**Universidade do Estado do Rio de Janeiro**
Centro de Tecnologia e Ciências
Instituto de Matemática e Estatística

Leonardo de Almeida Cavadas

# Hipergrafos de Dijkstra: Reconhecimento e Isomorfismo

Rio de Janeiro
2024

Leonardo de Almeida Cavadas

# Hipergrafos de Dijkstra: Reconhecimento e Isomorfismo

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Orientadores: Prof. Dr. Luerbio Faria
Profa. Dra. Lucila Maria de Souza Bento
Prof. Dr. Jayme Luiz Szwarcfiter

Rio de Janeiro
2024

CATALOGAÇÃO NA FONTE

UERJ/REDE SIRIUS/BIBLIOTECA CTC/A

C377 Cavadas, Leonardo de Almeida.
Hipergrafos de Dijkstra: reconhecimento e isomorfismo/ Leonardo de Almeida Cavadas. – 2024.
59 f.: il.

Orientadores: Luerbio Faria, Lucila Maria de Souza Bento, Jayme Luiz Szwarcfiter.
Dissertação (Mestrado em Ciências Computacionais) - Universidade do Estado do Rio de Janeiro, Instituto de Matemática e Estatística.

1. Programação estruturada - Teses. 2. Processamento parelelo (Computadores) - Teses. 3. Teoria dos grafos - Teses. I. Faria, Luerbio. II. Bento, Lucila Maria de Souza. III. Szwarcfiter, Jayme Luiz. IV. Universidade do Estado do Rio de Janeiro. Instituto de Matemática e Estatística. V. Título.

CDU 004.42

Patricia Bello Meijinhos CRB7/5217 - Bibliotecária responsável pela elaboração da ficha catalográfica

Leonardo de Almeida Cavadas

**Hipergrafos de Dijkstra: Reconhecimento e Isomorfismo**

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Ciências Computacionais, da Universidade do Estado do Rio de Janeiro.

Aprovada em 29 de Fevereiro de 2024.
Banca Examinadora:

---

Prof. Dr. Luerbio Faria (Orientador)
Instituto de Matemática e Estatística – UERJ

---

Profa. Dra. Lucila Maria de Souza Bento (Orientador)
Instituto de Matemática e Estatística – UERJ

---

Prof. Dr. Jayme Luiz Szwarcfiter (Orientador)
Instituto de Matemática e Estatística – UERJ

---

Prof. Dr. Rubens André Sucupira
Instituto de Matemática e Estatística – UERJ

---

Prof. Dr. André Luiz Pires Guedes
Universidade Federal do Paraná

---

Profa. Dra. Lilian Markenzon
Universidade Federal do Rio de Janeiro

---

Prof. Dr. Valmir Carneiro Barbosa
Universidade Federal do Rio de Janeiro

Rio de Janeiro

2024

## AGRADECIMENTOS

Agradeço, primeiramente, a Deus por ter me dado forças para seguir em frente e pelo aprendizado que tive dentro na universidade, tanto acadêmico como de vida, e a minha família pelo apoio durante o mestrado.

Agradeço a Fundação Carlos Chagas Filho de Amparo à Pesquisa do Estado do Rio de Janeiro - FAPERJ pela bolsa de mestrado concedida.

Agradeço também aos professores Luerbio Faria, Lucila Maria de Souza Bento e Jayme Luiz Szwarcfiter por aceitarem me orientar durante o mestrado e pela transmissão de conhecimento para que eu pudesse desenvolver o trabalho.

E por último, mas não menos importante, agradeço pelas pessoas que conheci no mestrado, tanto alunos como funcionários.

# RESUMO

CAVADAS, Leonardo de Almeida. Hipergrafos de Dijkstra: reconhecimento e isomorfismo. 59 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

Um programa de computador frequentemente pode ter seu fluxo de controle representado por meio de grafos direcionados. Dijkstra apresentou os conceitos da programação sequencial estruturada como aquela que não contém a estrutura GO TO. Hetch e Ullman apresentaram os grafos direcionados de fluxo redutível, onde os grafos direcionados associados a um programa sequencial formam uma subclasse própria da classe dos grafos de fluxo redutível e esses grafos direcionados podem representar o fluxo de execução de um programa sequencial estruturado como descrito por Dijkstra. Dijkstra também estabeleceu o conceito de programação estruturada paralela introduzindo o comando PARBEGIN/PAREND. Bento et al. caracterizaram a classe dos grafos de Dijkstra os quais correspondem a grafos de fluxo de programas escritos usando programação sequencial estruturada, conforme descrito por Dijkstra. Bento et al. provaram que o reconhecimento desses grafos é linear, bem como a checagem do isomorfismo entre grafos dessa classe. Guedes e Markenzon introduziram o conceito de hipergrafos de fluxo redutível que estende, naturalmente, a classe dos grafos de fluxo redutível. A classe dos hipergrafos de fluxo redutíveis é uma superclasse própria da classe dos grafos de fluxo de programas paralelos estruturados. Nessa dissertação definimos os hipergrafos direcionados de Dijkstra como a classe que caracteriza os grafos de fluxo de programas paralelos estruturados que têm a estrutura paralela PARBEGIN/PAREND. Provamos que o reconhecimento desses hipergrafos é linear, bem como a checagem do isomorfismo entre hipergrafos dessa classe.

Palavras-chave: hipergrafos. hipergrafos de fluxo redutível. programação paralela e estruturada. isomorfismo.

## ABSTRACT

CAVADAS, Leonardo de Almeida. Dijkstra Hypergraphs: Recognition and Isomorphism. 59 f. Dissertação (Mestrado em Ciências Computacionais) – Instituto de Matemática e Estatística, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2024.

A computer program's control flow can often be represented through directed graphs. Dijkstra introduced the concepts of structured sequential programming as one that does not contain the GO TO structure. Hetch and Ullman presented reducible flow directed graphs, where the directed graphs class associated with a sequential program is a proper subclass of the class of reducible flow graphs, and these directed graphs can represent the execution flow of a sequential structured program as described by Dijkstra. Dijkstra also established the concept of parallel structured programming by introducing the PARBEGIN/PAREND command. Bento et al. characterized the class of Dijkstra's graphs which correspond to flow graphs of programs written using structured sequential programming, as described by Dijkstra. Bento et al. proved that the recognition of these graphs is linear, as well as checking the isomorphism between graphs of this class. Guedes and Markenzon introduced the concept of reducible flow hypergraphs, which naturally extends the class of reducible flow graphs. The class of reducible flow hypergraphs is a superclass of the class of flow graphs of structured parallel programs. In this dissertation, we define Dijkstra's directed hypergraphs as the class that characterizes the flow graphs of structured parallel programs having the parallel structure PARBEGIN/PAREND. We prove that the recognition of these hypergraphs is linear, as well as checking the isomorphism between hypergraphs of this class.



Keywords: hypergraphs. reducible flow hypergraphs. parallel and structured programming. isomorphism.

## LISTA DE FIGURAS

## LISTA DE TABELAS

## LISTA DE ALGORITMOS

## SUMÁRIO

# INTRODUÇÃO

A programação estruturada é um paradigma de programação que visa melhorar a clareza, a qualidade e o tempo de desenvolvimento e depuração de um programa de computador através da utilização extensiva de sub-rotinas, estruturas de blocos, laços de repetição *para*, *enquanto* e outras construções de fluxo de controle, onde as ideias básicas apareceram em detalhes no trabalho de Dijkstra et al. [DAHL; DIJKSTRA; HOARE, 1972]. A principal ideia subjacente à programação estruturada é dividir um programa em partes menores e mais fáceis de gerir que possam ser facilmente compreendidas, modularizadas e mantidas. Esta abordagem ajuda a reduzir a complexidade de um programa, tornando-o mais fácil de ler, escrever e depurar.

As bases para definição da programação estruturada foram estabelecidas por Dijkstra [DIJKSTRA, 1968a]. Ele argumentou contra o uso indiscriminado da instrução GO TO na programação, destacando os problemas que ela pode acarretar na legibilidade, manutenção e confiabilidade do código. Dijkstra defendeu o uso de estruturas de controle mais estruturadas, como estruturas de repetição e condicionais, o que resulta em um código mais claro e menos propenso a erros. Outros autores, como Knuth [KNUTH, 1974] e Knuth e Floyd [KNUTH; FLOYD, 1971], também criticam o uso excessivo do GO TO no desenvolvimento de programas.

Em 1972, Dijkstra et. al. [DAHL; DIJKSTRA; HOARE, 1972] publicaram o livro *Structured Programming* que veio a se tornar um clássico na programação de computadores. O livro oferece uma visão profunda da programação estruturada, destacando os princípios do paradigma, com o uso de estruturas de sequência, seleção e repetição, técnicas para escrever código claro, conciso e eficiente, e práticas, como coesão, acoplamento e simplicidade para criar sistemas que sejam fáceis de entender, manter e modificar, considerados como essenciais para o desenvolvimento de software de alta qualidade.

Também em 1972, Hecht e Ullman introduziram o conceito de grafos de fluxo redutíveis. Eles argumentaram que a estrutura dos programas pode frequentemente ser descrita por meio de uma técnica chamada análise de intervalos aplicada aos grafos de fluxo de controle, especialmente em programas sem instruções GO TO [HECHT; ULLMAN, 1972]. Outras caracterizações dos grafos de fluxo redutíveis também foram apresentadas pelos autores mais tarde [HECHT; ULLMAN, 1974].

Baseados na modelagem proposta por Dijkstra et. al., Bento et. al. [BENTO et al., 2019] definiram grafos de Dijkstra, que modelam os grafos de fluxo de controle dos programas sequenciais estruturados. Os autores provaram o reconhecimento linear dessa classe, além de exibir um algoritmo linear para verificar se grafos de Dijkstra são isomorfos.

Além de suas contribuições para a definição da programação estruturada, Dijkstra

foi um dos pioneiros na discussão de conceitos fundamentais de programação concorrente e sincronização de processos. Em seu trabalho [DIJKSTRA, 1965], explica como ocorre a comunicação entre os processos por meio de variáveis compartilhadas, observando os riscos de inconsistência e concorrência desordenada. Para a coordenação dos processos, o autor propõe o uso de "semáforos" como um mecanismo de coordenação para garantir a sincronização entre processos concorrentes. A ideia de estruturas de controle de fluxo para coordenar a execução de processos concorrentes, comunicação entre processos, semáforos, monitores, *deadlocks* e PARBEGIN/PAREND são introduzidas como mecanismos para expressar a concorrência de forma mais explícita e estruturada. Semelhante ao conceito de blocos BEGIN e END em linguagens de programação convencionais, o PARBEGIN e o PAREND fornecem um bloco dentro do qual múltiplos processos podem ser iniciados simultaneamente e são executados concorrentemente, fornecendo uma abstração que lida com a complexidade da concorrência de forma mais organizada e estruturada, facilitando o desenvolvimento de sistemas concorrentes mais robustos e compreensíveis.

Dijkstra [DIJKSTRA, 1968b] também fala sobre multiprogramação, comunicação entre processos, concorrência e sincronização, o gerenciamento dos processos e sistema de arquivos. A multiprogramação viabiliza a execução simultânea de múltiplos programas em um computador, enquanto o gerenciamento de processos engloba atividades como escalonamento de CPU, alocação de memória e comunicação entre processos. A comunicação interprocesso permite a troca de informações entre os processos em execução, enquanto os sistemas de arquivos possibilitam o armazenamento e recuperação eficientes de dados de forma organizada.

Per Brinch Hansen [HANSEN, 1972], com a ideia da multiprogramação, propôs a multiprogramação estruturada baseada em associar explicitamente uma estrutura de dados compartilhada por processos concorrentes com operações definidas sobre ela. A multiprogramação estruturada permite esclarecer o significado dos programas e capturar uma grande classe de erros dependentes do tempo durante a compilação. Essa abordagem visa melhorar a compreensão, a verificação e a eficiência na utilização de recursos em ambientes concorrentes, contribuindo para a confiabilidade e robustez dos sistemas de software. Naturalmente a multiprogramação estruturada introduz conceitos-chave, tais como processos disjuntos que interagem entre si, regiões críticas e variáveis compartilhadas e controle de sincronização. Esses conceitos foram projetados para serem seguros não apenas em sistemas operacionais, mas também em programas de usuário, fornecendo propriedades simples e verificação eficiente em tempo de compilação.

Em 1973, Per Brinch Hansen [HANSEN, 1973] abordou a evolução dos recursos de linguagem para multiprogramação, destacando a importância de compreender programas concorrentes em termos independentes de tempo e verificar automaticamente as suposições sobre as relações invariantes entre os componentes do programa. O autor discute a necessidade de estruturar programas de forma hierárquica e utilizar construções de lin-

guagem adequadas para a multiprogramação, enfatizando a importância da abstração e da clareza na descrição dos programas. São mencionados conceitos como eventos de sincronização, semáforos, regiões críticas e monitores como ferramentas para lidar com a concorrência de forma eficiente e segura. O autor também aborda o uso de COBEGIN e COEND, novos nomes para o PARBEGIN e o PAREND definidos originalmente por Dijkstra, como notações para delimitar blocos de código que serão executados concorrentemente em programas. O COBEGIN marca o início de um bloco de código concorrente (como acontece com o PARBEGIN), enquanto o COEND indica o fim desse bloco (como o PAREND). Essa técnica ajuda a garantir a correta sincronização e comunicação entre processos concorrentes, ocultando os detalhes de implementação da concorrência do programador.

A multiprogramação estruturada, de Per Brinch Hansen [HANSEN, 1973], contém os hipergrafos de Dijkstra, apresentados nessa dissertação, dependendo da situação. A multiprogramação estruturada é uma extensão da programação estruturada para sistemas operacionais multiusuários e multitarefa. Com isso, no cenário de multitarefas não é necessário esperar uma tarefa terminar para seguir a execução de uma outra. Por exemplo, quando são utilizados um programa de planilhas e um navegador, que são tarefas independentes, não é necessário encerrar um dos dois para continuar usando o outro. Na programação estruturada de Dijkstra et al. [DAHL; DIJKSTRA; HOARE, 1972], para executar o próximo comando, é necessário esperar todos os comandos que estão executando terminarem.

Porém, pode ocorrer de uma tarefa precisar da resposta de diversas outras, que estão executando em paralelo, para que possa ser executada. Nessa situação, é possível modela-la utilizando hipergrafo de Dijkstra, apesar da programação estruturada de Dijkstra estar mais relacionada à estrutura interna dos programas individuais, enquanto a multiprogramação estruturada de Hansen lida com a estrutura e controle de múltiplos programas em um ambiente de sistema operacional multitarefa.

Charles Antony Richard Hoare [HOARE, 1978] apresenta o modelo de programação concorrente conhecido como *Communicating Sequential Processes* (CSP), que se tornou uma importante ferramenta no estudo da concorrência e da comunicação entre processos. O CSP de Hoare é uma linguagem de modelagem matemática que descreve sistemas concorrentes em termos de processos que se comunicam por meio de canais de comunicação. Ele introduz a ideia de processos independentes que executam em paralelo e se comunicam exclusivamente através da troca de mensagens. Principais conceitos abordados são: processos (unidades de execução independentes que interagem entre si por meio de comunicação síncrona através de canais), comunicação por canais (meio de comunicação unidirecionais e sincronizados entre os processos), paralelismo explícito (permitindo que os programadores expressem claramente a concorrência entre processos), composição de processos (para construir sistemas mais complexos a partir de processos mais simples)

e concorrência e comunicação. O modelo CSP distingue claramente entre concorrência, execução simultânea de processos independentes, comunicação e troca de mensagens entre processos.

Durante a década de 1960, em conjunto aos esforços relacionados à programação estruturada, matemáticos e pesquisadores começaram a explorar conceitos além dos grafos tradicionais, o que culminou no desenvolvimento da teoria dos hipergrafos. Claude Berge foi um dos principais contribuintes nessa área, sendo responsável pela publicação do livro "Graphs and Hypergraphs" [BERGE, 1970], no qual são definidos os conceitos básicos da teoria dos hipergrafos, propriedades estruturais dos hipergrafos, problemas clássicos (como cobertura e coloração), além de suas aplicações em diversas áreas, como matemática discreta, teoria dos grafos e ciência da computação.

Hipergrafos são uma generalização natural de grafos, de modo que os problemas mais comuns em grafos, tais como, coloração, planaridade, conexidade, caminho, podem naturalmente ser generalizadas para hipergrafos. Hipergrafos podem ser usados para uma variedade de situações onde as relações entre elementos não são necessariamente binárias (como nos grafos), mas podem envolver conjuntos de elementos [BRETTO, 2013]. Essa generalização tem sido aplicada em várias disciplinas, incluindo Redes Sociais, Banco de Dados, Processamento de Linguagem Natural, Biologia, Aprendizagem, Economia Ecológica e a Sociologia, e modelagem de correlações complexas [GAO et al., 2020].

A partição de hipergrafos é um problema fundamental em diversas áreas, incluindo a computação paralela. Consiste em dividir um hipergrafo em partes, buscando minimizar a função de custo dos hiper-arcos que conectam os vértices em diferentes conjuntos. Em muitos casos, essa definição é muito restritiva e requer mais do que duas partes [GAO et al., 2020]. A partição ajuda a saber qual o número máximo de processos para resolver determinado problema, com isso, o número de vértices internos do hipergrafo que indicarão a quantidade de processos em paralelo, no caso do *cluster*, quantos computadores serão usados, de forma paralela, para executar um programa.

Na programação paralela, a divisão de um hipergrafo em partes é semelhante à divisão de tarefas complexas em várias partes menores para serem processadas em paralelo por diferentes unidades de processamento, enquanto a minimização da função de custo das hiper-arcos pode ser interpretado como um problema de otimização, onde se busca minimizar uma função de custo, algo comum em algoritmos paralelos. A determinação do número máximo de processos para resolver um problema está diretamente relacionado à capacidade de paralelismo do problema. Na computação paralela, é importante determinar quantos processos podem ser executados em paralelo para otimizar o desempenho, ao mesmo tempo que o uso de métodos de particionamento de hipergrafos e *clustering* pode ser paralelizada para lidar com grandes volumes de dados e acelerar o processo de particionamento. Por último, as estratégias multinível são comumente usadas para melhorar o desempenho e a escalabilidade de algoritmos paralelos, especialmente em problemas de

grande escala.

Há diversos trabalhos que mostram o uso de métodos de particionamento de hipergrafos e definem *clustering* como "o processo de mesclar vértices em grupos maiores de vértices conhecidos como *clusters* para calcular um hipergrafo mais grosseiro a partir de um hipergrafo de entrada." São fornecidas várias aplicações de particionamento e *clustering*, incluindo álgebra linear numérica, prova automatizada de teoremas e verificação formal e são descritos na literatura. Estratégias multinível são frequentemente necessárias em *clustering* e particionamento, que têm sido bem estudadas em trabalhos anteriores [GAO et al., 2020].

