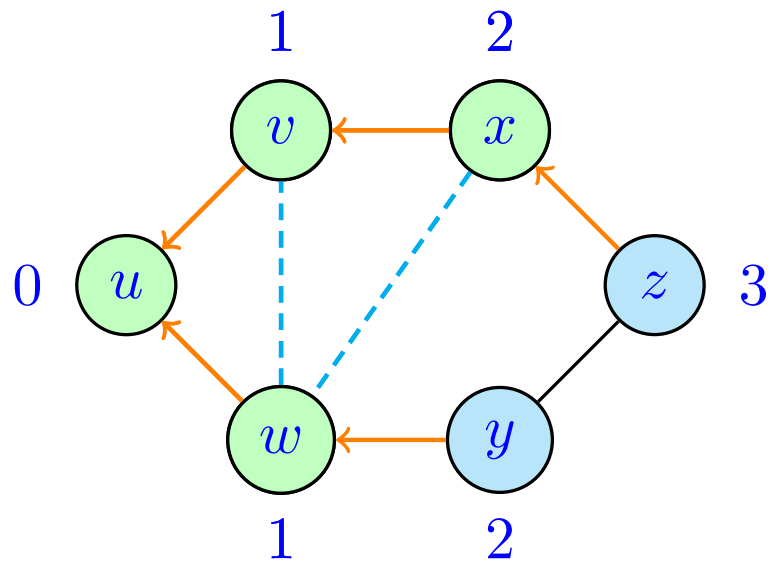
Simulação



$$
\begin{array}{lcl}
\mapsto & w & \\
Q & = & \langle x, y \rangle \\
d & = & (\underline{1, 1, 1, 2, 2}, \infty) \\
p & = & (\underline{u, u, u, v, w}, z)
\end{array}
$$

Busca em Largura (BFS)

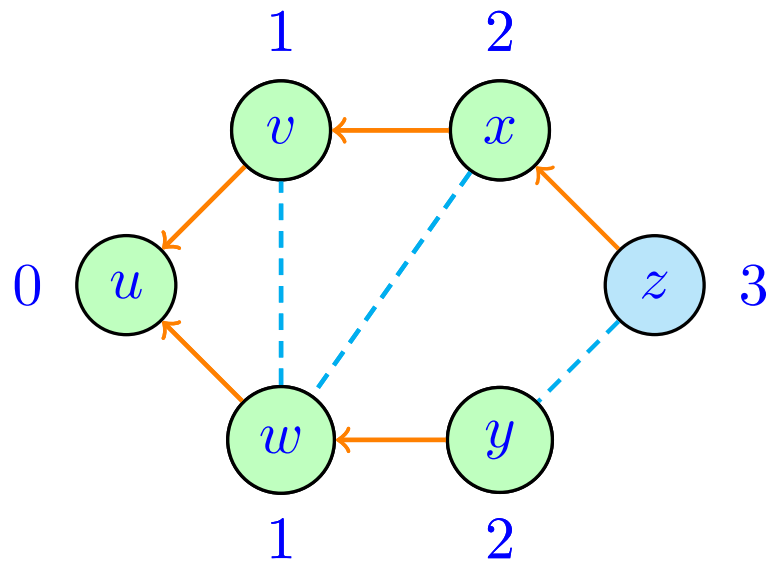
Simulação



$$
\begin{array}{lcl}
\mapsto & x & \\
Q & = & \langle y, z \rangle \\
d & = & (\underline{1, 1, 1, 2, 2, 3}) \\
p & = & (\underline{u, u, u, v, w, x})
\end{array}
$$

Busca em Largura (BFS)

