Análise e Implementação de Algoritmos de Busca de uma r-Arborescência Inversa de Custo Mínimo em Grafos Dirigidos com Aplicação Didática Interativa

Orientador: Mário Leston
Discentes: Lorena Silva Sampaio, Samira Haddad

25 de setembro de 2025

1 Introdução

Encontrar uma *r-arborescência inversa de custo mínimo* em grafos dirigidos é um problema estudado em ciência da computação desde os anos 1960, com formulações fundamentais apresentadas por Jack Edmonds em 1967 [**edmonds1967optimum**].

Essa busca dialoga com um princípio formulado na Idade Média por Guilherme de Ockham: a navalha de Occam (princípio da parcimônia), uma heurística filosófica segundo a qual, entre explicações concorrentes para um fenômeno, devemos preferir a mais simples ou a que faz menos suposições.

Podemos pensar na navalha de Occam como critério de escolha entre explicações por meio de uma *teia explicativa mínima*: uma estrutura que conecta fatos ou observações com o mínimo de relações explicativas necessárias. Quando tais relações envolvem dependência ou causalidade, podemos representá-las pictograficamente como setas direcionadas entre os fatos (as hipóteses aparecem como rótulos dessas setas). Para refinar o modelo, associamos um custo a cada relação (por exemplo, o esforço para validar a relação ou a complexidade da explicação).

Encontrar a *teia explicativa mínima* equivale, nessa metáfora, a encontrar uma *r-arborescência de custo mínimo*: fixamos um vértice raiz r (a explicação inicial) e escolhemos um conjunto mínimo de relações explicativas de modo que todos os fatos tenham um caminho dirigido que leve a r, minimizando o custo total das arestas.

A *r-arborescência* (também chamada *out-arborescência*) orienta as arestas para fora de r: cada vértice $v \neq r$ tem exatamente uma aresta de entrada, e há um caminho dirigido único de r até v. Já a *r-arborescência inversa* (*in-arborescência*) orienta as arestas em direção a r: cada $v \neq r$ tem exatamente uma aresta de saída, e de cada vértice parte um caminho dirigido único até r [**edmonds1967optimum**, **frank2014**].

Com essa distinção em mente, este trabalho concentra-se na variante inversa. Formalmente, dado um grafo dirigido $G = (V, E)$ com custos $c : E \to \mathbb{R}^+$ nas arestas e um vértice raiz $r \in V$, procura-se uma *r-arborescência inversa* — isto é, uma árvore direcionada que atinja todos os vértices por caminhos dirigidos até r — que minimize o custo total das arestas selecionadas (cf. [**edmonds1967optimum**, **frank2014**]).

Nosso interesse, porém, não é apenas encontrar a arborescência mínima: o percurso até ela também importa, pois revela propriedades estruturais dos dígrafos e ilumina técnicas distintas de otimização. Por isso, investigamos duas rotas clássicas e complementares: (i) o algoritmo de Chu–Liu/Edmonds, que opera por normalização dos custos das arestas de entrada, seleção sistemática de arestas de custo zero e contração de ciclos até obter um grafo reduzido, seguida pela reexpansão para reconstrução da solução [**chu1965**, **edmonds1967optimum**]; e (ii) a abordagem dual, em duas fases, de András Frank, fundamentada em cortes dirigidos, na qual se maximiza uma função de cortes c-viável para induzir arestas de custo zero e, em seguida, extrai-se a arborescência apenas a partir dessas arestas [**frank2014**]. Embora assentados em princípios distintos — contração de ciclos no plano primal versus empacotamento/dualidade por cortes —, ambos os paradigmas produzem soluções ótimas e tornam explícitas a variedade de abordagens matemáticas que podem ser empregadas para resolver o mesmo problema.

Assim sendo, analisamos e implementamos, em Python, essas duas abordagens, e apresentaremos detalhes das implementações, desafios enfrentados e soluções adotadas. Realizamos testes de volume com milhares de instâncias geradas aleatoriamente, registrando resultados em arquivos CSV e de log; os custos obtidos pelo Chu–Liu/Edmonds e pelas duas variantes de András Frank coincidem, corroborando a correção das implementações.

Adicionalmente, desenvolvemos uma aplicação web com fins didáticos, utilizando o framework PyScript e as bibliotecas NetworkX e Matplotlib, que permitem construir grafos dirigidos interativamente, escolher o vértice-raiz, executar o algoritmo de Chu–Liu/Edmonds e acompanhar, passo a passo, a evolução do grafo e o registro detalhado da execução (log). A interface inclui operações de adicionar arestas com pesos, carregar um grafo de teste e exportar a instância em formato JSON, facilitando a experimentação por estudantes e educadores.

1.1 Justificativa

A busca por uma *r-arborescência inversa de custo mínimo* em grafos dirigidos é um problema clássico com aplicações em diversas áreas, como redes de comunicação, planejamento de rotas, análise de dependências e modelagem de processos. Mas, não precisamos dessa justificação prática para nos interessarmos pelo problema: a riqueza estrutural dos dígrafos e a variedade de técnicas algorítmicas disponíveis o tornam um excelente caso de estudo em otimização combinatória.

Do ponto de vista didático, a metáfora da "teia explicativa mínima" torna concreto o porquê de estudarmos arborescências enraizadas: ela mapeia perguntas sobre explicação, alcance e economia de recursos para estruturas dirigidas, servindo de fio condutor nas implementações e nos experimentos que apresentamos.

1.2 Objetivos

O objetivo principal deste trabalho é analisar, implementar e comparar duas abordagens clássicas para o problema de *r-arborescência de custo mínimo* em grafos dirigidos oferecendo uma aplicação web interativa que facilite o entendimento e a experimentação com o algoritmo de Chu–Liu/Edmonds e o método de András Frank, tornando-o acessível para estudantes e educadores.

1.3 Estrutura do Trabalho

Resumidamente, o trabalho abrange as seguintes frentes:

1. **Fundamentação teórica**: revisão da literatura sobre arborescências em grafos dirigidos, incluindo definições, propriedades e resultados relevantes.

2. **Análise teórica**: consolidação dos conceitos de dígrafos e arborescências, compondo as formulações primal (normalização de custos e contração/reexpansão de ciclos no algoritmo de Chu–Liu/Edmonds) e dual (cortes dirigidos e função c-viável no método de András Frank), destacando resultados e intuições estruturais.

3. **Implementação computacional**: implementação em Python das rotinas de normalização dos custos de entrada, construção de F^*, detecção e contração de ciclos e reconstrução da solução (Chu–Liu/Edmonds), bem como das duas fases do método de András Frank; além de uma suíte de testes automatizados em larga escala sobre instâncias aleatórias com até centenas de vértices, verificando a coincidência dos custos entre os métodos e registrando resultados em CSV e log.

4. **Aplicação pedagógica**: desenvolvimento de uma aplicação web interativa (PyScript + NetworkX + Matplotlib) que permite montar instâncias, escolher o vértice-raiz e acompanhar, passo a passo, a execução do algoritmo com visualização do grafo e dos pesos das arestas, log textual e importação/exportação em JSON para facilitar a reprodução de experimentos.

Deste modo, o trabalho entrega implementações verificadas de Chu–Liu/Edmonds e András Frank, um visualizador web interativo e testes de volume que confirmam a equivalência de custos, úteis ao estudo e ao ensino de arborescências.

2 Definições Preliminares

Neste capítulo, reunimos as noções matemáticas básicas necessárias para compreensão completa do texto.

Fixaremos notações e conceitos (conjuntos, relações, funções, dígrafos, propriedade em dígrafos, dígrafos ponderados, ramificações geradoras, arborescências, funções de custo, dualidade, problemas duais e algoritmos), até chegar à formulação do problema da r-arborescência inversa de custo mínimo e adiamos descrições algorítmicas para capítulos posteriores.

2.1 Conjuntos

Este trabalho depende profundamente da teoria dos conjuntos, podemos dizer que todos os objetos matemáticos que iremos utilizar nessa dissertação se reduzem a conjuntos e operações entre eles.

Um **conjunto** é uma agregação de objetos distintos com características bem definidas, chamados elementos ou membros do conjunto. Os conjuntos são geralmente representados por letras maiúsculas (por exemplo, A, B, C) e seus elementos são listados entre chaves (por exemplo, $A = \{1, 2, 3\}$). Dois conjuntos são iguais se contêm exatamente os mesmos elementos.

Podemos ter conjuntos de qualquer tipo de objeto, incluindo números, letras e elementos da natureza. Para motivar as definições ao longo do texto, usaremos dois exemplos complementares que vamos chamar de exemplos-mestres:

- (i) Considere um universo U composto por três conjuntos: árvores T, plantas P e fungos F — para praticar pertinência, inclusão e operações; e

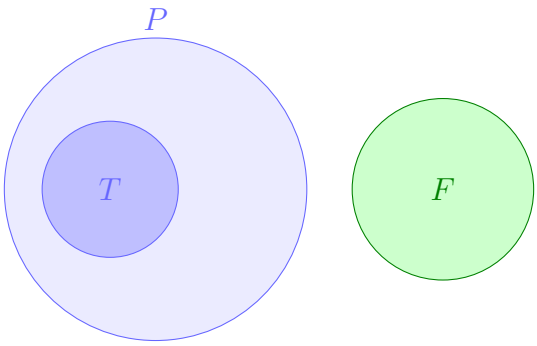


Figura 1: Relações entre os conjuntos de organismos: $T \subseteq P$ (toda árvore é planta) e F é disjunto de P.

- (ii) exemplo inspirado na navalha de Occam, com três famílias: evidências E, hipóteses H e explicações $\mathcal{M} \subseteq 2^H$. Nesse segundo exemplo, privilegiaremos explicações parcimoniosas: entre as que cobrem E, preferimos as minimais por inclusão.

No segundo exemplo, consideremos $E = \{\text{queda de temperatura, céu nublado}\}$ e $H = \{H_A, H_B\}$, em que H_A significa "frente fria" e H_B, "ilha de calor". Tanto $\{H_A\}$ quanto $\{H_A, H_B\}$ explicam E (cobrem ambas as evidências), mas, por parcimônia, preferimos $\{H_A\}$, por ser estritamente menor do que $\{H_A, H_B\}$ ($\{H_A\} \subset \{H_A, H_B\}$). Ao longo do texto, recorreremos às noções de pertinência (por exemplo, $H_A \in H$), de inclusão e às operações usuais sobre conjuntos (união, interseção etc.) para comparar explicações.

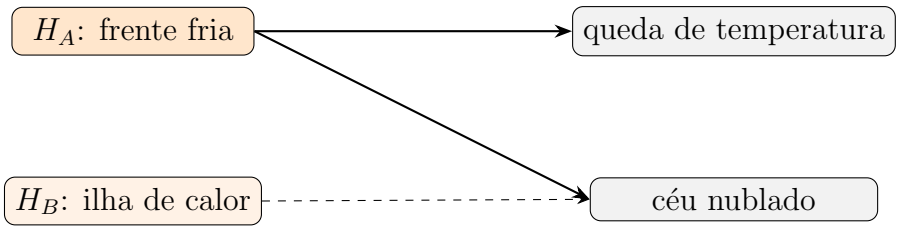


Figura 2: Exemplo inspirado na navalha de Occam: H_A cobre ambas as evidências (E), enquanto H_B seria redundante; prefere-se a explicação menor.

2.1.1 Subconjuntos

Dizemos que A é um **subconjunto** de B, denotado $A \subseteq B$, quando todo elemento de A também pertence a B. Se, além disso, $A \neq B$, escrevemos $A \subset B$ e chamamos A de **subconjunto próprio** de B. Ex.: $\{1, 2\} \subseteq \{1, 2, 3\}$ e $\{1, 2\} \subset \{1, 2, 3\}$.Por convenção, o conjunto vazio $\varnothing$ é subconjunto de qualquer conjunto X (isto é, $\varnothing \subseteq X$), e todo conjunto é subconjunto de si mesmo ($X \subseteq X$).

No primeiro exemplo-mestre, sejam P o conjunto de plantas, T o de árvores e F o de fungos; então $T \subseteq P$ (toda árvore é planta), ao passo que $F \nsubseteq P$.

No segundo, seja $H = \{H_A, H_B\}$ e $E = \{\text{queda de temperatura, céu nublado}\}$, vale $\{H_A\} \subset \{H_A, H_B\} \subseteq H$; ambas as opções explicam E, mas, por parcimônia, preferimos $\{H_A\}$.

2.1.2 Pertinência e inclusão

Pertinência e inclusão são os conceitos mais fundamentais da teoria dos conjuntos.

Começando pela **noção de pertinência** denotado por $\in$: dizemos que um elemento x pertence a um conjunto X quando $x \in X$ e não pertence quando $x \notin X$.

Seja o nosso universo U de organismos: $P = \{\text{todas as plantas}\}$, $T = \{\text{todas as árvores}\}$ e $F = \{\text{todos os fungos}\}$. Se x é um carvalho, então $x \in T$ e, como toda árvore é uma planta, $x \in P$. Já se y é um cogumelo, então $y \in F$ e, na taxonomia moderna, $y \notin T$ e $y \notin P$. Agora, considere $A = \{\text{árvores com folhas verdes}\}$; a pertinência fica clara: $x \in A$ se, e somente se, x é árvore e tem folhas verdes.

Continuando, vem a **relação de inclusão** entre conjuntos denotada por $\subseteq$: escrevemos $X \subseteq Y$ quando todo elemento de X também pertence a Y (e $X \subset Y$ quando, além disso, $X \neq Y$). No nosso exemplo, $A \subseteq T \subset P$ e $T \cap F = \varnothing$ (árvores e fungos não se sobrepõem).

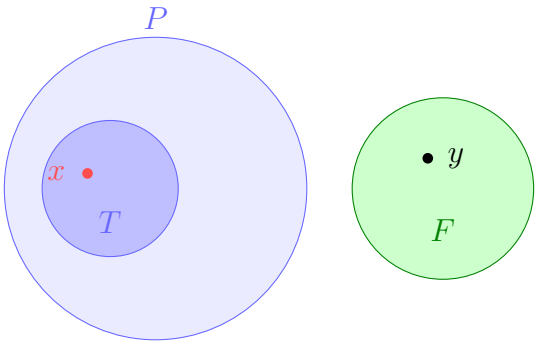


Figura 3: Pertinência: $x \in T \subseteq P$ (ponto vermelho dentro de T) e $y \in F$ (ponto preto); logo $y \notin P$ e $y \notin T$.

