BCM0506 – Comunicação e Redes

Aula 6

Caminhos Mínimos em Grafos e Digrafos; Florestas e Árvores

Prof. Dr. Aritanan Gruber

Centro de Matemática, Computação e Cognição
Universidade Federal do ABC

Quadrimestre Suplementar 2020

Grafos Bipartidionáveis

Na aula passada, deixamos o seguinte:

Exercício

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

Prova. Necessidade. Suponha que G é bipartidionável com bipartição (X, Y) e seja $C = \langle v_1, v_2, \ldots, v_k, v_1 \rangle$ um ciclo em G de comprimento $k > 2$. Suponha que $v_1 \in X$. Assim, $v_{2\ell} \in Y$ e $v_{2\ell+1} \in X$, para $0 \leq \ell \leq \lfloor k/2 \rfloor$.

Caso k seja ímpar, temos que $v_k \in X$ e que $|\{v_k, v_1\} \cap X| = 2$, ou seja, dois vértices em X são vizinhos, o que contraria a hipótese de G ser bipartidionável. Logo, C é um ciclo par.

Grafos Biparticionáveis

Suficiência. Suponha que G não possui ciclos ímpares e seja $r \in V$. Utilizando a função de distância, defina

$$X := \{v \in V : \mathsf{dist}(r, v) \equiv 0 \pmod 2\} \quad \text{e} \quad Y := V \setminus X.$$

Claramente, $X \cup Y = V$ e $X \cap Y = \emptyset$. Resta mostrar que $E \subseteq X \times Y$.

Suponha que $e = \{u, v\}$ é uma aresta com ambas as pontas em X ou em Y e sejam P_u e P_v (r, u)- e (r, v)-caminhos mínimos, com $P_u \triangle P_v$ mínimo.

Seja s o vértice mais distante de r comum a P_u e P_v. Defina:

- R como o (r, s)-subcaminho comum a P_u e P_v,
- Q_u como o (s, u)-subcaminho de P_u,
- Q_v como o (s, v)-subcaminho de P_v.

Grafos Biparticionáveis

Como $u, v \in X$ ou $u, v \in Y$, e R é comum a P_u e P_v, temos que

$$|P_u| + |P_v| = 2|R| + |Q_u| + |Q_v|$$

e Q_u e Q_v tem a mesma paridade. Assim, $Q_u \cup Q_v$, o caminho entre u e v passando por s tem comprimento par.

Mas isto implica em $(Q_u \cup Q_v)+e$ sendo um ciclo ímpar, contrariando a hipótese. Logo, $e \notin E$ e (X, Y) é uma bipartição de G. □

Deixamos ainda o:

Exercício

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

Grafos Biparticionáveis

A essa altura, você já deve ter percebido que uma modificação no algoritmo de busca em largura (BFS) resolve a questão.

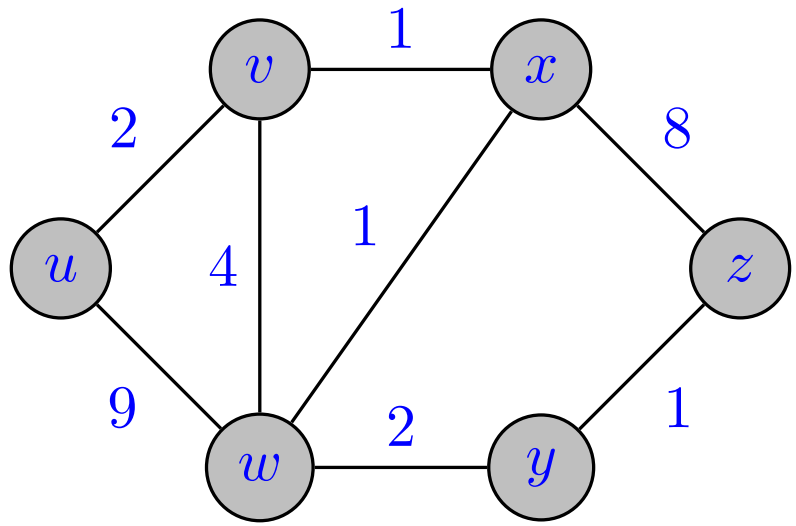
Especificamente, quando o vértice u é removido da fila Q e sua vizinhança está sendo inspecionada: se $v \in N(u)$ com $d(v) < \infty$ e tal que $d(u)$ e $d(v)$ têm a mesma paridade, foi descoberto um ciclo ímpar! E ele pode ser reconstruído via o vetor de predecessores.