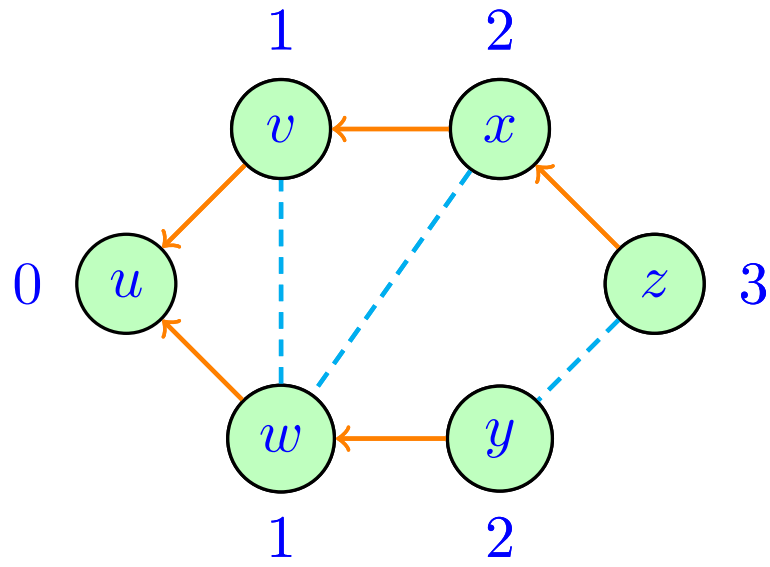
Simulação



$$
\begin{array}{lcl}
\mapsto & y & \\
Q & = & \langle z \rangle \\
d & = & (\underline{1,1,1,2,2,3}) \\
p & = & (\underline{u,u,u,v,w,x})
\end{array}
$$

Busca em Largura (BFS)

Simulação



$$
\begin{array}{lcl}
\mapsto & z & \\
Q & = & \langle\rangle \\
d & = & (\underline{1,1,1,2,2,3}) \\
p & = & (\underline{u,u,u,v,w,x})
\end{array}
$$

Diâmetros

O *diâmetro* de G é sua maior distância, o comprimento do caminho mínimo mais longo em G:

$$\mathsf{diam}(G) := \max_{u,v \in V} \mathsf{dist}(u,v) = \max_{u,v \in V} \min_{P \in \mathscr{P}(u,v)} \{|P|\}.$$

O diâmetro de um grafo é uma medida do quão espalhado ele é.

Por exemplo, grafos oriundos de redes sociais costumam possuir diâmetros pequenos, apesar de serem esparsos.

Podemos calcular o diâmetro de G em tempo $O\big(|V|^2 + |V| \cdot |E|\big) = O\big(|V|^3\big)$ resolvendo $|V|$ problemas de caminhos mínimos, um para cada vértice do grafo como origem, e tomando a maior dentre todas as $|V|^2$ distâncias determinadas.

Tours, Circuitos e Ciclos

Variações fechadas (iniciando e terminando num mesmo vértice) de passeios, trilhas e caminhos.

Assim, um passeio

$$C = \langle v_0, v_1, \ldots, v_k = v_0 \rangle$$

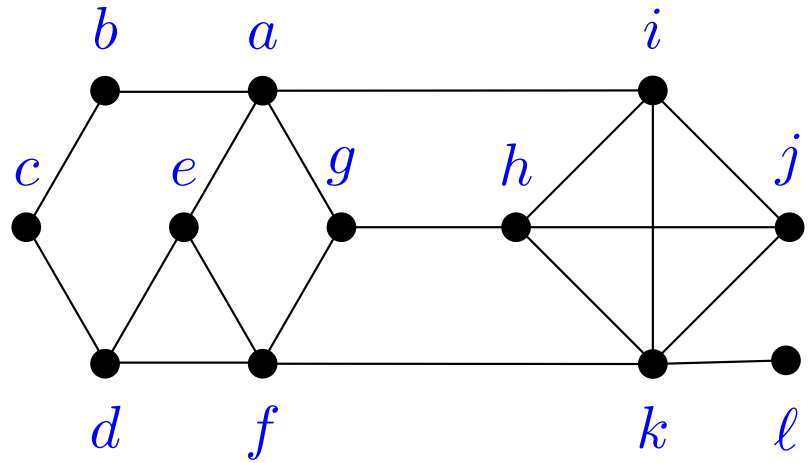
é um *tour* de *comprimento* $|C| = k$.

Um tour C é um *circuito* se não repete arestas e é um *ciclo* se $v_0, v_1, \ldots, v_{k-1}$ são distintos e $v_k = v_0$.

Um ciclo é *trivial* se $|C| = 0$. Ciclos não triviais tem comprimento maior ou igual a 3.

Um ciclo C é *par*/*ímpar* se seu comprimento é par/ímpar.

Tours, Circuitos e Ciclos



No grafo acima:

- $\langle a, e, f, g, a, g, h, k, \ell, k, f, g, a, b, c, d, e, a \rangle$ é um tour;
- $\langle a, i, j, k, i, h, g, a, e, d, c, b, a \rangle$ é um circuito;
- $\langle a, b, c, d, e, f, k, h, i, a \rangle$ é um ciclo;
- $\langle a, b, c, d, e, a \rangle$ e $\langle a, g, f, k, i \rangle$ são ciclos ímpares;
- $\langle a, e, f, g \rangle$, $\langle a, g, h, i \rangle$ e $\langle a, e, f, g, k, h, i, a \rangle$ são ciclos pares.