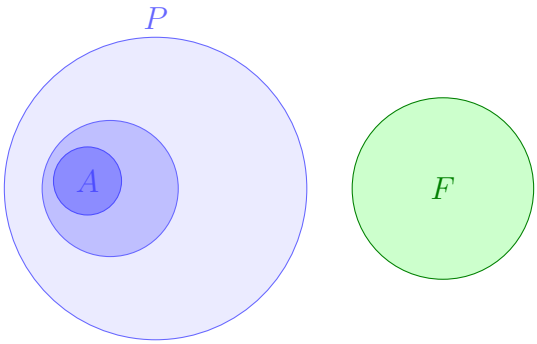


Figura 4: Inclusão: $A \subseteq T \subset P$ (círculos aninhados) e $T \cap F = \varnothing$ (círculos disjuntos).

2.1.3 Operações entre conjuntos

Com essas definições de pertinência, inclusão e subconjuntos, apresentamos as operações básicas entre conjuntos, que usaremos ao longo do texto (mantendo o exemplo com P, T, F, A).

Outras operações comuns entre conjuntos incluem:

- **União** ($A \cup B$): o conjunto de todos os elementos que pertencem a A, a B, ou ambos.

  Exemplo: $T \cup F$ é o conjunto de todos os organismos que são árvores ou fungos (ou ambos, se existissem tais organismos).

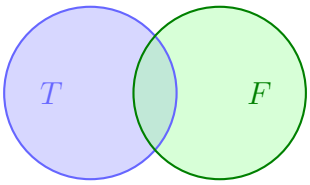


  Figura 5: União: a área colorida representa $T \cup F$; a sobreposição evidencia a intersecção $T \cap F$ (hipotética).

- **União disjunta** ($A \uplus B$): o conjunto de todos os elementos que pertencem a A ou a B, mas não a ambos; é igual a $A \cup B$ quando A e B são disjuntos.

Exemplo: $T \uplus F$ é o conjunto de todos os organismos que são árvores ou fungos, mas não ambos (o que é trivialmente igual a $T \cup F$ pois T e F são disjuntos).

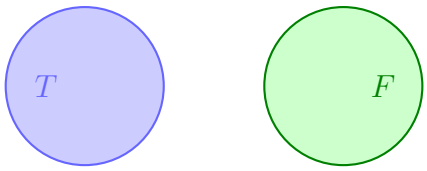


Figura 6: União disjunta: como $T \cap F = \varnothing$, tem-se $T \uplus F = T \cup F$.

- **Interseção** ($A \cap B$): o conjunto de todos os elementos que pertencem tanto a A quanto a B.

  Exemplo: $T \cap P = T$, pois todas as árvores são plantas.

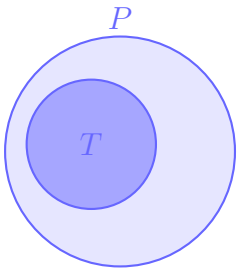


Figura 7: Interseção: como $T \subseteq P$, $T \cap P = T$ (a região escura é T).

- **Diferença** ($A \setminus B$): o conjunto de todos os elementos que pertencem a A mas não a B.

  Exemplo: $T \setminus A$ é o conjunto de todas as árvores que não têm folhas verdes.

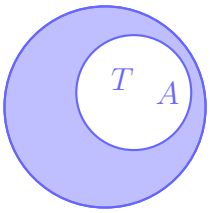


Figura 8: Diferença: região azul representa $T \setminus A$ (árvores que não têm folhas verdes).

- **Complemento** de X em um universo fixo U: $X^c := U \setminus X$ (também chamado de *complemento absoluto*); o *complemento relativo* de X em Y é $Y \setminus X$.

  Exemplo: $T^c = U \setminus T$ é o conjunto de todos os organismos que não são árvores. Ou seja, T^c inclui plantas que não são árvores, fungos e quaisquer outros organismos no universo U.

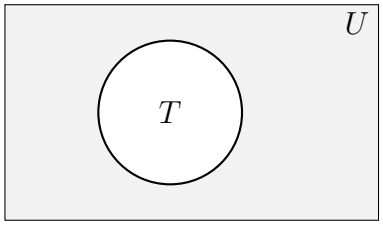


Figura 9: Complemento: a área cinza representa $T^c = U \setminus T$.

- **Diferença simétrica** $(A \,\Delta\, B)$: $(A \setminus B) \cup (B \setminus A)$; é igual a $A \cup B$ quando A e B são disjuntos.

  Exemplo: $P \,\Delta\, F$ é o conjunto de todos os organismos que são plantas ou fungos, mas não ambos (o que é trivialmente igual a $P \cup F$ pois P e F são disjuntos).

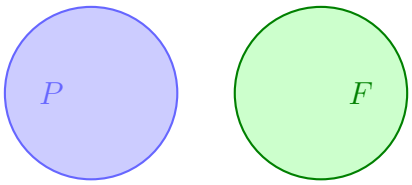


Figura 10: Diferença simétrica: como $P \cap F = \varnothing$, temos $P \,\Delta\, F = P \cup F$.

- **Produto cartesiano** $(A \times B)$: o conjunto de pares ordenados (a, b) com $a \in A$ e $b \in B$.

  Exemplo: $T = \{t_1, t_2\}$ e $F = \{f_1, f_2\}$. Então $T \times F = \{(t_1, f_1), (t_1, f_2), (t_2, f_1), (t_2, f_2)\}$.

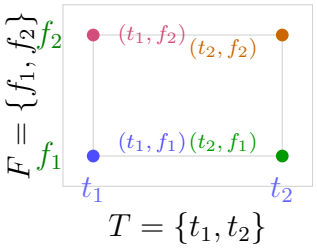


Figura 11: Produto cartesiano: pontos representam os pares de $T \times F$ para $T = \{t_1, t_2\}$ e $F = \{f_1, f_2\}$.

- **Conjunto das partes** (2^U): a família de todos os subconjuntos de U (inclui $\varnothing$ e o próprio U).

  Exemplo: se $U = \{x, y\}$, então $2^U = \{\varnothing, \{x\}, \{y\}, \{x, y\}\}$. Logo, $|2^U| = 4 = 2^{|U|}$.

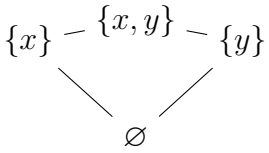


Figura 12: Conjunto das partes: diagrama de Hasse de 2^U para $U = \{x, y\}$.

Identidades úteis. Usaremos livremente as propriedades clássicas de conjuntos — comutatividade e associatividade de $\cup$ e $\cap$, distributividade e as **leis de De Morgan** — sem prova. Quando for relevante, explicitaremos a identidade no ponto de uso. Por exemplo, no nosso universo U, $(P \cup F)^c = P^c \cap F^c$.

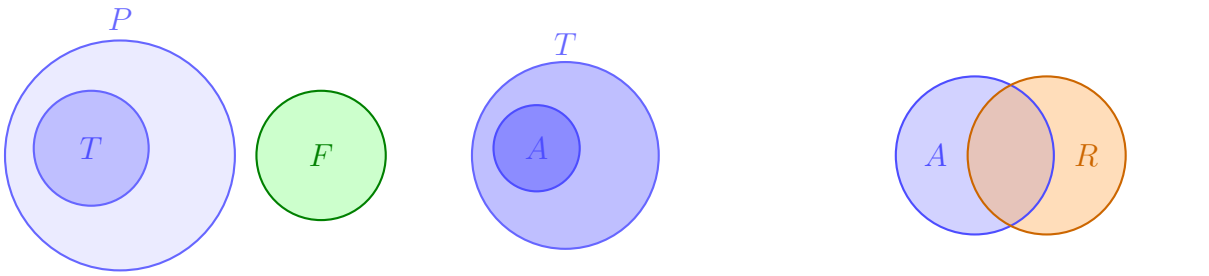
2.1.4 Coleção

Entre os objetos que podem pertencer a um conjunto, estão também eles mesmos, outros conjuntos. Chamaremos tais conjuntos de **coleções** (ou **famílias**) de conjuntos. Por exemplo, $\mathcal{C} = \{P, T, F\}$ é uma coleção formada pelos conjuntos de organismos já definidos: plantas P, árvores T e fungos F. Note que $\mathcal{C}$ é um conjunto como outro qualquer; seus elementos são, cada um, um conjunto.

Coleções são úteis para agrupar subconjuntos relacionados de um mesmo universo. Por exemplo, considere $\mathcal{D} = \{A, B\}$, onde $A = \{\text{árvores com folhas verdes}\}$ e $B = \{\text{árvores com folhas vermelhas}\}$. Assim, $\mathcal{D} \subseteq 2^T$ é uma coleção de subconjuntos de T.

Uma coleção $\mathcal{F}$ é dita **laminar** quando, para quaisquer $X, Y \in \mathcal{F}$, vale que $X \subseteq Y$, $Y \subseteq X$ ou $X \cap Y = \varnothing$; isto é, quaisquer dois conjuntos são aninhados (um está contido no outro) ou são disjuntos.

Por exemplo, na coleção $\mathcal{C} = \{P, T, F\}$: P é o conjunto de todas as plantas, T o de todas as árvores (portanto $T \subseteq P$) e F o de todos os fungos (disjunto de plantas e, logo, de árvores). Assim, quaisquer dois conjuntos em $\mathcal{C}$ são aninhados ou disjuntos, e $\mathcal{C}$ é laminar. Na coleção $\mathcal{D} = \{A, T\}$: A é o conjunto de árvores com folhas verdes e T o de todas as árvores; como toda árvore de A é árvore de T, temos $A \subseteq T$ e a coleção é laminar. Já em $\mathcal{E} = \{A, R\}$: R é o conjunto de árvores frutíferas; há árvores que são ao mesmo tempo frutíferas e de folhas verdes (a interseção é não vazia), mas nenhuma das classes contém a outra, então $\mathcal{E}$ não é laminar.



(a) $\mathcal{C} = \{P, T, F\}$ — laminar (b) $\mathcal{D} = \{A, T\}$ — laminar (c) $\mathcal{E} = \{A, R\}$ — não laminar

Figura 13: Laminaridade em coleções: em (a) e (b), quaisquer dois conjuntos são aninhados ou disjuntos; em (c), A e R se interceptam sem inclusão, violando a laminaridade.

Este é um importante conceito que aparecerá no restante do trabalho. A ideia de laminaridade retornará quando tratarmos de cortes dirigidos.

2.1.5 Comparando conjuntos: cardinalidade e maximalidade

Podemos comparar conjuntos através de relações de tamanho (cardinalidade) ou por relações de inclusão. Essas duas formas de comparação são distintas e importantes, especialmente quando lidamos com coleções de conjuntos.

A **cardinalidade** de um conjunto A, denotada por $|A|$, é o número de elementos de A. Para conjuntos finitos, é simplesmente a contagem dos elementos (por exemplo, se $A = \{1, 2, 3\}$, então $|A| = 3$). Para conjuntos infinitos, a cardinalidade pode ser mais complexa, envolvendo conceitos como infinito enumerável e não enumerável. Por exemplo, o conjunto dos números naturais $\mathbb{N}$ é infinito enumerável, enquanto o conjunto dos números reais $\mathbb{R}$ é infinito não enumerável.

Dizemos que $A \in \mathcal{C}$ tem **maior cardinalidade** se $|A| \geq |B|$ para todo $B \in \mathcal{C}$ (podendo haver empates). Esse critério não coincide, em geral, com a comparação por relação de inclusão. Em grafos, por exemplo, distinguem-se conjuntos independentes *maximais* (não ampliáveis) de conjuntos independentes *máximos* (de cardinalidade máxima).

Ao compararmos uma coleção $\mathcal{C}$ de conjuntos utilizando sua relações de inclusão $(\mathcal{C}, \subseteq)$, é imprescindível distinguir **maximal** de **máximo**.

Um conjunto $A \in \mathcal{C}$ é **maximal** se não existe $B \in \mathcal{C}$ tal que $A \subset B$. Em palavras: não dá para ampliar A estritamente dentro da coleção. Podem haver vários elementos maximais, e eles podem ser incomparáveis entre si. Ex.: em $\mathcal{C} = \big\{\{1\}, \{2\}\big\}$, ambos $\{1\}$ e $\{2\}$ são maximais, mas não existe máximo.

Um conjunto $A \in \mathcal{C}$ é **máximo** se $B \subseteq A$ para todo $B \in \mathcal{C}$. Se existe, é único. Ex.: em $\mathcal{C} = \big\{\{1\}, \{2\}, \{1, 2\}\big\}$, o conjunto $\{1, 2\}$ é o máximo.

Um bom exemplo para ilustrar a distinção entre conjuntos maximais e máximos é a coleção $\mathcal{C} = \big\{\{1\}, \{2\}, \{1,2\}, \{3\}\big\}$. Aqui, $\{1,2\}$ é o único conjunto máximo (contém todos os outros), enquanto $\{1\}$, $\{2\}$ e $\{3\}$ são todos maximais (não podem ser ampliados dentro da coleção).

Esses conceitos reaparecerão ao longo do texto, especialmente na diferença entre estruturas **maximais** (saturadas por inclusão) e **máximas/ótimas** (de maior cardinalidade ou menor custo). Para fixar ideias:

- Em muitos problemas, "**maximal**" quer dizer: não dá para ampliar uma escolha sem violar as regras; já "**máximo/ótimo**" quer dizer: entre todas as escolhas válidas, essa é a melhor segundo o critério (por exemplo, menor custo).
- No algoritmo de **Chu–Liu/Edmonds**, começamos com escolhas locais que já não podem ser ampliadas dentro das regras do problema e, a partir delas, chegamos a uma solução de menor custo.
- No método de **András Frank**, primeiro construímos uma estrutura organizada que garante escolhas suficientes; depois, usando apenas relações já ativadas por essa estrutura, extraímos a solução ótima.
- Moral: partimos da ideia de "não dá para aumentar" (maximal) e chegamos a "melhor possível" (máximo/ótimo). Os detalhes técnicos de cada método aparecerão nas seções próprias.