O objetivo dessa dissertação é a caracterização e o reconhecimento de uma nova classe de hipergrafos denominada "hipergrafos de Dijkstra". Esses hipergrafos são uma extensão dos grafos de Dijkstra que possuem a estrutura paralela adicional do PARBEGIN/PAREND. Nós mostramos que essa classe de hipergrafos tem seu reconhecimento e isomorfismo desempenhados em tempo linear.

A estrutura do documento é organizada da seguinte forma: no Capítulo 1 apresentamos as definições utilizadas neste trabalho; no Capítulo 2 abordamos e apresentamos um algoritmo de reconhecimento linear para a classe dos hipergrafos de Dijkstra; no Capítulo 3 mostramos um algoritmo linear que verifica o isomorfismo entre hipergrafos de Dijkstra; no último capítulo, sumarizamos os resultados.

# 1 CONCEITOS BÁSICOS

Nessa seção introdutória, serão apresentadas as definições básicas que são essenciais para a compreensão do trabalho. Para as definições da teoria de grafos, foram utilizados os livros de Bondy e Murty [BONDY; MURTY, 2008], Gross e Yellen [GROSS; YELLEN, 2006] e Szwarcfiter [SZWARCFITER, 2018] e para as definições da teoria de hipergrafos foi utilizado o livro de Berge [BERGE, 1970]. Também serão discutidos alguns trabalhos relacionados com essa dissertação.

## 1.1 Grafos

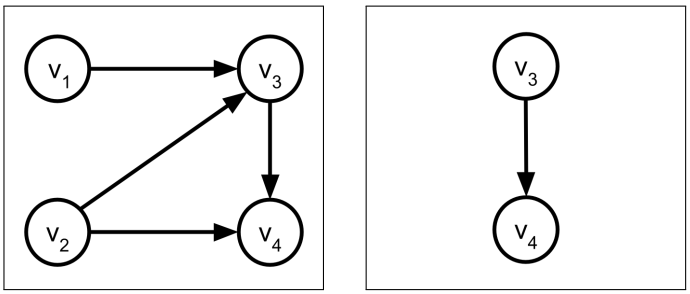
**Definição 1.1.** *Um **grafo trivial** possui apenas um vértice e nenhuma aresta.*

**Definição 1.2.** *Uma **aresta direcionada**, ou **arco**, $uv$ é um par ordenado de vértices $u, v$, onde $u$ é chamado de origem de $uv$ e $v$ é chamado de destino de $uv$.*

**Definição 1.3** ([**GUEDES, 2001**])**.** *Um **grafo direcionado** $D = (V, E)$ é um par onde $V$ é um conjunto finito de vértices e $E$ é um conjunto finito de arestas direcionadas (Figura 1a).*

**Definição 1.4.** *$H$ é um **subgrafo direcionado** de um grafo direcionado $G$ se $V(H) \subset V(G)$ e $E(H) \subset E(G)$ (Figura 1b).*

Figura 1 - Exemplo de grafo direcionado e um de seu subgrafos direcionados.




(a) (b)

Legenda: (a) grafo direcionado $G$; (b) Subgrafo direcionado $H$ de $G$.
Fonte: O autor, 2023.

**Definição 1.5.** *O conjunto de vizinhos, também chamado de **vizinhança**, de um vértice $v$ em um grafo $G$ é denotado por $N_G(v) = \{u \mid uv \in E\}$.*

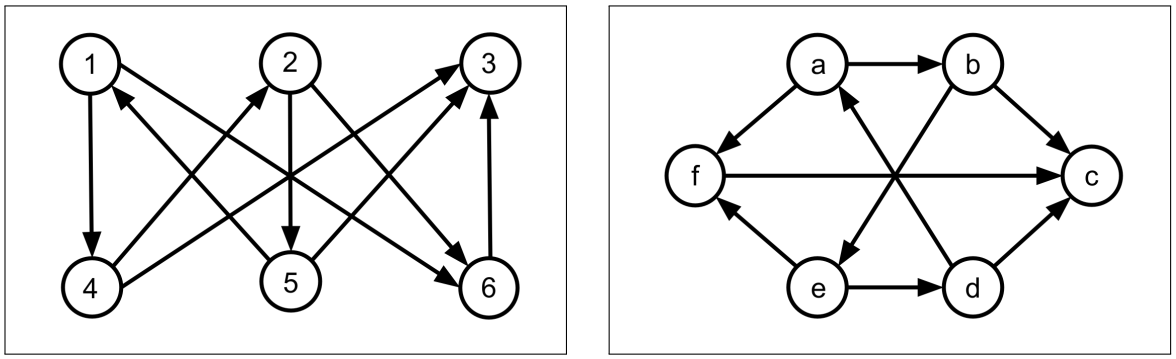
**Definição 1.6** ([**BONDY; MURTY, 2008**]). *Uma* ***função de incidência*** $\psi_G$ *que associa a cada aresta de* $G$ *um par não ordenado de vértices (não necessariamente distintos) de vértices de* $G$*.*

**Definição 1.7** ([**BONDY; MURTY, 2008**]). *Dois grafos,* $G$ *e* $H$ *são* ***isomorfos****, denotado por* $G \cong H$*, se existem bijeções* $\theta : V(G) \to V(H)$ *e* $\phi : E(G) \to E(H)$*, de tal forma que* $\psi_G(e) = uv$ *se, e somente se,* $\psi_H(\phi(e)) = \theta(u)\theta(v)$*, tal que um par de mapeamentos é chamado de isomorfismo entre* $G$ *e* $H$*.*

Para os grafos das Figuras 2a e 2b, existem dois mapeamentos, $\theta$ e $\theta'$ mostrados abaixo, que mostram o isomorfismo entre $G$ e $H$.

$$\theta = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ a & e & c & b & d & f \end{pmatrix} \qquad \theta' = \begin{pmatrix} 1 & 2 & 3 & 4 & 5 & 6 \\ e & c & a & f & b & d \end{pmatrix}$$

Figura 2 - Exemplo de isomorfismo.

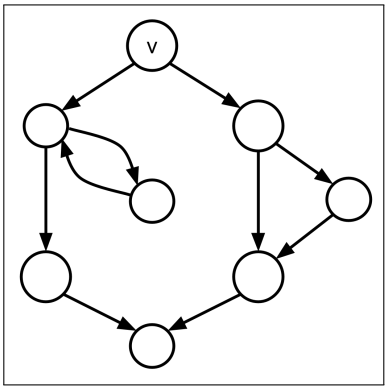



(a) (b)

Legenda: (a) Grafo G; (b) Grafo H.
Fonte: BONDY; MURTY, 2008.

**Definição 1.8** ([**GUEDES, 2001**]). *Um* ***grafo de fluxo*** *é uma tripla* $G = (V, E, s)$*, onde* $(V, E)$ *é um grafo direcionado,* $s \in V$ *é o vértice origem, e existe um caminho de* $s$ *para qualquer outro vértice em* $V$*.*

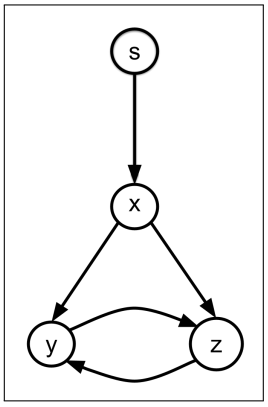
Figura 3 - Grafo direcionado de fluxo de controle.




Fonte: O autor, 2024.

**Definição 1.9** ([**HECHT; ULLMAN, 1972; HECHT; ULLMAN, 1974**]). *Grafos de fluxo são chamados de* ***grafos redutíveis*** *se, e somente se, não possuem loops e seus ciclos possuem uma única entrada.*

Figura 4 - Exemplo de grafo direcionado de fluxo **não** redutível. Veja que no ciclo $(y, z, y)$ temos uma entrada por $y$ e uma entrada por $z$.




Fonte: GUEDES, 2001.

**Definição 1.10** ([**BENTO et al., 2019**]). *Para um grafo direcionado $G$, $v, w \in V(G)$, uma aresta de $v$ para $w$ é escrita como $vw$. Dizemos que $vw$ é uma aresta de saída de $v$ e uma aresta de entrada de $w$, sendo $w$ um vizinho de saída de $v$, e $v$ um vizinho de entrada de $w$. Denotamos por $N_G^+(v)$ e $N_G^-(v)$ os conjuntos de vizinhos de saída e vizinhos de entrada de $v$, respectivamente.*

**Definição 1.11** ([**BENTO et al., 2019**]). *Um* ***grafo direcionado fonte-sumidouro*** *$G$ possui dois vértices especiais: um vértice fonte $\boldsymbol{s(G)}$ e um vértice sumidouro $\boldsymbol{t(G)}$ distintos. Onde existe um caminho a partir de $\boldsymbol{s(G)}$ para todos os outros vértices e existe um caminho de todos os vértices, com exceção do sumidouro, até o sumidouro $\boldsymbol{t(G)}$, enquanto não existe um caminho a partir de $\boldsymbol{t(G)}$ e nenhum vértice de $G$.*

Tabela 1 - Vizinhanças de entrada e de saída dos vértices do grafo direcionado da Figura 1a.

| $v$ | $N_G^+(v)$ | $N_G^-(v)$ |
|---|---|---|
| $v_1$ | $\{v_3\}$ | $\emptyset$ |
| $v_2$ | $\{v_3, v_4\}$ | $\emptyset$ |
| $v_3$ | $\{v_4\}$ | $\{v_1, v_2\}$ |
| $v_4$ | $\emptyset$ | $\{v_2, v_3\}$ |

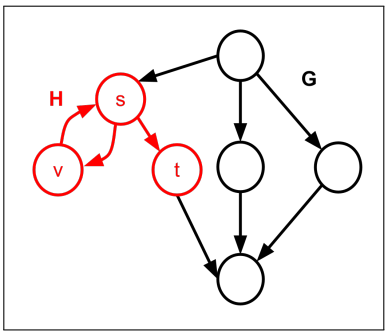
Fonte: O autor, 2023.

**Definição 1.12** ([BENTO et al., 2019]). *Em uma busca em profundidade de um grafo direcionado $G$, em cada passo um vértice $\boldsymbol{v}$, que está sendo avaliado naquele passo, é inserido ou removido de uma pilha, onde cada vértice é inserido e removido do topo da pilha exatamente uma vez. Uma aresta $vw \in E(G)$, tal que $v$ é inserido na pilha depois de $w$, e antes da remoção de $w$, é chamada de* ***aresta de ciclo****, ou* ***arco-ciclo****.*

**Definição 1.13** ([BENTO et al., 2019]). *Seja $G$ um grafo direcionado redutível, e um subgrafo $H$ de $G$ tendo fonte $s$ e sumidouro $t$. Dizemos que o $H$ é* ***fechado*** *quando*

- *se $v \in V(H)\backslash s$, então $N^-(v) \subseteq V(H)$;*
- *se $v \in V(H)\backslash t$, então $N^+(v) \subseteq V(H)$.*

Figura 5 - Exemplo de grafo fechado para mostrar as condições $v \in V(H)\backslash s$, $v \in V(H)\backslash t$ e $vs$ é uma arco-ciclo.




Fonte: O autor, 2023.

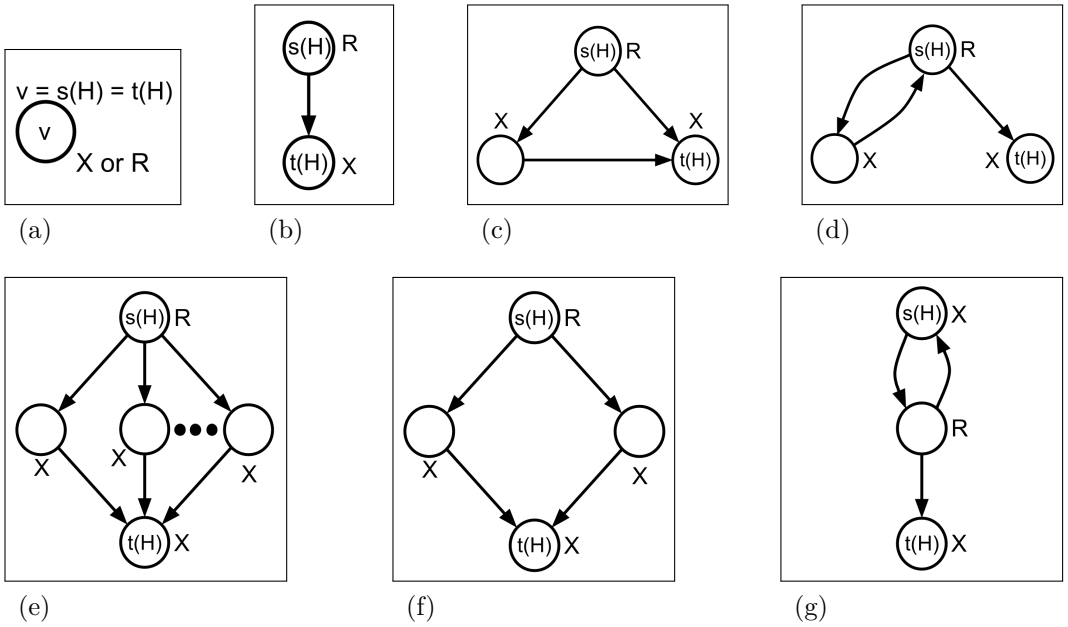
**Definição 1.14** ([BENTO et al., 2019]). *Dado $G, H$ dois grafos direcionados, onde $V(G) \cap V(H) = \emptyset$ e $H$ é um grafo fonte-sumidouro disjunto de $G$, $v \in V(G)$. A* ***Operação de Expansão*** *de $v$ em $H$, obtendo o grafo direcionado $G \uparrow H$, consiste na substituição de $v$ por $H$, em $G$, tal que $N^-_{G\uparrow H}(s(H)) = N^-_G(v)$, $N^+_{G\uparrow H}(t(H)) = N^+_G(v)$ e as adjacências remanescentes são preservadas. Onde os vértices expansíveis, que podem ser expandidos, são rotulados com $X$ e os vértices que já foram expandidos são rotulados com $R$, de regular, indicando que não podem ser expandidos novamente.*

**Definição 1.15** ([**BENTO et al., 2019**]). *Dado um grafo direcionado $G = (V, E)$ e um subgrafo direcionado fonte-sumidouro $H$ de $G$, onde $H$ é fechado. A* ***Operação de Contração*** *em um grafo direcionado consiste na união dos vértices de um subgrafo fonte-sumidouro $H$ em um único vértice, removendo todas as arestas paralelas e laços, gerando o grafo resultante $G \downarrow H$. Após a contração, onde $v$, em $G \downarrow H$, é o vértice $s(H)$ em $G$, $N^-_{G\downarrow H}(v) = N^-_G(s(H))$ e $N^+_{G\downarrow H}(v) = N^+_G(t(H))$, ou seja, a vizinhança de entrada de $v$ é igual a vizinhança de entrada da fonte $s(H)$ de $H$ e a vizinhança de saída de $v$ é igual a vizinhança de saída do sumidouro $t(H)$ de $H$.*

**Definição 1.16** ([**BENTO et al., 2019**]). *Um* ***grafo direcionado de declaração*** *é definido como sendo um dos sete grafo direcionados das Figura 6. Onde cada vértice é um vértice expansível, rotulado com* ***X****, ou um vértice regular, rotulado com* ***R****, mostrados nas Figura 6 com seus respectivos rótulos. Todos os grafos de declaração são do tipo fonte-sumidouro.*

Em um grafo direcionado $G$, que possui diversos subgrafo direcionados isomorfos a algum grafo direcionado de declaração, a fonte de cada subgrafo direcionado $H$ possui sua vizinhança de entrada $N^-_H(s(H))$.

Figura 6 - Grafos de Declaração.




Legenda: (a) Trivial; (b) Sequência; (c) *Se-Então*; (d) *Enquanto*; (e) *q-Caso*; (f) *Se-Então-Senão*; (g) *Repita*.

Fonte: BENTO et al., 2019.

Os Fatos 1 e 2 ajudam a entender a demonstração do Lema 2.2.

**Fato 1.** *O grafo direcionado de declaração Repita é o único grafo direcionado de declaração não trivial cujo vértice de entrada (fonte) é expansível.*

**Fato 2.** *Exceto no grafo direcionado de declaração trivial, todo vértice expansível $v$ tem grau de entrada positivo, $N^-(v) > 0$.*

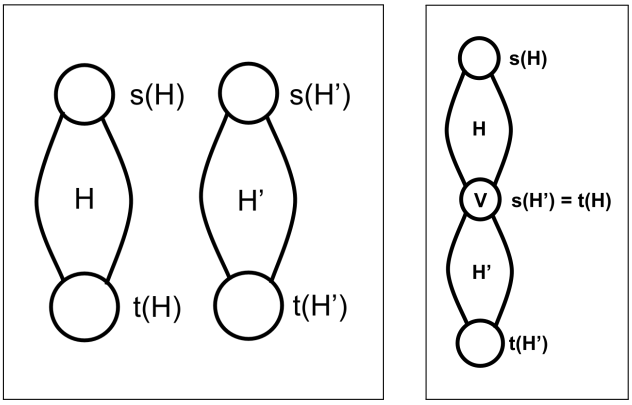
**Definição 1.17** ([**BENTO et al., 2019**])**.** *Um grafo direcionado $H$ é* ***primo*** *quando é fechado e isomorfo a algum grafo de declaração não trivial.*

A *coleção de subgrafos primos* não triviais de $G$ é denotada por $\mathcal{H}(G)$, notação que é usada durante todo o trabalho.

**Definição 1.18** ([**BENTO et al., 2019**])**.** *Sejam $H_1, H_2 \in \mathcal{H}(G)$. Chamamos $H_1, H_2$ de* ***independentes*** *quando*

- $V(H_1) \cap V(H_2) = \emptyset$ *(Figura 7a), ou*
- $V(H_1) \cap V(H_2) = \{v\}$*, onde $v = s(H_1) = t(H_2)$ ou $v = t(H_1) = s(H_2)$ (Figura 7b).*

Figura 7 - Condições para os grafos serem independentes.




(a) (b)

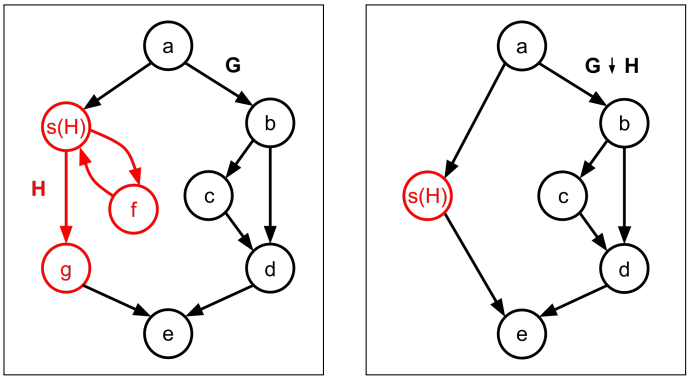
Legenda: (a) $V(H) \cap V(H') = \emptyset$; (b) $V(H) \cap V(H') = \{v\}$, onde $v = s(H) = t(H')$ ou $v = t(H) = s(H')$.
Fonte: O autor, 2023.

