**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**CAIO MACEDO**
**KAÍQUE MEDEIROS LIMA**
**IAN BATISTA FORNAZIERO**

# MEMORIAL DESCRITIVO
## Algoritmos em Grafos

**SANTA HELENA**
**2025/2**

**CAIO MACEDO**
**KAÍQUE MEDEIROS LIMA**
**IAN BATISTA FORNAZIERO**

# MEMORIAL DESCRITIVO
## Algoritmos em Grafos

**Descriptive Memorial - Graph Algorithms**

Trabalho de Conclusão de Disciplina de Graduação apresentado como requisito para conclusão da disciplina de Algoritmos em Grafos do Curso de Bacharelado em Ciência da Computação da Universidade Tecnológica Federal do Paraná

Docente: Dra. Leiliane Pereira de Rezende

**SANTA HELENA**
**2025/2**

**LISTA DE ALGORITMOS**

**LISTA DE FIGURAS**

## LISTAGEM DE CÓDIGOS FONTE

## LISTA DE ABREVIATURAS E SIGLAS

### Siglas

| | |
|---|---|
| ACID | Atomicidade, Consistência, Isolamento e Durabilidade |

**SUMÁRIO**

# 1 INTRODUÇÃO

A estrutura de dados grafos é aplicada em diversas áreas do conhecimento, as quais podem ser esquematizadas via um conjunto de conexões entre pares de objetos. A descrição do problema a ser trabalhado em todo o documento é dada na Seção 1.1. A composição do restante do documento é descrita na Seção 1.2.

## 1.1 Descrição do Problema

O problema a ser modelado diz respeito a logística de transporte de uma empresa qualquer, que utiliza veículos automotivos como caminhões e carros. Esse meio de transporte tem como caracteristica a alta emissão de gases de efeito estufa (GEEs), que colaboram a degradação do meio ambiente. Nesse contexto, para amenizar o problema, deseja-se encontrar a melhor rota de um ponto a outro de forma sustentável. Por exemplo, se a empresa deseja realizar uma entrega de um ponto de distribuição a outro, a escolha da rota modelada por meio dos grafos levará em a emissão de GEEs do veículo, de acordo com o combustível utilizado e o consumo por quilometro. A melhor rota será aquela que possui o mínimo de emissão possível. Dessa forma, os impactos ao meio ambiente são minimizados, evitando a intensificação do aquecimento global. É importante destacar que, o caminho com menos emissão de GEEs pode também ser a rota mais econômica. A equação que define a emissão de GEEs por Litro é dada por:

$$EF_{CO2,L} = \rho \ \times fc \ \times OF \ \times \frac{44}{12} \tag{1.1}$$

Onde,

- O $\rho$ é a densidade do combustível
- o $fc$ é a fração mássica de carbono no combustível ($kgC/kgcombustvel$)
- OF é o fator de oxidação ($\approx 0.99 - 10$; a fração do carbono efetivamente oxidada a $CO_2$)
- $\frac{44}{12}$ converte $C \rightarrow CO_2$ (massa molar).

A emissão total é dada pela multiplicação de emissão por litro e os litros.

## 1.2 Estrutura do Trabalho

A estrutura de dados grafo pode ser representada por diferentes meios, seja ela na visão computacional ou não. O capítulo 2 apresenta tanto as representações não computacionais quanto as computacionais mais usadas nos algoritmos em grafos.

No contexto da estrutura de dados grafo, diversas são as terminologias e tipagens existentes e, na sua grande maioria, independentes se o grafo é ou não orientado. No Capítulo 3, as principais terminologias e tipos que o grafo apresentado no Capítulo 1 contempla são descritos.

Para facilitar ou solucionar alguns problemas modelados por meio da estrutura de grafos, algumas operações matemática nos seus conjuntos de vértices e arestas são necessárias. As principais operações são exemplificadas no Capítulo 4.

A busca é umas das técnicas mais aplicadas na solução de problemas algorítmicos em grafos considerados eficientes. O Capítulo 5 apresenta tanto a busca em largura quanto a busca em profundidade, bem como uma aplicação das duas para obter os componentes fortemente conexos de um grafo.

Dentre as subestruturas de grafo que oferecem solução para problemas aplicados, os caminhos se destacam especialmente pelo potencial associado aos problemas de trânsito, transporte, localização em sistemas discretos, dentre outros. Quando a definição de distância é associada aos caminhos, surgem os problemas de caminho mínimo. O Capítulo 6 apresenta o cálculo do caminho mínimo tanto para grafos não valorados quanto para valorados.

Problemas de interligação (comunicações, redes de luz, água, esgotos, etc.) podem ser solucionados pela obtenção da árvore de cobertura de peso mínimo quando existe o interesse em se proceder à interligação de todos os pontos atendidos com o consumo mínimo de meios. O Capítulo 7 apresenta os dois algoritmos mais conhecidos na literatura para a obtenção de uma árvore de cobertura de peso mínimo.

A modelagem e solução de um problema de ciclos por Leonard Euler, durante o século XVIII, é responsável por definir os fundamentos da teoria dos grafos. O teorema definido por Euler demostra se é possível ou não, a partir de algum ponto do grafo, percorrer todas as arestas uma única vez e voltar ao ponto de partida. No Capítulo 8, o grafo apresentado geometricamente na Figura 2.1 é verificado se satisfaz ou não o teorema de Euler. Caso, sim, o grafo é dito Euleriano. Caso contrário, uma eulerização é aplicada para obter um ciclo com o menor número de arestas/arcos repetidos.

Problemas de roteamento como comunicação, logística, e, com ligeiras modificações, perfuração de placas de circuito impresso, entre outros, têm com a mesma premissa: a partir de algum ponto, percorrer todos os outros uma única vez e voltar ao ponto de partida. Semelhante ao problema de grafos eulerianos, este problema, nomeado de ciclo hamiltoniano, não conhece uma condição necessária e suficiente que seja trivial para verificar a existência ou não deste ciclo. O Capítulo 9 aplica o teorema mais importante, o teorema de Dirac, o teorema de Ore que é usado para deduzí-lo, além do teorema de Bondy e Chvátal no grafo estudado. Adicionalmente, mostra a (in)existência de um ciclo hamiltoniano também no grafo estudado.

O Capítulo 10 apresenta as principais definições do problema de emparelhamento.

Este problema pode ser aplicado em problemas de atribuição de pessoal, de casamentos, de construção de amostras, entre outros. A ideia geral é formar o maior conjunto de pares de vértices adjacentes somente entre si.

A planaridade de um projeto é questão importante em diversas situações práticas, como circuitos, cartografia, malhas de transporte terrestre e aéreo, construção de viadutos, entre outros. O Capítulo 11 verifica a planaridade do grafo estudado.

Problemas de competição/conflito por algum recurso, como a resolução de quebra-cabeças de Sudoku, podem ser solucionados pelo problema de coloração de grafos. O Capítulo 12 apresenta a aplicação de um algoritmo guloso para uma solução e, por meio do conceito das cadeias de Kempe, verifica se existe outra solução com um número cromático menor.

## 2 Representações de um Grafo

Um grafo pode ser representado por diferentes meios, seja ele na visão computacional ou não. Considerando a visão não computacional, o grafo pode ser representado tanto matematicamente quanto geometricamente. O primeiro modo é descrito na Seção 2.1 e o segundo na Seção 2.2, ambos considerando a descrição do grafo dado no Capítulo 1.

A representação por meio da visão computacional pode ser dada por três meios: matriz de adjacência, lista de adjacência e matriz de incidência. A descrição de cada um destes meios é dada, respectivamente, nas Seções 2.3, 2.4, e 2.5. As considerações acerca da eficiência computacional, tanto em relação ao tempo de processamento quanto ao espaço usado de memória, são dadas na Seção 2.6.

### 2.1 Matemática

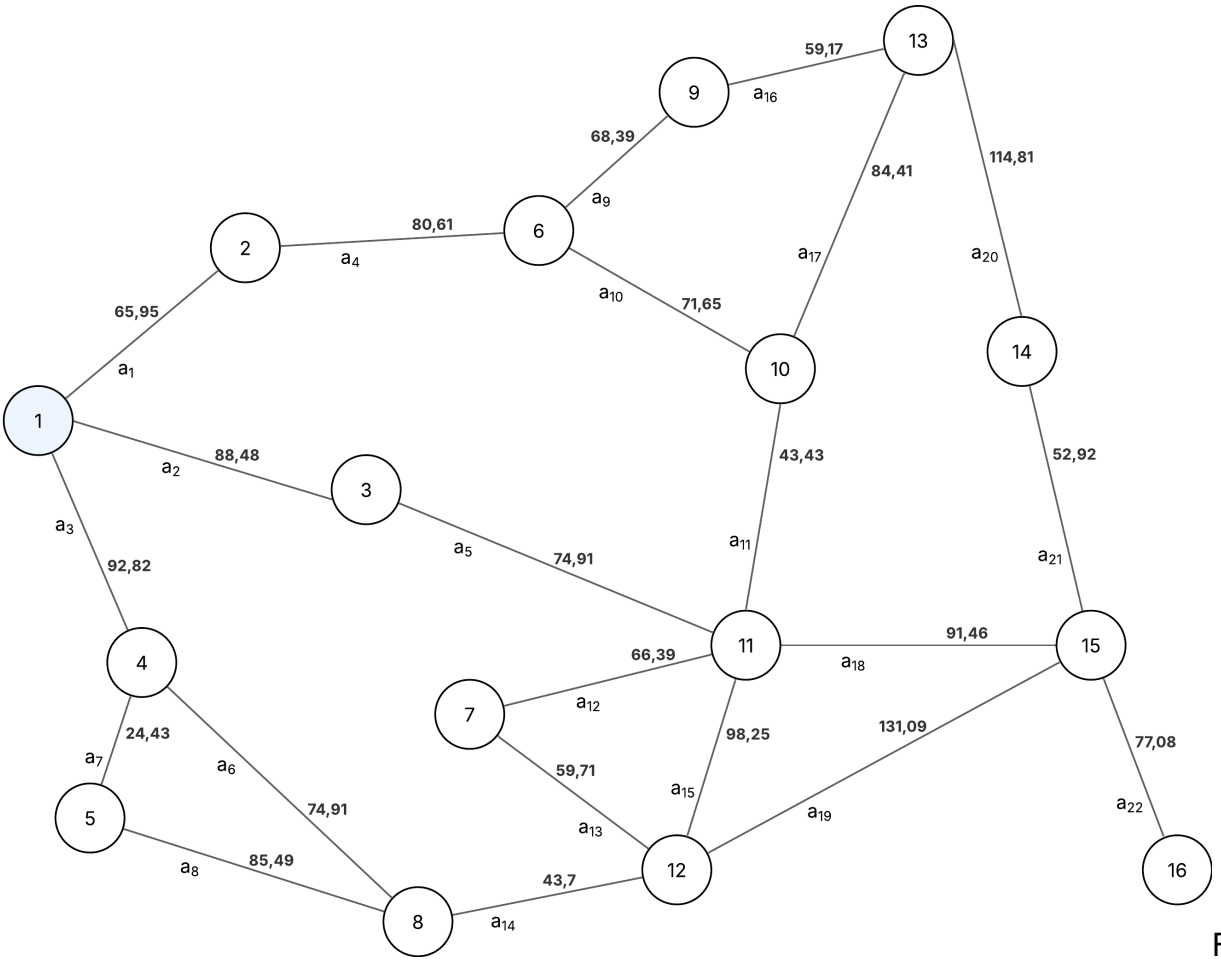
A representação matemática do nosso problema é:
$G_1 = \{V_1, A_1\}$
$V_1 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}$
$A_1 = \{a_1, a_2, a_3, a_4, a_5, a_6, a_7, a_8, a_9, a_{10}, a_{11}, a_{12}, a_{13}, a_{14}, a_{15}, a_{16}, a_{17}, a_{18}, a_{19}, a_{20}, a_{21}, a_{22}\}$
onde,
$a_1 = (1, 2)$; $a_2 = (1, 3)$; $a_3 = (1, 4)$; $a_4 = (2, 6)$; $a_5 = (3, 11)$; $a_6 = (4, 7)$; $a_7 = (4, 5)$; $a_8 = (5, 8)$; $a_9 = (6, 9)$; $a_{10} = (6, 10)$; $a_{11} = (10, 11)$; $a_{12} = (7, 11)$; $a_{13} = (7, 12)$; $a_{14} = (8, 12)$; $a_{15} = (11, 12)$; $a_{16} = (9, 13)$; $a_{17} = (13, 10)$; $a_{18} = (11, 15)$; $a_{19} = (12, 15)$; $a_{20} = (13, 14)$; $a_{21} = (14, 15)$; $a_{22} = (15, 16)$.
Cada vértice representa uma localidade na região da cidade de santa helena sendo elas:

- 1: Santa Helena
- 2: Entre Rios do Oeste
- 3: Diamante D'oeste
- 4: Missal
- 5: Itaipulândia
- 6: São José das Palmeiras
- 7: Ramilândia
- 8: Medianeira

- 9: Ouro Verde do Oeste
- 10: São Pedro do Iguaçu
- 11: Vera Cruz do Oeste
- 12: Matelândia
- 13: Toledo
- 14: Cascavel
- 15: Santa Tereza do Oeste
- 16:Lindoeste

## 2.2 Geométrica

Figura 2.1: Representação Geométrica do Grafo




Fonte: Criação própria

## 2.3 Matriz de Adjacência

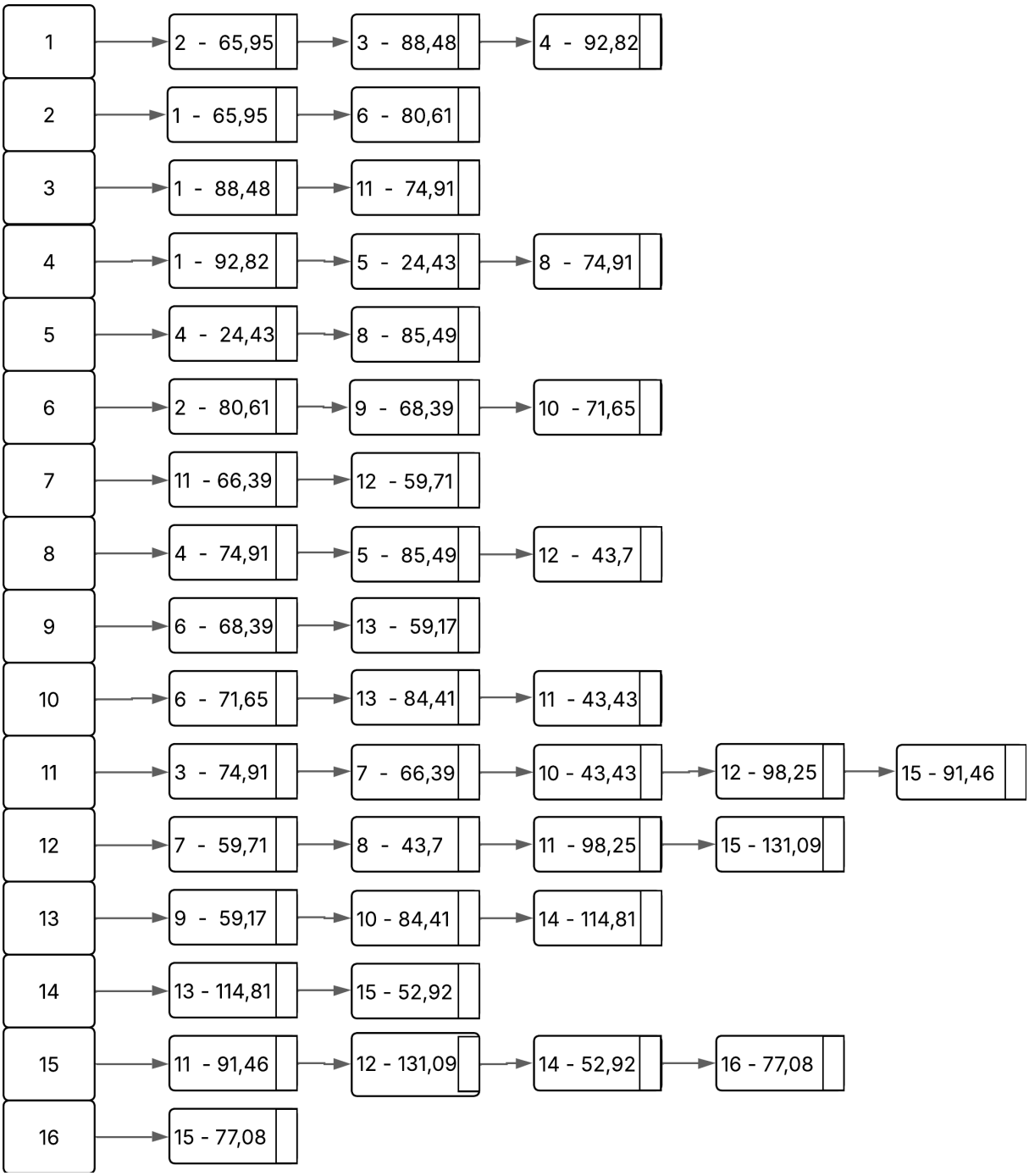
A matriz de Adjacência do nosso problema é representada por (Equação 2.1)

$$\begin{bmatrix}
0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0
\end{bmatrix} \tag{2.1}$$

## 2.4 Lista de Adjacência

A lista de Adjacência do nosso problema é representada na figura 2.2:

Figura 2.2: Representação Lista de Adjacência




Fonte: Criação própria

## 2.5 Matriz de Incidência

A matriz de Incidência do nosso problema é (Equação 2.2):

$$\begin{bmatrix}
1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 1 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1
\end{bmatrix} \tag{2.2}$$

### 2.6 Considerações de Eficiência

Para o nosso grafo, acredita-se que, a representação com melhor eficiência, ou seja, menor custo computacional é a lista de adjacências, por termos múltiplos vértices e arestas, e por seu O(|V| + |X|) = O(38). Calculo de todas as representações:

- Matriz de Adjacências:

$$O(V^2) = O(16^2) = O(256) \tag{2.3}$$

- Lista de Adjacências:

$$O(|V| + |X|) = O(16 + 22) = O(38) \tag{2.4}$$

- Matriz de Incidência:

$$O(V \times X) = O(16 \times 22) = O(352) \tag{2.5}$$

# 3 Definições em um Grafo

Como a estrutura de dados grafo é uma estrutura heterogenia, definições são necessárias para compreender partes ou o todo da estrutura. Na Seção 3.1, as principais terminologias, considerando o grafo representado graficamente na Figura 2.1, são apresentadas. Enquanto, na Seção 3.2, todos os tipos os quais o grafo estudado contempla são apresentados.