2.2 Relações e Funções

Desde a introdução, vimos a ideia filosófica de explicar como "ligar" fatos a hipóteses da forma mais parcimoniosa possível. Para tornar essa intuição precisa, precisamos de uma linguagem que descreva objetos (conjuntos) e como eles se conectam. É aqui que entram as **relações** e, de modo ainda mais disciplinado, as **funções**: regras que associam a cada elemento de um conjunto exatamente um elemento de outro. Com elas, passamos do discurso qualitativo sobre explicações para uma estrutura matemática que permite medir, comparar e, adiante, otimizar.

Na matemática, uma **relação** entre dois conjuntos A e B é uma maneira de associar elementos de A com elementos de B. Uma **função** é um tipo especial de relação que associa cada elemento de A a exatamente um elemento de B.

Uma **relação** R entre dois conjuntos A e B é um subconjunto do produto cartesiano $A \times B$. Ou seja, $R \subseteq A \times B$. Se $(a, b) \in R$, dizemos que a está relacionado a b pela relação R, denotado aRb.

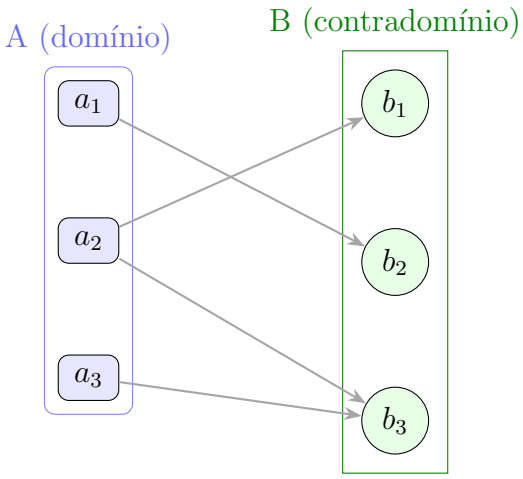


Figura 14: Relação $R \subseteq A \times B$. Cada seta representa um par $(a, b) \in R$ (isto é, $a\,R\,b$). As formas/cores distinguem domínio (A, retângulos azuis) de contradomínio (B, círculos verdes). Note que a_2 se relaciona com b_1 e b_3; logo, este R *não* é uma função. Para que R fosse uma função $f : A \to B$, cada $a \in A$ deveria ter *exatamente uma* seta saindo para algum $b \in B$.

No nosso primeiro exemplo-mestre, considere $P = \{\text{todas as plantas}\}$ e $F = \{\text{todos os fungos}\}$. Definimos a relação R como "é um organismo que compete com". Assim, se uma planta $p \in P$ compete com um fungo $f \in F$, então $(p, f) \in R$.

No nosso segundo exemplo, considere $H = \{\text{hipóteses}\}$ e $E = \{\text{evidências}\}$. Definimos a relação R como "explica". Se uma hipótese $h \in H$ explica uma evidência $e \in E$, então $(h, e) \in R$.

Em teoria dos grafos, uma relação pode representar conexões entre vértices. Por exemplo, em um grafo dirigido, a relação "existe uma aresta de u para v"pode ser representada como um conjunto de pares ordenados (u, v).

Uma **função** f de um conjunto A em um conjunto B é uma relação especial que associa cada elemento de A a exatamente um elemento de B. Denotamos isso como $f : A \to B$. Se $f(a) = b$, dizemos que b é a imagem de a sob f.

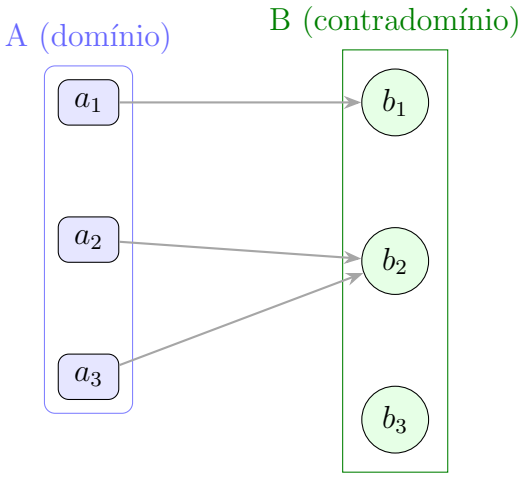


Figura 15: Função $f : A \to B$. Cada elemento de A tem *exatamente uma* seta saindo para um elemento de B (a imagem). Elementos distintos de A podem ter a mesma imagem (muitos-para-um), e nem todo elemento de B precisa ser imagem (aqui, b_3 não é atingido). Compare com a Fig. 14.

No nosso exemplo-mestre, considere $P = \{\text{todas as plantas}\}$ e $\mathbb{N} = \{0, 1, 2, \ldots\}$ (números naturais). Definimos a função $f : P \to \mathbb{N}$ que associa cada planta ao seu número de folhas. Se $p \in P$ é uma árvore com 100 folhas, então $f(p) = 100$.

Em teoria dos grafos, funções podem ser usadas para atribuir pesos às arestas. Por exemplo, se temos um grafo G com arestas $e_1, e_2, \ldots, e_n$, podemos definir uma função $c : E \to \mathbb{R}^+$ que atribui um peso $c(e_i)$ a cada aresta e_i.

Na ciência da computação, relações e funções são usadas para modelar conexões entre dados, estruturas de dados e operações. Por exemplo, em bancos de dados relacionais, tabelas representam relações entre diferentes entidades. Em programação funcional, funções são tratadas como cidadãos de primeira classe, permitindo a criação de funções de ordem superior que podem receber outras funções como argumentos ou retorná-las como resultados.

2.2.1 Conceitos em Funções

Alguns conceitos importantes relacionados a funções incluem:

- **Domínio**: o conjunto A de entrada da função $f : A \to B$.
- **Contradomínio**: o conjunto B de possíveis saídas da função.
- **Imagem**: o conjunto de valores efetivamente atingidos pela função, $f(A) = \{f(a) \mid a \in A\}$.

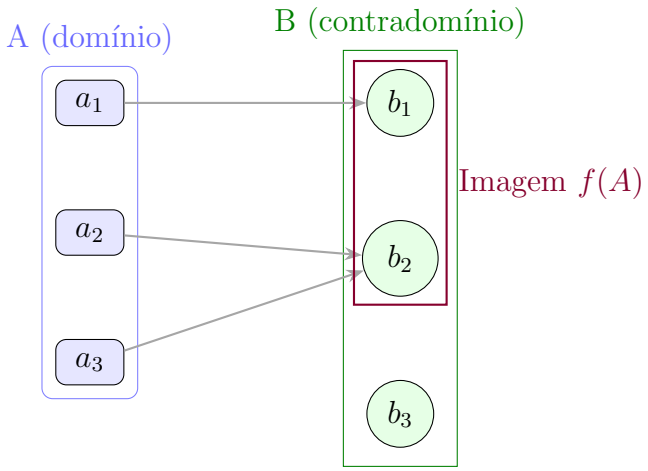


Figura 16: Domínio, contradomínio e imagem: A (retângulos azuis) mapeia via f para B (círculos verdes). A imagem $f(A)$ é o subconjunto de B efetivamente atingido (aqui, $\{b_1, b_2\}$).

- **Injetora**: uma função f é injetora se $f(a_1) = f(a_2)$ implica $a_1 = a_2$; ou seja, elementos distintos do domínio têm imagens distintas.
- **Sobrejetora**: uma função f é sobrejetora se para todo $b \in B$, existe $a \in A$ tal que $f(a) = b$; ou seja, a imagem é igual ao contradomínio.
- **Bijetora**: uma função que é tanto injetora quanto sobrejetora; estabelece uma correspondência um-para-um entre os elementos de A e B.

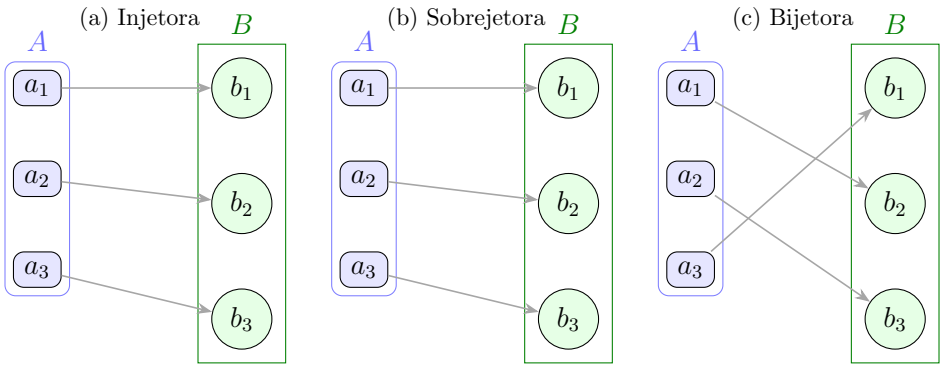


Figura 17: Funções especiais: (a) Injetora — elementos distintos em A têm imagens distintas em B; (b) Sobrejetora — todo elemento de B é imagem; (c) Bijetora — um-para-um e sobre B.

2.2.2 Funções de agregação e somatórios

Além de relacionar elementos de conjuntos, muitas operações familiares em matemática, envolvem *funções* que recebem coleções de números (ou funções) e devolvem um número.

Uma **função de agregação** é uma função que recebe um conjunto (ou sequência) de valores e retorna um único valor que representa algum aspecto agregado desses valores. Exemplos comuns incluem:

- **Média**: A média aritmética de um conjunto de números $x_1, x_2, \ldots, x_n$ é dada por $\frac{1}{n}\sum_{i=1}^{n} x_i$.
- **Máximo e Mínimo**: A função máximo retorna o maior valor em um conjunto, enquanto a função mínimo retorna o menor valor.
- **Produto**: O produto de um conjunto de números $x_1, x_2, \ldots, x_n$ é dado por $\prod_{i=1}^{n} x_i$.
- **Contagem**: A função contagem retorna o número de elementos em um conjunto.

O **somatório**, por exemplo, é uma função de agregação linear que mapeia uma sequência $(x_1, \ldots, x_n)$ em sua soma:

$$\sum_{i=1}^{n} x_i.$$

Esse conceito é especialmente útil em otimização e em análise combinatória: somatórios aparecem o tempo todo e serão explorados ao longo deste trabalho.

Exemplos com grafos. As sessões seguintes explorarão em maiores detalhes grafos e dígrafos, mas agora, consideremos a ideia básica: um **grafo** é um conjunto de pontos (*vértices*) ligados por linhas (*arestas*). No caso *não dirigido*, as linhas não têm seta; no caso *dirigido*, cada linha tem um sentido e é chamada de *arco*.

A noção de somatória aparecerá naturalmente quando lidamos com propriedades dos grafos. Por exemplo:

Seja um grafo não dirigido $G = (V, E)$. O **grau** de um vértice $v \in V$, escrito $\deg(v)$, é quantas arestas tocam em v. A soma dos graus de todos os vértices conta cada aresta *duas vezes* (uma por extremidade), portanto:

$$\sum_{v \in V} \deg(v) = 2\,|E|.$$

Em grafos dirigidos, distinguimos $\deg^-(v)$ (quantos arcos *chegam* em v) e $\deg^+(v)$ (quantos arcos *saem* de v). Cada arco contribui com 1 para um grau de saída e 1 para um grau de entrada, logo:

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

Agora suponha que cada aresta/arco $e \in E$ tenha um *peso* (ou *custo*) $c(e) \geq 0$. O **custo total** de um subconjunto $F \subseteq E$ é simplesmente a soma dos pesos das arestas escolhidas:

$$C(F) = \sum_{e \in F} c(e).$$

De maneira análoga, se $X \subseteq V$ é um conjunto de vértices, o **valor total** (ou peso total) dos arcos que *saem* de X é a soma dos pesos dessas setas. Usaremos mais adiante a notação $\delta^+(X)$ para o conjunto de arcos que saem de X (ver a seção de dígrafos); com essa notação,

$$\mathrm{val}^+(X) = \sum_{e \in \delta^+(X)} c(e).$$

Esses exemplos mostram como somatórios capturam propriedades estruturais do grafo por meio de funções simples de agregação.

2.2.3 Funções Especiais

Função de custo

Uma **função de custo** é uma função $c : A \to \mathbb{R}^+$ que atribui um valor numérico não negativo (custo) a cada elemento de um conjunto A. Essas funções são amplamente utilizadas em otimização, economia e teoria dos grafos para modelar despesas, penalidades ou recursos associados a escolhas ou ações.

Exemplo: Considere um conjunto de tarefas $T = \{t_1, t_2, t_3\}$. Uma função de custo $c : T \to \mathbb{R}^+$ pode ser definida como:

$$c(t_1) = 5, \quad c(t_2) = 10, \quad c(t_3) = 3.$$

Aqui, $c(t_i)$ representa o custo de realizar a tarefa t_i.

Depende diretamente do conceito de somatório, pois frequentemente queremos minimizar o custo total de um conjunto de escolhas. Se $S \subseteq A$ é um subconjunto de elementos escolhidos, o custo total associado a S é dado por:

$$C(S) = \sum_{a \in S} c(a).$$

Função c-disjunta Uma **função c-disjunta** é uma função $f : A \to B$ que, para quaisquer $a_1, a_2 \in A$ com $a_1 \neq a_2$, as imagens $f(a_1)$ e $f(a_2)$ são disjuntas, ou seja, $f(a_1) \cap f(a_2) = \varnothing$. Em outras palavras, elementos distintos do domínio são mapeados para conjuntos disjuntos no contradomínio.