**Definição 1.19** ([**BENTO et al., 2019**])**.** *Para $v \in V(G)$, a* ***Imagem*** *de $v$ em $G \downarrow H$, denotado por $I_{G\downarrow H}(v)$, é*

$$I_{G\downarrow H}(v) = \begin{cases} v, & v \notin V(H) \\ s(H), & \textit{caso contrário.} \end{cases} \tag{1}$$

No exemplo da Figura 8, a $I_{G\downarrow H}(a)$ é o próprio vértice $a$, já que o vértice $a$ não pertence ao grafo direcionado $H$. Quando $v \in V(H)$ a imagem de $v$ será a fonte de $H$, $I_{G\downarrow H}(v) = s(H)$.

Figura 8 - Exemplo de $I_{G\downarrow H}(v)$.




(a) (b)

Legenda: (a) grafo direcionado $G$; (b) $I_{G\downarrow H}$.
Fonte: O autor, 2023.

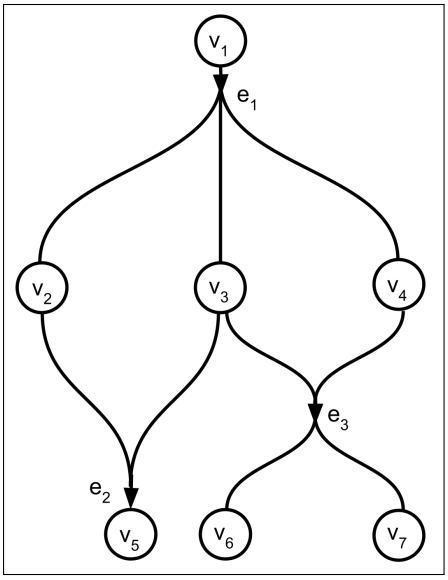
**Definição 1.20.** ***[BENTO et al., 2019]*** *Um grafo de Dijkstra é aquele cujos vértices são rotulados como $X$ ou $R$, definidos recursivamente como:*

- *Um grafo trivial é um grafo de Dijkstra;*
- *Qualquer grafo obtido de um grafo de Dijkstra ao expandir algum vértice rotulado com $X$ em um grafo de declaração (Figura 6) também é um grafo de Dijkstra.*

**Definição 1.21** ([GUEDES, 2001]). *Um* ***hipergrafo direcionado*** $H = (V, E)$ *onde $V$ é um conjunto de vértices e $E$, o conjunto de hiper-arcos, onde os hiper-arcos são pares de conjuntos disjuntos de vértices, $(X, Y)$.*

**Definição 1.22** ([GUEDES, 2001]). *Um* ***hiper-arco*** $\boldsymbol{e} \in E$ *é um par $e = (X, Y)$ onde $X$ e $Y$ são subconjuntos disjuntos de $V$, onde o subconjunto $X = Orig(e)$ é a origem do hiper-arco* $\boldsymbol{e}$ *e $Y = Dest(e)$ é o destino do hiper-arco* $\boldsymbol{e}$.

Figura 9 - Exemplo de hipergrafo direcionado $H = (V(G), E(G))$, onde $V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ e $E(G) = \{e_1, e_2, e_3\}$.




Fonte: O autor, 2023.

Na Figura 9, podemos ver um exemplo de hipergrafo $G = (V(G), E(G))$, onde $V(G) = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7\}$ e $E(G) = \{e_1, e_2, e_3\}$. Os hiper-arcos são definidos como $e_1 = (\{v_1\}, \{v_2, v_3, v_4\})$, $e_2 = (\{v_2, v_3\}, \{v_5\})$ e $e_3 = (\{v_3, v_4\}, \{v_6, v_7\})$.

**Definição 1.23** ([**GUEDES, 2001**]). *Dado um hipergrafo direcionado $H = (V, E)$, e dois vértices $s$ e $t$, um **hiper-caminho C** de $s$ a $t$, de tamanho $k$, é uma sequência de hiper-arcos $C = (e_{i_1}, e_{i_2}, \ldots, e_{i_{p-1}})$ onde $s \in Org(e_{i_1})$ e $t \in Dest(e_{i_k})$, e para cada hiper-arco $e_{i_p}$ de $C$, com $1 \leq p \leq k$, temos que:*

- $Org(e_{i_p}) \cap (Dest(\{e_{i_1}, e_{i_2}, \ldots, e_{i_{p-1}}\}) \cup \{s\}) \neq \emptyset$;
- $Dest(e_{i_p}) \cap (Org(\{e_{i_{p+1}}, e_{i_{p+2}}, \ldots, e_{i_k}\}) \cup \{t\}) \neq \emptyset$.

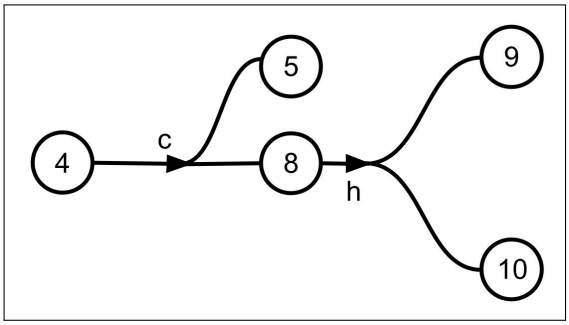
**Definição 1.24** ([**GUEDES, 2001**]). *Dado um hipergrafo direcionado $H$ e dois vértices $s$ e $t$, um $B$-caminho é um hiper-caminho $C$, de $s$ a $t$, de tamanho $k$, que tem a propriedade **B**: se, para cada hiper-arco $e_{i_p}$ de $C$, com $1 \leq p \leq k$, temos que*

$$Org(e_{i_p}) \subseteq (Dest(\{e_{i_1}, e_{i_2}, \ldots, e_{i_{p-1}}\}) \cup \{s\}).$$

Dizemos que um vértice $y$ está $B$-conectado ao vértice $x$ se existe um $B$-caminho de $x$ a $y$. Podemos ver na Figura 10 o $B$-caminho $(c, h)$ de 4 a 10. Observe que o vértice destino 10 não está $B$-conectado ao vértice 5 [GUEDES, 2001].

**Definição 1.25** ([**GUEDES, 2001**]). *Dado um hipergrafo direcionado $H = (V, E)$ e um vértice $v \in V$ chamamos de **hiper-arcos de entrada** e **hiper-arcos de saída**, os hiper-arcos dos conjuntos $BS(v)$ (**Backward Star**) e $FS(v)$ (**Forward Star**), respectivamente, definidos como:*

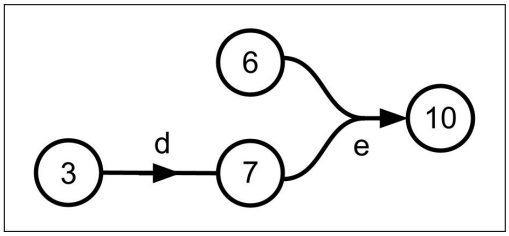
Figura 10 - Exemplo de *B-caminho*.




Fonte: GUEDES, 2001.

- $BS(v) = \{e | v \in Dest(e)\}$;
- $FS(v) = \{e | v \in Org(e)\}$.

Figura 11 - Exemplo de $BS(v)$ e $FS(v)$.




Fonte: GUEDES, 2001.

Na Figura 11, o $BS(v)$, para $v = 7$, é a aresta $d$ e $FS(v)$, para $v = 3$, é a aresta $d$.

**Definição 1.26** ([**GUEDES, 2001**]). *O* ***hipergrafo direcionado de fluxo*** *é uma tripla $H = (V, E, s)$, onde $(V, E)$ é um hipergrafo direcionado, $s \in V$ é o vértice de origem e existe um $B$-caminho de $s$ para qualquer outro vértice em $V$.*

Vale ressaltar que o vértice $s$, na Definição 1.26, equivale ao vértice fonte $s(G)$ em um grafo fonte-sumidouro $G$.

**Definição 1.27** ([**GUEDES, 2001**]). *Dado um programa paralelo $P$. O* ***hipergrafo de controle*** *$H_p = (V, E, s, t)$ de $P$ satisfaz que $(V, E, s)$ é um hipergrafo de fluxo e que $t \in V$ é um vértice onde:*

1. *cada instrução (ou conjunto de instruções elementares) de $P$ é um vértice de $V$;*
2. *cada dependência de execução entre instruções de $P$ é um hiper-arco de $E$;*
3. *$s$ é o vértice inicial de $P$ e $t$, o vértice final de $P$.*

**Teorema 1.1** ([**GUEDES, 2001**]). *Se $H$ é um* ***hipergrafo direcionado redutível****, então todo ciclo em $H$ tem uma única entrada ([GUEDES, 2001]-Teorema 7.10).*

## 1.2 Busca em Profundidade em grafos direcionados

A realização de buscas em um grafo ocorre de maneira sistemática, percorrendo suas arestas e visitando todos os vértices do grafo. Sua execução desempenha um papel crucial na obtenção de informações sobre a estrutura do grafo. Nessa seção, desenvolvemos a ideia da Busca em Profundidade primeiro em grafos e depois nos hipergrafos.

**Definição 1.28** ([**SZWARCFITER, 2018**])**.** *Dado um grafo direcionado $G = (V, E)$. Uma* ***busca*** *é dita* ***em profundidade*** *em $G$ quando o critério de escolha de vértice a ser analisado (a partir do qual será realizada a próxima exploração de aresta) obedecer a regra "dentre todos os vértices analisados e incidentes a alguma aresta ainda não explorada, escolher aquele mais recentemente alcançado na busca".*

O Algoritmo 1 divide o conjunto de arestas $E$ de $G$ em duas partes disjuntas: os *arcos visitados* em (I) denominados *arcos de árvore* $E_T$ e aqueles em (II), chamados *arcos-ciclo*. No algoritmo há necessidade de introduzir um parâmetro adicional $u$, além de um parâmetro adicional $v$, nas chamadas recursivas para evitar que um arco seja visitado mais de duas vezes [SZWARCFITER, 2018].

**Algoritmo 1:** Busca em Profundidade $P(v, u)$

```
Entrada: Grafo direcionado G = (V, E) e v ∈ V
Saída: Ordem do percurso.
1  Função P(vértice: v, vértice: u)
2      marcar v
3      para w ∈ N⁺(v) faça
4          se w é não marcado então
5              visitar (v, w)                  >arcos de árvore(I)
6              P(w, v)
7          senão
8              se w ≠ u então
9                  visitar (v, w)              >arcos de retorno(II)
10             fim
11         fim
12     fim
13     desmarcar todos os vértices
14     escolher uma raiz s
15     P(s, ∅)
16 end
```

A escolha de vértice marcado torna-se única e sem ambiguidade, segundo o critério apresentado. O Algoritmo 1, implementado de forma recursiva, reflete esse processo. Tanto a escolha da raiz de busca quanto do arco $(v, w)$ a ser explorada a partir do vértice

marcado $v$ são, por natureza, arbitrárias. A decisão do arco $(v, w)$ é implicitamente derivada da ordenação de $N^+(v)$, que é uma ordem arbitrária estabelecida [SZWARCFITER, 2018].

Dado um grafo direcionado $D(V, E)$ de fonte $s$. Ao considerar uma busca em profundidade, com fonte $s$, onde se chega em qualquer outro vértice a partir de $s$, executada em $D$, é obtida uma árvore de profundidade de fonte $s$ onde $E_T$ é o conjunto dos arcos de árvore. Então pode-se mostrar que $(V, E_T)$ é uma árvore direcionada enraizada geradora do grafo direcionado $D$ [SZWARCFITER, 2018].

**Definição 1.29** ([**SZWARCFITER, 2018**])**.** *Todo grafo direcionado acíclico $D(V, E)$ induz um conjunto parcialmente ordenado $(V, <)$, definido por*

$$v < w \text{ se, e somente se, } v \text{ chega em } w \text{ em } D, \forall v, w \in V, v \neq w.$$

*Com isso, é possível ordenar os vértices do grafo direcionado de modo a obter uma sequência $v_1, v_2, \ldots, v_n$, denominada* ***Ordenação Topológica****, satisfazendo que*

$$\text{se } v_i < v_j, \text{ então } i < j, \text{ para } 1 \leq i < j \leq n.$$

De maneira geral, a ordenação topológica ($OT$) não é única. No Algoritmo 2 [KNUTH, 1997] este fato se reflete na possibilidade de escolha do vértice $w$, dentre aqueles que possuem grau de entrada nulo.

**Definição 1.30** ([**SZWARCFITER, 2018**])**.** *Seja $D(V, E)$ um grafo direcionado e um vértice $v \in V$. O grau de entrada de $v$ é $N_D^-(v)$ e o grau de saída de $v$ é $N_D^+(v)$.*

A busca em profundidade é, como seu nome implica, uma busca mais profunda no grafo, sempre que possível. A busca em profundidade explora arestas partindo do vértice $v$ mais recentemente descoberto do qual ainda saem arestas inexploradas. Depois que todas as arestas de $v$ foram exploradas, a busca regressa pelo mesmo caminho para explorar as arestas que partem do vértice do qual $v$ foi descoberto. Esse processo continua até descobrirmos todos os vértices que podem ser visitados a partir do vértice fonte inicial. Se restarem quaisquer vértices não descobertos, a busca em profundidade seleciona um deles como fonte e repete a busca partindo dessa fonte. O algoritmo repete esse processo inteiro até descobrir todos os vértices. Essa busca, para identificar os vértices que já foram visitados e que ainda tem arestas que não foram analisadas e os vértices que todas as arestas já foram analisadas, pinta os vértices durante a busca para indicar o estado de cada um. Cada vértice é inicialmente branco, pintado de cinza quando descoberto na busca e pintado de preto quando terminado, isto é, quando sua lista de adjacências já foi totalmente examinada. Essa técnica garante que cada vértice acabe em exatamente uma árvore, de forma que essas árvores são disjuntas [CORMEN et al., 2012].

Além de criar uma árvore, ou um conjunto de árvores (floresta), a busca em profundidade também identifica cada vértice com um carimbo de tempo (Figura **??**). Cada vértice $v$ tem um carimbo de tempo $v.d$ que registra quando $v$ é descoberto pela primeira vez (e pintado de cinzento), e outro carimbo de tempo $v.f$ que registra quando a busca termina de examinar a vizinhança de $v$ (e pinta $v$ de preto) [CORMEN et al., 2012].

O procedimento da busca em profundidade a seguir registra no atributo $u.d$ o momento em que descobre o vértice $u$ e registra no atributo $u.f$ o momento em que liquida o vértice $u$. Para todo vértice $u$, $u.d < u.f$ [CORMEN et al., 2012].

O Algoritmo 2, baseado no algoritmo de busca em profundidade apresentado por Cormen et al. [CORMEN et al., 2012], segue a ideia apresentada nos parágrafos anteriores, porém adicionando os vértices a uma lista $L$ após o retorno das chamadas recursivas na linha 17. No final da execução desse algoritmo, a lista $L$ está preenchida com uma ordenação topológica para grafo de entrada.

**Algoritmo 2:** Busca em profundidade e Ordenação Topológica

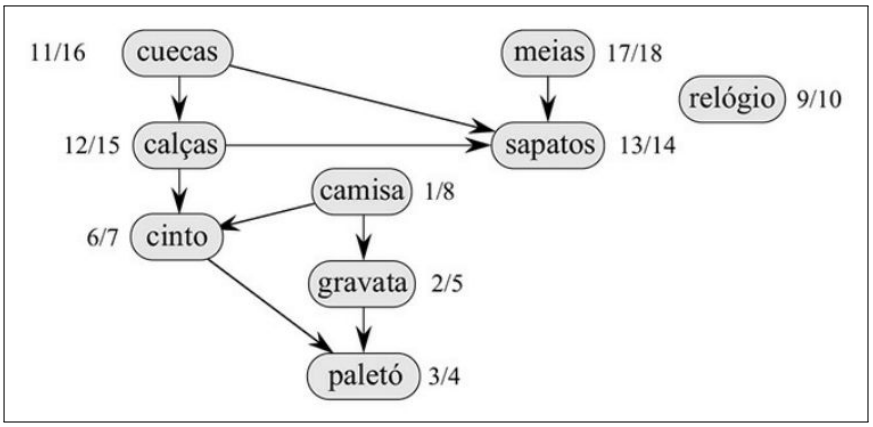
**Entrada:** Grafo direcionado acíclico $D(V, E)$.
**Saída:** Ordenação topológica.

```
1  início
2      L ← lista vazia que terá os vértices ordenados
3      enquanto ∃ vértices não marcados faça
4          Seleciona um vértice v
5          visita(v)
6      fim
7  fim
8  Função visita(v: vértice) : vértice
9      se v é marcado então
10         retornar
11     fim
12     se v tem marca temporária então
13         pára, porque o grafo possui, pelo menos, um ciclo
14     fim
15     Marcar v com marcação temporária
16     para cada vértice u com uma aresta de v até u faça
17         visita(u)
18     fim
19     remover marca temporária de v
20     marcar v com marca permanente
21     adiciona v no inicio da lista L
22 fim
```

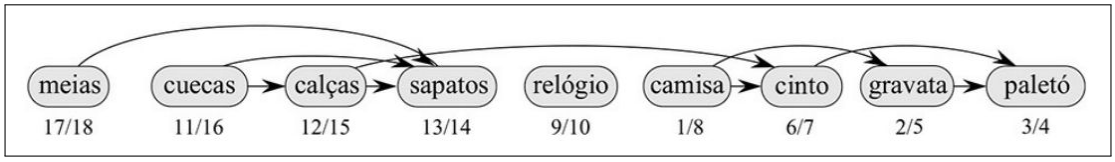
A Figura 12a [CORMEN et al., 2012] mostra um grafo direcionado, onde cada

vértice é é uma peça de roupa e a Figura 12b mostra uma possível a ordem para vestir cada peça de roupa.

Figura 12 - Grafo direcionado Acíclico e sua $OT$.




(a)




(b)

Legenda: (a) Grafo acíclico direcionado onde cada vértice é uma peça de roupa; (b) Ordenação topológica mostrando a ordem que cada peça é vestida.
Fonte: CORMEN et al., 2012.

## 1.3 **Ordenação topológica em hipergrafos direcionados**

Os conceitos e algoritmos descritos nessa seção foram extraídos da Tese de Guedes[GUEDES, 2001].

Guedes [GUEDES, 2001], em sua teste de doutorado, trata de hipergrafos direcionados, que são uma generalização de grafos direcionados. O autor apresenta diversas definições para os hipergrafos direcionados como os hipergrafos redutíveis, sendo uma generalização dos grafos redutíveis, onde, nessa generalização, partiu das ideias originais de redutibilidade de Hecht e Ulmann [HECHT; ULLMAN, 1972; HECHT; ULLMAN, 1974]. Também mostra formas de percorrer os hipergrafos mostrando definições de alguns caminhos, usaremos a definição de $B$-caminho.