Caso o grafo não tenha ciclos ímpares, ao final da execução de BFS definimos $X := \{u \in V : d(u) \equiv 0 \pmod 2\}$ e $Y := V \setminus X$.

Os detalhes de implementação ficam por sua conta.

Caminhos Mínimos com Distâncias Não Negativas

Suponha agora que temos um grafo $G = (V, E)$ e uma *função de custo* $c : E \to \mathbb{R}_{\geq 0}$ que associa valores reais não negativos às arestas.

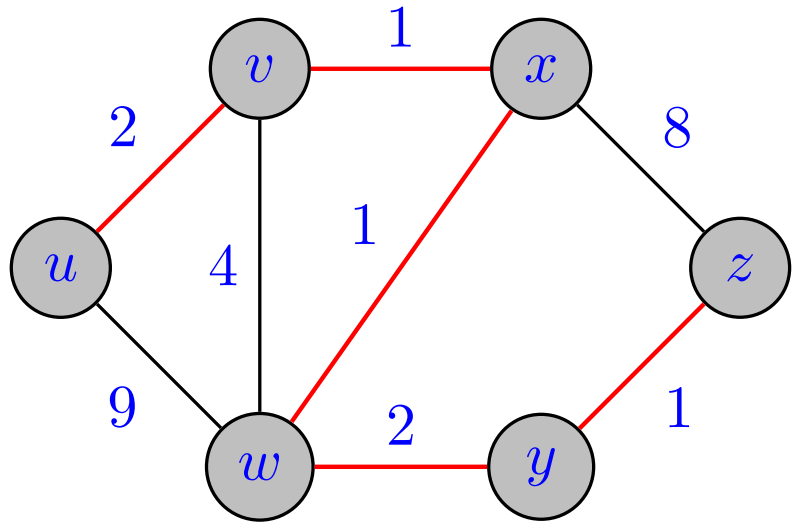


Se usarmos $c(e)$, o custo da aresta e, como distância, não é difícil perceber que a solução baseada em BFS não fornece uma resposta correta. De fato, um caminho de custo mínimo de u a z tem valor 7, mas uma busca em largura devolve 11, 12 ou 18.

Caminhos Mínimos com Distâncias Não Negativas

A explicação é imediata: a busca em largura prioriza caminhos "curtos," com um menor número de arestas (por isso, distâncias unitárias).

No exemplo com custos (distâncias) variáveis, temos que um caminho mais "barato" entre u e z é $\langle u, v, x, w, y, z \rangle$.



Nota:: A distinção entre distância e custo é conveniente.

Caminhos de Custo Mínimo

Origem única

Problema (caminhos de custo mínimo)

Dados $G = (V, E)$, uma origem $s \in V$ e uma função custo $c \in \mathbb{R}^E_{\geq 0}$, determine caminhos de custo mínimo de s aos demais vértices.

O *custo* $c(P)$ de um caminho $P = \langle v_0, v_1, \ldots, v_k \rangle$ é

$$c(P) := \sum_{i=1}^{k} c(\{v_{i-1}, v_i\}).$$

O custo entre dois vértices u e v é definido como

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

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

Caminhos de Custo Mínimo

Origem única

Claramente, $P \in \mathscr{P}(u, v)$ é um (u, v)-*caminho de custo mínimo* se $c(P) = \mathsf{cost}(u, v)$.

Não surpreendente, caminhos de custo mínimo também possuem sub-estrutura ótima.

Lema (Sub-estrutura ótima)

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

A prova é similar ao caso unitário e fica como exercício.

Temos uma recorrência semelhante, que considera os custos:

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

Caminhos de Custo Mínimo

Origem única

Lembre-se que caminhos não repetem vértices e que, apesar de reversíveis, induzem um sentido nas arestas via ordem dos vértices.

Nota: Além disso, como os custos são não-negativos, não haveria ganho algum no uso de qualquer aresta mais de uma vez.