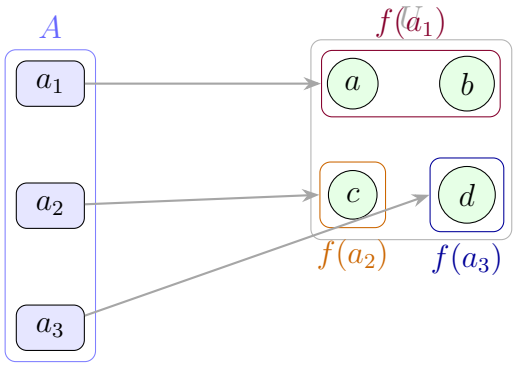


Figura 18: Função c-disjunta: cada $a \in A$ mapeia para um *subconjunto* de U, e as imagens são dois a dois disjuntas (sem sobreposição). No exemplo, $f(a_1) = \{a, b\}$, $f(a_2) = \{c\}$ e $f(a_3) = \{d\}$.

Exemplo: Considere $A = \{1, 2, 3\}$ e $B = \{\{a\}, \{b\}, \{c\}, \{d\}\}$. Definimos a função $f : A \to B$ como:

$$f(1) = \{a, b\}, \quad f(2) = \{c\}, \quad f(3) = \{d\}.$$

Aqui, f é c-disjunta, pois $f(1) \cap f(2) = \varnothing$, $f(1) \cap f(3) = \varnothing$ e $f(2) \cap f(3) = \varnothing$.

Função c-viável

Uma **função c-viável** é uma função $g : A \to \mathbb{R}^+$ que satisfaz certas condições de viabilidade relacionadas a um conjunto de restrições ou critérios. Essas funções são frequentemente usadas em otimização e teoria dos grafos para garantir que as soluções propostas atendam a requisitos específicos.

Exemplo: Considere um conjunto de projetos $P = \{p_1, p_2, p_3\}$ e uma função $g : P \to \mathbb{R}^+$ que atribui um valor de viabilidade a cada projeto. Suponha que temos a restrição de que a soma dos valores de viabilidade deve ser menor ou igual a um certo limite L. Se definirmos:

$$g(p_1) = 4, \quad g(p_2) = 6, \quad g(p_3) = 3,$$

então a função g é c-viável se $g(p_1) + g(p_2) + g(p_3) \leq L$.

Essas funções são essenciais para garantir que as soluções propostas em problemas de otimização sejam práticas e atendam aos critérios estabelecidos.

Funções de otimização

Do ponto de vista semiótico, "melhor" exprime uma preferência entre interpretações: ao comparar alternativas, escolhemos aquela cuja significação é mais adequada a um critério. Para tornar isso operacional, a matemática troca "fazer mais sentido" por "ter maior (ou menor) valor" em uma escala formal: fixamos (i) um conjunto de soluções viáveis $\mathcal{F}$ e (ii) uma função numérica sobre $\mathcal{F}$ que induz uma ordem de comparação.

Formalmente, usamos uma **função objetivo** (ou **função de otimização**)

$$h : \mathcal{F} \to \mathbb{R},$$

que atribui um número real a cada solução. Buscamos uma solução $S^* \in \mathcal{F}$ que *minimize* ou *maximize* h (isto é, um argmin ou argmax). Quando esse número resulta da soma de contribuições elementares, obtemos o caso aditivo, em ligação direta com os somatórios apresentados antes.

Caso *aditivo*. Quando cada elemento $a \in A$ tem um custo $c(a) \geq 0$ e as soluções são subconjuntos $S \subseteq A$, a função objetivo mais comum é o custo total

$$C(S) = \sum_{a \in S} c(a),$$

que desejamos *minimizar*. De modo análogo, se cada item tem um benefício $p(a) \geq 0$, podemos *maximizar* o benefício total $P(S) = \sum_{a \in S} p(a)$, possivelmente sujeito a restrições (por exemplo, de orçamento ou limite).

Outro exemplo: considere produtos $X = \{x_1, x_2, x_3\}$ com lucro $p(x_1) = 10$, $p(x_2) = 15$, $p(x_3) = 7$. Se houver um limite de custo que impede escolher todos, o objetivo típico é escolher um subconjunto $S \subseteq X$ que maximize $\sum_{x \in S} p(x)$ respeitando as restrições. Essa forma reflete exatamente os somatórios introduzidos antes.

Em todos os casos, a função objetivo explicita o critério de "melhor", e as restrições determinam quais soluções são aceitáveis.

2.2.4 Otimização

O princípio da navalha de occam nos diz que a explicação mais simples tende a ser a correta. Do ponto de vista semiótico, isso é escolher, entre interpretações possíveis, a que melhor satisfaz um critério. A matemática também se preocupa com identificar a "melhor" solução entre várias alternativas, mas traduz essa ideia em termos quantitativos: fixamos (i) um conjunto de soluções viáveis $\mathcal{F}$ e (ii) uma função numérica sobre $\mathcal{F}$ que induz uma ordem de comparação.

Assim, a otimização envolve a maximização ou minimização de uma função objetivo $h : \mathcal{F} \to \mathbb{R}$ sobre um conjunto de soluções viáveis $\mathcal{F}$.

Esse conceito pode aparecer em muitas necessidades do dia-a-dia: uma empresa pode querer minimizar custos de produção, um viajante pode buscar o caminho mais curto entre dois pontos, ou um investidor pode tentar maximizar o retorno de um portfólio. Em cada caso, a função objetivo quantifica o que significa ser "melhor" ou "mais eficiente".

Pensando em modelagem de problemas em grafos, podemos pensar em exemplos clássicos de otimização:

- **Caminho mais curto**: Dado um grafo com pesos nas arestas, encontrar o caminho entre dois vértices que minimize a soma dos pesos das arestas percorridas.
- **Árvore geradora mínima**: Encontrar uma árvore que conecte todos os vértices de um grafo com o menor custo total das arestas.
- **Fluxo máximo**: Em um grafo direcionado com limites nas arestas, encontrar o fluxo máximo que pode ser enviado de uma fonte a um sumidouro sem exceder esses limites.

A dualidade

Um tema recorrente em otimização é a *dualidade*, que conecta problemas de minimização a problemas de maximização.

Falando de forma intuitiva, a dualidade nos remete à contrastes, yin-yang, noite e dia, matéria e anti-matéria, máximos e mínimos. Em termos matemáticos, para cada problema de otimização (o *primal*), existe um problema associado (o *dual*) que oferece uma perspectiva complementar. Resolver um desses problemas pode fornecer insights ou soluções para o outro.

Considere como problema primal a minimização do custo de uma estrutura (como uma árvore geradora mínima) e como dual a maximização de um conjunto de pesos ou preços que justificam esse custo mínimo. A relação entre primal e dual é formalizada por teoremas de dualidade, que garantem que o valor ótimo do primal é igual ao valor ótimo do dual sob certas condições.

Esse ponto de vista leva a "teoremas min–max" que ligam problemas de *minimização* a problemas de *maximização* e fornecem certificados verificáveis de otimalidade.

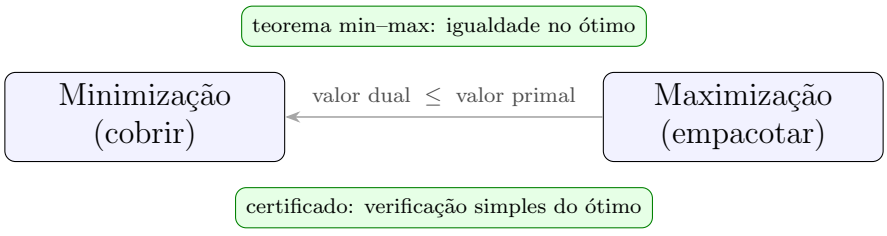


Figura 19: Intuição de min–max: um problema de cobrir (minimização) e um de empacotar (maximização) andam juntos. Sempre vale valor dual $\leq$ valor primal; quando há igualdade, temos um certificado de otimalidade.

No contexto de grafos, a otimização costuma aparecer como a busca por subestruturas (caminhos, árvores, cortes, fluxos) que minimizam ou maximizam um custo, sempre respeitando a topologia do grafo.

Nesta dissertação, essa noção de otimização é central: olhamos para o mesmo problema por dois ângulos que se completam. No lado "primal", queremos montar diretamente a arborescência de menor custo. O algoritmo de Chu–Liu/Edmonds faz isso de forma gulosa: ajusta os custos por vértice, cria arestas de custo zero (0-arestas), contrai ciclos quando aparecem e segue até montar a solução ótima. ([**chu1965**, **edmonds1967optimum**]).

No lado "dual", em vez de montar a árvore, colocamos custos em cortes do grafo com raiz r. A regra é simples: nenhum custo pode ultrapassar o custo das arestas que cruzam o corte. Buscamos escolher esses custos para somar o máximo possível. As arestas que "batem no limite" viram 0-arestas, e a partir delas conseguimos reconstruir uma arborescência ótima. Essa visão, desenvolvida por Frank, leva a um teorema min–max e a um procedimento em duas etapas: primeiro ajustamos os custos, depois extraímos a solução usando apenas 0-arestas. (cf. [**frank2014**, **schrijver2003comb**])

2.3 Problemas interessantes

Qual o número mínimo de cores necessárias para colorir um mapa de países, de modo que países vizinhos tenham cores diferentes? Qual o caminho mais curto entre duas cidades em um mapa rodoviário? Como encontrar a árvore geradora mínima que conecta todas as cidades com o menor custo total? Essas perguntas são exemplos clássicos de problemas que podem ser modelados e resolvidos usando a teoria dos grafos. Vistas sob a lente da navalha de occam, todas elas buscam a solução mais parcimoniosa que atende ao requisito: usar poucas cores, percorrer um caminho curto ou conectar tudo com custo mínimo.

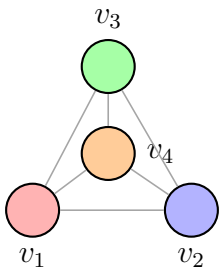


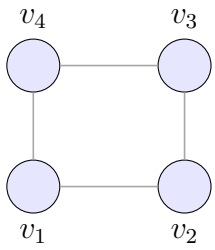
Figura 20: Coloração de grafos: exemplo de coloração própria do grafo completo K_4. Como K_4 é completo, precisamos de 4 cores para colorir seus vértices de modo que vértices adjacentes tenham cores diferentes. Uma coloração é uma função $\varphi : V \to C$ tal que, se $uv \in E$, então $\varphi(u) \neq \varphi(v)$.

Sem a teoria dos grafos, seria difícil formalizar e resolver esses problemas de maneira eficiente. Ao representar situações do mundo real como grafos, tornamos a parcimônia da navalha de occam algo operacional: escolhemos uma medida simples (número de cores, comprimento, custo) e aplicamos algoritmos que, entre as soluções viáveis, minimizam ou maximizam esse critério — produzindo soluções ótimas ou, quando necessário, boas aproximações.

2.4 Grafos

Falamos bastante de grafos ao longo do texto, aqui fixamos a noção básica.

Um **grafo** $G = (V, E)$ é uma estrutura matemática composta por um conjunto V de *vértices* (ou *nós*) e um conjunto E de *arestas* (ou *ligações*) que conectam pares de vértices.



$$V = \{v_1, v_2, v_3, v_4\}$$
$$E = \{\{v_1, v_2\}, \{v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_1\}\}$$

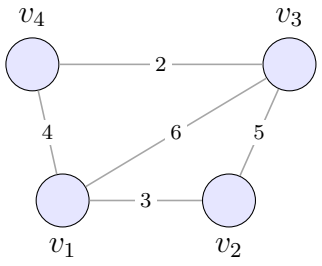
Figura 21: Definição de grafo: exemplo de grafo simples $G = (V, E)$. Pontos representam os vértices V e linhas representam as arestas E, que são pares não ordenados de vértices distintos.

O conjunto de vértices V pode ser definido como $V = \{v_1, v_2, \ldots, v_n\}$, onde cada v_i representa um ponto distinto no grafo. O conjunto de arestas E é um conjunto de pares não ordenados de vértices, ou seja, $E \subseteq \{\{u, v\} \mid u, v \in V, u \neq v\}$. Cada aresta $\{u, v\}$ indica uma conexão entre os vértices u e v.

Esses vértices e arestas podem representar uma variedade de entidades e relações no mundo real. Por exemplo, em um grafo que modela uma rede social, os vértices podem representar pessoas, e as arestas podem representar amizades entre elas. Em um grafo que representa uma rede de transporte, os vértices podem ser cidades, e as arestas podem ser estradas ou rotas de voo conectando essas cidades.

Tendo em mente esses problemas, podemos falar de custos associados às arestas. Por exemplo, em um grafo que representa uma rede de transporte, cada aresta pode ter um custo associado, como a distância entre duas cidades ou o tempo necessário para percorrer uma estrada. Em um grafo que modela uma rede de comunicação, as arestas podem ter custos relacionados à largura de banda ou à latência.

Esses custos representam uma função $c : E \to \mathbb{R}^+$ que atribui um valor numérico não negativo a cada aresta do grafo. Assim, para cada aresta $\{u, v\} \in E$, $c(\{u, v\})$ representa o custo associado a essa conexão.



$$c : E \to \mathbb{R}^+$$
$$c(\{v_1, v_2\}) = 3,\ c(\{v_1, v_3\}) = 6, \ldots$$

Figura 22: Grafo com custos nas arestas: a função $c : E \to \mathbb{R}^+$ atribui a cada aresta um custo $c(e)$ (mostrado junto às arestas). Por exemplo, $c(\{v_1, v_2\}) = 3$. O custo total de um subconjunto $F \subseteq E$ é $C(F) = \sum_{e \in F} c(e)$.

Grafos apresentam diversas estruturas especiais. Essas estruturas são definidas como subgrafos e são fundamentais para entender a topologia e as propriedades dos grafos, e muitas vezes são o foco de problemas de otimização.

2.4.1 Subgrafos

Um **subgrafo** $H = (V_H, E_H)$ de um grafo $G = (V, E)$ é um grafo cujos vértices e arestas são subconjuntos dos vértices e arestas de G. Formalmente, $V_H \subseteq V$ e $E_H \subseteq E$, e cada aresta em E_H conecta dois vértices em V_H.