No Algoritmo 3, o rótulo $P_v$, indica se o vértice $v$ já foi visitado, e qual hiper-arco foi usado para se chegar a ele[1]. E o rótulo $K_e$ indica quantos vértices da origem do hiper-arco $e$ já foram visitados, assim, o hiper-arco $e$ só será explorado quando $K_e = |Org(e)|$.

[1] Com $P_v$ podemos recuperar os $B$-caminhos percorridos.

Observe que $P_v = \emptyset$ quando $v$ não foi visitado, indicando que nenhum hiper-arco foi usado até agora para chegarmos em $v$. O símbolo $\lambda$ representa um hiper-arco fictício, que deve ser distinto do valor inicial ($\emptyset$) [GUEDES, 2001].

**Algoritmo 3:** B-Visita

**Entrada:** Hipergrafo $H(V, E)$ e $s \in V$.
**Saída:** Ordem do percurso em $Q$.

```
1  início
2      para cada v ∈ V faça
3          P_v ← ∅
4      fim
5      para cada e ∈ E faça
6          K_e ← 0
7      fim
8      P_s ← λ
9      Q ← {s}
10     repita
11         Escolha e Remova x ∈ Q
12         para cada e ∈ FS(x) faça
13             K_e ← K_e + 1
14             se K_e = |Org(e)| então
15                 para cada w ∈ Dest(e) tal que P_w = ∅ faça
16                     P_w ← e
17                     insira w em Q
18                 fim
19             fim
20         fim
21     até Q = ∅;
22 fim
```

## 1.4 Trabalhos relacionados

Dijkstra [DIJKSTRA, 1968a] não concordava com a utilização do uso da estrutura de GO TO no desenvolvimento de programas de computador. Com isso, Dijkstra et al. [DAHL; DIJKSTRA; HOARE, 1972] falam da facilidade ao desenvolver um programa de computador usando programação estruturada com blocos condicionais, *looping* entre outras estruturas, mostrando os fluxos dessas estruturas, que facilitam a leitura e o desenvolvimento desses programas além da facilidade na correção de erros.

Hecht e Ullman em seu artigo *Flow Graph Reducibility* [HECHT; ULLMAN, 1972]

fornecem uma análise abrangente dos grafos direcionados de fluxo e da sua redutibilidade. Os autores começam por introduzir os conceitos necessários da teoria dos grafos e depois definem redutibilidade. Em seguida, fornecem uma caraterização estrutural de grafos de fluxo não redutíveis e usam essa caraterização para obter um resultado interessante sobre grafos de fluxo para programas estruturados sem a estrutura de controle GO TO.

Bento et al. [BENTO et al., 2019] exploram o conceito de programação estruturada, introduzindo a classe de grafos formada pelos grafos de fluxo dos programas estruturados, chamada de Grafos de Dijkstra, e apresentam um algoritmo guloso de complexidade $O(n)$ para reconhecer esses grafos, onde essa classe possui grafos obtidos a partir das estruturas apresentadas no trabalho de Dijkstra et al. [DAHL; DIJKSTRA; HOARE, 1972], mostradas na Figura 6 com exceção de um deles. Além disso, descrevem um algoritmo de isomorfismo para essa classe de grafos, cuja complexidade é também linear no número $n$ de vértices do grafo. As ideias básicas da programação estruturada são discutidas com referências a trabalhos clássicos de Dijkstra et al. [DAHL; DIJKSTRA; HOARE, 1972] já citado, Dijkstra [DIJKSTRA, 1968a] e outros.

# 2 CARACTERIZAÇÃO DOS HIPERGRAFOS DE DIJKSTRA

A teoria de hipergrafos, com sua capacidade de modelar relações mais complexas, encontra uma gama diversificada de aplicações em diversas áreas, não se limitando apenas à esfera das ciências exatas. Neste capítulo, exploramos essa versatilidade ao desenvolver uma modelagem para programas executados de forma paralela estruturada, empregando os conceitos da teoria de hipergrafos e estendendo o trabalho de Guedes e Markenzon [GUEDES, 2001] e outros.

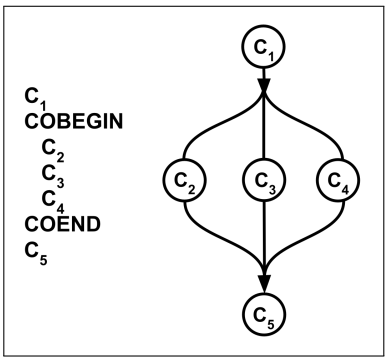
A Figura 13 mostra como seria um hipergrafo direcionado modelando o fluxo de controle de um trecho de programa. Nesta figura, os vértices $C_1, C_2, C_3, C_4$ e $C_5$ representam, respectivamente, os comandos $C_1, C_2, C_3, C_4$ e $C_5$ e os hiper-arcos $a$ e $b$ representam as relações de dependência entre os comandos [GUEDES, 2001].

## 2.1 PARBEGIN, PAREND e o grafo direcionado de declaração $p$-paralelo

Nessa seção vamos definir o grafo direcionado de declaração $p$-paralelo. Para isso nós consideramos um inteiro $p \geq 2$ e a sequência: $(S_0, PARBEGIN, S_1, \ldots, S_p, PAREND, S_{p+1})$, onde $S_0$ e $S_{p+1}$ são processos sequenciais e $S_1, \ldots, S_p$ são processos paralelos.

Em nosso modelo, serão executados inicialmente $S_0$ e depois do PARBEGIN, serão executados simultaneamente $S_1, \ldots, S_p$. Após o fim de todas as execuções paralelas (PAREND) será executado o processo $S_{p+1}$. Chamamos esse hipergrafo de declaração de *p-paralelo*, observando o uso da estrutura COBEGIN e COEND na Figura 13.

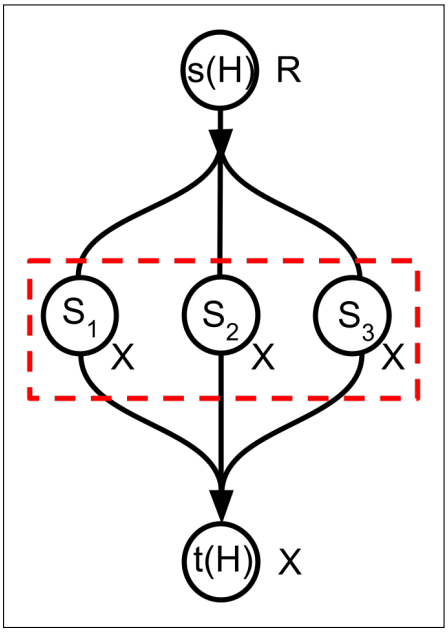
Figura 13 - Trecho de programa paralelo e seu hipergrafo de controle.




Fonte: GUEDES, 2001.

O hipergrafo de declaração $p$-paralelo é definido como hipergrafo de controle $H = (V, E, s, t)$, da Figura 14, o vértice $s$ é a fonte $s(H)$ e o vértice $t$, o sumidouro $t(H)$. A propriedade deste hipergrafo de declaração é ser um hipergrafo que representa um programa estruturado executando em paralelo com dois hiper-arcos.

Figura 14 - Grafo de declaração 3-Paralelo




Fonte: O autor, 2023.

**Definição 2.1.** *Um* ***hipergrafo de Dijkstra*** *possui seus vértices rotulados com $X$ ou $R$, recursivamente definido como:*

1. *Um hipergrafo de declaração trivial é um hipergrafo de Dijkstra;*
2. *Qualquer hipergrafo, obtido de um hipergrafo de Dijkstra pela expansão de um vértice rotulado com $X$ em algum hipergrafo de declaração da Figura 6, também é um hipergrafo de Dijkstra;*

**Lema 2.1.** *Se $G = (V, E)$ é um* ***hipergrafo de Dijkstra****, então*

1. *$G$ contém algum subgrafo primo;*
2. *$G$ é um grafo fonte-sumidouro;*
3. *$G$ é redutível;*

*Demonstração.* Por definição, existe uma sequência de grafos $G = \{G_i | i \in \{0, 1, \ldots, k\}\}$, onde $G_0$ é trivial, $G_k = G$ e $G_i$ é obtido de $G_{i-1}$ ao expandir algum vértice $v_{i-1} \in V(G_{i-1})$ rotulado com $X$ em um hipergrafo de declaração $H_i \subseteq G_i$. Então, nenhum vértice $v_i \in V(H_i)$, exceto $s(H_i)$, tem vizinhança de entrada fora de $H_i$, em $G_i$, e também nenhum vértice $v_i \in V(H_i)$, exceto $t(H_i)$, tem vizinhança de saída fora de $H_i$, em $G_i$. Além disso, se $H_i$ possui algum ciclo então $H_i$ é, necessariamente, um grafo *Enquanto* (Figura 6d) ou um grafo *Repetição* (Figura 6g). O grafo *Repetição* implica que tal ciclo é $s(H)v$, onde $v \in N^+(s(H))$. Portanto, $H_i$ é primo em $G_i$, o que significa que (1) vale. Para

mostrar (2) e (3), primeiro observe que qualquer hipergrafo de declaração é de fonte única e redutível. Em seguida, vamos aplicar indução em $i = 0, \ldots k-1$. Para $G_0$, não há nada a provar. Assuma que vale para $G_i, i > 1$. Seja $v_{i-1} \in V(G_{i-1})$ o vértice que se expandiu no subgrafo $H_i \subseteq G_i$. Então as vizinhanças externas de $H_i$ coincidem com as vizinhanças de $v_{i-1}$, respectivamente. Consequentemente, $G_i$ é de fonte única. Agora, seja $C_i$ qualquer ciclo de $G_i$, se existente. Se $C_i \cap H_i = \emptyset$, então $C_i$ é de entrada única, já que $G_{i-1}$ é redutível. Caso contrário, se $C_i \subset V(H_i)$ o mesmo é válido, pois qualquer hipergrafo de declaração é redutível. Finalmente, se $C_i \not\subset V(H_i)$ então $v_{i-1}$ está contido em um ciclo de entrada única $C_{i-1}$ de $G_{i-1}$. Então $C_i$ foi formado por $C_{i-1}$, substituindo $v_{i-1}$ por um caminho contido em $H_i$. Como $C_{i-1}$ é de entrada única, segue-se que $C_i$ deve ser assim. □

Considerando $H = (V, E)$ o *hipergrafo de declaração p-paralelo* (Figura 14), onde $V = \{s, s_i, t \mid i \in \{1, 2, \ldots, k\}\}$ e $E = \{(\{s\}, \{s_1, s_2, \ldots, s_p\}), (\{s_1, s_2, \ldots, s_p\}, \{t\})\}$. O vértice $s(H)$ é o vértice de entrada e o vértice $t(H)$ é o vértice de saída. Denominamos $s_1, s_2, \ldots, s_p$ os vértices de *execução paralela*, esses vértices estão marcados na Figura 14.

A execução de um programa em paralelo pode ser uma *execução síncrona* ou *execução assíncrona*. Na execução síncrona, o programa só irá continuar sua execução quando todas $p$ execuções em paralelo terminarem. Na execução assíncrona, o programa continua sua execução mesmo que as $N$ execuções em paralelo não tenham terminado. Vamos considerar nosso modelo síncrono. Portanto, no hipergrafo de declaração $p$-paralelo, mesmo que somente uma das $p$ execuções em paralelo tenha terminado, o programa irá esperar todas as $p$ execuções em paralelo terminarem para seguir sua execução.

No contexto de execução assíncrona, o uso de PARBEGIN/PAREND pode ser benéfico para coordenar a execução de processos paralelos. Essa abordagem é comumente empregada em programação concorrente para lidar com tarefas que podem ser executadas de forma independente e simultânea.

## 2.2 **Hipergrafos direcionados de declaração primos**

A seção aborda a expansão de um vértice rotulado com $t(H_1)$ ou $s(H_1)$ em um grafos de declaração $H_2$, e as condições sob as quais $H_1$ e $H_2$ são considerados primos. Ao explorar as características dos hipergrafos direcionados de declaração primos, buscamos estabelecer um arcabouço teórico sólido para a classificação e compreensão dessas estruturas, contribuindo para o avanço do conhecimento sobre hipergrafos de Dijkstra e sua aplicabilidade em contextos computacionais.

O Lema 4.2 de [BENTO et al., 2019] prova a independência de primos entre dois hipergrafos direcionados de declaração sequencial. No Lema 2.2 provamos a independência

de primos entre dois hipergrafos de declaração, quando um hipergrafos direcionado de declaração é sequencial e o outro é o hipergrafos direcionado de declaração $p$-paralelo ou quando os dois hipergrafos direcionados de declaração são $p$-paralelos.

**Lema 2.2** (**Independência de Primos**). *Se $H_1, H_2 \in \mathcal{H}(G)$ então $H_1$, $H_2$ são* ***independentes****.*

*Demonstração.* Suponha que $H_1$ e $H_2$ são grafos de declaração sequenciais. Neste caso, $H_1$ e $H_2$ são independentes devido ao resultado no Lema 4.2 de [BENTO et al., 2019].

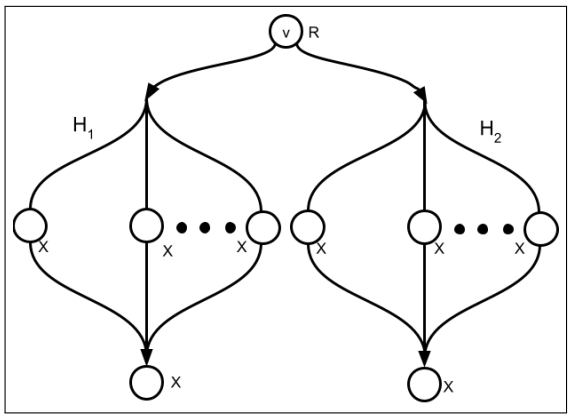
Assim, resta analisar os casos em que $H_1$ ou $H_2$ são grafos de declaração $p$-paralelo. Suponha que $V(H_1) \cap V(H_2) = \emptyset$. A partir do Lema 4.2 [BENTO et al., 2019], $H_1$ e $H_2$ são independentes.

Suponha que $V(H_1) \cap V(H_2) \neq \emptyset$. Seja $v \in V(H_1) \cap V(H_2)$. Vamos assumir que $H_1$ é obtido pela expansão de um vértice rotulado com $X$ antes de $H_2$ ser obtido. Temos os seguintes casos para analisar.

1. $H_1$ e $H_2$ são ambos hipergrafos de declaração $p$-paralelo

   (a) $v = s(H_1) = s(H_2)$. Como $s(H_1)$ é um vértice rotulado com $R$ e não poderia ser expandido de modo a também ser a fonte $s(H_2)$ de $H_2$, veja Figura 15. Uma contradição.

Figura 15 - $H_1$ e $H_2$ grafos de declaração $p$-paralelos com $v = s(H_1) = s(H_2)$.




Fonte: O autor, 2023.

   (b) $v = t(H_1) = t(H_2)$. Nesse caso, pela definição da operação de expansão, $H_2$ tem que ser obtido pela expansão do vértice $s(H_1)$. E portanto, $t(H_1) = s(H_2)$, uma contradição.

   (c) $v \neq s(H_1), t(H_1)$ ou $v \neq s(H_2), t(H_2)$

      i. $v \neq s(H_1), t(H_1)$ e $v = s(H_2)$: Observe que como $H_1$ é obtido pela expansão de um vértice antes de $H_2$ ser obtido, então $H_2$ seria obtido da expansão de um vértice interno de $H_1$ então $H_1$ não seria primo. Uma contradição.

ii. $v \neq s(H_1), t(H_1)$ e $v = t(H_2)$: Pelo mesmo argumento anterior, $H_2$ seria obtido da expansão de um vértice interno de $H_1$ e portanto $H_1$ não seria primo. Uma contradição.

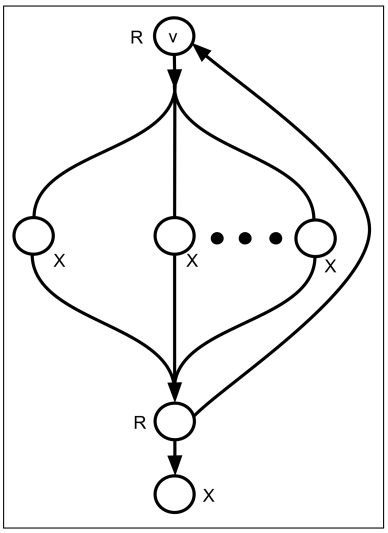
iii. $v \neq s(H_1), t(H_1)$ e $v \neq s(H_2), t(H_2)$: Igualmente, $H_2$ seria obtido da expansão de um vértice interno de $H_1$ e portanto $H_1$ não seria primo.

2. $H_1$ é um hipergrafo direcionado de declaração $p$-paralelo e $H_2$ é um grafo direcionado de declaração sequencial

(a) $v = s(H_1) = s(H_2)$**:**

i. $H_2$ é o hipergrafo de declaração *Repita* - Nesse caso consideramos as diversas possibilidades para $H_2$: Porque a fonte de um hipergrafo direcionado de declaração $H_1$ é um vértice do tipo $R$, precisamos expandir a fonte $s(H_2)$ do hipergrafo de declaração *Repita* para obter o hipergrafo de declaração $p$-paralelo $s(H_1)$. Assim, temos uma contradição, pois $H_2$ não seria um hipergrafo de declaração, como está ilustrado na Figura 16.

Figura 16 - $v = s(H_1) = s(H_2)$, sendo $H_2$ um grafo direcionado de declaração *Repita*.




Fonte: O autor, 2023.

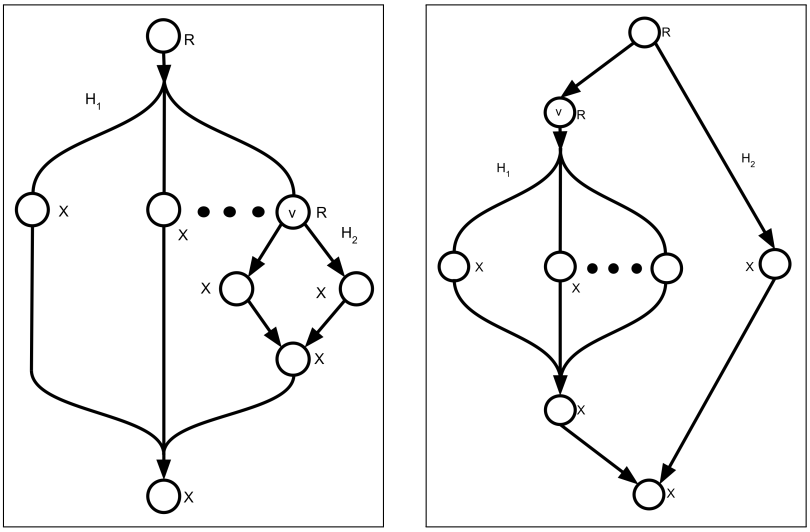
ii. $H_2$ não é o hipergrafo de declaração *Repita* - Nesse caso, ambos $s(H_1)$ e $s(H_2)$ são vértices rotulado com $R$. Portanto, não é possível expandir uma das fontes de um hipergrafo de declaração para obter o outro hipergrafo de declaração. Uma contradição.