Seja $\vec{G} = (V, \vec{E})$ uma ordenação das arestas de G: cada aresta em E é ordenada como um arco em $\vec{E}$.

Um *potencial* para os vértices de $\vec{G}$ é um vetor $\boldsymbol{y} \in \mathbb{R}^V$. Um potencial é *viável* se

$$\boldsymbol{y}(v) \leq \boldsymbol{y}(u) + c(u, v) \qquad \text{para todo arco} \qquad (u, v) \in \vec{E},$$

em que $c : E \to \mathbb{R}_{\geq 0}$ é uma função custo para as arestas de G.

Caminhos de Custo Mínimo

Origem única

Podemos supor que $\boldsymbol{y}(s) = 0$ para algum vértice $s \in V$. De fato, a subtração de $\boldsymbol{y}(s)$ em ambos os lados de todas as relações acima não inviabiliza o potencial.

Lema

Para todo (s,t)-caminho P em $\vec{G}$ e para todo potencial viável $\boldsymbol{y}$, $c(P) \geq \boldsymbol{y}(t) - \boldsymbol{y}(s)$.

Prova. Seja $P = \langle s = u_0, u_1, \ldots, u_k = t \rangle$ um (s,t)-caminho em $\vec{G}$. Temos que

$$\begin{aligned} c(P) &= \sum_{(u_{i-1},u_i) \in P} c(u_{i-1}, u_i) \\ &\geq \sum_{(u_{i-1},u_i) \in P} \big(\boldsymbol{y}(u_i) - \boldsymbol{y}(u_{i-1})\big) \\ &= \boldsymbol{y}(t) - \boldsymbol{y}(s). \end{aligned}$$

□

Caminhos de Custo Mínimo

Origem única

Em particular, o Lema implica a dualidade fraca

$$\min_{P \in \mathscr{P}(s,t)} c(P) \geq \max_{\rho \in \mathbb{R}^V} \boldsymbol{y}(t) - \boldsymbol{y}(s),$$

que um (s,t)-potencial máximo limita inferiormente o custo de um (s,t)-caminho de custo mínimo.

Logo, se começarmos com um potencial ∞ para todos os vértices com exceção da origem s, cujo potencial é 0 e diminuirmos os potenciais mediante a recorrência apresentada até que sejam viáveis, teremos um limitante inferior ao valor do caminho de custo mínimo.

Que a desigualdade acima vale com igualdade está além do escopo deste curso; mas certamente usaremos este fato no desenvolvimento de um algoritmo: o de Dijkstra.

Caminhos de Custo Mínimo

Origem única

Pergunta: E qual ordenação $\vec{G}$ de G devemos usar?

O belo da estória: a que for induzida por uma busca em largura modificada para em vez de uma fila *first-in, first-out*, usar uma fila de prioridades.

Se iterarmos a recorrência para cost como fizemos com a para dist, vemos que a escolha ainda é gulosa: o mínimo agora é tomado dentre os limitantes superiores de potenciais.

O uso de uma fila de prioridades H garante que, a cada passo, vamos retirar da fila um vértice u com o menor limitante – i.é, tal que $\boldsymbol{y}[u]$ é mínimo; este limitante estará correto, pois os custos são ≥ 0 – diminuir os limitantes dos vizinhos dele, e prosseguir até que tenhamos um potencial viável (que, no caso, será máximo).