Alguns subgrafos interessantes incluem: caminhos, ciclos, componentes conexas e árvores. Muitas propriedades e algoritmos em grafos dependem dessas estruturas, como encontrar o caminho mais curto entre dois vértices, detectar ciclos ou construir árvores geradoras mínimas. Dentre essas muitas estruturas especiais, vamos apresentar as que são relevantes para o desenvolvimento desta dissertação:

Caminhos

Um **caminho** em um grafo é uma sequência de vértices conectados por arestas. Formalmente, um caminho P de comprimento $k \geq 1$ é uma sequência de vértices $P = (v_1, v_2, \ldots, v_{k+1})$ tal que cada par consecutivo (v_i, v_{i+1}) é uma aresta em E. O comprimento do caminho é o número de arestas que ele contém, que é k.

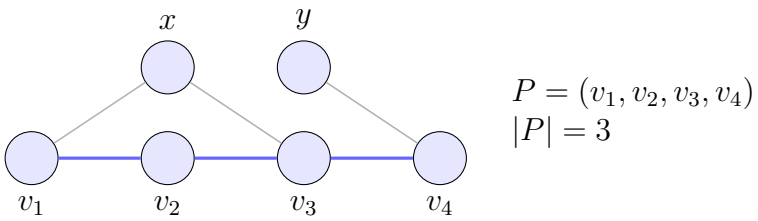


Figura 23: Caminho em grafo não dirigido: o caminho $P = (v_1, v_2, v_3, v_4)$ está destacado em azul. Seu comprimento é o número de arestas percorridas, $|P| = 3$.

Ciclos

Um **ciclo** é um caminho que começa e termina no mesmo vértice, ou seja, $v_1 = v_{k+1}$. Formalmente, um ciclo C é uma sequência de vértices $C = (v_1, v_2, \ldots, v_k, v_1)$ tal que cada par consecutivo (v_i, v_{i+1}) é uma aresta em E e $k \geq 2$. O comprimento do ciclo é o número de arestas que ele contém, que é k.

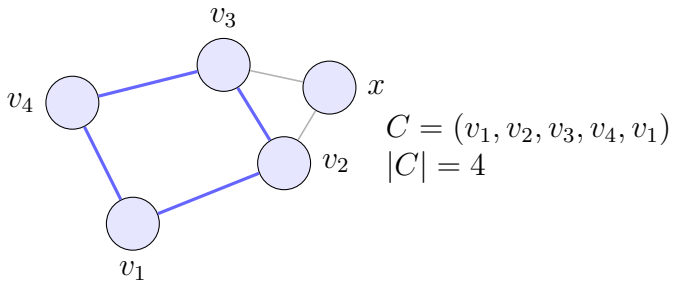


Figura 24: Ciclo em grafo não dirigido: o ciclo $C = (v_1, v_2, v_3, v_4, v_1)$ está destacado em azul. Seu comprimento é o número de arestas, $|C| = 4$.

Componentes conexas

Uma **componente conexa** de um grafo é um subgrafo maximal que é conexo. Formalmente, uma componente conexa C é um subgrafo $C = (V_C, E_C)$ onde $V_C \subseteq V$ e $E_C \subseteq E$, que satisfaz a seguinte propriedade:

- C é conexo: existe um caminho entre qualquer par de vértices em V_C.

Além disso, C é maximal, o que significa que não é possível adicionar mais vértices ou arestas a C sem perder a propriedade de conexidade.

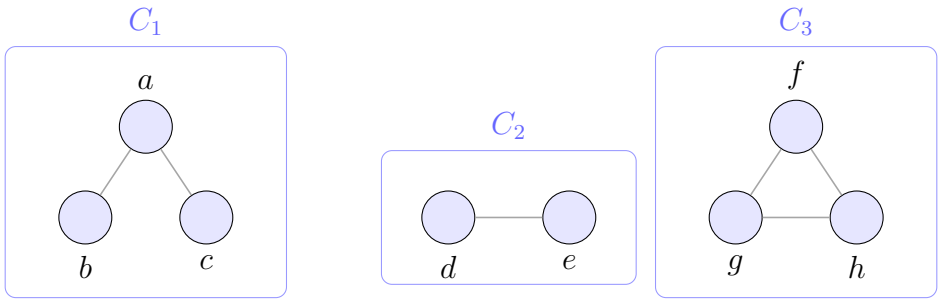


Figura 25: Componentes conexas: o grafo possui três componentes C_1, C_2 e C_3. Cada C_i é conexo e *maximal*: adicionar qualquer vértice de fora quebraria a propriedade de conexidade interna ou exigiria arestas ausentes.

Árvores

Uma **árvore** é um grafo conexo e acíclico. Formalmente, uma árvore T é um grafo $T = (V_T, E_T)$ onde $V_T \subseteq V$ e $E_T \subseteq E$, que satisfaz as seguintes propriedades:

- T é conexo: existe um caminho entre qualquer par de vértices em V_T.
- T é acíclico: não contém ciclos.

Além disso, uma árvore com n vértices sempre tem exatamente $n - 1$ arestas.

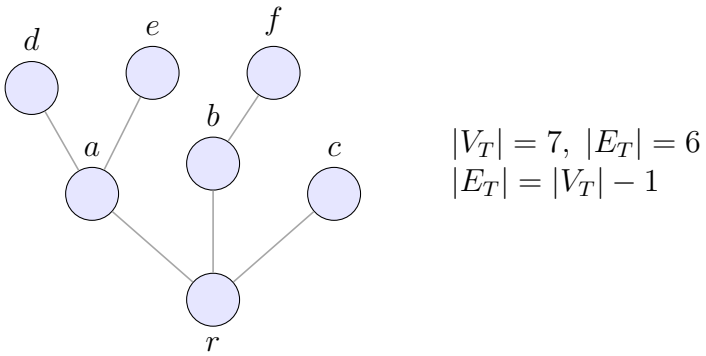


Figura 26: Árvore: grafo conexo e acíclico. No exemplo, $|V_T| = 7$ e $|E_T| = 6$, satisfazendo $|E_T| = |V_T| - 1$. Não há ciclos e existe um único caminho simples entre quaisquer dois vértices.

Para o nosso objetivo principal, nos interessa entendê-las em grafos que a direção das conexões (arestas) importa.

2.5 Dígrafos: quando a direção importa

Existem problemas que a direção das arestas faz toda a diferença. Por exemplo, em uma rede de tráfego, algumas ruas são de mão única, ou em uma rede de comunicação, os dados podem ser enviados em uma direção específica. Nesses casos, usamos *grafos dirigidos* (ou grafos direcionados ou simplesmente dígrafos), onde as arestas são pares de vértices ordenados.

Um **grafo dirigido - dígrafo** (grafos direcionados) é uma estrutura matemática composta por um conjunto V de *vértices* e um conjunto A de *arcos* (ou *arestas direcionadas*) que conectam pares ordenados de vértices.

Por vértices entendemos o mesmo conjunto que em grafos comuns, mas agora as arestas entre eles têm uma direção específica. Cada arco $(u, v) \in A$ indica uma conexão direcionada do vértice u para o vértice v, significando que a relação ou fluxo ocorre de u para v.

Assim, temos os conceitos de cabeça e cauda de um arco: em (u, v), u é a *cauda* (origem) e v é a *cabeça* (destino). Esses conceitos podem ser formalizados por meio de funções $s, t : A \to V$, onde $s((u, v)) = u$ (cauda) e $t((u, v)) = v$ (cabeça).

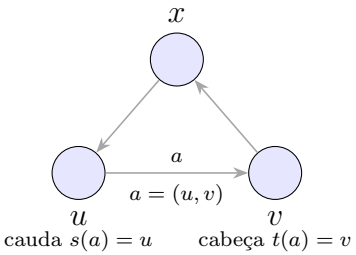


Figura 27: Dígrafos: arcos têm direção. No arco $a = (u, v)$, u é a *cauda* (origem) e v é a *cabeça* (destino). As funções $s, t : A \to V$ dão a origem e o destino de cada arco.

Em digrafos podem ocorrer laços (arcos que conectam um vértice a ele mesmo, como (u, u)) nesse caso as funções s e t coincidem. Também podem ocorrer múltiplos arcos entre o mesmo par de vértices (como (u, v) e (u, v) distintos). Pictograficamente representamos essas condições com setas com dupla ponta ou com rótulos diferentes.

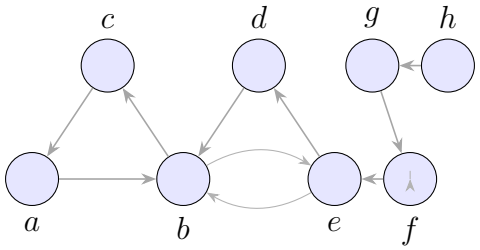
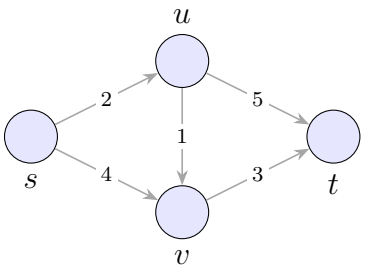


Figura 28: Dígrafo: exemplo de grafo dirigido $D = (V, A)$. Pontos representam os vértices V e setas representam os arcos A, que são pares ordenados de vértices. Laços (como (f, f)) e múltiplos arcos (como (b, e) e (e, b)) são permitidos.

Tal qual os grafos comuns, os dígrafos podem ter custos associados aos arcos. A função de custo $c : A \to \mathbb{R}^+$ atribui um valor numérico geralmente não negativo a cada arco do dígrafo. Assim, para cada arco $(u, v) \in A$, $c((u, v))$ representa o custo associado a essa conexão direcionada.



$$c : A \to \mathbb{R}^+$$
$$c((s, u)) = 2,\ c((v, t)) = 3, \ldots$$
$$C(P) = \sum_{a \in P} c(a)$$

Figura 29: Dígrafos com custos nos arcos: a função $c : A \to \mathbb{R}^+$ atribui a cada arco (u, v) um custo $c((u, v))$ (mostrado junto à seta). O custo de um caminho P é a soma dos custos de seus arcos, $C(P) = \sum_{a \in P} c(a)$.

Um conceito importante em dígrafos é o de grau de um vértice. O **grau de entrada** (ou *in-degree*) de um vértice v, denotado por $d^-(v)$, é o número de arcos que chegam a v (ou seja, o número de arcos cujo destino é v). O **grau de saída** (ou *out-degree*) de um vértice v, denotado por $d^+(v)$, é o número de arcos que saem de v (ou seja, o número de arcos cuja origem é v). Formalmente, temos:

$$d^-(v) = |\{(u, v) \in A \mid u \in V\}|$$
$$d^+(v) = |\{(v, w) \in A \mid w \in V\}|$$

Esse conceito é útil para analisar conectividade, o que nos leva ao próximo tópico, empacotamento de vértices.

Empacotamento de Vértices

Um **empacotamento de vértices** (conjunto independente) em um dígrafo é um conjunto $S \subseteq V$ tal que, no subdígrafo induzido por S, todo vértice tem grau de entrada e de saída iguais a zero. Em notação de graus, se denotamos por $D[S]$ o subdígrafo induzido, então para todo $v \in S$ vale $d^-_{D[S]}(v) = 0$ e $d^+_{D[S]}(v) = 0$. Isso é equivalente a dizer que não existe arco com ambas as extremidades em S (isto é, nenhum $(u, v) \in A$ com $u, v \in S$).

Empacotamento Máximo de Vértices

Um **empacotamento máximo de vértices** é um empacotamento de vértices que contém o maior número possível de vértices. Em outras palavras, é um conjunto $S \subseteq V$ tal que não existem arcos entre vértices em S e S é o maior possível em termos de cardinalidade. Encontrar um empacotamento máximo em um dígrafo é um problema NP-difícil, vamos falar sobre o que isso significa na sessão de algoritmos e complexidade.

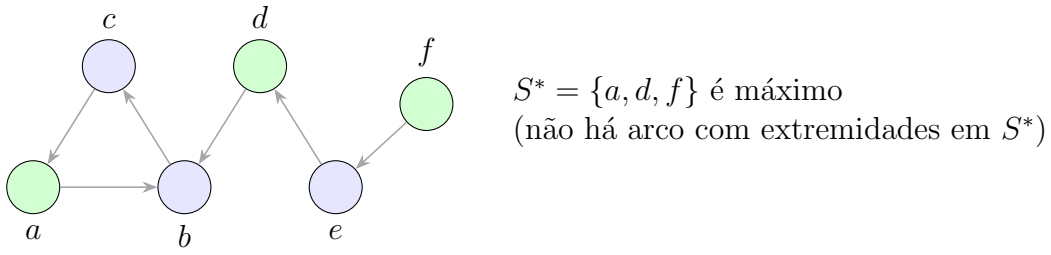


Figura 30: Empacotamento máximo de vértices: para este dígrafo, $S^* = \{a, d, f\}$ (vértices em verde) é um conjunto independente de tamanho máximo.

Esses conceitos de conectividade e empacotamento de vértices nos levam a explorar as subestruturas especiais que existem em dígrafos, que são similares às que vimos em grafos comuns, mas com algumas diferenças importantes devido à direção dos arcos.

2.5.1 Subestruturas em dígrafos

Tal qual os grafos que discutimos na sessão anterior, os dígrafos também possuem as mesmas estruturas especiais, essas estruturas chamadas sub-digrafos mudam um pouco em nomenclatura: caminhos quando direcionados são chamados de trilhas, e ciclos são chamados de circuitos e componentes conexas são componentes fortemente conexas e árvores viram arborescências. Além da nomenclatura, a direção dos arcos traz algumas nuances importantes, discutiremos sobre essas nuances apenas na sessão de arborescências e como os algoritmos de busca mudam bastante em complexidade se estamos tratando de arborescências ou árvores comuns.

Um **subdígrafo** $D' = (V', A')$ de um dígrafo $D = (V, A)$ é um dígrafo onde $V' \subseteq V$ e $A' \subseteq A$. Ou seja, D' é formado por um subconjunto dos vértices e arcos de D.