(b) $v = t(H_1) = t(H_2)$**:** Nesse caso, temos 2 possibilidades. Ou $H_1$ foi obtido antes de $H_2$ ou depois. Pela definição da operação de expansão, teremos que $t(H_1) = s(H_2)$, ou que $t(H_2) = s(H_1)$. De ambos os casos temos uma contradição.

(c) $v \neq s(H_1), t(H_1)$ ou $v \neq s(H_2), t(H_2)$

i. $v \neq s(H_1), t(H_1)$ **e** $v = s(H_2)$**:** $|N^+_{H_1}(v)| = 1$. Como $v$ é um vértice interno de $H_1$ rotulado com $X$ e $s(H_2)$ é rotulado com $R$, sabemos que $H_2$ foi obtido pela expansão de um vértice interno de $H_1$. Dessa forma, pela definição da operação de expansão, temos que $H_1$ deixa de ser um hipergrafo de declaração, uma contradição (Figura 17).

Figura 17 - Expansão de um vértice rotulado com $X$ pela definição e $v \neq s(H_1), t(H_1)$ ou $v \neq s(H_2), t(H_2)$.

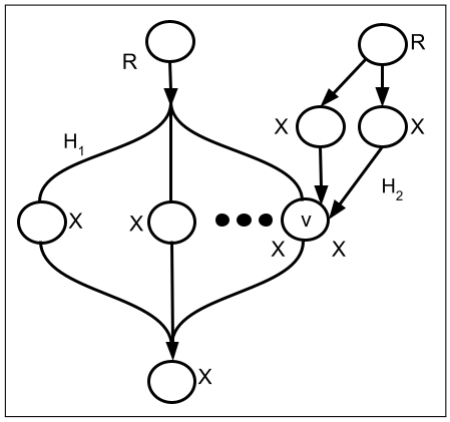



(a) (b)

Legenda: (a) $v \neq s(H_1), t(H_1)$; (b) $v \neq s(H_2), t(H_2)$.
Fonte: O autor, 2023.

ii. $v \neq s(H_1), t(H_1)$ **e** $v = t(H_2)$**:** Como $v$ é um vértice interno de $H_1$, $|N^-_G(v)| = |N^-_{H_1}(v)| = 1$, pela definição de grafo fechado. Porém, nesse caso, $|N^-_G(v)| = |N^-_{H_1}(v)| + |N^-_{H_2}(t(H_2))| \geq 1 + 1 = 2$ (Figura 18) pois em todo grafo direcionado de declaração o vértice $t(H_2)$ satisfaz a condição $|N^-_{H_2}(t(H_2))| \geq 1$. Uma contradição.

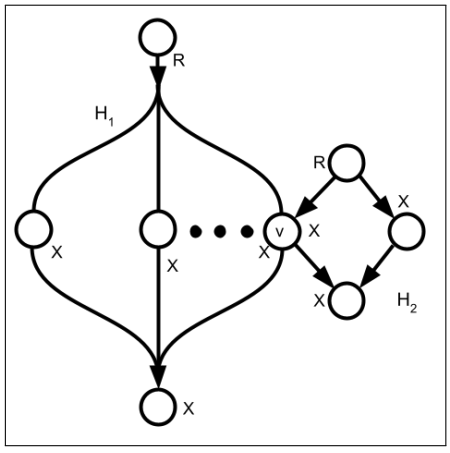
Figura 18 - $H_1$ hipergrafo de declaração $p$-paralelo e $H_2$ hipergrafo de declaração sequencial com $v \neq s(H_1), t(H_1)$ e $v = t(H_2)$. Neste exemplo usamos $H_2$ como o hipergrafo de declaração *Se-Então-Senão*.




Fonte: O autor, 2023.

iii. $v \neq s(H_1), t(H_1)$ **e** $v \neq s(H_2), t(H_2)$**:** Como $v$ é um vértice interno de $H_1$ e de $H_2$, e a expansão de um vértice interno de um hipergrafo de declaração não é um grafo direcionado de declaração (Figura 19), temos uma contradição.

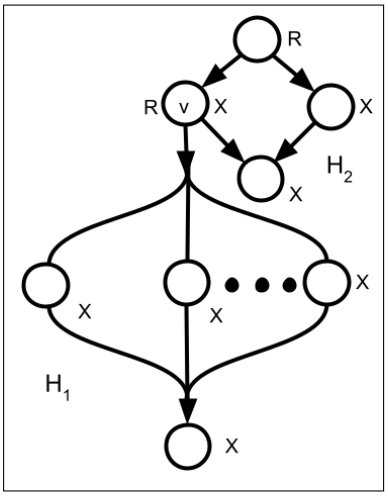
Figura 19 - $H_1$ hipergrafo de declaração $p$-paralelo e $H_2$ grafos de declaração sequencial com $v \neq s(H_1), t(H_1)$ e $v \neq s(H_1), t(H_2)$. Neste exemplo usamos $H_2$ como o hipergrafo de declaração *Se-Então-Senão*.




Fonte: O autor, 2023.

iv. $v = s(H_1)$ **e** $v \neq s(H_2), t(H_2)$**:** Como $s(H_1)$ é rotulado com $R$ e $v$ é um vértice interno de $H_2$ rotulado com $X$, sabemos que $H_1$ foi obtido pela expansão de um vértice interno de $H_2$. Dessa forma, pela definição da operação de expansão, temos que $H_2$ deixa de ser um hipergrafo de declaração, uma contradição. Veja Figura 20.

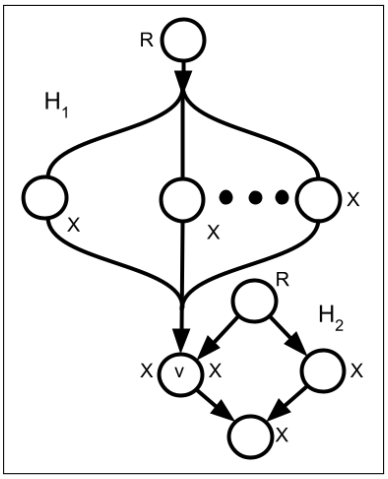
Figura 20 - $H_1$ hipergrafo de declaração $p$-paralelo e $H_2$ grafos de declaração sequencial com $v = s(H_1)$ e $v \neq s(H_2), t(H_2)$. Neste exemplo usamos $H_2$ como o hipergrafo de declaração *Se-Então-Senão*.




Fonte: O autor, 2023.

v. $v = t(H_1)$ **e** $v \neq s(H_2), t(H_2)$**:** Como $v$ é um vértice interno de $H_2$, $|N_G^-(v)| = |N_{H_2}^-(v)| = 1$, pela definição de grafo fechado. Porém, nesse caso, $|N_G^-(v)| = |N_{H_2}^-(v)| + |N_{H_1}^-(t(H_1))| \geq 1 + 1 = 2$ (Figura 21). Uma contradição.

Figura 21 - $H_1$ hipergrafo de declaração $p$-paralelo e $H_2$ grafos de declaração sequencial com $v = t(H_1)$ e $v \neq s(H_2), t(H_2)$. Neste exemplo usamos $H_2$ como o hipergrafo de declaração *Se-Então-Senão*




Fonte: O autor, 2023.

Com isso, tanto na situação em que são usados dois grafos de declaração $p$-paralelos como na situação em que são usados um hipergrafo direcionado de declaração $p$-paralelo e um grafo direcionado de declaração sequencial, a independência de primos só ocorre nas situações em que $V(H_1) \cap V(H_2) = \{v\}$, onde $v = s(H_1) = t(H_2)$ ou $v = s(H_2) = t(H_1)$, ou $V(H_1) \cap V(H_2) = \emptyset$. □

Em um grafo de Dijkstra a contração de um hipergrafo de declaração não modifica outro hipergrafo de declaração. Chamamos esta propriedade de preservação de primos, a qual é demonstrada no Lema 2.3.

**Lema 2.3** (**Preservação dos Primos**). *Dado um hipergrafo direcionado de Dijkstra $G$ se $H_1, H_2 \in \mathcal{H}(G)$, $H_1 \neq H_2$, então a imagem de $H_2$ no grafo resultante da contração de $H_1$ em $G$ pertence a coleção dos grafos de declaração do grafo resultante da contração de $H_1$ em $G$, isto é, $I_{G\downarrow H_1}(H_2) \in \mathcal{H}(G \downarrow H_1)$.*

*Demonstração.* Se considerarmos $H_1$ e $H_2$ como grafos de declaração sequencial, a preservação de primos é consequência do Lema 4.3 de [BENTO et al., 2019].

Quando $H_1$ é um hipergrafo de declaração $p$-paralelo e $H_2$ é um hipergrafo de declaração sequencial, o Lema 2.2 estabelece que $H_1$ e $H_2$ são independentes. Isso implica que $V(H_1) \cap V(H_2) = \emptyset$ ou $V(H_1) \cap V(H_2) = \{v\}$, onde $v = s(H_1) = t(H_2)$ ou $v = s(H_2) = t(H_1)$. Se $V(H_1) \cap V(H_2) = \emptyset$, a contração de $H_1$ em $G$ resulta no grafo $G \downarrow H_1$, e assim $H_2$ é um subgrafo de $G \downarrow H_1$. Consequentemente, $I_{G\downarrow H_1}(H_2)$ pertence a $\mathcal{H}(G \downarrow H_1)$. Se $V(H_1) \cap V(H_2) = \{v\}$, podem ocorrer duas situações: $v = s(H_1) = t(H_2)$ ou $v = s(H_2) = t(H_1)$.

Considerando $v = s(H_1) = t(H_2)$ (Figura 22a), ao contrair $H_2$, todas as vizinhanças dos vértices de $I_{G\downarrow H_2}(H_1)$ permanecem inalteradas, exceto a de $I_{G\downarrow H_2}(s(H_1))$, já que $N^-_{I_{G\downarrow H_2}}(s(H_1)) = N^-_G(s(H_2))$. A contração de $H_2$ não adiciona novos ciclos em $H_1$, preservando assim a propriedade de $H_1$ como um hipergrafo de declaração não trivial e fechado, além de primo, em $G \downarrow H_2$. Ao contrair $H_1$ no lugar de $H_2$, todas as vizinhanças dos vértices de $I_{G\downarrow H_1}(H_2)$ se mantêm inalteradas, exceto a de $I_{G\downarrow H_1}(t(H_2))$, uma vez que $N^+_{I_{G\downarrow H_1}}(t(H_2)) = N^+_G(t(H_1))$.

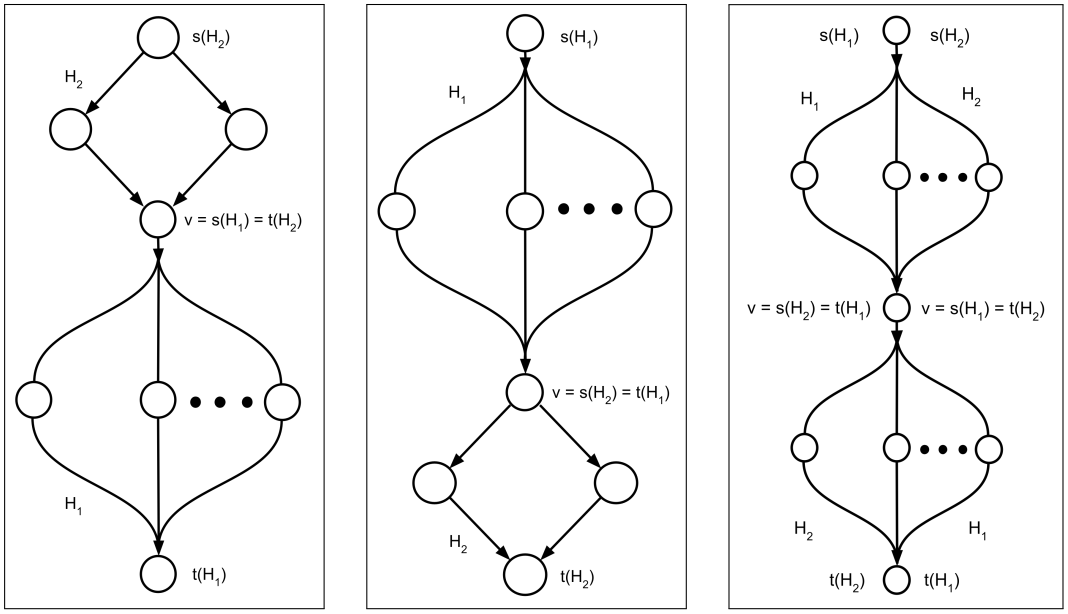
Considerando $v = s(H_2) = t(H_1)$ (Figura 22b), a contração de $H_1$ ou a contração de $H_2$ são análogas ao caso anterior com relação as vizinhanças de entrada e de saída e sobre a não inclusão de novos ciclos no grafo resultante da contração.

Considerando $H_1$ e $H_2$ como grafos de declaração $p$-paralelo (Figura 22c) e assumindo $v = s(H_1) = t(H_2)$, a contração de $H_1$ ou a contração de $H_2$ são análogas ao primeiro caso com relação as vizinhanças de entrada e de saída e sobre a não inclusão de novos ciclos no grafo resultante da contração. □

Dado uma coleção $\mathcal{C}$ de grafos primos de um hipergrafo direcionado de Dijkstra, uma propriedade crucial explorada pelo algoritmo de reconhecimento é que o hipergrafo direcionado resultante de qualquer sequência de contrações dos hipergrafos primos de $\mathcal{C}$ é invariante. Chamamos esta propriedade de Lei da Comutatividade a qual é provada no Lema 2.4.

**Lema 2.4** (**Lei da Comutatividade**). *Dado um hipergrafo direcionado de Dijkstra $G$, se os subgrafos $H, H' \in \mathcal{H}(G)$, então o hipergrafo direcionado resultante da contração de*

Figura 22 - Exemplo das considerações $v = s(H_1) = t(H_2)$, $v = s(H_2) = t(H_1)$ e $H_1$ e $H_2$ hipergrafos direcionados de declaração $p$-paralelo.




(a) (b) (c)

Legenda: (a) $v = s(H_1) = t(H_2)$; (b) $v = s(H_2) = t(H_1)$; (c) $H_1$ e $H_2$ hipergrafos direcionados de declaração $p$-paralelo.

Fonte: O autor, 2023.

*$H'$ do hipergrafo resultante da contração de $H$ em $G$ é isomorfo ao hipergrafo direcionado resultante da contração de $H$ do hipergrafo resultante da contração de $H'$ em $G$, isto é* $(G \downarrow H) \downarrow I_{G\downarrow H}(H') \cong (G \downarrow H') \downarrow I_{G\downarrow H'}(H)$.

*Demonstração.* Pelo Lema 2.2, $H, H'$ são independentes. Primeiro, suponha que $H, H'$ são disjuntos. Então $I_{G\downarrow H}(H') = H'$ e $I_{G\downarrow H}(H) = H$. Com isso, os subgrafos $H, H'$ são, respectivamente, substituídos por um par de vértices não adjacentes, onde as vizinhanças de entrada são $N^-(s(H))$ e $N^-(s(H'))$ e as vizinhanças de saída, $N^+(s(H))$ e $N^+(s(H'))$, respectivamente. Então $(G \downarrow H) \downarrow I_{G\downarrow H}(H') \cong (G \downarrow H') \downarrow I_{G\downarrow H'}(H)$. Na segunda alternativa, supondo que $H$ e $H'$ são não disjuntos, então $V(H) \cap V(H') = \{v\}$, onde $v = s(H) = t(H')$ ou $v = s(H') = t(H)$. Em ambos os casos e nos grafos $(G \downarrow H) \downarrow I_{G\downarrow H}(H')$ e $(G \downarrow H') \downarrow I_{G\downarrow H'}(H)$, os subgrafos $H$ e $H'$ são contraídos em um vértice comum $w$. Considerando $A = (G \downarrow H) \downarrow I_{G\downarrow H}(H')$ e $B = (G \downarrow H') \downarrow I_{G\downarrow H'}(H)$. Quando $v = s(H) = t(H')$, $N_A^-(w) = N_G^-(s(H')) = N_B^-(w)$ e $N_A^+(w) = N_G^+(t(H)) = N_B^+(w)$. Finalmente, quando $v = s(H') = t(H)$ obtemos um resultado similar. Consequentemente, $A \cong B$ em qualquer situação. □

**Definição 2.2** ([**BENTO et al., 2019**]). *Uma sequência de grafos* $G = \{G_i | i \in \{0, 1, 2, \ldots, k\}\}$ *é uma* ***sequência de contrações****, quando*

- $G \cong G_0$, *e*

- $G_{i+1} \cong (G \downarrow H_i)$, *para algum* $H_i \in \mathcal{H}(G_i)$, $i < k$. *Chamamos* $H_i$ *de* ***primo de contração*** *de* $G_i$.

**Definição 2.3** ([**BENTO et al., 2019**]). *Uma sequência de grafos* $G = \{G_i | i \in \{0, 1, 2, \ldots, k\}\}$ *é uma* ***sequência de contrações maximal****, quando* $\mathcal{H}(G_k) = \emptyset$. *Em particular, se* $G_k$ *é um grafo trivial, então* $\{G_i | i \in \{0, 1, 2, \ldots, k\}\}$ *é maximal.*

**Definição 2.4** ([**BENTO et al., 2019**]). *Seja uma sequência de contrações* $G = \{G_i | i \in \{0, 1, 2, \ldots, k\}\}$ *de* $G$ *e* $H_j$ *um primo contrátil de* $G_j$. *Isto é,* $G_{j+1} \cong (G_j \downarrow H_j)$, $0 \leq j < k$. *Para* $H'_j \subseteq G_j$ *e* $q \geq j$, *a* ***imagem iterada*** *de* $H'_j$ *em* $G_q$ *é o subgrafo* $I_{G_q}(H'_j)$ *de* $G_q$, *obtida através da procura iterativa da imagem* $I_{G_{j+1}}(H'_j)$ *de* $H'_j$ *em* $G_{j+1} = (G_j \downarrow H_j)$, *e então* $I_{G_{j+2}}(H'_j)$ *de* $I_{G_{j+1}}(H'_j)$ *em* $G_{j+2} = G_{j+1} \downarrow H_{j+1}$ *e assim por diante até encontrar a imagem* $I_{G_q}(H'_j)$ *de* $I_{G_{q-1}}(H'_j)$ *de* $I_{G_{q-1}}(H'_j)$ *em* $G_q = G_{q-1} \downarrow H_q$. *Com isso, podemos definir, recursivamente,* $I_{G_q}(H'_j)$ *como*

$$I_{G_q}(H'_j) = \begin{cases} H'_j, & \text{se } q = j \\ I_{G_{q-1}} \downarrow H_{q-1}(I_{G_{q-1}}(H'_j)), & \text{caso contrário.} \end{cases} \tag{2}$$

No Teorema 2.1, nós consideramos uma sequência de contrações de hipergrafos direcionados de declaração, onde o hipergrafo resultante não admita outra contração, ou seja, não possui hipergrafos de declaração não trivial como subgrafo. Com isso, nós provaremos que tomadas quaisquer duas dessas sequências de contrações, que são maximais, obteremos o mesmo hipergrafo direcionado.