## 3.1 Terminologias

A terminologia se inicia com a definição da orientação e adjacências do grafo. O grafo em questão, demonstrado na figura, não é orientado, tendo em vista sua representação (sem setas de direcionamento), bem como a sua ideia de representar apenas avenidas e/ou rodovias, que são sempre duas mãos. A adjacência se baseia em 5 fatores: adjacência de vértices, arestas, incidência, paralelismo e laço.

A adjacência de vértices é entendida quando dois vértices são ligados entre si com uma ou mais arestas. No caso do grafo não direcionado, dois vértices adjacentes são sempre adjacentes entre si. Exemplo: $V_6$ é adjacente a $V_9$, da mesma maneira $V_9$ é adjacente a $V_6$.

A adjacência de arestas é parecida, mas dessa vez, arestas são adjacentes quando compartilham um mesmo vértice. Exemplo: $A_{20}$ é adjacente a $A_{17}$, pois compartilham o mesmo vértice $V_{13}$.

As incidências são a notação para uma aresta que liga dois vértices. Exemplo: $A_5$ é incidente em $V_3$ e $V_{11}$ ao mesmo tempo devido ao grafo ser não orientado.

Arestas paralelas são duas arestas que ligam o mesmo par de vértices. No grafo em questão não existem arestas paralelas.

Por fim, o laço é uma aresta que liga o vértice nele mesmo. No grafo em questão não existe laço, tendo em vista que as arestas representam avenidas que vão de uma cidade e/ou distrito até outra, o que torna impensável uma aresta que liga um vértice a si próprio.

Terminando de falar das adjacências, tem-se a aplicação multívoca $\Gamma$, que representa os sucessores de um vértice, ou de forma mais simples, os vértices que são ligados ao vértice em questão.

$$\Gamma(1) = (2, 3, 4) \qquad \Gamma(9) = (6, 13)$$
$$\Gamma(2) = (1, 6) \qquad \Gamma(10) = (6, 11, 13)$$
$$\Gamma(3) = (1, 11) \qquad \Gamma(11) = (3, 7, 10, 12, 15)$$
$$\Gamma(4) = (1, 5, 8) \qquad \Gamma(12) = (7, 8, 11, 15)$$
$$\Gamma(5) = (4, 8) \qquad \Gamma(13) = (9, 10, 14)$$
$$\Gamma(6) = (2, 9, 10) \qquad \Gamma(14) = (13, 15)$$
$$\Gamma(7) = (11, 12) \qquad \Gamma(15) = (11, 14, 16)$$
$$\Gamma(8) = (4, 5, 12) \qquad \Gamma(16) = (15)$$

Existe também a aplicação gama inversa, que seria para os vértices que se ligam em $V$ mas $V$ não se liga nos vértices. No entanto, só é possível fazer essa aplicação em grafos direcionados, pois eles têm direcionamento. Já no grafo não direcionado, toda aresta que vai, volta.

Junto da aplicação, é possível calcular o grau de cada vértice, que em um grafo não direcionado é basicamente o número de vértices ligados ao $V$ em questão (evidenciado na aplicação gama). Em um grafo direcionado, o cálculo é basicamente o mesmo, só que somando também com a aplicação inversa.

$$V_1 = 3,\ V_2 = 2$$
$$V_3 = 2,\ V_4 = 3$$
$$V_5 = 2,\ V_6 = 3$$
$$V_7 = 2,\ V_8 = 3$$
$$V_9 = 2,\ V_{10} = 3$$
$$V_{11} = 5,\ V_{12} = 4$$
$$V_{13} = 3,\ V_{14} = 2$$
$$V_{15} = 3,\ V_{16} = 1$$

Ordem do grafo $= 16$ Tamanho do grafo $= 22$

Continuando na ligação entre os vértices, podemos gerar conjuntos de ligações chamadas de cadeia e ciclo. No caso do grafo não direcionado, uma cadeia é um conjunto de ligações no grafo que levam de um $V_X$ para um $V_Y$.

Exemplo de cadeia:

$$C(V_1 \to V_{16}) = ((V_1, V_4), (V_4, V_8), (V_8, V_{12}), (V_{12}, V_{15}), (V_{15}, V_{16})).$$

E o ciclo é uma cadeia, mas o último vértice visitado é o mesmo que se inicia, ou seja, um conjunto de ligações que levam de $V_X$ para $V_X$.

Exemplo de ciclo:

$$C(V_6 \to V_6) = ((V_6, V_9), (V_9, V_{13}), (V_{13}, V_{10}), (V_{10}, V_6)).$$

Podemos também classificar as cadeias e ciclos em alguns tipos, dentre eles Elementar e Simples. No caso, a cadeia/ciclo elementar é quando todos os vértices são distintos, ou seja, a cadeia ou ciclo passa pelo vértice apenas uma vez. Já a cadeia/-ciclo simples é quando a cadeia passa por uma aresta apenas uma vez.

Exemplos de cadeia/ciclo elementar:

$$C(V_4 \to V_4) = ((V_4, V_5), (V_5, V_8), (V_8, V_4))$$

$$C(V_7 \to V_7) = ((V_7, V_{12}), (V_{12}, V_{15}), (V_{15}, V_{11}), (V_{11}, V_7))$$

Também é possível classificar em cadeia/ciclo euleriano e hamiltoniano. Euleriano é quando todas as arestas são percorridas exatamente uma vez, e hamiltoniano é quando todos os vértices são percorridos exatamente uma vez.

No grafo utilizado não existe possibilidade de uma cadeia/ciclo desses dois tipos, pois qualquer caminho teria de passar por alguma aresta ou vértice mais de uma vez ou deixar alguma aresta/vértice de fora do caminho.

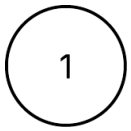
### 3.2 Tipos de Grafos

O grafo modelado neste trabalho é **não orientado, ponderado e simples**. A seguir, encontra-se todos possíveis tipos que um grafo pode assumir

***Grafo nulo:*** um grafo em que seu conjunto de vértices e seu conjunto de arestas são vazios.

***Grafo Trivial ou Singleton:*** grafo com apenas um vértice.

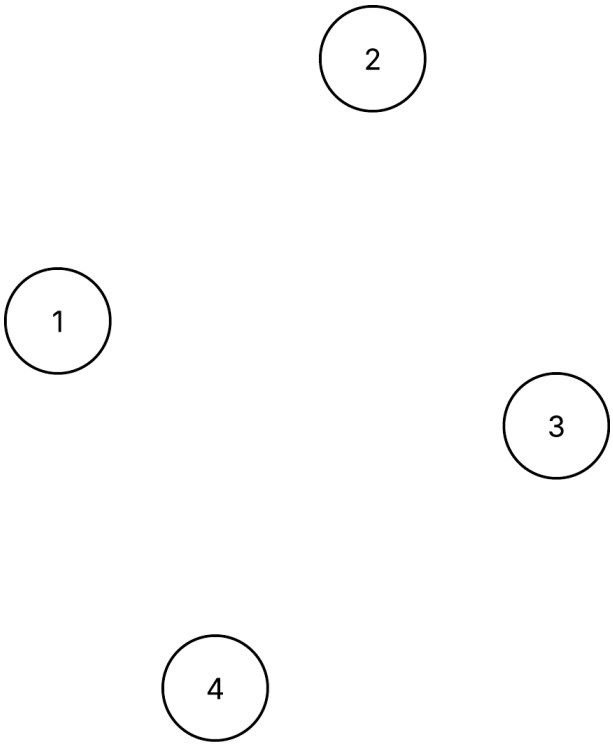
Figura 3.1: Singleton




Fonte: Criação própria

***Grafo Vazio:*** Grafo que não possui nenhuma aresta, apenas vértices.

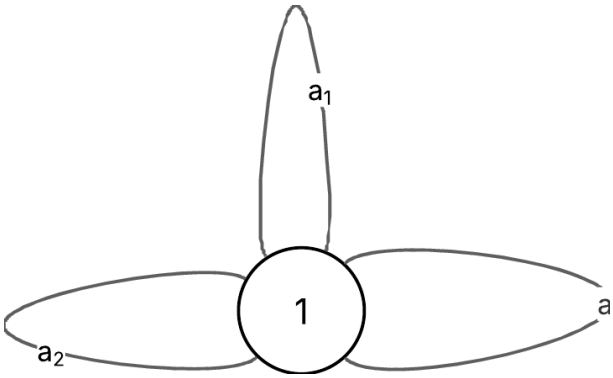
Figura 3.2: Grafo Vazio




Fonte: Criação própria

***Buquê:*** um vértice com n laços.

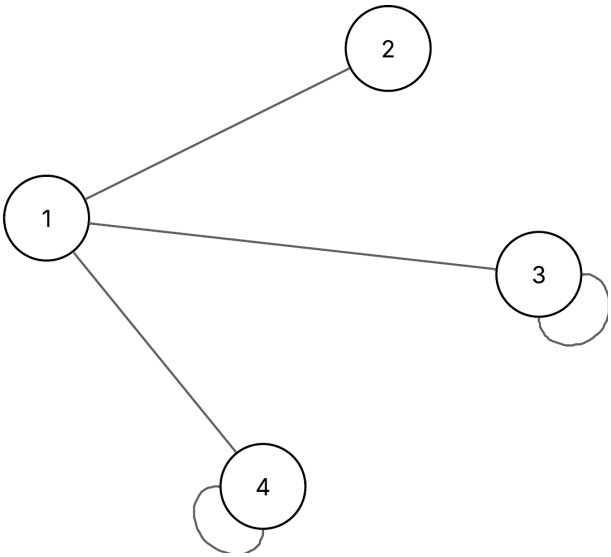
Figura 3.3: Buquê




Fonte: Criação própria

***Pseudografo:*** contém pelo menos um laço.

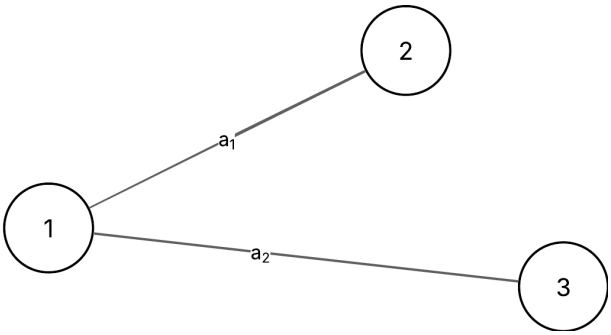
Figura 3.4: Pseudografo




Fonte: Criação própria

***Grafo Reflexivo:*** Pseudografo em que cada vértice possui um laço associado.

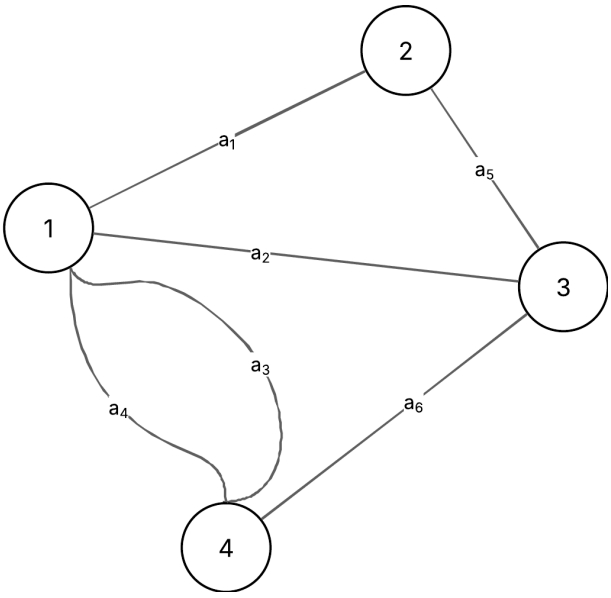
Figura 3.5: Grafo Reflexivo




Fonte: Criação própria

***Multigrafo:*** Um grafo não orientado sem laços e possui no mínimo duas arestas paralelas.

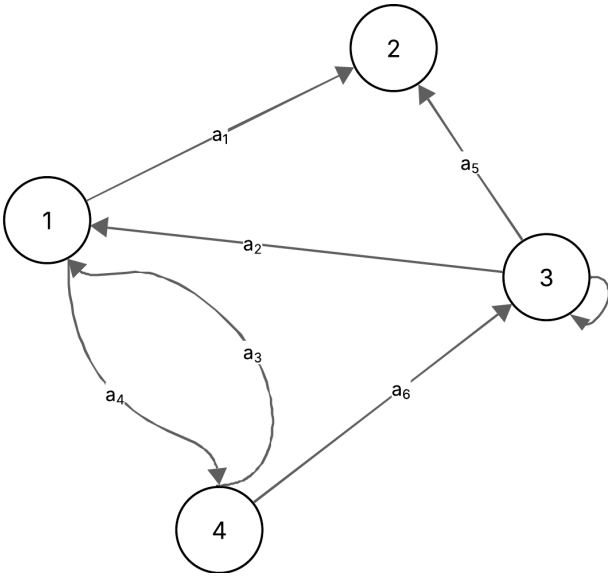
Figura 3.6: Multigrafo




Fonte: Criação própria

***Multigrafo Direcionado:*** Um grafo orientado que possui laços e arestas paralelas.

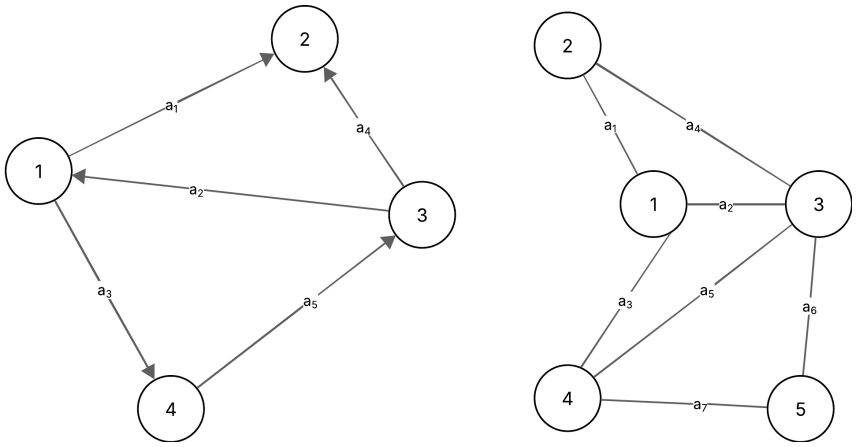
Figura 3.7: Multigrafo Direcionado




Fonte: Criação própria

***Grafo Simples:*** Grafo sem laços e arestas paralelas.

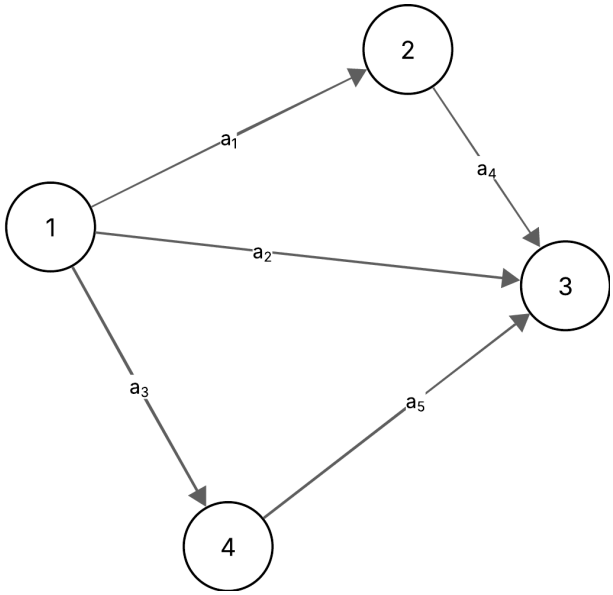
Figura 3.8: Grafo Simples




Fonte: Criação própria

***Grafo acíclico:*** Grafo sem ciclos ou circuitos.

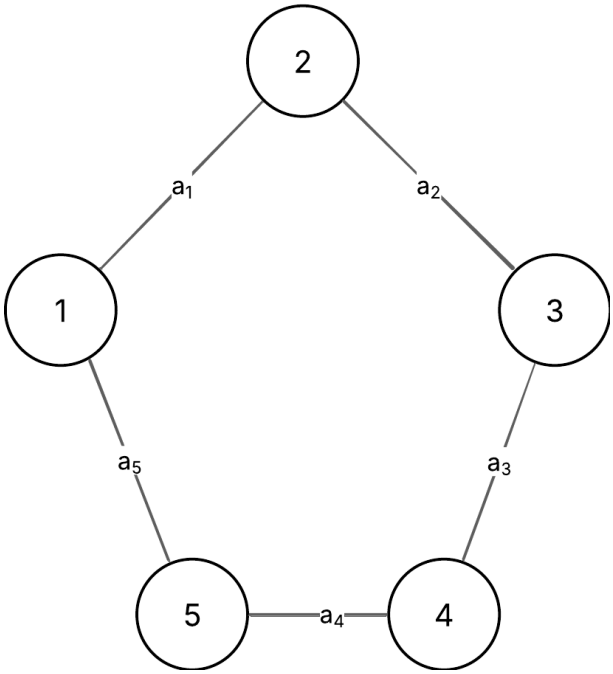
Figura 3.9: Grafo Acíclico




Fonte: Criação própria

***Grafo ciclo:*** É um grafo simples não orientado regular, $C_n$ com mais de dois vértices, onde forma um ciclo, ao partir de um vértice, é possível chegar nele novamente ao passar por todos os outros.