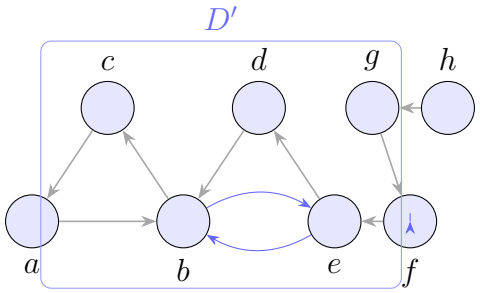


Figura 31: Subdígrafo: o subdígrafo $D' = (V', A')$ está destacado em azul. Aqui, $V' = \{b, c, d, e\}$ e $A' = \{(b, c), (c, b), (d, b), (e, d), (b, e), (e, b)\}$.

Subdigrafos Induzidos

Um subdigrafo pode ser *induzido* por um conjunto de vértices $V' \subseteq V$, denotado como $D[V']$. Nesse caso, o conjunto de arcos A' inclui todos os arcos em A que têm ambas as extremidades em V', ou seja, $A' = \{(u, v) \in A \mid u, v \in V'\}$.

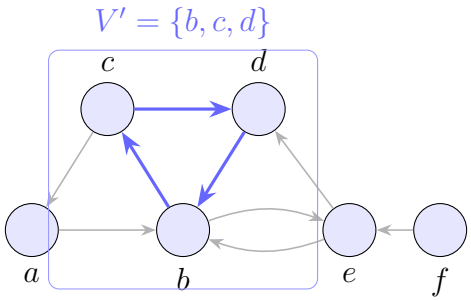
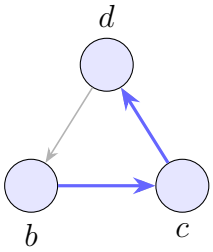


Figura 32: Subdígrafo induzido: para $V' = \{b, c, d\}$, o subdígrafo $D[V']$ contém todos os arcos de D cujas extremidades pertencem a V' (em azul). Arcos que incidem em V' mas conectam a vértices fora de V' não pertencem a $D[V']$.

Subdigrafo Maximal

Um subdigrafo é tido como maximal se não é possível adicionar mais vértices ou arcos a ele sem perder alguma propriedade específica, como conexidade ou aciclicidade.

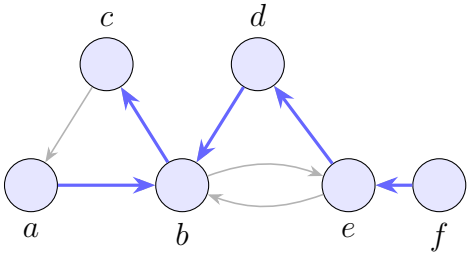


$D' = \{b, c, d\}$, $(b, c), (c, d)$ é acíclico.
Adicionar (d, b) cria o circuito $b \to c \to d \to b$.

Figura 33: Subdígrafo maximal (por aciclicidade): D' é acíclico e maximal em D; adicionar o arco restante (d, b) cria um circuito.

Subdigrafo Gerador

Um subdigrafo é tido como gerador se inclui todos os vértices do dígrafo original, ou seja, $V' = V$. Nesse caso, o subdigrafo é formado por um subconjunto dos arcos do dígrafo original.



$V' = V$
$A' \subseteq A$

Figura 34: Subdígrafo gerador: inclui todos os vértices do dígrafo original ($V' = V$) e apenas um subconjunto dos arcos (em azul).

Com essas definições em mente, podemos explorar as subdigrafos específicos que citaremos ao longo da dissertação, começando pelas trilhas, circuitos, componentes fortemente conexas, componentes-fonte e arborescências.

2.5.2 Subdigrafos Especiais

Trilhas

Uma **trilha** (ou caminho direcionado) em um dígrafo é uma sequência de vértices conectados por arcos que respeitam a direção. Formalmente, uma trilha P de comprimento $k \geq 1$ é uma sequência de vértices $P = (v_1, v_2, \ldots, v_{k+1})$ tal que cada par consecutivo (v_i, v_{i+1}) é um arco em A. O comprimento da trilha é o número de arcos que ela contém, que é k.

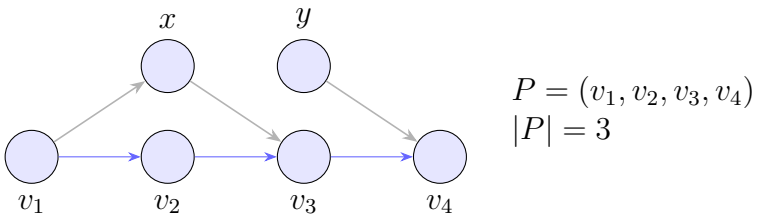


Figura 35: Trilha em dígrafo: a trilha $P = (v_1, v_2, v_3, v_4)$ está destacada em azul. Seu comprimento é o número de arcos percorridos, $|P| = 3$.

Uma conceito importante relacionado às trilhas é o de cortes.

Cortes e Min-cortes

Um **corte** em um dígrafo é um conjunto de arcos cuja remoção desconecta o dígrafo, ou seja, impede que haja uma trilha entre certos pares de vértices. Formalmente, dado um dígrafo $D = (V, A)$, um corte C é um subconjunto de arcos $C \subseteq A$ tal que a remoção dos arcos em C resulta em um dígrafo $D' = (V, A \setminus C)$ onde não existe mais uma trilha (caminho direcionado) entre pelo menos um par de vértices $u, v \in V$.

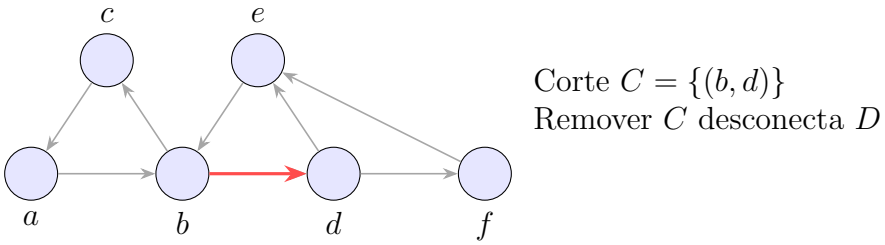


Figura 36: Corte em dígrafo: o corte $C = \{(b, d)\}$ está destacado em vermelho. Remover esse arco desconecta o dígrafo, impedindo caminhos direcionados entre certos pares de vértices.

De forma resumida, dado um dígrafo $D = (V, A)$ e um subconjunto $X \subseteq V$, denotamos por um corte s–t é determinado pela escolha de um conjunto de vértices $X \subseteq V$ tal que $s \in X$ e $t \notin X$; pensa-se nele como a "divisão" do grafo em dois lados: X e $V \setminus X$.

Para tornar a notação precisa e fácil de ler:

- s–t: lê-se "de s para t". Aqui, s é a fonte (onde o fluxo nasce) e t é o sumidouro (onde o fluxo chega).
- $\delta^+(X)$ (fronteira de saída de X): conjunto de todos os arcos que *saem* de X para fora, isto é, para $V \setminus X$.
- $\delta^-(X)$ (fronteira de entrada de X): conjunto de todos os arcos que *entram* em X vindos de $V \setminus X$. Note que $\delta^-(X) = \delta^+(V \setminus X)$.
- **Valor do corte**: dado um peso (ou custo) $c : A \to \mathbb{R}_+$ para cada arco, o valor do corte induzido por X é a soma dos pesos dos arcos que cruzam de X para fora:

$$c(\delta^+(X)) = \sum_{a \in \delta^+(X)} c(a).$$

  No caso não ponderado, esse valor coincide com a *quantidade* de arcos que saem de X.

Exemplo: se $\delta^+(X) = \{(u_1, v_1), (u_2, v_2)\}$ com $c((u_1, v_1)) = 2$ e $c((u_2, v_2)) = 3$, então $c(\delta^+(X)) = 2 + 3 = 5$.

Um min-corte é um corte de tamanho mínimo, ou seja, é o corte com o menor número possível de arcos cuja remoção desconecta o dígrafo. Formalmente, dado um dígrafo $D = (V, A)$, um min-corte C_{min} é um corte tal que para qualquer outro corte C, $|C_{min}| \leq |C|$. O tamanho do min-corte é o número de arcos em C_{min}.

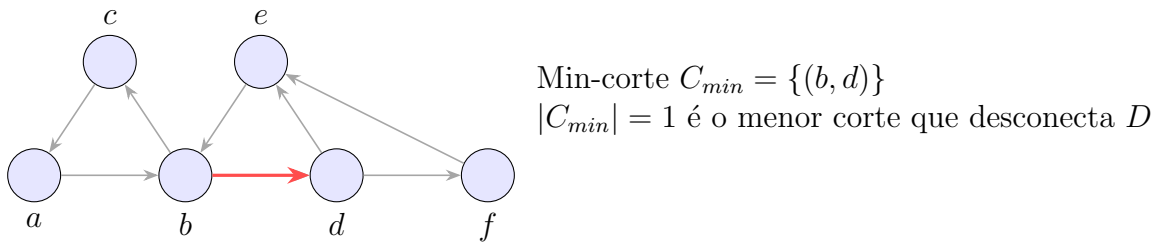


Figura 37: Min-corte em dígrafo: o min-corte $C_{min} = \{(b, d)\}$ está destacado em vermelho. Remover esse arco desconecta o dígrafo, e é o menor corte possível, com tamanho $|C_{min}| = 1$.

Um teorema importante relacionado a min-cortes é o Teorema do Fluxo Máximo - Corte Mínimo, que estabelece uma relação entre o fluxo máximo que pode ser enviado de uma fonte s para um sumidouro t em um dígrafo e o valor do min-corte que separa s de t. Escolhemos apresensentar esse teorema aqui pois ele traz uma intuição interessante sobre a relação entre fluxos e cortes em dígrafos, relevantes para os algoritmos que discutiremos mais adiante.

Teorema: Fluxo–Corte Máximo = Mínimo Corte.

Em dígrafos com limites/pesos não negativos nos arcos, o valor de um fluxo máximo de s para t é igual ao valor de um min-corte s–t. Em símbolos: $\max \text{valor}(f) = \min c(\delta^+(X))$, onde o mínimo é sobre $X \subseteq V$ com $s \in X$, $t \notin X$, e $c(\delta^+(X)) = \sum_{a \in \delta^+(X)} c(a)$.

Prova (esboço):

(i) Desigualdade $\leq$. Seja f um fluxo qualquer. Para um corte $(X, V \setminus X)$ com $s \in X$, $t \notin X$, a conservação de fluxo implica que o fluxo líquido que sai de X é exatamente $\text{valor}(f)$.
Logo,

$$\text{valor}(f) = \sum_{a \in \delta^+(X)} f(a) - \sum_{a \in \delta^-(X)} f(a) \leq \sum_{a \in \delta^+(X)} f(a) \leq \sum_{a \in \delta^+(X)} c(a) = c(\delta^+(X)),$$

pois $f(a) \leq c(a)$ para todo arco (limite). Como isso vale para todo X, obtemos $\text{valor}(f) \leq \min_X c(\delta^+(X))$.

(ii) Desigualdade $\geq$ *e igualdade.* Considere um fluxo máximo f sem caminho aumentante no *grafo residual* R_f (isto é, não há como aumentar o valor do fluxo). Defina X como o conjunto de vértices alcançáveis a partir de s em R_f. Então, não existe arco residual de X para $V \setminus X$; logo, todo arco original que sai de X está saturado ($f(a) = c(a)$), e todo arco que entra em X carrega fluxo zero. Assim,

$$\text{valor}(f) = \sum_{a \in \delta^+(X)} f(a) = \sum_{a \in \delta^+(X)} c(a) = c(\delta^+(X)).$$

Portanto, f atinge exatamente o valor de um corte s–t; em particular, esse corte é mínimo e f é máximo. □

Comentário: Esse resultado é um protótipo de teorema *min–max*: "empacotar" muito fluxo (caminhos) equivale a "cobrir" pouco com um corte. Ver, por exemplo, [**schrijver2003comb**].

Quando falamos em trilhas, precisamos também falar sobre circuitos, que são ciclos direcionados. A diferença entre trilhas e circuitos é que trilhas são caminhos direcionados que não necessariamente retornam ao ponto de origem, enquanto circuitos são caminhos direcionados que começam e terminam no mesmo vértice.

Circuitos

Um **circuito** é um caminho direcionado que começa e termina no mesmo vértice, ou seja, $v_1 = v_{k+1}$. Formalmente, um circuito C é uma sequência de vértices $C = (v_1, v_2, \ldots, v_k, v_1)$ tal que cada par consecutivo (v_i, v_{i+1}) é um arco em A e $k \geq 2$. O comprimento do circuito é o número de arcos que ele contém, que é k.

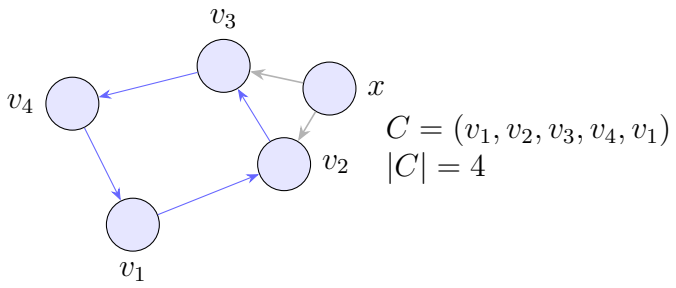


Figura 38: Ciclo direcionado em dígrafo: o ciclo $C = (v_1, v_2, v_3, v_4, v_1)$ está destacado em azul. Seu comprimento é o número de arcos, $|C| = 4$.

Propriedades importantes dos circuitos incluem:

- **Circuito simples**: um circuito é dito simples se não repete vértices, exceto o vértice inicial/final. Ou seja, $v_i \neq v_j$ para $1 \leq i < j \leq k$.
- **Circuito Euleriano**: um circuito que percorre cada arco exatamente uma vez. Um dígrafo possui um circuito euleriano se e somente se é fortemente conexo e o grau de entrada é igual ao grau de saída para cada vértice.
- **Circuito Hamiltoniano**: um circuito que visita cada vértice exatamente uma vez, exceto o vértice inicial/final. Determinar a existência de um circuito hamiltoniano é um problema que chamamos de NP-completo. Vamos explicar mais sobre isso na seção de algoritmos e complexidade computacional.