**Teorema 2.1.** *Seja* $G$ *um hipergrafo de fluxo arbitrário, com* $G = \{G_i | i \in \{0, 1, 2, \ldots, k\}\}$ *e* $G' = \{G'_i | i \in \{0, 1, 2, \ldots, k'\}\}$, *duas sequências de contrações maximais de* $G$. *Então* $G_k \cong G'_{k'}$, *além de* $k = k'$.

*Demonstração.* Sejam $G = \{G_i | i \in \{1, 2, \ldots, k\}\}$ e $G' = \{G'_i | i \in \{1, 2, \ldots, k'\}\}$ duas sequências de contrações maximais, denotadas por $S$ e $S'$, respectivamente, de um hipergrafo $G$. Sejam $H_j$ e $H'_j$ primos contraídos de $G_j$ e $G'_j$. Isto é, $G_{j+1} \cong (G_j \downarrow H_j)$ e $G'_{j+1} \cong (G'_j \downarrow H'_j)$, com $j < k$ e $j < k'$. Sem perda de generalidade, suponha $k \leq k'$. Seja $i$ o menor índice, de modo que $G_j \cong G'_j$, $j \leq i$. Esse índice existe, uma vez que $G \cong G_0 \cong G'_0$. Se $i = k$, então $G_k \cong G'_{k'}$, o que implica que $k = k'$ e o teorema é válido. Caso contrário, $i < k$, $G_i \cong G'_i$ e $G_{i+1} \ncong G'_0$. Uma vez que $G_i \cong G'_i$, segue $H_i \in \mathcal{H}(G'_i)$. Pelo Lema 2.3, a imagem iterada $H_{i_q}$, de $H_i$ em $G'_q$ é preservada como um subgrafo primo para todo $G'_q$, desde que não se torne o primo contraído de $G'_{q-1}$. Uma vez que $G'_{k'}$ não tem um subgrafo primo, existe um índice $p$, com $i < p < k'$ de modo que $G'_{p+1} \cong (G_p \downarrow H_{i_p})$, onde $H_{i_p}$ representa a imagem iterada de $H_i$ em $G'_p$. Seja $H_{i_{p-1}}$ a imagem iterada de $H_i$ em $G'_{p-1}$. Claramente, $H'_{p-1}, H_{i_{p-1}} \in \mathcal{H}(G'_{p-1})$, e pelo Lema 2.2, $H'_{p-1}$ e $H_{i_{p-1}}$ são independentes em $G'_{p-1}$. Desde que $((G'_{p-1} \downarrow H'_{i_{p-1}}) \downarrow H_{p-1}) \cong G'_{p+1}$, pelo Lema

2.3, segue que $(G'_{p-1} \downarrow H_{i_{p-1}}) \downarrow H''_{p-1} \cong G'_{p+1}$, onde $H''_{p-1}$ representa a imagem de $H'_{p-1}$ em $G'_{p-1} \downarrow H_{i_{p-1}}$. Consequentemente, as posições em $S'$ de dois primos contraídos foram trocadas, respectivamente nos índices $p-1$ e $p$, enquanto os grafos $G'_q$, para $q < p-1$ e $q > p$, foram preservados. Em particular, também foram preservados $G'_{p+1}$ e os grafos que se encontram depois de $G'_{p+1}$ em $S'$, juntamente com seus primos contraídos. □

No Lema 2.5 mostramos que o número de arcos $m$, considerando também o grafo direcionado de declaração $p$-paralelo, em um hipergrafo direcionado de Dijkstra é limitado por $m \leq 2n-2$, mesmo limite do Lema 4.6 de [BENTO et al., 2019].

**Lema 2.5.** *Se $G = (V, E)$ é um hipergrafo direcionado de Dijkstra, com $n = |V|$ e $m = |E|$, então $m \leq 2n-2$.*

*Demonstração.* Para reconhecimento de um hipergrafo de Dijkstra, precisamos considerar o hipergrafo de declaração $p$-paralelo (Figura 14). Se $G$ é um hipergrafo direcionado de Dijkstra, existe uma sequência de grafos $G = \{G_i | i \in \{1, 2, \ldots, k\}\}$ onde $G_0$ é o grafo direcionado trivial, $G_k \cong G$ e $G_i$ é obtido de $G_{i-1}$ ao expandir um vértice rotulado com $X$ de $G_{i-1}$ em algum grafo direcionado de declaração da Figura 6 ou em um grafo direcionado de declaração $p$-paralelo.

Vamos aplicar indução ao número de expansões empregadas na construção de $G$. Se $k = 0$, então $G$ é um grafo direcionado de declaração trivial, que satisfaz o lema. Para $k > 0$, suponha o lema como verdadeiro para qualquer grafo direcionado $G' \cong G_i$, $i < k$.

Consideremos agora a expansão de um vértice rotulado com $X$ em um hipergrafo de declaração $H$ em um hipergrafo direcionado de Dijkstra $G$. O incremento $n'$ de vértices para o número de vértices de $G$ e o incremento $m'$ de arestas para o número de arestas de $G$, os incrementos $n'$ e $m'$, referentes a cada hipergrafo direcionado de declaração, são mostrados na Tabela 2.

Tabela 2 - Hipergrafos direcionados de declaração e quantos vértices e arestas adicionam ao hipergrafo direcionado de Dijkstra $G$ no momento da expansão de um vértice $v$ rotulado com $X$.

| **Hipergrafo direcionado de declaração** | ***n'*** | ***m'*** |
|---|---|---|
| **Trivial** | +0 | +0 |
| **Sequencial** | +1 | +1 |
| **Se-então** | +3 | +3 |
| **Se-então-senão** | +4 | +4 |
| ***q*-Caso** | +q+1 | +2q |
| **Enquanto** | +2 | +3 |
| **Repita** | +2 | +3 |
| ***p*-paralelo** | +p+1 | +2 |

Fonte: O autor, 2023.

Vamos fazer uma prova por indução no número $x \geq 1$ de operações de expansão. Dado $n_i, m_i$ serem respectivamente os números de vértices e de arestas em um hipergrafo direcionado de Dijkstra obtido com $i$ operações de expansão.

Para $x = 1$, veja que $m_1 \leq 2n_1 - 2$. (Base)

Suponha por indução que para $x = k$ operações tenhamos que $m_k \leq 2n_k - 2$ com $k \geq 1$. (Hipótese de Indução)

Assim, para um hipergrafo direcionado de Dijkstra obtido com $k+1$ operações de expansão, usando a base e a hipótese de indução, temos que

$$m_{k+1} = m_k + m' \leq 2n_k + 2n' - 2 = 2(n_k + n') - 2 = 2n_{k+1} - 2.$$

Como queríamos demonstrar.

□

O Algoritmo 4 decorre basicamente do Teorema 2.1 e do Lema 2.5. No entanto, o Lema 2.5 depende do fato de $G$ ser um hipergrafo redutível, porém a entrada para o Algoritmo 4 é um hipergrafo arbitrário e nenhuma etapa explícita para reconhecer se $G$ é um hipergrafo redutível é executado. O objetivo era evitar esse reconhecimento prévio, cuja complexidade não é linear. O Lema 2.6 a seguir justifica isso.

**Lema 2.6.** *Seja $G$ um hipergrafo de fluxo arbitrário de entrada do Algoritmo 4. Se $G$ não é um grafo redutível, então o algoritmo responderá* ***Não****.*

*Demonstração.* Se $G$ não é um grafo redutível, seja $E_C$ o conjunto de arestas ciclos, relativas a alguma busca em profundidade iniciando pelo $s(G)$. Então $G$ contem algum ciclo $C$, tal que $w$ não separa $s(G)$ de $v$, onde $vw \in E_C$ é a aresta do ciclo de $C$. Sem perda de generalidade, considere-se o mais interior destes ciclos. A única forma em que a aresta $vw$, ou qualquer uma das suas possíveis imagens, pode ser contraída é no contexto de um subgrafo primo $H$, no qual o ciclo seria contraído no vértice $w$, ou numa possível imagem iterada do mesmo. No entanto, não há possibilidade de $H$ ser identificado como tal, porque a aresta que entra no ciclo pelo exterior impede que o subgrafo seja fechado. Por conseguinte, o algoritmo responderia necessariamente **Não**. □

**Algoritmo 4:** Reconhecimento dos hipergrafos de Dijkstra

**Entrada:** Grafo de fluxo $G$ arbitrário.

**Saída:** **Verdadeiro** se $G$ é hipergrafo de Dijkstra e **Falso**, caso contrário.

```
1  início
2      hipergrafoDijkstra ← Falso
3      Calcula o número m de hiper-arcos de G, pára a contagem quando
         m = 2n − 2
4      se m ≤ 2n − 2 então
5          E_c conjunto de hiper-arcos ciclo de uma busca em profundidade
             (DFS), iniciando no s(G)
6          v_1, v_2, ..., v_n OT de G − E_c
7          i ← n
8          enquanto i ≥ 1 faça
9              se G é o grafo trivial então
10                 hipergrafoDijkstra ← Verdadeiro
11             fim
12             se v_i = s(H) de um subgrafo primo H de G então
13                 G ← G ↓ H
14                 i ← i − 1
15             fim
16         fim
17     fim
18     retornar hipergrafoDijkstra
19 fim
```

**Definição 2.5.** *[SZWARCFITER, 2018] Suponha que $v$ pertença ao caminho de $r$ a $w$ em uma árvore $T$. Então $v$ é ancestral de $w$, sendo $w$ descendente de $v$. Se ainda $v \neq w$, $v$ é ancestral próprio de $w$, e este é descendente próprio de $v$.*

**Definição 2.6.** *É dito que $\mathcal{C}$ é uma sequência de baixo para cima de $G$ quando $s(H_i)$ não é descendente de $s(H)$, para qualquer primo $H \neq H_i$ de $G_i$.*

**Lema 2.7.** *Seja $G$ um hipergrafo de fluxo arbitrário e subgrafos $H$ e $H'$, $H \in \mathcal{H}(G)$, $H' \in \mathcal{H}(G \downarrow H) \backslash \mathcal{H}(G)$. Então $s(H)$ é descendente próprio de $s(H')$ em $G \downarrow H$.*

*Demonstração.* Seja $G$ um hipergrafo redutível, $\{G_i | i \in \{0, 1, 2, \ldots, k\}\}$ uma sequência de contrações $\mathcal{C}$ de $G$, e $H_i$ um primo contraído de $G_i$, $0 \leq i < k$. □

A complexidade do Algoritmo 4 é $O(n)$, pelo Lema 2.5, pois existem alguns passos necessários para que o hipergrafo de fluxo $G$ seja reconhecido ou não como hipergrafo direcionado de Dijkstra. É necessário $O|N^+(v)|$ passos para decidir se o hipergrafo contém

um subgrafo primo não trivial com origem em um vértice $v \in V(G)$. Para considerar todos os vértices de $G$, é necessário $O(m)$ passos. Considerando a possibilidade de $G$ possuir $|V(G)|$ subgrafos primos $H$ que serão contraídos, a complexidade deste passo é $O(|V(G|)$ e, na contração, $G$ diminui o número de arestas $(|E(G)| - |E(H)|)$ e o número de vértices $(|V(G)| - |V(H)| + 1)$.

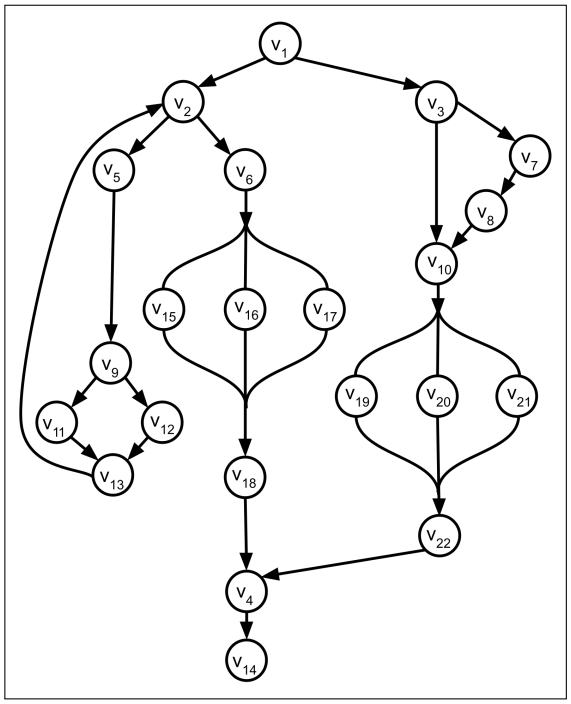
O algoritmo de reconhecimento de hipergrafo direcionado de Dijkstra $G$ possui uma complexidade de tempo linear, ou seja, $O(n)$, onde $n$ representa o número de vértices no grafo. Isso significa que o tempo de execução do algoritmo aumenta de forma proporcional ao número de vértices no grafo. Apesar de fazer algumas verificações como se o hipergrafo contém um subgrafo primo não trivial cuja origem é um dado vértice $v \in V(G)$, sendo necessário $O|N^+(v)|$ passos.

A eficiência do algoritmo é alcançada por meio de um processo de construção de uma sequência de contração de baixo para cima de cada subgrafo primo $H$, o que requer $O(n)$ passos ao verificar se um vértice $v \in V(G)$ é um vértice fonte $s(H)$ de um hipergrafo direcionado de declaração. Os vértices $v \in V(H)$ e as arestas $e \in E(H)$ não são verificadas novamente após a contração pois , para a próxima verificação é considerado o hipergrafo $G - H$.

A Figura 23 ilustra um hipergrafo de Dijkstra específico que será utilizado para ilustrar o funcionamento do Algoritmo 4 e, posteriormente, do Algoritmo 5. No caso do Algoritmo 4, a $OT$ obtida foi $v_1, v_2, v_5, v_9, v_{12}, v_{13}, v_6, v_{15}, v_{16}, v_{17}, v_{18}, v_3, v_7, v_8, v_{10}, v_{19}, v_{20}, v_{21}$, $v_{22}, v_4, v_{14}$.

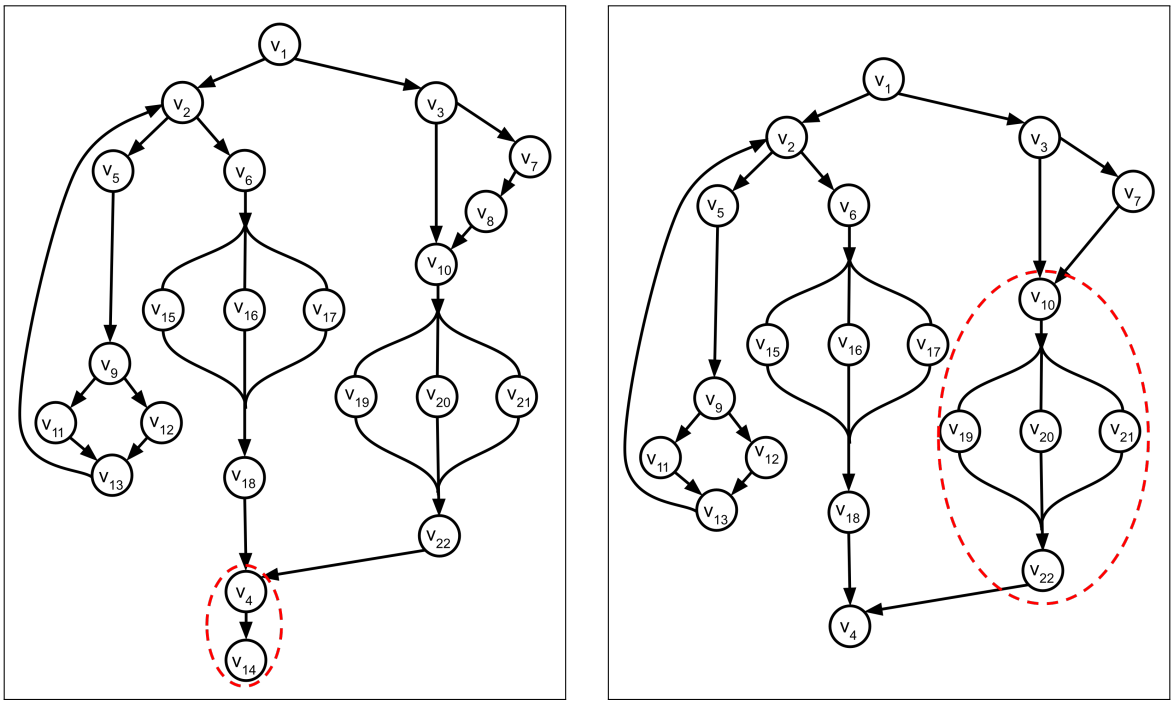
As Figuras 24 e 25 a seguir mostram a sequência de contrações $v_4, v_{10}, v_7, v_3, v_6, v_9$, $v_5, v_2, v_1$ do hipergrafo de Dijkstra da Figura 23, onde a $OT$ usada foi mostrada.

Figura 23 - Hipergrafo de Dijkstra.




Fonte: O autor, 2023.

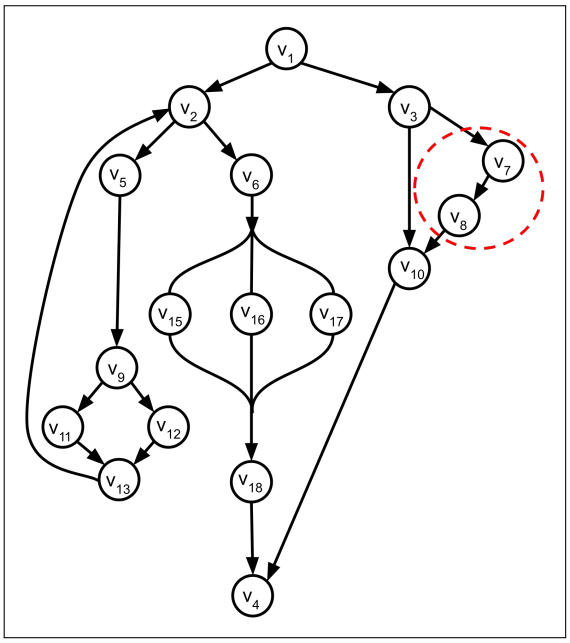
Figura 24 - Sequência de contrações do hipergrafo de Dijkstra (Figura 23) na execução do Algoritmo 4.