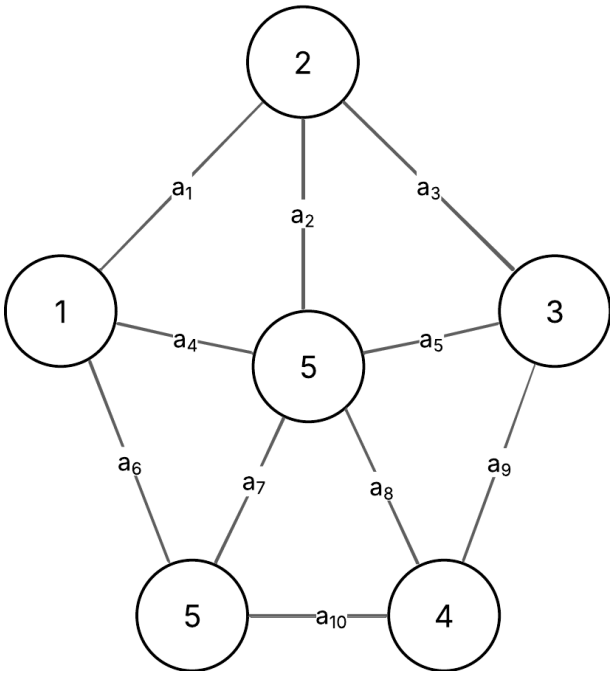
Figura 3.10: Grafo Cíclico




Fonte: Criação própria

***Grafo roda:*** É um grafo não orientado simples, $W_n$, com $n + 1$ vértices, onde n é maior que 2, onde existe um vértice $V_n + 1$ responsável por conectar os demais vértices entre si.

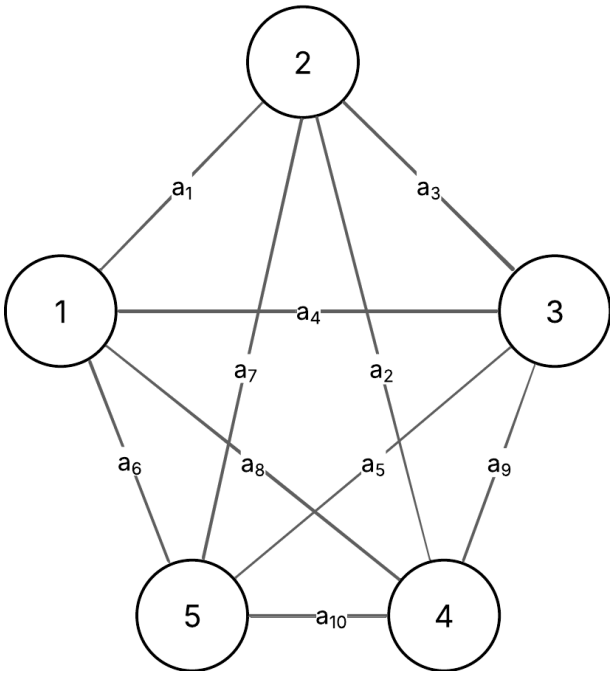
Figura 3.11: Grafo Roda:




Fonte: Criação própria

***Grafo regular:*** Grafo, $K_v$, onde todos os vértices possuem o mesmo grau, ou seja, mesmo número de vértices o que resulta em um grafo onde todos se conectam.

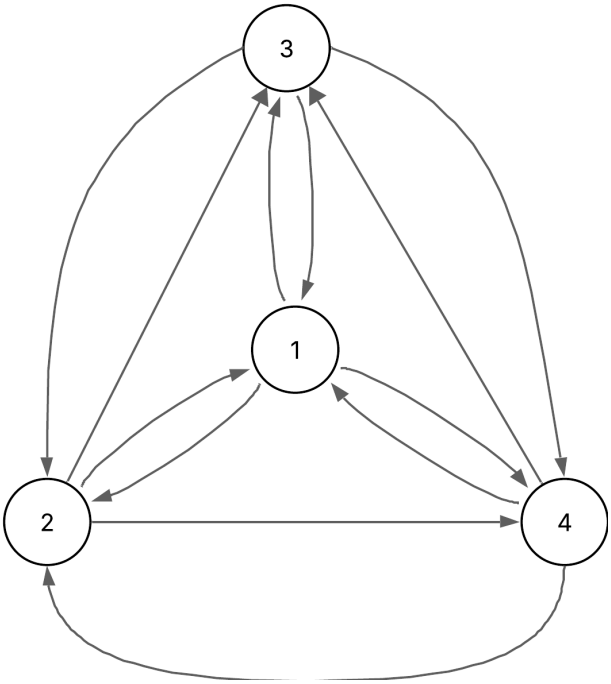
Figura 3.12: Grafo Regular




Fonte: Criação própria

***Grafo simétrico:*** Um grafo orientado, onde, para cada arco $(v_i, v_j) \in X$ existe um arco $(v_j, v_i) \in X$, em outras palavras, cada vértice possui um arco nos dois sentidos, formando simetria entre os arcos

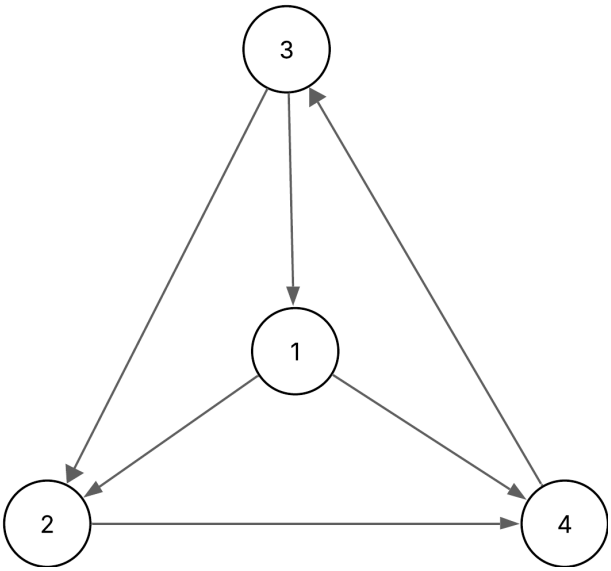
Figura 3.13: Grafo Simétrico




Fonte: Criação própria

***Grafo assimétrico:*** Um grafo orientado, onde, para cada arco $(v_i, v_j) \in X$ não existe um arco $(v_j, v_i) \in X$, em outras palavras, cada vértice possui apenas um arco em apenas um sentido

Figura 3.14: Grafo assimétrico

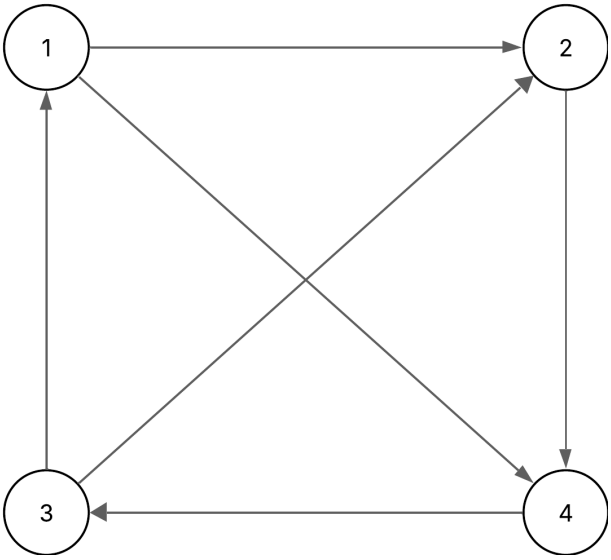



Fonte: Criação própria

***Torneio:*** um grafo orientado que, para cada par $(v_i, v_j) \in V$ com $v_i \neq v_j$, $(v_i, v_j) \in E$ é um arco ou $(v_j, v_i) \in E$ é um arco, mas não ambos, fazendo com que exista apenas

um arco entre dois vértices.

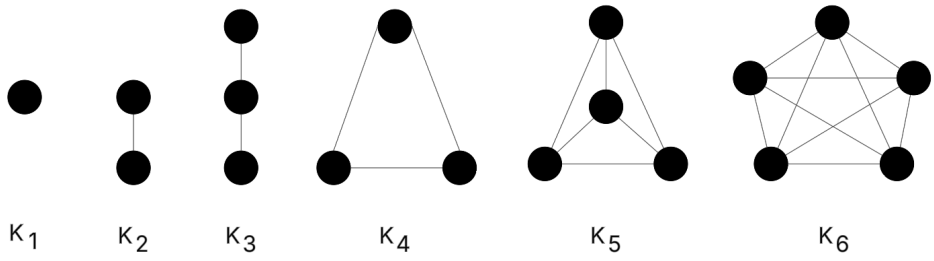
Figura 3.15: Torneio




Fonte: Criação própria

***Completo:*** Grafo não direcionado simples onde todo par de vértices $v_i, v_j \in V$ é ligado por uma aresta, sendo essa $(v_i, v_j)$, sendo um grafo simples que contém o número maximo possível de arestas.

Figura 3.16: Grafo Completo



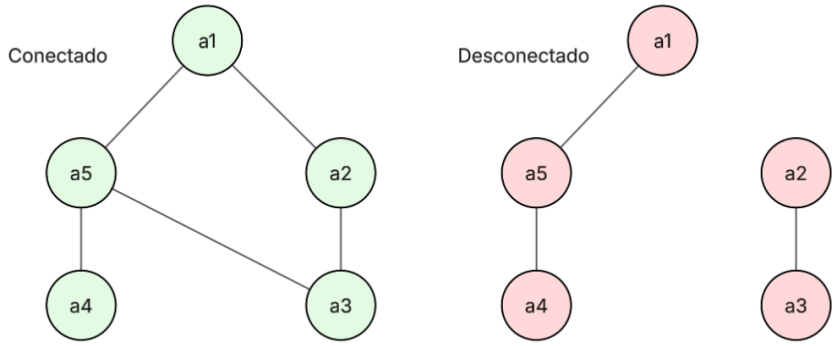

Fonte: Criação própria

Esse tipo de grafo possui algumas observações:

- Ele também é um grafo regular $K_{|V|-1}$, pois todos os vértices possuem o grau de $(|V| - 1)$.
- O grafo orientado é completo apenas se todo o par ordenado de vértices distintos é um arco.
- O grafo orientado completo com $V$ vértices tem exatamente $|V| \times (|V| - 1)$ arcos.

***Grafo conexo:*** Para todos os pares de vértices $\{v_i, v_j \subset V\}$, existe uma cadeia de arcos e arestas de $v_i$ para $v_j$

Figura 3.17: Grafo Conexo



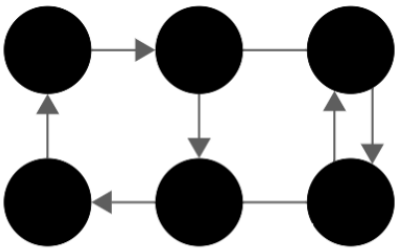

Fonte: Criação própria

Observações:

- O grafo $G(V, X)$ é chamado de desconexo, caso exista, pelo menos, um par de vértices que não é ligado por nenhuma cadeia.
- O grafo $G(V, X)$ desconexo é formado por, no mínimo dois subgrafos conexos, disjuntamente em relação aos vértices e maximais em relação à inclusão desses.
- Um vértice é de corte caso sua remoção, juntamente das arestas conectadas, provoca a redução da conexividade do grafo.
- Uma aresta é uma ponte caso sua remoção provoque a redução da conexividade do grafo.

***Grafo Fortemente Conexo (f-conexo):*** É um grafo direcionado com todos seus pares de vértices ligados por, pelo menos, um caminho de cada sentido. Cada vértice pode se alcançado se parte de qualquer outro vértice e pode alcançar qualquer outro.

Figura 3.18: Grafo Fortemente Conexo

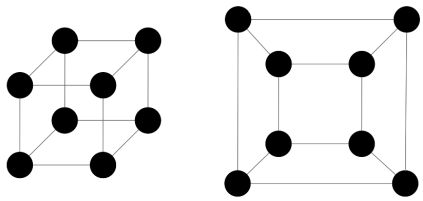


Fonte: Criação própria

- Cada par de vértices participa de um circuito.
- O grafo $G(V, E)$ que não é f-conexo é formado por, no mínimo, dois subgrafos fortemente conexos, disjutnos em relação aos vértices e maximais em relação à inclusão.

**Grafo Planar**: Um grafo é dito planar quando pode ser desenhado em um plano de forma que nenhuma de suas arestas se intercepte com outra.

Figura 3.19: Grafo Planar



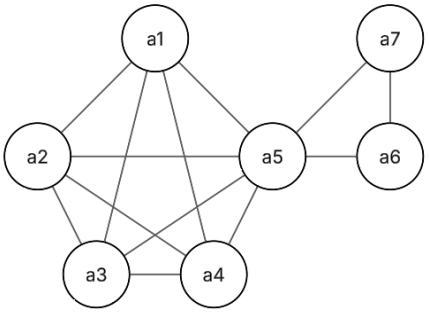
Fonte: Criação própria

*Observações:*

1. Para $n > 4$, o grafo completo $K_n$ não pode ser representado de forma planar.
2. As ligações de uma placa de circuito impresso devem obedecer a representações planas, evitando cruzamentos.
3. O grafo permanece o mesmo; o que varia é apenas sua representação gráfica.

**Grafo Euleriano**: Um grafo é euleriano quando admite um ciclo que percorre todas as arestas exatamente uma vez. Exemplo: $C_{11} = \{u_1, u_2, u_3, u_4, u_5, u_3, u_1, u_6, u_2, u_7, u_3, u_6, u_7, u_1\}$.

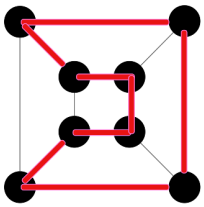
Figura 3.20: Grafo Euleriano




Fonte: Criação própria

**Grafo Hamiltoniano**: Um grafo é dito hamiltoniano quando possui um ciclo que visita todos os vértices exatamente uma vez.

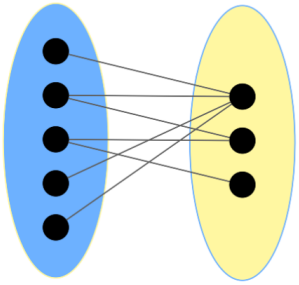
Figura 3.21: Grafo Hamiltoniano



Fonte: Criação própria

**Grafo Bipartido**: Um grafo bipartido é aquele em que o conjunto de vértices pode ser dividido em duas partes disjuntas, $V_A$ e $V_B$, de modo que não existam arestas conectando dois vértices do mesmo conjunto. Toda aresta liga um vértice de $V_A$ a outro de $V_B$.

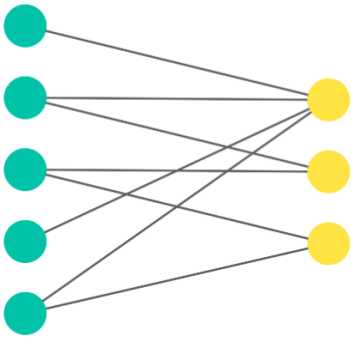
Figura 3.22: Grafo Bipartido



Fonte: Criação própria

**Grafo Bipartido Completo**: É um grafo bipartido no qual cada vértice de $V_A$ está conectado a todos os vértices de $V_B$.

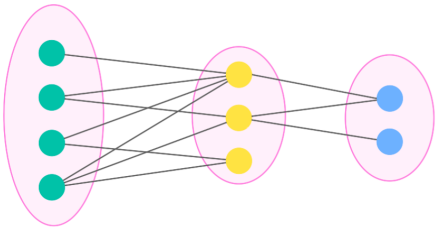
Figura 3.23: Grafo Bipartido Completo



Fonte: Criação própria

**Grafo $k$-partido (ou $k$-colorível)**: Um grafo é $k$-partido quando seus vértices podem ser separados em $k$ subconjuntos disjuntos, sem arestas internas a cada subconjunto. *Nota:* um grafo $2$-partido é equivalente a um grafo bipartido.

Figura 3.24: Grafo k-partido

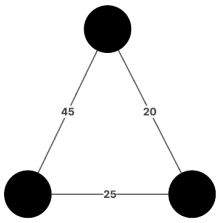


Fonte: Criação própria

**Grafo Rotulado**: Cada vértice (ou aresta) possui associado um identificador ou rótulo.

**Grafo Ponderado (ou Valorado)**: É um grafo no qual cada aresta possui um valor (peso) associado. Formalmente, seja $G = (V, X)$, em que $V$ é o conjunto de vértices e $X$ o conjunto de arestas; existe então uma função $P : X \to C_v$, onde $C_v$ representa o conjunto de pesos atribuídos às arestas.

Figura 3.25: Grafo Ponderado




Fonte: Criação própria

*Observações:*

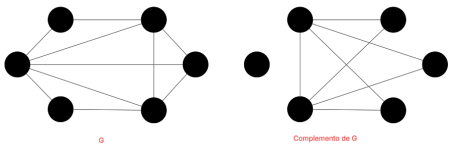
1. Esse tipo de grafo é amplamente utilizado em problemas que envolvem informações quantitativas.
2. Caso não haja pesos nas arestas, diz-se que o grafo é não ponderado, sendo de interesse apenas a estrutura das conexões.
3. Exemplo: em um grafo de rotas aéreas, a distância entre dois aeroportos pode ser usada como peso da aresta que os conecta.

**Grafos Complementares**: Dado um grafo não orientado $G(V, A)$, o grafo complementar $G^c(V, A')$ é definido pelo mesmo conjunto de vértices $V$, mas em $G^c$ existem

arestas justamente onde elas não estão em $G$. Assim:

$$(v_i, v_j) \in A \implies (v_i, v_j) \notin A' \quad \text{e} \quad (v_i, v_j) \notin A \implies (v_i, v_j) \in A'.$$

Figura 3.26: Grafo Complementar




Fonte: Criação própria

**Subgrafo gerador ou grafo parcial**: Grafo que contém o mesmo conjunto de vértices, porém, com pelo menos uma aresta removida.