Existe um princípio chamado de princípio da casa dos pombos, que diz que se você tem mais pombos do que casas, pelo menos uma casa deve conter mais de um pombo. Em termos de grafos, isso se traduz na ideia de que se um grafo tem mais arestas do que vértices, ele deve conter pelo menos um ciclo.

Esse princípio também se aplica a dígrafos, mas com uma nuance importante: em dígrafos, o critério correto para garantir a existência de um circuito dirigido é que o grau mínimo de saída (ou de entrada) seja pelo menos 1. Ou seja, se cada vértice em um dígrafo tem pelo menos um arco saindo dele (ou entrando nele), então o dígrafo deve conter pelo menos um circuito dirigido. Abaixo apresentamos esse resultado formalmente.

Lema: Princípio da casa dos pombos para circuitos.

Se $D = (V, A)$ é um dígrafo finito em que todo vértice tem grau de saída ao menos 1, isto é, $d^+(v) \geq 1$ para todo $v \in V$, então D contém pelo menos um circuito dirigido. (De forma equivalente, a afirmação vale trocando "saída" por "entrada".)

Prova: Escolha para cada $v \in V$ um arco $(v, f(v))$ que sai de v (possível porque $d^+(v) \geq 1$). Fixado um vértice v_0, considere a sequência $v_0, v_1 = f(v_0), v_2 = f(v_1), \ldots$. Como V é finito, algum vértice repete: existem $i < j$ com $v_i = v_j$. O trecho $v_i \to v_{i+1} \to \cdots \to v_j = v_i$ é um circuito dirigido em D. A versão com graus de entrada segue aplicando o argumento ao dígrafo com arcos invertidos. □

Observação: A condição $|A| > |V|$ *não* garante a existência de circuito dirigido em geral (há orientações acíclicas com muitos arcos). O critério correto, simples e útil, é o grau mínimo de saída (ou de entrada) ser pelo menos 1. Ver, por exemplo, [**schrijver2003comb**].

Esse princípio ajuda a entender o comportamento básico dos métodos que os algoritmos empregam para encontrar as arborescências de custo mínimo. Vamos elaborar melhor esse ponto na seção de algoritmos em arborescências, especialmente ao discutir o algoritmo de Chu–Liu/Edmonds.

No algoritmo de Chu–Liu/Edmonds que exploraremos em detalhes em capítulo posterior, para cada vértice $v \neq r$, escolhemos a aresta de menor custo que entra em v, o subgrafo obtido fica com grau de entrada igual a 1 em todos os $v \neq r$. Pelo lema, enquanto esse subgrafo ainda não for uma arborescência, ele necessariamente contém um circuito: assim o algoritmo se baseia em detectar o circuitos, contraí-lo a um único vértice e repetir a seleção sob custos reduzidos. Quando não houver mais circuitos, as arestas escolhidas formam uma arborescência ótima enraizada em r.

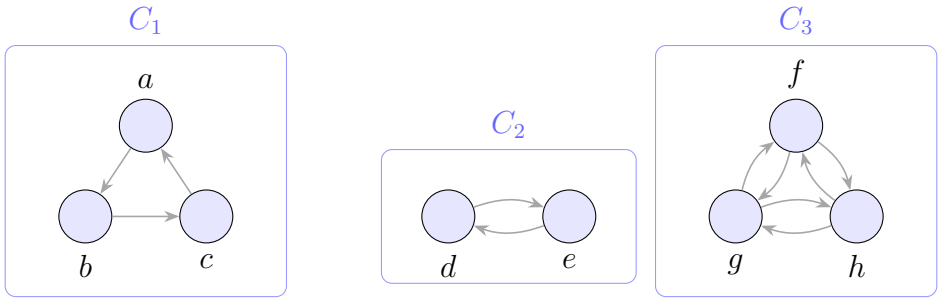
Componentes fortemente conexas

Uma **componente fortemente conexa** (abreviaremos como **CFC**) é um subdígrafo maximal onde existe um caminho direcionado (trilha) entre qualquer par de vértices.

Formalmente, uma componente fortemente conexa C é um subdígrafo $C = (V_C, A_C)$ onde $V_C \subseteq V$ e $A_C \subseteq A$, que satisfaz a seguinte propriedade:

- C é fortemente conexo: existe um caminho direcionado entre qualquer par de vértices em V_C.

Além disso, C é maximal, o que significa que não é possível adicionar mais vértices ou arcos a C sem perder a propriedade de forte conexidade.



Cada caixa destaca uma *componente fortemente conexa*.
Não há arcos entre C_1, C_2 e C_3.

Figura 39: Componentes fortemente conexas: o dígrafo possui três componentes C_1, C_2 e C_3. Cada C_i é fortemente conexo e *maximal*: adicionar qualquer vértice de fora quebraria a propriedade de forte conexidade interna ou exigiria arcos ausentes.

A seguir falaremos sobre a propriedade de alcançabilidade mútua em CFC, que é fundamental para entendermos a relação entre elas e arborescências (que apenas mencionaremos aqui, para aprofundarmos na sessão sobre arborescências e com os grafos acíclicos dirigidos (DAGs))[1]:

- **CFCs e alcançabilidade mútua:** dois vértices u e v pertencem à mesma CFC se, e somente se, $u \rightsquigarrow v$ e $v \rightsquigarrow u$. Ou seja, existe um caminho direcionado de u para v e um caminho direcionado de v para u. A relação de pertencer à mesma CFC é uma relação de equivalência que particiona o conjunto de vértices V em subconjuntos disjuntos, cada um correspondendo a uma CFC.

- **Arborescências em dígrafos fortemente conexos:** um dígrafo é fortemente conexo se, e somente se, para algum (equiv., para todo) vértice r existem uma *arborescência de saída* enraizada em r que alcança todos os vértices e uma *arborescência de entrada* enraizada em r que é alcançada por todos os vértices. De fato, se D é fortemente conexo, basta rodar buscas a partir de r; no sentido inverso, $r \rightsquigarrow v$ pela arborescência de saída e $v \rightsquigarrow r$ pela de entrada, implicando alcançabilidade mútua entre quaisquer dois vértices via r.

- **CFC e DAG.** Ao *contrair* cada CFC a um único vértice obtemos o grafo condensado $\mathrm{Cond}(D)$[2]. Não há circuitos dirigidos em $\mathrm{Cond}(D)$; portanto, ele é um DAG.

  Consequências: todo DAG tem ao menos uma componente-fonte (falaremos em seguida sobre eles) e uma componente-sumidouro; logo, $\mathrm{Cond}(D)$ também tem ao menos uma CFC-fonte e ao menos uma CFC-sumidouro.

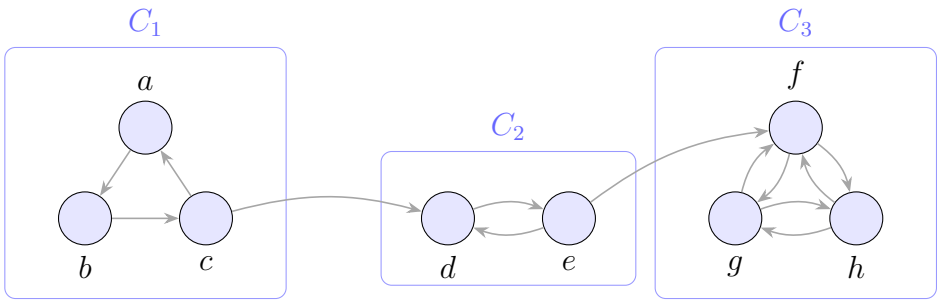
Componentes-fonte

[1] Um DAG (Directed Acyclic Graph) é um grafo direcionado que não contém ciclos. Em outras palavras, não é possível começar em um vértice e seguir uma sequência de arcos que retorne ao mesmo vértice. Os DAGs são estruturas fundamentais em muitas áreas da computação, incluindo a representação de dependências e a modelagem de processos.

[2] O grafo condensado é uma representação simplificada do dígrafo original, onde as CFCs são representadas como vértices únicos.

Uma **componente-fonte** é uma componente fortemente conexa que não possui arcos direcionados saindo dela para outras componentes. Formalmente, uma componente-fonte C é uma componente fortemente conexa $C = (V_C, A_C)$ onde $V_C \subseteq V$ e $A_C \subseteq A$, que satisfaz a seguinte propriedade:

- Não existem arcos $(u, v) \in A$ tais que $u \in V_C$ e $v \notin V_C$.
- C é maximal: não é possível adicionar mais vértices ou arcos a C sem perder a propriedade de forte conexidade.
- C é fortemente conexo: existe um caminho direcionado entre qualquer par de vértices em V_C.



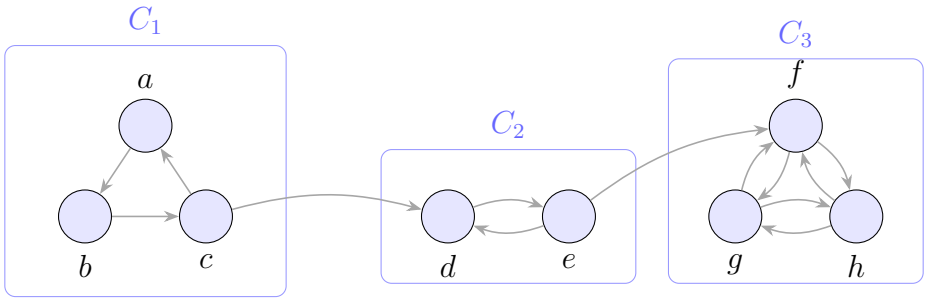
A componente C_1 é uma *componente-fonte* porque não há arcos saindo dela para outras componentes. Já C_2 e C_3 não são componentes-fonte, pois há arcos entrando nelas.

Figura 40: Componente-fonte: o dígrafo possui três componentes C_1, C_2 e C_3. A componente C_1 é uma componente-fonte porque não há arcos saindo dela para outras componentes. Já C_2 e C_3 não são componentes-fonte, pois há ar cos entrando nelas.

Componentes-sumidouro

Uma **componente-sumidouro** é uma componente fortemente conexa que não possui arcos direcionados entrando nela vindos de outras componentes. Formalmente, uma componente-sumidouro C é uma componente fortemente conexa $C = (V_C, A_C)$ onde $V_C \subseteq V$ e $A_C \subseteq A$, que satisfaz a seguinte propriedade:

- Não existem arcos $(u, v) \in A$ tais que $u \notin V_C$ e $v \in V_C$.
- C é maximal: não é possível adicionar mais vértices ou arcos a C sem perder a propriedade de forte conexidade.
- C é fortemente conexo: existe um caminho direcionado entre qualquer par de vértices em V_C.



A componente C_3 é uma *componente-sumidouro* porque não há arcos entrando nela vindos de outras componentes. Já C_1 e C_2 não são componentes-sumidouro, pois há arcos saindo delas.

Figura 41: Componente-sumidouro: o dígrafo possui três componentes C_1, C_2 e C_3. A componente C_3 é uma componente-sumidouro porque não há arcos entrando nela vindos de outras componentes. Já C_1 e C_2 não são componentes-sumidouro, pois há arcos saindo delas.

Existe um resultado clássico que relaciona o número de componentes-fonte e componentes-sumidouro em um dígrafo com o número mínimo de arcos necessários para torná-lo fortemente conexo.

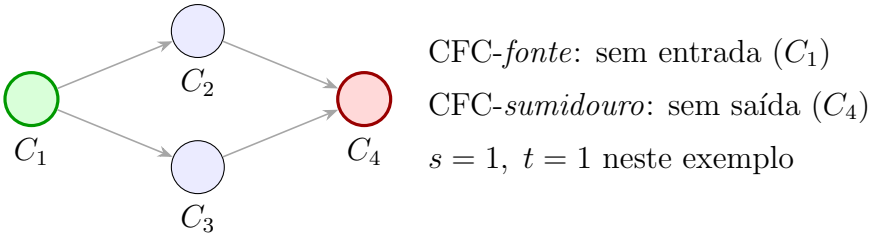


Figura 42: Grafo condensado Cond(D) e a contagem de CFCs-fonte/sumidouro: C_1 não recebe arcos (fonte) e C_4 não emite arcos (sumidouro).

Seja $D = (V, A)$ um dígrafo com k componentes fortemente conexas (CFCs):

Denote por Cond(D) o grafo *condensado*, obtido ao contrair cada CFC de D em um único vértice; Cond(D) é sempre um DAG. Escreva s para o número de CFCs-*fonte* (vértices de Cond(D) sem arcos de *entrada*) e t para o número de CFCs-*sumidouro* (vértices de Cond(D) sem arcos de *saída*). Com essa notação, vale o seguinte:

Lema: mínimo de arcos para conexidade forte

Se $k = 1$, então D já é fortemente conexo e o mínimo de arcos a adicionar é 0. Se $k \geq 2$, o número mínimo de arcos a adicionar para tornar D fortemente conexo é

$$\boxed{\min = \max\{s,\, t\}}.$$

Prova (esboço).

Necessidade: em qualquer supergrafo fortemente conexo, cada CFC-fonte deve receber ao menos um arco *entrando* e cada CFC-sumidouro deve ter ao menos um arco *saindo*; logo, são necessários ao menos $\max\{s, t\}$ arcos novos.

Suficiência: numa ordenação topológica de $\mathrm{Cond}(D)$, conecte CFCs-sumidouro a CFCs-fonte de modo a formar um ciclo que percorra todas as CFCs. Isso pode ser feito com exatamente $\max\{s, t\}$ arcos, mesmo quando $s \neq t$, emparelhando sobras de um lado com o outro.

Ver, por exemplo, textos clássicos sobre dígrafos (e.g., [**schrijver2003comb**]).

Esse lema explica por que todo dígrafo com mais de uma CFC-fonte ou mais de uma CFC-sumidouro não pode ser fortemente conexo. Além disso, ele é útil em algoritmos que buscam tornar um dígrafo fortemente conexo, pois fornece um limite inferior para o número de arcos que precisam ser adicionados.