Caminhos de Custo Mínimo: Dijkstra

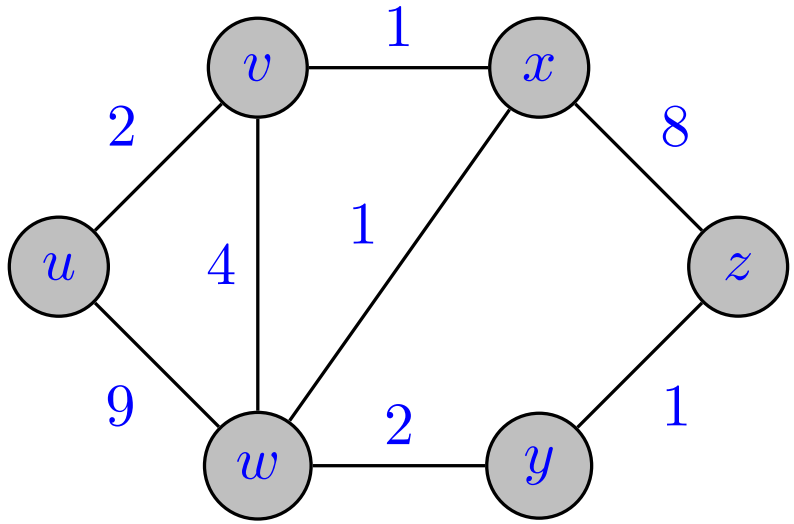
```
1  Dijkstra (G, c, s)
2  foreach u ∈ V(G) do
3    y[u], p[u] ← ∞, u
4  y[s] ← 0
5  H ← priority-queue (V(G))
6  while H ≠ ∅ do
7    u ← remove-minimum (H)
8    foreach v ∈ N(u) do
9      if y[v] > y[u] + c(u, v) then
10       y[v], p[v] ← y[u] + c(u, v), u
11 return y, p
```

Lembre-se: na linha 7, u é escolhido em H tal que $\boldsymbol{y}[u]$ é mínimo.

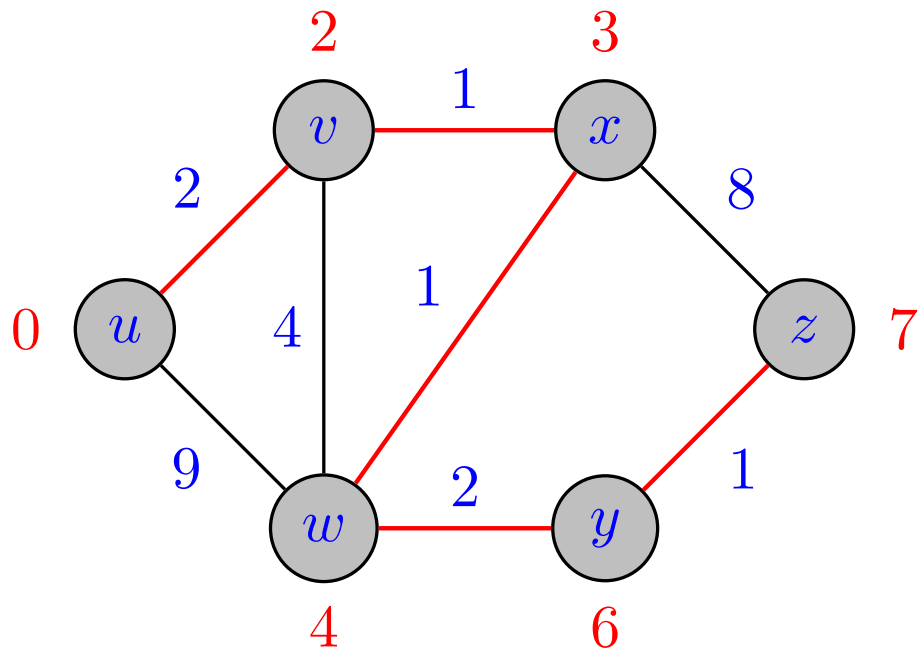
O tempo consumido pelo algoritmo claramente depende das operações na fila de prioridade H. No caso de G denso, uma simples implementação de H como um vetor garante que Dijkstra devolve uma resposta correta em tempo $O(|V|^2)$.

Caminhos de Custo Mínimo: Dijkstra

Simulando Dijkstra no grafo abaixo (lousa)



obtemos



Caminhos de Custo Mínimo

Entre todos os pares

É possível utilizarmos o algoritmo de Dijkstra $|V| - 1$ vezes para determinarm todos os (s, t)-caminhos de custo mínimos em G, para todos os pares de vértices s e t.

Uma forma mais fácil e rápida consiste na utilização do algoritmo desenvolvido por Floyd e por Warshall.

Fixe a ordenação dos vértices em $V = \{1, 2, \ldots, n\}$. Para todo par $i, j \in V$ e todo $k = 0, 1, 2, \ldots, n$, defina $\mathsf{cost}_k(i, j)$ como o custo do melhor caminho de i a j podendo utilizar somente $1, 2, \ldots, k$ como vértices internos (se $k = 0$, nenhum vértice pode ser utilizado).