Figura 3.27: Exemplo de grafo parcial

**Subgrafo induzido**: Subgrafo induzido é um subgrafo feito a partir de um conjunto de vértices do grafo original, não podendo ser descartada nenhuma aresta entre o conjunto.

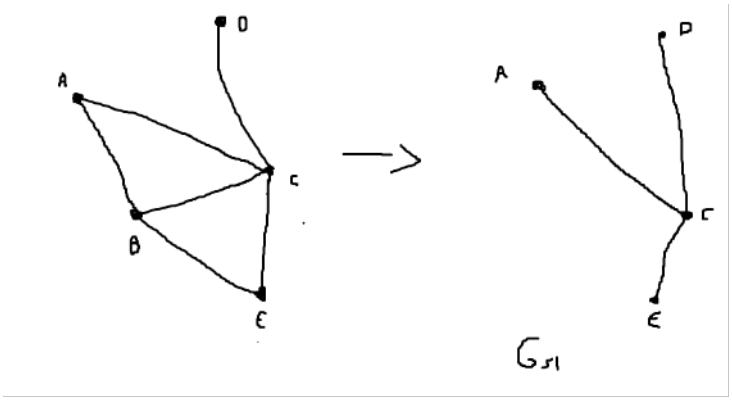



Figura 3.28: Exemplo de subgrafo induzido

**Clique**: É um subgrafo completo.

**Árvore**: É um grafo não direcionado que tem ordem (quantidade de vértices) maior ou igual a 2.

É importante ressaltar algumas propriedades para definir se um grafo é uma árvore:

1. $G$ é conexo e sem ciclos;
2. $G$ é sem ciclos e tem $|V| - 1$ arestas;
3. $G$ é conexo e tem $|V| - 1$ arestas;
4. $G$ é sem ciclos e a adição de uma aresta entre dois vértices não adjacentes cria um ciclo e somente um;
5. $G$ é conexo, mas deixa de sê-lo se uma aresta é suprimida (todas as arestas são pontes);
6. Todo par de vértices de $G$ é unido por uma e somente uma cadeia simples.

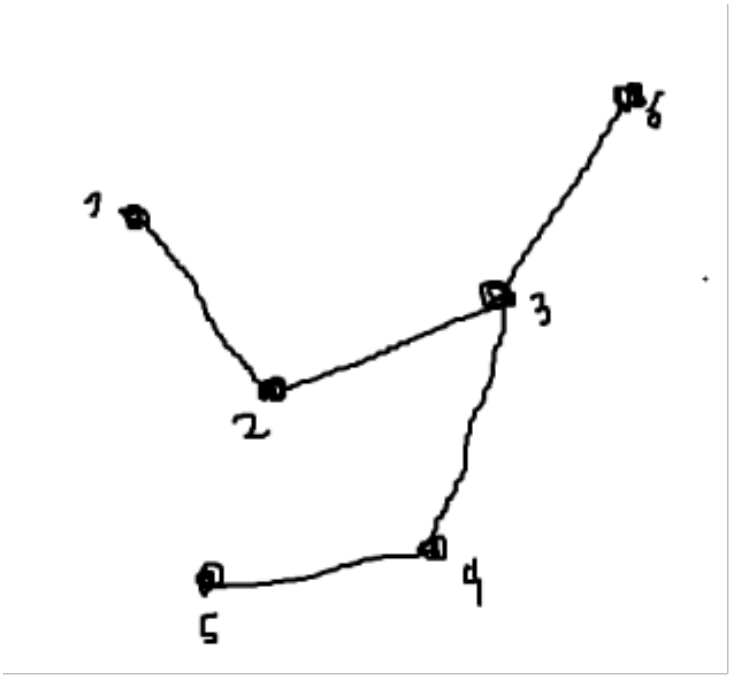



Figura 3.29: Exemplo de árvore

**Árvore geradora de um grafo conexo**: É um subgrafo conexo e acíclico que contém todo o conjunto de vértices e um subconjunto de arestas do grafo original.

Obs: Todo grafo conexo tem pelo menos uma árvore geradora.

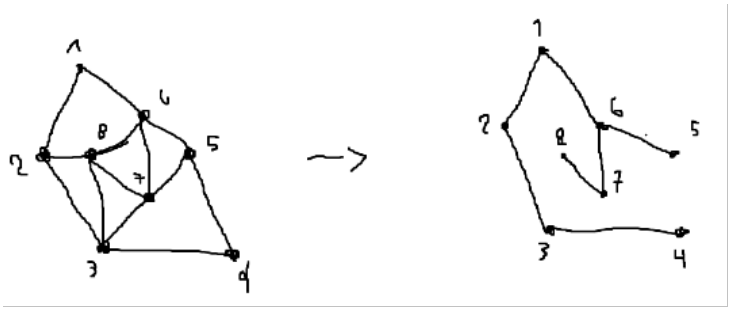


Figura 3.30: Exemplo de árvore geradora

**Floresta**: Um grafo acíclico que pode ser conectado ou não, cujas partes são árvores, sendo uma união de árvores.

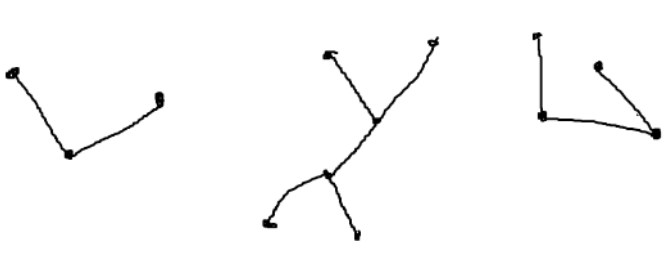


Figura 3.31: Exemplo de floresta

**Floresta geradora de um grafo**: Uma floresta que tem todos os vértices de um grafo.

**Grafos Isomórficos**: Um grafo é isomorfo a outro se ele mantém a mesma correspondência de incidências ao outro grafo.

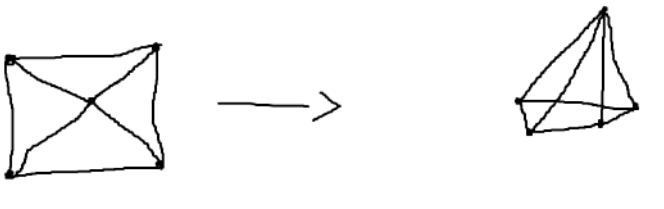


Figura 3.32: Exemplo de grafo isomorfo

Obs:

1. Se um grafo $G_2$ é isomorfo a $G_1$, então $G_1$ é isomorfo a $G_2$.
2. Se um grafo $G_2$ é isomorfo a $G_1$ e $G_3$, então $G_3$ é isomorfo a $G_2$ e vice-versa.
3. Um grafo é isomorfo a si próprio.
4. Um grafo $G_2$, se isomorfo a $G_1$, carrega as mesmas propriedades de $G_1$.

**Conjunto independente de vértices**: É um conjunto de pelo menos dois vértices de um grafo que não são adjacentes entre si.

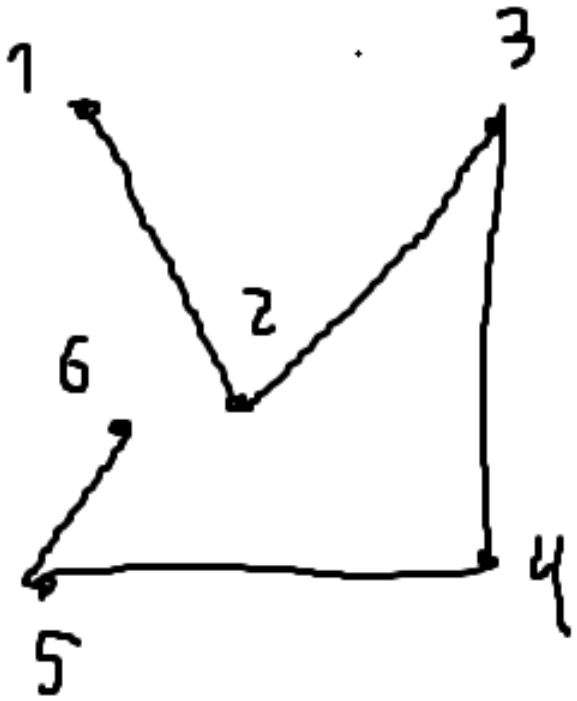



Figura 3.33: Exemplo de conjunto independente de vértices

Exemplos de conjuntos independentes de vértices do grafo da imagem:

$$S_1 = \{1, 3\}, \quad S_2 = \{2, 4, 6\}, \quad S_3 = \{5, 1, 3\}$$

**Grafo misto**: Quando um grafo pode possuir simultaneamente arestas e arcos.

**Hipergrafo**: Grafo não orientado onde cada aresta conecta um número arbitrário de vértices.

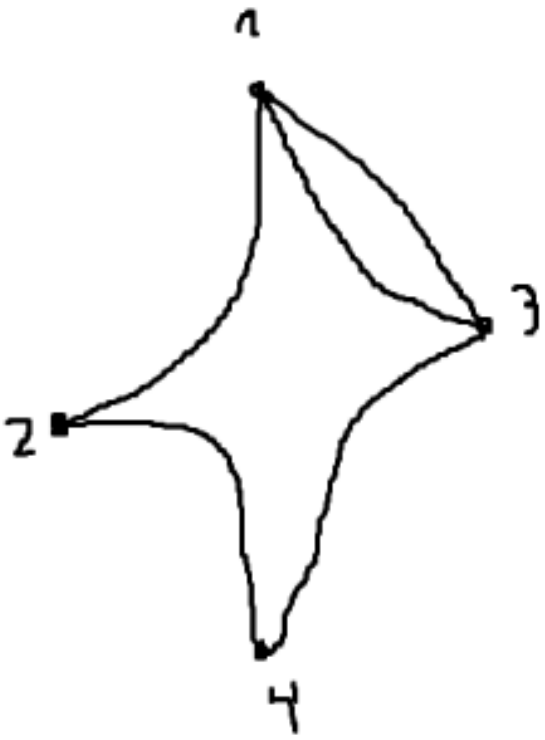



Figura 3.34: Exemplo de hipergrafo

Exemplo:

$$\text{Arestas} = \{1, 2, 4\}, \{1, 3, 4\}, \{1, 3\}$$

## 4 Operações

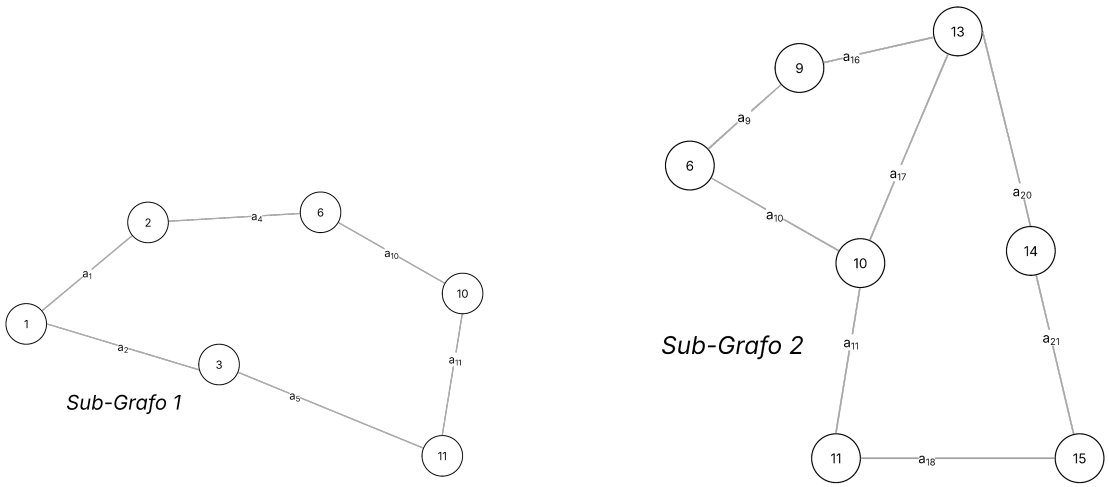
A estrutura de dados grafos permitem algumas operações matemáticas nos seus conjuntos de vértices e arestas de modo a facilitar a execução de alguns algoritmos. As principais operações são: união, intersecção, soma, decomposição, remoção, fusão e contração. Cada uma delas são exemplificadas, respectivamente, nas Seções 4.1, 4.2, 4.3, 4.4, 4.5, 4.6 e 4.7.

### 4.1 União

A união de dois grafos, sendo eles $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$, é dada por:

$$G_1 \cup G_2 = (V_1 \cup V_2, A_1 \cup A_2)$$

Esta é uma operação que apenas une os vértices e arestas dos grafos envolvidos, não adicionando nenhuma outra parte. Temos como exemplo a adição do $Sub\_Grafo1$ ao grafo $Sub\_Grafo2$. Onde $Sub\_Grafo1 = (V_1, V_2, V_3, V_6, V_{10}, V_{11}, a_1, a_2, a_4, a_5, a_{10}, a_{11})$ e $Sub\_Grafo2 = (V_6, V_9, V_{10}, V_{11}, V_{13}, V_{14}, V_{15}, A_9, A_{10}, A_{11}, A_{16}, A_{17}, A_{18}, A_{20}, A_{21})$.




(a) subgrafo 1 (b) subgrafo 2

Figura 4.1: subgrafos

A aplicação da operação de união resultaria em $Sub\_Grafo1 \cup Sub\_Grafo2$, representando graficamente se dá na figura seguinte:

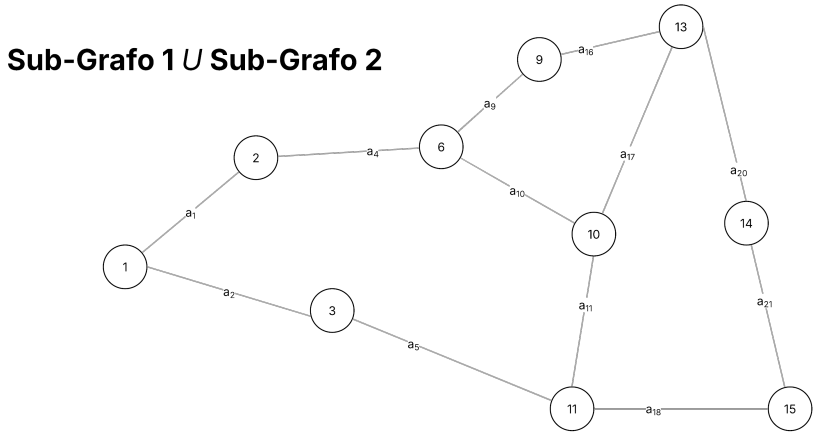



Figura 4.2: União dos subgrafos

### 4.2 Intersecção

A intersecção de dois grafos, sendo eles $G_1 = (V_1, A_1)$ e $G_2 = (V_2, A_2)$, gera um terceiro grafo, esse que é apenas o conjunto onde os dois grafos se encontram, ou seja, os vértices e arestas que são comuns aos dois grafos. Quando os grafos intersectam e resultam em nulos ou vazios $V_3 = 0$. A intersecção é dada por:

$$G_3 = G_1 \cap G_2 = (V_1 \cap V_2, A_1 \cap A_2)$$

Sendo $G_3$ a intersecção de $G_1$ e $G_2$, $G_3$ é um subgrafo de ambos os grafos originais. Aplicando a operação de intersecção nos subgrafos da seção 4.1, temos:

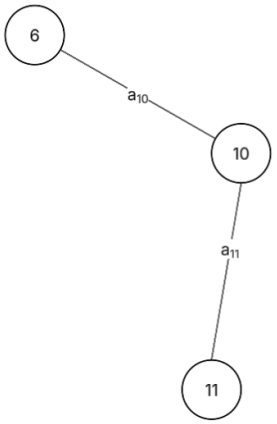



Figura 4.3: Intersecção dos subgrafos da figura 4.1

### 4.3 Soma

A soma de dois grafos é dividida entre dois tipos, a soma e soma direta. A soma junta os vértices e arestas dos dois grafos, criando uma conexão intercomplexa. A soma dos grafos resulta em um novo, como $G_3 = G_1 + G_2$ A fórmula é dada por:

$$G_3 = G_1 + G_2 = (V_1 \cup V_2, A_1 \cup A_2 \cup \{\forall vi \in V_1, \forall vj \in V_2, \exists (vi, vj)\})$$

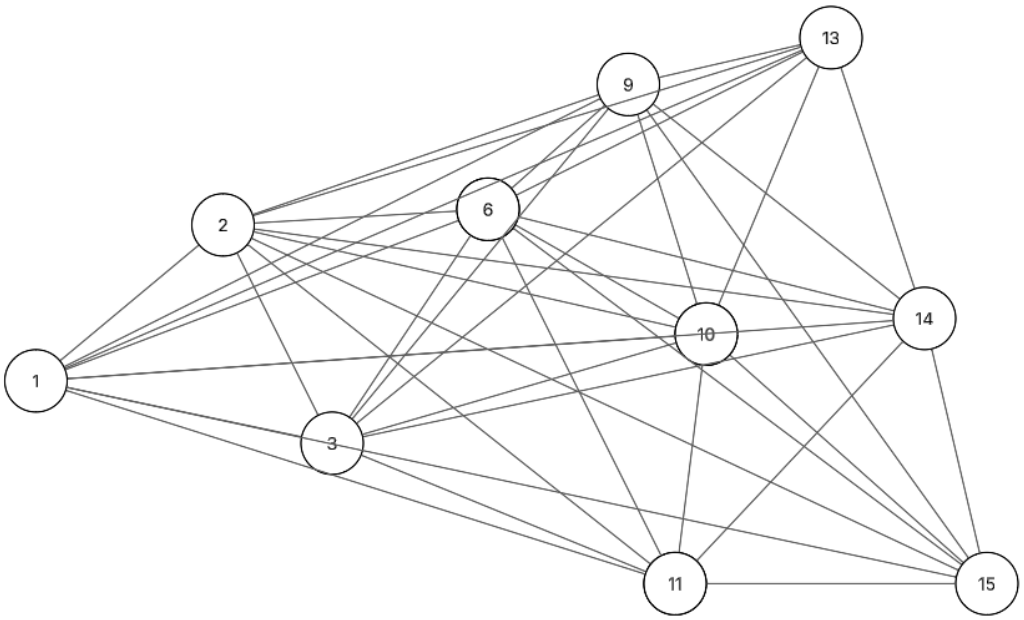



Figura 4.4: Soma dos subgrafos da figura 4.1

Seguindo o exemplo, utilizando as fig 4.1 novamente, podemos observar as novas arestas serem criadas, se diferindo da soma direta, a qual é uma forma de combinar dois grafos baseados em multiplicação estrutural. O conjunto de vértices resultante do produto cartesiano dos conjuntos de vértices dos grafos originais, sendo todos pares ordenados, onde o primeiro elemento pertence a $V_1$ e o segundo a $V_2$.