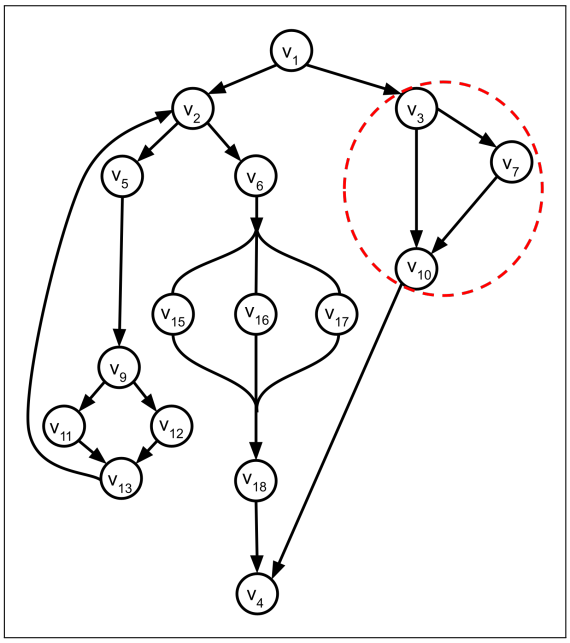

(a) (b)

Legenda: (a) $G \downarrow H$, onde $s(H) = v_4$; (b) $G \downarrow H$, onde $s(H) = v_{10}$.
Fonte: O autor, 2023.

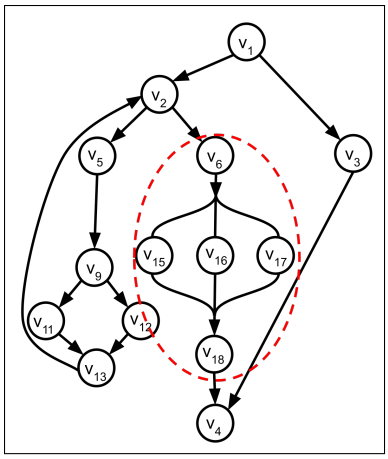
Figura 25 - Continuação da Figura 23.



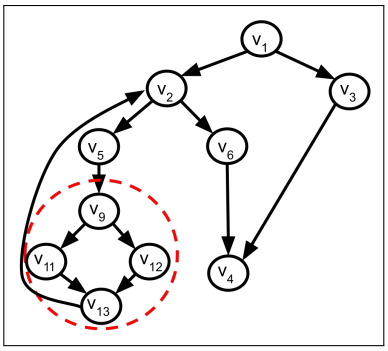

(c)



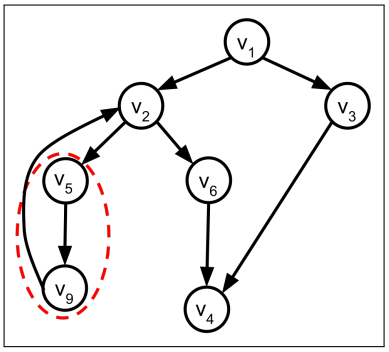

(d)



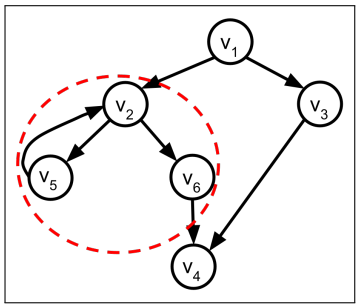

(e)



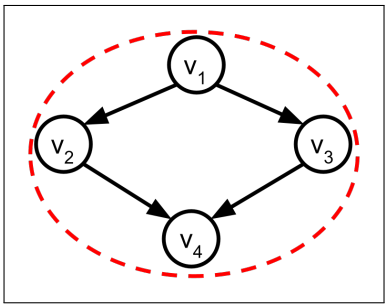

(f)



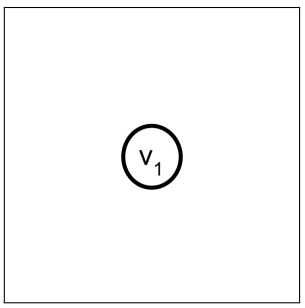

(g)




(h)




(i)




(j)

Legenda: (c) $G \downarrow H$, onde $s(H) = v_7$; (d) $G \downarrow H$, onde $s(H) = v_3$; (e) $G \downarrow H$, onde $s(H) = v_6$; (f) $G \downarrow H$, onde $s(H) = v_9$; (g) $G \downarrow H$, onde $s(H) = v_5$; (h) $G \downarrow H$, onde $s(H) = v_2$; (i) $G \downarrow H$, onde $s(H) = v_1$; (j) Grafo Trivial.

Fonte: O autor, 2023.

# 3 ISOMORFISMO DOS HIPERGRAFOS DE DIJKSTRA

Como na teoria dos grafos, a teoria dos hipergrafos também possui o conceito de isomorfismo entre hipergrafos. Neste capítulo exploramos uma forma de verificar o isomorfismo entre hipergrafos de Dijkstra baseado nos grafos de declaração que os compõem e apresentamos um algoritmo para ajudar nessa verificação.

Como no algoritmo de reconhecimento, os códigos são obtidos através da construção de uma sequência contráctil de baixo para cima de cada hipergrafo. Nosso algoritmo de verificação do isomorfismo entre 2 hipergrafos $G$ e $G'$ consiste em codificar os hipergrafos com códigos $C(G)$ e $C(G')$ de tal forma que $G$ é isomorfo a $G'$ se e somente se $C(G) = C(G')$. Essa estratégia segue em linha com a que foi utilizada no artigo original para grafos de Dijkstra [BENTO et al., 2019].

Na Tabela 3 são mostrados os tipos $Tipo(H)$ e os códigos $C(H)$ dos grafos de declaração sequencial que foram usados no artigo fundamental [BENTO et al., 2019]. Em nosso trabalho, foi adicionado o grafo de declaração $p$-paralelo junto ao código correspondente.

A criação dos códigos $C(H)$ na Tabela 3 é feita da seguinte forma: seja $A, B$ um par de cadeia de caracteres numéricos, $A||B$ é a cadeia de caracteres numéricos formada pela cadeia $A$ concatenada com a cadeia $B$.

Para o hipergrafo direcionado de declaração $p$-Paralelo, o $Tipo(H) = 2(p + 1) + 1$, onde $p$ é a quantidade de vértices de execução paralela, que representa a quantidade de processos do programa executados simultaneamente, onde a quantidade mínima é $p \geq 2$. Com isso, o código $C(H)$ para o $p$-paralelo será $C(H) = 2(p + 1) + 1||lex(C(N^{+}(v_i)))||C(N^{+2}(v_i))$.

O $Tipo(H)$ do grafo de declaração $q$-Caso é $2(q+1)$, onde $q \geq 3$ pois é a quantidade miníma de vértices internos para esse grafo de declaração, onde cada vértice interno representa uma condição, pois se for $q = 2$ será equivalente ao grafo de declaração *Se-Então-Senão* ou, se $q = 1$, será equivalente ao grafo de declaração *Se-Então*. Então, o código $C(H)$ para o $q$-Caso será $C(H) = 2(p + 1)||lex(C(N^{+}(v_i)))||C(N^{+2}(v_i))$.

Os tipos do $q$-Caso e do $p$-Paralelo são, respectivamente, $2(q + 1)$ e $2(p + 1) + 1$ para que o tipo do $p$-Paralelo seja ímpar e maior ou igual a 7 e o tipo do $q$-Caso seja par e maior ou igual a 8. Pois, quando não há essa diferenciação entre par ou ímpar e só considerando os tipos do $q$-Caso e do $p$-Paralelo maiores que 7, existiria a possibilidade dos tipos serem iguais, quando $q = p = 3$ por exemplo.

Tabela 3 - Grafos de declaração, seus tipos e códigos $C(H)$ de subgrafo primo $H$.

| **Grafo de Declaração ($H$)** | **Tipo($H$)** | $C(H)$, $v = s(H)$ |
|---|---|---|
| **Trivial** | 1 | |
| **Sequência** | 2 | $2 \vert\vert C(N^+(v))$ |
| **Se-então** | 3 | $3 \vert\vert C(N^+(v) \backslash N^{+2}(v)) \vert\vert C(N^{+2}(v))$ |
| **Enquanto** | 4 | $4 \vert\vert C(N^+(v) \cap N^-(v)) \vert\vert C(N^+(v) \backslash N^-(v))$ |
| **Repita** | 5 | $5 \vert\vert C(N^+(v)) \vert\vert C(N^{+2}(v) \backslash \{v\})$ |
| **Se-então-senão** | 6 | $6 \vert\vert lex(C(N^+(v))) \vert\vert C(N^{+2}(v))$ |
| $p$**-Paralelo** | $2(p+1)+1$ | $2(p+1)+1 \vert\vert lex(C(N^+(v))) \vert\vert C(N^{+2}(v))$ |
| $q$**-Caso** | $2(q+1)$ | $2(q+1) \vert\vert lex(C(N^+(v))) \vert\vert C(N^{+2}(v))$ |

Fonte: O autor, 2023.

**Definição 3.1** ([**BRETTO, 2013**]). *Um hipergrafo direcionado $H$ é* ***simples*** *se para todas as arestas $e_i, e_j, e, a \in E$, as condições a seguir são satisfeitas*

- *se $e_i \subseteq e_j$, então $i = j$;*
- *se $e^+ \cap a^+ = \emptyset$, então $e^- \cap a^- = \emptyset$; se $e^+ \cap a^- = \emptyset$, então $e^- \cap a^+ = \emptyset$;*
- *se $e^- \cap a^- = \emptyset$, então $e^+ \cap a^+ = \emptyset$; se $e^- \cap a^+ = \emptyset$, então $e^+ \cap a^- = \emptyset$;*

**Definição 3.2** ([**BRETTO, 2013**]). *Considerado um hipergrafo direcionado $H$ simples. Sejam $H = (V; E = \{e_i : i \in I\})$ e $H' = (V; E' = \{a_j : j \in J\})$ dois hipergrafos direcionados. $H$ e $H'$ são* ***isomorfos*** *se existe uma bijeção $f : V \to V'$ e uma bijeção $\pi : I \to J$ que induzem uma bijeção $g : E \to E'$.*

**Teorema 3.1** (**Isomorfismo de hipergrafos de Dijkstra**). *Sejam $G, G'$ hipergrafos de Dijkstra e $C(G), C(G')$ seus códigos, respectivamente. Então $G, G'$ são isomorfos se, e somente se, $C(G) = C(G')$.*

*Demonstração.* Primeiro, considere $G, G'$ isomorfos. Mostramos que isso implica $C(G) = C(G')$. Seguindo o Algoritmo 5 de isomorfismo, observe que a quantidade de $1's$ nos códigos $C(G), C(G')$ representa o número de vértices de $G, G'$, respectivamente, considerando que cada número inteiro maior que 1 no código representa a contração de um subgrafo primo. Além disso, cada subgrafo primo que está inicialmente contido no grafo de entrada $G$, corresponde em $C(G)$, a uma cadeia numérica formada pelo inteiro $Tipo(H)$ seguido de um 1 se $Tipo(H) = 2$, ou dois $1's$ se $3 \leq Tipo(H) \leq 5$, ou três $1's$ se $Tipo(H) = 6$, ou $\left\lfloor \frac{Tipo(H)}{2} \right\rfloor$ $1's$ se $Tipo(H) \geq 7$, respectivamente. O mesmo ocorre com $G'$ e seu código $C(G')$. □

**Algoritmo 5:** Algoritmo de Isomorfismo para hipergrafos de Dijkstra

**Entrada:** Hiperfrafo de Dijkstra $G$ e conjunto $E_c$ de arcos-ciclos de $G$.

**Saída:** Código $C(G)$ do hipergrafo $G$.

**início**

  Encontra uma ordenação topológica $v_1, v_2, \ldots, v_n$

  **para** $i \leftarrow n, n-1, \ldots, 1$ **faça**

    $C(v_i) \leftarrow 1$

    **se** $v_i$ *é* $s(H)$ *de um subgrafo primo* $H$ **então**

      $$C(v_i) \leftarrow C(v_i)||\begin{cases} ;2;||C(N^+(v_i)) \\ ;3;||C(N^+(v_i)\backslash N^{+2}(v_i));||N^{2+}(v_i)) \\ ;4;||C(N^+(v_i)\cap N^-(v_i));||C(N^+(v_i)\backslash N^-(v_i)) \\ ;5;||C(N^+(v_i));||C(N^{+2}(v_i)\backslash\{v_i\} \\ ;6;||lex(C(N^+(v_i)));||C(N^{+2}(v_i)) \\ ;2(p+1)+1;||lex(C(N^+(v_i)));||C(N^{+2}(v_i)) \\ ;2(q+1);||lex(C(N^+(v_i)));s||C(N^{+2}(v_i)) \end{cases}$$

    **fim**

    **se** $i > 1$ **então**

      $C(v_i) \leftarrow C(v_i)+$";"

    **fim**

  **fim**

  $C(G) \leftarrow C(v_1)$

**fim**

Como os tipos $Tipo(H)$ dos hipergrafos de declaração $q$-Caso e $p$-Paralelo são variáveis, existe a possibilidade de serem confundidos com o tipo de algum outro hipergrafos de declaração. Por exemplo, o $Tipo(H)$ da fonte $s(H)$ de $H$, sendo $H$ o hipergrafo de declaração $p$-Paralelo e $p = 5$, será 13, então o código para esse vértice fonte $s(H)$, de acordo com o algoritmo, será $C(s(H)) = 1131111$ e, com isso, fica difícil identificar qual hipergrafo de declaração pertence esse código, se é o $p$-Paralelo ou o Se-então, no código final do hipergrafo. O uso do ";", no Algoritmo 5, é para separar o código de cada vértice com a finalidade de não ocorrer essa dificuldade de identificação.

Outro ponto, que poderia causar problema, é o hipergrafo ter mais de uma ordenação topológica. O código para esse hipergrafo, gerado pelo Algoritmo 5, será sempre o mesmo porque o algoritmo trata essas diferentes ordenações topológicas no momento de criação do código de forma lexicográfica para os hipergrafos de declaração Se-então-senão, $p$-Paralelo e $q$-Caso.

A seguir, para a conveniência do leitor, mostramos a criação dos códigos de três hipergrafos de Dijkstra $H_1$, $H_2$ e $H_3$, onde $H_1$ é o hipergrafo de Dijkstra da Figura 23, $H_2$ é o hipergrafo de Dijkstra da Figura 26 e $H_3$ é o hipergrafo de Dijkstra da Figura 27.

Para $H_1$, a ordenação topológica de $H_1$ é dada por $OT_{H_1} = v_1, v_2, v_5, v_9, v_{11}v_{12}, v_{13}, v_6, v_{15}, v_{16}, v_{17}, v_{18}, v_3, v_7, v_8, v_{10}, v_{19}, v_{20}, v_{21}, v_{22}, v_4, v_{14}$ com a sequência de contração correspondente $v_4$, $v_{10}, v_7v_3, v_6, v_9, v_5$, $v_2$, $v_1$, já apresentadas.

$C(v_{14}) = 1;$
$C(v_4) = 1; 2; ||C(v_{14});$
$C(v_4) = 1; 2; 1;$
$C(v_{22}) = 1;$
$C(v_{21}) = 1;$
$C(v_{20}) = 1;$
$C(v_{19}) = 1;$
$C(v_{10}) = 1; 9; ||C(v_{19}); C(v_{20}); C(v_{21}); ||$
$C(v_{22});$
$C(v_{10}) = 1; 9; 1; 1; 1; 1;$
$C(v_8) = 1;$
$C(v_7) = 1; 2; ||C(v_8);$
$C(v_7) = 1; 2; 1;$
$C(v_3) = 1; 3; ||C(v_7)||C(v_{10});$
$C(v_3) = 1; 3; 1; 2; 1; 1; 9; 1; 1; 1; 1;$
$C(v_{18}) = 1;$
$C(v_{17}) = 1;$
$C(v_{16}) = 1;$
$C(v_{15}) = 1;$
$C(v_6) = 1; 9; ||C(v_{15}); C(v_{16}); C(v_{17}); ||$
$C(v_{18});$
$C(v_6) = 1; 9; 1; 1; 1; 1;$
$C(v_{13}) = 1;$
$C(v_{12}) = 1;$
$C(v_{11}) = 1;$
$C(v_9) = 1; 6; ||C(v_{11}); C(v_{12}); ||C(v_{13});$
$C(v_9) = 1; 6; 1; 1; 1;$
$C(v_5) = 1; 2; ||C(v_9);$
$C(v_5) = 1; 2; 1; 6; 1; 1; 1;$
$C(v_2) = 1; 4; ||C(v_5); ||C(v_6);$
$C(v_2) = 1; 4; 1; 2; 1; 6; 1; 1; 1; 1; 9; 1; 1; 1; 1;$
$C(v_1) = 1; 6; ||C(v_2); C(v_3); ||C(v_4);$

$$C(H_1) = C(v_1) = 1; 6; 1; 4; 1; 2; 1; 6; 1; 1; 1; 1; 9; 1; 1; 1; 1; 1; 3; 1; 2; 1; 1; 9; 1; 1; 1; 1; 1; 2; 1$$

Outra $OT$ para o hipergrafo $H_1$ é $v_1, v_2, v_5, v_9, v_{11}, v_{13}, v_6, v_{15}, v_{18}, v_4, v_{14}, v_{16}, v_{17}, v_3, v_7, v_8, v_{10}, v_{19}, v_{22}, v_{20}, v_{21}, v_{12}$. Com essa $OT_{H_1}$, o código $C(H_1)$ será o mesmo como mostrado a baixo.