Caminhos de Custo Mínimo

Entre todos os pares

Temos então a seguinte recorrência: $\mathsf{cost}_k(i,j) =$

$$\begin{cases} 0 & \text{se } i = j, \\ c(\{i,j\}) & \text{se } i \neq j \text{ e } k = 0 \\ \min\left\{\mathsf{cost}_{k-1}(i,j), \mathsf{cost}_{k-1}(i,k) + \mathsf{cost}_{k-1}(k,j)\right\} & \text{se } i \neq j \text{ e } 1 \leq k \leq n. \end{cases}$$

Não é difícil perceber que ela fornece um algoritmo para o cômputo dos caminhos de custo mínimo entre todos os pares de vértices de G e que o tempo requerido por este algoritmo é $\Theta(n^3)$.

A implementação fica mais clara em forma matricial. Seja $W \in \mathbb{R}_{\geq}^{V \times V}$ a matriz de pesos do grafo G, isto é, para todo $i, j \in V$,

$$W_{i,j} = \begin{cases} 0 & \text{se } i = j, \\ c(\{i,j\}) & \text{se } i \neq j \text{ e a aresta } \{i,j\} \in E, \\ \infty & \text{em caso contrário.} \end{cases}$$

Caminhos de Custo Mínimo: Floyd-Warshall

```
1 Floyd-Warshall (G, W)
2 foreach k ∈ V(G) do
3   foreach i ∈ V(G) do
4     foreach j ∈ V(G) do
5       W_{i,j} ← min{W_{i,j}, W_{i,k} + W_{k,j}}
6 return W
```

Para a matriz W devolvida, temos que $W_{i,j} = \mathsf{cost}_n(i, j)$.

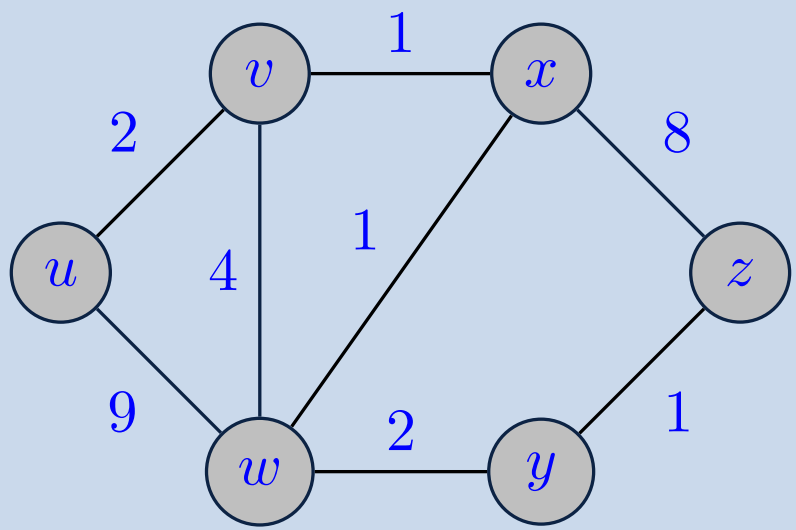
Exercício (trivial)

Escreva uma versão em Python; use $V = \{0, 1, \ldots, n-1\}$.

Caminhos de Custo Mínimo: Floyd-Warshall

Exercício

Simule o algoritmo de Floyd-Warshall no grafo abaixo.



Digrafos (= Digraphs)

Definição

Digrafos ou grafos dirigidos (ou ainda, orientados) são úteis para modelagem e tratamento de situações em que as relações entre pares de objetos não são simétricas. Por exemplo, existe um sentido natural de uso ou de precedência.

Definição

Um digrafo é um par ordenado (V, E), com V um conjunto finito e

$$E \subseteq V \times V := \{(u, v) : u, v \in V\}$$

uma família de pares ordenados de elementos em V. Cada $v \in V$ é chamado vértice e cada $e \in E$ é denominado arco. Um arco do tipo (u, u) é um laço e arcos (u, v) e (v, u), com $u \neq v$, são anti-paralelos.

Para um arco $e = (u, v)$, u é a *cauda* de e, e v é a *cabeça* de e.

Digrafos

Existem 2^{n^2} digrafos rotulados distintos em n vértices: agora, podemos ter dois arcos anti-paralelos entre quaisquer par de vértices e cada vértice pode ter um laço.