$$V_3 = V_1 \times V_2 = \{(u, v) : u \in V_1,\ v \in V_2\}$$

O conjunto de arestas se baseia na regra da adjacência, uma aresta entre dois vértices $(u, v)$ e $(u_1, v_1)$ em $V_3$ se e somente se eles são adjacentes em $G_1$ e $G_2$, tendo assim um grafo $G_1 \times G_2$. Traduzindo para a fórmula:

$$G_3 = G_1 + G_2 = \big(V_1 \cup V_2,\ E_1 \cup E_2 \cup \{\{u, v\} : u \in V_1,\ v \in V_2\}\big).$$

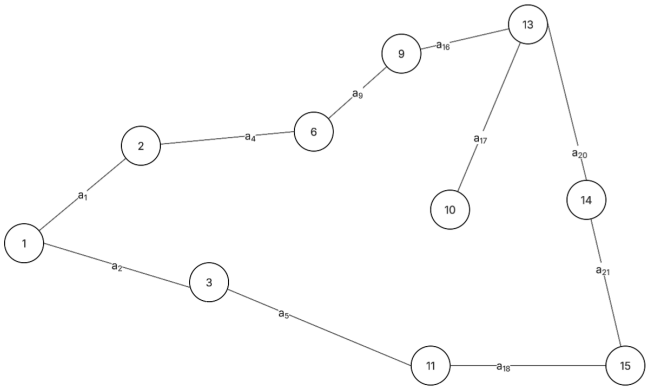



Figura 4.5: Soma Direta dos subgrafos da figura 4.1

### 4.4 Decomposição

A decomposição de um grafo $G$ é a partição de seu conjunto de arestas em subgrafos distintos. Sendo assim, um grafo é decomposto em dois subgrafos $G_1$ e $G_2$ se sua união resulta no grafo original ($G = G_1 \cup G_2$) e sua intersecção é um grafo nulo ($G_1 \cap G_2 = \emptyset$). A Figura 4.6 demonstra como o subgrafo gerado na operação de união (*Figura 4.2 do PDF*) é decomposto de volta nos seus subgrafos originais.

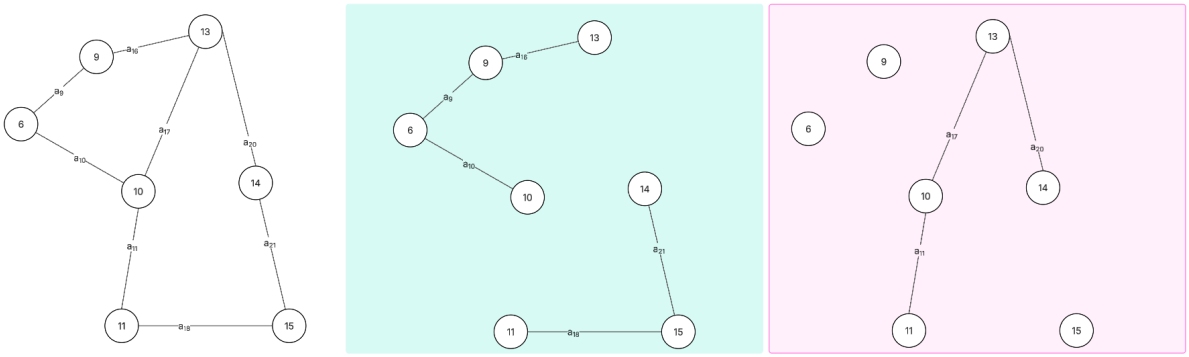


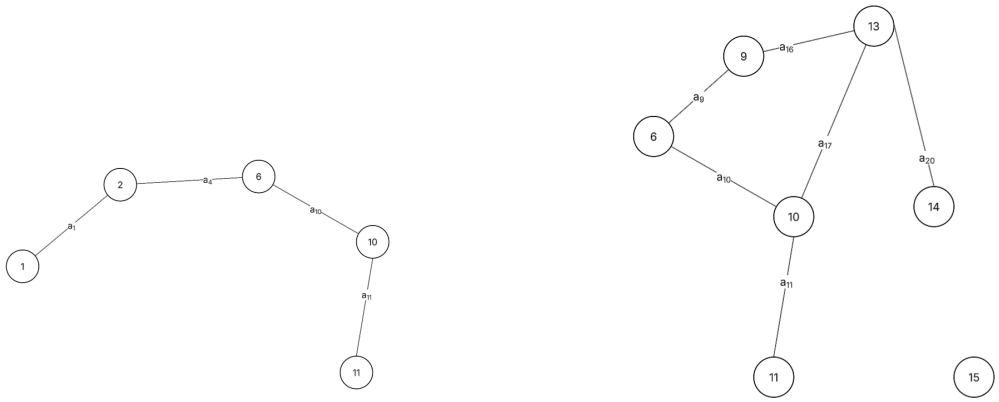

Figura 4.6: Decomposição do subgrafo resultante da operação de união.

### 4.5 Remoção

A remoção de um vértice ou aresta de um grafo $G$ resulta em um subgrafo $G'$. Na remoção de um vértice, todas as arestas incidentes a ele também são removidas. A remoção de uma aresta/arco é representada por $G - a$ e a de um vértice por $G - v$. O impacto da remoção na conectividade é uma métrica crucial para a avaliação de robustez do grafo. A remoção de uma aresta-ponte ou de um vértice de articulação pode aumentar o número de componentes conexos do grafo.

A Figura 4.7 ilustra essas operações. Em (a), é demonstrada a remoção do vértice $V_3$ do subgrafo (a) da Figura 4.1, gerando o resultado em Figura 4.7a. Em (b), as arestas $A_{18}$ e $A_{21}$ são removidas do subgrafo (b) da Figura 4.1, resultando no grafo da

Figura 4.7b.




(a) Remoção do vértice $V_3$ do subgrafo (a) da Fig. 4.1.

(b) Remoção das arestas $A_{18}$ e $A_{21}$ do subgrafo (b) da Fig. 4.1.

Figura 4.7: Exemplos de remoção de vértice e de arestas.

### 4.6 Fusão de vértices

A fusão de vértices é uma operação que transforma um grafo ao substituir dois ou mais vértices por um único, mantendo as arestas que conectavam os vértices originais ao restante do grafo. Essa operação reduz o número de vértices em uma unidade, mas o número de arestas permanece inalterado. A Figura 4.8 exemplifica essa operação, onde os vértices $V_{11}$ e $V_{15}$ são fundidos em um novo vértice.

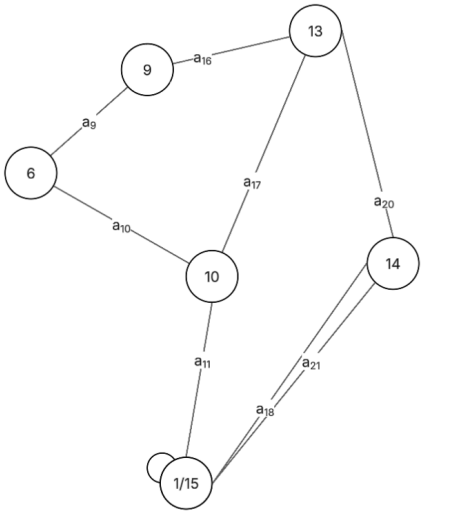



Figura 4.8: Fusão dos vértices 11 e 15.

### 4.7 Contração

A contração de aresta é um caso especial da fusão de vértices. Nesta operação, uma aresta é removida e, simultaneamente, os dois vértices que ela conectava são

fundidos. Se existiam arestas paralelas, a contração pode resultar em um laço no novo vértice. O número de arestas do grafo diminui em 1 após a contração.

O grau do novo vértice $w$ resultante da contração de uma aresta $uv$ é a soma dos graus dos vértices originais, subtraída de duas unidades, como demonstrado na Figura 4.9. A fórmula é:

$$deg(w) = deg(u) + deg(v) - 2$$

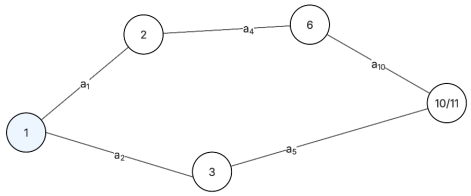



Figura 4.9: Contração da aresta entre os vértices $V_{10}$ e $V_{11}$.

# 5 Buscas

A busca é umas das técnicas mais aplicadas na solução de problemas algorítmicos em grafos considerados eficientes. As duas técnicas de busca em grafos, a dfs e a bfs, são apresentadas, respectivamente, nas Seções 5.1 e 5.2.

A Seção 5.3 apresenta uma aplicação das duas buscas para obter os componentes fortemente conexos de um grafo. As considerações acerca da eficiência computacional, tanto em relação ao tempo de processamento quanto ao espaço usado de memória pelas buscas, são dadas na Seção 2.6.

## 5.1 Busca em Largura

A Busca em Largura (Breadth-First Search - BFS) é um algoritmo de busca que explora os vértices de um grafo em camadas, ou níveis de distância, a partir de um vértice inicial. Diferente da busca em profundidade, a BFS utiliza uma estrutura de dados de fila (FIFO - First-In, First-Out) para garantir que todos os vértices a uma distância $k$ sejam visitados antes de qualquer vértice a uma distância $k + 1$. Essa característica torna o algoritmo ideal para encontrar o caminho mais curto (em número de arestas) entre dois vértices em um grafo não valorado.

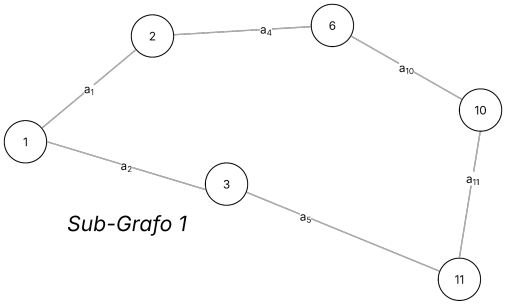



Figura 5.1: Subgrafo 1 da seção 4.1

Aplicando o algoritmo de Busca em Largura (BFS) no subgrafo acima, partindo do vértice $V_1$, o processo de descoberta dos vértices ocorre na seguinte ordem de níveis:

- **Nível 0:** $V_1$ (vértice inicial)
- **Nível 1:** Vértices adjacentes a $V_1 \rightarrow \{V_2, V_3\}$
- **Nível 2:** Vértices adjacentes ao Nível 1 (e ainda não visitados) $\rightarrow \{V_6, V_{11}\}$

- **Nível 3:** Vértices adjacentes ao Nível 2 (e ainda não visitados) $\rightarrow \{V_{10}\}$

Nesta busca, a aresta $(V_{10}, V_{11})$ é uma aresta de cruzamento, pois conecta um vértice do Nível 3 a um vértice do Nível 2 que não é seu pai na árvore de busca gerada.

### 5.2 Busca em Profundidade

A busca em profundidade serve para auxiliar algoritmos de verificação de grafos acíclicos, ordenação topológica e componentes fortemente conexos. Como exemplo, foi feita uma busca em profundidade do grafo da Figura 5.2, iniciando do vértice $V_1$ até o $V_2$.

Para realizar a busca, foi escolhida uma raiz ($V_1$) e, a cada vértice, um vértice filho para se seguir. O caminho utilizado foi:

$$V_1 \rightarrow V_4 \rightarrow V_8 \rightarrow V_{12} \rightarrow V_{11} \rightarrow V_{15} \rightarrow V_{14} \rightarrow V_{13} \rightarrow V_9 \rightarrow V_6 \rightarrow V_2$$

Após seguir para o $V_2$, é feito o retorno e verificado se todos os filhos estão na busca. Assim, surgem os vértices $V_{10}$, filho de $V_6$; $V_{16}$, filho de $V_{15}$; $V_3$ e $V_7$, filhos de $V_{11}$; e $V_5$, filho de $V_8$.

A destacar também devemos falar sobre os tempos de abertura e fechamento, na figura, o tempo de abertura está em vermelho e o de fechamento em azul. Para cada vértice novo, é colocado um tempo de abertura, que se segue linear até o último vértice da coluna, e no momento da volta, é feito o tempo de fechamento, que é definido quando um vértice não tem mais nenhum filho a ser referenciado, como pode ser visto no $V_6$, onde ele não fecha antes do $V_{10}$.

Também são analisadas as arestas de retorno (simbolizadas pelas linhas tracejadas na figura), que são arestas que ligam um vértice de tempo menor a um de tempo maior, sendo identificadas quatro:

$$(V_2 \rightarrow V_1),\ (V_{10} \rightarrow V_{13}),\ (V_3 \rightarrow V_1),\ (V_7 \rightarrow V_{12})$$

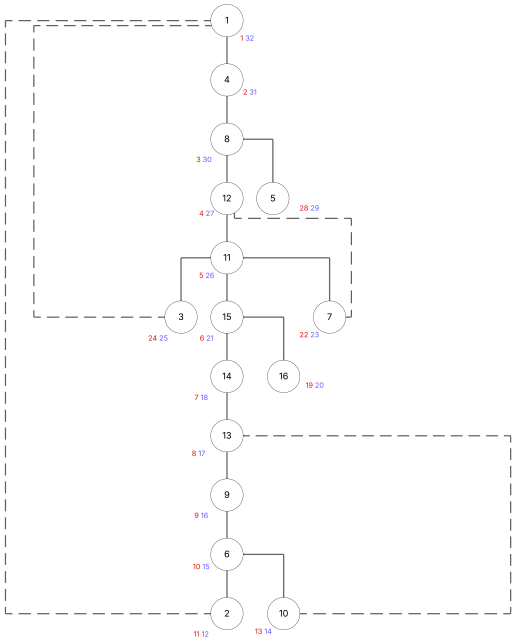



Figura 5.2: Busca em profundidade

## 5.3 Componentes Fortemente Conexos

Um componente fortemente conexo (CFC) é um subgrafo máximo de um grafo orientado, onde cada vértice alcança todos os outros. Para encontrar um CFC, considerando um grafo G = ()V, X) deve-se seguir o seguinte passo a passo:

Etapa 1: Realizar busca em largura considerando a lista de sucessores

Etapa 2: Realizar busca em largura considerando a lista de predecessores

Etapa 3: O componente fortemente conexo, um subgrafo do grafo de G, é encontrado:

$$V_r = \{ V^+ \cap V^- \} \cup \{ v_r \} \text{ e } X_r = \{ (v_i, v_j) \in X[G] \mid \{v_i, v_j\} \in V_r \}. \quad (5.1)$$

No contexto desse trabalho, não há necessidade de encontrar um CFC, já que, em um grafo não orientado, todo componente já é fortemente conexo.

## 5.4 Considerações de Eficiência

O tempo de execução gasto do algoritmo de busca em profundidade é $O(|V| + |X|)$, onde $O(|V|)$ é obtido a partir da inicialização do vetor responsável por definir o estado do vértice durante a busca e $O(|X|)$ corresponde a varredura total das listas de adjacências. Quando a representação utilizada for em matriz, sua eficiência será $O(|V|)$ Para o nosso grafo, é $O(38)$.

O custo do algoritmo de busca em largura é o mesmo, ou seja, $O(|V| + |X|) = O(38)$.

# 6 CAMINHO MÍNIMO

Quando a definição de distância é associada aos caminhos, surgem os problemas de caminho mínimo amplamente usados na solução de problemas aplicados. O caminho mínimo mais básico ocorre em grafos não valorados, onde o caminho mínimo é dado pelo número de arestas, não pelo somatório do valor associado às arestas. A Seção 6.1 apresenta a definição e o cálculo do caminho mínimo em grafos não valorados. No momento em que se considera grafos valorados, o caminho mínimo pode ser definido por diferentes algoritmos dependendo das características particulares do grafo a ser analisado. A seção 6.2 apresenta os algoritmos de caminho mínimo de única origem e a seção 6.3 os algoritmos de caminho mínimo entre todos os pares de vértices. As considerações acerca da eficiência computacional, tanto em relação ao tempo de processamento quanto ao espaço usado de memória pela definição do caminho mínimo, são dadas na Seção 6.4.

## 6.1 Distâncias

Quando um grafo é não valorado, o cálculo de caminho mínimo se dá na busca em largura (BFS), essa busca garante que o primeiro caminho encontrado entre o ponto de partida e o destino será o mais curto, em termos do menor número de arestas percorridas.

O cálculo a seguir, será feito a partir da imagem 6.1, que é um trecho retirado do grafo utilizado no restante do documento.

E para fazer algum sentido, será desconsiderado os pesos apresentados nas arestas.

Como já explicado no tópico 5.1, a busca em largura é realizado utilizando a ordem de fila (FIFO), no seguinte processo:

$1 \rightarrow$ Pegamos o 1 como raiz e o enfileira [1].

$2 \rightarrow$ Desenfileira e visitamos os vértices adjacentes não visitados, no caso, o $V_4$, e enfileira [4].