As noções de CFCs e componentes-fonte conversam diretamente com o conceito de *arborescências*. Pois, em uma arborescência, todos os vértices são alcançáveis a partir da raiz, o que não implica que a arborescência é fortemente conexa, mas encontrar essas componentes dentro de um dígrafo pode nos ajudar em uma busca otimizada por arborescências.

Arborescências

Uma **arborescência** é um dígrafo acíclico e conexo, onde há um vértice especial chamado *raiz* que tem um caminho direcionado para todos os outros vértices. Formalmente, uma arborescência T é um dígrafo $T = (V_T, A_T)$ onde $V_T \subseteq V$ e $A_T \subseteq A$, que satisfaz as seguintes propriedades:

- T é conexo: existe um caminho direcionado da raiz para qualquer vértice em V_T.

- T é acíclico: não contém ciclos direcionados.

Além disso, uma arborescência com n vértices sempre tem exatamente $n - 1$ arcos.

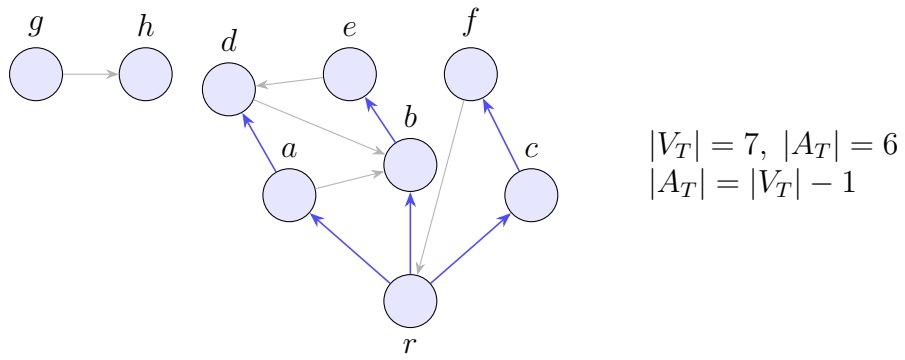


Figura 43: Arborescência: dígrafo conexo e acíclico com raiz r de onde há um caminho direcionado para todos os outros vértices em azul. No exemplo, $|V_T| = 7$ e $|A_T| = 6$, satisfazendo $|A_T| = |V_T| - 1$. Em cinza, arcos que não fazem parte da arborescência.

As arborescências são o principal objeto de investigação desse trabalho, portanto vamos usar uma sessão dedicada a elas para apresentar suas características, variações e aplicações.

2.6 Arborescências em foco

Já tratamos do conceito básico de arborescência, agora falaremos de arborescências especiais:

Arborescência Geradora: Uma arborescência é considerada geradora se inclui todos os vértices do dígrafo original, ou seja, $V_T = V$. Nesse caso, a arborescência é formada por um subconjunto dos arcos do dígrafo original.

Arborescência Maximal: Uma arborescência é dita maximal se não é possível adicionar mais vértices ou arcos a ela sem perder a propriedade de ser uma arborescência, ou seja, sem criar ciclos ou desconectar o dígrafo.

Ramificações Geradoras

Uma **ramificação geradora** é um subdígrafo que é uma arborescência que inclui todos os vértices do dígrafo original. Formalmente, uma ramificação geradora R é um subdígrafo $R = (V_R, A_R)$ onde $V_R = V$ e $A_R \subseteq A$, que satisfaz as seguintes propriedades:

- R é uma arborescência: existe um vértice especial chamado raiz que tem um caminho direcionado para todos os outros vértices.
- R é maximal: não é possível adicionar mais arcos a R sem perder a propriedade de ser uma arborescência.

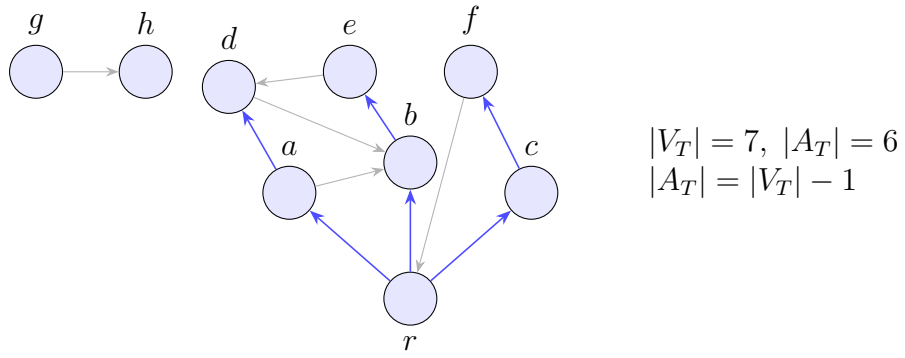


Figura 44: Ramificação geradora: arborescência que inclui todos os vértices do dígrafo original, em azul. No exemplo, $|V_T| = 7$ e $|A_T| = 6$, satisfazendo $|A_T| = |V_T| - 1$. Em cinza, arcos que não fazem parte da ramificação geradora.

Quando falamaos de ramificações geradoras, podemos falar de uma estrutura que fixa um vértice raiz r e constrói uma arborescência que alcança todos os outros vértices a partir dessa raiz. Essa estrutura é conhecida como *r-arborescência*.

Arborescência de Raiz Específica: uma arborescência de raiz específica é uma arborescência onde a raiz é um vértice pré-determinado do dígrafo. Isso é útil em situações onde um ponto de origem específico deve ser o início dos caminhos direcionados para todos os outros vértices. Podemos chamá-la de r-arborescência, onde r é o vértice raiz.

Arborescência inversa (in-arborescência): uma *arborescência inversa* enraizada em r — também chamada de *in-arborescência* — é o resultado de inverter a orientação de todos os arcos de uma arborescência (out-arborescência) enraizada em r. Equivalentemente: é um dígrafo acíclico no qual, para todo $v \neq r$, existe *exatamente um* caminho direcionado de v até r (isto é, todos os arcos estão orientados *em direção* à raiz). Em termos de graus, numa in-arborescência cada vértice $v \neq r$ tem grau de saída igual a 1 (o arco para seu "pai") e a raiz r tem grau de saída 0; os graus de entrada são complementares aos de uma out-arborescência.

As arborecências podem ter custos associados aos seus arcos, o que nos leva ao conceito de arborescência de custo mínimo.

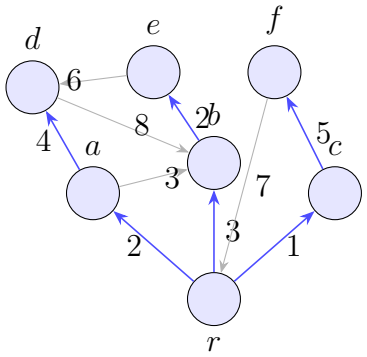
Arborescência de Custo Mínimo:

Uma **arborescência de custo mínimo** é uma arborescência que minimiza a soma dos pesos dos arcos que a compõem. Esse conceito é especialmente relevante em aplicações onde os arcos têm custos associados, como em redes de transporte ou comunicação.

Finalmente podemos conceituar a principal estrutura que estudaremos nesta dissertação: a r-arborescência de custo mínimo e sua variante, a r-arborescência inversa de custo mínimo.

r-arborescência de custo mínimo: é uma arborescência enraizada em um vértice específico r que minimiza a soma dos pesos dos arcos que a compõem. Formalmente, dada uma função de custo $c : A \to \mathbb{R}_{\geq 0}$ que atribui um custo a cada arco do dígrafo $D = (V, A)$, uma r-arborescência de custo mínimo T é uma arborescência $T = (V_T, A_T)$ onde $V_T \subseteq V$ e $A_T \subseteq A$, que satisfaz as seguintes propriedades:

- T é uma arborescência enraizada em r: existe um caminho direcionado de r para qualquer vértice em V_T.
- T minimiza o custo total: a soma dos custos dos arcos em A_T é mínima, ou seja, $\sum_{a \in A_T} c(a)$ é minimizada.

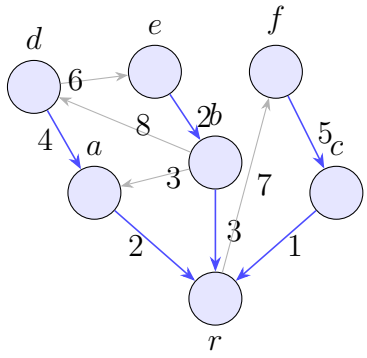


Custo total da r-arborescência: $2 + 3 + 1 + 4 + 2 + 5 = 17$

Figura 45: r-arborescência de custo mínimo: arborescência enraizada em r que minimiza a soma dos custos dos arcos, em azul. No exemplo, o custo total é 17. Em cinza, arcos que não fazem parte da r-arborescência de custo mínimo.

r-arborescência inversa de custo mínimo: é uma arborescência inversa enraizada em um vértice específico r que minimiza a soma dos pesos dos arcos que a compõem. Formalmente, dada uma função de custo $c : A \to \mathbb{R}_{\geq 0}$ que atribui um custo a cada arco do dígrafo $D = (V, A)$, uma r-arborescência inversa de custo mínimo T é uma arborescência inversa $T = (V_T, A_T)$ onde $V_T \subseteq V$ e $A_T \subseteq A$, que satisfaz as seguintes propriedades:

- T é uma arborescência inversa enraizada em r: existe um caminho direcionado de qualquer vértice em V_T até r.
- T minimiza o custo total: a soma dos custos dos arcos em A_T é mínima, ou seja, $\sum_{a \in A_T} c(a)$ é minimizada.



Custo total da r-arborescência inversa: $2 + 3 + 1 + 4 + 2 + 5 = 17$

Figura 46: r-arborescência inversa de custo mínimo: arborescência inversa enraizada em r que minimiza a soma dos custos dos arcos, em azul. No exemplo, o custo total é 17. Em cinza, arcos que não fazem parte da r-arborescência inversa de custo mínimo.

As arborescências são a principal estrutura que exploraremos ao longo desta dissertação, especialmente a r-arborescência de custo mínimo e r-arborescência inversa de custo mínimo, abordaremos o problema de encontrá-las eficientemente em dígrafos com custos associados aos arcos.

Noções aprofundadas em arborescências

Vamos explorar algumas propriedades e teoremas importantes relacionados a arborescências, que serão úteis para entender os algoritmos que discutiremos posteriormente.

Grau e contagem de arcos:

Seja T uma out-arborescência enraizada em r com n vértices. Ela é exatamente a estrutura que satisfaz as três condições combinadas abaixo (todas muito fáceis de checar):

1. (Contagem) $|A_T| = n - 1$.
2. (Entrada única) Cada vértice $v \neq r$ recebe exatamente um arco: $d_T^-(v) = 1$.
3. (Raiz sem entrada) A raiz não recebe arcos: $d_T^-(r) = 0$.

De forma simétrica, numa in-arborescência (arborescência inversa) valem as versões "espelhadas": cada $v \neq r$ tem exatamente um arco *saindo* ($d_T^+(v) = 1$) e a raiz tem grau de saída zero ($d_T^+(r) = 0$).

Reciprocamente, qualquer subdígrafo que satisfaça (1)–(3) é uma out-arborescência enraizada em r (e análogamente no caso inverso). Essas três condições serão nosso critério operacional para reconhecer arborescências.

Teorema das arborescências disjuntas de Edmonds: Seja $D = (V, A)$ e $r \in V$. Para um inteiro $k \geq 1$, existem k out-arborescências enraizadas em r par de arcos disjuntas se, e somente se, para todo $X \subseteq V \setminus \{r\}$ tem-se $|\delta^-(X)| \geq k$. Essa condição de cortes é necessária (cada arborescência deve "entrar" em X por um arco distinto) e suficiente (prova por matroides ou via teoremas de enxerto). Referências clássicas: Edmonds (1973), e [**schrijver2003comb**].

Nota sobre custo mínimo e dualidade (visão de alto nível): No problema de r-arborescência de custo mínimo, algoritmos como Chu–Liu/Edmonds e o primal–dual de Frank operam com *custos reduzidos*: subtraem de cada componente-fonte o menor custo das arestas que a "alimentam", fazendo surgir arestas de custo reduzido zero e circuitos/condensações que guiam contrações. No ótimo, há uma escolha de potenciais (variáveis duais) tal que todas as arestas escolhidas têm custo reduzido zero e cada conjunto "apertado" recebe exatamente uma aresta, certificando otimalidade. Ver [**frank2014connections**, **schrijver2003comb**]. Esses dois conceitos nos levam à ideia de ramificações geradoras, que são arborescências que abrangem todos os vértices do dígrafo original.

Cortes dirigidos e arborescências: uma visão dual

As ideias de cortes também aparecem no problema central desta dissertação. Para custos não negativos e um vértice-raiz r, o custo mínimo de uma arborescência geradora dirigida é igual ao valor máximo de uma função z definida em subconjuntos $X \subseteq V \setminus \{r\}$ que respeita as desigualdades

$$c(a) \;\geq\; z(X) \quad \text{para todo arco } a \text{ que entra } X.$$

Intuitivamente, $z(X)$ é um "preço" pago para entrar em X; nenhum arco pode cobrar menos do que a soma dos preços dos conjuntos que ele entra. Esse é o teorema de Fulkerson (ver, por exemplo, [**frank2014**, **schrijver2003comb**]).

Essa formulação leva a um algoritmo em duas fases (Fulkerson–Frank). Na Fase 1, ajustamos os custos de modo a tornar certos arcos de custo zero (0-arcos), aumentando gradualmente valores $z(X)$ para conjuntos mínimos sem 0-arcos entrando. Na Fase 2, extraímos uma arborescência apenas com 0-arcos, escolhendo-os de forma gulosa. Quando os dois lados "se encontram" (valor dual = valor primal), obtemos ao mesmo tempo a arborescência ótima e um certificado de otimalidade. Essa visão se conecta diretamente à normalização de custos e à contração/reexpansão de ciclos no algoritmo de Chu–Liu/Edmonds [**chu1965**, **edmonds1967optimum**].

2.7 Algoritmo Chu e Liu

O algoritmo de Chu e Liu (1965) é um método clássico para encontrar a r-arborescência de custo mínimo em um dígrafo com custos associados aos arcos. O algoritmo funciona da seguinte maneira:

3 Considerações Finais