$C(v_{12}) = 1;$
$C(v_{21}) = 1;$
$C(v_{20}) = 1;$
$C(v_{22}) = 1;$
$C(v_{19}) = 1;$
$C(v_{10}) = 1; 9; ||C(v_{19}); C(v_{20}); C(v_{21}); ||$
$C(v_{22});$
$C(v_{10}) = 1; 9; 1; 1; 1; 1;$
$C(v_8) = 1;$
$C(v_7) = 1; 2; ||C(v_8);$
$C(v_7) = 1; 2; 1;$
$C(v_3) = 1; 3; ||C(v_7)||C(v_{10});$
$C(v_3) = 1; 3; 1; 2; 1; 1; 9; 1; 1; 1; 1;$
$C(v_{17}) = 1;$
$C(v_{16}) = 1;$
$C(v_{14}) = 1;$
$C(v_4) = 1; 2; ||C(v_{14});$
$C(v_4) = 1; 2; 1;$
$C(v_{18}) = 1;$
$C(v_{15}) = 1;$
$C(v_6) = 1; 9; ||C(v_{15}); C(v_{16}); C(v_{17}); ||$
$C(v_{18});$
$C(v_6) = 1; 9; 1; 1; 1; 1;$
$C(v_{13}) = 1;$
$C(v_{11}) = 1;$
$C(v_9) = 1; 6; ||C(v_{11}); C(v_{12}); ||C(v_{13});$
$C(v_9) = 1; 6; 1; 1; 1;$
$C(v_5) = 1; 2; ||C(v_9);$
$C(v_5) = 1; 2; 1; 6; 1; 1; 1;$
$C(v_2) = 1; 4; ||C(v_5); ||C(v_6);$
$C(v_2) = 1; 4; 1; 2; 1; 6; 1; 1; 1; 1; 9; 1; 1; 1; 1;$
$C(v_1) = 1; 6; ||C(v_2); C(v_3); ||C(v_4);$

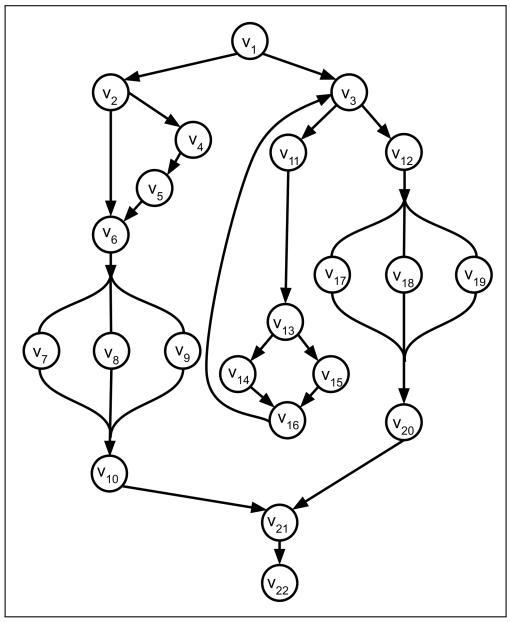
O $C(H_1)$ para essa outra $OT$ é

$$C(H_1) = C(v_1) = 1;6;1;4;1;2;1;6;1;1;1;1;9;1;1;1;1;1;3;1;2;1;1;9;1;1;1;1;1;2;1$$

Com isso, independente da ordenação topológica usada, o tratamento lexicográfico na Tabela 3, usada no Algoritmo 5, fará com que o código do hipergrafo seja sempre o mesmo, independente da ordenação topológica usada.

A seguir, o código $C(H_2)$ para o hipergrafo de Dijkstra da Figura 26, com $OT_{H_2} = v_1, v_3, v_{11}, v_{13}, v_{14}, v_{15}, v_{16}, v_{12}, v_{17}, v_{18}, v_{19}, v_{20}, v_2, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}, v_{21}, v_{22}$. No hipergrafo de Dijkstra $H_2$, a sequência de contrações, baseada em $OT_{H_2}$, será $v_{21}, v_6, v_4, v_2, v_{12}$, $v_{13}, v_{11}, v_3, v_1$. Com isso, será criado o código $C(H_2)$ para a verificação de isomorfismo.

Figura 26 - Hipergrafo de Dijkstra ($H_2$).



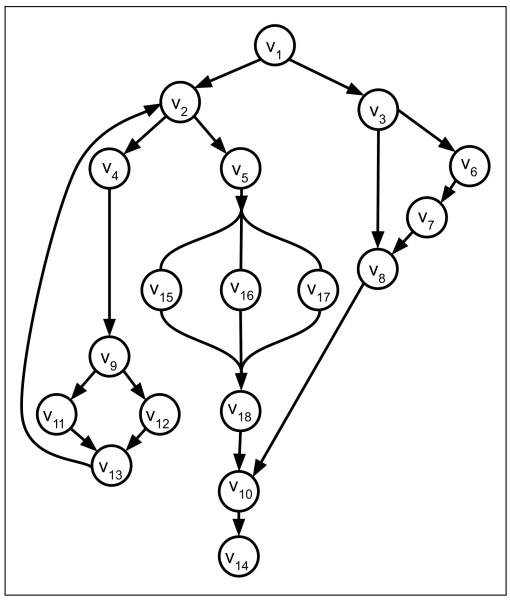

Fonte: O autor, 2023.

$C(v_{22}) = 1;$
$C(v_{21}) = 1; 2; ||C(v_{22});$
$C(v_{21}) = 1; 2; 1;$
$C(v_{10}) = 1;$
$C(v_9) = 1;$
$C(v_8) = 1;$
$C(v_7) = 1;$
$C(v_6) = 1; 9; ||C(v_7); C(v_8); C(v_9); ||$
$C(v_{10});$
$C(v_6) = 1; 9; 1; 1; 1; 1;$
$C(v_5) = 1;$
$C(v_4) = 1; 2; ||C(v_5);$
$C(v_4) = 1; 2; 1;$
$C(v_2) = 1; 3; ||C(v_4)||C(v_6);$
$C(v_2) = 1; 3; 1; 2; 1; 1; 9; 1; 1; 1; 1;$
$C(v_{20}) = 1;$
$C(v_{19}) = 1;$
$C(v_{18}) = 1;$
$C(v_{17}) = 1;$
$C(v_{12}) = 1; 9; ||C(v_{17}); C(v_{18}); C(v_{19}); ||$
$C(v_{20});$
$C(v_{12}) = 1; 9; 1; 1; 1; 1;$
$C(v_{16}) = 1;$
$C(v_{15}) = 1;$
$C(v_{14}) = 1;$
$C(v_{13}) = 1; 6; ||C(v_{14}); C(v_{15}); ||C(v_{16});$
$C(v_{13}) = 1; 6; 1; 1; 1;$
$C(v_{11}) = 1; 2; ||C(v_{13});$
$C(v_{11}) = 1; 2; 1; 6; 1; 1; 1;$
$C(v_2) = 1; 4; ||C(v_{11}); ||C(v_{12});$
$C(v_2) = 1; 4; 1; 2; 1; 6; 1; 1; 1; 1; 9; 1; 1; 1; 1;$
$C(v_1) = 1; 6; ||C(v_2); C(v_3); ||C(v_{21});$

$$C(H_2) = C(v_1) = 1; 6; 1; 4; 1; 2; 1; 6; 1; 1; 1; 1; 9; 1; 1; 1; 1; 1; 3; 1; 2; 1; 1; 9; 1; 1; 1; 1; 1; 2; 1$$

Para o hipergrafo de Dijkstra $H_3$ (Figura 27), a $OT_{H_3}$ será $v_1, v_2, v_4, v_9, v_{11}, v_{12}, v_{13}, v_5, v_{15}, v_{16}, v_{17}, v_{18}, v_3, v_6, v_7, v_8, v_{10}, v_{14}$. Em $H_3$, a sequência de contrações, baseada na ordenação topológica $OT_{H_3}$ será $v_{10}, v_6, v_3, v_5, v_9, v_4, v_2, v_1$, sendo criado o código $C(H_3)$ para a verificação de isomorfismo.

Figura 27 - Hipergrafo de Dijkstra ($H_3$).




Fonte: O autor, 2023.

$C(v_{14}) = 1;$
$C(v_{10}) = 1; 2; ||C(v_{14});$
$C(v_{10}) = 1; 2; 1;$
$C(v_8) = 1;$
$C(v_7) = 1$
$C(v_6) = 1; 2; ||C(v_7;$
$C(v_6) = 1; 2; 1;$
$C(v_3) = 1; 3; ||C(v_6); ||C(v_8)$
$C(v_3) = 1; 3; 1; 2; 1; 1;$
$C(v_{18}) = 1;$
$C(v_{17}) = 1;$
$C(v_{16}) = 1;$
$C(v_{15}) = 1;$
$C(v_5) = 1; 9; ||C(v_{15}); C(v_{16}); C(v_{17}); ||$
$C(v_{18});$
$C(v_5) = 1; 9; 1; 1; 1; 1;$
$C(v_{13}) = 1;$
$C(v_{12}) = 1;$
$C(v_{11}) = 1;$
$C(v_9) = 1; 6; ||C(v_{11}); C(v_{12}); ||C(v_{13});$
$C(v_9) = 1; 6; 1; 1; 1;$
$C(v_4) = 1; 2; ||C(v_9);$
$C(v_4) = 1; 2; 1; 6; 1; 1; 1;$
$C(v_2) = 1; 4; ||C(v_4); ||C(v_5);$
$C(v_2) = 1; 2; 1; 2; 1; 6; 1; 1; 1; 1; 9; 1; 1; 1; 1;$
$C(v_1) = 1; 6; ||C(v_2); C(v_3); ||C(v_{10});$

$C(H_3) = C(v_1) = 1; 6; 1; 2; 1; 2; 1; 6; 1; 1; 1; 1; 9; 1; 1; 1; 1; 1; 3; 1; 2; 1; 1; 1; 2; 1.$

Baseado nos códigos $C(H_1)$, $C(H_2)$ e $C(H_3)$ apresentados como resposta do Algoritmo 5 para os hipergrafos $H_1$, $H_2$ e $H_3$, respectivamente, é possível verificar que $H_1 \cong H_2$, $H_1 \ncong H_3$ e $H_2 \ncong H_3$. Logo, $H_1$ e $H_2$ são isomorfos, $H_1$ e $H_2$ não são isomorfos a $H_3$.

**Teorema 3.2.** *O algoritmo de isomorfismo tem complexidade de tempo igual a $O(n)$.*

*Demonstração.* Recorde-se que $m = O(n)$, pelo Lema 2.5. A construção de uma sequência contrátil de baixo para cima requer $O(n)$ passos. Para cada $v \in V(G)$, seguindo o algoritmo de isomorfismo, $C(v)$ pode ser construído em tempo $|C(v)|$. A ordenação lexicográfica demora tempo linear no comprimento total das cadeias numéricas que serão ordenadas. Segue-se que o algoritmo não requer mais do que $O(n)$ tempo para construir o código $C(G)$ de $G$. □

Como para os grafos de Dijkstra apresentados em [BENTO et al., 2019], o Corolário 3.1 pode ser aplicado aos hipergrafos direcionados de Dijkstra.

**Corolário 3.1.** *Seja $G$ um hipergrafo de Dijkstra. As seguintes afirmativas valem.*

- *Existe a correspondência de um para um entre os números $1's$ de $C(G)$ e os vértices de $G$.*
- *O código $C(G)$ de $G$ é único e é uma representação de $G$.*

# CONCLUSÃO

Dijkstra et al. [DAHL; DIJKSTRA; HOARE, 1972] estabeleceram e popularizaram os fundamentos da *programação estruturada sequencial*.

A programação estruturada é, desde então, encarada por muitos como fator de melhoramento e de manutenção facilitada para o desenvolvimento dos programas de computador.

Bento et al. [BENTO et al., 2019] estudaram os grafos de fluxo de controle dos programas estruturados sequenciais. Eles nomearam esses grafos de *grafos de Dijkstra*.

Em 1972, Hecht e Ullman [HECHT; ULLMAN, 1972] introduziram os *grafos de fluxo redutível*, os quais possuem propriedades necessárias para um grafo ser de Dijkstra. Dado um digrafo de entrada $G = (V, E)$ com $m = |E|$ arcos. Hecht and Ullman [HECHT; ULLMAN, 1972] provaram que o reconhecimento dos grafos de fluxo redutível pode ser feito em tempo $O(m \log m)$. No ano seguinte, Tarjan [TARJAN, 1973] provou que poderia ser feito em tempo quasi linear que é tempo $O(m\ell)$, onde $\ell = \max\{i \mid \log^i m \leq 1\}$.

A condição do impasse (*deadlock*) [TANENBAUM, 1995] deve ser evitada e é fundamental na programação distribuída. Em uma rede, uma estratégia eficiente para confrontar o impasse é interromper um número mínimo de processos ou canais. Na combinatória [GAREY; JOHNSON, 1979] os problemas de decisão que estudam essa estratégia são chamados de CONJUNTO DE VÉRTICES DE RETROALIMENTAÇÃO e CONJUNTO DE ARCOS DE RETROALIMENTAÇÃO.

Em 1979, Shamir [SHAMIR, 1979] provou que CONJUNTO DE VÉRTICES DE RETROALIMENTAÇÃO é polinomial para os grafos de fluxo redutível. Em 1988, Ramachandran [RAMACHANDRAN, 1988] resolveu o problema CONJUNTO DE ARCOS DE RETROALIMENTAÇÃO para grafos de fluxo redutível em tempo polinomial.

Bento et al. [BENTO et al., 2019] mostraram um algoritmo linear para o reconhecimento e para o isomorfismo dos grafos de Dijkstra.

Guedes [GUEDES, 2001] propôs os hipergrafos Serial-Paralelo-Disjunção (HSPD) que usava hipergrafos para modelar os *hipergrafos de fluxo redutíveis*, os quais proveem propriedades necessárias para um programa paralelo ser estruturado. Guedes et al. [FARIA; GUEDES; MARKENZON, 2021] mostraram que CONJUNTO DE VÉRTICES DE RETROALIMENTAÇÃO é NP-completo para hipergrafos de fluxo redutíveis.

Dijkstra [DIJKSTRA, 1965] introduziu um modelo paralelo através da estrutura *PARBEGIN/PAREND*, onde o PARBEGIN inicia um bloco com $p \geq 2$ processos executados de forma paralela e que após sua execução retorna ao ponto PAREND correspondente.

Neste trabalho, apresentamos a classe dos *hipergrafos direcionados de Dijkstra* que estende naturalmente os grafos de Dijkstra [BENTO et al., 2019; DAHL; DIJKSTRA; HOARE, 1972] possuindo a estrutura paralela PARBEGIN/PAREND, também do modelo paralelo de Dijkstra [DIJKSTRA, 1965], e que é um hipergrafo de fluxo redutível [GUEDES, 2001].

Além da definição, contribuímos com esse trabalho provando que o reconhecimento e o isomorfismo da classe dos hipergrafos de Dijkstra pode ser feito em tempo linear.

Elencamos como trabalhos futuros:

1. Propor um modelo de programação paralela estruturada mais geral que estenda os hipergrafos de Dijkstra e que use o conceito dos HSPD apresentado por Guedes [GUEDES, 2001].

2. O estudo dos hipergrafos Serial-Paralelo-Disjunção e seus relacionamentos com os hipergrafos de Dijkstra.

3. Determinar o status de complexidade de tempo de CONJUNTO DE ARCOS DE RETROALIMENTAÇÃO para hipergrafos de fluxo redutiveis.

4. Determinar o status de complexidade de tempo de CONJUNTO DE ARCOS DE RETROALIMENTAÇÃO para hipergrafos Serial-Paralelo-Disjunção.

5. Investigar um algoritmo para CONJUNTO DE VÉRTICES DE RETROALIMENTAÇÃO linear para grafos de Dijkstra.

6. Investigar um algoritmo para CONJUNTO DE VÉRTICES DE RETROALIMENTAÇÃO linear para hipergrafos de Dijkstra.

# REFERÊNCIAS


BENTO, Lucila Maria de Souza et al. Dijkstra graphs. *Discrete Applied Mathematics*, Elsevier, v. 261, p. 52–62, 2019.

BERGE, C. Graphs and hypergraphs, dumond, paris. *English translation, North-Holland, Amsterdam*, 1970.

BONDY, John Adrian; MURTY, Uppaluri Siva Ramachandra. *Graph theory*. [S.l.]: Springer Publishing Company, Incorporated, 2008.

BRETTO, Alain. *Hypergraph Theory: An Introduction*. [S.l.]: Springer, 2013.

CORMEN, Thomas et al. Algoritmos-teoria e prática (3a. ediçao). *Editora Campus*, 2012.

DAHL, Ole-Johan; DIJKSTRA, Edsger Wybe; HOARE, Charles Antony Richard. *Structured programming*. [S.l.]: Academic Press Ltd., 1972. 1-81 p.

DIJKSTRA, Edsger W. Cooperating sequential processes. In: GENUYS, F. (Ed.). *The origin of concurrent programming: from semaphores to remote procedure calls*. [S.l.]: Academic Press, New York, New York, 1965.

DIJKSTRA, Edsger Wybe. Letters to the editor: go to statement considered harmful. *Communications of the ACM*, ACM New York, NY, USA, v. 11, n. 3, p. 147–148, 1968.

DIJKSTRA, Edsger W. The structure of"the"-multiprogramming system. *Communications of the ACM*, v. 11, n. 5, p. 341–346, 1968.

FARIA, Luerbio; GUEDES, André LP; MARKENZON, Lilian. On feedback vertex set in reducible flow hypergraphs. *Procedia Computer Science*, Elsevier, v. 195, p. 212–220, 2021.

GAO, Yue et al. Hypergraph learning: Methods and practices. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, IEEE, v. 44, n. 5, p. 2548–2566, 2020.

GAREY, Michael R; JOHNSON, David S. Computers and intractability. *A Guide to the*, 1979.

GROSS, Jonathan L.; YELLEN, Jay. *Graph Theory and its aplications*. 2ª. [S.l.]: Editora Chapman & Hall/CRC, 2006.

GUEDES, André Luiz Pires. *Hipergrafos Direcionados*. 2001. 136 p. Tese (Doutorado em Teoria dos Grafos) — Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia (COPPE) - Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2001.

HANSEN, Per Brinch. Structured multiprogramming. *Communications of the ACM*, ACM New York, NY, USA, v. 15, n. 7, p. 574–578, 1972.

_____. Concurrent programming concepts. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 5, n. 4, p. 223–245, 1973.

HECHT, Matthew S.; ULLMAN, Jeffrey D. Flow graph reducibility. In: *Proceedings of the fourth annual ACM symposium on Theory of computing*. [S.l.: s.n.], 1972. p. 238–250.

_____. Characterizations of reducible flow graphs. *Journal of the ACM (JACM)*, ACM New York, NY, USA, v. 21, n. 3, p. 367–375, 1974.

HOARE, Charles Antony Richard. Communicating sequential processes. *Communications of the ACM*, ACM New York, NY, USA, v. 21, n. 8, p. 666–677, 1978.

KNUTH, Donald E. Structured programming with go to statements. *ACM Computing Surveys (CSUR)*, ACM New York, NY, USA, v. 6, n. 4, p. 261–301, 1974.

KNUTH, Donald Ervin. *The Art of Computer Programming*. 3ª. [S.l.]: Addison-Wesley, 1997. v. 1.

KNUTH, Donald E.; FLOYD, Robert W. Notes on avoiding "go to" statements. *Information processing letters*, Elsevier, v. 1, n. 1, p. 23–31, 1971.

RAMACHANDRAN, Vijaya. Finding a minimum feedback arc set in reducible flow graphs. *Journal of Algorithms*, Elsevier, v. 9, n. 3, p. 299–313, 1988.

SHAMIR, Adi. A linear time algorithm for finding minimum cutsets in reducible graphs. *SIAM Journal on Computing*, SIAM, v. 8, n. 4, p. 645–655, 1979.

SZWARCFITER, Jayme Luiz. Teoria computacional de grafos, 1ª Ediçao. *Editora Elsevier - Série Sociedade Brasileira de Computação*, 2018.

TANENBAUM, A.S. *Distributed Operating Systems*. Pearson Education, 1995. (Always learning). ISBN 9788177581799. Disponível em: <https://books.google.com.br/books?id=l6sDRvKvCQ0C>.

TARJAN, Robert. Testing flow graph reducibility. In: *Proceedings of the fifth annual ACM symposium on Theory of computing*. [S.l.: s.n.], 1973. p. 96–107.

## ÍNDICE