$3 \rightarrow$ Mesmo processo do passo anterior, mas dessa vez com o $V_4$, no caso, temos $V_5$, $V_8$ e $V_1$, porém como $V_1$ já foi visitado, usaremos apenas os dois primeiros, os enfileirando, [5,8].

$4 \rightarrow$ Desenfileira o 5 e verifica os vértices adjacentes, no caso temos o $V_8$, mas como ele já foi visitado, então, não fazemos nada, seguimos então com o desenfileiramento do $V_8$, e ao verificar os vértices adjacentes, é percebido que não tem mais nenhum vértice sem marcação, nesse caso, a fila está vazia, e então, é finalizado a busca.

Nesse caso, verificamos a ordem recebida:

$V_1 \rightarrow V_4$
$V_4 \rightarrow V_5$
$V_4 \rightarrow V_8$
Então nesse caso, temos o caminho mínimo nos seguintes casos:
Raiz = 1, Destino = 4, caminho mínimo = 1, (1,4).
Raiz = 1, Destino = 5/8, caminho mínimo = 2 (1,4),(4,5/8).

## 6.2 Origem Única

O problema de caminhos mínimos de única origem visa encontrar um caminho mínimo de um determinado vértice de origem até todos os outros vértices do grafo. Diversos são os algoritmos responsáveis por determinar os caminhos mínimos de única origem em um grafo valorado, seja ele orientado ou não. Entretanto, cada um tem algumas limitações. As Seções 6.2.1, 6.2.2 e 6.2.3 apresentam, respectivamente, os algoritmos de Dijkstra, de Ordenação topológica e de Bellman–Ford.

### 6.2.1 Algoritmo de Dijkstra

O Algoritmo de Dijkstra é o mais utilizado no mundo, porém, ele não pode ser utilizado, caso exista, arestas com pesos negativos.

O processo do algoritmo é definir um ponto de origem e a cada passo seguinte, selecionar um vértice com o menor peso em relação ao caminho mínimo de $V_0$ até $V_N$.

O Processo é descrito a seguir:

Passo 1 $\rightarrow$ É definido um V de origem e inicializado duas filas, uma de $VPai$ [NULL,...,NULL] e uma de pesos [$\infty$,... $\infty$].

Passo 2 $\rightarrow$ É selecionado o primeiro vértice mais próximo de $V_0$ e é atualizado a fila de pesos e a fila $VPai$.

Passo 3 $\rightarrow$ É selecionado o próximo vértice de menor peso e a partir dele, é analisado, e caso algum vértice tenha um peso menor com relação ao $VAtual$, é atualizado tanto na fila de pesos quanto na fila $VPai$, e assim se segue até que todos os vértices sejam analisados, ao terminar, as filas recebidas serão utilizadas para definir o caminho mínimo.

Usando o mesmo subgrafo utilizado no tópico 6.1, dessa vez, considerando os pesos na imagem.

1 $\rightarrow$ definimos o $V_1$ como vértice origem, e a partir disso, inicializamos a fila de pesos dist[$\infty, \infty, \infty, \infty$], e a fila $VPai$ [NULL,NULL,NULL,NULL].

2 $\rightarrow$ como o $V_1$ tem apenas 1 $V$ adjacente, seguimos para ele e atualizamos as filas, dist[0 ; 92,82 ; $\infty; \infty$] e $VPai$ [NULL,1,NULL,NULL].

3 $\rightarrow$ dessa vez, o $V_4$ tem 2 vértices adjacentes a ele, então é verificado qual dos dois tem o menor caminho, e nesse caso, é selecionado o $V_5$, tendo em vista que 24,43 < 74,93, e é atualizado as filas dist[0 ; 92,82 ; 117,25 ; 167,75] e $VPai$ [NULL,1,4,4].

4 $\rightarrow$ Como o grafo é não orientado, podemos apenas ignorar a adjacência com de $V_5$ com $V_4$ pois o caminho tem o mesmo peso na ida e na volta, caso fosse orientado, teria que se analisar todos os caminhos para atualizar as filas corretamente, e analisando o caminho de $V_5 \rightarrow V_8$ nós temos um peso de 117,25 + 85,49 = 202,74 que é maior que o valor já inserido no $V_8$, que é 167,75, que faz com que as filas permaneçam do mesmo jeito que o passo anterior.

5 $\rightarrow$ E por fim, indo visitar o $V_8$, pelo $V_4$, temos o valor de 167,75 + 85,49 = 253,24 para o $V_5$, que é maior que o valor já inserido de 117,25, então, as filas permanecem do mesmo jeito.

Com isso, todos os vértices foram visitados e analisados, a partir disso temos as filas de peso e de $VPai$, que ficaram respectivamente:

[0 ; 92,82 ; 117,25 ; 167,75].

[NULL,1,4,4].

A partir disso, é definido os mínimos caminhos para cada $V_0$ e $V_Destino$.

### 6.2.2 Ordenação topológica

O algoritmo de ordenação topológica é aplicado exclusivamente em grafos direcionados e acíclicos (DAGs — *Directed Acyclic Graphs*). Sua finalidade é estabelecer uma sequência linear dos vértices, de modo que, para toda aresta $(u, v)$, o vértice $u$ preceda $v$ na ordenação. Essa propriedade é essencial em problemas que envolvem precedência, como planejamento de tarefas, compilação de dependências e execução de processos com restrições de ordem.

A ideia central consiste em iterativamente selecionar vértices sem predecessores (grau de entrada igual a zero), inserindo-os na sequência ordenada e removendo suas arestas do grafo. O processo se repete até que todos os vértices tenham sido processados ou até que não restem vértices com grau de entrada zero, o que indicaria a presença de ciclos.

Considerando o grafo da Figura 6.1, representando dependências entre tarefas, cada aresta $(u, v)$ indica que a tarefa $u$ deve ser concluída antes de $v$. Aplicando o algoritmo, obtém-se uma possível sequência válida:

$$V_1, V_4, V_5, V_8$$

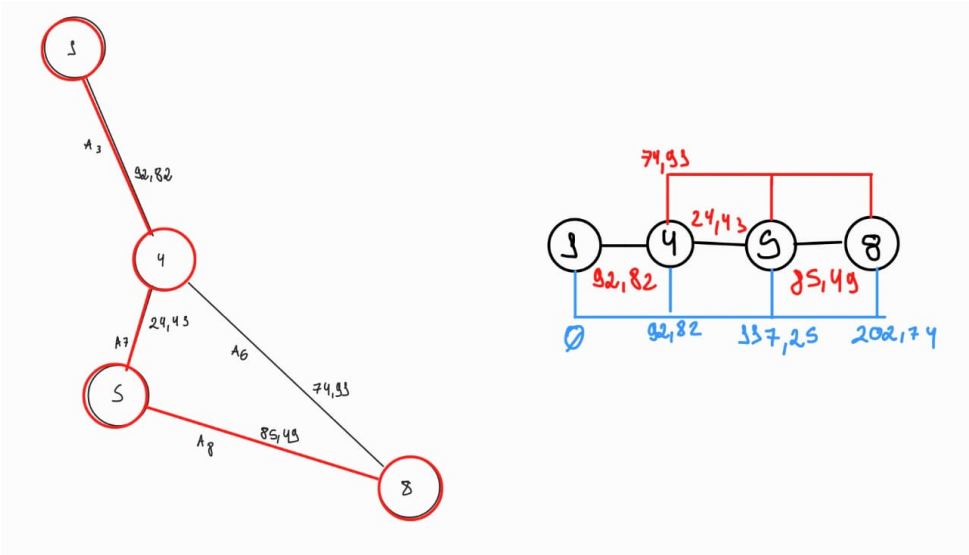



Figura 6.1: Exemplo de grafo direcionado acíclico (DAG) para aplicação do algoritmo de ordenação topológica.

O algoritmo percorre todos os vértices e arestas uma única vez, resultando em complexidade de tempo $O(|V| + |E|)$. O custo de espaço também é $O(|V|)$ devido às estruturas auxiliares utilizadas (fila e vetor de graus de entrada).

### 6.2.3 Algoritmo de Bellman–Ford

O algoritmo de Bellman–Ford é utilizado para determinar o caminho mínimo a partir de uma única origem em grafos ponderados, podendo conter arestas com pesos negativos. Diferentemente de Dijkstra, ele é capaz de identificar a presença de ciclos de peso negativo, tornando-se mais versátil em situações em que tais ciclos podem ocorrer.

A abordagem baseia-se no *relaxamento* iterativo das arestas. Para cada aresta $(u, v)$, verifica-se se o caminho até $v$ pode ser melhorado passando por $u$. Esse processo é repetido $(|V| - 1)$ vezes, pois o caminho mais curto entre dois vértices não pode conter mais do que $(|V| - 1)$ arestas. Ao final, uma iteração adicional permite verificar a existência de ciclos de peso negativo.

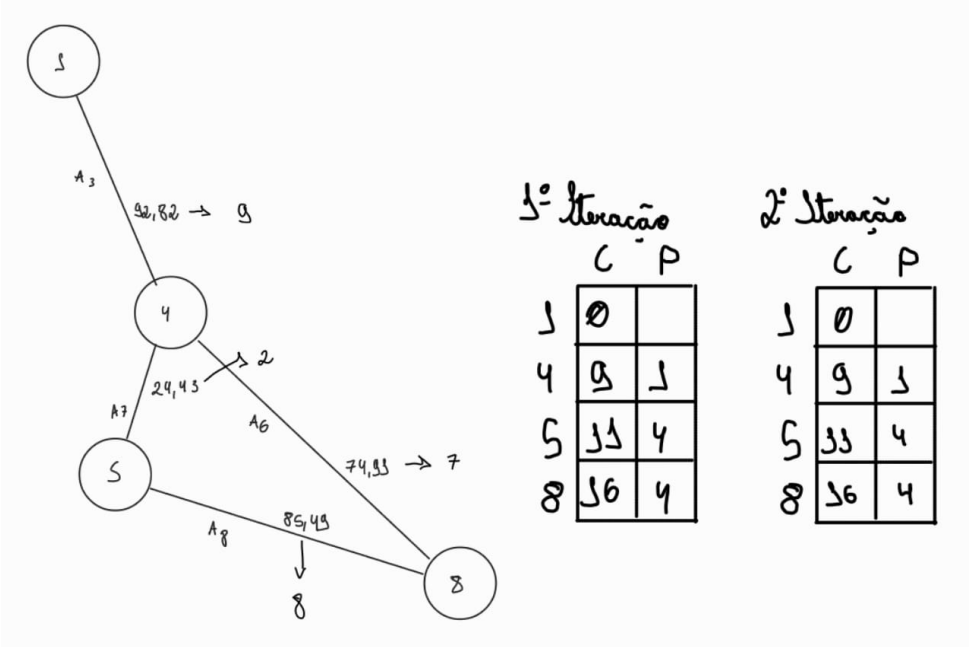



Figura 6.2: Exemplo de grafo ponderado aplicado ao algoritmo de Bellman–Ford.

Como não houve otimização na segunda iteração, a existência das outras $(|V| - 1)$ é desnecessária.

O custo de tempo é $O(|V| \times |E|)$, pois todas as arestas são verificadas em cada uma das $(|V| - 1)$ iterações. O custo de espaço é $O(|V|)$, utilizado para armazenar as distâncias e predecessores.

## 6.3 Entre todos os Pares

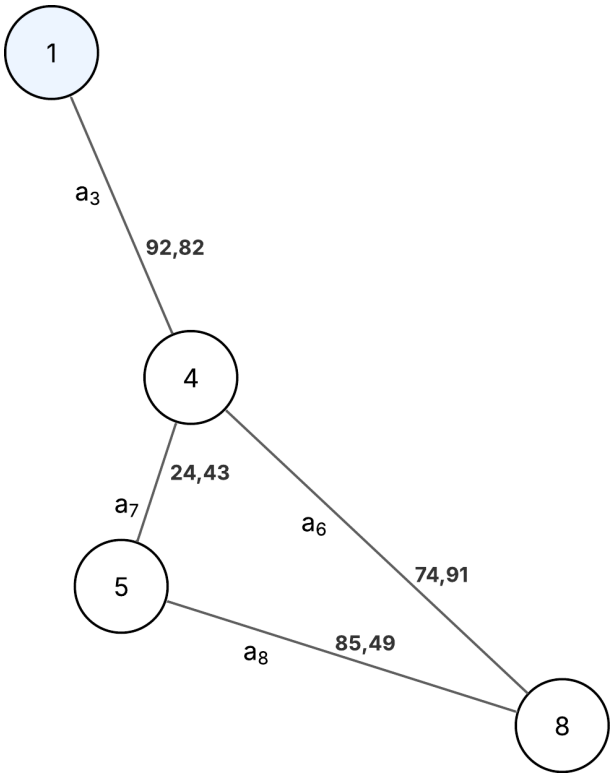
Os algoritmos de caminho mínimo de origem única, ao executar uma vez para cada vértice pertencente ao grafo, pode solucionar o caminho mínimo entre todos os pares de vértices. Entretanto, o custo computacional desta configuração não é interessante dado que existem algoritmos mais eficientes que entregam a mesma solução em um

tempo computacional inferior. A Seção 6.3.1 apresenta o algoritmo de Floyd-Warshall que calcula os caminhos mínimos entre todos os pares de vértices se o grafo for f–conexo, para grafos orientados, ou conexo para grafos não orientados.

### 6.3.1 Algoritmo de Floyd Warshall

O algoritmo de Floyd Warshall foi concebido com o objetivo de reduzir o custo computacional, a partir de uma programação dinâmica: Se o caminho mínimo de um vértice $v_0 \in V[G]$ a um $v_d \in V[G]$ passa por $v_k \in V[G]$, então $v_0$à$v_k$ é caminho parcial de $v_0$à$v_d$, que devem ser mínimos. Apenas existirá caminhos mínimos entre todos os pares de vértices se o grafo for F-conexo, para orientados, ou conexo, para não orientados. Esse algoritmo permite o uso de pesos negativos, desde que, o grafo de entrada não possua ciclos negativos. É aplicado o relaxamento de peso, ou seja, ele aprimora a estimação de peso a cada iteração até que atinja o valor ótimo. Para utilizar o algoritmo, deve-se utilizar a matriz de adjacência como representação, numerando os vértices de 1,2,3,4... . Para identificar pares de vértices, o algoritmo utiliza do conceito de vértices intermediários, isto é, qualquer vértice que está no caminho entre um vértice a outro. Esses vértices são percebidos de forma recursiva, selecionando sempre o menor peso para obter o melhor caminho. O passo a passo do algoritmo consiste em selecionar um índice correspondente a um vértice intermediário, de forma incremental. Dessa forma, é feita a varredura de todos os possíveis caminhos, o menor é selecionado para comparar com o que se já está mapeado (caso já esteja). Esse processo é feito até que todos os vértices intermediários sejam percorridos. Introduzido o algoritmo, pode-se aplicá-lo a um trecho do grafo trabalhado nesse memorial a fim de compreendê-lo na prática:

Figura 6.3: Trecho Selecionado para Aplicar Floyd Warshall




Fonte: Criação própria

Primeiro, devemos inicializar nossas matrizes, evidenciar como os caminhos são percebidos à principio:

$$p_c = \begin{bmatrix} 0 & 92,82 & \infty & \infty \\ 92,82 & 0 & 24,43 & 74,91 \\ \infty & 24,43 & 0 & 85,49 \\ \infty & 74,91 & 85,49 & 0 \end{bmatrix} \tag{6.1}$$

$$pai = \begin{bmatrix} \text{NULL} & 1 & \text{NULL} & \text{NULL} \\ 4 & \text{NULL} & 4 & 4 \\ \text{NULL} & 5 & \text{NULL} & 5 \\ \text{NULL} & 8 & 8 & \text{NULL} \end{bmatrix} \tag{6.2}$$

Para $k = 1$ temos:

$$p_c = \begin{bmatrix} 0 & 92,82 & \infty & \infty \\ 92,82 & 0 & 24,43 & 74,91 \\ \infty & 24,43 & 0 & 85,49 \\ \infty & 74,91 & 85,49 & 0 \end{bmatrix} \tag{6.3}$$

$$pai = \begin{bmatrix} \text{NULL} & 1 & \text{NULL} & \text{NULL} \\ 4 & \text{NULL} & 4 & 4 \\ \text{NULL} & 5 & \text{NULL} & 5 \\ \text{NULL} & 8 & 8 & \text{NULL} \end{bmatrix} \tag{6.4}$$

Note que, nada muda: o $v_1$ não é intermediário em nenhum dos caminhos mínimos possíveis.

Para $k = 2$ temos:

$$p_c = \begin{bmatrix} 0 & 92,82 & 117,25 & 167,73 \\ 92,82 & 0 & 24,43 & 74,91 \\ 117,25 & 24,43 & 0 & 85,49 \\ 167,73 & 74,91 & 85,49 & 0 \end{bmatrix} \tag{6.5}$$

$$pai = \begin{bmatrix} \text{NULL} & 1 & 4 & 4 \\ 4 & \text{NULL} & 4 & 4 \\ 4 & 5 & \text{NULL} & 5 \\ 4 & 8 & 8 & \text{NULL} \end{bmatrix} \tag{6.6}$$

Nessa iteração, Pode-se afirmar que todos os caminhos mínimos foram descobertos, e é possível concluir que o principal vértice intermediário é o $v_4$, já que ele conecta todos os outros vértices.

## 6.4 Considerações de Eficiência

### 6.4.1 Algoritmo de Dijkstra

O algoritmo de Dijkstra é utilizado para que não possuem pesos negativos e possui melhor custo computacional quando é aplicado o heap de fibonacci, resultando em $O(Vlog(V) + X)$

### 6.4.2 Ordenação topológica

A ordenação topológica é utilizado para grafos orientados acíclicos (DAGs) e possui melhor custo computacional quando é representado com a lista de adjacência, resultando em $O(V + E)$, e, no pior caso, $O(v^2)$

### 6.4.3 Algoritmo de Bellman–Ford

O algoritmo de Bellman-Ford é utilizado e possui melhor custo computacional quando é quando se utiliza lista de adjacências, resultando em $O(V + E)$, e no pior caso $O(V \times X)$. Para grafos densos, o melhor caso é $O(V^2)$ e, no pior caso, $O(V^3)$.