Muitos dos conceitos que introduzimos para grafos possuem análogos em digrafos. Alguns são trivialmente transportados, como igualdade, isomorfismo e subgrafos.

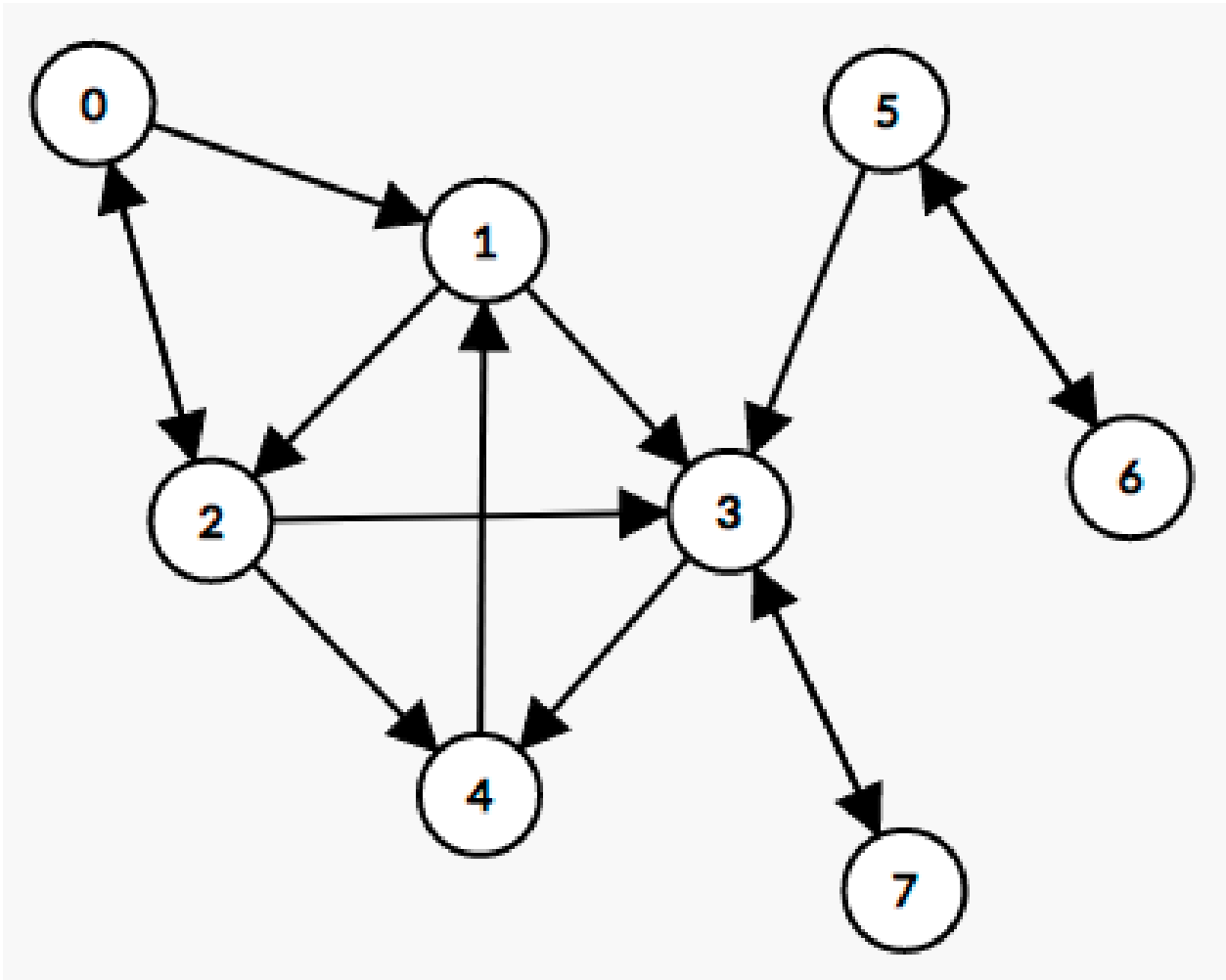
Outros, requerem pequenas adaptações para lidar com as direções impostas pelos pares ordenados de vértices. Exemplos:

- vizinhanças e graus:
  $u \in V$ tem a vizinhança de *saída* $N^+(u) := \{(u, v) \in E : v \in V\}$ e a de *chegada* $N^-(u) := \{(v, u) \in E : v \in V\}$;
- matrizes de adjacência (não-simétricas):
  para $(u, v) \in E$, $\boldsymbol{A}_{u,v} = 1$ e $\boldsymbol{A}_{v,u} = 0$; $\boldsymbol{A}_{u,v} = 0$ em c.c.;
- matrizes de incidência:
  tratamento similar; caudas recebem 1, cabeças -1.