Tours, Circuitos e Ciclos

Teorema

O conjunto de arestas de um grafo G pode ser particionado em ciclos distintos se e somente se todo vértice de G possui grau par.

Prova. A necessidade é imediata.

Sejam $C_1, C_2, \ldots, C_k \subseteq G$ ciclos dois a dois distintos tais que $\bigcup_{i=1}^{k} V(C_i) = V(G)$ e $\bigcup_{i=1}^{k} E(C_i) = E(G)$.

Logo, $d_G(u) = \sum_{i=1}^{k} d_{C_i}(u)$, para todo $u \in V(G)$. Por definição, $d_{C_i}(u)$ é par para todo $i \in [k]$ e o resultado segue.

Para suficiência, suponha que todos os vértices de G tem grau par.

Tours, Circuitos e Ciclos

Continuando...

A prova é por indução no número de arestas $m = |E(G)|$. Caso $m = 0$, o resultado é trivialmente válido.

Suponha então que $m > 0$ e que o resultado é válido para qualquer subgrafo de G com menos de m arestas.

Seja $u \in V(G)$ tal que $d(u) > 0$ e seja P um caminho mais longo em G começando em u – um tal caminho pode ser obtido via `DFS-Visit` à partir de u.

Como todos os vértices em G tem grau par, existem vértices distintos $v, w \in V(P)$ tais que $\{v, w\} \in E(G) \setminus E(P)$. Temos, assim, que $C = \langle v, \ldots, w, v \rangle$ é um ciclo em G.

Por hipótese de indução, $G \setminus E(C)$ tem uma decomposição em ciclos $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$. Claramente, $\mathcal{C} \cup \{C\}$ é uma decomposição em ciclos para G. □

Circuitos Eulerianos

Um circuito é *euleriano* se contém todas as arestas do grafo. Um grafo é euleriano contém um circuito euleriano. Claramente, qualquer grafo euleriano tem de ser conexo.

Corolário

Um grafo é euleriano se e somente se é conexo e todos os vértices possuem grau par.

Prova. A necessidade segue diretamente do teorema anterior.

Para suficiência, seja G conexo com todos os vértices de grau par, e seja $C_1, C_2, \ldots, C_k$ uma decomposição de G em ciclos fornecida pelo teorema anterior.

Como G é conexo, $V(C_i) \cap V(C_j) \neq \emptyset$ para todo par $1 \leq i \neq j \leq k$.

Circuitos Eulerianos

Continuando...

Suponha que $C_i = \langle u_0^i, u_1^i, \ldots, u_{k_i}^i = u_0^i \rangle$. Para $j = 2, 3, \ldots, k$, faça: seja $u_\ell^1 = u_r^j \in V(C_1) \cap V(C_j)$ e redefina C_1 como

$$C_1 = \langle u_0^1, u_1^1, \ldots, u_\ell^1 = u_r^j, u_{r+1}^j, \ldots, u_{k_j}^j, u_1^j, \ldots, u_{r-1}^j, u_{\ell+1}^1, \ldots, u_{k_1}^1 \rangle;$$

renomeie os $t = k_1 + k_j - 1$ vértices em C_1, à partir de u_1^1, como $\langle u_1^1, u_2^1, \ldots, u_t^1 \rangle$ e faça $k_1 = t$; proceda com o próximo passo do laço em j. Ao final, C_1 é claramente um circuito euleriano. □

A prova acima fornece um teste simples para decidir se um grafo é euleriano e, em caso afirmativo, um algoritmo para determinar tal circuito (exiba-o!).

Grafos Biparticionáveis

Lembre-se que um grafo $G = (V, E)$ é biparticionável se admite uma bipartição (X, Y) — $X \cup Y = V$ e $X \cap Y = \emptyset$ — de V tal que todas as arestas em E tem uma ponta em X e a outra em Y.

Exercício

Prove que G é biparticionável se e somente se todo ciclo em G tem comprimento par.

Exercício

Forneça um algoritmo que recebe um grafo G, decide se ele é biparticionável e, em caso afirmativo, devolva uma bipartição para $V(G)$.