### 6.4.4 Algoritmo de Floyd Warshall

O algoritmo de Floyd Warshall, pensado para possuir o melhor custo computacional, é utilizado para todos os pares em grafos densos e possui custo de $O(V^3)$

### 6.4.5 Aplicações dos algoritmos

Quando possuir um grafo esparso e sem arestas negativas, deve-se optar pelo algoritmo Dijkstra. Quando possui um grafo denso e sem arestas negativas, deve-se optar pelo Floyd Warshall. Quando possuir um grafo com arestas negativas, deve-se optar por Bellman-Ford ou Floyd Warshall. Quando seu grafo for aciclico, com arestas negativas e direcionado, deve-se optar por ordenação topológica.

# 7 ARVORE DE COBERTURA MÍNIMA

### 7.0.1 Algoritmo de Kruskal

Para obter a árvore de cobertura mínima pelo algoritmo de Kruskal, primeiramente, temos que pegar o grafo e ordenar todas as arestas por peso, da mais leve para a mais pesada, o grafo usado será o grafo inteiro, que está na imagem 2.1.

A partir do grafo escolhido e com as arestas ordenadas de forma crescente, separamos duas filas, uma de conjunto e outra de quantidade de cada conjunto, inicializando-as de 1 até 16 e 1 em todas as 16 posições, respectivamente.

Com as filas preparadas, começamos a pegar cada aresta e comparamos o conjunto de cada um, pela ordenação, a aresta (4,5) é a primeira, e a partir dela, temos que o conjunto 4 é diferente do de 5, então a partir disso, um dos dois conjuntos se tornará parte do outro, para definir qual será, temos que olhar para a outra fila de quantidade, no qual todos estão como 1, então podemos escolher qualquer um para tal, no caso, vamos colocar 4 no conjunto 5, então nas próximas comparações, o V4 será 5 e agora mudamos na fila de quantidades, que no conjunto 5, se tem 2 membros, e no conjunto 4, se tem 0.

Para simplificar a explicação, vou apenas destacar alguns detalhes do algoritmo com exemplos reais do processo a ser realizado.

No momento em que vamos usar a aresta (7,11), temos que o conjunto 7 faz parte do 12, então a comparação a ser feita com 11 é de 12 com 11, e não 7 com 11, nesse caso 12 != 11, e ao analisar, vemos que o 12 tem maior quantidade de membros em seu conjunto, nesse caso, obrigatoriamente, o 11 começa a fazer parte do conjunto 12, então a quantidade de membros em 11 será reduzida pra 0 e iremos aumentar o de 12 com a diferença restante do 11.

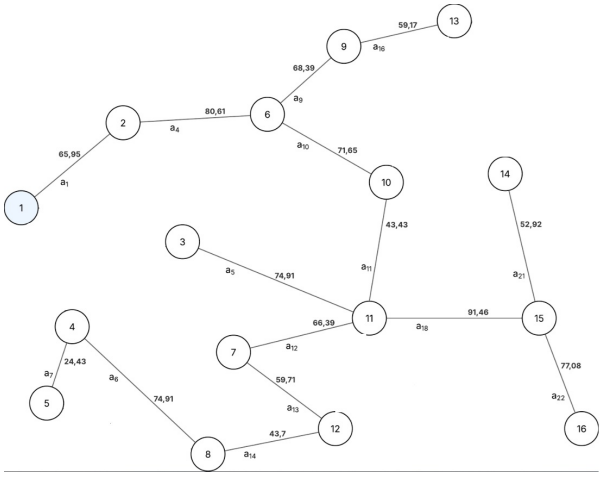
Quando analisamos a aresta (6,10) vemos que 10 equivale a 11, porém, o 11 equivale a 12, nesse caso, a comparação é feita com o 12, e nada se altera no 10, porque por mais que 11 não tenha mais nenhum membro em conjunto, o 11 ainda aponta para 12, o que faz com que 10 automaticamente aponte para 12 também, nesse caso 12 != 13 (6 equivale a 13), e adicionamos 13 para o conjunto 12.

Quando analisamos a aresta (13,10) vemos que os dois levam ao conjunto 12, nesse caso, essa aresta não é adicionada, pois se for, criará um ciclo, que é proibido em uma árvore, nesse caso, nada se altera, apenas se segue para a próxima aresta.

Depois de analisar a aresta (11,15), vemos que na fila de quantidades, todos os espaços estão em 0 e apenas o 12 tem 16 membros, como o total de vértices é 16, já podemos garantir que nenhuma aresta a mais será adicionada, já que todas já foram contempladas, e qualquer aresta adicional, criaria um ciclo novo.

No fim de todas as comparações, temos a árvore de cobertura mínima a partir do algoritmo de kruskal.

Figura 7.1: Árvore de cob mínima por Kruskal




Fonte: Criação própria

### 7.0.2 Algoritmo de Prim

Podemos obter a árvore de cobertura mínima também pelo algoritmo de Prim, que é melhor em relação a processamento computacional que o de Kruskal.

O algoritmo de Prim é parecido com o anteriormente mencionado algoritmo de Dijkstra no capitulo 6.2.1, a única diferença entre os dois, é que o de Prim não soma os pesos, eles são comparados em seus valores brutos, já que não buscamos um caminho, mas sim uma árvore, de resto, o processo é idêntico, o que o torna relativamente mais simples que o de Dijkstra.

No processo, usamos 3 filas, fila Pai, que vai definir o vértice pai de cada um, o que no fim define as arestas que estão na árvore; fila de pesos, que vai mostrar o peso de cada aresta na árvore e a fila de prioridade, que vai definir qual o próximo vértice que será contemplado, a fila é definida por peso, então caso a fila de prioridade já tenha todos os membros, e algum peso se altere, ela vai se alterar também, e depois que se contempla algum vértice, ele é desenfileirado, e a partir disso seu vértice pai nem seu peso pode ser alterado.

As filas são inicializadas da seguinte forma: A fila pai será inicializada com NULL em todas as posições, e será alterado seus valores ao longo do algoritmo; A fila de pesos será definida com todos os membros com valores muito altos, para não atrapalharem a fila de prioridade; e por fim, a fila de prioridade, que será iniciada em ordem crescente, já que a ordem da fila no momento inicial não importa necessariamente, porque os pesos são "iguais"no momento da criação da fila .

Ao realizar o processo do algoritmo, priorizamos nas comparações os menores pesos, então quando temos 2 arestas para o mesmo vértice, o mais leve é o que entrará

na árvore, e o outro será esquecido, para evitar ciclos dentro da árvore. Exemplo: Quando o 13 é desenfileirado da fila de prioridades, vemos que ele é adjacente a 10, o qual já estava na fila Pai o 6, então é feito a comparação dos pesos entre (6,10) e (13,10), e ao analisar, mantemos o (6,10), pois seu peso é menor, e na árvore final, não terá a aresta (13,10).

No fim, temos a seguinte fila pai:

[NULL,1,11,8,4,2,11,12,6,6,10,7,9,15,11,15]

E ao fazer a árvore de cobertura mínima, vemos que ela é identica a que pegamos pelo algoritmo de Kruskal, mostrando que os dois chegam ao mesmo lugar, mas o de prim é melhor computacionalmente falando.

## 8 GRAFOS EULERIANOS

A modelagem e solução de um problema de ciclos por Leonard Euler, durante o século XVIII, é responsável por definir os fundamentos da teoria dos grafos. A Seção 8.1 demonstra que o grafo apresentado geometricamente na Figura 2.1 não satisfaz o teorema de Euler. Para torná-lo euleriano, uma eulerização é descrita na Seção 8.2. As Seções 8.3 e 8.4 apresentam, respectivamente, a aplicação do algoritmo de Hierholzer e de Fleury para a obtenção de um ciclo euleriano no grafo apresentado geometricamente na Figura 1. As considerações acerca da eficiência computacional, tanto em relação ao tempo de processamento quanto ao espaço usado de memória pelos algoritmos de Hierholzer e de Fleury, são dadas na Seção 8.5.

### 8.1 Grafo Euleriano

Para compreender como que um grafo pode ou não ser Euleriano, é necessário olhar para os requisitos de um, que são:

Existência de um caminho chamado Euleriano, que se baseia em passar por todos os vértices pelo menos 1 vez, para isso, o grafo precisa que todos os vértices sejam de grau par, ou seja, a quantidade de arestas em cada vértice tem que ser par, sendo possível ter um com no máximo 2 vértices impares, isso torna o grafo semi-euleriano pois permite apenas um caminho, de um ponto A para o B, porém, para de fato, se ter um grafo euleriano, é necessário que todos os vértices tenham graus pares, para que seja possível a existência de um ciclo euleriano, que seria ir do ponto A e voltar até A passando por todas as arestas, sem repetição alguma.

Ao analisar o grafo da figura 2.1, logo de cara é percebido o vértice 1 tendo 3 ligações, o que o torna já um vértice de grau impar, e logo a frente, no vértice 4, tem-se novamente um vértice de grau impar, o $V_4$, que é ligado ao $V_1$ , $V_5$ e $V_8$, então nesse caso, o grafo já chegou em seu limite de vértices de grau impar, porém, seguindo para o outro lado, o $V_6$ também é impar, isso já o desqualifica para ser um grafo euleriano, já que sem nem olhar metade, já é possível identificar 3 vértices de grau impar.

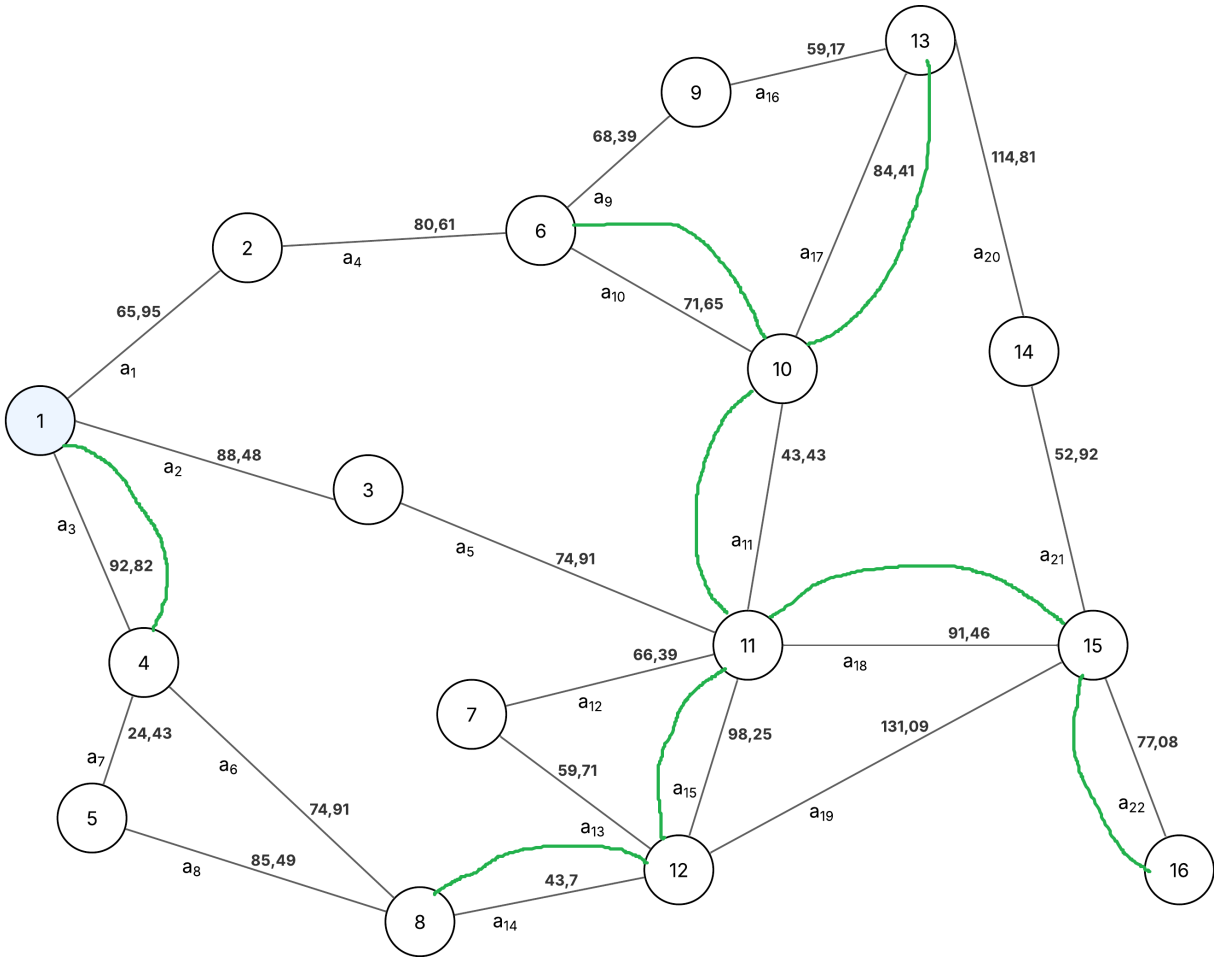
### 8.2 Eulerização de grafo

Com o grafo da figura 2.1 já comprovadamente não euleriano, agora, é possível transforma-lo.

Para transformar um grafo não euleriano para euleriano, é necessário duplicar arestas para tornar os vértices impares em pares, geralmente religando vértices impares entre si, determinando um multigrafo euleriano.

No grafo em questão, foi duplicado as arestas (1 - 4, 8 - 12, 12 - 11, 11 - 15, 15 - 16, 6 - 10, 10 - 13 e 10 - 11), ficando da seguinte maneira:

Figura 8.1: Grafo não euleriano transformado




Fonte: Criação própria

### 8.3 Algoritmo de Hierholzer

O algoritmo de Hierholzer, proposto em 1873, foi um dos primeiros a tratar ciclos eulerianos. A ideia é a partir de um vértice inicial qualquer $V_{Ini}$, percorrer as arestas até voltar para o $V_{Ini}$, porém, fazendo isso pode ocorrer de criar ciclos sem contemplar todas as arestas, nesse caso, é iniciado um novo ciclo a partir de um vértice com arestas inexploradas $V_{nov}$ e percorrido as arestas que não foram exploradas, até que volte para $V_{nov}$ e repetindo o mesmo processo até que todas as arestas tenham sido contempladas.

Algoritmo de forma computacional: Primeiramente vai ser criado um grafo reduzido $G_{red}$, que inicialmente é igual ao grafo escolhido.

É escolhido o vértice inicial de forma aleatória $V_{Ini}$

É iniciado um loop que a condição vai ser a verificação se existe alguma aresta não explorada. Dentro do loop, é escolhido um vértice $V_c$ que esteja no ciclo euleriano e que tenha grau maior que 0 no grafo reduzido Seguindo, é definido um ciclo auxiliar $C_{Aux}$ contendo o $V_c$ como extremidade inicial e final, e todos os vértices pertencentes

ao $G_{Red}$ que tenham grau maior que 0 E então é retirado do $G_{Red}$ as arestas que estão no $C_{Aux}$. Por fim o segmento a partir do $V_c$ no ciclo é substituído pelo $C_{Aux}$. A partir daqui é reiniciado o ciclo caso a condição do loop seja atendida, caso não seja, ele passa e retorna o ciclo euleriano completo formado no loop.

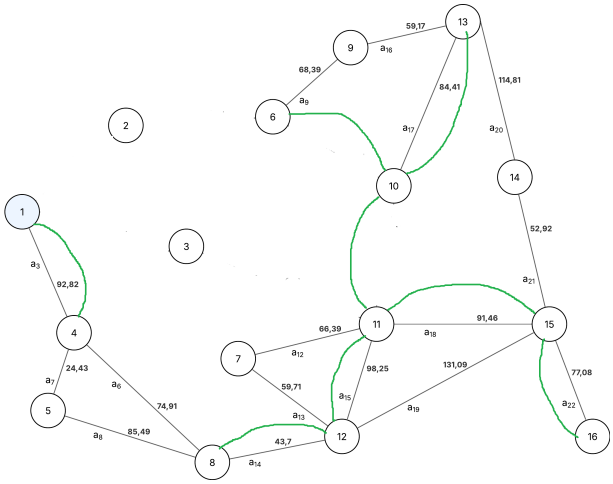
Agora no grafo da figura 8.1:

Vamos escolher o vértice 1 para iniciar o ciclo, logo vamos definir o ciclo auxiliar, utilizando o 1 como extremidades. $C_{Aux}$ = 1,2,6,10,11,3,1

Logo depois vamos apagar as arestas utilizadas no ciclo. E com isso adicionamos o segmento para o $C_E$. $C_E$ = 1,2,6,10,11,3,1

Com esse iteração, ficamos com o seguinte grafo:

Figura 8.2: Hierholzer ciclo 1

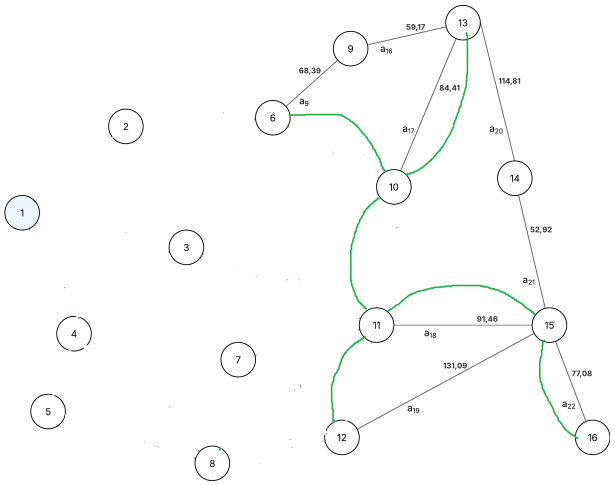



Fonte: Criação própria

Verificamos então mais vértices com grau maior que zero, e ao fazer, podemos escolher novamente o $V_1$ pois ele ainda tem arestas inexploradas. A partir dele percorremos até voltar nele. $C_{Aux}$ = 1,4,5,8,12,11,7,12,8,4,1. Apagamos do $G_{Red}$ e substituímos o segmento no ciclo. Ce = 1,4,5,8,12,11,7,12,8,4,1,2,6,10,11,3,1.