Digrafos

Exemplo: $D_1 = (V, R)$ com $V = \{0, 1, 2, 3, 4, 5, 6, 7\}$, e

$$R = \{(0,1), (0,2), (1,2), (1,3), (2,0), (2,3), (2,4), (3,4), (3,7), (4,1), (5,3), (5,6), (6,5), (7,3)\}.$$



Digrafos

Matriz de adjacências $\boldsymbol{A}(D_1)$ não simétrica:

$$\boldsymbol{A}(D_1) = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{pmatrix}.$$

Digrafos

O lema dos apertos de mãos para digrafos é:

Lema

Para todo digrafo $D = (V, E)$, vale que

$$\sum_{u \in V} d^+(u) = \sum_{v \in V} d^-(v) = |E|.$$

Passeios, trilhas, caminhos e tours, circuitos e ciclos já induziam uma ordem natural nas arestas do grafo. A diferença em digrafos é que temos que respeitar o sentido dos arcos existentes. Isto é, podemos ir de u a v com o arco $(u, v) \in E$, mas não podemos ir de v para u se $(v, u) \notin E$.

Ponto importante: os algoritmos desenvolvidos até aqui (DFS, BFS, Dijkstra, Floyd-Warshall) funcionam tanto em grafos quanto em digrafos. Nenhuma modificação é necessária.

Digrafos

Já usamos, intuitivamente, o conceito de *orientação* de um grafo em um digrafo quando desenvolvemos algoritmos para caminhos de custo mínimo.

A operação recíproca é igualmente intuitiva, módulo dois detalhes: dado um digrafo, obtemos um grafo *subjacente* ignorando a direção dos arcos e tratando-os como arestas, desde que eliminemos os laços e um dos arcos anti-paralelos — em caso contrário, obteríamos um pseudo-grafo e não um grafo.

Um digrafo D é *conexo* se o grafo subjacente é conexo. D é *fortemente conexo* se para quaisquer dois vértices, existe um caminho (dirigido) entre eles. Logo, um digrafo pode ser conexo e não fortemente conexo!

Digrafos

Conceitos como cliques e conjuntos estáveis são, em geral, estudados junto aos grafos subjacentes dos digrafos; há exceções, mas não neste curso.

Uma classe de objetos que possui um tratamento "misto" é a de florestas e árvores.

Já esbarramos com as duas antes. Nos próximos slides, vamos desenvolver um mínimo de nomenclatura e resultados para elas em grafos. As versões orientadas de digrafos serão feitas em outro curso.

Nota: Em vez de ficarmos adaptando todos os conceitos e elicitando as diferenças aqui, vamos fazê-los quando o for necessário. Algumas adaptações e diferenças serão trabalhadas em listas de exercícios.

Caminhos Mínimos em Grafos e Digrafos

Aplicações:

- waze, Google Maps;
- iRobot Roomba, Braava;
- Roteamento em redes de computadores (internet);
- Justificação de texto em LaTeX;
- Planejamento em IA;
- Importância estrutural em redes complexas;
- ...

Florestas e Árvores

Um grafo livre de ciclos é dito *acíclico*. Grafos acíclicos são também chamados *florestas*. Um grafo acíclico e conexo é uma *árvore*.

Toda floresta é composta por árvores, e toda árvore é um componente em uma floresta. Uma floresta T para um grafo G é *maximal* se T e G possuem o mesmo número de componentes.

Theorem

Um grafo conexo G é uma árvore se e somente se para qualquer par de vértices $u, v \in V(G)$ existe um único caminho entre u e v.

Prova. Para necessidade, seja G conexo e acíclico e suponha que, para algum par de vértices $u, v \in V(G)$, existem dois caminhos distintos

$$P_1 = \langle u = u_0, u_1, \ldots, u_k = v \rangle \quad \text{e} \quad P_2 = \langle u = v_0, v_1, \ldots, v_\ell = v \rangle$$

entre u e v.

Florestas e Árvores

Seja x o primeiro vértice em que P_1 e P_2 diferem partindo de u, e seja $y \neq x$ o primeiro vértice após a separação em que P_1 e P_2 se encontram.

Certamente x existe, pois P_1 e P_2 são distintos, e y existe, pois P_1 e P_2 terminam ambos em v. Considere agora os subcaminhos $S_1 = \langle x = u_i, \ldots, u_j = v \rangle$ de P_1 e $S_2 = \langle u = v_p, \ldots, v_q = v \rangle$ de P_2.

Temos então que o subgrafo $S_1 + S_2$ é um ciclo e que isto contradiz a hipótese de aciclicidade de G. Portanto, existe somente um caminho entre u e v em G.

Para suficiência, apenas observe que G é conexo e que a existência de qualquer ciclo envolvendo quaisquer dois vértices $u, v \in V(G)$ com $u \neq v$ implica a existência de dois (u, v)-caminhos distintos (e disjuntos internamente), contradizendo a hipótese. $\square$

Florestas e Árvores

Corolário

Um grafo é uma floresta se e somente se toda aresta do grafo é uma ponte.

O Corolário acima implica que todo grafo minimamente conexo — com o menor número de arestas — é uma árvore. O próximo resultado quantifica esta relação e fornece uma boa caracterização de florestas e árvores.

Teorema (exercício)

Um grafo G é uma floresta com $\kappa(G)$ árvores se e somente se $|E(G)| = |V(G)| - \kappa(G)$.

Florestas e Árvores Geradoras

Seja $G = (V, E)$ um grafo conexo. Uma *árvore geradora* para G é um subgrafo gerador $T \subseteq G$ que é uma árvore. Isto é, $T = (V, F)$ é acíclico, conexo, com $F \subseteq E$ e $|F| = |V| - 1$.

No caso de G não ser conexo, falamos em *floresta geradora* e *floresta geradora maximal*.

Seja $T = (V, F)$ uma árvore geradora para um grafo $G = (V, E)$ e seja $e = \{x, y\} \in E \setminus F$ uma aresta em G mas não em T. O ciclo em $T + e$ envolvendo x e y é chamado *ciclo fundamental* induzido por e em T.

Proposição (exercício)

Para $G = (V, E)$ conexo e $T = (V, F)$ uma árvore geradora para G, cada aresta $e \in E \setminus F$ induz um e somente um ciclo fundamental em $T + e$.

Conectores e Árvores Geradoras Mínimas

Sejam $G = (V, E)$ um grafo conexo e $w : E \to \mathbb{R}$ uma função que associa um *peso* $w(e)$ a cada aresta $e \in E$. Para qualquer subconjunto de arestas $F \subseteq E$, defina

$$w(F) := \sum_{e \in F} w(e)$$

como a extensão da função w sobre F. Finalmente, para qualquer subgrafo $H \subseteq G$, defina $w(H) := w(E(H))$.

Um *conector* para G é um subgrafo gerador conexo. Um conector C^* para G é *mínimo* ou tem *peso mínimo* se

$$C^* = \operatorname{argmin} \{w(C) : C \subseteq G \text{ é um conector}\}.$$

Isto é, $w(C^*) \leq w(C)$ para todo conector C de G.

Observe que toda árvore geradora de G é um conector, mas que a recíproca não é verdadeira.

Conectores e Árvores Geradoras Mínimas

Uma árvore geradora T^* para G é *mínima* ou tem *peso mínimo* se

$$T^* = \operatorname{argmin}\{w(T) : T \subseteq G \text{ é uma árvore geradora}\}.$$

Isto é, $w(T^*) \leq w(T)$ para toda árvore geradora T de G.

Exercício

Mostre que $w(C^*) \leq w(T^*)$ para todo conector mínimo C^* e toda árvore geradora mínima T^* de G.

Problema

Dados um grafo conexo $G = (V, E)$ e uma função peso $w \in \mathbb{R}^E$, determine uma árvore geradora mínima para G.