Agora o grafo reduzido fica da seguinte forma:

Figura 8.3: Hierholzer ciclo 2



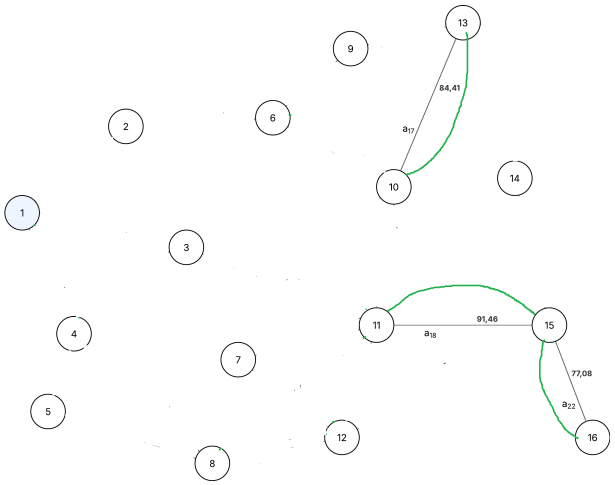

Fonte: Criação própria

Novamente vamos verificar se ainda existem vértices com grau maior que 0, dessa vez, o 1 já tem todas as arestas contempladas, então vamos seguir pelo ciclo até que apareça um vértice com grau maior que 0. Escolhemos o 12 pois ele ainda tem arestas inexploradas. Definimos o $C_{Aux}$ = [12,11,10,6,9,13,14,15,12] Apagamos do grafo reduzido e substituímos o segmento no Ciclo.

Ce = [1,4,5,8,12,11,10,6,9,13,14,15,12,11,7,12,8,4,1,2,6,10,11,3,1]

O grafo atualmente está do seguinte jeito:

Figura 8.4: Hierholzer ciclo 3




Fonte: Criação própria

Olhando para a imagem 8.4, já é possível identificar que o processo vai terminar em apenas 2 iterações, provando que o grafo em questão pode ser um grafo euleriano contando que esteja transformado.

Analisando os vértices que tem grau maior que 0, vamos começar o novo ciclo com o vértice 11.

No caso, o $C_{Aux}$ vai ser igual a [11,15,16,15,11] Apagamos do grafo reduzido e adicionamos o seguimento. CE = [1,4,5,8,12,11,15,16,15,11,10,6,9,13,14,15,12,11,7,12,8,4,1,2,6,10,1

Agora por último, iremos usar o último ciclo, que vai se iniciar pelo 10. $C_Aux$ = [10,13,10]

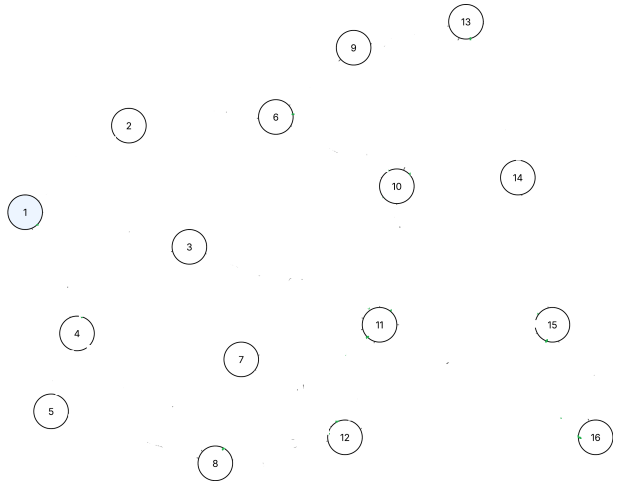
Apagamos do grafo e adicionamos o seguimento.

CE = [1,4,5,8,12,11,15,16,15,11,10,13,10,6,9,13,14,15,12,11,7,12,8,4,1,2,6,10,11,3,1]

E assim obtemos o ciclo euleriano completo do grafo, tendo a possibilidade de testar, apagando cada aresta mencionada no ciclo.

E no fim, o grafo deve estar desse jeito no auxiliar, com todos os vértices flutuando sem nenhuma aresta:

Figura 8.5: Hierholzer ciclo Final




Fonte: Criação própria

### 8.4 Algoritmo de Fleury

Algoritmo proposto em 1883, funcionando em grafos com todos os vértices pares, porém, também funcionando em grafos com no máximo 2 vértices de grau impar (semi-Eulerianos).

Para grafos eulerianos, o primeiro vértice escolhido pode ser qualquer um, já que todos são de grau par, mas se for semi-euleriano, o primeiro vértice deverá ser um dos impares.

A partir do vértice, é feito as mesmas inicializações do algoritmo de Hierholzer, que é um grafo reduzido, com os mesmos parâmetros do grafo em questão, e uma lista do ciclo euleriano a ser formado.

Seguindo, é feito o loop verificando novamente se existe algum vértice com aresta inexplorada, caso seja, entra no loop, caso não, segue e retorna a lista que se formou.

Entrando no loop temos a seguinte verificação: Caso o grau do vértice escolhido seja igual a 1, ele muda a escolha de vértice para o adjacente, pois o vértice se tornou

ponte, que seria um vértice com uma ligação isolada para outro vértice que a um primeiro momento, não será contemplada.

Caso não seja igual a 1, será escolhido algum dos vértices adjacentes que não sejam pontes.

Em seguida o vértice escolhido é adicionado no ciclo euleriano e a aresta utilizada é apagada do grafo reduzido.

No fim do loop, o vértice escolhido se torna a extremidade do ciclo euleriano.

Caso não exista mais nenhum vértice sem ser ponte, ai as pontes começam a ser contempladas e apagadas do grafo.

Para formar o ciclo utilizando esse algoritmo, será utilizado o mesmo grafo do tópico anterior, a fins de comparação do ciclo final:

Primeiramente, é possível escolher qualquer vértice, já que temos um grafo euleriano transformado, então será escolhido como vértice inicial o 1 do mesmo jeito que o exemplo do tópico anterior.

$V_Es$ = [1] $C_E$ = [1]

O $V_1$ tem grau igual a 4, então podemos escolher qualquer vértice que não seja uma ponte, e nesse caso, será escolhido o $V_4$.

$V_Es$ = [4] $C_E$ = [1,4]

E é apagado do grafo a aresta utilizada e adicionado ao Ce o vértice escolhido.

A seguir, será otimizado o processo e encurtado a explicação, pois é um grafo relativamente grande, caso tenha alguma observação importante, será documentada.

Iteração 2:

$V_Es$ = 3 > 0

$V_Es$ = [8]

$C_E$ = [1,4,8]

Iteração 3:

$V_Es$ = 3 > 0

$V_Es$ = [12]

$C_E$ = [1,4,8,12]

Iteração 4:

$V_Es$ = 5 > 0

$V_Es$ = 11 *nesse caso, 12-7 não é escolhido por ser ponte

$C_E$ = [1,4,8,12,11]

Iteração 5:

$V_Es$ = 7 > 0

$V_Es$ = 10

$C_E$ = [1,4,8,12,11,10]

Iteração 6:

$V_Es$ = 5 > 0

$V_Es$ = 6
$C_E$ = [1,4,8,12,11,10,6]
Iteração 7:
$V_Es$ = 3 > 0
$V_Es$ = 9 *nesse caso é escolhido o 9 pois não opção nenhuma na adjacência
$C_E$ = [1,4,8,12,11,10,6,9]
Iteração 8:
$V_Es$ = 1 > 0
$V_Es$ = 13 *sem escolha novamente
$C_E$ = [1,4,8,12,11,10,6,9,13]
Iteração 9:
$V_Es$ = 3 > 0
$V_Es$ = 10 *unico sem ser ponte
$C_E$ = [1,4,8,12,11,10,6,9,13,10]
Iteração 10:
$V_Es$ = 3 > 0
$V_Es$ = 11
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11]
Iteração 11:
$V_Es$ = 5 > 0
$V_Es$ = 12
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12]
Iteração 12:
$V_Es$ = 3 > 0
$V_Es$ = 15
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15]
Iteração 13:
$V_Es$ = 5 > 0
$V_Es$ = 16
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16]
Iteração 14:
$V_Es$ = 1 > 0
$V_Es$ = 15
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15]
Iteração 15:
$V_Es$ = 3 > 0
$V_Es$ = 11
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11]
Iteração 16:

$V_Es$ = 3 > 0
$V_Es$ = 7
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7]
Iteração 17:
$V_Es$ = 1 > 0
$V_Es$ = 12
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12]
Iteração 18:
$V_Es$ = 1 > 0
$V_Es$ = 8
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8]
Iteração 19:
$V_Es$ = 1 > 0
$V_Es$ = 5
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5]
Iteração 20:
$V_Es$ = 1 > 0
$V_Es$ = 4
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4]
Iteração 21:
$V_Es$ = 1 > 0
$V_Es$ = 1
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1]
Iteração 22:
$V_Es$ = 2 > 0
$V_Es$ = 2
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2]
Iteração 23:
A partir daq todos os graus são 1 até finalizar
$V_Es$ = 6
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2,6]
Iteração 24:
$V_Es$ = 10
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2,6,10]
Iteração 25:
$V_Es$ = 13
$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2,6,10,13]
Iteração 26:
$V_Es$ = 14

$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2,6,10,13,14]

Iteração 27:

$V_Es$ = 15

$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2,6,10,13,14,15]

Iteração 28:

$V_Es$ = 11

$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2,6,10,13,14,15,11]

Iteração 29:

$V_Es$ = 3

$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2,6,10,13,14,15,11,3]

Iteração 30:

$V_Es$ = 1

$C_E$ = [1,4,8,12,11,10,6,9,13,10,11,12,15,16,15,11,7,12,8,5,4,1,2,6,10,13,14,15,11,3,1]

Agora terminado, é possível ver que o ciclo euleriano foi um sucesso, começa pelo 1, termina pelo 1 e passa por todos as arestas sem repetição alguma.

# 9 GRAFOS HAMILTONIANOS

## 10 EMPARELHAMENTO

O emparelhamento em grafos (ou *matching*) é um conceito fundamental na teoria dos grafos, com vastas aplicações em problemas de alocação de recursos, logística e otimização combinatória. No contexto do problema logístico descrito neste trabalho, o emparelhamento pode ser interpretado como a atribuição de veículos a rotas, ou de motoristas a caminhões, de modo que nenhum recurso seja compartilhado simultaneamente por duas tarefas distintas.

Formalmente, dado um grafo $G = (V, A)$, um emparelhamento $M$ é um subconjunto de arestas $M \subseteq A$ tal que nenhum par de arestas em $M$ compartilha um vértice em comum. Ou seja, para todo $v \in V$, o vértice $v$ incide em no máximo uma aresta de $M$.

### 10.1 Caminho Alternante

Para compreender a maximização de emparelhamentos, é necessário definir o conceito de caminho alternante. Dado um emparelhamento $M$ em um grafo $G$, um vértice é dito saturado se ele é extremidade de alguma aresta em $M$, e livre (ou não saturado) caso contrário.

Um caminho alternante é um caminho simples cujas arestas se alternam entre pertencerem a $M$ e não pertencerem a $M$ ($A \setminus M$).

Considerando o grafo $G_1$ do nosso estudo, suponha um emparelhamento inicial arbitrário $M = \{(1, 2), (4, 5)\}$.

- Arestas em $M$: $(V_1, V_2)$ e $(V_4, V_5)$.
- Vértices saturados: $\{1, 2, 4, 5\}$.
- Vértices livres (exemplo): $\{3, 6, 8, \dots\}$.

Um exemplo de caminho alternante partindo do vértice livre $V_3$ seria:

$$P = (V_3, V_1, V_2, V_6)$$

Onde $(V_3, V_1) \notin M$, $(V_1, V_2) \in M$ e $(V_2, V_6) \notin M$.

### 10.2 Caminho Aumentante

Um caminho aumentante é um tipo especial de caminho alternante que começa e termina em vértices livres (não saturados). A propriedade fundamental deste caminho é que ele possui um número ímpar de arestas, tendo uma aresta a mais fora de $M$ do que dentro de $M$.

Utilizando o exemplo da seção anterior com $M = \{(1, 2), (4, 5)\}$ e o caminho $P = (V_3, V_1, V_2, V_6)$:

- Início: $V_3$ (Livre).
- Fim: $V_6$ (Livre).

Ao inverter a pertinência das arestas deste caminho em relação a $M$ (operação de diferença simétrica $M \oplus P$), obtemos um novo emparelhamento $M'$ com uma aresta a mais.

$$M' = M \oplus P = \{(3,1), (2,6), (4,5)\}$$

O tamanho do emparelhamento aumentou de 2 para 3 arestas. Segundo o Teorema de Berge, um emparelhamento $M$ é máximo se, e somente se, não existe caminho aumentante em relação a $M$.

### 10.3 Emparelhamento Maximal

Um emparelhamento $M$ é dito maximal se ele não pode ser estendido pela simples adição de uma aresta. Ou seja, se adicionarmos qualquer aresta $e \in A \setminus M$ ao conjunto, a propriedade de emparelhamento é violada (a aresta $e$ compartilha um vértice com alguma aresta já existente em $M$).

É importante notar que um emparelhamento maximal não é necessariamente um emparelhamento máximo (o maior possível). Ele é apenas um máximo local em relação à inclusão de arestas.

No nosso grafo, o conjunto $M_{maximal} = \{(4,8), (6,10), (11,15)\}$ é maximal, pois qualquer outra aresta que tentarmos adicionar conectará a um vértice já utilizado (por exemplo, $(1,4)$ conecta ao 4 que já está em uso).

### 10.4 Emparelhamento Máximo

Um emparelhamento é máximo se possui a maior cardinalidade possível entre todos os emparelhamentos do grafo. Todo emparelhamento máximo é maximal, mas a recíproca não é verdadeira.

O número de arestas em um emparelhamento máximo é denotado pelo número de emparelhamento do grafo, $\nu(G)$. Para encontrar o emparelhamento máximo em grafos gerais, algoritmos iterativos buscam caminhos aumentantes até que estes se esgotem.

### 10.5 Emparelhamento Perfeito

Um emparelhamento $M$ é perfeito se ele satura todos os vértices do grafo. Ou seja, todo vértice do grafo é extremidade de exatamente uma aresta em $M$. Para que um emparelhamento perfeito exista, é condição necessária (mas não suficiente) que o número de vértices $|V|$ seja par.

$$|M| = \frac{|V|}{2}$$

Como o grafo $G_1$ possui $|V| = 16$ (par), um emparelhamento perfeito conteria exatamente 8 arestas.

### 10.6 Algoritmo Húngaro

O Algoritmo Húngaro é um método de otimização combinatória que resolve o problema de atribuição em tempo polinomial $O(V^3)$. No entanto, ele é aplicável especificamente a grafos bipartidos ponderados. Como o grafo $G_1$ estudado não é bipartido (possui ciclos de comprimento ímpar, ex: $V_4 - V_5 - V_8$), simularemos uma aplicação do algoritmo considerando um subproblema de logística.

Suponha um subgrafo bipartido onde o conjunto $X$ representa centros de distribuição (CDs) e o conjunto $Y$ representa pontos de entrega, com os pesos das arestas representando custos (distâncias).

**Exemplo de Matriz de Custos:**

$$\begin{bmatrix} & Y_1 & Y_2 & Y_3 \\ X_1 & 10 & 19 & 8 \\ X_2 & 10 & 18 & 7 \\ X_3 & 13 & 16 & 9 \end{bmatrix}$$

O algoritmo segue os passos:

1. **Redução das linhas:** Subtrair o menor elemento de cada linha de todos os elementos daquela linha.

2. **Redução das colunas:** Subtrair o menor elemento de cada coluna da matriz resultante.

3. **Cobertura de zeros:** Cobrir todos os zeros com o número mínimo de linhas horizontais ou verticais. Se o número de linhas for igual à dimensão da matriz, a solução ótima foi encontrada.

4. **Ajuste:** Caso contrário, subtrai-se o menor elemento não coberto dos não cobertos e soma-se às interseções das linhas.

Aplicando ao exemplo, a atribuição de custo mínimo seria encontrada, garantindo a eficiência logística entre os dois conjuntos disjuntos.

### 10.7 Algoritmo de Edmonds (Blossom)

Para encontrar o emparelhamento máximo em grafos gerais (não bipartidos), como é o caso do grafo $G_1$ deste projeto, o algoritmo padrão de busca por caminhos aumentantes falha devido à presença de ciclos ímpares.

O Algoritmo de Edmonds, também conhecido como *Blossom Algorithm*, resolve este problema tratando os ciclos ímpares.

Um *blossom* é um ciclo ímpar $C$ de $2k + 1$ arestas onde $k$ arestas pertencem ao emparelhamento $M$, e existe um vértice $v$ no ciclo (a base) tal que existe um caminho alternante de um vértice livre até $v$.

1. O algoritmo busca caminhos aumentantes a partir de vértices livres.
2. Se encontrar um ciclo ímpar durante a busca (um *blossom*), o algoritmo contrai todo o ciclo em um único super-vértice.
3. O grafo contraído é então examinado. Se um caminho aumentante é encontrado no grafo contraído, ele pode ser expandido de volta para o grafo original.

No grafo $G_1$, a estrutura triangular $V_4 - V_5 - V_8$ é um exemplo de ciclo ímpar que, dependendo da configuração do emparelhamento, poderia formar um *blossom*, exigindo a contração para que o algoritmo prosseguisse corretamente na identificação do emparelhamento máximo.

# 11 PLANARIDADE

# 12 COLORAÇÃO DE VÉRTICES

**REFERÊNCIAS**